

A Top-Down Approach for Realtime Industrial-Ethernet Networks using Edge-Coloring of Conflict-Multigraphs

F. Dopatka, R. Wismüller

University of Siegen, Institute of Operating Systems and Distributed Systems, Hölderlinstr. 3, 57068 Siegen, Germany
{frank.dopatka, roland.wismueller}@uni-siegen.de

Abstract--This paper presents a top-down approach to design switched Ethernet networks based upon tree topologies for realtime automation systems. The approach considers each port of a switch as an exclusive networking resource and thus permits concurrent communication over different ports. We firstly focus on a class of special cases with unitary packet-sizes and transmission of each request in every production cycle-time. We point out that an off-line schedule can be generated independently for each switch with half-duplex connections. With the networking infrastructure and communication requirements as inputs, we present a scheduling method, which is based on the creation of a conflict-multigraph for each switch. The schedules are then realized by using a greedy edge-coloring. We also present first results of the coloring-process. Finally, we discuss various ways to implement the computed schedules with their assets and drawbacks.

Index Terms--Automation, Communication systems, Field buses, Real time systems

I. INTRODUCTION

Within the scope of the automation technology, a trend to substitute the established field buses with realtime Ethernet networks can already be observed for a long time [1] [2] [3]. The aim is to use a single network from office area to the field devices like sensors, actuators or user-panels.

However, in automation technology, it is indispensable to obtain a deterministic realtime behavior of the link protocol [4], which is missing in the common IEEE 802.3 standardization for CSMA/CD (Carrier Sense Multiple Access/Collision Detection) Ethernet networks [5].

Nowadays, determinism is mostly achieved by introducing a uniform time-slot protocol based on TDMA (Time Division Multiple Access). The purpose of TDMA is to avoid collisions on the Ethernet with the resulting non-deterministic backoff-strategy when hubs are used, as well as to avoid buffering inside switches, which violates realtime requirements like delay and jitter.

Automated production lines are typically based on one basic production cycle-time. According to this, each set of TDMA time-slots is executed periodically with this cycle-time or at multiples of the cycle-time. Because of this, it is unusual to define arbitrary cycle-times for every communication. In addition to this, the cyclic transmitted packets contain only several bytes in contrast to standard Ethernet traffic [6].

There already exist some solutions suited for industry, which are able to expand into hard realtime classes [7] defined by IAONA (Industrial Automation Open Networking Alliance) [8].

EPL (Ethernet PowerLink) [9] is a solution that works with standard hubs and one central managing node to realize deterministic access control. The managing node polls the controlled network devices, so that always only one device is transmitting at each time, with multicast option. Furthermore, a time period für non-realtime traffic exists in every cycle. However, EPL does not allow multiple independent transmissions at the same time, which is a severe drawback in modern, decentral automation systems.

SIEMENS ProfiNet [10] divides the communication cycle into a synchronization phase, an isochronous period, and an asynchronous period. It is based upon in-house SIEMENS realtime 4-port switch ASICs (Application Specific Integrated Circuits); standard Ethernet switches cannot be used. A line-structure can be built to save wiring costs, but each switching component increases the packet delay.

EtherCAT (Ethernet for Control Automation Technology) [11] is using standard Ethernet NICs (Network Interface Cards), while physical data is transmitted similar to a shifting register: A FMMU (Fieldbus Memory Management Unit) in each connector extracts the data for this device and injects the data to be transmitted while the telegram is shifted through the network. The problem here is a missing compatibility of data transmission with standard Ethernet.

Besides the introduced technologies, many other realtime Ethernet approaches arise, like SERCOS [12], Ethernet/IP [13] or JetSnc [14], which are working in a similar way.

Altogether, any approach has to find a compromise between compatibility to packet switched IEEE 802.3 standards with its minimal Ethernet packet size of 64Bytes on one hand and optimal cycle-time, throughput and speed on the other hand. In the area of standard ethernet, multiple independent full-duplex communications are allowed without collisions. Thus, the utilization of bandwidth is enhanced. On the other hand, store-and-forward and even cut-through switches increase transmission delays. The demands of automation technology consist of minimal cycle-time, small payload, and minimal delay and jitter [6]. Often, the compatibility to common Ethernet standards is violated in realtime

environments in order to reach this demands. An additional trend is well-founded in decentral periphery with intelligent sensors and actors and without a central controller. The two mainstream trends, Ethernet and automation technology, have to fit together in the context of vertical integration.

In our approach, we follow a top-down strategy with a given network infrastructure and communication requirements of the devices. These requirements are always known a priori in automation technology. The aim is to find and implement a method, which is able to generate off-line schedules for each switch. The ratio between automation speed and compatibility to Ethernet standards ought to be variable, depending on the requirements.

The basic preconditions of our approach are treated in the next section. In the third section, the view on this problem is narrowed down in order to find a solution for a special but important subclass, which is discussed with the help of an example. Afterwards, the focus of this paper is directed to the development of generally applicable methods for satisfying the requirements. It is shown, that these methods can be applied to each network switch individually. The fourth section discusses various ways to implement the generated schedules, together with their advantages and disadvantages. Finally, we summarize the algorithmic proceeding briefly and evaluate the success with perspectives for further research.

