On Optimal Communication Spanning Trees in Embedded Ethernet Networks

Jörg Sommer

University of Stuttgart, Institute of Communication Networks and Computer Engineering (IKR)
Pfaffenwaldring 47, 70569 Stuttgart, Germany
Email: joerg.sommer@ikr.uni-stuttgart.de

***Abstract*— In network design, the *Optimal Communication Spanning Tree* (OCST) problem is to find a spanning tree that connects the network entities and satisfies their communication requirements with minimal total cost. In an embedded Ethernet network, we organize the full duplex links into bundles and install these bundles into ducts. The *traditional* OCST problem does not take into account this fact.**

In this paper, we introduce a mathematical model of the communication cost of an embedded Ethernet network. We propose novel algorithms to find an OCST. Their principle idea is that they iteratively examine a set of neighboring trees that differ in one duct and choose the tree with the lowest cost. Finally, we evaluate the performance of the algorithms in terms of the quality of the solution found as well as the running time.

I. Introduction

IN the last decades, Ethernet evolved from a bus topology to a micro segmented network with full duplex links. Today, it is the predominant *Local Area Network* (LAN) technology and is considered as a *de facto* standard for network infrastructure. The flexibility and the plug-and-play feature of Ethernet are its keys to success. Thanks to the wide availability of its components, its large bandwidth, its reliability, and its backward compatibility, Ethernet has become an attractive option in many application areas [1].

A prominent example of Ethernet's application area is the industrial domain where Ethernet is gaining ground over traditional fieldbuses. Another area is avionics, where today's Ethernet proved its ability to fulfill real-time requirements needed in such an environment [2]. The automotive industry is investigating Ethernet as a suitable in-vehicle network technology [3]–[5].

A motivation to use Ethernet in embedded networks is the use of commercial off-the-shelf components. This allows manufactures to cut down the development cost as well as the time needed to build new components. Furthermore, the large number of Ethernet vendors and the wide range of products promote the competition to provide the best equipment at the lowest price.

Today, we use wiring harnesses to interconnect and to supply with power the electrical components in an embedded network. The motivation of using wiring harnesses, which bundle individual wires or smaller bundles together, is cost savings. The wires or bundles leave/join ducts at various points known as junction points or breakouts. Since we do not consider the wires required to supply the components with electrical power, we use the term *link harness*.

When designing an embedded Ethernet network, we have to minimize the total cost of the link harness while satisfying the communication requirements. In graph theory, this problem is generally well known as the *Optimal Communication Spanning Tree* (OCST) problem [6]. Normally, the OCST problem does not take into account that the links are organized into link bundles and installed into ducts as done in embedded networks.

In this paper, we propose and evaluate novel algorithms to find an OCST in an embedded Ethernet network. In such A network, the numbers of nodes, switches, and junction points as well as their positions are known. In addition, the occurring traffic is predictable and thus we are able to compute the communication demands.

The rest of the paper is organized as follows. First, we introduce the characteristics of an embedded Ethernet network and a mathematical model for the communication cost. Besides, we discuss the economy of scale of bundling links. In Section III, we propose algorithms to find an OCST. In Section IV, we evaluate the algorithms' performance. Finally, we conclude the paper in Section V and give an outlook for future work.

II. Characteristics and Modeling

A. Ethernet

The main requirements of a communication technology for a company and a manufacturer are planning reliability, future proof hardware, and easy management (i.e., plug-and-play capabilities). This especially includes scalability with respect to network size and link speed as well as simple migration. Consequently, the fast bandwidth evolution in the mid/late 1990-ies while maintaining backward-compatibility laid the fundamentals of Ethernet's success story.

Since its invention, Ethernet's line rate has evolved from 2.94 Mbps to higher rates: 10 Mbps in 1993 (IEEE 802.3a), 100 Mbps in 1995 (IEEE 802.3u), 1 Gbps in 1998 (IEEE 802.3z, [7]), and 10 Gbps in 2002 (IEEE 802.3ae). Currently, the IEEE P802.3ba task force is working on 40 Gbps and 100 Gbps [8]. In an embedded environment, the predominant line rate used is 100 Mbps [1]. If a link between two entities is overloaded, we have two options: Either we replace the link by a faster one or we add a number of parallel physical links having the same line rate and make use of Ethernet's *Link Aggregation* feature [7]. This is an optional feature for full duplex capable Ethernet devices. It provides a single logical interface to parallel physical links between two devices. This increases the link availability and enables a linear increase in the bandwidth. For further reading of this feature, we refer to Watanabe et al. They show in [9] the benefits and cost savings using Ethernet link aggregation. Throughout this work, in case of an overloaded link, we use a greedy strategy and make use of the second option (installing physical parallel links).

In [1], we describe the common Ethernet LAN technology and highlight its main features. Furthermore, we give a survey of Ethernet application fields.

B. Communication Demands

In traditional LANs, a client (workstation) typically communicates with a (workgroup) server. The workgroup server in turn communicates with a back-end server. This leads to a hierarchical structure, where on lower layers the clients send their packets to a switch that forwards the packets, e.g., to a router. In such a network, the traffic is aggregated on each hierarchical level. Consequently, the traffic of a link depends on the level in the hierarchy. In an embedded network, each node interacts with one or more nodes. Besides, we have different types of services and protocols: Peer-to-peer vs. multi-peer and multicast, confirmed and unconfirmed, acknowledgment and not, and connection and connectionless protocols [10].

For some applications, temporal characteristics like delay, jitter, and response time play an important role. Today, a number of different Ethernet solutions supporting different classes of QoS exist [1]. As defined in the IEEE 802.1Q standard [11], Ethernet allows frame tagging that enables traffic prioritization and a separation of real-time and best effort traffic (which gets the lowest priority).

