Master’s Thesis

Distributed task Mapping in Reconfigurable Networked Embedded Systems

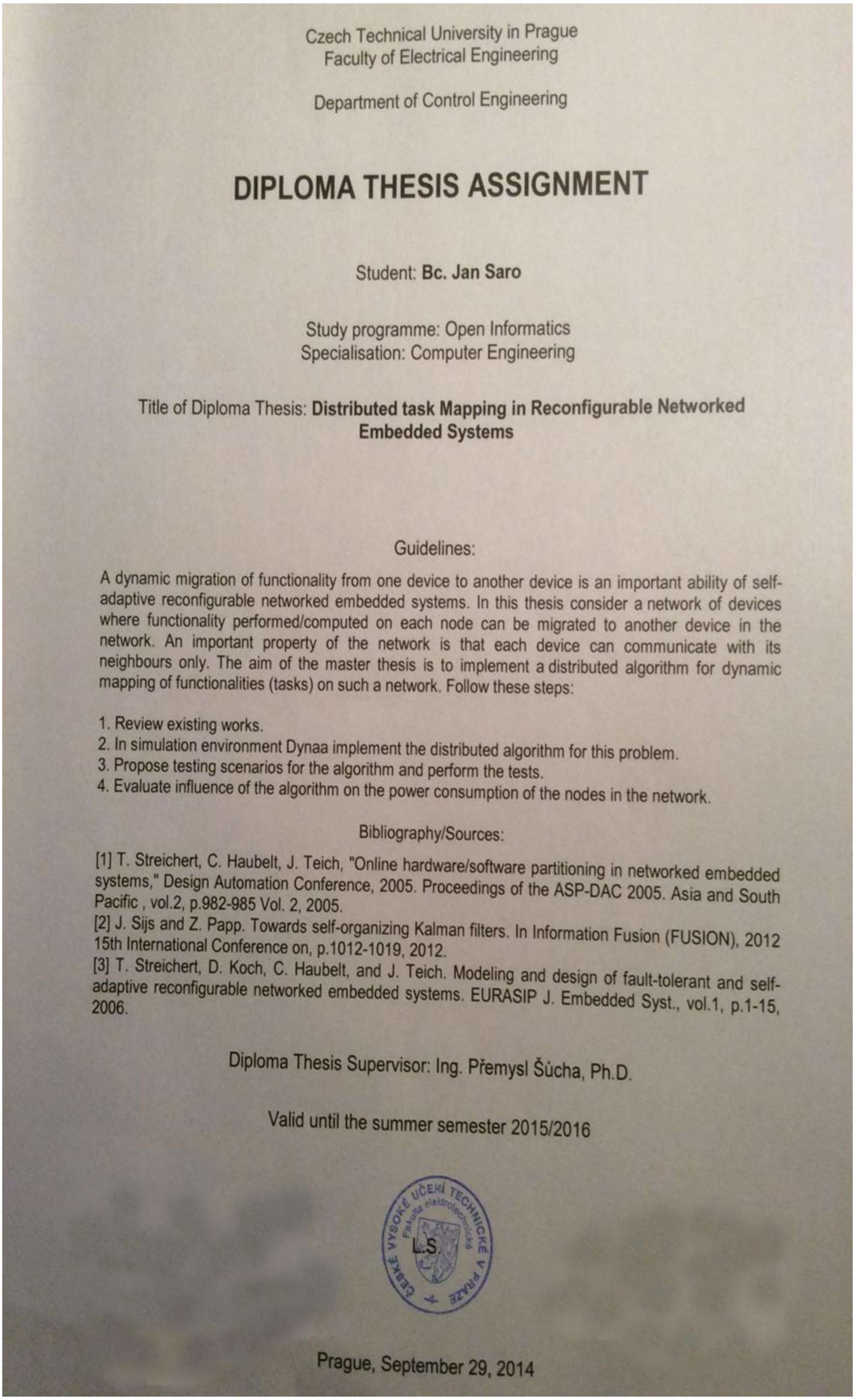
Bc. Jan Saro

May 7, 2015

Supervisor: Ing. Přemysl Šůcha, Ph.D.

Czech Technical University in Prague Faculty of Electrical Engineering

Department of Control Engineering



Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Control Engineering

DIPLOMA THESIS ASSIGNMENT

Student: **Bc. Jan Saro**

Study programme: Open Informatics
Specialisation: Computer Engineering

Title of Diploma Thesis: **Distributed task Mapping in Reconfigurable Networked Embedded Systems**

Guidelines:

A dynamic migration of functionality from one device to another device is an important ability of self-adaptive reconfigurable networked embedded systems. In this thesis consider a network of devices where functionality performed/computed on each node can be migrated to another device in the network. An important property of the network is that each device can communicate with its neighbours only. The aim of the master thesis is to implement a distributed algorithm for dynamic mapping of functionalities (tasks) on such a network. Follow these steps:

1. Review existing works.
2. In simulation environment Dynaa implement the distributed algorithm for this problem.
3. Propose testing scenarios for the algorithm and perform the tests.
4. Evaluate influence of the algorithm on the power consumption of the nodes in the network.

Bibliography/Sources:

[1] T. Streichert, C. Haubelt, J. Teich, "Online hardware/software partitioning in networked embedded systems," Design Automation Conference, 2005. Proceedings of the ASP-DAC 2005. Asia and South Pacific , vol.2, p.982-985 Vol. 2, 2005.
[2] J. Sijs and Z. Papp. Towards self-organizing Kalman filters. In Information Fusion (FUSION), 2012 15th International Conference on, p.1012-1019, 2012.
[3] T. Streichert, D. Koch, C. Haubelt, and J. Teich. Modeling and design of fault-tolerant and self-adaptive reconfigurable networked embedded systems. EURASIP J. Embedded Syst., vol.1, p.1-15, 2006.

Diploma Thesis Supervisor: Ing. Přemysl Šůcha, Ph.D.

Valid until the summer semester 2015/2016

[redacted]

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
L.S.

[redacted]

Prague, September 29, 2014

Declaration

I hereby declare that I have completed this thesis independently and that I have listed all the literature and publication used.

I have no objection to usage of this work in compliance with the act §60 Zákon 4. 121/2000Sb. (copyright law), and with the rights connected with the copyright act including the changes in the act.

In Prague on May 7, 2015

Acknowledgement

I would like to thank to my supervisor Ing. Přemysl Šůcha, Ph.D. for his professional leading of the thesis and helpful consultations. Further, I would also like to thank to my family, namely my parents, Petr a Zdeňka Saro, for invaluable moral and material support during my studies.

Abstrakt

V současné době schopnost měnit a migrovat funkcionalitu na samo-adaptivních a rekonfigurovatelných embedded zařízeních v bezdrátových senzorových sítích, se stává běžnou vlastností. Vedle této schopnosti zařízení jsou schopna počítat/vykonávat tyto úlohy reprezentující dané funkcionality a také komunikovat bezdrátově s ostatními zařízeními v síti. Každé samostatné zařízení je napájeno baterií a musí průběžně sledovat poměr množství zatížení reprezentované mapovanými úlohami na daném zařízení a kapacitou napájecího zdroje. Pokud se na jakémkoliv zařízení poměr zátěže a napájecí zdroje sníží pod určitou úroveň, zařízení vyvolá chybový stav v síti. Cílem této práce je navrhnout distribuovaný algoritmus, který je schopen opravit tento chybový stav změnou mapování úloh na zařízeních v síti.

Abstract

Nowadays, the ability to change and migrate functionality on self-adaptive and reconfigurable embedded devices in the wireless sensor networks becomes a common property. Beside that ability, devices are able to compute/perform tasks representing functionalities and also communicate wirelessly with other devices in the network. Each standalone device is power supplied by a battery and has to regularly monitor a ratio between amount of load represented by mapped tasks on the device and capacity of the power resource. When on any device this ratio decreases under an intended level, the device invokes a failure state in the network. The aim of that thesis is to propose the distributed algorithm, which is able to repaired that fault state by changing the task mapping on devices in the network.

Content

List of Figures

List of Tables

1. Motivation

Dynamic migration of functionality from one device to another device is an important ability of self-adaptive reconfigurable networked embedded systems. There are two reasons why dynamic functionality migration is considered in networked embedded systems. The first one is adaptability of these systems, e.g. when a device is added/removed into/from the system all functionalities are preserved. The second one is efficiency of the system expressed via an objective function(s) aimed at, e.g. energy consumption minimization or reliability maximization. Functionality in the systems is described via a set of tasks and functionality migration is realized by dynamic mapping and scheduling of these tasks with respect to actual state of the system. Mapping in this context means assignment of tasks to devices.

Figure 1: Temperature measurement in a greenhouse using reconfigurable networks

Our application considers a network of ultra low power wireless embedded devices in a greenhouse. Each device executes tasks specially focused on precise calculations of temperature and the other assignments like measuring and controlling. These devices in the network have ability to execute tasks and communicate wirelessly with its neighbours. The structure of our embedded device contains two core processors. First one, concretely in our case was chosen processor MSP430F2418 from company Texas Instruments, which is designated for executing tasks mapped on the device. The second one is CC1101, with a role of a transceiver, it also made by company Texas Instruments, it services assignments are related with wireless communication like broadcast transmitting and receiving messages from its neighbour in the network. You can see the physical appearance of the embedded in Figure 2.

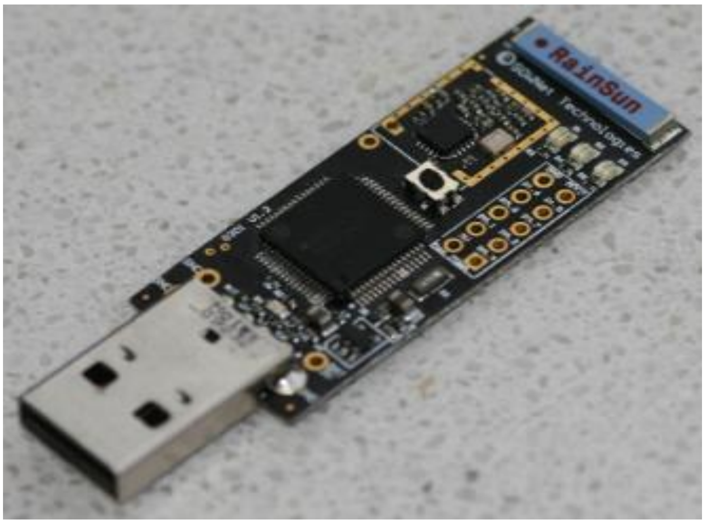


Figure 2: Embedded device representing a physical node in the network

When we look closely at one of the essential property of distributed system - behind the intention of its creation. In the area of the greenhouse it is required to calculate very precisely the temperature in the space. It means for every device which has to measure the temperature. However, there is formatted request for calculating temperature in space among devices. So consequently a certain subset of devices has to calculate from received temperatures of its neighbours an approximate value of temperature in the interspace. In Figure 3 illustrates the network of devices with mesh structure in 3D space. Nodes are formed into two layers, where in the upper layer there are placed green coloured nodes whereas in the lower nodes with red colour. Black lines illustrate bidirectional communication links among devices. Finally, the yellow points show the measured and calculated values of temperatures.

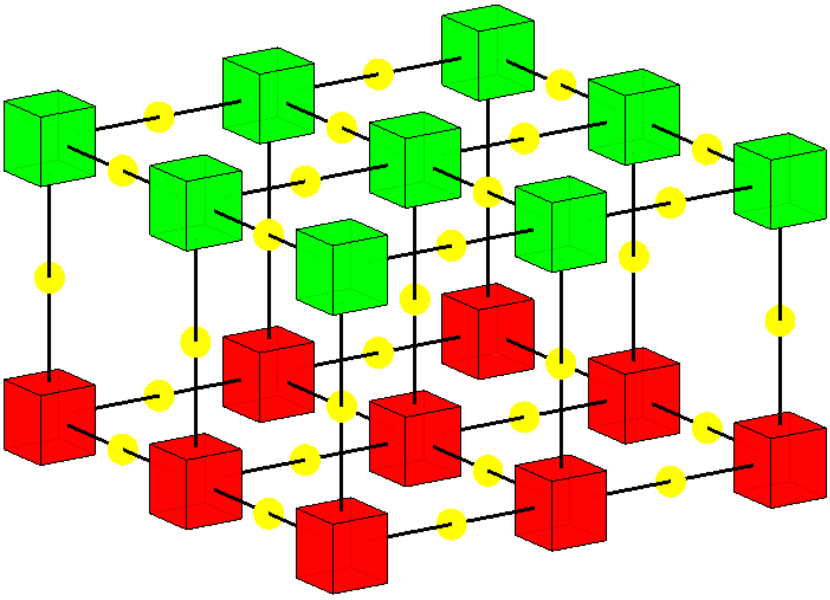


Figure 3: Mesh of embedded devices in 3D space

In practise, there is estimated and monitored finer resolution of the space. In Figure 3 – there weren't shown all points from readability reasons. However, there are at least estimated points in the centre of each four or eight devices. Computing the temperature is based on using algorithm including Kalman filtering.

Next important feature of embedded device is the fact, that its power supply is a battery. For a network lifetime it is necessary to coordinate an adequate amount of load, represented by mapped tasks on the device, with respect to the current battery capacity. In this work, we express this aspect via an objective function. The objective function has responsibility for checking, on each device ratio between current battery capacity and the appropriate workload on the device, consisted of all tasks bound to it. When any device in the network, i.e. using a watchdog checking mechanism, finds out that its objective function does not meet the given conditions, then it has to invoke an alert state. As recommended reparation, the failure node has to launch a rescheduling algorithm, which wakes up a set of its neighbours and triggers on them distributed task migration algorithm in order to repair a source problem node with an unfulfilled objective function.

For getting a better view on the investigated problem, it is illustrated on an example in Figure 4. Assuming that the network operates in the operational state and all embedded devices execute tasks mapped on them. At once a watchdog on Node 5 detects, that it does not fulfil the condition of the objective function. Consequently Node 5 has to start distributed reparation of mechanism that will fulfil the requirements of the objective function. As it can be seen on the left side in Figure 4, the result of the reparation revealed the problem. Task 6 migrated to Node 4 and task 7 migrated to Node 6. After execution of the algorithm, the Node 5 fulfils the condition of the objective function. Consequently, all involved devices, taken part in task migration, return to the operational state and starts up calculation of the tasks bound to them.

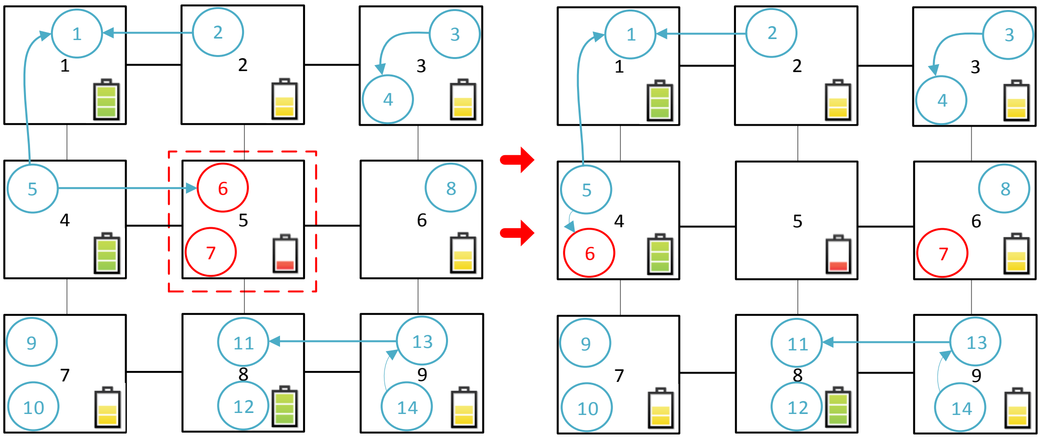


Figure 4: Reparation of failure state by task remapping

The aim of this thesis is to propose and test a distributed algorithm, which solves the mentioned problem by migrating and remapping tasks on devices with output of load balanced adequately to the amount of power supply from battery. Due to energy consumption, caused by executing tasks and transmitting messages to neighbours devices, it is required to propose such algorithm, which uses minimal communication between devices.

2. Related works

Firstly, we will look at a general overview of works interested on field of task mapping problem. Afterwards, we will concentrate on partial articles with the most similar solved problem. In these parts we will explain in detail their manner of solving problem and emphasise the differences among them and the measures.

The problem of task mapping is mainly addressed in parallel and grid computing area, e.g.: **[KA99]**. Much less attention is paid to this problem in networked embedded systems. According to the quality indicator of task mapping expressed by the objective function(s) there are three possible approaches to task mapping and scheduling **[GCL02]**:

1. Global approach - There is only one decision maker (task mapper and scheduler) having a single objective function.
2. Cooperative approach - There are several decision makers (e.g. devices) that cooperate in making the decisions.
3. Non-cooperative approach - There are several decision makers, but each decision maker optimizes its own objective.

Majority of existing, works dealing with on-line mapping and tasks scheduling, consider global approach. Authors in **[ASEP11]** deal with task mapping and scheduling problem on networked embedded systems. Moreover, they also involved a control synthesis into the design process. The problem is solved off-line by a genetic algorithm. An ILP (Integer Linear Programming) problem formulation is proposed in **[YWXEA09]**. This algorithm considers non-preemptive tasks and pipelining. Moreover, in order to exploit parallelism as much as possible, the replication of tasks is allowed.

Even less works are dealing with distributed algorithms for tasks scheduling and mapping. A self-organizing sensor network is described in **[SP12]**. The authors show a case study illustrating Kalman filtering on a distributed network of embedded systems. A distributed algorithm for on-line task mapping on reconfigurable networked embedded systems is described in **[KA99]**. The algorithm is based on diffusion algorithm, first introduced by Cybenko **[Cyb89]**. The disadvantage of the algorithm **[KA99]** is that it could generate huge communication traffic in the network. A diffusion algorithm is also used in **[N12]**. Their algorithm is applied to a homogeneous system, i.e. system where capabilities of all devices are equal. Unlike the algorithm in **[KA99]** they consider integer granularity of tasks.

For the last non-cooperative approach, authors **[GCL02]** created an algorithm using the Divisible Load Scheduling (DLS) theory, where it is observed a scheduling of divisible load in distributed systems. The designed distributed mechanism is dedicated for the tree network of processors. Processors provide incentives to their true capacities and executing their assigned load at full processing capacity. However there is assumed, that processors can cheat, because they are autonomous. Finally, in the paper it is proved that DLS-T mechanism computes the optimal allocation in an ex post Nash equilibrium.

We want to draw a special attention to the most similar article connected with our topic from authors **[SKHT06]**. They consider FPGAs in combination with CPUs and allow migrating tasks implemented in HW or SW from one node to the other one, where each node represents some embedded device. They use a local iterative load balancing algorithm from **[Cyb89]** performed iterations on all nodes, determining a load exchange between the adjacent nodes. However, the designed algorithm also allows migrating of non-discrete values of loads and that property is not applicable in our observed problem. That property is solved by the same authors in the next their article in **[SHT05]**. In this paper, there is proposed a modification of this diffusion algorithm, that leads to its discrete version, which overcomes the following problems:

1. One task cannot be split and distributed to multiple nodes
2. It cannot occur that negative loads are assigned to computational nodes.