II. DEFINING BASIC PRECONDITIONS

We assume an Ethernet infrastructure with tree topology based on IEEE 802.3 [5], with uniform bandwidth, which is either given or whose minimum value has to be calculated. The leaves of the tree are the network devices, while inner nodes are either broadcasting hubs or switches. Hence, we use the term “distributor” for both hubs and switches. In order to save wiring costs, 3-port-distributors are able to emulate the often used line-topology in automation technology. We presume a sufficient synchronized time for each networking element, which could be based on the IEEE1588 [15] protocol.

Each communication requirement can be described by the source-address of the sender, the target-address(es) of the receiver and the amount of bytes, which have to be transferred. Unicast, multicast or broadcast addressing can be used. In order to store a limit value of a delay time, we add one parameter as the maximum delay between sending to network medium and receiving from network medium. In addition to the delay, we add a maximum jitter time to each communication requirement. The jitter characterizes the maximum allowed variance of the delay-time at the point of receiving data.

The network path used by a message can be represented by a CT (communication tree). A broadcast message creates for example a CT equal to the whole network infrastructure. A CT is characterized by the set of the ports of the distributors, which the packets are

passing. In general, two or more CTs can use common ports.

Because a switched network is free of collisions, the term “conflict” is introduced for every overlapping of CTs. In a switched network, a conflict requires that data packets are buffered in some switches, which increases delay and jitter and violates realtime requirements. In general, a conflict subsumes possible non-deterministic collisions inside a collision domain with distributing hubs as well as all situations where packets need to be buffered in a switched network.

The aim of our work is to define the CTs, identify all possible conflicts and to find a way to avoid them. The overlapping of two or more CTs in any part of the network can be solved by their temporal separation. This means that the trees have to be executed consecutively at least at all points, where they are in conflict with each other. This means that a schedule for the transmission time of each device has to be computed. The schedules result in the TDMA multiplex operation.

III. SOLUTION FOR UNICAST ADDRESSING AND UNIFORM PACKET SIZE

The solution for the preconditions we described in section II would be too complex to reach in one single step. So, we decided to define a reasonable restricted problem-class in order to simplify the solution finding.

In the solution we present in this paper, network delays and the effect of jitter are neglected. The set of distributors in the network consists exclusively of switches or hubs without delay and jitter. We assume that each switch with n ports is able to handle at least $n/2$ independent communications at the same time without buffering. We consider unicast addressing with half-duplex and outline the procedure for full-duplex connections. The unicast addressing reduces the CTs to one-dimensional CLs (communication lines). Broadcasting is feasible as well. It can be trivially taken into account, because a broadcast does not allow any other concurrent communication. Further on, we assume that there exist no causal dependencies between transmissions.

In addition to this, all packets are assumed to be sent exactly once in every basic production cycle-time and to be of uniform length. For perpetuation of compatibility with Ethernet frames according to IEEE802.3, its minimum packet length can be used, which contains 64 Byte of data with 46 Byte of payload in a 100 Mbit/s network. This payload is sufficient for requirements of automation technology. Although smaller packets could be used with switched Ethernet, the compatibility would be violated. In our approach, will work with the assumption of one general minimum packet length and one general available bandwidth.

A. Preparation of the Input Data

For our class of special cases specified above, the communication requirements consist only of source and destination address, the other parameters are not relevant. With the given infrastructure and communication

requirements, we can build the CLs representing the path from the source to the destination device in the tree. The calculation and storage of the CLs is realized by using an attribution technique similar to our presented method in [16]. The algorithm to calculate the CLs is presented in figure 1. It firstly detects the path from the source device d1 to a arbitrarily chosen root node r and from the destination device d2 to this root node as well. The path consists of sets of links (x and y) between the distributors. The CL can then be calculated by using simple list operations:

```

FindCL(NetworkTree T, Device d1, Device d2)
{
  Path p1 = T.root_path(d1); // (x/x1,x1/x2,...,xi/d1)
  Path p2 = T.root_path(d2); // (x/y1,y1/y2,...,yj/d2)
  while (p1.first() == p2.first())
  {
    p1.remove_first();
    p2.remove_first();
  }
  CL = new List();
  while not (p1.is_empty())
  {
    CL.add(p1.remove_last());
  }
  while not (p2.is_empty())
  {
    CL.add(p2.remove_first());
  }
  return CL;
}

```

Fig. 1. Algorithm for calculating a communication line.

After calculating the CLs, we know from the links, which distributor participates in which requirements. Thus, we are able to allocate the relevant requirements to each distributor. Figure 2 starts a continuous example in this paper and shows a given networking infrastructure with its communication lines, which result from the given requirements. It consists of 2 hubs (which emulate a line-topology with minimal delay), 4 switches, 17 devices, and 9 CLs. The CLs are inscribed at the links, whereas each equal number means the same CL going through the network. For example, CL2 goes from device 1 to device 7 over switch 1 and switch 2.

Hubs can easily be added into the switched network, because they simply repeat the incoming transmission to all ports except the incoming port. While one CL uses one port of a hub, no other communication lines are allowed to use another port of the same hub. This has to be considered at all switches connected with the hub. Cascaded hubs can be seen as one single hub, where no concurrent communication is allowed.

