Network Topology Transformation for Fault Tolerance in SpaceWire Onboard Networks

Irina Lavrovskaya, Valentin Olenev
Saint-Petersburg State University of Aerospace Instrumentation
Saint Petersburg, Russia
{irina.lavrovskaya, valentin.olenev}@guap.ru

***Abstract*—The paper presents a network transformation algorithm for fault tolerance in SpaceWire onboard networks which is implemented in SANDS computer-aided design system. We give general notions on fault tolerance for onboard networks, introduce our algorithm for network transformation and give several examples of running the algorithm on different topologies.**

I. Introduction

Nowadays there are a lot of computer-aided design systems (CAD) but none of them is intended for SpaceWire networks. According to our review in [1] there are several network simulation tools for SpaceWire such as MOST [2], Sandia National Laboratories simulator [3], VisualSim [4] and DCNSimulator [5]. However, these are just simulation tools without any functionality for network design and analysis of structural and fault tolerance characteristics of the designed topology. We are currently working on development of a SpaceWire Automated Network Design and Simulation (SANDS) computer-aided design system which will support full onboard network design and simulation flow, which begins from the network topology design and finishes with getting the simulation results, statistics and different diagrams.

Such computer-aided design system is highly-demanded as evolution of microelectronics has led to the growth of the onboard networks and systems sizes. The onboard networks for spacecraft and avionics require good fault tolerance characteristics in order to continue their operation in case of faults and failures. One of the most popular spacecraft onboard technologies is SpaceWire [6]. It is a technology which is being actively integrated into new generation spacecraft.

This paper is a logical continuation of our previous paper [7] where we introduced a fault-tolerance analysis algorithm for SpaceWire onboard networks. The problem we address in the current paper is network topology transformation for achieving required by user fault tolerance in onboard networks. We propose an algorithm which uses already implemented functionality of network topology fault tolerance evaluation.

The paper is organized as follows. In section II we start with a brief description of a new computer-aided design system for SpaceWire onboard networks SANDS. Section III gives general notions on fault tolerance for onboard networks. Then in section IV we review the current state-of-art in the field of network topology transformation and reconfiguration. In section V we describe our algorithm of network topology transformation for fault tolerance. Section VI deals with a problem of transformation of a network consisting of several regions. Then, in section VII we give several examples of network transformation in SANDS. Finally, section VIII concludes the paper.

II. SpaceWire Automated Network Design and Simulation

In [1], [7] we proposed a concept of SANDS, which will provide wide functionality for onboard network design. Implementation of such kind of a design and simulation toolset will give an ability for spacecraft designers to design the onboard network with all its technical characteristics and features, distribute the data flows and simulate it taking into account real latencies, different errors, etc. We plan to implement a flexible system, with possibility of addition new protocols. Currently, we plan to include in SANDS implementation of the SpaceWire protocol and two transport layer protocols: RMAP [8] and STP-ISS [9].

SANDS architecture includes four main components:

- Component #1: A component for onboard network topology design and evaluation of its structural characteristics;
- Component #2: A component for tracking of the routes for the data transmission in a network;
- Component #3: A component for generation of the scheduling table for the STP-ISS transport protocol for the transmission of the data with Scheduled quality of service;
- Component #4: A component for simulation of the network operation with all the data that component got from other 3 components and graphical user interface (majorly redesigned DCNSimulator) [5].

Visualization and graphical interface will be taken from the VIPE project [10]. Graphical user interface (GUI) will provide the visual network composition and management capabilities. It will allow designing SpaceWire network topology in visual interactive way from components. The component library is a replenish set of network nodes and switches relevant to physical devices that are available for network building. The graphical user interface to work with the Component #1 is shown in Fig. 1.

