Fault-Tolerance Analysis Algorithm for SpaceWire Onboard Networks

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

Abstract—**The paper presents algorithms for fault tolerance evaluation which will be applied in a new computer-aided design system for SpaceWire onboard networks. We give general notions on fault tolerance for onboard networks, and introduce our algorithm for evaluation of fault tolerance in SpaceWire networks. Finally we address our future work direction on network topology transformation for fault tolerance.**

I. Introduction

Evolution of microelectronics has led to the growth of the onboard networks and systems sizes. The onboard networks especially for spacecraft and avionics require good fault tolerance characteristics in order to continue their operation in case of faults and failures.

SpaceWire is a data-handling network for the spacecraft, which combines simple, low-cost implementation with high performance and architectural flexibility [1]. It is a technology which is being actively integrated into new generation spacecraft.

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 [2] there are several network simulation tools for SpaceWire such as MOST [3], Sandia National Laboratories simulator [4], VisualSim [5] and DCNSimulator [6]. However, these are just simulation tools without any functionality for network design and analysis of structural and fault tolerance characteristics of the designed topology. Therefore, we decided to create a new computer-aided design system for SpaceWire networks 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. This CAD system will contain fault tolerance analysis features as a part of it.

The problem we address in this paper is analysis of fault tolerance in SpaceWire onboard networks which is solved through k-connectivity search on a graph.

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 and introduce its first component which is responsible for network topology design and estimation of its structural characteristics. Section III gives general notions on fault tolerance for onboard networks. Then in section IV we review the current state-of-art in this field. In section V we describe our algorithm for evaluation of fault tolerance in networks. Section VI proposes an additional feature for future work and, finally, section VII concludes the paper.

II. Network Topology Design and Estimation of Structural Characteristics of SpaceWire Onboard Network

In [2] we proposed a concept of a new computer-aided design system for SpaceWire networks, 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 SpaceWire network with all its technical characteristics and features, distribute the data flows and simulate it taking into account real latencies, different errors, etc.

Each node of the designed network will include implementation of the SpaceWire protocol and two transport layer protocols: RMAP [3] and STP-ISS [8] which we plan to implement in the first version of the CAD for SpaceWire.

For some future implementations of the CAD system we plan to give a possibility to add other space protocols to the CAD system (e.g. SpaceFibre). For this purpose we will use particular library for each protocol implementation, so that in future we will be able to replace the SpaceWire for any other communication protocol (for example, SpaceFibre).

The proposed architecture of the CAD system is shown in Fig. 1. Architecture of the proposed CAD system 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) [2].

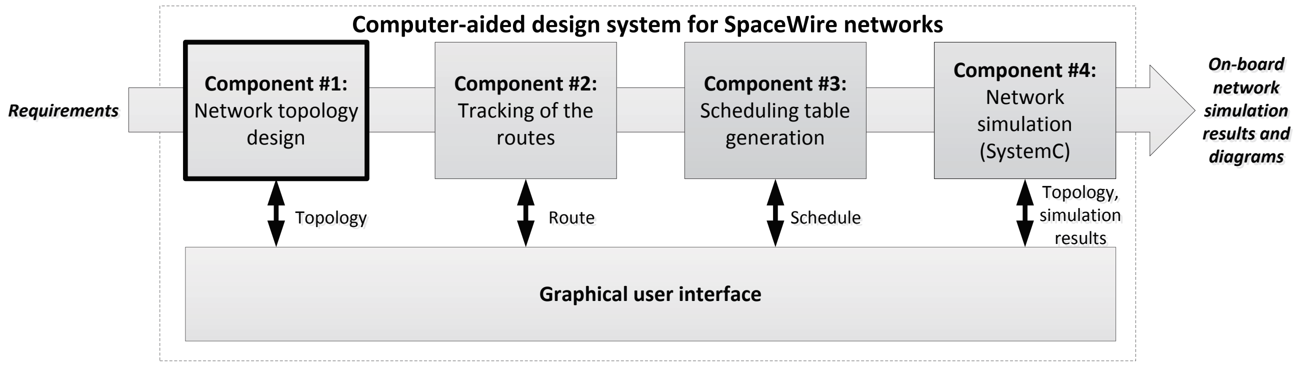
Computer-aided design system for SpaceWire networks
Requirements
Component #1: Network topology design
Component #2: Tracking of the routes
Component #3: Scheduling table generation
Component #4: Network simulation (SystemC)
On-board network simulation results and diagrams
Topology
Route
Schedule
Topology, simulation results
Graphical user interface

Fig. 1. A new computer-aided design system architecture

Visualization and graphical interface will be taken from the VIPE project [9]. 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 (see Fig. 2).