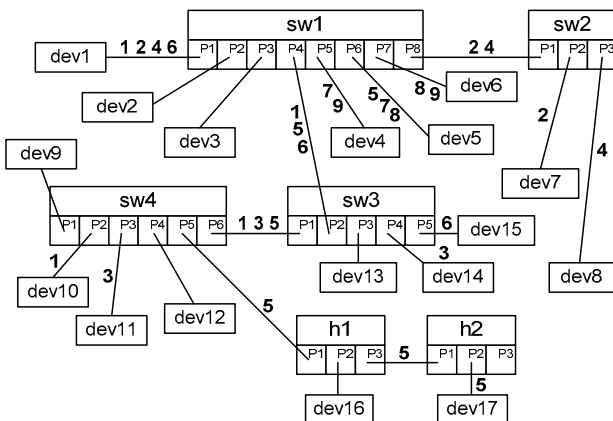


Fig. 2. Networking infrastructure and communication lines.

B. Building a Conflict-Multigraph

The question is, why a conflict-multigraph can be calculated independently for each switch in half-duplex mode. The answer is located in the tree-topology of the network. Because of this, two CLs meeting in a switch either have a conflict or they never have one, e.g. CL2 and CL7 in switch 1, see figure 2. They meet without a conflict, and they will never meet again in other switches caused by the chosen tree-topology. According to this, the corresponding communications can be executed concurrently. On the other hand, if two CLs have a conflict, the conflict raises in exactly one connected region in the network: An example for this is CL1 and CL5. They have a conflict in switch 1, port 4 upto switch 4, port 6. When they leave switch 4 using port 2 and port 5, they will not be able to meet again because of the tree structure of the network. The same explanation is valid for switch 1, port 1 and switch 1, port 6. As long as no causal dependencies are existing, the sequence of the resulting schedules for each switch can be chosen arbitrarily.

To avoid conflicts in a half-duplex connection, two CLs are not allowed to use the same port at the same time. So, each port of a switch can be seen as an exclusive resource, which can be used by at most one CL at any given time. Because all of the CLs are known a-priori, we are able to generate a conflict graph $G=(V, E)$ in order to execute graph-coloring algorithms and find an off-line schedule for each switch. Valerio, Moser and Melliar-Smith [17] as well as Marx [18] describe the solvability of a scheduling problem with methods of graph coloring, either by coloring vertices or edges.

In the beginning, we focus on edge-coloring of undirected multigraphs. “In graph theory, [...] an edge coloring of a graph [...] is always assumed to be a proper coloring on the edges, meaning no two adjacent edges are assigned the same color. Here, ‘adjacent’ means sharing a common vertex. [...] The smallest number of colors needed in a (proper) edge coloring of a graph G is the chromatic index, or edge chromatic number, $\chi'(G)$ ” [19]. In general, colors are numbered with integer values beginning with 1.

For these purposes, we transfer each port of the switch into a vertex, while each CL is allegorized as an edge of the graph. The result is a multigraph, because more than one CL can use the same source and target port. In a half-duplex-connection, every port can be used only exclusively. Therefore, an undirected multigraph results, see figure 3a, in which every edge at one vertex has to get a different color. If full-duplex-connections are allowed, we can separate the edges at each vertex into a set of incoming and a set of outgoing edges. The result is a directed multigraph, whereas the two sets of each vertex can be treated independently, see figure 3b. We assume for example, that CL2 is sent from device 1 to the receiving device 7. Therefore, the CL goes from port 1 of the first switch to port 8, as shown in figure 3b.

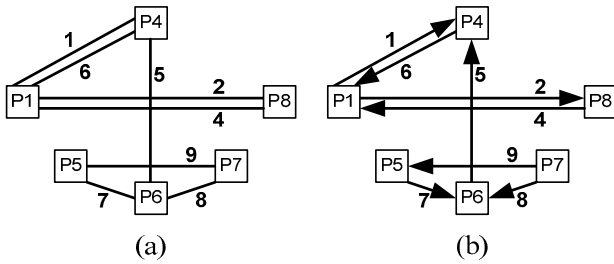


Fig. 3. Conflict-multigraph with half-duplex (a) and full-duplex (b) connections for switch 1.

C. Calculating the Schedules

A lower bound for the chromatic index of an undirected multigraph can be described with the theorem of Vizing as equal to the degree $\Delta(G)$ of the graph, which means the maximum numbers of adjacent edges at any node [20]. Furthermore, an upper bound is given by the degree plus the multiplicity d of the graph, so we can say $\Delta(G) \leq \chi'(G) \leq \Delta(G) + d$. “The multiplicity of a multigraph is the maximum number of edges between any specified two vertices” [21]. Even if $\chi'(G)$ is previously known, it is NP-hard to find an associated coloring. However, we can add the condition to find not always the optimal coloring, but a coloring with a number of colors close to $\chi'(G)$. The result would be in worst case a heuristically generated schedule with “some” more colors than the minimum amount of colors. Moreover, there exist numerous coloring heuristics, which approximate the chromatic index in different ways.