ISSN 2305-7254

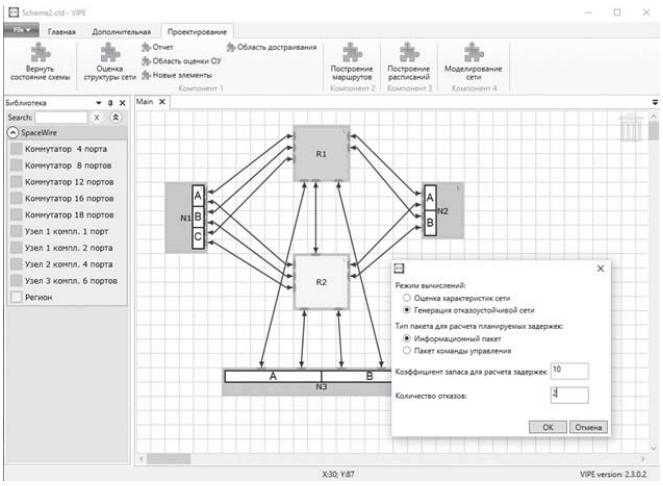

Fig. 1. SANDS graphical user interface

In this paper we focus on the Component #1 which is responsible for the following tasks:

- network topology design;
- evaluation of structural characteristics of the designed topology;
- network topology transformation for achieving required fault-tolerance.

Network topology design assumes creation of a network topology by means of GUI and setting up parameters for nodes, switches and links. Fault-tolerance is a very important characteristic for onboard networks, especially for the domain of long-term satellites. It is a common practice when the network topology contains redundant nodes and links.

Nowadays, sizes of the network topologies increase with the growing demands of industry. That is why it is so important to have an opportunity for an automated network topology modification to achieve the required by the user fault tolerance.

III. Fault tolerance in onboard networks

In designing or selecting a topological structure of onboard networks for a system, one of fundamental considerations is the fault-tolerance [11]. In the context of this paper we will assume that fault tolerance is a property that enables a system to continue operating properly in the event of the failure or one or more faults within some of its components. Basically, any system containing redundant components or functions has some of the properties of fault tolerance [12].

Systems such as communication onboard networks have many nodes representing processors, sensors, control units, memory, etc. that desire to communicate and also have several links providing a number of interconnected pathways. These many interconnections increase reliability and topology complexity. As all these devices are connected to such a network, a failure or fault affect many people; thus the reliability goals must be set at a high level especially in the domain of spacecraft and avionics.

Generally, the onboard network is assembled from three main types of elements: terminal nodes, routers and links. Each of these elements can fail so that it cannot be repaired in any considerable time. However, a fault-tolerant system should continue its proper operation. This can be achieved by adding redundancy to the onboard network. For example, one terminal node can be represented by two or three redundant units. When one of redundant units fails another unit continues to operate properly, replacing the failed one.

IV. Related work

The problem of network transformation and reconfiguration in onboard communication networks with irregular topology is not widely covered in related studies. However, there are some works which discuss similar problems in the domain of networks-on-chip and wireless networks.

In [13] the authors solve a problem of generation of a topology such that all communicating cores of the application can transmit data to each other over the network with at least two alternative paths and with minimal energy consumption. Fault-Tolerant Topology Generation (FTTG) algorithm, presented in this paper, has two main phases: 1) generating non-fault-tolerant irregular topology using a minimum number of routers and links, and 2) adding extra routers and links to obtain a fault-tolerant version of the topology. FTTG algorithm generates topologies such that each router of the topology can be reached from any router with at least two alternative paths. The generated topology can be used to tolerate at least one link failure by applying the packet's alternative routings.

There are some works on similar problems in the domain of wireless networks. In [14] Sitanayah et al. address the problem of finding a minimal set of relays which ensures k node-disjoint paths for each sensor. The authors propose GRASP-ARP, a local search algorithm based on greedy randomized adaptive search procedure (GRASP) [15], to be run as a centralised offline algorithm during the initial topology design, i.e. prior to network deployment and operation. GRASP is a metaheuristic intended to capture the good features of pure greedy algorithms and of random construction procedures. The algorithm requires repeated counting of the number of node-disjoint paths for each node, for which a dynamic programming procedure is used.