Furthermore, the designed discrete version is able to balance load in constant number of steps. Nevertheless, for the dedicated number of steps it is needed to introduce the Laplacian matrix of the network. That last fact implies the disadvantage of that algorithm for our purposes. For calculation a Laplacian matrix it is needed to use the information about adjacency matrix of the whole network and for our algorithm there is a defined request on the using the distributive. Our second constrain, which is not also solved in the article **[SHT05]** is that two tasks connected via edge can be mapped on the same node or on two neighboring nodes.

The contribution of this thesis is based on the following aspects. We will design a distributed algorithm in which every device work without knowledge of the whole network structure, that is used in **[SHT05]**. The algorithm works by a dynamic manner it means that reacts on changes on the partial devices in the network. Next benefit is that the designed algorithm is usable on all network topologies in comparison with **[CG12]**. An algorithm especially depends on the minimum use of communication. It implies that algorithm saves energy on all devices involved in the algorithm.

3. Problem statement

In this chapter we want to declare mathematical model and notation of the solved problem.

3.1. Notation of the problem statement

This section formally introduces the problem of tasks mapping in networked embedded systems supporting dynamic reconfiguration. The network is formed by nodes connected via bidirectional communication links. The considered remapping problem is defined by a tuple $\langle G^T, G^A, P, M, cost\rangle$, where:

- $G^T(N^T, E^T)$ refers to a *task graph* having a set of particular tasks $i \in N^T = \{1, .., n\}$ and their data dependencies expressed by oriented edge $E^T = \{(i,j): i,j \in N^T\}$. Moreover, each task $i \in N^T$ is associated with power energy consumption per one iteration $r_i \in \mathbb{R}^+$. Executing task graph in the network is repeated in iterations. Each edge $(i,j) \in E^T$ is parameterized by demanded for transmitting energy $c_i \in \mathbb{R}^+$, needed when the output of task i is sent to task j mapped on a different node.
- $G^A(N^A, E^A)$ is an *architecture graph* reflecting reflecting the structure of the self-organizing network. Nodes $k \in N^A = \{1, .., m\}$ in the network are connected via bidirectional communication links $E^A = \left\{\{k,l\}: k,l \in N^A\right\}$. Each node $k \in N^A$ is parameterized by its size of energy capacity (resource capacity) $R_k \in \mathbb{R}^+$.
- The permitted mapping is a function $P: N^T \times N^A \to \{0,1\}$, where $P(i,j) = 1$ means that task i can be assigned on node j and 0 otherwise.
- The *task mapping* $M: N^T \to N^A$ is an assignment of tasks $i \in N^T$ to nodes $k \in N^A$, i.e. $M(i) = k$ means that task $i \in N^T$ is mapped on node $k \in N^A$. Mapping M is feasible if and only if:
  1. $\forall (i,j) \in E^T$ there is node $k \in N^A$ such that $M(i) = M(j) = k$ or there is node $l \in N^A$ that is connected via bidirectional link $(k,l) \in E^A$ with the node $k \in N^A$, such that task $M(i) = k \wedge M(j) = l$
  2. $\forall i \in N^T, k \in N^A: P(i,k) = 1$

  The first constraint expresses that two tasks can be mapped on the same node $k \in N^A$ or on two different nodes connected via a communication link in an architecture graph. The second constrain says that the given task can be assigned only on a certain subset of nodes defined by the permitted mapping. $M(i) = k$ means that task $i \in N^T$ is mapped on node $k \in N^A$.

- The *cost of the mapping*, denoted as $cost: M \rightarrow \mathbb{R}^+$, is a function defining energy consumption of mapping per one iteration. We consider the cost of mapping given by $cost(i,k)$ defining how expensive is to map task $i \in N^T$on the node $k \in N^A$. The cost of mapping is defined by

$$cost(i,k) = \begin{cases} r_i + c_i, & S \neq \emptyset \\ r_i, & S = \emptyset \end{cases}$$

$$, where\ S = \{\forall (i,j) \in E^T : \forall k \in N^A, \forall l \in N^A\ (i,k) \in M \wedge (j,l) \in M \wedge l \neq k\}$$

Then the objective in this problem is to maximize the number of possible iterations, i.e. repetitions of the task graph. It is denoted as *repetition objective*, singed as $rep: N^A \rightarrow \mathbb{R}^+$, which is a function defining a ratio between resource capacity R_k and sum of all costs of tasks mapped on the node $k \in N^A$. The function is expressed in Equation (1):

$$rep_k = R_k / \sum_{i \in N^T:(i,k) \in M} cost(i,k) \qquad (1)$$

,where the sum in the denominator defines, how expensive is mapping of all tasks mapped on node $k \in N^A$.

Aim of the whole problem statement is to maximize the minimum value of repetition objective in the whole network in Equation (2).

$$\max \{\min_{\forall k \in N^A} \{rep_k\}\} \qquad (2)$$

3.2. Illustrative example of the problem statement

This section aims to illustrate formulations from the previous section. An example of the network is in Figure 5. Rectangle boxes represent a set of nodes N^A with resource capacities R_i of each node i, that are connected via blue marked communication links represented by edges from set E^A. Secondly, the blue circles are tasks N^T with task requirements r_i, which have data dependencies represented via edges E^T with prices c_i. Figure 5 also illustrates a task mapping e.g. task 1 is mapped on node 1, task 2 is mapped on node 2, tasks 3 and 5 on node 3 and finally tasks 4, 5 and 6 on node 4. Finally Figure 5 also shows calculation of rep_3 objective function on node 3.

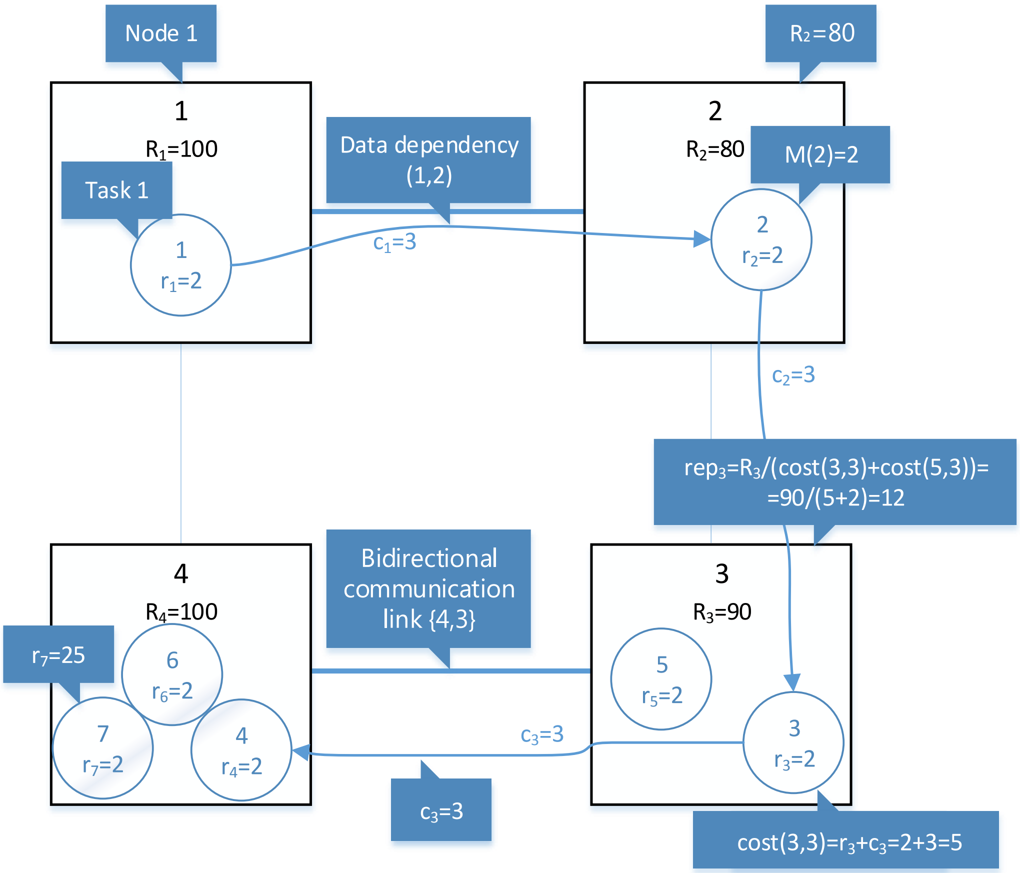


Figure 5: An example of the network desbribed by formulations

4. ILP model of task mapping

The problem of task mapping introduced in the previous chapter is NP-hard, how it was shown in **[SFHP13]**, a global (centralized) solution, proposed in this section, is based on integer linear programming. It uses a binary decision variable $x_{i,k}$ equal to 1 if and only if task $i \in N^T$ is mapped on node $k \in N^A$ and equal to 0 otherwise. We define auxiliary decision variable $y_{i,k}$ equal to 1 if and only if there is edge $(i,j) \in E^T$ such that $M(i) = k$ and $M(j) = l$ where $l \in N^A$. The integer linear programming model is expressed:

objective function: minimize A

subject to:

$$B_k \leq A, \qquad \forall k \in N^A/n \qquad (3)$$

$$\sum_{\forall i \in N^T} x_{i,k} r_i + \sum_{\forall i \in N^T} y_{i,k} c_i \leq B_k R_k, \qquad \forall k \in N^A \qquad (4)$$

$$\sum_{\forall k \in N^A} x_{i,k} = 1, \qquad \forall i \in N^T \qquad (5)$$

$$x_{i,k} = 0,\ P(i,k) = 0\text{:} \qquad \forall i \in N^T, \forall k \in N^A \qquad (6)$$

$$x_{j,k} + \sum_{\forall l \in \partial(k)} x_{j,l} \geq x_{i,k}, \qquad \forall k \in N^A, \forall i \in N^T, \forall (i,j) \in E^T \ (7)$$

$$y_{i,k} \geq x_{i,k} - x_{j,k}, \qquad \forall k \in N^A, \forall i \in N^T, \forall (i,j) \in E^T \ (8)$$

where: $A \in \mathbb{R}_{\geq 0},\ B_{k \in N^A} \in \mathbb{R}_{\geq 0},\ x_{i,k} \in \{0,1\},\ y_{i,k} \in \{0,1\}$

Aim of the model is to minimize objective function $= \frac{1}{\max_{\forall k \in N^A}\{rep_k\}}$. Objective function A is not linear, thus we have to make a substitution $B_k = \frac{\sum_{i \in N^T:(i,k) \in M} cost(i,k)}{R_k}$. The threshold value A defines the upper bound for all elements of the vector $B_{k \in N^A} \in \mathbb{R}_0^+$ in Equation (3). Equation (4) expresses a constrain, in which due to minimized value of $B_{k \in N^A}$ load on node k has to smaller or equal to $B_k R_k$. Equation (5) denotes that exactly one task $i \in N^T$ can be mapped on one node $k \in N^A$ only. Equation (6) expresses permitted mapping for every task $i \in N^T$and node $k \in N^A$. If the Permitted mapping $P(i,k) = 0$ then the value $x_{i,k}$ must be also zero and task i can't be mapped on node k. Equation (7) describes the constrain, when task $i \in N^T$ has data dependency with task $j \in N^T$ has to be mapped on the same node $k \in N^A$, on which is mapped task $i \in N^T$ or on neighbors of node $k \in N^A$. The set of neighbors of the node $k \in N^A$ is the set of all nodes $\forall l \in \partial(k)$, which are connected with an edge $(k,l) \in E^A$. The last Equation (8) declares a constrain by using additional decision variable $y_{i,k}$. The constrain ensures, when the task $i \in N^T$is mapped on node $k \in N^A$ and it is has also a data dependency $(i, j) \in E^T$ with a task $j \in N^T$ that is mapped on the another node than $k \in N^A$, then it has to be value of $y_{i,k} = 1$. The consequent of the last constrain is connected with Equation (4) in which is encountered the $cost_i$ expressing energy consumption for sending data from one task mapped on one node $k \in N^A$ to another node $l \in N^A$.

5. Description of the network properties

In this chapter we aim to classify properties and capabilities of the network. Among these properties, we can include a description of individual devices, communication and synchronization of devices in the network and so on. We want to introduce aspects of the network, which has to be necessary taken into account during the design of the distributed algorithm.

5.1. Device characteristics

Due to the fact that our system is distributed, therefore each device behaves like standalone unit. Each device has an identification number that responds to node number k from the set N^A. We define the set of the *known* and *unknown attributes*. Consequently every node $k \in N^A$ knows the following list of attributes:

1. task graph G^T
2. permitted mapping P
3. task mapping $M(i) = k, i \in N^T$
4. repetition objective rep_k

However for each node $k \in N^A$, is declared the complement of set *known attributes* called *unknown attributes*, which has to following content:

1. architecture graph G^A
2. mapping $M(i) \neq k, i \in N^T$
3. repetition objective $rep_l, l \in N^A, l \neq k$

The only way how node k can obtain *unknown attributes* is to communicate with the neighbors.

5.2. Communication in the network

Each device has ability to communicate wirelessly. It means that the node in the network transmits messages wirelessly, which can be received by all near devices in the network i.e. connected to the node. When the node decides to transmit some data to the network, it always packs them into a single message and transmits it. The message is a communication unit used for communication and distribution any information among the nodes. Therefore we have to make a standard structure of the content of message. The standard pattern of the content of the message states the following list of elements:

1. name
2. origin node identification
3. init time stamp
4. time duration

5. distance
6. path
7. optional attributes

These seven elements have to be contained inside every message, when it is initiated by any node in the network. The first element defines the purpose of the message. According to this attribute each node has ability to determine the content of the seventh element. The third element named time stamp means the time, when the message was sent. Next, the fourth element, called time duration is used for time synchronization of nodes. The time synchronization will be discussed in the next subchapter. The fifth element called distance, defines the maximal count of hops up to the message has to be sent. When the message crosses an edge from the set E^A, it makes one hop. The penultimate element introduces the list of all node identifications that retransmitted the message. The path includes also an identification number of the origin node. The last element called optional attributes, contains the message specific data, e.g. current mapping of tasks on a different node.

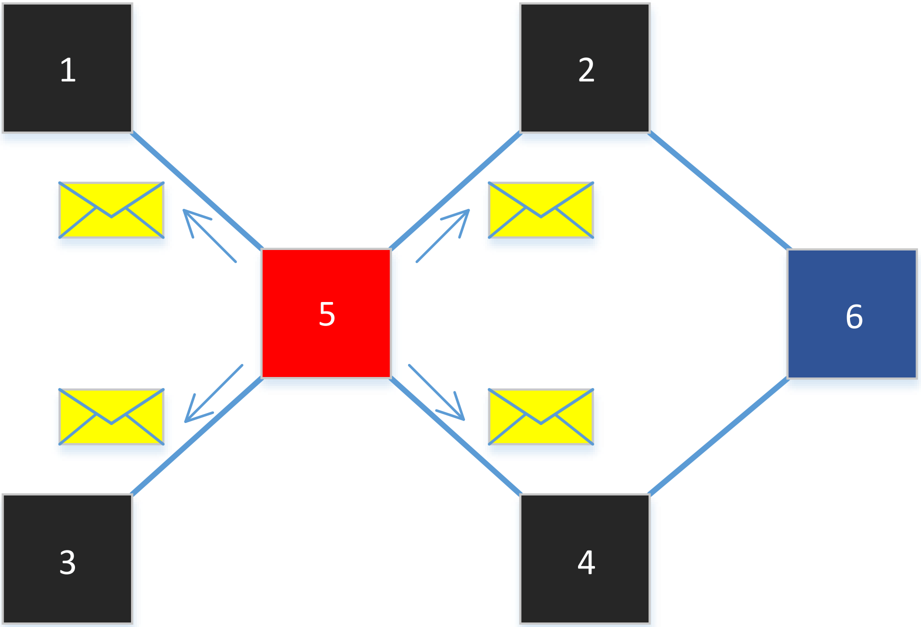


Figure 6: A demonstration of transmitting message from node 5

Transmission of a message from node 5 is illustrated in Figure 6. The red coloured node 5 creates the message and sends it to all its neighbours. Under the term neighbours, we mean all nodes adjacent with node 5 in E^A. In the other words, all black coloured nodes 1,2,3 and 4 will receive at one time moment the message initiated by node 5. We can also notice that node 6 does not receive the message. It is caused by the fact that node 6 is distanced more than one from transmitting node 5. So the node 6 is not directly connected via an edge from node 5. The node 6 can receive the message, when node 2 or 3 transmits the message. In that case the message would be received also node by 5 apart from node 6.

In the next three subchapters we want to introduce three communication schemas. These schemas are used in the proposed algorithm.

5.2.1. One to All Communication

The aim of this technique of communication is to distribute a message from one node to the subset of nodes. Firstly one node in the subset creates a message and transmits it. Consequently all nodes in its neighbourhood will receive that message and if the message has the nonzero count of hops then it is retransmitted further. So the source node defines the size of the informed area of nodes by count of hops. In Figure 7 we can see that node 3 initiates and transmits the message. Then node 2 and 3 receive a message and resend it further. Finally, node 1 and 5 receive the message. We can also observe that the size of set informed nodes has diameter equal to 2 from the source node.

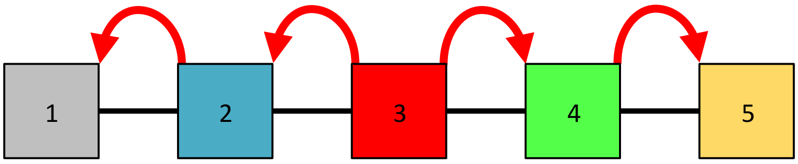


Figure 7: Demonstration of one to all communication

The common way of using this communication scheme is for informing about some event, which happened in the source node.

5.2.2. All to All Communication

Each node in this subset has to initiate and transmit the message. And also all nodes set at the same count of hops in a message. It means that all nodes in the subset will transmit the message to the same distance. After initiating each node only receives, stores and retransmits the message, only if the message has the nonzero count of hops, representing a subset of nodes in the network. Nevertheless, this communication manner is the most demanding for energy. That scheme also describes the non- centralized way of communication.

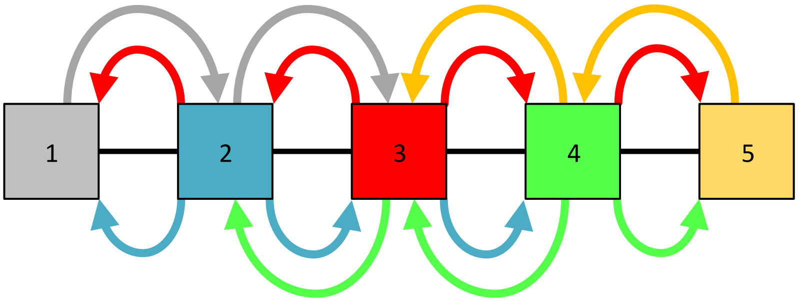


Figure 8: Demonstration of all to all communication

5.2.3. All to one Communication

The last communication technique called All-to-One, describing a way, where all nodes from the subset of the network transmit a message to the destination node. It means that every node in the subset initiates a message apart from destination node and transmits it. When the destination node receives a message then stores its content and does not transmit it next to the network. Usual purpose of this technique is an announcement from all nodes from the set to the one destination node. We can view on this technique like on inverses communication scheme from subchapter 6.2.1. In Figure 9, we can notice that in the subset nodes 1,2,4 and 5 transmit the message to node 3, which is the destination node. This communication is used when a group of nodes wants to send some result to the specific node.