Feige, Ofek and Wieder [21] calculate a schedule for a satellite-based communication network with edge coloring with an own off-line greedy algorithm. The simplest greedy algorithm for multigraphs is a first-fit algorithm, which belongs by declaration of Davis and Impagliazzo [22] to the class of greedy-algorithms with invariant prioritization. A greedy-algorithm provides already a valid - but not always optimal - coloring in every case:

```

while (not all edges colored)
{
  x = any non-colored edge
  M = set of neighbor-edges from x
  color(x) = lowest color not present in M
}

```

Fig. 4. A simple greedy-algorithm.

Wanka [24] describes a more efficient algorithm of edge-coloring of a simple graph in $O(|V| \cdot |E|)$ time. In his work, he uses additional algorithms to re-color an already colored subgraph. “If G is edge-colored with the colors $\{1, \dots, \Delta(G) + 1\}$ and if u and v are vertices with $\{u, v\} \notin E$ and $\deg(u), \deg(v) < \Delta(G)$. Then it is possible to re-color G in a way that the same color is missing at u and v .” This statement can be transferred to multigraphs. Based on this idea, an improved greedy coloring can be formulated as follows: The algorithm starts with a graph devoid of edges and adds any edge. If $\Delta(G)$ does not change, G is re-colored in such a way, that the same color is absent at both ends of the inserted edge. If necessary,

the amount of used colors increases by one. The edge is then colored with this absent color. In the other case, if $\Delta(G)$ increases, the new edge will be colored with the lowest missing color at its two ends.

It can be shown for vertex coloring, that any greedy algorithm is able to find an optimal coloring, if the vertices are processed in a suitable optimal order [20]. This statement can be transferred to edge-coloring of multigraphs as well. It relocates the problem of finding an optimal coloring to a problem to find a beneficial sequence of the edges for initializing the greedy algorithm. An optimal coloring can be found in any case, when all permutations of the edges are examined. This would create a time-complexity of $O(n!)$ with n edges. We outlined some additional heuristics like smallest-last and saturation-largest-first in order to pre-sort the edges before greedy-coloring of the multigraph in [16].

D. Data-Modeling

In order to empirically evaluate different coloring algorithms, we implemented a Microsoft Visual Basic .NET framework. The network topology and the communication lines can be read in dynamically. Different pre-sorting coloring, edge-coloring and re-coloring algorithms can be chosen and configured. The first aim is the exploration of dependencies between execution speed of the greedy-algorithm from its input data.

A conflict-multigraph is generated with vertices representing ports and edges representing CLs. Now, any edge-coloring algorithm can be connected with the graph-class in order to generate a valid coloring. To model full-duplex transmission, two lists of colors are stored in each vertex. The first list contains the incoming transmissions and the second list the outgoing transmissions from this port. While one incoming and outgoing connection can use one port at the same time, two or more transmissions in the same direction at one time would raise a conflict. Thus, the edges of the two lists can be colored independently from each other. The testing environment uses the `Now.Ticks` method from operating system with μ s-granularity for time-measurement of the greedy-algorithm.

E. Coloring with Greedy-Algorithm

We implemented a greedy algorithm for edge-coloring of undirected and directed multigraphs for the class of restricted problems.

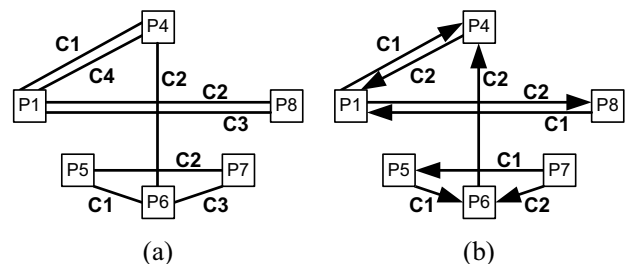


Fig. 5. Greedy-colored multigraph for switch 1.

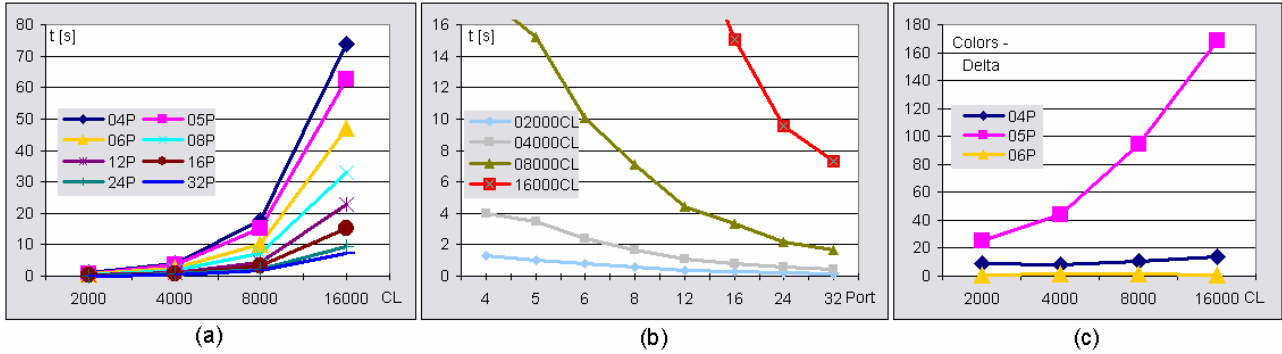


Fig. 6. Half-duplex measurement results in dependency of the amount of CL (a) and ports of the switches (b) as well as quality of the results (c)