Another related work is [16]. Kashyap et al. consider a problem of constructing a fault-tolerant backbone wireless network using additional relay nodes. They give *O(1)*-approximation algorithms for 2-edge and 2-vertex connectivity in terms of the number of relay nodes required. The algorithms also work for achieving k-connectivity for higher values of k. In the presented algorithm the number of additional relay nodes depends on the weight of the edges in the spanning tree graph. Placing of relay nodes in the graph is followed by optimization phase. Optimization is performed by one by one removing added relay nodes. K-edge connectivity is checked after each removal. If the graph is still k-edge connected, repeat for another relay node, else put back the current node and corresponding edges, and repeat with the next relay node.

There is only one research result, that considers fault tolerance for SpaceWire networks, is available for public - [17]. In that paper the authors propose the methodology and toolset for SpaceWire network design. This methodology allows generating fault tolerant networks with variable level of tolerance.

V. NETWORK TOPOLOGY TRANSFORMATION ALGORITHM FOR FAULT TOLERANCE

The problem we address in this paper is network topology transformation for increasing fault tolerance in the network. Modern onboard networks consist of a huge number of computers, telemetry, radio-transmitting and data transmitting devices. The bigger and more complex the onboard network is, the easier for a network topology designer to make a mistake in fault-tolerance structure. Therefore, we propose to apply computer aided network topology transformation to achieve a required fault tolerance. This feature is included to the Component #1 in SANDS.

The problem can be described as follows. Given a network topology, place additional redundant units for terminal nodes, additional routers and communication links to achieve the required by user fault tolerance.

We propose to solve the stated problem in two stages which are described below.

- Stage 1: Initialise a topology of a network with required fault tolerance f.
- Stage 2: Iterative improvement of the topology obtained on the stage 1.

Below we will give a general description of these two stages.

A. Stage 1. Creation a topology with required fault-tolerance

1) Step 1. Fault tolerance analysis. Firstly, it is necessary to evaluate fault tolerance of the initial network or its part. In case of fault tolerance of the initial network is less than required fault tolerance, then we move to the Step 2. Otherwise, there is no need to run the algorithm.

2) Step 2. Addition of redundant units to terminal nodes. This step checks the required number of units in terminal nodes in order to provide required fault tolerance. If it is necessary, additional redundant units should be added to terminal nodes. This step should be performed only when the initial network or its part contains terminal nodes.

All terminal nodes shall be checked one by one. For each node we propose to calculate the number of missing ports which are needed to provide required fault tolerance. The calculation is performed according to the formula below:

$$need\_ports = (F + 1) - current\_ports,$$

where *need_ports* – is a number of missing ports in a terminal node, F – required fault tolerance of the network, *current_ports* – current number of ports in a terminal node. If *need_ports* is more than *1*, it means that it is necessary to add redundant units to the node.

When redundant units are added to the terminal nodes, these units should be connected to routers. Just before connecting redundant units and routers, we should check if there are enough free ports in routers to connect all redundant units. If there are not enough free ports in routers, then move to Step 3. Otherwise, connect redundant units to routers. Each unit of a terminal node should be connected to different routers in order to increase fault tolerance of a network.

If all ports of all terminal nodes are connected to routers, then it is necessary to evaluate fault tolerance of the resulting network or its part. The network transformation can be considered as completed if new fault tolerance corresponds to the fault tolerance required by the user. If this is not the case, move to Step 3.

3) Step 3. Addition of redundant communication links and routers. This step checks the number of routers in the network or its part, adds new communication links and new routers to the network topology. This can be done in the following ways:

- connecting all routers in the initial network structure through additional links.
- replicating an initial network structure with all routers and links (except nodes) and then by connection of replicas into one network structure. Connection of replicas is performed by adding links between the routers' replicas.

Different actions should be performed in dependence on the number of routers in the network or its considered part. Two main cases should be considered on this step:

- The number of routers is more than the required fault tolerance;
- The number of routers is less or equal to the required fault tolerance.

Let us firstly consider the case, when the number of routers in a network or its considered part is more than the required fault tolerance. First of all, the routers should be connected with each other by additional links according to the rule below:

- There are free ports available in considered routers;
- These routers are not already connected by a direct communication link.

We should add only one link between each pair of routers. Addition of links between routers can impact the connectivity of the network and, consequently, increase its fault tolerance.

