A New SpaceWire Protocol for Reconfigurable Distributed On-Board Computers

SpaceWire Networks and Protocols, Long Paper

Ting Peng, Benjamin Weps, Kilian Höflinger
Simulation and Software Technology
German Aerospace Center
Lilienthalplatz 7
38108 Braunschweig, Germany
ting.peng@dlr.de, benjamin.weps@dlr.de,
kilian.hoeflinger@dlr.de

Kai Borchers
Institute of Space Systems
German Aerospace Center
Robert-Hooke-Str. 7
28359 Bremen, Germany
kai.borchers@dlr.de

Daniel Lüdtke, Andreas Gerndt
Simulation and Software Technology
German Aerospace Center
Lilienthalplatz 7
38108 Braunschweig, Germany
daniel.luedtke@dlr.de, andreas.gerndt@dlr.de

***Abstract*—There are several standardized protocols based on SpaceWire which provide data exchange between several nodes. SpaceWire is also suitable for interprocess communication (IPC), by the help of higher level protocols. However, currently there is no standardized protocol which is targeting IPC on SpaceWire networks. This paper proposes a protocol, which uses the capabilities of SpaceWire to build up networks for distributed computing on a spacecraft. The core of this protocol is the IPC mechanism for communication between the nodes and methods to support a reconfiguration of the network. A key feature of this protocol is an interface for a reconfiguration mechanism, which can be implemented on application level. This enables the utilization of unreliable commercial off the shelf (COTS) nodes, allowing system recovery from erroneous state. Additionally, the reconfiguration can be used to adapt the distributed computer to different mission phases. The protocol has the potential to build the foundation of a distributed on-board computer consisting of COTS components. Such distributed computer could be capable of fulfilling high performance demands as well as high reliability needs. Though, the protocol itself is not restricted to be used solely in fully-featured reconfigurable distributed systems. The IPC methods can be applied stand-alone as well, to establish a lightweight communication between nodes on a SpaceWire network by excluding the reconfiguration parts of the protocol.**

***Index Terms*— SpaceWire, Network, Protocol, Reconfigurable, Interprocess Communication, COTS, High Reliability.**

I. Introduction

Distributed systems with COTS components use multiple computing nodes to share the workload and offer significantly higher computing performance than currently used space-qualified on-board computers. It is necessary to offer complex IPC services and satisfy strict requirements for satellite missions, such as real time and reliable transmission as well as high transmission speed. Reliability of COTS can be realized via redundancy by the execution of equivalent tasks on different nodes and by the reconfiguration of nodes and tasks, i.e., migration of tasks to other available nodes after some nodes fail. The distributed system should support upgrade, maintenance and failure detection, isolation as well as recovery.

SpaceWire is suitable for IPC with further protocols. However, currently there is no standardized protocol that is explicitly targeting IPC on SpaceWire networks and to be utilized in reconfigurable distributed on-board computers. Therefore, a new protocol, based on SpaceWire, is necessary to support the reconfigurable distributed on-board computers.

We will introduce a new protocol called SpaceWire-IPC, which is beneficial for reliable and fault tolerant distributed on-board computers

The paper is organized as follows. Section II presents the related work, which was taken into account during development of our proposed SpaceWire-IPC protocol. Section III describes the requirements for the protocol, derived from our project. Structure and properties of the new protocol SpaceWire-IPC are introduced in section IV. Finally, section V provides the comparison between SpaceWire-IPC and already existing protocols, followed by a conclusion in section VI.

II. Related Work

This section provides an overview of existing SpaceWire compatible protocol specifications. With the exception of

The final publication appeared: T. Peng, B. Weps, K. Höflinger, K. Borchers, D. Lüdtke and A. Gerndt, "A New SpaceWire Protocol for Reconfigurable Distributed On-Board Computers," in International SpaceWire Conference, Yokohama, Japan, 2016. URL: http://2016.spacewire-conference.org/proceedings/

SpaceWire-R, all protocols are referenced and officially adopted by an according ECSS standard [1]. Additionally, the trends of IPC and reconfiguration in space systems are presented.

A. Overview of Existing SpaceWire Protocols

The subsequent paragraphs list SpaceWire protocols, which were considered for the development of the distributed on-board system.