To gain data of algorithm efficiency, we generated 32 switch-objects with 4 ports, 32 switch-objects with 5 ports and with 6, 8, 12, 16, 24 and 32 ports. We assigned 2000, 4000, 8000 and 16000 randomized communication lines to each set of objects. Each data point represents the average value of 8 measurements of every combination of port number and CLs. The time of generating an edge-coloring of undirected conflict-multigraphs has been measured with an Acer TravelMate 250 Laptop with Intel Pentium 4 Processor, 2.4GHz and 512MB RAM.

The implemented greedy-algorithm has a runtime complexity of $O(|E| \cdot \Delta(G))$ with n CLs, see figure 6a. In addition to this, it can be detected in figure 6b, that the calculating time is reduced with complexity of $O(1/m)$ with m ports and constant number of CLs. The reason for this is the density of edges at a vertex. The time-complexity of the greedy algorithm is determined in locating the next free color at both ends of an edge. Thus, algorithmic complexity raises with the density of edges inside the graph. You can see in figure 6b as well, that a greedy edge coloring of 4000 half-duplex CLs can be done very fast on a standard Laptop. It needs about 4 seconds using a 4-port switch and about 1 second with a 16-port switch. Besides, this number of independent communications at one switch is rather high for automation requirements. When considering full-duplex connections, the list of the edges connected at each vertex is splitted as described in section III/D into a set of incoming and outgoing edges. Therefore, the complexity decreases with bisection of the amount of CLs.

In addition to the speed of the algorithm, we have examined the quality of the results as well. We know from theorem of Vizing, that no graph can be edge-colored with less colors than its degree. So, we subtract the degree of the multigraph from the calculated number of colors. The result of the difference is the number of colors exceeding vizing's lower bound. Although this does not mean, that a graph can be colored with $\Delta(G)$ colors. However, the difference can be used as a quality indicator.

Figure 6c shows for even amount of ports, that the greedy algorithm needs constantly at most 10 colors more than the average degree over 8 measurements of the graph. For example, when considering the graphs for a 4-port switch with 8000 CLs, the outcome is 5102 as the average degree of the graphs. This means that we found a result less than 0.2% ($10/5102=0.002$) away from the

optimum solution, because the number of colors is equal to the required number of time-slots in the schedule for the switches. It is not necessary to check the proximity to the upper bound $\Delta(G)+d$, because our results are all located near to the lower bound. The average multiplicity d over 8 measurements for the 4-port switch with 8000 CLs is 2663, which leads to the upper bound of $5102+2663=7765$ colors. The results from switches with more than 6 ports are not displayed in figure 6c, because they are all located nearby 0.

Figure 6c also shows, that the results for 5-port switches differ from the other results. Here, the amount of used colors increases linear with the number of CLs. Here, we need for example 95 colors more than the lower bound with 8000CLs. The average degree of the randomized graphs is measured with 3948, so that we need about 2.4% ($95/3948=0.024$) more colors than the theoretical lower bound are needed. This is caused by specific properties of graphs with an odd number of vertices. For example, circles with an odd number of participating vertices larger than 3 need more colors than their degree. In addition, we have to consider that a switch with 5 ports can only handle at most 2 communications at the same time, which is the same as a 4-port switch. Unfortunately, this effect is not limited to 5-port switches. If in an 8-port switch only 7 or 5 ports are used for communication, a similar conflict-multigraph is generated almost with an odd number of vertices. This effect is not visible in the testing environment with even numbers of ports because of the randomized CLs, which on the average allocate the CLs equally to all ports.

In summary it can be said, that the greedy approach is fast and efficient in most cases. The time-complexity is even better using full-duplex transmission with directed multigraphs.

F. Calculating the Schedules

If the colored multigraph is available, the schedule can be generated by allocating the CLs to their colors. CLs with the same color can be executed at the same time. With uniform packet-size, the cycle-time results from the number of colors multiplied with the time needed to send a packet. This is approximately equal to the packet size divided by the bandwidth of the network. Figure 7 shows independently calculated schedules for all switches with their colors and respectively time-slots as well as the communication lines:

switch 1			
C1	C2	C3	C4
CL1	CL2	CL4	CL6
CL7	CL5	CL8	
	CL9		

switch 3		
C1	C2	C3
CL3	CL5	CL1
CL6		

switch 2	
C1	C2
CL4	CL2

switch 4		
C1	C2	C3
CL5	CL3	CL1

Fig. 7. Generated half-duplex schedules from the colored graphs.