In order to take into account the communication or traffic demands, in an embedded network, we introduce the principle of a *demand matrix* $\underline{R} = (r_{i,j})$, where $r_{i,j}$ specifies the amount of traffic between node i and j.

Normally, in an embedded network we know the type of applications and the number of sources. Furthermore, such networks are mostly closed. I.e., all occurring traffic is predictable (or shaped) [1]. Thus, we are able to specify the communication demands in detail. If an embedded network is interconnected with, e.g., an office LAN, traffic shapers and policers are applied to restrict the unforeseen traffic. In the following, we will not consider any temporal requirements. We assure that the resulting communication tree is able to carry the required traffic.

C. Link Harness Design

In today's embedded environments, as mentioned in Section I, wire harnesses are used to connect the network entities and to supply them with power. By bundling multiple wires together and deploying them inside ducts, they can be protected against various adverse effects such as vibrations, moisture, and over-heat [12]. Moreover, installing a wire harness instead of individual, unbundled wires reduces the installation time during the assembly and the installation process can be easily standardized.

Since Ethernet evolved from a bus to a micro-segmented network without collision domains, we have full duplex links between each node and its attached switch as well as between the switches. In Figure 1, we give an example of a full duplex link harness that does not consider other types of wires or cables (e.g., electrical power). Since the links do not have the same endpoints, some full duplex links may leave a duct at some points referred to as junction points or breakouts. At these junction points, a bifurcation is created into the duct in order to protect the leaving links as well as the remaining links in the original link harness.

By allowing multiple links following the same path to be bundled together and installed within a single duct,

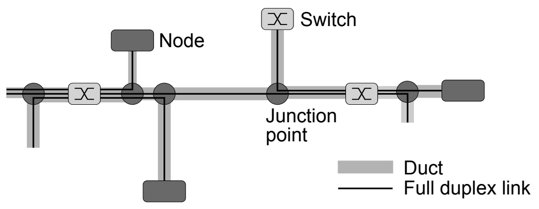

Fig. 1. Example of a link harness of an embedded Ethernet network.

the optimal topology is no more obtained by using the paths with the lowest cost. In fact, by allowing some links to be deployed along paths with higher cost, they can be bundled together along a common path segment and installed in a common duct. Thus, the cost penalty due to the use of individual higher link costs can be compensated by the cost benefit due to the use of a common duct instead of separate ones. This holds if we assume that the cost of ϕ links bundled together and installed within a common duct is smaller than the sum of the costs of the ϕ links deployed separately within ϕ different ducts. According to the before-mentioned property, we define the cost $\theta_{i,j}(\phi)$ of a single duct comprising ϕ links between an entity i at position (x_i, y_i) and an entity j at position (x_j, y_j) as:

$$\theta_{i,j}(\phi) = \begin{cases} 0 & \text{if } \phi = 0, \\ (\lambda \cdot \phi + c) \cdot d_{i,j} & \text{otherwise.} \end{cases} \quad (1)$$

with $\lambda + c = 1$, $\frac{\lambda}{c}$ is the ratio of the cost of a single link to the cost of a single duct for a unit length, and $d_{i,j}$ is the Manhattan distance between entity i and j [13]:

$$d_{i,j} = |x_i - x_j| + |y_i - y_j|. \quad (2)$$

The cost function can be generalized to more complex functions without affecting the proposed algorithms. Such generalization reflects more general assumptions such as the *Economy of Scale* where higher number of links bundled together within one duct results in lower cost per link.

The correspondence between the terminology employed for a link harness and the more conventional language of graph theory is as

- duct $\leftrightarrow$ edge
- nodes, switches, and junction points $\leftrightarrow$ vertexes
- cost of a single duct (*cf.* Equation 1) $\leftrightarrow$ cost (or weight) of an edge. Thus, the cost function, which associates an edge $e = \{i, j\}$ cost, is in that case $c : e \rightarrow \theta_{i,j}(\phi)$.

We can also model the link harness of an embedded Ethernet network as an undirected, weighted graph $G = (V, E, c)$. Since we investigate an Ethernet network without redundant paths, the graph G is acyclic and forms a tree.

D. Communication Cost

We have a given set of nodes $N_i(x_i, y_i)$ $(i = 1, \ldots, |N|)$, a given set of switches $S_i(x_i, y_i)$ $(i = 1, \ldots, |S|)$, and a given set of junction points $J_i(x_i, y_i)$ $(i = 1, \ldots, |J|)$. The objective is to find a tree at minimum cost that satisfies the communication demands and connects the nodes either directly or crossing junction points to the switches, and interconnects the switches either directly or crossing junction points. For an instance of this problem, we define the following matrices:

- $\underline{\Psi} = (\psi_{i,j})$ is a $|N| \times (|S| + |J|)$ matrix representing the existence of ducts between the nodes and the switches or the nodes and the junction points where $\psi_{i,j}$ is a binary variable specifying the presence or the absence of a duct between node N_i and switch S_j $(1 \leq j \leq |S|)$ or between node N_i and junction point $J_{j-|S|}$ $(|S| + 1 \leq j \leq |S| + |J|)$.

$$\psi_{i,j} = \begin{cases} 1 & \text{if } N_i \text{ is connected to } S_j \text{ or } J_{j-|S|}, \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

Since we have a tree structure, there exists exactly one duct between a node and either a switch or a junction point. Thus, we have the following constraint:

$$\sum_{j=1}^{|S|+|J|} \psi_{i,j} = 1, \ \forall i = 1, \ldots, |N| \quad (4)$$

- $\underline{\Delta} = (\delta_{i,j})$ is a $|N| \times (|S| + |J|)$ matrix representing the number of link segments between nodes and switches or between nodes and junction points respectively where $\delta_{i,j}$ is a non-negative integer variable.