1) Remote Memory Access Protocol (RMAP)

The Remote Memory Access Protocol (RMAP) protocol, defined by standard [2], is commonly used in space applications for reading from and writing to memory in remote SpaceWire nodes. The protocol provides the ability to address destinations by the use of path or logical addressing over the *Target SpW Address* fields as defined in [3]. However, in case only logical addressing is required it is also possible to skip the *Target SpW Address*. The protocol can be directly integrated into a standard SpaceWire protocol by using the *Protocol Identifier*. By use of an *Instruction* byte, the following modes/message types are possible:

- Read command/ write command
- Verify / no verify of data before write
- Acknowledge / no acknowledge of write command
- Read-Modify-Write

If data shall be read or an acknowledgment is required, a set of *Reply Address* fields are available, which are usable for path and/or logical addressing. The source of the received command is stored inside the *Initiator Logical Address*. To prevent a lock-step limitation during communication two *Transaction Identifier* bytes are available, which allow the user to apply out of order transfers. To define the target location for read or write commands, a set of four *Address* fields, plus an additional *Extended Address* field is used.

Written and read data is secured by Cyclic Redundancy Checks (CRCs) for Header and Payload data independently.

2) CCSDS Packet Transfer Protocol (CCSDS PTP)

The CCSDS protocol is intended to encapsulate a user defined protocol that needs to be transferred through a SpaceWire network [4]. Similar to the RMAP, an arbitrary amount of *Target SpW Address* fields can be used for routing. Alternatively, the *Target Logical Address* is used to define the destination. The *Protocol Identifier* distinguishes between different SpaceWire packet types. The interpretation of data of the *CCSDS Packet* fields is user specific and defined inside the *User Application* field.

3) GOES-R Reliable Data Delivery Protocol (GRDDP)

The main purpose of GRDDP is to transfer data of sensors, telemetry and commands among peripheral instruments and the on-board computer [5]. The *Destination SLA* serves as a logical address, related to the targeted destination. To provide information about the source of the packet *Source SLA* is used. Four different packet types can be used by defining the *Packet Type* field:

- Application Data
- Acknowledge
- Reset Command
- Urgent Message Data

To detect packet loss or to order out-of-order packets, a *Sequence Number* is provided. The whole packet, except End-Of-Packet (EOP), is covered and checked by a CRC.

4) Serial Transfer Universal Protocol (STUP)

The STUP protocol, defined in [6], serves as a light weighted protocol with the intention to implement a more complex protocol, inside the data field. To define the source of the packet the *Source Logical Address* is used. Different kinds of data structures can be defined by the *Data* fields. The standard defines an example where the first data byte defines a kind of message type, which is used to interpret the left data bytes. Only write, read and read reply commands are offered in this example.

5) SpaceWire-R

SpaceWire-R is used for reliable data transmission within SpaceWire networks [7]. It is based on the GRDDP and the Joint Architecture Standard Reliable Data Delivery Protocol (JAS RDDP). SpaceWire-R provides features like multiplexing, message segmentation, reliable transfer, network traffic flow control (optional) and heartbeat signaling (optional) [7].

B. Trends in Space Systems

Several space projects use distributed on-board computers to meet the increasing demands of on-board processing ability. The On-Board Computer - System Architecture (OBC-SA) consists of two on-board computers, one of which is COTS from Freescale's PowerPC multicore CPU [8]. The High-Performance Reconfigurable Computing Space Processor (CSP)'s hardware structure is based on both COTS and radiation-hardened technologies. ISS SpaceCube Experiment Mini (ISEM)'s hardware has two CSP boards which are interconnected by SpaceWire and UART [9]. CSP aims to offer space image processing, distributed parallel computation and fault tolerance [9]. The Fault-Tolerant Distributed On-Board Computer (FTD-OBC) gains higher reliability and higher processing performance by multiple processing nodes connected by CAN buses of 1 Mbps [10].

III. Protocol Requirements

Inspired by the rise of distributed computing techniques and advantages of SpaceWire, the project Scalable On-board computing for Space Avionics (ScOSA) and its predecessor project On-board Computer - Next Generation (OBC-NG) at German Aerospace Center (DLR) use COTS hardware besides radiation-hardened components to establish a distributed on-board computing network, based on SpaceWire. Their goal is to leverage performance of a distributed architecture and still maintain the required reliability.

