ACHIEVING REALTIME CAPABILITIES IN ETHERNET NETWORKS
BY EDGE-COLORING OF COMMUNICATION CONFLICT-MULTIGRAPHS

Frank Dopatka, Roland Wismüller
Institute of Operating Systems and Distributed Systems
University of Siegen, Department 12, Hölderlinstr. 3
Siegen, Germany
{frank.dopatka, roland.wismueller}@uni-siegen.de

ABSTRACT
This paper presents a top-down Quality-of-Service (QoS) approach to design realtime Ethernet networks for automation systems based upon tree topologies. Therefore, the parallelism of switched unicast communication is utilized by generating an off-line schedule, which considers each port of a switch as an exclusive networking resource. We point out that the schedule can be generated independently for each switch. Starting from a given network infrastructure and QoS-requests, we present a method consisting of attribution to identify routes, specification of conflicts inside the switches, and building of a conflict-multigraph for each switch. The schedule is realized using greedy edge-coloring of the graph together with optional edge pre-sorting and re-coloring after execution of the greedy-algorithm. The method is shown for a class of special cases with unitary packet-sizes and transmission of every request in each cycle-period. Finally, some first results of this method are presented.

KEY WORDS
Industrial automation, graph coloring, greedy, and QoS

1. Introduction
Within the scope of the automation technology, a trend towards vertical integration, which attempts to substitute the established field busses like Profibus-DP, Interbus, CAN or AS-i with realtime Ethernet networks can already be observed for a long time [1] [2]. However, in automation technology, it is indispensable to obtain a deterministic behavior of the link protocol, which is missing in the common IEEE 802.3 standardization for CSMA/CD-Ethernet networks [3]. Nowadays, determinism is mostly achieved by introducing a uniform timeslot protocol (TDMA) for all network devices, like switches, sensors, actors, user-panels or PLCs (Programmable Logic Controllers). 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 store-and-forward buffering inside switches, which would increase delay and jitter. There already exist some solutions suited for industry, which are able to expand into hard realtime classes defined by Industrial Automation Open Networking Alliance (IAONA) [4]. However, these solutions do not utilize well-known communication requests or the possibility of multiple independent full-duplex connections without collisions like switched home-office Ethernet. Often, the compatibility to common Ethernet standards is violated in realtime environments.

EPL (Ethernet PowerLink) is one of the popular solutions, working with standard hubs and one central managing node to realize deterministic access control. The managing node polls the controlled nodes (network devices) in the realtime isochronous part of the cycle, so that always only one device is transmitting at each time with a multicast option. Furthermore, a time period for non-realtime traffic exists in every cycle. This architecture allows a minimal cycle-time of 200µs with a jitter < 1µs. However, EPL does not allow multiple transmission at the same time, which is advantageous for modern, decentral automation systems [5].

SIEMENS ProfiNet divides the communication cycle into a synchronization, an isochronous period, and an asynchronous period as well as many other technologies. It is based upon in-house SIEMENS realtime 4-port switch ASICs, standard Ethernet switches cannot be used. A cycle-time of 500µs can be reached with a jitter < 1µs in ProfiNet V3.0. A line-structure can be built to save wiring costs, but each switching component raises the packet delay in magnitude of some µs [6].

EtherCAT (Ethernet for Control Automation Technology) is using standard Ethernet NICs (Network Interface Cards), while physically 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. With a line-, tree- or star-topology, a very fast cycle-time of 30µs can be accomplished with a jitter of some ns [7]. But compatibility of network data transmission with standard Ethernet is missing. Altogether, any approach has to 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 home-office, switched tree networks are established and will overcome the border of GBit-LAN soon. They allow multiple independent full-duplex connections without collisions and enhance the exploitation of bandwidth. But store-and-forward and even cut-through switches raise the transmission delay. The demands of automation technology consist of minimal cycle-time, less payload and minimal delay and jitter. An additional trend is well-founded in decentral periphery with intelligent sensors and actors and without a central controller. These two mainstream trends have to fit together in the context of vertical integration.