$$\delta_{i,j} = \begin{cases} \geq 1 & \text{if } N_i \text{ is connected to } S_j/J_{j-|S|}, \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

- $\underline{\Upsilon} = (\upsilon_{i,j})$ is a $(|S| + |J|) \times (|S| + |J|)$ matrix representing the existence of ducts between the switches and the junction points where $\phi_{i,j}$ is a binary variable specifying the presence or the absence of a duct between switch S_i/junction point $J_{i-|S|}$

and switch S_j/junction point $J_{j-|S|}$.

$$\upsilon_{i,j} = \begin{cases} 1 & \text{if } S_i/J_{i-|S|} \text{ is connected to } S_j/J_{j-|S|}, \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

- $\underline{\Phi} = (\phi_{i,j})$ is an $(|S| + |J|) \times (|S| + |J|)$ matrix representing the number of link segments between switch S_i/junction point $J_{i-|S|}$ and switch S_j/junction point $J_{j-|S|}$ where $\phi_{i,j}$ is a non-negative integer variable.

$$\phi_{i,j} = \begin{cases} \geq 1 & \text{if } S_i/J_{i-|S|} \text{ is connected} \\ & \text{to } S_j/J_{j-|S|}, \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

The total communication cost C is defined as

$$C = \sum_{\substack{i=1,\ldots,|N| \\ j=1,\ldots,|S|+|J|}} \psi_{i,j} \cdot \theta_{i,j}(\delta_{i,j}) + \sum_{\substack{i=1,\ldots,|S|+|J|-1 \\ j=i+1,\ldots,|S|+|J|}} \upsilon_{i,j} \cdot \theta_{i,j}(\phi_{i,j}) \tag{8}$$

Throughout this paper, we denote an instance of this problem as T that holds all the above mentioned matrices. Furthermore, we denote the cost of an instance as $C(T)$ (*cf.* Equation 8).

Our OCST problem becomes the *Minimum Spanning Tree* (MST) problem if we have in each duct exactly one link segment ($\delta_{i,j} = 1$ and $\phi_{i,j} = 1$) or if the cost of a single duct (*cf.* Equation 1) is independent of the number of comprising links ($\lambda = 0$ and $c = 1$).

III. OCST Algorithms

The OCST problem is to find a spanning tree that connects the network entities and satisfies their communication demands at a minimum total cost [6]. Like other constrained spanning tree problems, the OCST problem is $\mathcal{NP}$-hard [14]. Cayley's formula identifies the number of spanning trees in a graph with n vertexes as n^{n-2} [15]. In this section, we propose algorithms to find an OCST. Since we do not prove the convergence behavior of the algorithms, these are heuristics [16].

As mentioned in Section I, the *traditional* OCST problem does not take into account that we bundle the links together and install them inside ducts. In order to compute the communication cost of an embedded Ethernet network, two steps are necessary: First, deploying the ducts that connect all the nodes, switches, and junction points. Second, deploying the full duplex links required to satisfy the communication demands in the ducts. Since we deploy links exclusively in ducts, we have to compute the link deployment whenever the duct deployment changes.

A. Grow Tree Algorithm

Assume that we have a set of *empty* ducts E that connects all the switches S and junction points J and we wish to grow it to a *complete* communication spanning tree T. This tree connects all the entities, including the nodes, and in each duct the number of links required is deployed. In other words, we wish to compute the variables of the matrices $\underline{\Psi}$, $\underline{\Delta}$, $\underline{\Upsilon}$, and $\underline{\Phi}$. In this section, we propose an algorithm that applies a greedy strategy to connect the nodes as well as a greedy strategy to deploy the links required.

In Algorithm 1, we first create and initialize an instance T of a communication spanning tree (Line 1 in Algorithm 1). Next, we compute the matrix $\underline{\Upsilon}$ with a given set of ducts E as an input (Line 2). Each duct has a *from* node and a *to* node. Since E connects all the switches S and junction points J, E induces the tree $G_E = (S \cup J, E)$.

Then, we have to connect the nodes. Thereby, we connect each node to its nearest (minimum distance) switch or junction point by deploying a duct (Lines 5 to 8) and compute the matrices $\underline{\Psi}$, $\underline{\Delta}$, and $\underline{\Phi}$ (Lines 9 to 11).

In order to compute the matrix $\underline{\Delta}$ based on the demand matrix $\underline{R}$, for each node we sum the traffic that goes from it to all other nodes and the traffic that goes to it, choose the maximum of these values, and round it up to the next smallest integer value. According to this integer value, we deploy the number of (parallel) links required on the shortest path from this node to its connected switch. Deploying parallel links means making use of Ethernet's link aggregation mechanism as mentioned in Section II-B.

After deploying the *node links*, we compute the matrix $\underline{\Phi}$, i.e., deploying the links required between the switches. We first compute for each switch pair the shortest path. Next, we compute a matrix that contains the demand values between neighboring switches. Based on this matrix, we finally calculate the number of (parallel) links required between these switches.

B. Algorithms

In this section, we propose algorithms to find an OCST. The principle idea of these algorithms is that they iteratively examine a set of neighboring trees that differ in one duct and choose the tree with the lowest cost.