In ScOSA, three types of nodes, High-Performance Nodes (HPNs), Reliable Computing Nodes (RCNs) and Interface Nodes (IFNs) are used (see Fig. 1). HPNs are based on a Xilinx Zynq XC7Z020 architecture (CPU + FPGA) while RCNs have a LEON3 as FPGA soft-core implementation. The SpaceWire router is integrated in the FPGAs of the RCNs, the HPNs and the IFNs.

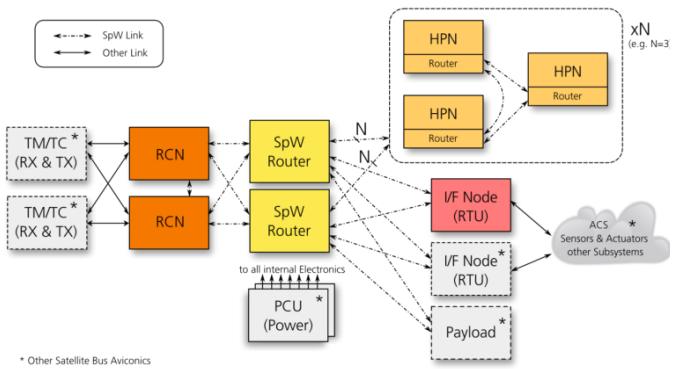

Fig. 1. ScOSA system overview

The middleware offers monitoring, task management, checkpointing and reconfiguration services for the system. These services are coordinated by three types of roles among the RCNs and HPNs: Master, Observer and Worker. A global configuration for all nodes means the deployment of the Master, Observers and Workers on RCNs and HPNs, settings of monitoring behavior, and channels availability and subscriber lists of IFNs, etc. The Master is responsible to initiate the configuration for all nodes in the network, by broadcasting the configuration command. The Master also monitors the distributed system via a periodical heartbeat mechanism and a plausibility check of some control values of application tasks. Some internal states of the application tasks are periodically sent to the mass memory storage for checkpointing. If a node fails, the Master will trigger a system reconfiguration and redistribute the tasks to other nodes. After the reconfiguration finishes, the checkpoints will be retrieved from storage back to the nodes, which are running the corresponding tasks. Two or more Observers are assigned to monitor the Master. In case the Master fails, a decision will be made to choose one Observer to take over the failed Master's tasks.

This reconfigurable distributed system intensively relies on IPC. For the consideration of scalability and throughput, a bus topology can't be used [11]. Nodes are interconnected with point-to-point links. An irregular network topology structure is used to avoid a single-point failure and to maintain flexibility [11].

To summarize the analysis of requirements for the ScOSA distributed system, the network should

- be scalable and flexible,
- be able to transfer arbitrary large messages,
- have high reliability supported by redundant routes among nodes in case of failed nodes,
- guarantee the reliability of messages delivery,
- deal with message losses,
- support monitoring, error notification and reconfiguration.

IV. Protocol Description

The ScOSA project mainly uses the SpaceWire-IPC for the communication among the distributed computing nodes. The SpaceWire-IPC is located at the transport layer of ISO-OSI model [12]. SpaceWire acts as the underlying protocol (see Fig. 2).

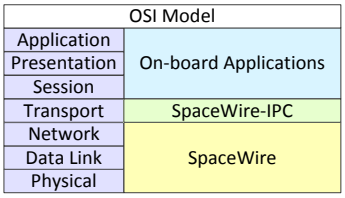

Fig. 2. Protocols and OSI model

A. Features

SpaceWire-IPC offers communication for:

- IPC among nodes
- Management services from and to Master

SpaceWire-IPC supports:

- Multiple logical nodes on one physical device
- Reliable transmissions of data as well as unreliable transmissions
- Recognition of failed connections and failover mechanisms
- Transmission of messages with arbitrary size
- Transmission of large-size messages
- Transparent use for different underlying protocols
- Multiple APIs rather than only read and write commands

The protocol is message-based, meaning that instead of streams, single messages are sent from one node to another. These messages can be reliable or unreliable.

SpaceWire offers no regulation regarding the maximum packet sizes. SpaceWire-IPC implements a sequencing technique, which allows splitting large messages into smaller packets, with their size being user-defined. The message is subdivided in packets on the sender and reassembled at its destination node. In case of reliable message, it is a bidirectional packet transfer with acknowledgments. Each packet has a checksum to verify the integrity.