We follow a top-down strategy with a given network infrastructure and requirements of the devices (QoS-requests) for this infrastructure. These requests are well-known in automation technology and have to be satisfied. The aim is to find a method, which is able to generate offline schedules for each switch independently of existing technologies and protocols. Half-duplex and full-duplex connections should be supported as well as the existence of a minimal packet-size. The common framework of the network and the definition of QoS-requests are treated in the next section. Afterwards, the view on this problem is narrowed down in order to find a solution for an important subclass, which is discussed with the help of an example. In the third section, the focus of this paper is directed on the development of generally applicable methods for satisfying QoS requests. It is shown that these methods can be applied on each network switch individually. The fourth chapter summarizes the algorithmic proceeding briefly and evaluates the success with perspectives for further research.

2. Common Framework and QoS-Definition

We assume an Ethernet infrastructure with tree topology, based on IEEE 802.3 [3] with uniform bandwidth, which is either given or which minimum size has to be calculated to satisfy QoS. The leaves of the tree are the network devices, while inner nodes are either broadcasting hubs or collision-free switches. For brevity, we use the term “distributor” for both hubs and switches in the rest of this paper. The edges of the network tree are formed by link cables, which connect exactly two nodes. The ports of each distributor are numbered. In order to save wiring costs, 3-port-distributors are able to emulate the often used line-topology in automation technology. Since no redundant links are considered in this work, no routing problem needs to be solved. We presume a sufficient synchronized time for each networking element, e.g., based on the IEEE1588 [8] protocol. Every communication request can be described as a data structure, which contains the source-address of the sender AddrS, the target-address(es) of the receiver (AddrD) with unicast, multicast or broadcast addressing and the cycle-time T. It is unusual in automation technology to define arbitrary time cycles for every request. Automated production lines are typically based on a basic production clock cycle t. Hence, every request is sent periodically with a period of t or at multiples of t. In order to store limit values of delay and jitter, we add Td as the maximum delay between sending to network and receiving from network medium and Tj as the maximum jitter from receiving Td. Tj characterizes the maximum allowed variance of Td.

In this paper, we assume that there exist no causal dependencies between the requests. Every message is sent by one transmitter and travels through a set of network distributors to one or several receivers. Thus, the network paths used by a message can be represented by a communication tree (CT). For example, a broadcast message creates 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. If the port sets of two CTs do not overlap, the corresponding communications can be executed concurrently. Otherwise, the common port numbers of the overlapping trees superpose in exactly one sequence because of the tree topology of the network infrastructure without alternative routes. On the other hand, if two non-overlapping trees co-exist in one network switch, they are not able to overlap at any other point in the network. 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 both delay and jitter. Thus, this situation has to be avoided. In general, a conflict subsumes possible collisions inside a collision domain with distributing hubs as well as all situations where packets need to be buffered in a switched network. Conflicts caused by switch ports shared between different CTs can be avoided by consecutively execution, i.e., by computing a schedule for the transmission time of each device, as well as the transmission and reception times of every port of a switch.

![Figure 1: Network infrastructure with two CTs](image-url)
Based on the tree topology of the network without alternate routes and the existence of CTs upon this structure, the schedule for each switch can be calculated independently. The schedule has to ascertain, whether trees can be executed at the same time or not. Figure 1 shows a sample network with two trees: CT1 from device dev_1 to devices dev_3, dev_8, dev_9, and CT2 from dev_2 to dev_12, dev_13, having a conflict at switch_1/port_4, switch_2/port_1, and port_2, as well as switch_3/port_6.

In the example, neither the schedule of switch_1, nor of switch_3 or switch_4 will allow a concurrent execution of CT1 and CT2. And by reason of the given tree topology, these trees will not appear together in any other switch. The sequence of execution — e.g., first CT1 and then CT2 — can be fixed after all schedules are calculated. As long as no causal dependencies are existing, the sequence can be determined arbitrarily.

3. Solution for a Restricted Problem-Class

In the following solution, network delays and the effect of jitter are neglected. The set of distributors in the network consists exclusively of switches or hubs without delay, this means \( T_d = 0 \) and \( T_j = \infty \). We assume that each switch with \( n \) ports is able to handle (at least) \( n/2 \) communications at the same time without buffering. Besides, only unicast addressing is considered. This reduces the CTs to one-dimensional communication lines (CL). Broadcasting is feasible as well. It can be trivially taken into account, because a broadcast does not allow any other concurrent communication. Further on, all packets are assumed to be of uniform length and to be sent exactly once in every basic cycle-time.