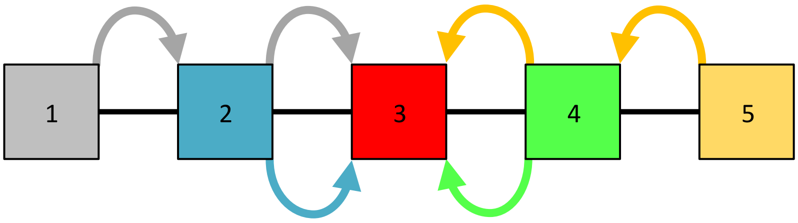


Figure 9: Demonstration of all to one communication

5.3. Time synchronization

The last important property in the distributed network is the time synchronization. The time synchronization is one of the most important aspects for determinism in our distributed system. It ensures that all nodes in the network will work within the same time interval. We will call the time interval as a state or a stage.

In the network in case of the device failures, the assignment is to inform and involve other devices into solution of the problem. The failure node creates the message and stores current time t time into message attribute init time stamp t_i. Time length of stage or state is stored into time duration t_d attribute of message. Alert is stored into message attribute name. When the failure node finishes the creation of the message, then it transmits the message to the network. If any node in the network receives that message with name attribute Alert, it stores its time init stamp and time duration attributes and if the message attribute counts of hop is not zero, the node retransmits it to the network. However, an assignment of each node, which received the message or failure, is to check timeout of state. Timeout of state occurs for every node, which does not fulfil the condition in Equation (9).

$$t_i + t_d > t \qquad (9)$$

After elapsing timeout node transits into next state, which is defined after Alert state.

We would like to demonstrate example of the time synchronization in Figure 12. The network has chain topology consisted of four nodes. The horizontal axes represent time. Each green line represents time flow of each node. In time line, time events, which are represented by colour points can occur. Time events are described in text area showing on these points. Firstly, node 2 initiates a message in time equal with t_i=0, t_d=6 that action is represented in 1.Event. In time t=2 nodes 1 and 3 receive a message transmitted by node 2. That action are represented by 2. and 3.event. This pair of nodes stores the value init time and time duration attributes. Consequently, node 4 receives the message from node 3 in time t=4 and that is represented by 4.event in picture. Finally all nodes simultaneously reached timeout in t=6 that is expressed in 5.event. We can see that all nodes are synchronized and simultaneously transit in time t=6 into next state.

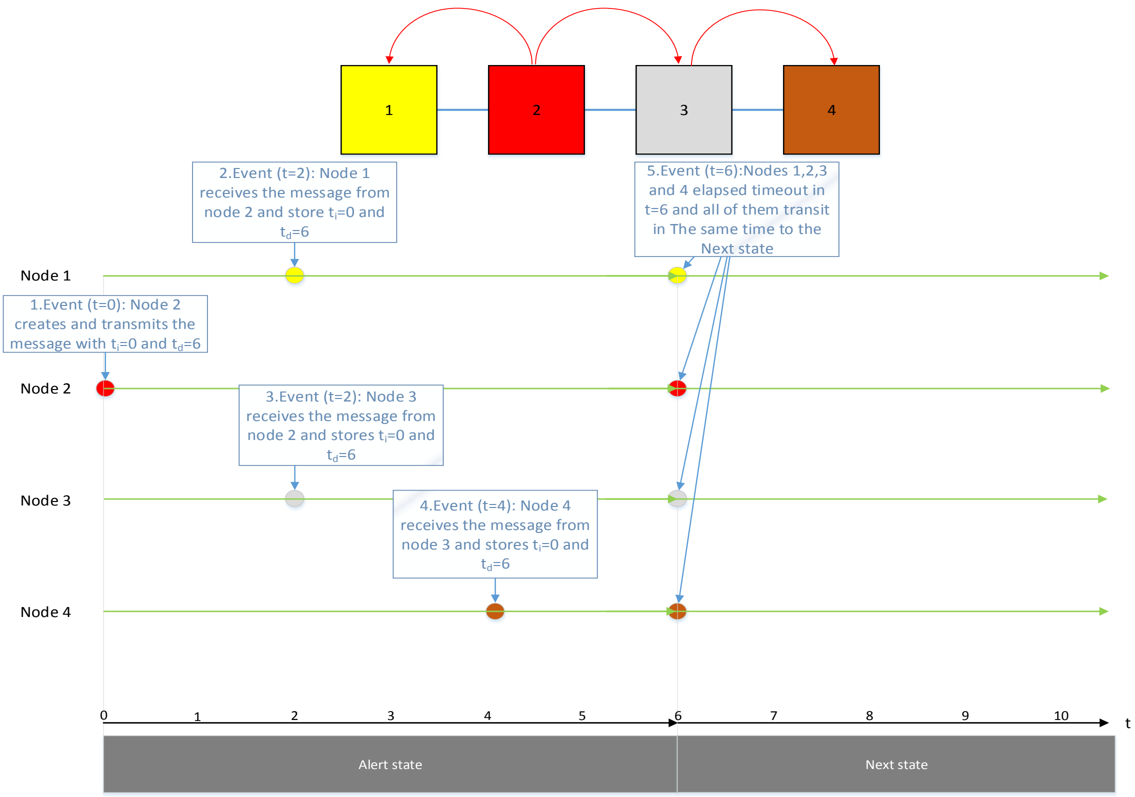

1
2
3
4
2.Event (t=2): Node 1 receives the message from node 2 and store t_i=0 and t_d=6
5.Event (t=6):Nodes 1,2,3 and 4 elapsed timeout in t=6 and all of them transit in The same time to the Next state
Node 1
1.Event (t=0): Node 2 creates and transmits the message with t_i=0 and t_d=6
Node 2
3.Event (t=2): Node 3 receives the message from node 2 and stores t_i=0 and t_d=6
Node 3
4.Event (t=4): Node 4 receives the message from node 3 and stores t_i=0 and t_d=6
Node 4
0
1
2
3
4
5
6
7
8
9
10
t
Alert state
Next state

Figure 10: Demonstration of the time synchronization

6. General description of the algorithm

In this chapter, we intend to describe the proposed distributed algorithm. In the beginning of this chapter, we will focus on the top level description and the basic idea of the algorithm. Finally, we will declare syntax notation used in the algorithm description.

6.1. Aim of the algorithm

When the node k fulfils a condition in Equation (10), then it is able to work in *Operating state*. It means, that every node executes all tasks, which are mapped on it. In other words, the network works correctly, when all nodes are in the Operate state. However, when any node k does not fulfill the condition in Equation (10), then the repairing distributed algorithm has to be launched.

$$TR < rep_k, \forall k \in N^A, TR \in \mathbb{R}_{\geq 0} \qquad (10)$$

The purpose of the Distributed algorithm is to change a task mapping that way, that all nodes involved in the algorithm fulfill the condition in Equation (10). The node, which does not fulfill the condition transits into Error state and initiates task remapping.

In Figure 11, there is illustrated state diagram for all nodes in the network. The circle shape marks the state, in which can be node. The rectangle shape contains a set of states. The green circle denotes Operate state. If the node does not fulfill the condition in Equation (10), then it transits into set of states named Distributed algorithm. We will focus on interpretation of individual states of the Distributed algorithm in the next subchapter. Consequently, when the Distributed algorithm finishes successfully then all nodes transits return to Operating state. The return is labeled by edge in graph named Repaired. In the opposite case, when the algorithm does not find solution, it means that some node does not fulfill a condition in Equation (10). Subsequently, that node transits from Distributed algorithm via edge named Non-repaired to Error state.

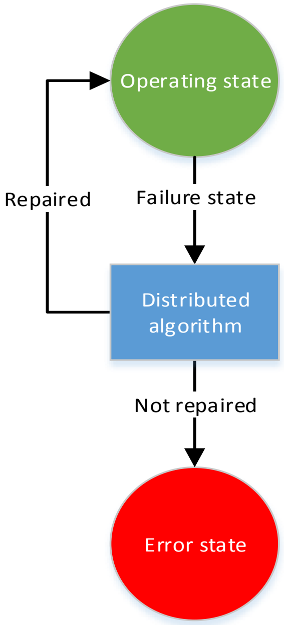


Figure 11: State machine of the network

6.2. Notations used in algorithm

This subchapter declares terms used in the distributed algorithm. The list of terms contains the following:

- *Threshold condition*: denotes a condition expressed in Equation (10).
- *Failure state*: is the situation or state in the network, when at least one node $k \in N^A$, does not fulfill the Threshold condition
- *Failure node*: is the node $k \in N^A$, which does not fulfill the Threshold condition and invokes the distributed algorithm
- *Alert Range (AR)*: denotes a distance in count of hops. The failure node declares the message, which has an attribute name=Alert. Alert Range is equal to the distance attribute of that message.
- *Alert Field (AF)*: denotes a set of nodes informed by a message with attribute name=Alert. Nodes included in Alert Field take part in the distributed algorithm.
- *Discovery Range (DR)*: denotes a distance in count of hops. Each node in Discovery state declares the message with attribute name Discovery Message. The distance attribute in message is given by a count of hops.
- *Discovery Field (DF_k)*: represents a set of nodes distanced by the size of DR from node k. When the node is not included in AF, then it isn't included in the Discovery Field of node k.
- *Minimum repetition in the Discovery Field of node k (MDF_k)*: is the minimum repetition value in Discovery Field of node k and is calculated $MDF_k = \min_{\forall i \in DF_k} \{rep_i\}$.

- *Improvement of the minimum repetition in the Discovery Field of node k* $\mathrm{MDF}_{k(imp)}$ denotes difference between MDF_k before($\mathrm{MDF}_{k(before)}$) and after ($\mathrm{MDF}_{k(after)}$) calculating new task mapping M in DF_k.

$$\mathrm{MDF}_{k(imp)} = \begin{cases} 0, & \mathrm{MDF}_{k(after)} - \mathrm{MDF}_{k(before)} \leq 0 \\ \mathrm{MDF}_{k(after)} - \mathrm{MDF}_{k(before)}, & \mathrm{MDF}_{k(after)} - \mathrm{MDF}_{k(before)} > 0 \end{cases}$$

6.3. State diagram of distributed algorithm

The state diagram of the distributed algorithm is illustrated in Figure 12. The state diagram is represented by the blue box in Figure 11. The distributed algorithm is launched, when any node k is in the failure state. The failure state occurs, when the Threshold condition is not fulfilled by any node k. Then the node k transits from Operating state to the first state of the distributed algorithm and starts reparation. The first state of the algorithm is named *Alert*. When the failure node transits into Alert state, then it initiates a message with attribute name Alert and into the distance attribute pastes AR. When any node receives that message, then it retransmits the message, if the distance is greater than zero. Secondly all nodes from AF transit to the Discovery state. In Discovery state every node needs to find out the *unknown attributes*. Consequently every node in AF transmits a message with attribute name Discovery and distance DR. By that way every node k in AF finds out *unknown attributes* from all nodes in DF_k. In Remapping state each node k in AF calculates new mapping in its DF_k and finds out value of $\mathrm{MDF}_{k(imp)}$. Afterwards each node k transits into Negotiation state. Node k with a higher value $\mathrm{MDF}_{k(imp)}$ will transmit its value to the network earlier than node l with a smaller value of $\mathrm{MDF}_{l(imp)}$. The aim of the Negotiation state is winning nodes with the highest values of MDF. The next state is named Solution into which nodes transits from Negotiation state. In that state nodes, which won in the Negotiation state, will transmit their newly calculated task mapping M, from which was calculated value of $\mathrm{MDF}_{k(after)}$. Thereafter every node k in Result state sends a message to the failure node, whether it was remapped or not in Solution state. Finally all nodes transits into Verdict state. In Verdict state, failure node evaluates, if all nodes in AF fulfil the Threshold condition. If any node in AF does not fulfil the Threshold condition and any node change its task mapping less three times within the same DR, then failure node sends a message to all nodes in AF that the next state will be Discovery. However when the failure node found out that, all nodes fulfil the condition then sends a message, all nodes in AF transit into Operating state. In this case the distributed algorithm solved the failure state of the network. In the remain case, failure node figured out that the failure state could not be solved by distributed algorithm and transits to Error state.

6. General description of the algorithm

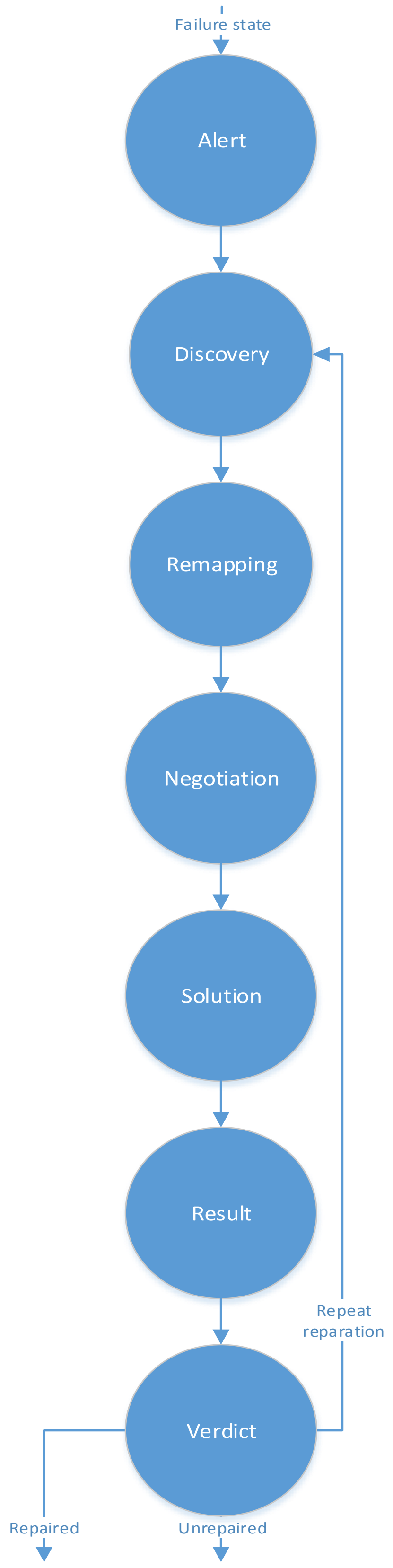


Figure 12: State diagram of distributed algorithm

7. Description of individual states of algorithm

In this chapter we aim to describe all states of algorithm in detail.

7.1. Operating state

Operating state is desired for each node in the network. In the Operating state the devices executes tasks mapped on it. If all nodes in the network work in this state, then the whole network works in correct way. We can see pseudo-code of Operating state:

- **input:** node k (problem node), which fulfils Threshold condition
- **process:** node k executes all tasks mapped on it
- **output:** node k, which made one iteration in the network

Algorithm 1: Pseudo-code of Operating state

7.2. Alert state

When in the network any node k becomes a failure node, then it has to transit into Alert state send message to involve other nodes to the reparation of the network. The main purpose of that state is to inform and involve into algorithm all nodes up to distance AR from the failure node. When any node receives a message with attribute name Alert, it stops to work in Operating state and transits to Alert state.

- **input:** failure node
- **process:** failure node initiates a message with attribute name Alert, with distance attribute equals to AR and sends it to the network.
- **output:** which sent initiated and sent the message

Algorithm 2: Pseudo-code of Alert state

- **input:** node, which received message with attribute name Alert
- **process:** if node works in Operating state then it transits into Alert state. Node stores a message content (failure node identification, AR, distance from failure node). If the message attribute distance has nonzero count of hops, then the node broadcasts that message into the network.
- **output:** node, which sent or did not send message type Alert to the network

Algorithm 3: Pseudo-code 2 of Alert state

Communication manner used in this part of the algorithm is One-to-All. The failure node represents one device, which has to send a message with attribute name Alert to all other nodes in AF. *In further states of the distributed algorithm is spoken only about nodes in AF. In other words, nodes which were included into reparation algorithm by failure node.*

7.3. Discovery state

Aim of each node k in Discovery state is to find out *unknown set* of attributes about each node in DR_k. In the beginning of Discovery state, every node k initiates and transmits message with attribute name Discovery with distance attribute equals to DR. Each node k stores its *known attributes* into optional attributes of the message. Consequently, when any node receives the message, then it stores its content (*known attributes* of node l, which created the message in DR_k) and resends it.

- **input:** node k, which transited after elapsed timeout from Alert to Discovery state
- **process:** node k initiates a message with attribute name Discovery with attribute distance equal to DR. Into optional attributes of the message are stored its *known attributes*.
- **output:** node k, which transmitted a message with attribute name Discovery

Algorithm 4: Pseudo-code of Discovery state

- **input:** node k, which received a message with attribute name Discovery
- **process:** node stores a message content (*known attributes* of node l in DR_k). If the message attribute distance is nonzero count of hops, then the node resends it into the network. From the path attribute in the message, node k is able to find out path in G^A from node l.
- **output:** node k, which sent or did not send message type Message to the network or waits for receiving a message type Discovery

Algorithm 5: Pseudo-code 2 of Discovery state

Each node k in Discovery state has to find out unknown attributes by collecting messages. It means that it must know all attributes about each node $i \in DF_k$:

- path in from node k to node i in architecture graph G^A
- tasks mapping M_i
- resource capacity R_i

7.4. Remapping state

In the Remapping state node k has to compute value of $\mathrm{MDF}_{k(imp)}$. Firstly node stores value of tasks mapping of $\mathrm{MDF}_{k(before)}$ before launching of the remapping algorithm. Secondly node k calculates new task mapping using by remapping algorithm. From newly calculated task mapping, node k computes the value of $\mathrm{MDF}_{k(after)}$. Finally node k is able to compute $\mathrm{MDF}_{k(imp)}$.

- **input:** node k, which transited from Discovery state to Remapping state
- **process:** node k, computes value of $\mathrm{MDF}_{k(before)}$ from all nodes in DF_k. Afterwards node k by using remapping algorithm calculates new tasks mapping in DF_k and then it stores value of $\mathrm{MDF}_{k(after)}$. Finally it makes difference between $\mathrm{MDF}_{k(after)}$ and $\mathrm{MDF}_{k(before)}$ and stores its value into $\mathrm{MDF}_{k(imp)}$.
- **output:** each node k calculated the value of $\mathrm{MDF}_{k(imp)}$.

Algorithm 6: Pseudo-code of Remapping state

In case that node k managed to compute $\mathrm{MDF}_{k(imp)} > 0$, then it found out a better tasks mapping in DF_k. In the opposite case, when node k found the value of $\mathrm{MDF}_{k(imp)} = 0$, then it wasn't able to find out an improving task mapping in DF_k.

7.4.1. Description of remapping algorithm

The remapping algorithm is based on a list scheduling approach with a backtracking step. The algorithm computes the maximal value of repetition objective for all nodes in DF_k, for which it exists the feasible task mapping M defined in Chapter 3.