B. Design Decisions

This section lists and explains the message structures used in SpaceWire-IPC (see Fig. 3).

Target SpW Address (1 byte)	.....	Target SpW Address (1 byte)
Target Logical Address (1 byte)	Potocol Identifier (1 byte)	Sender Node ID (2 bytes)
Receiver Node ID (2 bytes)	Timestamp (8 bytes)	Message Type (1 byte)
Payload Data (0 to n bytes)	Checksum (4 bytes)	EOP (1 byte)

Fig. 3. Structure of a SpaceWire-IPC Packet

1) Message Header

The message header shown in Fig. 3 is identical for all types of messages. The header contains source and destination of the packet, the timestamp it has been created and the size of the payload. While the header stays the same, the message type determines the structure of the payload data.

a) Sender Node ID and Receiver Node ID

Sender Node ID and *Receiver Node ID* are the logical addresses of the nodes that participate in this transmission. An ID determines exactly one entity in the network capable of sending and receiving messages. This does not necessarily mean, that it has to be unique for each physical node (regarding to SpaceWire Addressing), which is connected to the network. A physical node can have multiple software components running, which are able to send and receive messages.

b) Timestamp

This marks the time the packet has been created. The timestamp, together with the sender and receiver node ID, is the unique identifier for a packet.

c) Message Type

This field determines the structure of the payload data. It also indicates how the receiving node should handle this message. The possible values are listed in TABLE I.

One exception in this scheme is the *Large Message Transfer*. The bits 0-6 define the encapsulated message type as usual. The most significant bit determines if this message is part of a *Large Message Transmission.*

TABLE I. SUMMARY OF MESSAGE TYPES

Integer Value	Message Type
0	Unreliable Data Transmission
1	Reliable Data Transmission
2	Data Request
3	Data Response
4	Reconfiguration Request
5	Message Acknowledge
6	Heartbeat
7	Error Notification
128+	Large Message Transfer

d) Payload Data

To stay as versatile as possible, the payload data is just an arbitrary-sized byte array. The structure can be derived from the message type. For some message types, the array has a fixed size, other message types have a variable-sized array of data. The exact structure for each data type is described later in this paper.

e) Checksum

The checksum provides a way to check the integrity of the transmission. As it is not guaranteed that the underling protocol has a mechanism to detect erroneous messages, the integrity check will be implemented in this protocol. The protocol does not dictate a specific algorithm for the checksum. The only restriction is, that it must not exceed the size of the 32-bit value provided by this field. The value of this checksum should consider all of the previous fields of the transmission to guarantee the integrity of the whole packet.

The checksum algorithm can be selected by considering the mission requirements and available resources of the nodes in the network. An example for checksums is the CRC32 algorithm as described in the IEEE802.3 (Ethernet) Standard.

2) Data Transmissions

The *Data Transmission* types are the central message type for transmitting data inside the distributed system. The data is handled by the protocol as an arbitrary byte array. Hence, it has no influence on the handling of the message. The structure and handling of the data is not part of the protocol and has to be conducted at the application level.

The payload structure of *Data Transmissions* is the same for both, reliable and unreliable transmissions and its layout is shown in Fig. 4. The Data Size contains the number of elements of the following byte array. The data byte array contains the actual data.

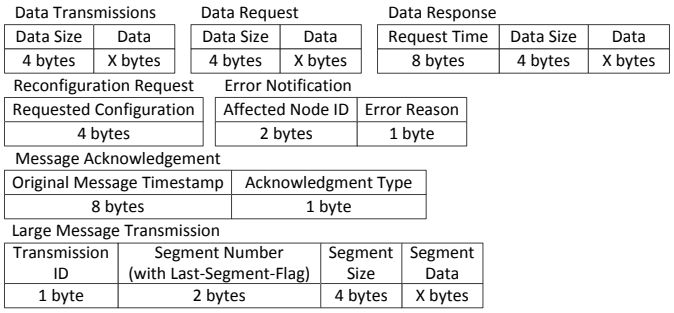

Fig. 4. Summary of payload structures

Data can be transmitted in a reliable or in an unreliable manner. Therefore, two message types are available for this purpose: reliable and unreliable *Data Transmission*. The only difference between these two types is that the receiver, when received successfully, will acknowledge the reliable *Data Transmission*. Though, the sender can resend a packet if it was lost or falsely transmitted. Unreliable messages will be dropped when received erroneous.