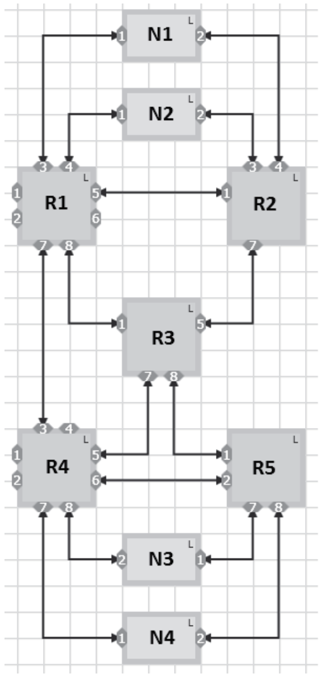

Fig. 2. SpaceWire network scheme in GUI

The component library is a replenish set of network nodes and switches relevant to physical devices that are available for network building. It may also include flexible components if the developer wants to try various combinations of network equipment. For all nodes, switches and channels the GUI will provide the configuration interface to set up their parameters, configure transport protocols and application-level traffic generators.

The designed network will be exported to the intermediate representation format to be used in other CAD components for simulator, routes tracking, scheduling calculation and other tools. GUI will be also able to show results after running each component.

In this paper we will 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. Once this step is completed, the designed network can be analysed. We suppose to estimate the following characteristics of the designed network:

- Network mass: cable mass, switches mass or both;
- Power consumption of network switches;
- Network diameter
- Fault-tolerance of the network or its cluster.

While estimation of the first three characteristics is rather easy, the last one is worth additional research and discussion. 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 obtain an automated estimation of fault-tolerance characteristic in the designed network and remove the human factor for the most part.

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 [10]. 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 [11].

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.

In the current paper we address failures and faults that can occur in terminal nodes, routers and links, i.e. in the elements from which the network is assembled.

IV. Related work

Among a wide range of modern CAD systems for network and SoC design there are no such CAD which provide facilities for fault tolerance analysis of the designed network [12].

In [13] it is stated that a fault tolerance analysis feature is provided by Fault Tolerance Simulation and Evaluation Tool for Artificial Neural Networks (FTSET) which is based on Matlab. This tool is capable of evaluation the fault tolerance degree of a previously trained Artificial Neural Network given its inputs ranges, the weights and the architecture. However, this FTSET evaluates graceful degradation characteristic but not connectivity characteristic of a network which is a subject of interest of this paper.

There has not been any work, other than [14], that considers fault tolerance for SpaceWire networks. 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. However, there are no fault-tolerance analysis algorithm provided in this paper.

V. Fault tolerance evaluation algorithm

A. Graph representation of a network

In the design of an interconnection and onboard networks, one of the most fundamental considerations is the fault-tolerance of the network, which can be usually characterized by connectivity and edge-connectivity of the topological structure of the network. Therefore, we consider a network as a graph $G = (V, E)$, where V is a set of network nodes and routers and E is a set of links between them.

The connectivity of a graph is a good estimate of the fault tolerance (f) of the network, since higher connectivity means more elements can fail without disrupting the communication among the rest of the onboard network [15]. It is important that a graph stay connected when an edge or a vertex is removed. That is, it is necessary to have more than one route between each pair of vertices in a graph, so as to handle possible failures.

A graph is k-connected if there are at least k vertex-disjoint paths connecting every pair of vertices in the graph. There are two types of connectivity: edge connectivity and vertex connectivity. The vertex connectivity ($k(G)$) of a graph is the minimum number of vertices that need to be removed to separate it into two pieces. The edge connectivity ($\lambda(G)$) of a graph is the minimum number of edges that need to be removed to separate it into two pieces [16].

Fault tolerance analysis of onboard networks requires evaluation of vertex connectivity of the graph. This is due to failures of routers and nodes that can disable proper operation of the entire network. Moreover, there is a Whitney's inequality which relates edge connectivity, vertex-connectivity and minimum degree ($\delta(G)$) and states that for any graph G:

$$k(G) \leq \lambda(G) \leq \delta(G)$$

This means that vertex connectivity is less or equal to edge connectivity. Therefore, in this paper we will evaluate vertex connectivity of a graph. Consequently, in this paper we will consider failures of nodes and routers, which correspond to vertices in a graph. Failures of links are not considered separately, because node or router failure leads to all adjacent links disconnection.