Rothlauf [17] has shown that starting from a *Minimum Spanning Tree* (MST) increases the quality of the solution in comparison to start from a random solution. Besides, as mentioned in Section II-D, if we have exactly

Algorithm 1 Grow Tree

```
Input: $N, S, J, E, \underline{R}$
Output: $T$                         ▷ Communication spanning tree
 1: Create and initialize an instance $T$ with $|N|, |S|, |J|$
 2: Compute variables $\upsilon_{i,j}$ of $T$ with $E$ as an input
 3: $V \leftarrow S \cup J$
 4: $E' \leftarrow \emptyset$
 5: for all $u \in N$ do
 6:     $v \leftarrow \arg\min_{v \in V}(d_{u,v})$
 7:     $E' \leftarrow E' \cup \{\{u, v\}\}$
 8: end for
 9: Compute variables $\psi_{i,j}$ of $T$ with $E'$ as an input
10: Compute variables $\delta_{i,j}$ of $T$ with $\underline{\Psi}, \underline{\Upsilon}, \underline{R}$ as an input
11: Compute variables $\phi_{i,j}$ of $T$ with $\underline{\Psi}, \underline{\Upsilon}, \underline{R}$ as an input
12: return $T$
```

one link segment in each duct, the problem becomes the MST problem. Thus, we first create an MST as a starting point for our algorithms.

1) MST: There are two commonly known algorithms to compute an MST: *Prim*'s algorithm [18] and *Kruskal*'s algorithm [19]. Both are greedy algorithms. Normally, a weighted graph G with a set of edges is given. In our case, the edges represent ducts and we can deploy ducts arbitrarily. Thus, we may assume a fully meshed graph. In Algorithm 2, we propose a simplified MST algorithm based on Prim's algorithm. This algorithm connects all the switches S and junction points J together; but not the nodes. We have to ensure that a node is not connected to another node directly. This would be an invalid network configuration. But this may happen with a *traditional* MST algorithm. Thus, we connect each node with a greedy strategy, as mentioned in Section III-A, to either a switch or a junction point, while we grow the tree T with the Grow Tree algorithm (*cf.* Algorithm 1).

The principle idea of our MST algorithm is that the duct $\{u, v\}$ added to the set of ducts E (Line 8 in Algorithm 2) is always a least-distance duct connecting the tree to an entity, either a switch or a junction point, not in the tree. We first initialize an empty set of entities V' that contains later the already connected entities. After the initialization phase, the tree starts from an arbitrary root entity $u \in V$ (Line 4), and grows until the ducts in E connect all the switches and junction points in V. In each step, the duct at the minimum cost, i.e., the minimum distance, is added to E. At the end, we return E that induces the tree $G_E = (S \cup J, E)$.

We now propose two local search algorithms and a Greedy Search algorithm that try to find an OCST.

Algorithm 2 Minimum Spanning Tree

```
Input: $S, J$            ▷ Set of switches and junction points
Output: $E$                                   ▷ Set of ducts
 1: $V \leftarrow S \cup J$
 2: $V' \leftarrow \emptyset$
 3: $E \leftarrow \emptyset$
 4: $V' \leftarrow V' \cup \{u\}$ with $u \in V$
 5: $V \leftarrow V \setminus \{u\}$
 6: while $V \neq \emptyset$ do
 7:     $(u, v) \leftarrow \arg\min_{(u,v) \in V' \times V}(d_{u,v})$
 8:     $E \leftarrow E \cup \{\{u, v\}\}$
 9:     $V' \leftarrow V' \cup \{v\}$
10:     $V \leftarrow V \setminus \{v\}$
11: end while
12: return $E$
```

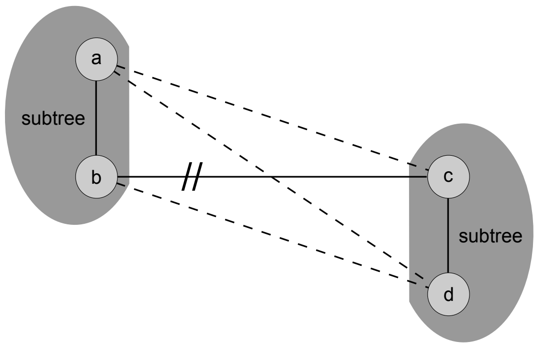

Fig. 2. Example of cutting a tree at duct $\{b, c\}$ and all the possible ducts $\{a, c\}, \{a, d\}, \{b, d\}$ that would form a neighboring tree (dashed lines).

2) Local Search I: The idea of the first local search algorithm is that we cut an (initial) tree G_E into two disjoint subtrees by removing the duct e_i from the set of ducts E (Line 7 of Algorithm 3). Afterwards, we discover in Line 8 the disjoint partition sets U and V with $U \uplus V = S \cup J$.

Then, we examine the possible ducts that connect the resulting subtrees ($\{u, v\} | u \in U, v \in V$). In Figure 2, we give an example of this procedure. There, we remove duct $\{b, c\}$. The disjoint partition sets are $U = \{a, b\}$ and $V = \{c, d\}$. The ducts $\{\{a, c\}, \{a, d\}, \{b, d\}\}$ come into consideration to connect the subtrees. We call the resulting trees *neighboring trees*. The number of neighbors of a tree with n vertexes depends on its structure and varies between $(n-1)(n-2)$ and $1/6\ n\ (n-1)(n+1) - n + 1$ [17].