3) Data Request

For transmitting data with the request-response method, a *Data Request* message can be sent. This message triggers the receiver to execute an action and send data to the sender of this request.

To assign the response to the according request, the responding data will be transmitted with a *Data Response* message instead of a normal *Data Transmission*. The *Data Response* message, which is following the request, will be sent asynchronously. In that way, the action, which has been requested, can take longer time than the normal acknowledgment.

The payload of a *Data Request* has the same structure as in Fig. 4. The *Data Size* contains the number of elements of the

following *Data Byte Array*. This array contains the data that will be sent to the remote node. The structure of this array is not specified in this protocol. The data has to be parsed at the application layer.

To ensure the request will trigger an action and a data response, *Data Requests* are reliable messages, which have to be acknowledged.

4) Data Response

The *Data Response* is used to send data back to a node that has sent a *Data Request* message. This message will be sent asynchronously after the action that was triggered by the request has been executed.

The payload of a *Data Response* has also the structure shown in Fig. 4. The Request Time field contains the timestamp of the original *Data Request* that triggered this response. The Data Size contains the number of elements of the following byte array. The Data byte array contains the data that will be sent to the remote node. The structure of this array is not specified in this protocol.

To ensure the request will trigger an action and a data response, *Data Responses* are reliable messages that will be acknowledged.

5) Reconfiguration Request

The *Reconfiguration Request* notifies all nodes in the network to switch to a certain configuration. Only the Master node is capable of sending these requests, as it is the only instance authorized to define the global state of the system. Every other node than the Master node shall only be able to receive this message, but not sending it.

Reconfiguration Requests are always transmitted reliable. Nodes that are not responding to a *Reconfiguration Request* have to be disabled by a new reconfiguration.

6) Message Acknowledgment

The *Message Acknowledgment* is the central element in the reliability mechanism of this protocol. Reliably sent messages will be acknowledged with this message type. The acknowledgment can be either a positive acknowledgment, notifying that the message has been received successfully, or a negative acknowledgment to inform the sender of the original message, that it arrived erroneous.

Acknowledgments are not transmitted reliable. When an acknowledgment packet is lost, the original message will be sent again.

7) Heartbeat

The *Heartbeat* is a message type, used to check if a certain node is responsive. The *Heartbeat* itself is a request for an acknowledgment message. This mechanism allows a verification of the bidirectional communication link. The management instance sends out these *Heartbeat* messages periodically.

8) Error Notification

The *Error Notification* is used to inform the Master of the distributed system about an error that occurred. As soon as a reliable connection is not acknowledged, after a certain amount of attempts, this notification will be sent to the Master node. This message can also be used to notify other nodes that an error has occurred.

Error Notifications will be sent reliable. This enables the possibility to detect whether the sending node has lost the connection to the network, or the erroneous node is the source of the failure. If this error message cannot be delivered to a management node it can been assumed that the node itself has lost its connection to the rest of the distributed system.

9) Large Message Transmission Packet

SpaceWire itself does not limit the packet size. But to avoid long blocking of paths in the network, a restriction on packet size is defined in this protocol. Here, the *Large Message Transfer* mechanism offers a way to split a message into smaller packets, which then can be sent sequentially to the destination node. The receiver collects all parts of the messages and assembles them to the original message.

The capability to send and receive *Large Message Transmissions* is optional, if a node sends a *Large Message Transmission* to a node not capable of this feature the receiver shall reply with an *Error Notification Message*.

The *Large Message Transmission* is a special message inside the protocol. To keep the transmission size as small as possible, the complete header of the original message provided by the sending application will be integrated into the header of the *Large Message Transfer* with the modifications that the most significant bit of the message type field is set and the timestamp of each packet is independent from the timestamp of the original message.

Large Message Transmissions are always reliable transmissions. Every segment of this transmission will be acknowledged individually (either positively or negatively) according to the timestamp of the segment.

C. Behavior Description

1) Reliability

Central paradigms of the protocol are reliable messages and error detection and handling. The protocol provides guaranteed delivery services and timeout mechanism for reliable message transmission. The messages transmitted follow the reliability mechanisms that are described as follows.

a) Single Packet Messages

As every message is sent independently, the reliability mechanisms are also applied to every single message. Therefore, each message that is received and has a reliable message type has to be acknowledged. The reliability mechanism is divided into three phases: acknowledgment, resending and error notification phase.