So, in order to estimate fault tolerance of a designed network, it is necessary to build a graph on the basis of the network designed by user. While graph creation it is necessary to take into account that the network structure can be of arbitrary topology and may contain redundant nodes.

While evaluation of fault tolerance we should consider a general case and singular case. Singular case corresponds to point-to-point topologies and it shall be considered separately from the general algorithm. An example of point-to-point topology is given in Fig. 3.

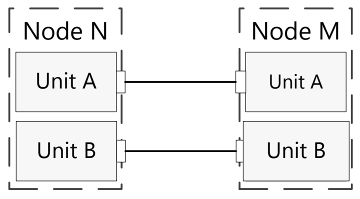

Fig. 3. Point-to-point topology

The system from the example above can tolerate 1 failure, i.e. it is 1-fault-tolerant ($f = 1$).

General case corresponds to a network topology with nodes and switches. The input network topology should be transformed into a graph. We have set up a rule for graph construction (see Fig. 4):

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

Input data for the algorithm are obtained from the intermediate file generated by the graphical user interface. This file can contain either full network structure or a part of the network topology which should be analyzed for fault-tolerance. In both cases the transformation rules are the same.

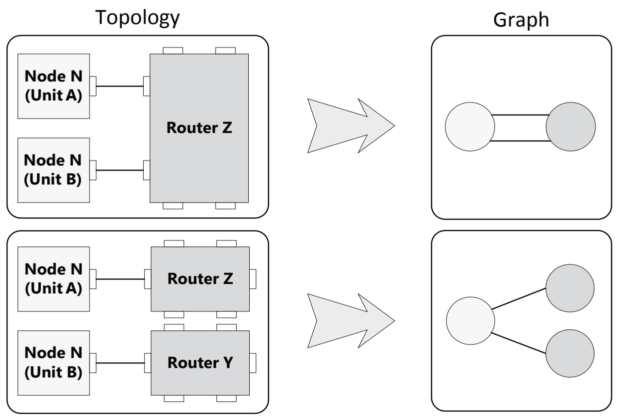

Fig. 4. Transformation of network topology to a graph

The connectivity analysis is performed on the basis of directed graphs. A directed graph, or digraph D, consists of a finite nonempty set V of points together with a prescribed collection X of ordered pairs of distinct points. The elements of X are directed lines or arcs which are called edges [17]. SpaceWire standard provides bidirectional links, so each link shall be represented by a pair of edges with opposite orientations (see Fig. 5). We use the digraph for the k-connectivity analysis because we should apply maxflow algorithms which work in directed graphs.

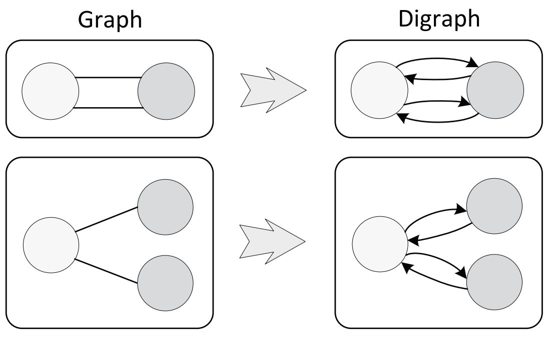

Fig. 5. A digraph of a network topology

B. Generalized algorithm of fault tolerance evaluation

Once the graph is created from the network topology we can start the general algorithm of fault-tolerance evaluation. The algorithm is divided into several steps which are described below in this subsection.

Step 1. Check the connectivity of a graph. If a graph is not connected, then there is no need to continue fault tolerance evaluation as it obviously does not tolerate any failure. Otherwise, if the graph is connected, move to Step 2.

Step 2. Evaluation of vertex connectivity of the graph. It is necessary to search through all pairs of graph vertices s and t, find the number of vertex disjoint paths from s to t and get the minimum which corresponds to search value of the graph connectivity k.

The problem of finding vertex connectivity reduces to a problem of finding edge connectivity. For each pair of vertices s and t we should create new graph G' where each vertex v from the graph is split into two vertices v_1 and v_2. All incoming edges to the original vertex go to v_1 while all outgoing edges come from v_2 (see Fig. 6). Moreover, we add an edge v_1-v_2 of the capacity equal to 1.