It is important to note that new links can be added not for all pairs of routers, as some of routers do not have enough free ports. Furthermore, new links can even not be added, due to free ports lack. New links can be added between only particular routers, just for those that provide free ports and were not connected in the initial network topology.

Provided that new links were added, it is necessary to evaluate fault tolerance of the resulting network or its part. If the evaluated fault tolerance corresponds to the required by the user fault tolerance, then the algorithm moves to the Stage 2.

Otherwise, it is necessary to introduce new routers to the network or its considered part.

New routers are introduced by replicating an initial network structure or its part with all routers and links (except nodes). The number of replicas shall be calculated as a difference between the required fault tolerance and current fault tolerance of the network or its considered part. For example, if the required fault tolerance $F_{req} = 1$ and current fault tolerance $F_{cur} = 0$, then *1* replica should be created.

Replicating shall be performed in accordance with the following rules:

- A copy for each router in the network or in its part shall be created;
- The network structure replica shall contain similar links as the initial network structure;
- Nodes are not included into replicas;
- Connections to nodes shall not be replicated.

The replicas of network structure should be connected with each other and with the initial network structure. If the initial network structure contains a router R and a replica of the network structure contains a copy R' of this router, then R and R' should be connected by a link. This is done in order to increase network connectivity. However, providing that there is a small quantity of ports in router, R and its copy R' can be left unconnected.

Finally, if there are unconnected ports in terminal nodes, they should be connected to routers. When all terminal nodes ports are connected, the algorithm moves to Step 4.

4) Step 4. Fault tolerance analysis of the transformed network. The final step of Stage 1 evaluates the fault tolerance of resulting network structure which was obtained after previous steps. If the resulting fault tolerance is equal or even more than the required fault tolerance, then the algorithm moves to the Stage 2. If this is not the case, it means that the particular initial network topology cannot be transformed to obtain the required fault tolerance. This can occur when network routers have a very small number of ports.

This is the final step of stage 1 which is followed by stage 2.

B. Stage 2. Iterative improvement of the topology obtained on the stage 1.

Improvement consists in one by one removing added routers and links and then checking fault tolerance of the modified network. Addition of new links and routers can cause excessive redundancy. This excessive redundancy can be unnecessary for achieving required fault tolerance. For the purpose of reducing hardware costs, we need to remove extra routers and links.

The algorithm of iterative improvement of the topology is defined below:

1. The resulting network shall be represented as a graph. While graph creation it is necessary to take into account that the network structure can be of arbitrary topology and may contain redundant nodes. We have set up a rule for graph construction (see Fig. 2):

- All redundant units of a node shall be associated with only one vertex of the graph.
- All routers shall be represented by separate graph vertices.
- All links shall be represented by graph edges.

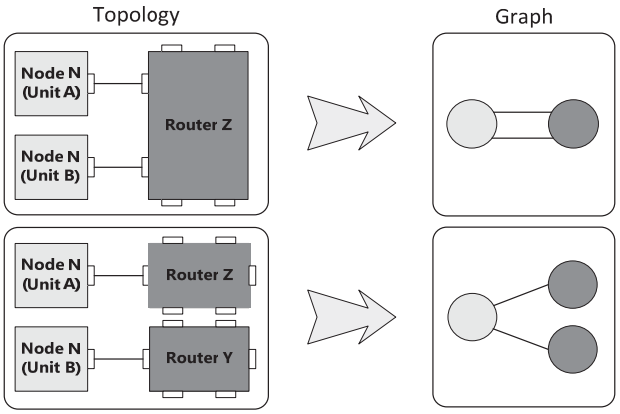

Fig. 2. Transformation of network topology to a graph

SpaceWire standard provides bidirectional links, so each link shall be represented by a pair of edges with opposite orientations (see Fig. 3). We use the digraph for the *k*-connectivity analysis because we should apply maxflow algorithms which work in directed graphs.