Acknowledgment Phase:

Three different cases for a sent message have to be considered:

When the message was received successfully, a positive acknowledgment will be returned to the sender and the transmission is complete.

The second case is that the message has been received erroneous. With help of the checksum appended to every message, the receiver can check the integrity of the message. If it was received with errors, a negative acknowledgment will be sent, which triggers the sender node to switch to the resend phase.

The last possibility is the loss of the message on its way through the network. In this case, the receiver will not send any acknowledgment either positive or negative. Therefore, the sender waits a defined time for the acknowledgment to arrive and if this time passes, it will switch to the resending phase.

Resending Phase:

When a message was not transmitted successfully on the first try the sender will attempt to resend it. The number of attempts is configurable. The sender again expects a message acknowledgment. If, at a certain try, the message will be acknowledged the transmission is completed. If the limit of resending is reached the system will assume that the link to the receiver is faulty and switches to the error notification phase.

Error Notification Phase:

In this phase of a transmission, it is very likely that the receiving node has lost the connection to the network, as it does not respond, although the sender has repeatedly tried to communicate with it. Another error could be that the sender itself has lost the connection to the network and could not communicate to other nodes.

To check which node lost the connection and to inform the Master in the network about the error an error notification message will be sent to the Master node.

This error notification message is also a reliable message but it is handled differently. This message will only follow the process up to the resending phase. If that phase fails, most likely the sending node has lost its connection to the network and cannot even reach the Master node. At this point, the node should shut itself down, to save energy and not to interfere with the rest of the system.

b) Large Messages

Messages which are too large to fit into one packet should be treated as *Large Message*. For single messages transmitting segments of the *Large Message Transmission*, the reliability mechanisms work as they do for normal single messages. Additional to the reliability mechanisms for single segments, there are some extensions for the *Large Message Transmission*. When receiving the last segment of such transmission, the application has to check that no segment is missing. A *Large Message Transmission* is only successful when all segments of this transmission have been received. If segments are missing, a negative acknowledgment is sent to initiate retransmission.

2) Push Transmissions

Push Transmission are following the publish/subscribe pattern. A producer of data can have many consumers, which subscribe to it. Whenever new data is available, the producer will send the new data to all of its consumers.

The central points of this transmission are the two *Data Transmission* messages. Whenever a producer of information has new datasets, it will create a *Data Transmission* message for each subscribed node and send it.

Depending on the requirements to the delivery of the data, the application can send either a reliable or unreliable data messages.

3) Pull Transmissions

The *Pull Transmission* follows a request-response behavior. It can be used to either trigger an action on a remote node of the network or requesting specific data from it.

To start a request the initiating node has to send a *Data Request* message to the destination node. The request will always be acknowledged, which tells the requesting node that the request will be handled.

The response to these requests will be transmitted with a *Data Response* message. Additional to the normal *Data Transmission*, which is used by the push transmission, it carries the timestamp of the requesting message with it to assign the response to its requesting message.

Data Requests will be sent asynchronously to enable long responding times for the requested action and data.

4) Reconfigurations

The ScOSA system is designed to have one global configuration for all nodes. Therefore, the protocol has to provide means to distribute reconfiguration information to all nodes, to maintain a concise system state.

Reconfiguration can have several reasons. One reason is the change to a new mission phase of the system so that the nodes of the network can be assigned different tasks. Another reason can be the failing of a node so that another node has to take over the tasks of the failed node. Despite the reason, all changes of the configuration have to be initiated by the Master by sending a *Reconfiguration Request* Message. The other nodes in the network are not allowed to send this request message.

On reception of this message the receiving node will change into the so-called “reconfiguration state”. When it reaches this state, it will send all pending messages but does not accept sending new messages. Messages received in this state will be handled as usual. With this method, it can be assured that most of the messages will not get lost during reconfiguration.

After a certain timeout, which has to be configured mission-specifically, the node will delete all of its pending messages, switch into the new state and go back into running state.

The reconfiguration only affects the endpoint nodes in the network. For other network components (e.g. routers and switches), a proper protocol for configuring those components has to be chosen. In a SpaceWire network one can choose the RMAP Protocol [2] to configure the Routing tables. Therefore, the SpaceWire-IPC is implemented in that way that it does not interfere with other protocols for reconfiguring other network components (e.g. using different protocol identifiers at the underlying protocol).