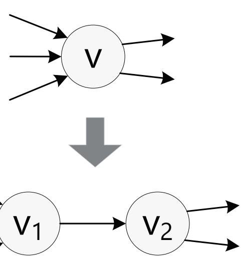

Fig. 6. Splitting of vertex

The algorithm of creation such graph G' for the pair of vertices s and t from the initial graph $G = (V, E)$ is given below. Input: G, s, t; output: G'.

Algorithm 1 Creation of graph with split vertices

```
0: $V' \leftarrow \{s, t\}$; $E' \leftarrow \emptyset$;
1: for $v \in V \setminus \{s, t\}$ do $V' \leftarrow V' \cup \{v_1, v_2\}$;
   $E' \leftarrow E' \cup \{v_1v_2\}$ od
2: for $e \in E$ do
3: if $e = sv$ with $v = t$ then $E' \leftarrow E' \cup \{sv_1\}$ fi
4: if $e = tv$ with $v = s$ then $E' \leftarrow E' \cup \{v_2t\}$ fi
5: if $e = uv$ with $u, v = s, t$ then $E' \leftarrow E' \cup \{u''v_1, v_2u'\}$ fi
6: od
7: $G' \leftarrow (V', E')$;
```

In theoretical computer science, graph connectivity has been well studied for more than forty years. It has a strong relationship with the problems of maximal network flow and minimal cut.

In order to find the number of disjoint paths from s to t we propose to use maxflow algorithms such as Ford-Fulkerson or Edmonds-Karp algorithms [18]. Let us assign each edge of the graph G' a capacity equal to 1. This allows to apply maxflow algorithm to evaluate st-connectivity.

The minimal maxflow among all s-t pairs of the initial graph G yields the graph vertex connectivity value k. In order to decrease the number of iterations in the algorithm we propose to terminate the search once the connectivity value k becomes equal to 1. This means that the graph cannot be more than 1-connected, i.e. it cannot tolerate any failure. The connectivity search algorithm stops its execution and moves to Step 3.

Step 3. Calculate fault tolerance of the graph using the following formula:

$$f = k - 1$$

The upper bound of complexity of this algorithm is $O(|V|^{1/2}|E|^2)$ [18].

C. Bottlenecks search in the designed network topology

If the value of fault tolerance is 0 then we propose to search all bridges and articulation points in a graph. This can give an opportunity for the CAD user to find out all bottlenecks of the designed network.

Let us start with a search of bridges. A bridge in a graph is an edge that, if removed, would separate a connected graph into two disjoint subgraphs. Fig. 7 shows a graph which contains two bridges – edges (1, 4) and (2, 5).

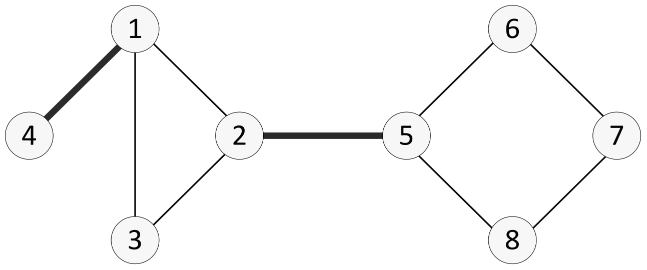

Fig. 7. An edge-separable graph

Finding the bridges in a graph seems, at first blush, to be nontrivial graph-processing problem, but it actually is an application of depth first search. Depth-first search (DFS) is an algorithm for traversing or searching tree or graph: to visit a vertex, we mark it as having been visited, then recursively visit all the vertices that are adjacent to it and that have not yet been marked [16]. It is one of the most important algorithms as it deceptively simple, it is easy to implement, takes linear time and in fact it is a powerful algorithm which is used for numerous difficult graph processing tasks.

On the basis of initial topology graph using the DFS algorithm we get an ordered tree. This tree contains all the vertices of the initial graph. Applying DFS to a graph we check both representations of each edge. For the edge *v*-*w* we either make a recursive call (if *w* is not marked) or just skip this edge (if *w* is already marked). When we come across this edge the second time but with the opposite direction, we should ignore it as the target vertex *v* has already been visited.

For the further discussion we need to introduce some notions referring trees, such as ancestor, descendant and parent. The parent of a vertex is the vertex connected to it on the path to the root; every vertex except the root has a unique parent. A child of a vertex *v* is a vertex of which *v* is the parent. A descendant of vertex *v* is any vertex *w* whose path from the root contains *v*. An ancestor is a vertex on the path from the root to the vertex. Vertex *v* is an ancestor of vertex *w* if and only if *w* is a descendant of *v*.