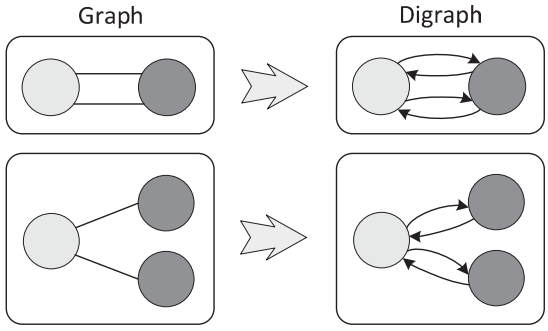

Fig. 3. A digraph of a network topology

2. Specify a set of graph vertices V_r which correspond to newly added routers.

3. Specify a set of graph edges E_l which correspond to newly added communication links.

4. If set V_r is not empty, perform the following actions for each vertex from V_r:

a. Remove the current vertex and all its incident edges.

b. Check fault tolerance of the modified network.

– If the router removal leads to deletion of links with one or more terminal nodes, then the current router with all corresponding edges shall be put back to the network topology.

- If the modified network is still k-connected, then continue with the next router.
- Otherwise, put the removed router with all corresponding edges back to the network topology.

c. Stop when all added routers have been considered.

5. If set E_l is not empty, perform the following actions for each edge from E_l:

a. Remove the current edge from the graph. It should be mentioned that one communication link is represented in a graph by two edges with opposite directions. Consequently, removal of one communication link leads to deletion of two edges in a graph.

b. Check fault tolerance of the modified network.

- If the modified network is still k-connected, then continue with the next link.
- Otherwise, put the removed edges back to the graph and continue with the next link.

c. Stop when all added links have been considered.

6. Finally, the resulting network should be passed to the graphical user interface and presented to the user.

The network topology transformation algorithm can be applied either to the whole network or its part, defined by the user. This feature makes possible to provide different fault tolerance for different parts of a network. The user should perform network transformation sequentially for particular parts of the network.

The proposed algorithm is able to increase fault-tolerance for almost any network structure or its part. However, there are particular cases in which the transformation could not be performed – network topologies with routers with small amount of ports and big amount of nodes.

VI. Transformation of a Network with Regions

SpaceWire network can consist of a large number of terminal nodes, so the network is better to be divided into a set of special network regions. Each region can consist of routers and nodes. The problem of network transformation for such a network is more complex than for a single region SpaceWire network. Providing that the initial network consists of several regions, network transformation shall be done in the following way:

1) Perform transformation for each network region separately (Stage 1, Stage 2). Region transformation shall be done similarly to transformation of a network part.

2) Perform transformation for terminal nodes and routers which are not included into regions (Stage 1, Stage 2).

3) Connect all routers by additional links. New links shall be added only for those pairs of routers which are located in different regions or out of regions. Routers which belong to one network region shall not be connected by additional links. Then perform Stage 2 of the general algorithm.

VII. Examples of Network Transformation in SANDS

The presented algorithm of network topology transformation for achieving the required fault tolerance was implemented as a part of Component #1 in SANDS. It is implemented in C++ and uses the functionality of fault tolerance evaluation described in [7]. In this section we present the results of implemented algorithm work. Firstly, we show examples for network topologies without regions, and then, we present an example of transformation of a network with regions.

A. Example without regions

The first example is a network, which consists of two terminal nodes and five routers (see Fig. 4). This initial network topology is not fault-tolerant, as the correspondent graph is *1*-connected.

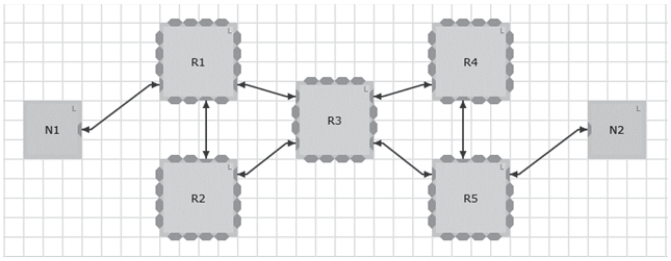

Fig. 4. Example 1, initial topology

We ran our algorithm so that the network can tolerate 2 faults. The result of network transformation is shown in Fig. 5. All new links are shown in heavy lines and extended nodes are shown in dark frames.