For remapping algorithm, we have to declare the *initial set of variables*:

- *set of nodes* S_n: $S_n = \{\forall l \in N^A: (i,k) \in M \wedge l \in DF_K\}$
- *set of tasks* S_t: $S_t = \{\forall i \in N^T: (i,k) \in M \wedge k \in S_n\}$
- *tasks mapping* M: task mapping of tasks $\mathrm{S_T}$ before launching of remapping algorithm
- *the upper bound* rep_{max} : $\mathrm{rep}_{max} = \frac{\max_{i \in S_n}\{R_i\}}{\min_{j \in S_t}\{r_j\}}$
- *the lower bound* rep_{min} : $\mathrm{rep}_{min} = \mathrm{MDF}_{k(before)}$
- *number of tasks* $numTasks$: $\mathrm{numTasks} = |\mathrm{S_t}|$
- *number of nodes* $numNodes$: $\mathrm{numNodes} = |\mathrm{S}_n|$
- *found repetition value* rep^*: $\mathrm{rep}^* = \mathrm{rep}_{min}$
- *found tasks mapping* M_F: $\mathrm{M_F}$ found tasks mapping in function $findMapping$
- *result tasks mapping* M^*: $\mathrm{M}^* = \mathrm{M}$

In Figure 13, a state diagram of the remapping algorithm is illustrated. In the beginning, the initial set of variables is calculated. Algorithm iterates between the upper bound rep_{max} and the lower bound rep_{min}. Every time the algorithm computes an average value of rep_{max} and rep_{min} and stores it into variable rep. The value of rep is an input parameter of function $findMapping(\mathrm{rep})$. The function $findMapping$ has an assignment to find out the feasible tasks mapping for input number repetitions rep. As output of function $findMapping$ is tasks mapping, which stored into variable $\mathrm{M_F}$. Afterwards it is checked the condition, if the tasks mapping $\mathrm{M_F}$ is feasible or not. In case that the mapping $\mathrm{M_F}$ is feasible then it is stored into variable $\mathrm{M^*}$ and also is stored rep value into $\mathrm{rep^*}$. After that is increased the lower bound rep_{min}. In the opposite case, when the returned mapping $\mathrm{M_F}$ isn't feasible then it decreased the upper bound rep_{max}. Finally, when the remapping algorithm finished then it calculates the value of $\mathrm{MDF_{k(imp)}}$. If the algorithm found a better tasks mapping then it had calculate $\mathrm{MDF_{k(imp)}} > 0$ and otherwise.

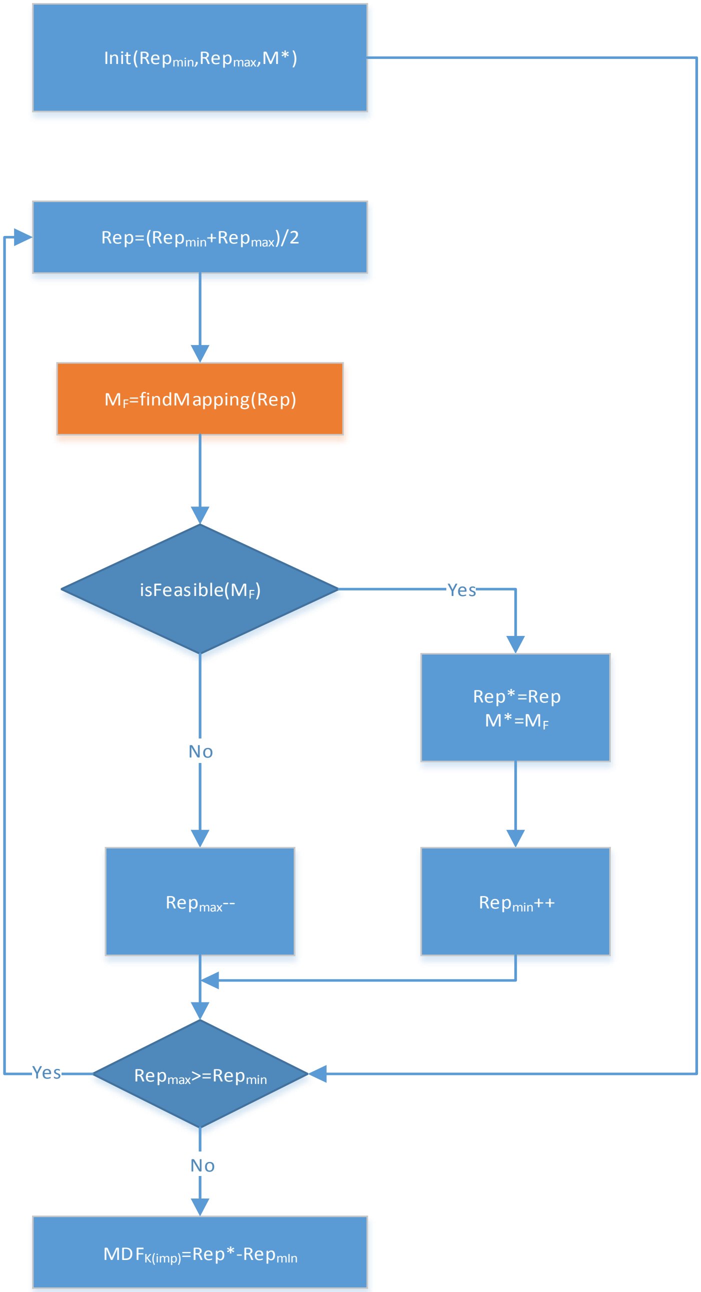

Init(Rep_{min},Rep_{max},M*)
Rep=(Rep_{min}+Rep_{max})/2
M_F=findMapping(Rep)
isFeasible(M_F)
Yes
No
Rep*=Rep
M*=M_F
Rep_{max}--
Rep_{min}++
Yes
Rep_{max}>=Rep_{min}
No
MDF_{K(imp)}=Rep*-Rep_{min}

Figure 13: State diagram of the algorithm

7.4.2. Description of the function for searching the mapping

In the following section we will concentrate on the explanation of how function $findMapping$ works. Aim of the $findMapping(rep)$ function is to find feasible task mapping M for defined size of the rep value. It implies that on each node $i \in S_n$ has to be fulfilled the following condition $rep_i < rep$.

Firstly, all tasks in S_t are ordered into $taskList$. Tasks are ordered in the $taskList$ according to longest paths. Secondly, all node resource capacities R_i where $i \in S_n$ are divided by input argument rep. Thirdly the $budget$ variable is calculated. After that algorithm iterates in while loop up to size of budget variable. In the loop a task from $taskList$ a is taken and then is passed as input argument into function $findDevice$. That function has to find the most appropriate node $i \in S_n$ on which can be mapped the input task. The $findDevice$ function returns an empty node or node identification. When it is returned an empty node, then it wasn't found any node $i \in S_n$ and the task can't be mapped on any node $i \in S_n$. Then the $findMapping$ function is terminated and returns an empty mapping. In the opposite case when any node $i \in S_n$ is found for the input task from $taskList$, then the function continues and chooses a new task from the $taskList$. When the $taskList$ is empty then it was found tasks mapping for all tasks from S_t. That mapping is returned by function $findMapping$ and it denotes that it was found tasks mapping for input given value of input argument rep.

```
function findMapping(rep)
    budget=(numTasks*numTasks);
    forbiddenMapping=(numTasks,numNodes);
    taskList=orderTasks(St);
    mapping=empty;
    for i=1:size(Sn)
        Ri=Ri/rep;
    end
    while(budget>0)
        task=getTaskFromList(taskList);
        [device,forbiddenMapping]=findDevice(task);
        if(isempty(device))
            mapping=empty;
            return mapping;
        end
        mapping(task)=device;
        if(isempty(taskList))
            return mapping;
        end
        budget--;
    end
    mapping=empty;
    return mapping;
end
```

Algorithm 7

7.4.3. Find mapping device function

In the function $findMapping$ is used the function $findDevice(task)$. The function has to find for input task $j \in S_t$ node $i \in S_n$. Selected node $i \in S_n$ has to fulfill the constrain that the task binding has to be feasible. The main purpose of the function $findDevice$ is to construct six subsets of nodes $i \in S_n$. Each subset is constructed using restrictions on set S_n. We define four restrictions for input task i:

- **C(capacity):** a set C includes all nodes from S_n, which fulfill the following inequality:

$$C = \left\{ \forall k \in S_n : (R_K/rep) \geq \left(cost(i,k) + \sum_{\forall l \in S_t : (l,k) \in M} cost(l,k) \right) \right\}$$

- **P (predecessor):** set P includes all nodes from S_n, which fulfill the following restriction: $P = \left\{ \forall k \in S_n : \{ \forall j \in N^T, \forall l \in \partial(k) : \left((i,j) \in E^T \vee (j,i) \in E^T \right) \wedge (j,l) \in M \} \right\}$, where:

$$\partial(k) = \{ k \cup D(k) \}$$

$$D(k) = \left\{ \forall m \in S_n : \left((k,m) \in E^A \vee (m,k) \in E^A \right) \right\}$$

- **F(forbidden):** a set F includes all nodes from S_n, which fulfill the following restriction:

$$F = \{ \forall k \in S_n : FM(i,k) = 0 \}$$

- **E(enable):** a set E includes all nodes from S_n, which fulfill the following restriction:

$$E = \{ \forall k \in S_n : (R_K/rep) \geq cost(i,k) \}$$

In function $createSets()$ are made these four sets (C, P, F, E). After that, the condition, if the set E is empty is checked. In case that set E is empty, then there does not exist any device, on which can mapped input task i. It implies also that we can't map task i on any node $k \in S_n$. It also implies that it is impossible find the feasible mapping. When the set E is not empty after that is made up six sets. Sets (S1, S2, S3, S4, S5 and S6) are generated by intersections from four sets (C, P, F, E). Aim of the function $findDevice$ is to find first node $k \in S_n$, for which it exists feasible tasks mapping defined in chapter 3. Finally, when the feasible tasks mapping is not found, then it is returned to an empty device. Function $chooseDeviceMinPrice$ has as input argument set of devices. The function chooses node k for which exists feasible tasks mapping and has the minimum sum of $cost(i,k)$ all tasks, which are already mapped. The function can return the new forbidden mapping table and tasks, which had to be unmapped from devices.

```
function findDevice(task)
price=empty;
[C,P,F,E]=createSets();
if(isempty(E))
   feasibleDevice=false;
   return feasibleDevice;
end
S1=intersection(C,P,F,E);
S2=intersection(C,P,E);
S3=intersection(C,F,E);
S4=intersection(C,E);
S5=intersection(F,E);
S6=E;
tmpPrecedors=empty;
tmpForbidden=forbiddenMapping;
```

```
tmpPrice=-Inf;
foundPredescorsTasks=empty;
foundForbidden=empty;
[device]=chooseDeviceMinPrice(S1);
if(!isempty(device))
   return device;
end
[device]=chooseDeviceMinPrice(S2);
if(!isempty(device))
   return device;
end
[device]=chooseDeviceMinPrice(S3);
if(!isempty(device))
   return [device,forbiddenMapping];
end
[device]=chooseDeviceMinPrice(S4);
if(!isempty(device))
   return [device,forbiddenMapping,tasksOut];
end
[device]=chooseDeviceMinPrice(S5);
if(!isempty(device))
   return [device,forbiddenMapping,tasksOut];
end
[device]=chooseDeviceMinPrice(S6);
if(!isempty(device))
   return [device,forbiddenMapping,tasksOut];
end
return empty;
end
```

Algorithm 8

In the end of this subchapter we will show the time complexity of algorithm. Firstly, in main loop of remapping algorithm makes count of iterations equals to $(\mathrm{rep}_{\mathrm{max}} - \mathrm{rep}_{\mathrm{min}} + 1)$. Secondly in the function $findMapping$ makes count of iterations equals to budget variable. Finally, we call the function $findDevice$, which iterates throw six sets of devices. The maximum size of the set of devices is equal to $numNodes$. So it means that maximum count of iteration throw these six sets is $6 * \mathrm{num\ Devices}$. So finally, the time complexity responds the multiplications in the loop.

$$O\big((\mathrm{rep}_{max} - \mathrm{rep}_{min} + 1) * (numTasks)^2 * (numNodes * 6)\big) \qquad (11)$$

7.5. Negotiation solution

In Negotiation state every node k has a calculated value of $\mathrm{MDF}_{k(imp)}$. Aim of that state is that each node k in AF has to get know the biggest value of $\mathrm{MDF}_{l(imp)}$ of node l, which is maximally distanced DR and $2 * \mathrm{DR}$ from node k. Furthermore in Negotiation state is used a heuristic, which reduces a communication of nodes during negotiating. The main idea of heuristic is to delay transmitting smaller value of $\mathrm{MDF}_{k(imp)}$, because in its neighborhood can be another node l with a higher value of $\mathrm{MDF}_{l(imp)}$. When node k receives from node l the value of $\mathrm{MDF}_{l(imp)} > \mathrm{MDF}_{k(imp)}$, then node l would transmit unnecessarily its value of $\mathrm{MDF}_{k(imp)}$ to network.

For getting a better view on Negotiation state, we illustrated a possible situation in Figure 14. It can be seen that maximum values of $\text{MDF}_{(imp)}$ for node 3. The biggest value is equal to $\text{MDF}_{4(imp)} = 7$ in DF_3. The biggest value is $\text{MDF}_{5(imp)} = 9$ up to $2 * DR = 2$ from node 3. By the way illustrated in Figure 14, each node k has to find out the highest values of $\text{MDF}_{l(imp)}$. We can observe one fact from Figure 14, that node 3 does not need to transmit its value because by it will be overwritten by the higher value of $\text{MDF}_{2(imp)}$ or $\text{MDF}_{4(imp)}$ and etc.. It implies that if node 3 will be delay with transmitting its value $\text{MDF}_{3(imp)}$ and during this delay receives the value of $\text{MDF}_{4(imp)}$ or $\text{MDF}_{2(imp)}$, then it needn't to transmit its value $\text{MDF}_{3(imp)}$, because it received the higher one from another node.

Every node k has defined four variables in negotiation state:

- $maxValue_{k(DR)}$: contains the biggest value of $\text{MDF}_{x(imp)}$, where $x \in \text{DF}_k$
- $indexMaxNode_{k(DR)}$: contains the node x, with the $maxValue_{k(DR)}$
- $maxValue_{k(2DR)}$: contains the biggest value of $\text{MDF}_{x(imp)}$, where $x \in (DR, 2DR)$
- $indexMaxNode_{k(2DR)}$: contains the node x, with the $maxValue_{k(2DR)}$
- $winFlag$: flag denoting, if the node won in Negotiation state

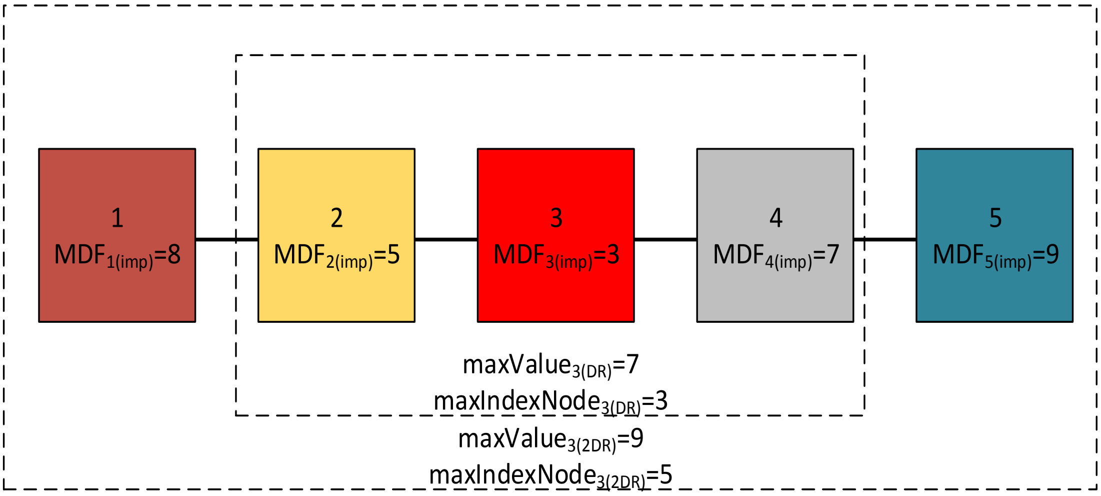


Figure 14: Maximum values for node 3 in Negotiation state

We declare the Delay function. The delay function is defined by the following way:

$$delay_k(\text{MDF}_{k(imp)}) = \begin{cases} \infty, \ \text{MDF}_{k(imp)} = 0 \\ MaxValue - \text{MDF}_{k(imp)}, \ \text{MDF}_{k(imp)} > 0 \ \wedge \text{MDF}_{k(imp)} < MaxValue \\ 0, \ \text{MDF}_{k(imp)} \geq MaxValue \end{cases}$$

$$MaxValue \in \mathbb{R}_{>0}$$

Equation (11): Delay function of node k

After elapsing $delay_k$ of node k, the node if it wasn't overwritten from another node l via received message, then it starts to transmit. In the opposite case, when the node k receives a message from node l during $delay_k$, then it has to make the following decision:

$$maxValue_{k(DR)} = \begin{cases} \mathrm{MDF}_{k(imp)}, & otherwise \\ \mathrm{MDF}_{l(imp)}, & \left(\mathrm{MDF}_{k(imp)} + \frac{k}{2^{16}}\right) < \left(\mathrm{MDF}_{l(imp)} + \frac{l}{2^{16}}\right) \end{cases}$$

$$maxIndexValue_{k(DR)} = \begin{cases} k, & otherwise \\ l, & \left(\mathrm{MDF}_{k(imp)} + \frac{k}{2^{16}}\right) < \left(\mathrm{MDF}_{l(imp)} + \frac{l}{2^{16}}\right) \end{cases}$$

Equation (12): Maximum value of node k in Negotiation state in DR

$$maxValue_{k(2DR)} = \begin{cases} \mathrm{MDF}_{k(imp)}, & otherwise \\ \mathrm{MDF}_{l(imp)}, & \left(\mathrm{MDF}_{k(imp)} + \frac{k}{2^{16}}\right) < \left(\mathrm{MDF}_{l(imp)} + \frac{l}{2^{16}}\right) \end{cases}$$

$$maxIndexValue_{k(2DR)} = \begin{cases} k, & otherwise \\ l, & \left(\mathrm{MDF}_{k(imp)} + \frac{k}{2^{16}}\right) < \left(\mathrm{MDF}_{l(imp)} + \frac{l}{2^{16}}\right) \end{cases}$$

Equation (13): Maximum value of node k in Negotiation state in 2DR

If the node k receives the message from node l, then it makes two comparisons declared in Equations (12) and (13). If in the received message is higher value of $\mathrm{MDF}_{k(imp)}$.