G. Synchronizing the Schedules

The execution sequence of CLs with same color has to be synchronized with the schedules of the other switches. We show in this section, that the synchronization can be done easily in half-duplex mode by swapping the execution of whole time-slots among each other. This means, that is not necessary to swap individual CLs, which reduces the time-complexity of the synchronization. The reason for this is, that a conflict raises in exactly one connected region in the network, see section III/B. The tree-topology of the network prevents circular dependencies between the time-slots. So, the synchronization can be done in any linear order as long as no casual dependencies exist between the CLs. We phrased the following algorithm for half-duplex connections:

1. Find the switch with most colors and time-slots, respectively. This switch is called root r . If more than one switch exists with the same amount of colors, choose a switch arbitrarily.
2. The schedule of the root-switch $s(r)$ is final. Let us assume a second switch sw , which is connected with r . This can occur only over exactly one link. The CLs running through this link cannot be executed in a shared time-slot, neither in r , nor in sw .
3. Order the time-slots in sw with CLs that also occur in r as well in such a way, that these CL are executed in the same slot as in r . It is possible to get time-slots without any communication. The number of slots is the same as in the root node.
4. If there are any CLs left in sw , they do not belong to the set of CLs connected with r . They can be assigned to any free slots. It is impossible, that the total number of slots increases.
5. We define sw as new root for the subtree and continue recursively with 2., until all switches are processed. The same procedure is executed for all other subtrees of the root r .

Figure 8 shows the synchronized schedules for all switches of our example. We can see, that the schedule for switch 1 is still the same as in figure 7:

switch 1			
C1	C2	C3	C4
CL1	CL2	CL4	CL6
CL7	CL5	CL8	
	CL9		

switch 3			
C1	C2	C3	C4
CL1	CL5	X	CL3
			CL6

switch 2			
C1	C2	C3	C4
X	CL2	CL4	X

switch 4			
C1	C2	C3	C4
CL1	CL5	X	CL3

Fig. 8. Synchronized half-duplex schedules.

Note that the unused time-slots can be used to transport non-realtime traffic, as explained in section IV.

In case of full-duplex connections, we observed that the synchronization is more difficult. The cause for this is, that a link is not exclusively reserved for exactly one CL at one time any more. If the schedules are calculated independently, we can get a multitude of CL-permutations in the time-slots of different schedules. So, time-slots can not be exchanged as a whole any more.

As a first approach to solve this problem, we examine to color the root-switch and propagate the depending colors of the CLs to the subtrees of the root. This simplifies the following greedy-coloring and combines the coloring with the synchronization. However, the execution of pre-sorting algorithms becomes more complicated or not as efficient as before.

IV. WAYS OF IMPLEMENTING THE SCHEDULE

Due to the fact that non-realtime traffic with CSMA/CD and exponential-backoff strategy is processed in the given infrastructure by default, we have to consider, where and how the generated schedules are enforced. This decision determines the feasibility of non-realtime traffic together with realtime traffic. Further on, it gains influence on the compliance with compatibility to Ethernet-standards, on the number of used time-slots, and hardware costs.

We found four reasonable ways to implement the schedules into the given infrastructure and we will show, that the calculation of the schedules depend on their way of implementation after their generation. So, the way of implementing has to be considered before a conflict multigraph can be built.

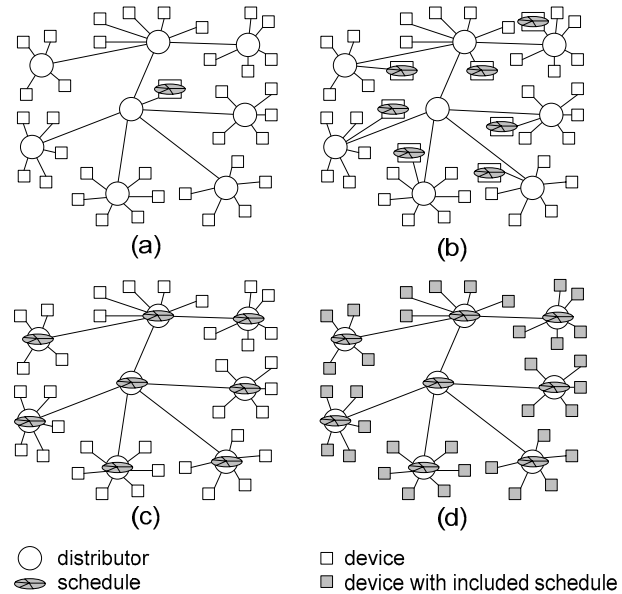


Fig. 9. Four ways of implementing the schedules.

The first way is the usage of one central arbiter as an additional device with a central schedule, shown in figure 9a. It polls the devices with sending grants, so that they are able to transmit a packet to one or more receivers. This approach is comparable to EPL, if the arbiter sends

unicast grants. The polling request has to be represented as a separate CL. In order to minimize the delay time to the devices and because no concurrent communication is possible, it is recommended to use only hubs as distributors. Only the arbiter is different from common low-priced Ethernet-hardware, but each device has to handle the polling protocol. If any device starts to send without permission, each hub will repeat the transmission to its output ports and destroy the realtime capabilities of the whole network.

As an improvement to EPL, concurrent communication can be permitted, if the central arbiter sends multicast grants to sending devices. This is possible, if the senders communicate to receiving devices without raising any conflict in the network. For this purpose, switching hardware is needed instead of hubs. Unfortunately, you have to consider additional switching-delays.