The assumption of equal packet sizes is motivated as follows: The automation pyramid [1] shows that the payload in the realtime area is small, it just averages a few bytes. Thus, 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. Although smaller packets could be used with Ethernet in some circumstances, our approach will abstract from an existing architecture and act on the assumption of a general available bandwidth and a general minimum packet length. Furthermore, we consider half-duplex and full-duplex connections. In a half-duplex connection, only one message can exist on one cable at any time. In case of injecting messages at both sides of a cable, a collision signal will appear. A switched network separates collision domains and works in full-duplex mode. This means, that packets can be sent and received through one cable at the same time without disturbance or delay.

3.1 Preparation of the Input Data

The solution for the class of special cases specified above is outlined with the help of an example out of this class. We have developed an algorithm which takes the network structure and QoS-requests (i.e., communication requests) as an input and then determines a valid schedule for each switch. For our example, the network infrastructure presented in figure 1 is assumed, whose devices enquire QoS-requests, consisting only of source and destination address. The other parameters are not relevant in the considered problem subclass.

<table>
<thead>
<tr>
<th>ReqNo</th>
<th>AddrS</th>
<th>AddrD</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>5</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

Table 1: QoS-requests of the devices

With the given infrastructure and requests, we can build the CLs representing the path from the source to the destination device. In the first step, one arbitrary distributor is chosen as a root node, and for all other distributors, the uplink port towards this root is identified. This is trivial because of the given tree structure of the network. For each device, the connection path to the root node ends with “0” for the output port. With these attributes, a unicast CL can be computed by concatenating the path from source device to root node with the reversed path from destination device to root node, after the (longest) common prefix of both paths has been deleted. By considering the root node itself as a part of the communication, like e.g. in the third request of table 1, the related line CL_3 can be calculated as follows:

\[
\text{path}_2 = ((2, 1, 0))
\]

\[
\text{path}_{13} = ((3, 4, 6), (2, 3, 1), (4, 1, 0))
\]

\[
\text{CL}_3 = \text{path}_2 \oplus \text{path}_{13}
\]

Beginning with the source device, the concatenation results in adding the target distributors with swapping the input and output ports, because the connection is now directed from the root note to the destination device. The “0”, which marks the end of the connection to the root, is canceled as well as one of the two root node numbers. In the second case, the root note itself is not part of the communication, e.g. in the eighth request of table 1. Then, not only the root note (x,1,0) appears in both paths, but a whole part of the path to the root. In this case, exactly one distributor has to appear in both paths as a result of the network tree structure. The CL is then composed from the source device in direction to the common distributor. The distributors of the destination device are added again in reversed order. In this case, it has to be started with the output port of the common distributor:
path₁₀ = ((5, 3, 1), (4, 1, 0))
CL₈ = path₁₀ ⊕ path₁₀
= ((1, 4, 6), (2, 3, 1), (4, 1, 0)) ⊕ ((5, 3, 1), (4, 1, 0))
= ((1, 4, 6), (2, 3, 5))

As explained in the second section, the schedule for each switch can now be calculated independently. We know after calculating the CLs from the network infrastructure and the QoS-requests, which distributor participates in which request. Thus, we are able to allocate the relevant requests to each switch. We continue our example with the calculation of a schedule for switch 1:

Figure 2: Switch 1 and its communication lines

3.2 Building a Conflict-Multigraph

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 of the existence of many previous known CLs, we are able to generate a conflict graph in order to find an off-line schedule for this switch. Valero, Moser and Melliar-Smith [9] describe as well as Marx [10] the solvability of a scheduling problem with methods of graph coloring, either by coloring vertices or edges.

Now, we have to transfer the switch with its ports and CLs into a conflict-graph G = (V, E) in order to execute graph-coloring algorithms in a way, that the coloring will represent the schedule for this switch. 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 consist of the same source and target port. In a half-duplex-connection, every port can be used only exclusive. Therefore, a undirected multigraph results, in which every edge at one vertex has to get a different color. By considering also full-duplex-connections, we can separate the numbers of edges at each vertex in a set of incoming and outgoing connections, see figure 3b. Thus, the result is a directed multigraph where the two sets of every vertex can be treated independently.

Figure 3: Conflict-Multigraph with half-duplex (a) and full-duplex (b) connections

The conflict-multigraph for each switch can be easily generated from the data of the switch ports and the relevant CLs for this switch. The usage of a switch with more than 32 ports is unusual. Though, the generated graph consists of a small amount of vertices. In realtime environments, up to several hundreds CLs are usual, this means |E| >> |V|.