The algorithm for any node in Negotiation state is defined as follows:

- **input:** node k with value of $\mathrm{MDF}_{k(imp)}$
- **process:** in the begging node k calculates value of time delay $delay_k(\mathrm{MDF}_{k(imp)})$. During time $delay_k$ node k does not transmit its value of $\mathrm{MDF}_{k(imp)}$. If node receives a message then it executes adequate operation in Figure 15. If the node k during time $delay_k$ was overwritten by another node l during a *waiting phase*, then it does not transmit its value of $\mathrm{MDF}_{k(imp)}$ after elapsing time $delay_k$. In the opposite case node transmits its value of $\mathrm{MDF}_{k(imp)}$ after elapsing time $delay_k$.
- **output:** node k, which has set the $winFlag$

Algorithm 9: Pseudo-code of Negotiation state

The process in Algorithm 9, we also expressed in Figure 15 like a state diagram for a better overview. In the beginning each node k declares init variables:

- $maxValue_{k(DR)} = \mathrm{MDF}_{k(imp)} + \frac{k}{2^{16}}$
- $indexMaxNode_{k(DR)} = k$
- $maxValue_{k(2DR)} = \mathrm{MDF}_{k(imp)} + \frac{k}{2^{16}}$
- $indexMaxNode_{k(2DR)} = k$
- $winFlag = true$

The node k also calculates $delay_k$ function in Equation (11). Afterwards node transits into status named $Wait\ for\ delay_k$. In that waiting state node k, can receive message from node l and then it has to evaluate distance attribute in message, which it denotes the distance from the message was sent. If the distance is the smaller or equal to DR, then it calculates function in Equation (12). In the opposite case, when the node receives the message with the strictly higher value of distance attribute than DR, then it has to calculate Equation (13). If the value of $maxValue_{k(DR)}$ or $maxValue_{k(2DR)}$ was changed then the node sets the $winFlag = false$ and resends the message next to the network. Consequently, node transits into state named Evaluate received message. However the node k can't sends its value of $\mathrm{MDF}_{k(imp)}$, because it received the bigger value of $\mathrm{MDF}_{l(imp)}$ from node l. In the opposite case, when the node k did not received in state $Wait\ for\ delay_k$ any message from node l, which has the higher value of $\mathrm{MDF}_{l(imp)}$ than $maxValue_{k(DR)}$ or $maxValue_{k(2DR)}$, then the node k has to transmit the value of $\mathrm{MDF}_{k(imp)}$ in message up to distance $2DR$.

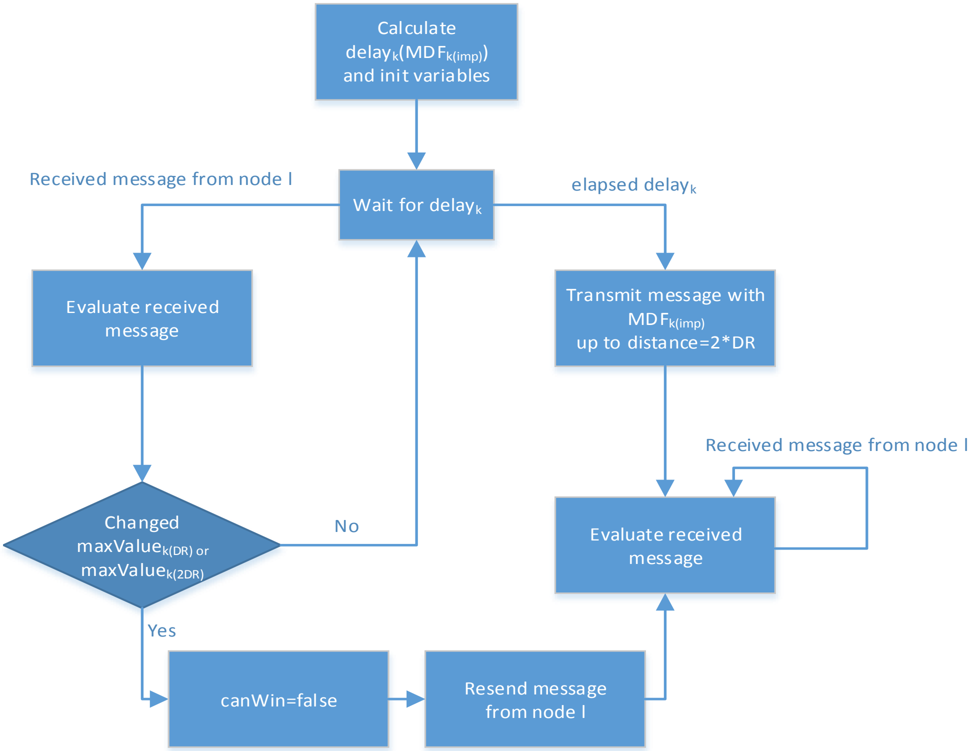

Calculate delayk(MDFk(imp)) and init variables
Received message from node l
Wait for delayk
elapsed delayk
Evaluate received message
Transmit message with MDFk(imp) up to distance=2*DR
Received message from node l
Changed maxValuek(DR) or maxValuek(2DR)
No
Evaluate received message
Yes
canWin=false
Resend message from node l

Figure 15: State diagram of process of each node in Negotiation state

7.6. Solution state

In that state each node k in AF has set $winFlag$ on true or false from Negotiation state. The purpose of the Solution state is transmitting of tasks mapping of nodes, which have set the $winFlag = true$. That nodes have to send the message with the optional attribute task mapping newly computed in Remapping state up to distance equals to DR.

- **input:** node k with $winFlag = true$
- **process:** node k transmits the message with attribute name Solution and with optional attribute equal to newly calculated task mapping in Remapping state up to distance DR.
- **output:** node k, which changed tasks mapping from received message or did not change the tasks mapping

Algorithm 10: Pseudo-code of Solution state

- **input:** node k, which received the message with attribute name Solution
- **process:** the node changes its tasks mapping according to task mapping in optional attribute in the received message. If the message has nonzero count of hops in distance attribute, then the message is resend.
- **output:** node k, which changed its task mapping from received message

Algorithm 11: Pseudo-code 2 of Solution state

7.7. Result state

Aim of that state is to send result state of node k in AF to failure node. When the node changed its tasks mapping in Solution state, then it will have to send message with attribute name Solution and optional attribute $remapFlag = true$ to the failure node, in the opposite case the node will send message with $remapFlag = false$. Communication used in that state is All-to-One.

- **input:** node k, which was or not remapped in Solution phase
- **process:** node k, initiates the message with the optional attribute $remapFlag$ with distance attribute equal to AR. If the failure node receives the message, then it stores its content and does not transmit it further to the network.
- **output:** node k, which transmitted the message into the network, if the node is failure then it received message from all nodes in AF

Algorithm 12: Pseudo-code of Result state

- **input:** failure node received the message with attribute name Result
- **process:** failure node stores *identification of node* and $remapFlag$ from optional attribute of the received message
- **output:** failure node collected information from message from node in AF

Algorithm 13: Pseudo-code 2 of Result state

- **input:** node k, which is not failure node received the message with attribute name Result
- **process:** node k only resends the message to the network if the message has nonzero count of hops
- **output:** node k, which resent the message into the network

Algorithm 14: Pseudo-code 3 of Result state

7.8. Verdict state

In Verdict state the failure node has to make decision process. The failure node can decide, whether all nodes in AF will return to Operating state or go to Error state or the algorithm will be repeated from Discovery state. Failure node sends a message with attribute name *Verdict* to inform all nodes in AF about next state.

- **input:** failure node
- **process:** failure node makes a decision about the next state of all nodes in AF. The decision process is illustrated in Figure 16. The failure node makes decision, then it sends message to all nodes in AF.
- **output:** node k, which transits into next state from message of type Verdict.

Algorithm 14: Pseudo-code of Verdict state

- **input:** node k, which received the message with attribute name Verdict
- **process:** node stores the name of the next stat from optional attribute of received message into that node transits after elapsing timeout of Verdict state. If the next state is Discovery node stores reads out from optional attribute of the message size of DR. If the message has nonzero count of hops in distance attribute, then it is resent into the network.
- **output:** node k, which obtain name of the next state from the received message

Algorithm 15: Pseudo-code 2 of Verdict state

The decision process is shown in a state diagram representing the decision process in Figure 16. The node has to make a decision based on the information gained in the Result state. Informing about the result of decision is made by sending a message with distance equals to AR. One exception is when during the decision process is decided that the next state will be an Error state.

First of all, the failure node starts the decision process in the block labelled as Start. Secondly is checked, if all nodes in AF fulfil the Threshold condition. When all nodes fulfil the condition the failure node informs all in AF that they can move into Operating state. When in AF exists any node which does not fulfill the Threshold condition, then the failure node has to evaluate was remapped at least one node in AF. When there is exists one node that was remapped and the algorithm was repeated in with the same size of DR less than three times, then all nodes will repeat the algorithm with the same size of DR. However when the algorithm was repeated with the same size of DR then it has to be DR increased. The increasing of DR is continued until is higher than AR. Finally when the $DR > AR$, then all nodes overcomes to Error state.

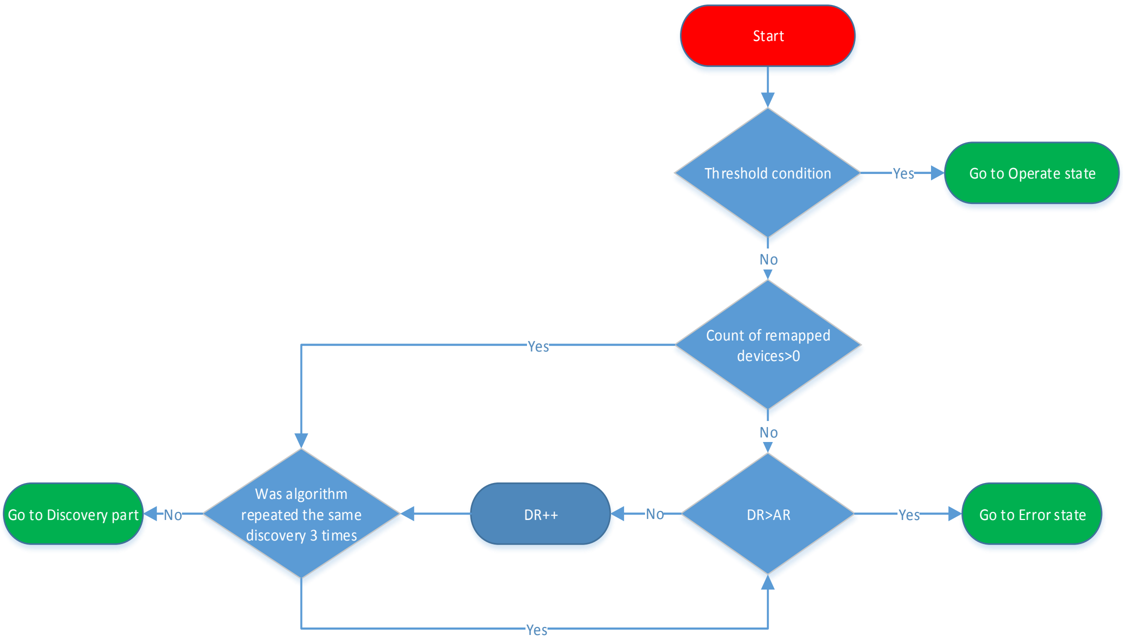


Figure 16: State diagram representing decision process of source node

8. Demonstration of the algorithm

In this chapter we will illustrate a workflow of the algorithm on a practical illustrated example.

8.1. Alert state

In Figure 17, we can notice that in the network occurred a Failure state. Node 5 does not fulfil the Threshold condition. The repetition value of node 5 is smaller than threshold value and node 5 is in Alert state. In that state node 5 notifies by a message all nodes up to AR=3. Consequently nodes 2,3,4,6,7 and 8 received the message initiated by node 5. These nodes were involved by node 5 into Alert state. In Figure 17 nodes 2,3,4,5,6,7 and 8 represents AF, because all of them are distanced maximally 3 hops from node 5.

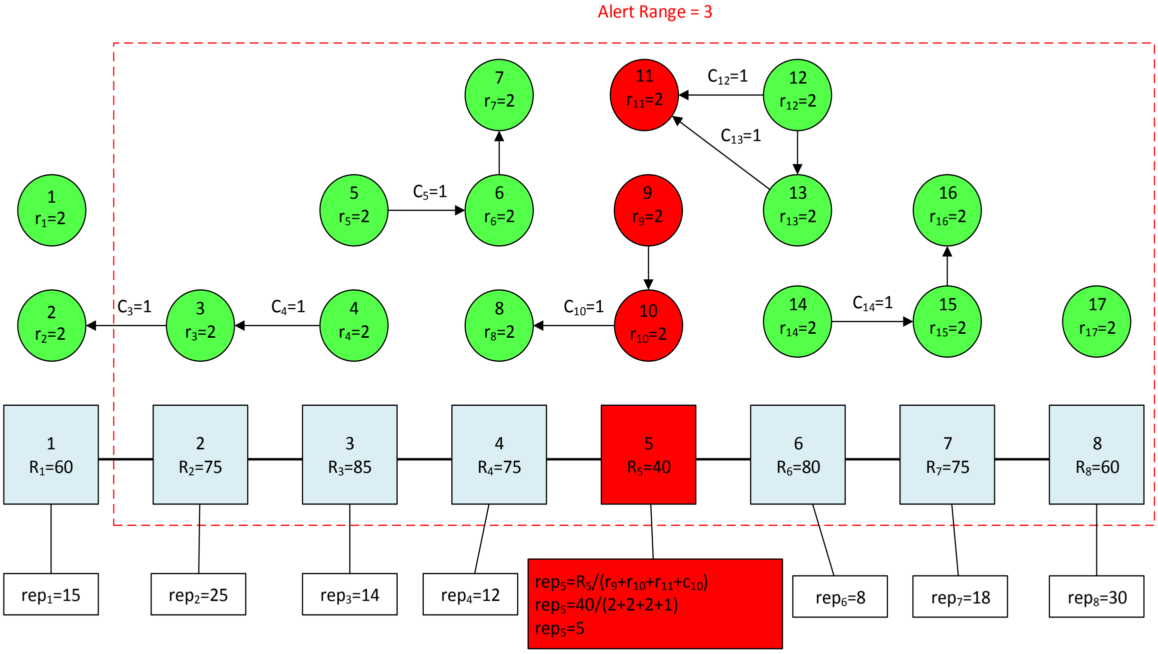


Figure 17: Alert state

8.2. Discovery state

Each node $\boldsymbol{k}$ in $\boldsymbol{AF}$ has to obtain unknown attributes from all nodes in its $\mathrm{DF_k}$. Consequently every node $\boldsymbol{k}$ in $\boldsymbol{AF}$ creates and starts to transmit a message into the network up to distance $\boldsymbol{DR} = \mathbf{1}$. In Figure 18 it is shown in radius rectangles $\mathrm{DF_k}$. The color of rectangle responds to color of the node.

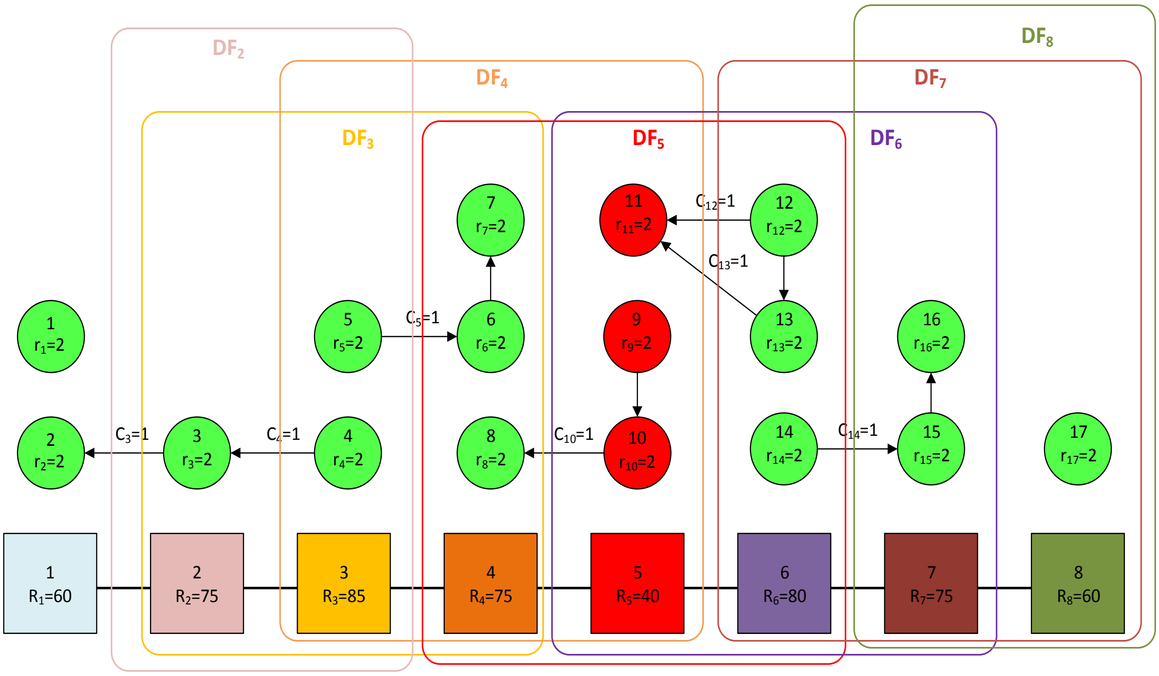
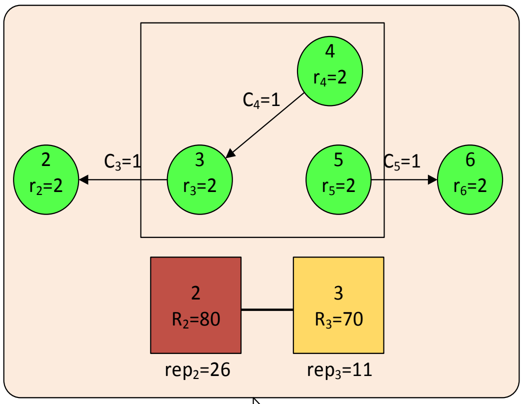


Figure 18: Discovery state with DR=1

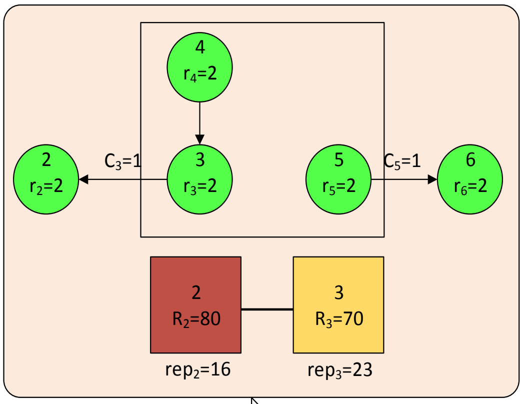
8.3. Remapping state

Every node k has to launch the remapping algorithm for calculating a new task mapping in its $\mathrm{DF_k}$. In Figure 19 it can be seen, how each node k calculates the value of $\mathrm{MDF_{k(before)}}$ before and after $\mathrm{MDF_{k(after)}}$ launching the remapping algorithm. Afterwards each node k also has to compute the value of $\mathrm{MDF_{k(imp)}}$.