For each set of ducts $E'' = E' \cup \{\{u, v\}\}$, we compute the communication spanning tree T' by calling the function GROW-TREE. If this tree has lower cost

Algorithm 3 Local Search I

```
Input: $N, S, J, \underline{R}$
Output: $T$                                              ▷ OCST
 1: $E \leftarrow \mathrm{MST}(S, J)$
 2: $T \leftarrow$ GROW-TREE($N, S, J, E, \underline{R}$)
 3: $i \leftarrow 0$
 4: while $i < |E|$ do
 5:     $i \leftarrow i + 1$
 6:     $e_i \leftarrow i^{th}$ element of $E$
 7:     $E' \leftarrow E \setminus e_i$
 8:     Discover partition sets $U$ and $V$ of
        $G_{E'} = (S \cup J, E')$ with $U \uplus V = S \cup J$
 9:     for all $(u, v) \in U \times V$ do
10:         $E'' \leftarrow E' \cup \{\{u, v\}\}$
11:         $T' \leftarrow$ GROW-TREE($N, S, J, E'', \underline{R}$)
12:         if $C(T') < C(T)$ then
13:             $T \leftarrow T'$
14:             $E \leftarrow E''$
15:             $i \leftarrow 0$
16:         end if
17:     end for
18: end while
19: return $T$
```

Algorithm 4 Local Search II

```
Input: $N, S, J, \underline{R}$
Output: $T$                                              ▷ OCST
 1: $E \leftarrow \mathrm{MST}(S, J)$
 2: $T \leftarrow$ GROW-TREE($N, S, J, E, \underline{R}$)
 3: do
 4:     improvement $\leftarrow$ FALSE
 5:     $E' \leftarrow E$
 6:     for all $e \in E'$ do
 7:         $E'' \leftarrow E' \setminus \{e\}$
 8:         Discover partition sets $U$ and $V$ of
            $G_{E'} = (S \cup J, E')$ with $U \uplus V = S \cup J$
 9:         for all $(u, v) \in U \times V$ do
10:             $E''' \leftarrow E'' \cup \{\{u, v\}\}$
11:             $T' \leftarrow$ GROW-TREE($N, S, J, E''', \underline{R}$)
12:             if $C(T') < C(T)$ then
13:                 $T \leftarrow T'$
14:                 $E \leftarrow E'''$
15:                 improvement $\leftarrow$ TRUE
16:             end if
17:         end for
18:     end for
19: while improvement
20: return $T$
```

than the (current) tree T (Line 12), we accept T' as the (current) OCST (Line 13), update the set of ducts E (Line 14), and reset the index i (Line 15). At the end, we return T (Line 19).

3) Local Search II: In Algorithm 4, we propose a second version of a local search algorithm that starts again with an (initial) tree T and repeats until this tree cannot be improved any more. In principle, this algorithm works similar to the previous local search algorithm. Here, we examine all the neighboring trees that grow from the set of ducts E' and do not stop as soon as we have found an improved solution as we did in the previous version by resetting the index i. Consequently, this version searches a larger portion of the solution space, which may be very time-consuming.

4) Greedy Search: Finally, we present a *Greedy Search* algorithm to find an OCST. This algorithm has two additional input parameters: d that defines the depth of the search and n that defines the number of neighboring trees that we wish to examine.

As the name implies, this algorithm follows a greedy strategy as we cut the tree $G_{E''}$ by iteratively removing the i^{th}-expensive duct.

Thereby, we tag each duct of E with its cost (*cf.* Equation 1), sort the ducts in a descending order by their cost and store them in a working copy E' (Line 8 and 9 of Algorithm 5). Next, we cut the tree $G_{E'}$ by removing a duct and store in N the ducts that would connect the two resulting subtrees.

Next, we sort the ducts of N in an ascending order by their length (Line 15), i.e., by their distance $d_{i,j}$ (*cf.* Equation 2). After sorting, we compute the communication spanning tree T' for each of the n cheapest neighboring trees by calling the function GROW-TREE. If one of these trees has lower cost than the (current) OCST, we accept it as the (current) OCST and update E. We repeat this procedure until we do not achieve an improvement. At the end, we return T.

IV. EVALUATION

In this section, we evaluate the algorithms' performance. We focus on the quality of the solution found as well as on the computational cost. In terms of the computational cost, we evaluate the total number of neighboring trees examined (*#ngb trees*). The more neighboring trees we examine, the longer the computation time. Normally, the computation time to find an OCST of an embedded Ethernet network plays not an important role. This is not true, if we aim to embed one of the algorithms in an

Algorithm 5 Greedy Search

```
Input: $N, S, J, \underline{R}, d, n$
Output: $T$                                   ▷ OCST
 1: $E \leftarrow$ MST$(S, J)$
 2: $T \leftarrow$ GROW-TREE$(N, S, J, E, \underline{R})$
 3: do
 4:   improvement ← FALSE
 5:   for all $e \in E$ do
 6:     Tag $e$ with its cost                 ▷ cf. Equation 1
 7:   end for
 8:   Sort ducts of $E$ by their cost
      $c(e_1) \geq c(e_2) \geq ... \geq c(e_{|E|})$
 9:   $E' \leftarrow E$
10:   for i ← 1 to $\min(d, |E|)$ do
11:     $e_i \leftarrow i^{th}$ element of $E'$
12:     $E'' \leftarrow E' \setminus e_i$
13:     Discover partition sets $U$ and $V$ of
        $G_{E''} = (S \cup J, E'')$ with $U \uplus V = S \cup J$
14:     $N \leftarrow \{u, v\} | (u, v) \in U \times V$
          ▷ $N$: Set of ducts to create neighboring trees
15:     Sort ducts of $N$ by their length
        $l(n_1) \leq l(n_2) \leq ... \leq l(n_{|N|})$
16:     for j ← 1 to $min(n, |N|)$ do
17:       $n_j \leftarrow j^{th}$ element of $N$
18:       $E''' \leftarrow E'' \cup n_j$
19:       $T' \leftarrow$ GROW-TREE$(N, S, J, E''', \underline{R})$
20:       if $C(T') < C(T)$ then
21:         $T \leftarrow T'$
22:         $E \leftarrow E'''$
23:         improvement ← TRUE
24:       end if
25:     end for
26:   end for
27: while improvement
28: return $T$
```

outer optimization algorithm that tries to find an overall optimal solution with the optimal number of switches and junction points as well as their positions as we did in [20]–[22].