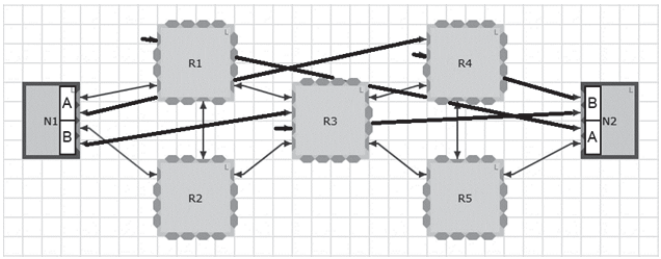

Fig. 5. Example 1, transformed topology

Our network transformation algorithm added redundant units to the terminal nodes N1 and N2. After transformation each terminal node contains two units – *A* and *B*. Each unit of a terminal node contains two SpaceWire ports which are connected to different routers in order to increase fault tolerance. The algorithm added 6 additional communication links. The resulting network topology is 2-fault-tolerant.

Next example initial topology is shown in Fig. 6.

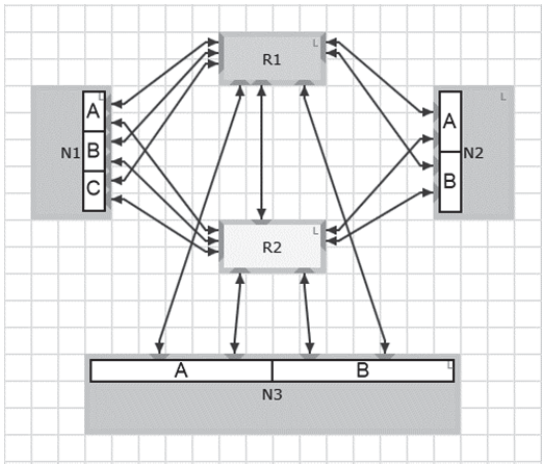

Fig. 6. Example 2

This network topology consists of three terminal nodes with multiple redundant units (units correspond to letters *A, B* and *C* in *N1, N2*, and *N3*), two routers and it is *1*-fault-rolerant.

Similarly to the previous example, we ran our algorithm to obtain a network which is *2*-fault-tolerant. The result of network transformation is shown in Fig. 7. All new links are shown in heavy lines and new routers are shown with dark frames.

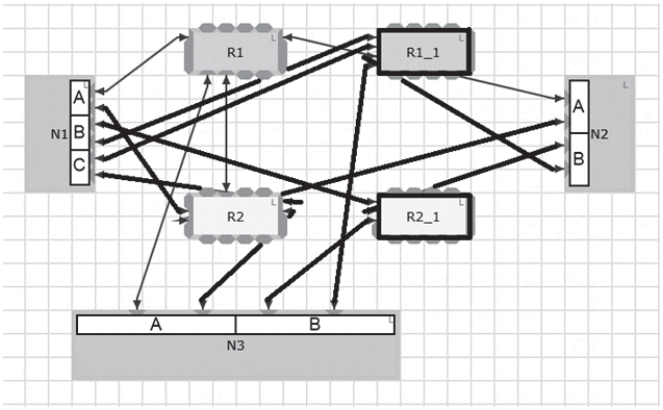

Fig. 7. Example 2, transformed topology

The network transformation algorithm added two routers and a number of new links. In order to increase fault tolerance, the algorithm disconnected all terminal nodes ports except one, added one routers structure replica and connected nodes' ports with a new set of routers. In order to reduce redundant links during the stage 2 of the algorithm 5 communication links were removed. The transformed network topology is *2*-fault-tolerant.

B. Example with regions

Finally, let us show an example of algorithm application to a network with regions (see Fig. 8). The initial network topology consists of three regions. The whole network is not fault-tolerant, however, Region 3 is *1*-fault-tolerant, because of cross-connections between nodes' units and routers R3 and R4.