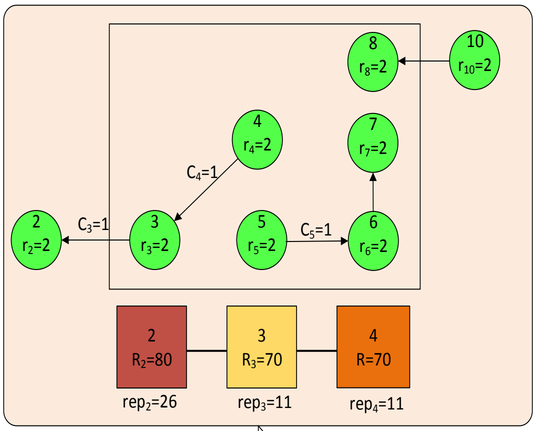
8. Demonstration of the algorithm



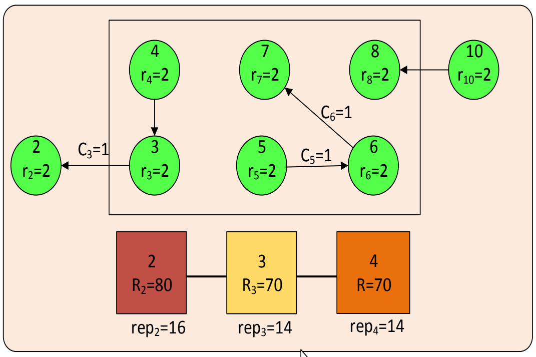
$MDF_{2(before)}=\min\{rep_2,rep_3\}=\min\{26,11\}=11$



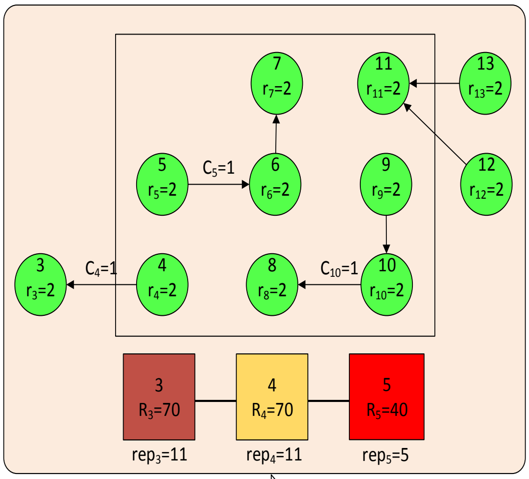
$MDF_{2(after)}=\min\{rep_2,rep_3\}=\min\{16,23\}=16$



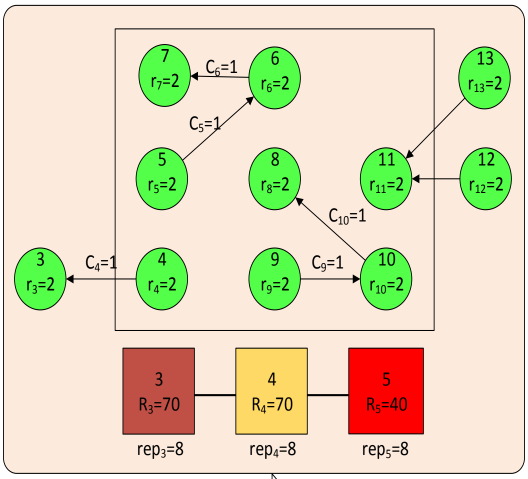
$MDF_{3(before)}=\min\{rep_2,rep_3,rep_4\}=\min\{26,11,11\}=11$



$MDF_{3(after)}=\min\{rep_2,rep_3,rep_4\}=\min\{26,14,14\}=14$

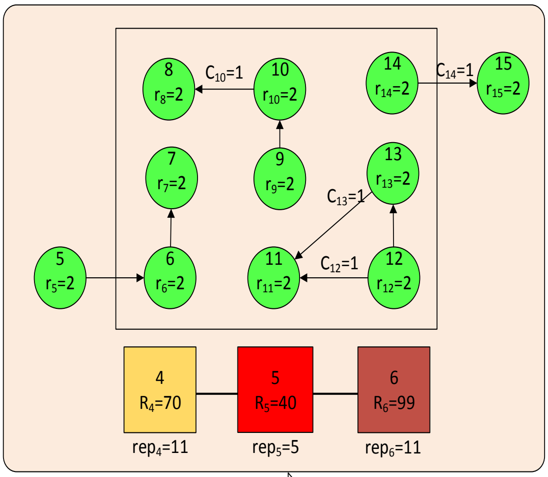


$MDF_{4(before)}=\min\{rep_3,rep_4,rep_5\}=\min\{11,11,5\}=5$

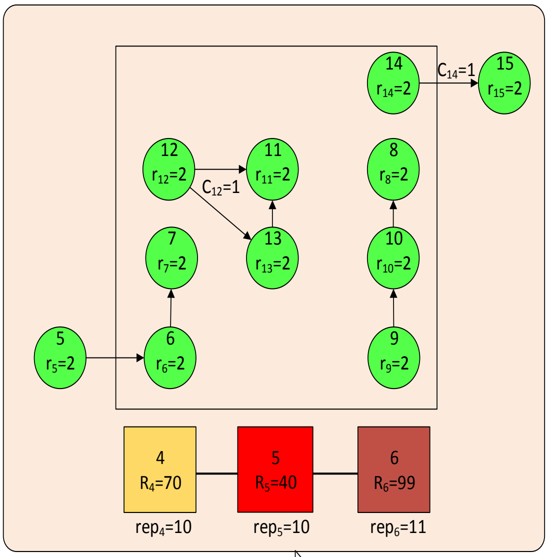


$MDF_{4(after)}=\min\{rep_3,rep_4,rep_5\}=\min\{8,8,8\}=8$

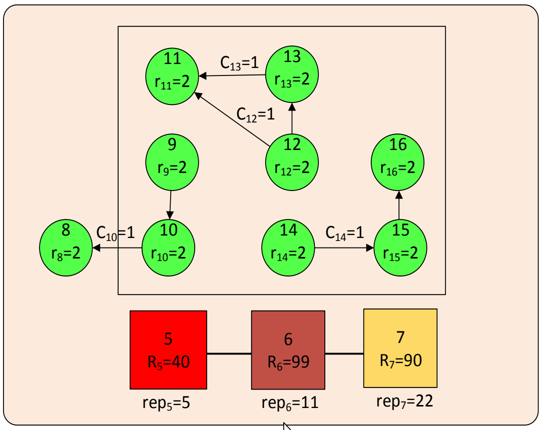
8. Demonstration of the algorithm



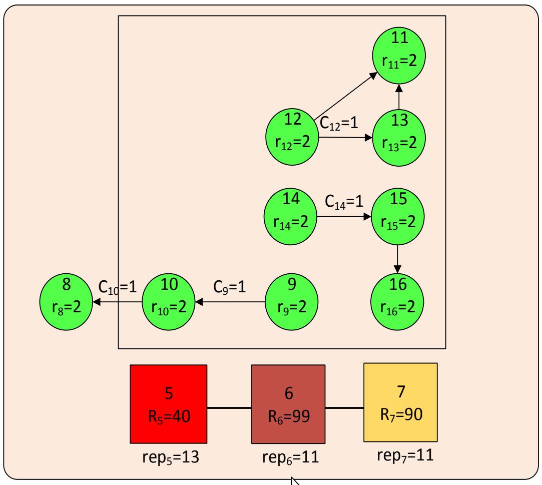
$MDF_{5(before)}$=min{rep_4,rep_5,rep_6}=min{11,5,11}=5



$MDF_{5(before)}$=min{rep_4,rep_5,rep_6}=min{11,10,11}=10



$MDF_{6(before)}$=min{rep_5,rep_6,rep_7}=min{5,11,22}=5



$MDF_{6(after)}$=min{rep_5,rep_6,rep_7}=min{13,11,11}=11

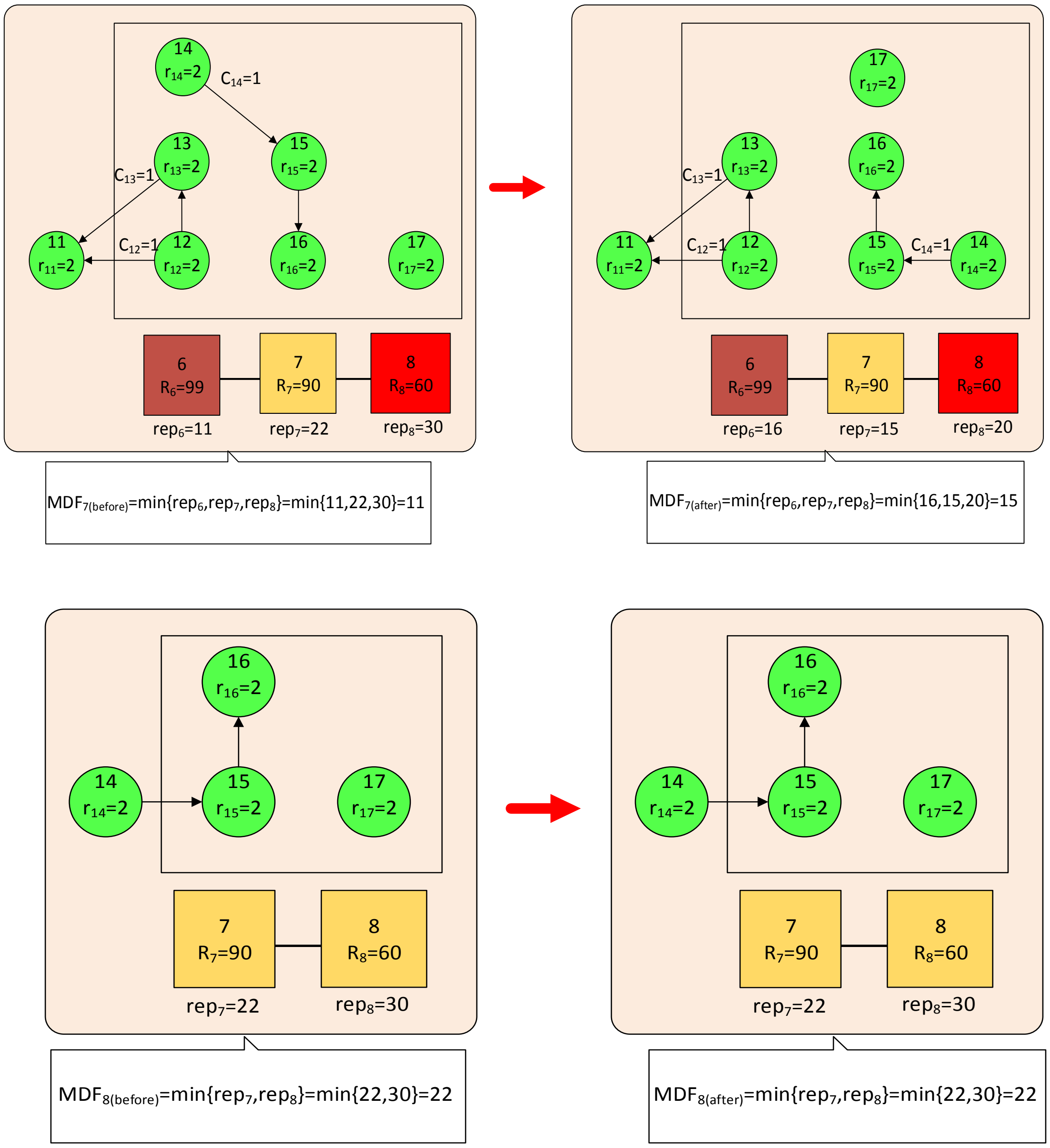


Figure 19: Application of remapping algorithm

8.4. Negotiation state

In the begging of this phase each node has to calculate the value of $\mathrm{delay}_{\mathrm{k}}(\mathrm{MDF}_{\mathrm{k(imp)}})$. In function $\mathrm{delay}_{\mathrm{k}}$ is chosen the $MaxValue = 7$. Consequently, each node initializes its variables for Negotiation state. In Figure 20: Negotiation state is illustrated Negotiation state. The first transmitting node is node 6, that will transmit iteration of the network. On the other side node 8 will not transmit, because has $\mathrm{MDF}_{8(\mathrm{imp})} = 0$ and it will only receive and resend messages. In the lower part of Figure 20: Negotiation state, the result of the Negotiation state is demonstrated. Nodes 2 and 6 won the Negotiation because in DF_2 and DF_6 did not exist the higher value of $\mathrm{MDF}_{\mathrm{k(imp)}}$.

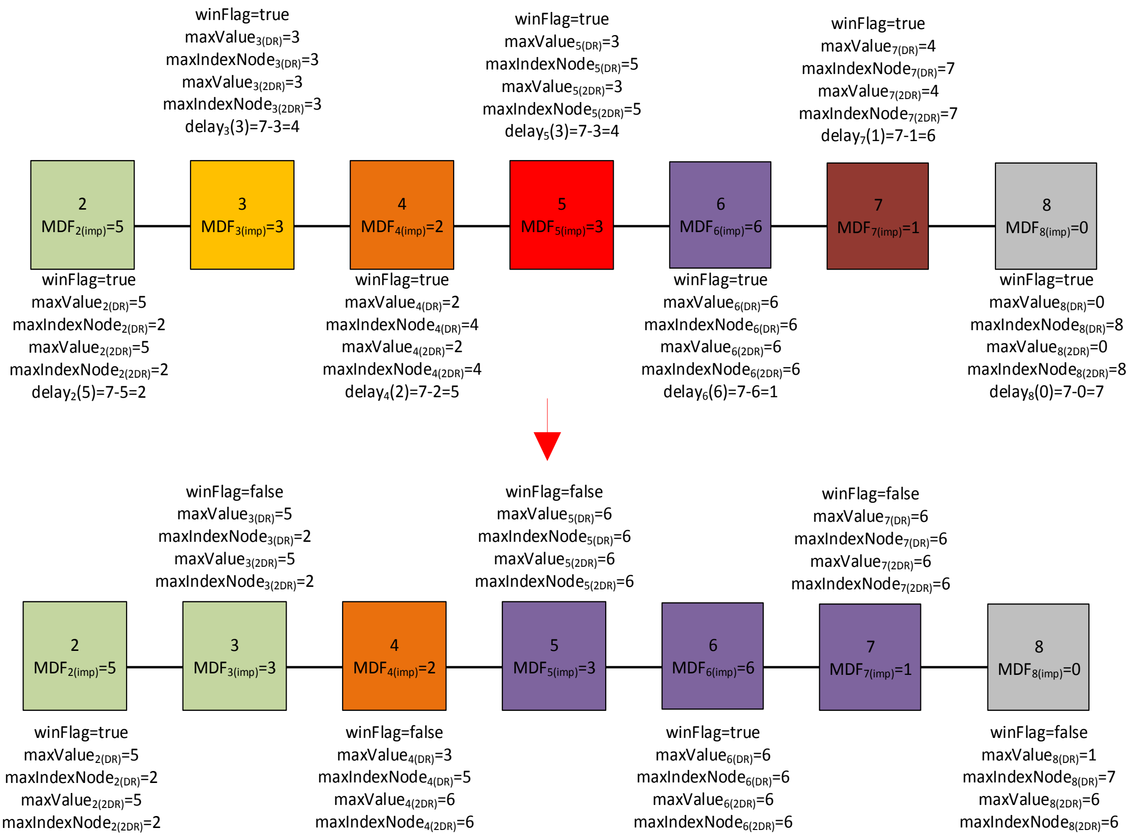


Figure 20: Negotiation state

8.5. Solution state

In Solution state will transmit all nodes, which have set $winFlag = true$. In Figure 21 are shown transmitted nodes 2 and 6 message with the newly calculated mapping Remapping state. Firstly, nodes 2 and 6 will transmit the message up to distance $DR = 1$. It means that all nodes in DF_2 and DF_6 will be remapped. All nodes in DF_6 will accept task mapping calculated in node 6, tasks which are remapped by node 6 are colored by purple color. Respectively node 3 will receive task mapping calculated by node 2. All tasks remapped in DF_2 are marked by jungle-green color.

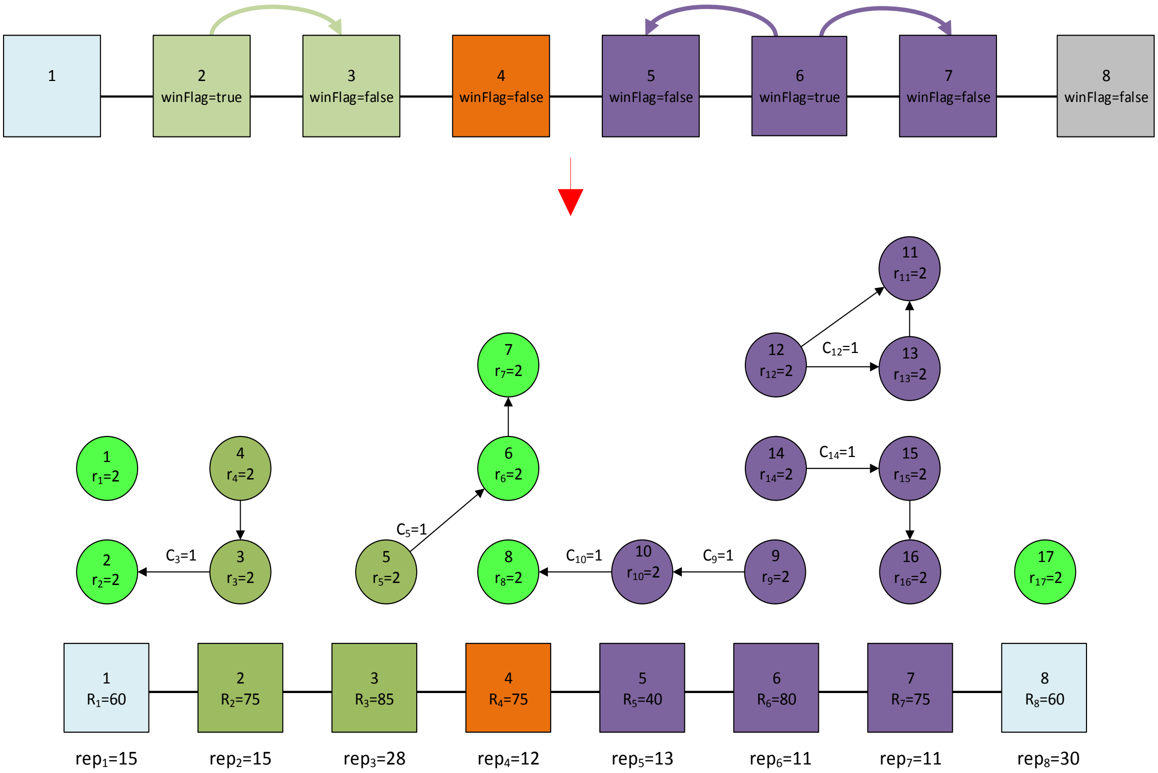


Figure 21: Solution state

8.6. Result state

All nodes in AF send a message to node failure node 5. Message contains $remapFlag$. When the $remapFlag$ is true it means that node was remapped in Solution state of the algorithm and otherwise. In Figure 22 is shown transmitting in Result state. Nodes 2,3,5,6 and 7 were remapped and sent the $remapFlag = true$ in message. On the other side remain nodes 4 and 8 in AF will transmit $remapFlag = false$.

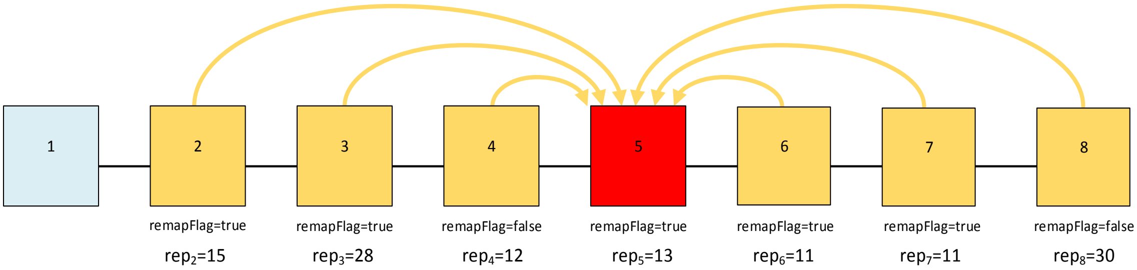


Figure 22: Result state

8.7. Verdict state

In verdict state the failure node 5 make decision about the next state. In Figure 23 failure node checks the Threshold condition. Node 5 finds out that all nodes fulfil the Threshold condition and then they can return to Operating state. After that consideration Node 5 will transmit the message that next state of all nodes will be Operating state.