A. Comparison

In order to evaluate the algorithms, we examine different problem sizes (number of nodes, of switches, and of junction points) as shown in Table I. The network entities are placed randomly on a grid with size 1000×1000 fields. For each problem size, we generate 100 random problem instances with a randomly generated demand matrix $\underline{R}$ and randomly placed entities (quasi Monte Carlo simulation). Our demand matrix generator works according to a *Peer-to-Peer* model in which each node communicates with one or more nodes. The principle of the demand generator is shown in the Appendix. Furthermore, we chose different values for λ (*cf.* Equation 1) and μ (*cf.* Appendix). In all the cases, the number of data streams s (*cf.* Appendix) is set to the number of nodes. Besides, we parametrize the Greedy Search algorithm with $d = 10$ and $n = 10$.

The column *Initial solution* of Table I contains the cost of the initial communication spanning tree. This corresponds to the cost of the initial tree that we compute in Algorithm 3, 4, and 5 at Line 2. The column *#ngb trees* contains the total number of neighboring trees examined. The values in the columns C and *#ngb trees* are mean values over the 100 experiments. The column ΔC contains the delta in percentage between the mean costs (C) of the Local Search II (100%) and the Greedy Search algorithm.

As expected, the algorithms improve the initial solution dramatically. As shown in Table I, the computational cost of the algorithms increases with the network size. However, the computational cost of the Greedy Search algorithm does not increase as much as of the local search algorithms. The Greedy Search algorithm keeps its pace and is thus applicable for large embedded Ethernet networks. On average, this algorithm examines less than 1100 neighboring tress and runs in our implementation approximately 10 seconds per experiment.

The Local Search II algorithm considers a larger solution space. Thus, its computational cost is higher than that of the Local Search I algorithm and achieves better mean cost.

In comparison to the local search algorithms and taking into account the computational cost, the Greedy Search algorithm performs excellently. Its mean costs C are in many cases only 1% above these of the Local Search II algorithm. The largest gap between the mean cost of the Greedy Search algorithm and the Local Search II algorithm is 11.86%. Thus, if we are interested in a solution with a very high quality (low cost), we should apply the Local Search II algorithm, otherwise the Greedy Search algorithm. Especially, if we wish to integrate one of the algorithms in an outer optimization algorithm that tries to find an overall optimal solution with the optimal number of switches and junction points as well as their placement, the Greedy Search algorithm is the first choice.

TABLE I
COMPARISON OF THE ALGORITHMS' PERFORMANCE

$\|N\|$	$\|S\|$	$\|J\|$	μ	λ	Initial solution	Local Search I		Local Search II		Greedy Search		ΔC
					C	C	*#ngb trees*	C	*#ngb trees*	C	*#ngb trees*	[%]
25	5	10	0,01	0,2	8 359,3	7 989,9	994	**7 982,4**	1 643	**7 985,4**	371	0,04
				0,4	9 561,1	8 817,1	1 035	**8 807,3**	1 873	**8 814,8**	438	0,09
				0,8	11 964,6	10 313,4	1 096	**10 304,4**	2 268	**10 332,4**	527	0,27
			0,1	0,2	8 390,9	8 032,7	981	**8 023,1**	1 651	**8 025,9**	376	0,03
				0,4	9 624,2	8 882,5	1 030	**8 878,3**	1 896	**8 886,4**	448	0,09
				0,8	12 091,0	10 429,9	1 070	**10 410,0**	2 269	**10 456,3**	530	0,44
50	10	10	0,01	0,2	12 497,9	12 075,3	2 699	**12 068,8**	4 354	**12 093,2**	394	0,20
				0,4	13 963,0	13 045,9	2 979	**13 040,0**	5 214	**13 073,4**	482	0,26
				0,8	16 893,3	14 837,2	2 954	**14 825,8**	6 482	**14 912,5**	584	0,58
			0,1	0,2	12 776,5	12 331,8	2 758	**12 324,9**	4 521	**12 353,2**	417	0,23
				0,4	14 520,2	13 519,8	2 959	**13 509,2**	5 462	**13 556,9**	515	0,35
				0,8	18 007,7	15 722,6	2 912	**15 665,2**	6 492	**15 827,1**	623	1,03
50	10	25	0,01	0,2	12 615,0	11 514,2	18 331	**11 505,3**	39 469	**11 724,8**	586	1,91
				0,4	15 073,1	12 896,4	19 166	**12 884,4**	46 807	**13 173,8**	729	2,25
				0,8	19 989,2	15 337,3	18 267	**15 328,9**	56 294	**15 905,1**	892	3,76
			0,1	0,2	12 858,4	11 773,4	18 264	**11 754,8**	38 309	**11 976,6**	616	1,89
				0,4	15 559,7	13 373,4	18 924	**13 331,2**	45 917	**13 701,1**	722	2,77
				0,8	20 962,5	16 181,0	19 008	**16 114,2**	55 088	**16 905,6**	861	4,91
100	25	25	0,01	0,2	17 050,9	16 285,7	64 937	**16 274,0**	119 799	**16 516,8**	555	1,49
				0,4	19 106,1	17 564,8	73 928	**17 559,7**	146 991	**17 932,1**	653	2,12
				0,8	23 216,7	19 892,1	66 690	**19 877,0**	181 351	**20 532,5**	844	3,30
			0,1	0,2	17 984,1	17 093,7	60 265	**17 072,6**	116 836	**17 354,1**	586	1,65
				0,4	20 972,8	19 053,3	67 015	**19 017,4**	139 060	**19 564,6**	724	2,88
				0,8	26 950,1	22 572,4	62 415	**22 488,1**	169 181	**23 777,0**	837	5,73
100	25	50	0,01	0,2	17 759,0	15 817,6	270 757	**15 812,2**	622 287	**16 387,3**	771	3,64
				0,4	21 168,8	17 410,8	259 470	**17 393,0**	725 817	**18 269,9**	910	5,04
				0,8	27 988,4	20 158,3	231 145	**20 125,3**	828 552	**21 897,0**	1 035	8,80
			0,1	0,2	18 633,6	16 595,0	255 718	**16 566,9**	580 304	**17 247,0**	792	4,11
				0,4	22 917,9	18 845,8	247 671	**18 766,8**	661 868	**19 978,4**	930	6,46
				0,8	31 486,6	22 726,7	221 827	**22 615,4**	735 741	**25 297,4**	1 032	11,86