In order to find bridges it is necessary to find all vertices in the tree with the following property: no back edge connecting descendant with an ancestor, and no other nodes have that property. Therefore, breaking the edge between such node and its parent would disconnect the subtree rooted at that node from the rest of the graph.

Bridges in a graph correspond to links that are bottlenecks in the designed network. However, not only links can be bottlenecks in the network. A failure of router can lead to breaking a network into two separate subnetworks with no means to communicate with each other. Therefore, we will now discuss the problem of articulation points search.

An articulation point in a graph is a vertex that, if, removed, would separate a connected graph into at least two disjoint subgraphs. Fig. 8 shows a graph with two articulation points – vertices 1 and 2. A graph that has no articulation points is vertex connected [16].

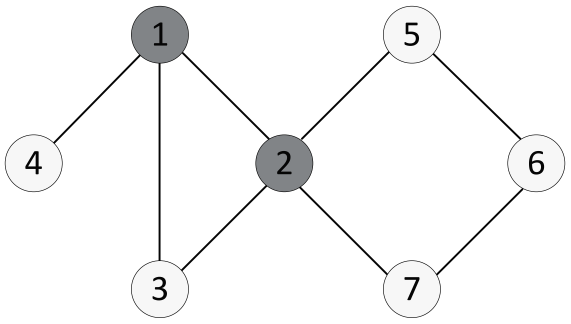

Fig. 8. Articulation points in a graph

Similarly to the search of bridges we use the same DFS-based approach to identify articulation points. Articulation point is a tree vertex *v* which satisfies the following conditions:

- *v* is a root of DFS tree and it has at least two children. This means that it is impossible to reach all vertices of a graph traversing a tree via an arbitrary edge from the root of DFS tree.
- *v* is not a root of DFS tree and it has a child u such that no vertex in subtree rooted in u has a back edge to one of the ancestors of v.

Articulation points can be found in linear time.

D. Fault tolerance evaluation examples

In the scope of our project we implemented the described above algorithm in C++ and performed several tests of its operation. In this subsection we give two examples with their results.

The first example is a network, which consists of four terminal nodes, each of which is represented by to redundant units, four switches and a number of communication links (see Fig. 9).

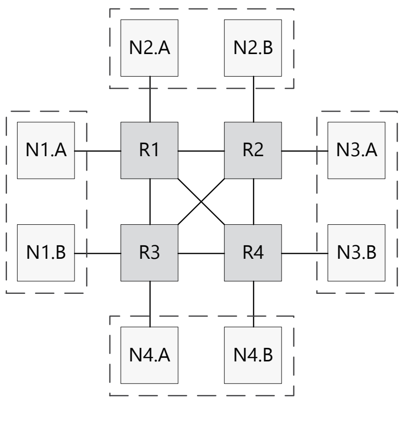

Fig. 9. Example 1

This network topology is 1-fault-tolerant (i.e. $f = 1$), as connectivity of corresponding graph is $k = 2$. There are no bridges and articulation points in this network structure.

The network topology for the second example is given in Fig. 10. It consists of four terminal nodes, five routers and a number of communication links. Similarly to the first example each node is represented by two units.

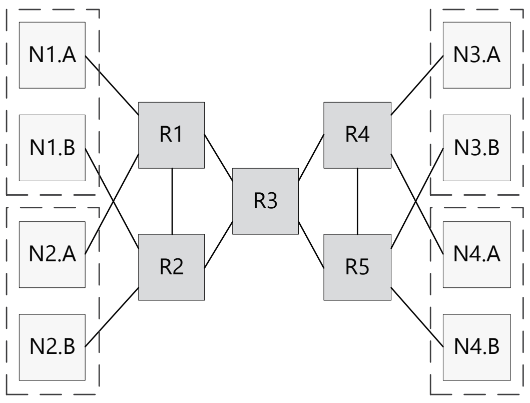

Fig. 10. Example 2

Running fault tolerance evaluation algorithm for this network structure results in zero fault tolerance ($f = 0$), as this network is 1-connected ($k = 1$). The algorithm identified one bottleneck – router *R3*, i.e. articulation point in the corresponding graph.

VI. NETWORK TOPOLOGY TRANSFORMATION FOR FAULT TOLERANCE

Another issue that we want to discuss 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 more big and 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 automatic network topology transformation to achieve a required fault tolerance. This is planned be an additional feature of the Component #1 in our CAD system.

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:

Stage 1: Initialise a topology of a network with required fault tolerance *f*. This can be done in the following ways:

- 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.
- connecting all routers in the initial network structure trough additional links.