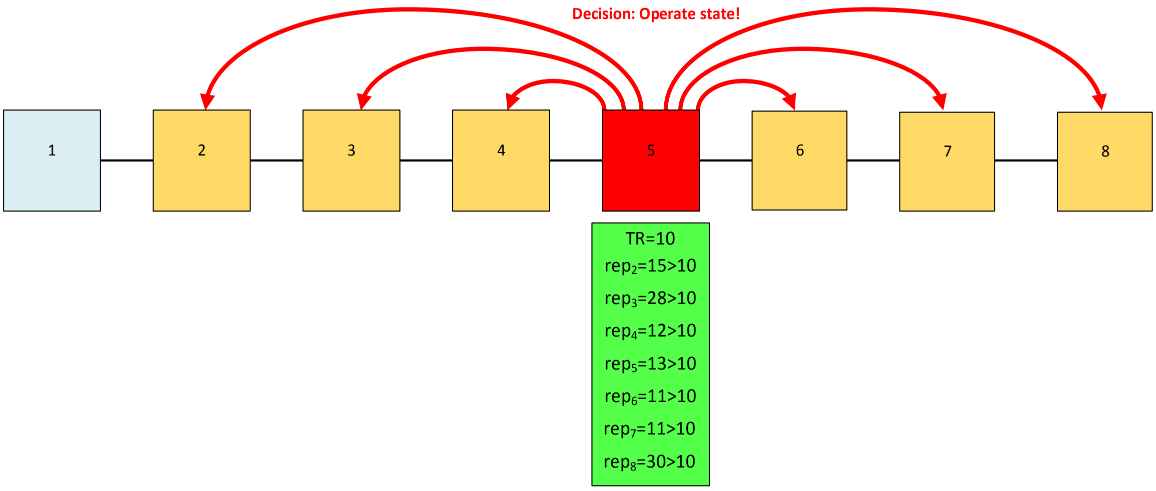


Figure 23: Verdict state

8.8. Operating state

In Figure 24 is illustrated Operating state of the network. All nodes fulfil the Threshold condition and they can execute tasks mapped on them. The distributed algorithm repaired the failure state of the network and the all nodes can work desired state.

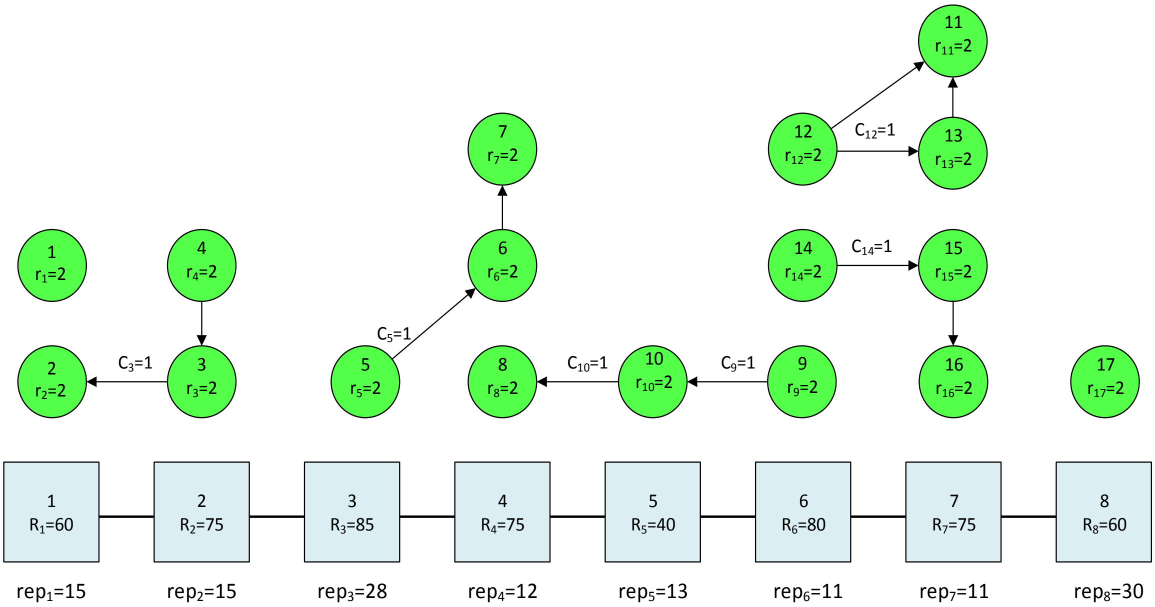


Figure 24: Operating state

9. Testing of the algorithm

In this chapter we want to interpret the test results of the simulation algorithm.

9.1. Description of the testing environment

For testing of the distributed algorithm we chose DynAA development tool created by the TNO team, which is available on **[Dynaa]**. DynAA is a computer-aided analysis and design tool for the development of large, distributed, adaptive, and networked systems. DynAA includes a simple, yet powerful language able to describe large and complex system architectures. DynAA model can be simulated and/or analysed to reveal system wide performance indicators, such as amount of communication, power consumption and etc..Thanks to mentioned properties of DynAA, our distributed system was programmed and modelled as object oriented model. Each physical node can be represented like *Device* object, which is able to communicate by using *link* object that substitutes a communication connection among nodes in the network. Each node also contains a state machine, which does steps triggered by time events in the network. In addition, DynAA works with discrete events. It means that system works in discrete time events and in each event every node makes one step in its state machine.

9.2. Testing of the distributed algorithm

For testing of the distributed algorithm we created 50 benchmark instances. Each instance represents a network of nodes with mapped tasks on them. In each instance there is one failure node, which has a low battery capacity and thereby it does not fulfil the Threshold condition. The main observed property of the algorithm was, whether the algorithm manages to increase the minimum repetition of failure device above the defined threshold in the Threshold condition. The second examined attribute was dispersion of the repetition values in Alert Field. We calculate the dispersion according to mathematical formula in Equation (14).

$$disp = \sum\nolimits_{\forall k \in AF} \left(rep_k - \frac{\sum_{\forall k \in AF} rep_k}{|AF|} \right)^2 \qquad (14)$$

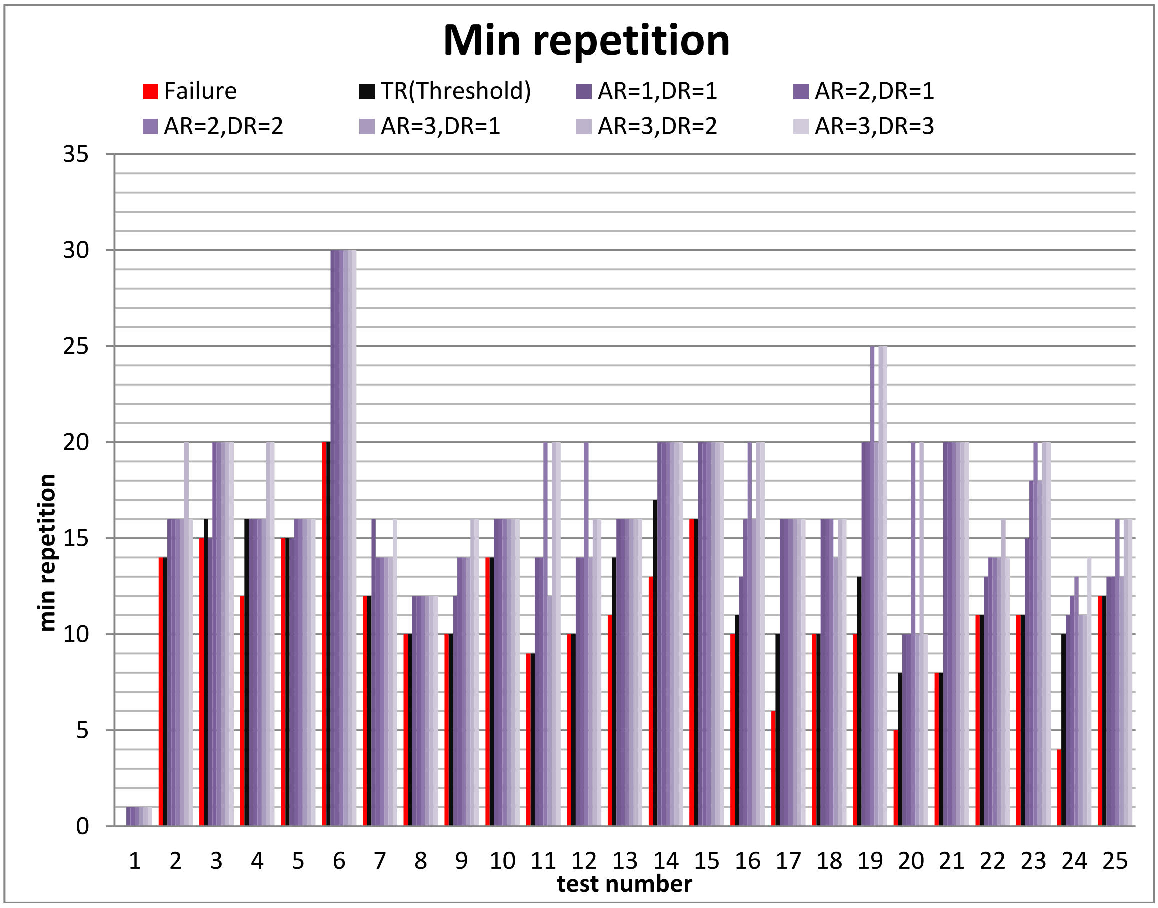
Table 1 shows tests results of simulation of the network with the minimum repetition value before and after simulation with variable sizes of AR and DR. In Table 1 the column named *Failure* represents the repetition value of the failure node. In the column titled TR(Threshold) is Threshold value from the Threshold condition. In remaining columns there are result values of the minimum repetition values in AF after finishing the algorithm for defined values of AR and DR.

Test	Failure	TR (Threshold)	AR=1, DR=1	AR=2, DR=1	AR=2, DR=2	AR=3, DR=1	AR=3, DR=2	AR=3, DR=3
1	0	0	1	1	1	1	1	1
2	14	14	16	16	16	16	20	16

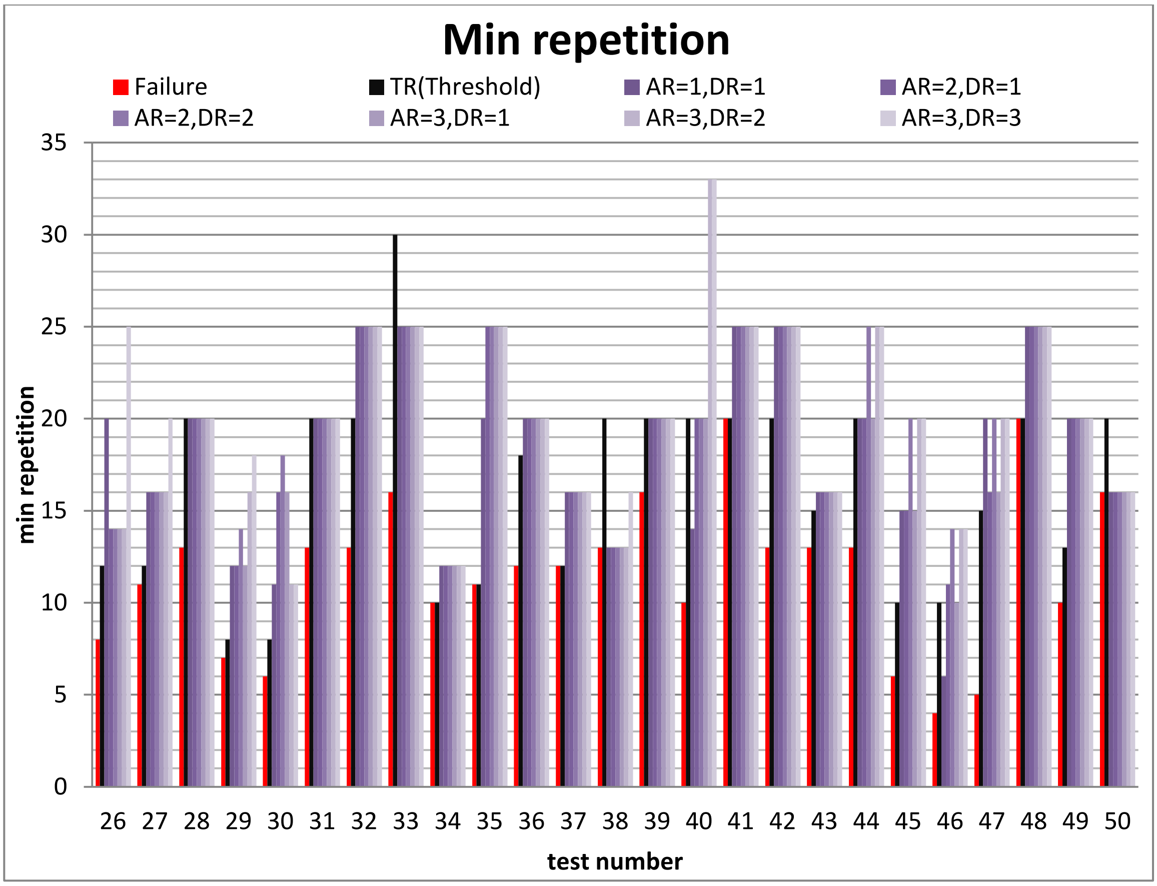
3	15	16	15	20	20	20	20	20
4	12	16	16	16	16	16	20	20
5	15	15	15	16	16	16	16	16
6	20	20	30	30	30	30	30	30
7	12	12	16	14	14	14	14	16
8	10	10	12	12	12	12	12	12
9	10	10	12	14	14	14	16	16
10	14	14	16	16	16	16	16	16
11	9	9	14	14	20	12	20	20
12	10	10	14	14	20	14	16	16
13	11	14	16	16	16	16	16	16
14	13	17	20	20	20	20	20	20
15	16	16	20	20	20	20	20	20
16	10	11	13	16	20	16	20	20
17	6	10	16	16	16	16	16	16
18	10	10	16	16	16	14	16	16
19	10	13	20	20	25	20	25	25
20	5	8	10	10	20	10	20	10
21	8	8	20	20	20	20	20	20
22	11	11	13	14	14	14	16	14
23	11	11	15	18	20	18	20	20
24	4	10	11	12	13	11	11	14
25	12	12	13	13	16	13	16	16
26	8	12	20	14	14	14	14	25
27	11	12	16	16	16	16	16	20
28	13	20	20	20	20	20	20	20
29	7	8	12	12	14	12	16	18
30	6	8	11	16	18	16	11	11
31	13	20	20	20	20	20	20	20
32	13	20	25	25	25	25	25	25
33	16	30	25	25	25	25	25	25
34	10	10	12	12	12	12	12	12
35	11	11	20	25	25	25	25	25
36	12	18	20	20	20	20	20	20
37	12	12	16	16	16	16	16	16
38	13	20	13	13	13	13	13	16
39	16	20	20	20	20	20	20	20
40	10	20	14	20	20	20	33	33
41	20	20	25	25	25	25	25	25
42	13	20	25	25	25	25	25	25
43	13	15	16	16	16	16	16	16
44	13	20	20	20	25	20	25	25
45	6	10	15	15	20	15	20	20
46	4	10	6	11	14	10	14	14
47	5	15	20	16	20	16	20	20
48	20	20	25	25	25	25	25	25
49	10	13	20	20	20	20	20	20
50	16	20	16	16	16	16	16	16

Table 1: Minimum repetitions before and after reparation

Graphs 1 and 2 summarize results from Table 1. The first graph shows results for instances 1 to 25 and the second one for 26 to 50. As a positive result of test, we can see that in most of the cases the minimum repetition value was increased above the TR value. In test instances, where the algorithm was not able increase the minimum repetition value was conditioned by two reasons. The first reason is that the value of AR is still small. The algorithm within a small AF does not have enough information to compute task mapping, which has the minimum repetition value higher than Threshold value. The second reason is that in the whole network doesn't exist feasible task mapping with the repetition value higher than Threshold. It can be seen that with the higher value of AR and DR, it is possible to reach the higher value of the minimum repetition value.



Graph 1: Test results number (1.-25.)



Graph 2: Test results number (26.-50.)

In Table 2 are represented dispersion values before, marked as *(dispB)*, and after *(dispA)* launching of the algorithm for defined values of AR and DR. It can be seen from the result values that the algorithm decreases dispersion of repetition values in AF and thereby it has balancing properties. However in some test instances the algorithm increase the result dispersion of repetitions. Increasing dispersion is caused by the prime purpose of the algorithm, which is repairing the failure state. It means, once the algorithm increase the minimum repetition value above threshold, then the algorithm is successfully terminated. After that termination the algorithm returns all nodes into Operating state, regardless on decreasing of dispersions repetitions on devices.