Concurrent network traffic can be enabled as well with one arbiter for each switch with connected devices (see figure 9b), which would be more expensive in contrast to the first solution. Using this technology, each arbiter holds an individual schedule for its switch. However, the arbiters have to be synchronized to each other. An additional advantage is the reduction of delay-time, because only the arbiters of the switches next to the sending devices have to send the polling requests. The time needed for the requests has to be considered in the schedules. On the one hand, both ways of using arbiters do not intervene into common Ethernet hardware, but they also do not allow to add non-realtime components dynamically.

The third way of implementing the schedules is illustrated in figure 9c and means their embedding into the switches themselves. The outcome of this is - as a disadvantage of this solution - the modification of switching software and possibly switching hardware. The switches could send special polling signals or short Ethernet packets to their connected realtime-devices directly, without passing two ports of the switch. Just after sending, the switches can interconnect their receiving and transmitting port. Thus, it is not required to inspect the target Ethernet address of incoming packets any more. Inferential, the switching delay-times could be even reduced in comparison to cut-through switches. If any traffic arrives on ports without being polled before, it has to be non-realtime traffic. This traffic can be stored in the switching buffers and forwarded during time-slots, where the destination port of the switch is not busy with realtime-traffic. This can be done in every first and fourth time-slot of switch 2, see figure 8, for example. The MTU (Maximum Transfer Unit) has to be equal to the defined uniform packet-size in this first approach. In order to permit non-realtime traffic at the switch with most colors - this is switch 1 in our example - as well in any case, we have to add at least one additional time-slot at each schedule. The ratio between the two types of traffic can be set at the run-up phase of the automated facilities by adding additional slots.

The polling requests can be omitted completely, if the devices get to know the time-slots when they are allowed to send, see figure 9d. In this case, the switching hardware just needs to interconnect the proper ports in order to forward the messages. However, this way assumes a synchronization between all distributors and all devices. As a disadvantage it can be said, that the devices have to possess active sending and synchronizing capabilities without being polled. As a result of this, the unit price - for example for simple sensors - would raise. On the other hand, this approach allows fastest realtime capabilities and the network will be able to handle non-realtime traffic as well by buffering in switches.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we firstly motivated the trend of using Ethernet in realtime-environments and outlined its difficulties. After discussing the problems of compatibility versus minimal cycle-time, we decided to pursue a top-down strategy independent of existing technologies with a given tree-based network infrastructure and requirements of the devices. We have shown, that our method can solve a class of restricted problems with unicast addressing, uniform packet-size, half-duplex and outlined full-duplex transmission, ideal distributors and devices without delay or jitter. Our method firstly consists of generating CLs from the input data, which is the networking infrastructure and the communication requirements. We have shown, that the further process can be executed independently for each switch in half-duplex mode. In dependency of the way of implementing the schedule, the conflict-multigraph is built for edge-coloring, which is undirected for half-duplex and directed for full-duplex transmissions. Then, the greedy-coloring of the multigraph is executed. In the last step, the schedule is generated for each switch based upon the coloring. The number of colors multiplied with inverse network bandwidth and the amount of bytes in one packet will result the cycle-time for this switch.

We implemented a greedy-algorithm and measured the time to compute the coloring. With a standard-PC, the coloring was accomplished very fast upto 4000 CLs at one switch, which are more then typically used in automation environment. A good quality of the greedy approach has emerged (0.2% inferior to theoretically optimal results) as long as no odd number of ports is used in communication. Otherwise, the results are acceptable with about 2% more colors used then needed in best-case. Further on, the whole process of coloring will be fast enough even if an additional optimization with smallest-last or saturation-largest-first heuristics is used.

After this, we described a way to get the schedules from the colored multigraphs and an algorithm to synchronize the schedules. This algorithm works for half-duplex connections. Finally, we discussed four ways of implementing the schedules with their assets and drawbacks. Summarized, the process of coloring and implementing can be expressed by using the following steps:

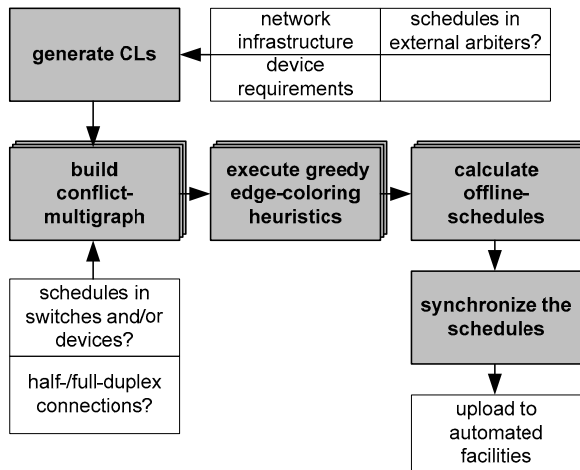


Fig. 10. Process of coloring and implementing.

The future scientific work will handle various aspects. Firstly, we finish to examine the solution for full-duplex connections with the objective of finding a solution similar to the half-duplex solution in this paper.