5) Large Message Transmission

The *Large Message Transmission* is a special mode for transmitting messages in the distributed system. This mode of transmission provides a way to send encapsulated messages that would otherwise exceed this size restriction. Every other message used in this system can be encapsulated into a *Large Message Transmission*.

Sending of an oversized message is completely transparent to the application whatever transmission (normal or large

message transmission) is needed. The protocol implementation automatically determines if it is needed to send the message as *Large Message Transmission* depending on its size.

The size of one single packet must be defined between all nodes in the network uniformly. The data itself will be handled as an array of bytes.

The sender first assigns a unique transmission ID to this data and then separates the array into segments. These segments will then transmitted with the same Transaction ID and the corresponding sequence number. The sequence number is used to calculate the offset of this segment in the array.

On the other end of the connection, the receiver will provide a special handling of incoming *Large Message Transmissions*. Instead of notifying the application for every received packet, the handler will collect all the parts belonging to this *Large Message Transmission* according to the same transmission ID and the same Sender ID.

After receiving all parts of a transmission, the handler will reconstruct the encapsulated message and then send it to the normal handler where the original message will be handled transparently.

D. Integration with SpaceWire

The SpaceWire Specification allows custom protocols to be transported as payload. Therefore, a field in the header is reserved to specify the used protocol [1].

The SpaceWire Protocol supports two addressing modes, logical addressing and path addressing [3]. Both methods are possible with SpaceWire-IPC, but for simplicity, only logical addressing is supported by now.

The *Node ID* will be mapped to a SpaceWire logical address with the following pattern. The least significant byte will be directly mapped to the *SpaceWire Address*. The most significant byte will then determine the service running on this node. This mapping limits the maximum addressable services to 256 services per physical node and 256 physical nodes connected to the SpaceWire Network.

V. Protocols Assessment for Reconfigurable Distributed Environment

In this section, SpaceWire-IPC and other SpaceWire based protocols mentioned in Section II are assessed focusing on IPC in distributed on-board computers.

Although reliable communication in RMAP can be established by requesting acknowledgments, the protocol does not fit completely into the requirements for our distributed system. In detail the lack of distributing timestamps and especially heartbeats is a problem. Additionally, fragmentation of large data is not supported by RMAP. Besides this, a specific reconfiguration message type is required to modify the state of the distributed system.

The CCSDS PTP only serves as a frame for more complex protocols without providing properties like data validity checks or reliable data transfers, which are required for our distributed system.

Packet types of GRDDP are defined. However they are insufficient to cover all requirements given by ScOSA, such as the lack of error notification or reconfiguration handling.

For STUP, data retransmission, segmentation of large messages and flow control need to be implemented explicitly by application users. Therefore this protocol does not cover any of our requirements related to IPC communication.

Although SpaceWire-R supports reliable data transmission and heartbeat, it does not include any message types for error notification and reconfiguration. The pull request is not implemented within this protocol. SpaceWire-R can only send reliable data and lacks the unreliable data transmission. This is necessary for high-frequency transmissions, where new data will arrive quickly, and losing some packets is considered uncritical. Although, it shares some concept with the SpaceWire-IPC protocol, it is still not fully suitable for the ScOSA use case.

TABLE II summarizes these SpaceWire based protocols and SpaceWire-IPC in terms of features of IPC. As it can be seen from TABLE II, RMAP, CCSDS PTP, GRDDP, STUP and SpaceWire-R are not targeting IPC services in SpaceWire networks. However, the IPC services are necessary for a pure COTS or hybrid reconfigurable distributed on-board computers. SpaceWire-IPC offers features for IPC, supporting monitoring, management and reconfiguration, which then can be implemented on higher level.