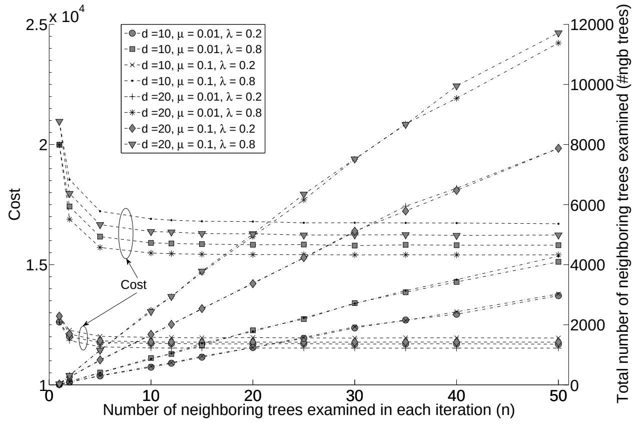

Fig. 3. The impact of the parameter n on the quality of the solution found and the computational cost (total number of neighboring trees examined).

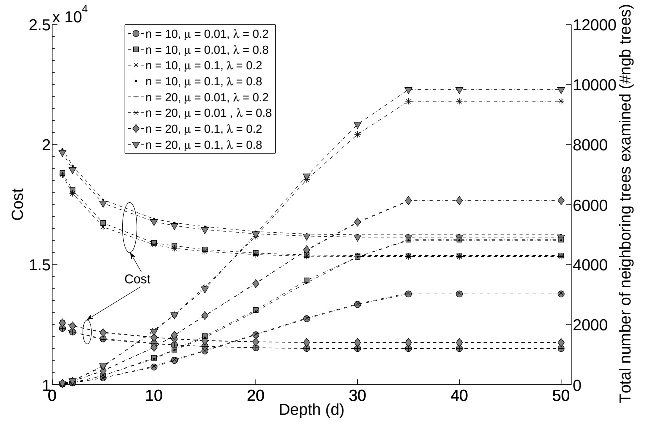

Fig. 4. The impact of the parameter d on the quality of the solution found and the computational cost (total number of neighboring trees examined).

B. Greedy Search Algorithm

In this section, we evaluate the impact of the parameters d and n on the Greedy Search algorithm on the solution's quality and the computational cost. For this purpose, we pick out a network with 50 nodes, 10 switches, and 25 junction points. Again, in all the cases, the number of data streams is set to the number of nodes. We chose $\mu = 0.1$ and 0.01 as well as $\lambda = 0.2$ and 0.8.

In Figure 3 and 4, the dashed lines show the cost and the dash-dot lines show the computational cost characteristics in terms of total number of neighboring trees examined. As shown in Figure 3, with more than 10 neighboring trees examined we achieve only a small improvement while the computational cost increases nearly linear. Up to a certain threshold, the same holds for the *depth* parameter d. As we see in Figure 4, in this example, it makes no sense to choose the parameter d larger than 35. As shown in Line 10 in Algorithm 5, we choose the smaller value of the number of ducts ($|E|$) and the parameter d. Since we investigate an Ethernet network without redundant paths, the maximal number of ducts $|E|$ equals $|S| + |J| - 1$; in this example 34.

V. Conclusion and Outlook

Today, Ethernet is the predominant LAN technology and it becomes also an attractive option in embedded environments, e.g., avionics and industrial domain. In embedded environments, wire harnesses are used to connect the network entities. The wires or bundles leave/join ducts at various points. Since Ethernet evolved from a bus topology to a micro segmented network, we have full duplex links between each node and switch as well as between the switches. By allowing multiple links following the same path to be bundled together, the optimal topology is no more obtained by using the paths with the lowest cost.

In this paper, we proposed novel algorithms to find an OCST of an embedded Ethernet network. Furthermore, we introduced a mathematical model of the communication cost. We evaluated the algorithms' performance and discussed the trade-off between the quality of the solution obtained and the computational cost. We have shown that the Greedy Search algorithm finds high quality solutions comparable to the results of the local search algorithms while its computational cost is much lower. Consequently, we can integrate this algorithm in an outer optimization algorithm that tries to find an overall optimal solution with the optimal number of switches and junction points as well as their placement. Furthermore, we have evaluated different parameter settings of the Greedy Search algorithm.

The algorithms in this paper are not limited to optimize the link harness of embedded Ethernet networks. With minor modifications, they can be generalized and can take into account, e.g., power wires.

In contrast to traditional LANs, embedded Ethernet networks have to fulfill resilience requirements due to safety reasons. In such a resilient Ethernet network each node has to be connected at least to two switches. The switches themselves have to be interconnected in a resilient structure. Currently, we are working on algorithms that minimize the link harness cost of a resilient Ethernet network.

Acknowledgment

The author would like to thank Benjamin Fischbach for the implementation of the Greedy Search algorithm and parts of the local search algorithms. The author would also like to thank Andreas Reifert and Thomas Werthmann for their reviews and for fruitful discussions.