Further on, the usage of switches with more than 32 ports is unusual and because of this, the generated graphs consist of a small amount of vertices. In realtime environments, upto several hundreds CLs are usual, which means $|E| \gg |V|$. It has to be discussed, whether this property of the graph can be used for further optimization of the algorithms. We will also add more constraints like device and distributor jitter as well as delay times. The ambition is to model realistic switch behaviour with usual requests in order to generate feasible schedules. Another interesting constraint is the transmission of multiple values of one standard packet-size, like 46/92/184/... Bytes of payload. The result would be a variable length of time-slots in the schedule, which could be optimized by implementation of additional heuristics.

With our approach, we expect the realization of adaptable industrial Ethernet networks, which can balance between compatibility and throughput. It supports the usage of polling devices, arbiters or managing nodes like EPL as well as a realization in the switches themselves. The schedules can be generated off-line depending on the individual configuration of automated production lines.

REFERENCES

- [1] Rockwell Automation: *Vertikale Integration: Mit einem einzigen Feldbusprotokoll vom Internet zum Sensor* (in german), <http://www.rockwellautomation.de/applications/gs/emea/gsde.nsf/pages/Vertikale_Integration>.
- [2] Schnell, G.: *Bussysteme in der Automatisierungs- und Prozesstechnik* (in german), 5th revised and extended edition, Braunschweig, Vieweg, 2003.
- [3] Tschimpke, D.: *Dezentrale Prozess-Automatisierung und vertikale Integration - In die Zukunft mit Plug & Play* (in german), elektro Automation 4/2001, 2001, pp.74-75, <<http://www.pma-online.de/de/fachau/a8996eaz.pdf>>.
- [4] Gould, L.: *Industrial Ethernet: Wiring the Enterprise*, in: Automotive Design and Production, Field Guide to Automotive Technology, <<http://www.autofieldguide.com/articles/080307.html>>.
- [5] Institute of Electric and Electronic Engineers: *IEEE 802.3 - LAN/MAN CSMA/CD Access Method*, 2005, <<http://standards.ieee.org/getieee802/802.3.htm>>.
- [6] Flüs, O.; et al.: *Ethernet in Industrie-Umgebungen* (in german), extract from comconsult-research, 2004, <www.comconsult-research.de/bilder/EIU-Probe.pdf>.
- [7] The online Industrial-Ethernet Book: *News from IAONA*, <<http://ethernet.industrial-networking.com/news/newsdisplay.asp?id=87>>.
- [8] Industrial Automation Open Networking Alliance: <<http://www.iaona.ch/>>.
- [9] Ethernet PowerLink: <<http://www.ethernet-powerlink.com/>>.
- [10] SIEMENS ProfiNet: <<http://www.profinet.com/pn/>>.
- [11] EtherCAT: <<http://www.ethercat.org/>>.
- [12] SERCOS: <<http://www.sercos.de/>>.
- [13] Ethernet/IP: <<http://www.ethernet-ip.org/>>.
- [14] Jetter AG: *Motion and Control Systems Unified*, <<http://www.jetter.de/68.0.html?&L=1>>.
- [15] National Institute of Standards and Technology: *IEEE 1588TM-2002 Standard for A Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*, 2005, <<http://iee1588.nist.gov/>>.
- [16] Dopatka, F.; Wismüller, R.: *Achieving Realtime Capabilities in Ethernet Networks by Edge-Coloring of Communication Conflict-Multigraphs*, in: Parallel and Distributed Computing and Networks, Proc. of the International IASTED Conference PDCN 2006, Innsbruck, Austria, Acta Press, Zurich, 2006, pp. 180-185.
- [17] Valerio, L.; Moser E.; Melliar-Smith P. M.: *Avoiding memory contention on tightly coupled multiprocessors*. Proc. 1st International Conference on Massively Parallel Computing Systems, Ischia, Italy, 1994, pp. 643-654.
- [18] Marx, D.: *Graph Coloring Problems and their Applications in Scheduling*, John von Neumann PhD students Conference, Budapest University of Technology and Economics, 2003, <<http://citeseer.ist.psu.edu/672702.html>>.
- [19] Wikipedia: *Edge coloring*, free encyclopedia, <http://en.wikipedia.org/wiki/Edge_coloring>.
- [20] Emden-Weinert T.; et al.: *Einführung in Graphen und Algorithmen* (in german), lecture notes, Humboldt University of Berlin, 1996, <<http://www.informatik.hu-berlin.de/Institut/struktur/algorithmen/ga/>>.
- [21] Feige U.; Ofek E.; Wieder U.: *Approximating Maximum Edge Coloring in Multigraphs*, Lecture Notes In Computer Science, Vol. 2462, Proc. 5th International Workshop on Approximation Algorithms for Combinatorial Optimization, Roma, Italy, 2002, pp. 108-121.
- [22] Davis S.; Impagliazzo R.: *Models of greedy Algorithms for Graph Problems*, in: Proc. 5th ACM-SIAM symposium on Discrete algorithms, New Orleans, Louisiana, 2004, pp. 381-390.
- [23] Wanka R.: *Approximationsalgorithmen* (in german), lecture notes, version 2.1, University of Paderborn, <<http://wwwcs.uni-paderborn.de/cs/ag-madh/vorl/Approx02/Skript/Skript.ps>>.