Hubs simply repeat the incoming transmission to all ports except the incoming port. So, we can easily add hubs into the switched network. While one CL uses a port of a hub, no other lines are allowed to use the hub at the same time.

3.3 Calculating the Schedule using Edge Coloring

Firstly, we focus on edge-coloring of undirected multigraphs as shown in figure 3a. The smallest number of colors needed in an edge coloring of a graph G is called chromatic index χ′(G). Holyer [11] transformed the edge-coloring problem into the decision, whether the edges of a given multigraph can be edge-colored with t colors. He showed that the coloring is NP-hard for t ≥ 3.

As a result of the theorem of Vizing [12], the degree Δ(G) of the graph, is a lower bound for the chromatic index of an undirected multigraph. Further-more, an upper bound results upon the degree plus the multiplicity d of the graph. The outcome of this is Δ(G) ≤ χ′(G) ≤ Δ(G)+d, which means 5 ≤ χ′(G) ≤ 7 in our example. However, we can add the condition to find not always the chromatic index, but a coloring close to χ′(G). The result would be in worst case a schedule with “some” more colors then the minimum amount of colors.

A very simple coloring heuristics for multigraphs is a greedy first-fit algorithm, which belongs by declaration of Davis and Impagliazzo [13] to the class of greedy-algorithms with invariant prioritization. It 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
END WHILE
```

Figure 4: Simple greedy-algorithm

It can be shown for vertex coloring, that a greedy-algorithm is able to find an optimal coloring in the case of getting the optimal arrangement of vertices as an input sequence [12]. This statement can be transferred for edge-coloring of multigraphs as well. It relocates the problem on locating a beneficial sequence of the edges for initializing the greedy-algorithm. The optimal coloring can be found in any case with permutation of all egdes, which would result in a complexity of O(n!) for n edges. Emden-Weinert et al. [12] have already discussed heuristic approaches for pre-sorting the vertices in a vertex-coloring which are transferable as well: With a smallest-last arrangement, any edge with minimal degree is removed from the graph and is added to the beginning of a arrangement-list for greedy-input. Thus, the degree of
the graph decreases. As a result of this, the critical parts of the graph will be greedy-colored at first. Matula and Beck [14] verified, that smallest-last heuristics can be realized in O(|V|+|E|) steps. Brélaz [15] defines the saturation-largest-first heuristics (called dsatur), which is a greedy-coloring with dynamic calculation of the input vertex sequence. After coloring any edge with minimal color, we allocate every uncolored edge with a saturation level. This level represents the amount of forbidden colors for this edge. Thereafter, the edge with most forbidden colors is colored next, the saturation levels of the uncolored edges are updated and the algorithm continues.

3.4 Coloring with Greedy-Algorithm

We implemented a greedy algorithm for edge-coloring of undirected and directed multigraphs for the class of restricted problems defined in the beginning of the third chapter. To gain data of algorithm efficiency, we generated 32 switches with 4, 5, 6, 8, 12, 16, 24 and 32 ports as well as 2000, 4000, 8000 and 16000 randomized CLs. The time of generating a edge-coloring of directed conflict-multigraphs is 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(n^2) with n CLs, visible in figure 5a. It can also be detected, that the calculating time is reduced in a logarithmic way - see figure 5b - with constant amount of CL and increasing number of ports. The reason for this is the density of edges at one vertex. When 8000 CL are used in a 4-port switch, every port has 2000 adjacent edges on average. The complexity of 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. It can be said, that a greedy edge coloring of 4000 CLs can be done very fast on a standard PC. Besides, this number of independent communications at one switch is rather high for automation requirements. In addition to the speed of the algorithm, we have examined the quality of the results as well. For these purposes, we depict the calculated number of colors and subtract the degree of the multigraph. We know from theorem of Vizing, that any graph can not be edge-colored with less colors then its degree. So, the result of the difference contains the number of used colors more then the minimal quantity. This does not mean, that any graph can be colored with Δ(G) colors, e. g. a undirected circle with 5 vertices needs 3 colors and its degree amounts only 2. But it can be used as an indication, how many colors had to be used beyond the absolute minimum. The optimal coloring can not be found by examining all permutations of the edges or with a backtracking algorithm, because there are too many edges. However, the difference of the acquired number of edges and the degree of the multigraph is an indication for the quality of the algorithm. On average, the greedy algorithm needs at most 15 colors more