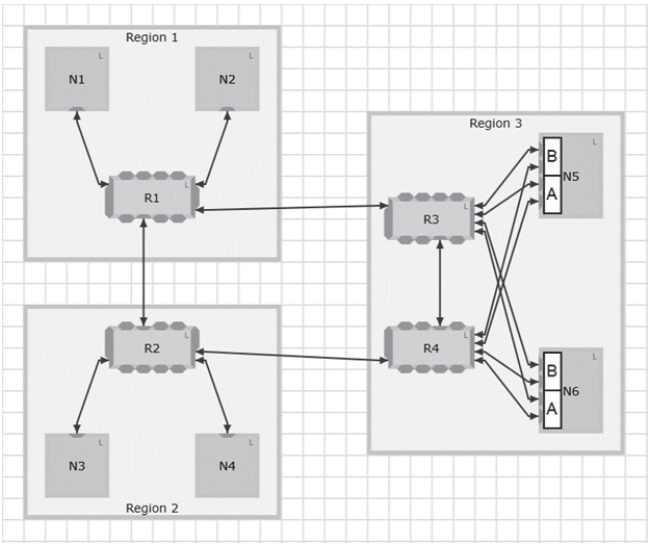

Fig. 8. Example 3

We ran our algorithm to obtain a network which is *1*-fault-tolerant. The result of network transformation is shown in Fig. 9. All new links are shown in heavy lines and new routers are shown with dark frames.

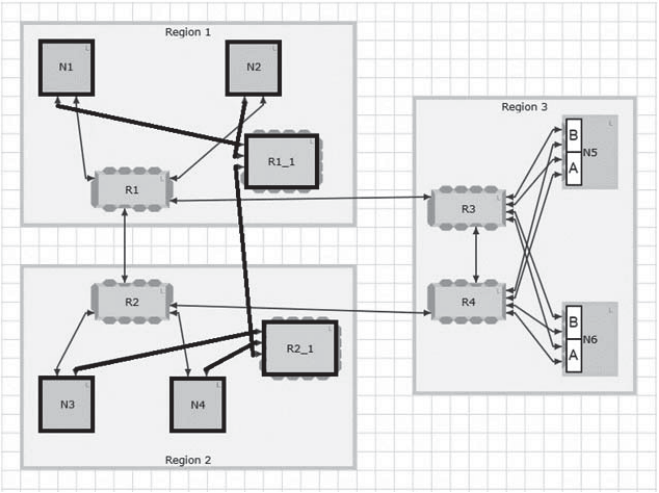

Fig. 9. Example 3, transformed topology

Our network transformation algorithm added additional redundant ports to the terminal nodes N1, N2, N3 and N4 for cross-connections. Each terminal node in Region 1 and Region 2 contains two SpaceWire ports which are connected to different routers in order to increase fault tolerance. Moreover, we can observe two new routers R1_1 and R2_1, which were added to increase fault tolerance in Region 1 and Region 2. The algorithm added 5 additional communication links. In order to reduce redundant links during the stage 2 of the algorithm 7 communication links were removed (not shown in Fig. 9). Region 3 was not transformed as it was originally *1*-fault-tolerant. The resulting network topology is *1*-fault-tolerant.

VIII. CONCLUSION

In the current paper we address the problem of network

topology transformation to achieve required by the user fault tolerance. Firstly, we introduced SANDS – a toolset for SpaceWire Automated Network Design and Simulation intended for SpaceWire networks. SANDS is designed for solving important tasks, which developers face with during implementation of onboard systems and networks. SANDS consist of four main functional components. One of the key features of Component#1 in SANDS is network topology transformation for fault tolerance.

Our contribution is a proposal of an algorithm for network transformation for required fault tolerance. This algorithm uses already implemented in Component #1 functionality of network topology fault tolerance evaluation. Network transformation can be done either for the whole network topology or for its part. Moreover, we specified a solution for a network consisting of several SpaceWire network regions.

We are still working on improvement of implementation performance. Moreover, we plan to make an additional study on optimization of the resulting network topology to decrease the number of additional routers.

ACKNOWLEDGMENT

The research leading to these results has received funding from the Ministry of Education and Science of the Russian Federation under the contract RFMEFI57816X0214.