If it is necessary, additional redundant units should be added to terminal nodes. Each unit of a terminal node should be connected to different routers in order to increase fault tolerance of a network.

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. 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. Stop when all added routers have been considered.

An example of network transformation for fault tolerance is given in Fig. 11.

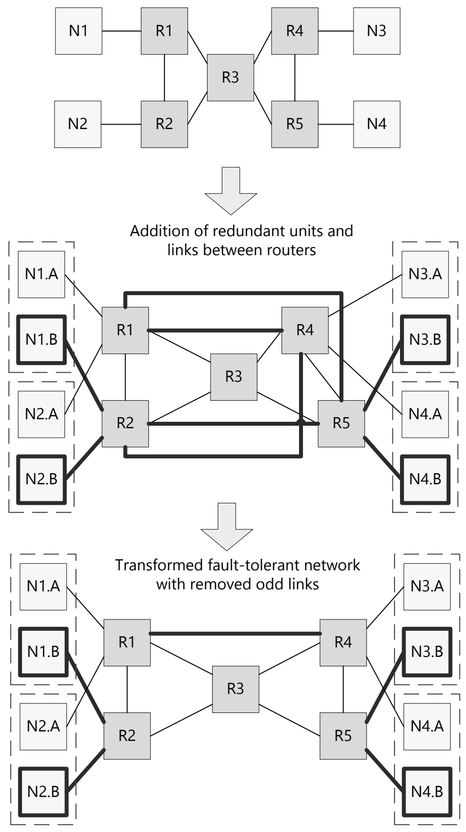
N1
R1
R4
N3
R3
N2
R2
R5
N4
Addition of redundant units and links between routers
N1.A
N1.B
N2.A
N2.B
N3.A
N3.B
N4.A
N4.B
Transformed fault-tolerant network with removed odd links

Fig. 11. Example of network transformation for fault tolerance

SpaceWire network can consist of a large number of terminal nodes and routers which are grouped into special network regions. The problem of network transformation for such a network is more complex than for a single region SpaceWire network. Firstly, we need to transform each network region to achieve the required fault-tolerance and then connect the regions in such a way that fault tolerance stays at the same level. This problem is not a trivial one, so it is a subject for discussion in our next paper.

VII. CONCLUSION

In the current paper we address the problem of fault tolerance analysis of SpaceWire onboard networks. Firstly, we introduced an architectural concept of a new computer aided design system for SpaceWire networks. This CAD system is intended to solve important tasks, which developers face with during implementation of onboard systems and networks. A component of onboard network topology design and evaluation of its structural characteristics is a part of this CAD. One of its key features is fault-tolerance evaluation of the designed network.

Our contribution is a proposal of an algorithm for fault tolerance analysis and bottlenecks search. Fault tolerance analysis is performed through *k*-connectivity search in a graph corresponding to a designed network topology.

There are still some open problems to be considered in future work. Our future steps will be focused on performance analysis of our algorithm. Finally, we are planning to extend Component #1 with an additional feature of network topology transformation for achieving the required fault tolerance.

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

[2] 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.

[3] 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.

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

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

[6] 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.

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

[8] 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.

[9] 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.

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

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

[12] Elena Suvorova, Yuriy Sheynin, Nadezhda Matveeva, "Reconfigurable NoC development with fault mitigation", *in Proceedings of the 18th FRUCT & ISPIT Conference*, Technopark of ITMO University, Saint-Petersburg, Russia, 2016, pp. 335-344.

[13] Rui Borralho, Pedro Fontes, Ana Antunes, Fernando Morgado Dias, "A Matlab Tool for Analyzing and Improving Fault Tolerance of Artificial Neural Networks", *in IFAC Proceedings Volumes*, Volume 42, Issue 19, 2009, Pages 158-163.

[14] 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.

[15] Alejandro Cornejo, Nancy Lynch, "Fault-Tolerance Through k-Connectivity", Workshop on Network Science and Systems Issues in Multi-Robot Autonomy: ICRA, 2010.

[16] Robert Sedgewick, "Algorithms in C++. Part 5 Graph Algorithms", Third edition, Addison-Wesley, USA, 2002, 511 p.

[17] Frank Harari, "Graph Theory", Addison-Wesley Publishing Company, 1969, 274 p.

[18] Dieter Jungnickel, "Graphs, Networks and Algorithms", Third Edition, Springer-Verlag Berlin Heidelberg 2008, 671 p.