than the degree of the graph, like Figure 5c shows. For example, with 6 port switches and 4000 randomized CL, an average degree of 1610 results. With 15/1610 = 0.009, we found a result less then 0.1% different to the possible optimal solution. Because the number of colors is equal to the required number of time-slots in the schedule for the switches, the results are 0.1% inferior to optimal results. The coloring results of Colors-Delta from switches with more then 6 ports are not displayed in figure 5c, because they are located nearby 0. Exceptions are graphs with 5-port switches. Here, the amount of used colors increases rapidly with the number of CLs. This is caused by specific attributes of graphs with an odd number of vertices. For example, circles with an odd number of participating vertices larger then 3 need more colors then their degree. Even if the graph is overfull, the problems of the odd numbers of vertices arises: With 16000 CL, we need over 10% more colors then the degree of the graph. On the other hand, we have to consider the switching technology: 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, without raising a conflict. Unfortunately, this effect is not limited on 5-port switches. If in a 8-port switch mostly 7 or 5 ports are used for communication, the effect arises as well, because the CLs generate a graph with mostly odd number of vertices. This effect is not visible in the testing environment because of the randomized CLs, which allocates the CLs equally to all ports on the average.

Figure 5: Graphical view of the results
To improve the results, a smallest-last or dsatur heuristics can be preceded. In addition to this, it is even possible to improve an existing edge coloring by re-coloring, if the result is not sufficient. A greedy algorithm tends to use low numbered colors more often. Because of this, a coloring often uses the highest colors only a few times. It can be attempted to re-color these edges with highest colors selectively and mark all re-colored edges. For this, we select an edge with the highest color and try to assign a lower color to it. Then, we look at the appearance of color-conflicts and try to find a free color within the range of used colors. If this is not possible, we repeat the procedure with an unmarked edge. In worst-case, the re-coloring is unsuccessful when an already marked edge has to be re-colored again.

3.5 Creating the Schedule

If the colored multigraph is available, the schedule can be generated easily by allocation of the CLs to their colors. CLs with the same color can be executed at the same time. If the colored multigraph is available, the schedule can be generated off-line depending on the individual configuration of automated production lines.

4. Conclusion and Future Work

We have shown in this paper, that our method can solve a class of restricted problems with unicast addressing, uniform packet-size, either half-duplex or full-duplex transmission, ideal distributors and devices without delay or jitter. This method consists of the following steps:

- Generating CLs out of the input data by using attribution technique. The method can be continued independently for each switch.
- Building a conflict-multigraph for edge-coloring.
- Optional pre-sorting of the input-edges will optimize the result of the coloring.
- Execution of greedy-coloring.
- Optional re-color heuristics in order to optimize the existing coloring, if the result is not sufficient.
- Generating the schedule for a switch out of the colors. A good quality of the implemented greedy algorithm has emerged as long as no odd amount of ports are used in communication. Otherwise, the results are acceptable with about 10% more colors used than needed in best-case.

The future scientific work will handle various aspects. Firstly, we will deal with the question, where the schedules will be set up. Up to now, we considered the schedules inside the switches, which would have to be modified. Otherwise, external arbiters could also be used to realize the schedule by polling requests. It would also be possible to realize the schedules in the devices directly with standard-switches. All methods vary in compatibility to Ethernet standards and speed. An additional field of research could be the consideration of additional constraints like device and distributor jitter as well as delay times. The ambition is to model realistic switch behavior with usual requests in order to generate feasible schedules.

We expect with our approach the realization of adaptable industrial Ethernet networks, which can vary between compatibility and throughput. The schedules can be generated off-line depending on the individual configuration of automated production lines.

References

[3] [http://standards.ieee.org/getieee802/802.3.htm](http://standards.ieee.org/getieee802/802.3.htm)
[10] D. Marx, Graph Coloring Problems and their Applications in Scheduling, John von Neumann PhD students Conference, Budapest University of Technology and Economics, 2003
[15] D. Brélaz, New methods to color the vertices of a graph, Communications of the ACM 22(4), 1979, 251-256, ISSN 0001-0782