	AR=1,DR=1		AR=2,DR=1		AR=2,DR=2		AR=3,DR=1		AR=3,DR=2		AR=3,DR=3	
test	dispB	dispA	dispB	dispA	dispB	dispA	dispB	dispA	dispB	dispA	dispB	dispA
1	2,9	0,0	2,2	0,6	2,2	0,6	2,0	1,5	2,0	1,5	2,0	1,5
2	57,5	144,6	54,6	29,1	54,6	29,1	55,3	141,8	55,3	24,5	55,3	31,3
3	17,2	17,2	171,4	83,7	171,4	83,7	152,7	7,1	152,7	7,1	152,7	7,1
4	56,7	3,0	45,2	21,9	45,2	23,7	40,7	19,6	40,7	83,3	40,7	17,6
5	0,2	0,2	17,4	12,1	17,4	11,3	15,9	17,9	15,9	92,7	15,9	17,4
6	159,2	62,3	132,6	71,1	132,6	71,1	102,3	64,6	102,3	64,6	102,3	64,6
7	11,6	3,6	9,0	4,8	9,0	4,8	7,4	3,8	7,4	3,8	7,4	2,6
8	5,0	0,8	3,7	0,9	3,7	2,2	3,7	0,9	3,7	2,2	3,7	1,0
9	6,9	0,9	28,2	1,0	28,2	1,0	177,7	40,2	177,7	34,6	177,7	36,5

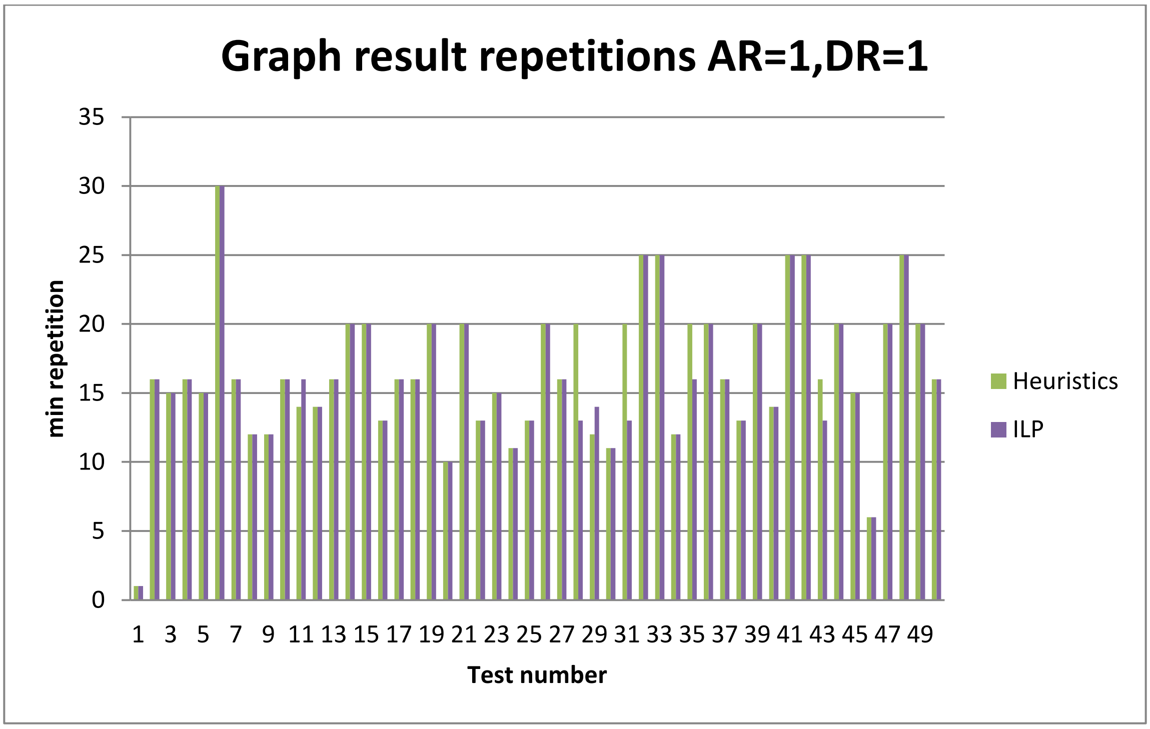
10	8,0	3,6	12,1	16,2	12,1	10,8	28,9	13,6	28,9	6,2	28,9	28,6
11	249,8	4,8	146,8	65,9	146,8	6,2	113,7	80,6	113,7	28,3	113,7	28,3
12	71,2	63,7	211,9	200,2	211,9	89,3	202,8	175,1	202,8	86,8	202,8	86,8
13	33,6	0,0	21,8	2,6	21,8	2,6	22,3	10,2	22,3	10,2	22,3	10,2
14	24,2	5,6	53,2	28,3	53,2	5,6	43,8	25,8	43,8	144,0	43,8	144,0
15	51,4	40,6	43,0	37,6	43,0	37,6	136,6	25,8	136,6	80,9	136,6	128,7
16	18,7	1,6	322,2	41,2	322,2	4,0	303,4	45,6	303,4	110,2	303,4	103,1
17	26,2	0,0	26,2	0,0	26,2	2,6	26,2	0,0	26,2	2,6	26,2	3,8
18	38,0	4,0	28,7	3,6	28,7	3,6	24,2	4,8	24,2	3,0	24,2	15,2
19	202,3	151,7	182,2	154,5	182,2	16,0	141,7	120,3	141,7	15,0	141,7	59,4
20	252,7	177,0	174,2	812,5	174,2	21,2	174,2	87,7	174,2	65,9	174,2	50,4
21	222,8	6,2	140,1	19,3	140,1	3,9	162,1	84,8	162,1	141,6	162,1	76,3
22	66,6	1,7	72,3	12,0	72,3	53,5	117,7	46,9	117,7	35,6	117,7	46,8
23	249,6	1,6	167,6	15,2	167,6	20,1	167,6	15,2	167,6	20,1	167,6	20,1
24	36,4	2,2	163,1	19,9	163,1	13,0	151,5	22,0	151,5	1,7	151,5	0,5
25	173,6	44,9	170,7	87,6	170,7	132,9	163,7	85,6	163,7	129,4	163,7	14,1
26	152,0	16,4	129,8	31,7	129,8	27,5	129,8	31,7	129,8	27,5	129,8	5,7
27	162,0	140,0	125,0	32,2	125,0	32,2	125,0	32,2	125,0	32,2	125,0	93,1
28	171,7	150,9	86,0	65,1	86,0	102,3	74,1	59,2	74,1	66,2	74,1	41,7
29	172,2	27,8	135,0	75,7	135,0	28,8	248,3	259,4	248,3	163,2	248,3	15,7
30	41,4	7,5	177,5	30,8	177,5	4,1	163,1	29,8	163,1	94,6	163,1	94,5
31	171,7	150,9	86,0	65,1	86,0	102,3	74,1	59,2	74,1	66,2	74,1	41,7
32	215,0	83,2	161,4	56,6	161,4	56,6	152,8	87,5	152,8	87,5	152,8	87,5
33	256,9	14,2	185,0	128,2	185,0	12,0	141,6	128,2	141,6	86,7	141,6	78,8
34	0,9	28,2	0,6	171,0	0,6	138,4	0,6	301,5	0,6	111,2	0,6	111,2
35	147,1	6,1	188,4	124,2	188,4	68,7	188,4	124,2	188,4	124,2	188,4	102,9
36	197,0	103,8	133,1	69,7	133,1	84,1	119,9	63,6	119,9	83,7	119,9	83,7
37	53,8	43,4	44,3	35,7	44,3	30,2	44,3	35,7	44,3	33,8	44,3	13,9
38	75,0	75,0	43,8	43,8	43,8	43,8	62,6	62,6	62,6	62,6	62,6	54,2
39	46,2	0,0	40,1	31,7	40,1	23,5	35,4	40,6	35,4	20,7	35,4	20,7
40	0,0	9,0	0,0	200,0	0,0	200,0	0,0	200,0	0,0	46,2	0,0	46,2
41	91,0	67,4	43,4	34,7	43,4	34,7	63,9	57,4	63,9	57,4	63,9	57,4
42	171,7	14,2	128,9	42,9	128,9	42,9	112,4	68,8	112,4	52,4	112,4	52,4
43	88,9	72,3	55,6	46,2	55,6	72,3	36,0	28,5	36,0	54,2	36,0	70,8
44	137,4	42,3	98,9	82,7	98,9	31,4	81,7	73,8	81,7	54,2	81,7	45,0
45	163,4	47,3	89,5	22,9	89,5	49,0	147,5	68,8	147,5	98,6	147,5	44,7
46	24,2	156,2	37,0	70,2	37,0	0,0	35,5	34,5	35,5	5,8	35,5	7,2
47	175,5	25,8	141,7	43,5	141,7	37,8	128,4	45,0	128,4	38,0	128,4	38,0
48	150,9	14,2	91,0	10,2	91,0	67,4	89,2	57,4	89,2	74,6	89,2	74,6
49	288,9	5,6	215,8	147,8	215,8	154,1	190,3	147,1	190,3	154,2	190,3	154,2
50	40,1	40,1	83,8	83,8	83,8	83,8	83,8	83,8	83,8	83,8	83,8	83,8

Table 2: Dispersions of before and after reparation

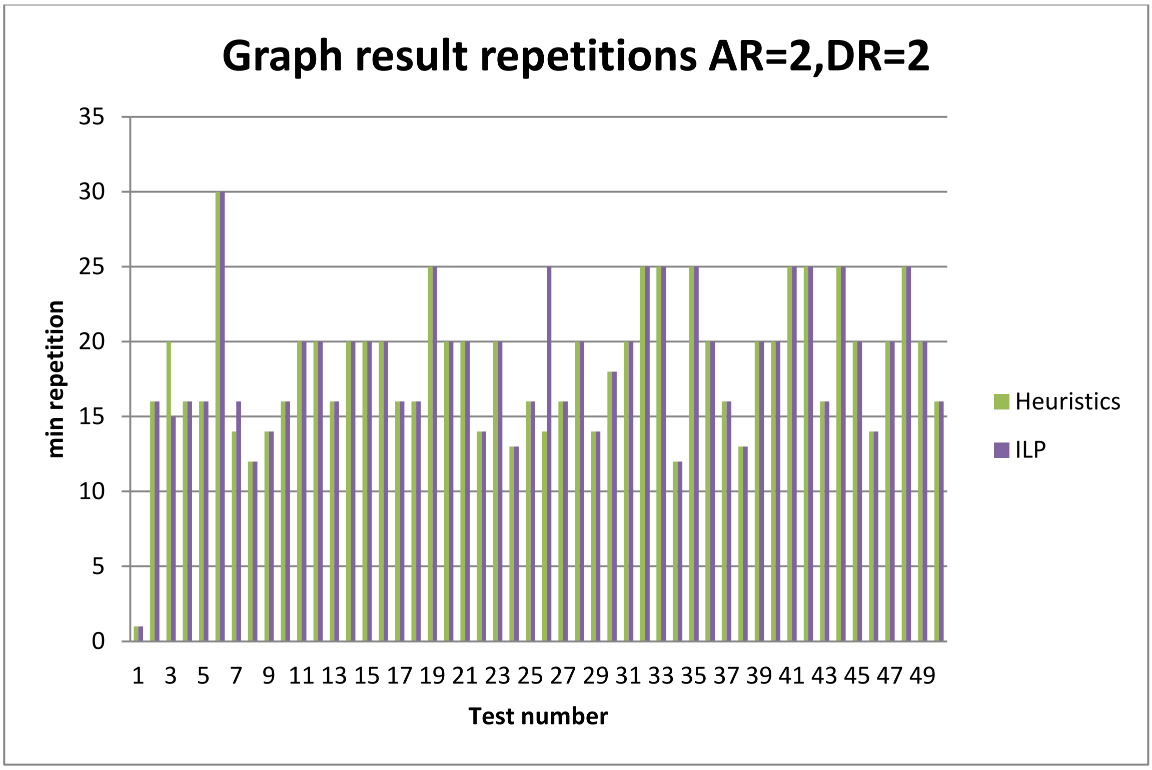
9.3. Quality comparison of the remapping heuristic

The distributed algorithm uses a heuristic for calculating a new task mapping on all nodes in the Alert Field. To be able to evaluate quality of that heuristics, we implemented an exact method based on Inter Linear Programming (ILP). It comes as surprise that the ILP method can't be implemented on embedded devices in practise. We choose the ILP method only for quality comparison. Because we know that the ILP method finds out the task mapping with the maximum repetition value, which is the optimal solution. In Graphs 3, 4 and 5, we can see the minimum repetition values after finishing of the algorithm.

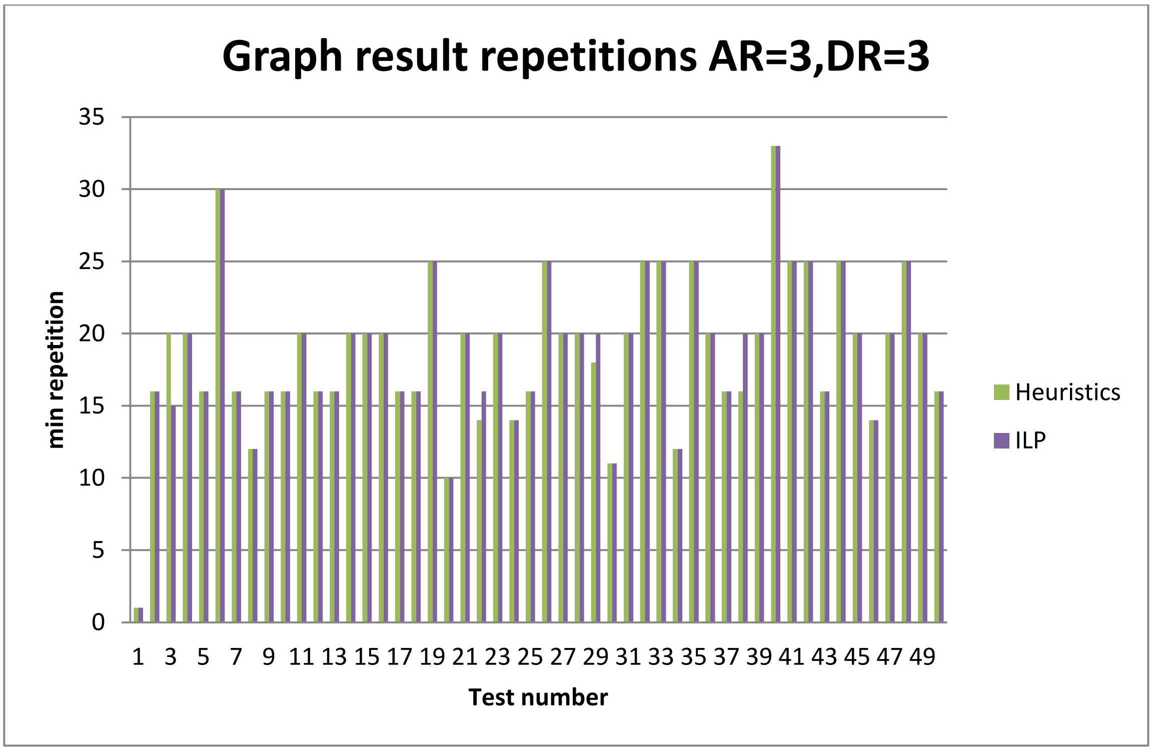
From results in Graphs 3, 4 and 5, we can notice that the heuristic algorithm reaches the same results as the ILP method. In some cases the heuristics returns better results after finishing algorithm. It is because the algorithm is distributed and devices make local decisions only. Another reason is that any node cannot see all connections in the architecture graph and thus it can't find the optimal task mapping.



Graph 3: Comparison of the heuristic and the ILP remapping algorithm for AR=1 and DR=1



Graph 4: Comparison of the heuristic and the ILP remapping algorithm for AR=2 and DR=2



Graph 5: Comparison of the heuristic and the ILP remapping algorithm for AR=3 and DR=3

10. Conclusion

Our prime aim was to propose and test a distributed algorithm, which repairs a failure states in the network by changing task mapping on nodes. The main criterion for the algorithm's design was to minimize amount of communication during its execution. Since the problem of task mapping is NP-hard, we had to develop a heuristic algorithm. The similar problem was not observed in the literature yet, how we pointed in related works in Chapter 3. We defined constrain that any two tasks, which have data dependency among them, can be mapped on the same device or on two neighbouring devices in the network. It ensures that task, which communicates with another one via messages, makes maximally one hop distance communication in the network. Unlike authors from **[SKHT06]** our approach saves a communication in the network. The authors in **[SKHT06]** enable to map these two tasks on devices that can be distanced across the whole network and the message needs multiple hops to reach the destination node. The distributed algorithm enables to increase lifetime and reliability in the wireless sensors networks. The heuristic used for local remapping of tasks is very efficient in comparison with an exact method based on ILP. Our heuristics provides similar results to the ILP method. The designed algorithm also illustrates a scheme that can be used for solving other related problems from wireless sensor networks domains.

However, the proposed algorithm can be still improved. The first improvement is based on the reducing communication during repeated iterations of the algorithm. Devices from information obtained in the first iteration, need not be involved into communication in some states of the algorithm. For instance, when the device in the second iteration obtains the same information in Discovery state as in the first iteration, then it need not launch the remapping algorithm in Remapping state in the second iteration. The second improvement can be applied on evaluating of the objective function and the final decision of the failure node. The failure node terminates the algorithm once it found out that all devices have feasible task mapping. Although the failure node finds out that all nodes have feasible mapping, it can decide to iterate algorithm again and obtain even a better mapping on devices.

Bibliography

[AFMH14] Alexandra Aguiar, Sergio Johann Filho, Felipe Magalhaes and Fabiano Hessel. On the design space exploration through the hellfire framework. *Journal of Systems Architecture*, 60(1):94-107, 2014.

[CG12] T.E. Carroll and D. Grosu. An incentive-based distributed mechanism for scheduling divisible loads in tree networks. *Journal of Parallel and Distributed Computing*, 72(3):389-401, 2012.

[Cyb89] George Cybenko. Dynamic load balancing for distributed memory multiprocessors. *Journal of Parallel and Distributed Computing*, 7(2):279-301, 1989.

[GV08] Lee Kee Goh and Bharadwaj Veeravalli. Design and performance evaluation of combined first-fit task allocation and migration strategies in mesh multiprocessor systems. *Parallel Computing*, 34(9):508-520, 2008.

[SC13] Pradip Kumar Sahu and Santanu Chattopadhyay. A survey on application mapping strategies for network-on-chip design. *Journal of Systems Architecture*, 59(1):60-76, 2013.

[SHT05] Thilo Streichert, Christian Haubelt and Jürgen Teich. Online hardware/software partitioning in networked embedded systems. In *Proceedings of the 2005 Asia and South Pacific Design Automation Conference*, ASP-DAC '05, pages 982-985, New York, NY, USA,2005. ACM.

[SKHT06] Thilo Streichert, Dirk Koch, Christian Haubelt, and Jürgen Teich. Modelling and design of fault tolerant and self-adaptive reconfigurable networked embedded systems. *EURASIP J. Embedded Syst.*, 2006(1):9-9, January 2006.

[SSKJ10] Amit Kumar Singh, Thambipillai Srikanthan, Akash Kumar and Wu Jigang. Communication-aware heuristics for run-time task mapping on noc-based {MPSoC} platforms. *Journal of Systems Architecture*, 56(7):242-255, 2010. Special Issue on HW/SW Co-Design: Systems and Networks on Chip.

[SFHP13] P. Sucha, J.Olivera de Filho, Z. Hanzalek and Z. Papp. Scheduling in Self-adaptive Reconfigurable Networked Embedded Systems. *Technical report CTU and TNO*, pages 1-4, 2013.

[KA99] Yu-Kwong Kwok and Ishfaq Ahmad. Benchmarking and comparison of the task graph scheduling algorithms. *Journal of Parallel and Distributed Computing*, 59(3):381-422, 1999.

[YHZEA09] Ying Yi, Wei Han, Xin Zhao, A.T. Erdogan, and T. Arslan. An ILP formulation for task mapping and scheduling on multi-core architectures. In *Design, Automation Test in Europe Conference Exhibition*, 2009. DATE '09., pages 33-38, 2009.

[SP12] J. Sijs and Z. Papp. Towards self-organizing kalman filters. In *Information Fusion* (FUSION), *2012 15th International Conference on*, pages 1012-1019, 2012.

[N12] P. Neelakantan. Load balancing in distributed systems using diffusion technique. *International Journal of Computer Applications*, 39(4):1-7, 2012.

[ASEP11] A. Aminifar, S. Samii, P. Eles, and Zebo Peng. Control-quality driven task mapping for distributed embedded control systems. In *Embedded and Real-Time Computing Systems and Applications* (RTCSA), 2011 IEEE 17th *International Conference on*, volume 1, pages 133-142, 2011.

[GCL02] D. Grosu, A.T. Chronopoulos, and Ming-Ying Leung. Load balancing in distributed systems: an approach using cooperative games. In *Parallel and Distributed Processing Symposium., Proceedings International, IPDPS 2002, Abstracts and CD-ROM*, pages 10 pp-, 2002.

[Dynaa] DynAA framework created by TNO company. URL: https://www.tno.nl/en/focus-area/industry/networked-information/embedded-systems-innovations-esi/dynaa-design-of-large-adaptive-networked-systems/

Content of the attached CD

1. text of thesis in PDF format
2. source code of distributed algorithm implemented in DynAA development tool