REFERENCES

[1] Sheynin Y., Olenev V., Lavrovskaya I., Korobkov I., Kochura S., Shkolniy V., Dymov D. "Computer-Aided Design System for On-board SpaceWire Networks Simulation and Design", *in Proceedings of the 20th Conference of Open Innovations Association FRUCT*, LETI University, St. Petersburg, Russia, 2017, pp. 398-405.

[2] B. Dellandrea, B. Gouin, S. Parkes, D. Jameux, *"MOST: Modeling of SpaceWire & SpaceFiber Traffic-Applications and Operations: On-Board Segment"*, Proceedings of the DASIA 2014 conference, Warsaw, 2014.

[3] B. van Leeuwen, J. Eldridge, J. Leemaster, *"SpaceWire Model Development Technology for Satellite Architecture"*, Sandia Report, Sandia National Laboratories 2011, 30 p.

[4] Mirabilis Design, "Mirabilis VisualSim data sheet", 2003. 4 p.

[5] A. Eganyan, E. Suvorova, Y. Sheynin, A. Khakhulin, I. Orlovsky, *"DCNSimulator – Software Tool for SpaceWire Networks Simulation"*, Proceedings of International SpaceWire Conference 2013, 2013, pp. 216-221.

[6] ECSS, *SpaceWire — Links, nodes, routers and networks (ECSS-E-ST-50-12C)*. Noordwijk: ESA Requirements and Standards Division, July 2008.

[7] Lavrovskaya I., Olenev V., Korobkov I. "Fault-Tolerance Analysis Algorithm for SpaceWire Onboard Networks" *in Proceedings of the 21st Conference of Open Innovations Association FRUCT,* University of Helsinki, Helsinki, Finland, 2017, pp. 217-223.

[8] ESA. *Standard ECSS-E-ST-50-52C, SpaceWire — Remote memory access protocol.* Noordwijk : Publications Division ESTEC, February 5, 2010.

[9] Y. Sheynin, V. Olenev, I. Lavrovskaya, I. Korobkov, D. Dymov "STP-ISS Transport Protocol for Spacecraft On-board Networks", *Proceedings of 6th International SpaceWire Conference 2014 Program*; Greece, Athens, 2014. pp. 26-31.

[10] Syschikov, A., Sheynin, Y., Sedov, B., Ivanova, V. "*Domain-specific programming environment for heterogeneous multicore embedded systems*", International Journal of Embedded and Real-Time Communication Systems, Volume 5, Issue 4. 2014, pp. 1-23.

[11] Junming Xu "*Topological Structure and Analysis of Interconnection Networks*", Kluwer Academic publishers, Netherlands, 2001, 350 p.

[12] Martin L. Shooman, "*Reliability of Computer Systems and Networks. Fault Tolerance, Analysis, and Design*", Wiley, New York, 2002, 551 p.

[13] S. Tosun, V. B. Ajabshir, O. Mercanoglu and O. Ozturk, "Fault-Tolerant Irregular Topology Design Method for Network-on-Chips," *2014 17th Euromicro Conference on Digital System Design, Verona*, 2014, pp. 631-634.

[14] L. Sitanayah, K. N. Brown and C. J. Sreenan, "Fault-tolerant relay deployment for k node-disjoint paths in wireless sensor networks," *2011 IFIP Wireless Days (WD)*, Niagara Falls, ON, 2011, pp. 1-6.

[15] T. Feo and M. Resende, "Greedy Randomized Adaptive Search Procedures," *Journal of Global Optimization*, vol. 6, pp. 109–133, 1995.

[16] A. Kashyap, S. Khuller and M. Shayman, "Relay Placement for Higher Order Connectivity in Wireless Sensor Networks," *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, Barcelona, Spain, 2006, pp. 1-12.

[17] Alexey Syschikov, Elena Suvorova, Yuriy Sheynin, Boris Sedov, Nadezhda Matveeva, Dmitry Raszhivin, "Toolset for SpaceWire Networks Design and Configuration", *in Proceedings of the 5th International SpaceWire Conference 2013*, 2013, pp.149-153.