TABLE II. Comparison of SpaceWire Based Protocols

Features	**RMAP**	**CCSDS PTP**	**GRDDP**	**STUP**	**SpaceWire-R**	**SpaceWire-IPC**
Data Correctness Check	×		×	×	×	×
Data Retransmission			×		×	×
Multiplexing			×		×	×
Segmentation / Large Message Transmission					×	×
Flow Control					×	
Keep Alive / Heartbeat / Monitoring Support					×	×
Reconfiguration Support						×
Error Notification to Manager						×
Publish /Subscribe						×
Request-Response	×			×		×

With SpaceWire-IPC, *Data Request*, *Data Response* or *Data Transmissions* can be used for application data exchange and to request or to publish state values for plausibility checks. Applications can set the timestamp for data transmission and let SpaceWire-IPC take care of the sending timestamp. Heartbeats can be used by the Master to monitor the whole distributed network and by Observers to monitor the Master or Observers of higher priorities. Message Acknowledgment is

for reliable data transmission and detecting failures of a link or no response of a node. Reconfiguration Request can be used for initial configuration, reconfiguration due to failures and reconfiguration for new-phase missions. Error Notification is to inform Master the error reason for FDIR. Large Message Transmission can meet the increasing demands of image processing on-board for earth observation activities by transferring raw large images to several nodes for parallel processing.

VI. Conclusions

In this paper we presented the SpaceWire-IPC for reconfigurable distributed on-board computers. With this protocol, SpaceWire networks can support IPC for distributed computing on a spacecraft. We highlighted the reconfiguration feature supported by the SpaceWire-IPC, which enables COTS hardware to be used on-board with reliability and fault tolerance. With COTS nodes, high performance demands can be enhanced for future applications.

Because the SpaceWire network is not fully integrated yet, it will be part of the ScOSA project to address this issue and to embed the introduced SpaceWire-IPC. Besides the physical implementation of a SpaceWire network and the proposed IPC protocol, it is also required to provide software driver support for all peripheries depending on the selected operating system.

After implementation, the measurement and performance analysis will be carried out.

References

[1] "Space engineering. SpaceWire protocol identification," ECSS-E-ST-50-51C, ESA-ESTEC Requirements & Standards Division, Noordwijk, 2010.

[2] "Space engineering. SpaceWire - Remote memory access protocol," ECSS-E-ST-50-52C, ESA-ESTEC Requirements & Standards Division, Noordwijk, 2010.

[3] "Space engineering. SpaceWire - Links, nodes, routers and networks," ECSS-E-ST-50-12C, ESA-ESTEC Requirements & Standards Division, Noordwijk, 2008.

[4] "Space engineering. SpaceWire - CCSDS packet transfer protocol," ECSS-E-ST-50-53C, ESA-ESTEC Requirements & Standards Division, Noordwijk, 2010.

[5] "GOES-R Reliable Data Delivery Protocol (GRDDP)," 417-R-RTP-0050, NASA Goddard Space Flight Center GOES-R Project, Greenbelt, 2008.

[6] P. Rastetter, U. Liebstückel and S. Fischer, "STUP SpaceWire Protocol," SMCS-ASTD-PS-001, 2009.

[7] "SpaceWire-R," SCDHA 151-0.4, Japan Aerospace Exploration Agency (JAXA), Institute of Space and Astronautical Science (ISAS), 2015.

[8] "Project information OBC-SA," [Online]. Available: https://scrivito-public-cdn.s3-eu-west-1.amazonaws.com/fokus/public/57b85e4561eb7de5/1683f2d1b44c512887644ed0eac105fd/Projektblatt_OBCSA_EN.pdf. [Accessed 15 August 2016].

[9] C. Wilson, J. Stewart, P. Gauvin, J. MacKinnon, J. Coole, J. Urriste, A. George, G. Crum, E. Timmons, J. Beck, T. Flatley, M. Wirthlin, A. Wison and A. Stoddard, "CSP Hybrid Space Computing for STP-H5/ISEM on ISS," in *Small Satellite Conference*, Logan, 2015.

[10] M. Fayyaz and T. Vladimirova, "Fault-Tolerant Distributed approach to satellite On-Board Computer design," in *2014 IEEE Aerospace Conference*, Big Sky, 2014.

[11] D. Lüdtke, K. Westerdorff, K. Stohlmann, A. Börner, O. Maibaum, T. Peng, B. Weps, G. Fey and A. Gerndt, "OBC-NG: towards a reconfigurable on-board computing architecture for spacecraft," in *Proceedings of IEEE Aerospace Conference*, Big Sky, Montana, 2014.

[12] I. T. Union, "X.200: Information technology - Open Systems Interconnection - Basic Reference Model: The basic model," 11 June 1994. [Online]. Available: http://www.itu.int/rec/T-REC-X.200-199407-I. [Accessed 08 August 2016].