References

[1] J. Sommer, S. Gunreben, A. Mifdaoui, F. Feller, M. Köhn, D. Saß, and J. Scharf, "Ethernet – A Survey on its Fields of Application," *IEEE Communications Surveys & Tutorials*, vol. 12, no. 2, April 2010.

[2] ARINC 664, *Aircraft Data Network, Part 7: Deterministic Networks*, 2003.

[3] J. Hillebrand, M. Rahmani, R. Bogenberger, and E. Steinbach, "Coexistence of Time-Triggered and Event-Triggered Traffic in Switched Full-Duplex Ethernet Networks," in *Proceedings of the IEEE Second International Symposium on Industrial Embedded Systems (SIES 2007)*, 2007, pp. 217–224.

[4] M. Rahmani, R. Steffen, K. Tappayuthpijarn, E. Steinbach, and G. Giordano, "Performance analysis of different network topologies for in-vehicle audio and video communication," *4th International Telecommunication Networking Workshop on QoS in Multiservice IP Networks (IT-NEWS 2008)*, pp. 179–184, February 2008.

[5] M. Rahmani, K. Tappayuthpijarn, B. Krebs, E. Steinbach, and R. Bogenberger, "Traffic Shaping for Resource-Efficient In-Vehicle Communication," *IEEE Transactions on Industrial Informatics*, 2009, accepted for publication.

[6] T. C. Hu, "Optimum Communication Spanning Trees," *SIAM Journal on Computing*, vol. 3, no. 3, pp. 188–195, September 1974. [Online]. Available: http://link.aip.org/link/?SMJ/3/188/1

[7] IEEE Computer Society, "802.3: IEEE Standard for Local and Metropolitan Area Networks–Carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications," 2005.

[8] ——, "P802.3ba: IEEE Standard for Local and Metropolitan Area Networks–Carrier Sense Multiple Access with Collision Detection (CSMA/CD) Access Method and Physical Layer Specifications - Amendment: Media Access Control Parameters, Physical Layers and Management Parameters for 40 Gb/s and 100 Gb/s Operation," 2008. [Online]. Available: http://www.ieee802.org/3/ba/

[9] W. Takafumi, N. Masahiro, T. Hiroyasu, T. Otsuka, and M. Koibuchi, "Impact of topology and link aggregation on a PC cluster with Ethernet," *2008 IEEE International Conference on Cluster Computing*, pp. 280–285, October 2008.

[10] J.-P. Thomesse, "Fieldbus technology in industrial automation," in *Proceedings of the IEEE*, vol. 93, no. 6. IEEE, June 2005, pp. 1073 – 1101.

[11] IEEE Computer Society, "802.1Q: IEEE Standard for Local and Metropolitan Area Networks–Virtual Bridged Local Area Networks," 2005.

[12] E. Aguirre and B. Raucent, "Performances of wire harness assembly systems," *IEEE International Symposium on Industrial Electronics (ISIE '94)*, pp. 292–297, May 1994.

[13] P. E. Black, "Manhattan distance," in *Dictionary of Algorithms and Data Structures*. U.S. National Institute of Standards and Technology, May 2006. [Online]. Available: http://www.itl.nist.gov/

[14] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, April 1979.

[15] A. Cayley, *The Collected Mathematical Papers of Arthur CayleyGarey*. BiblioBazaar, May 2009.

[16] H. Müller-Merbach, "Heuristics and their design: a survey," *European Journal of Operational Research*, vol. 8, no. 1, pp. 1–23, September 1981.

[17] F. Rothlauf, "On Optimal Solutions for the Optimal Communication Spanning Tree Problem," *Operations Research*, vol. 57, no. 2, pp. 413–425, March 2009. [Online]. Available: http://or.journal.informs.org/cgi/content/abstract/57/2/413

[18] R. Prim, "Shortest connection networks and some generalizations," *Bell System Technical Journal*, vol. 36, pp. 1389–1401, 1957.

[19] J. B. Kruskal, "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem," *Proceedings of the American Mathematical Society*, vol. 7, no. 1, pp. 48–50, February 1956.

[20] J. Sommer and E. A. Doumith, "Topology Optimization of In-vehicle Multimedia Communication Systems," in *Proceedings of the First Annual International Symposium on Vehicular Computing Systems (ISVCS 2008)*, Dublin, July 2008.

[21] J. Sommer, E. A. Doumith, and Q. Duval, "On Link Harness Optimization of Embedded Ethernet Networks," in *Proceedings of the IEEE International Symposium on Industrial Embedded Systems (SIES 2009)*, July 2009, pp. 191–200.

[22] J. Sommer, E. A. Doumith, and A. Reifert, "Cost-based Topology Optimization of Embedded Ethernet Network," *Journal of Embedded and Real-Time Communication Systems (IJERTCS)*, 2010, accepted for publication.

APPENDIX

The principle of the communication, or traffic, demand generator is shown in Algorithm 6. This algorithm expects the following input parameters:

- $|N|$: A non-negative integer variable that defines the size of the demand matrix.
- μ: A non-negative integer variable that defines the mean traffic demand.
- s: A non-negative integer variable that defines the number of data sources or streams.

Algorithm 6 Communication Demand Generator

```
Input: $|N|, \mu, s$
Output: $\underline{R}$                ▷ Communication demand matrix
 1: Generate an empty $|N| \times |N|$ demand matrix $\underline{R}$
 2: for i ← 1 to $|N|$ do
 3:     for j ← 1 to $|N|$ do
 4:         if $i \neq j$ then
 5:             $r_{i,j} \leftarrow \varepsilon$
 6:         end if
 7:     end for
 8: end for
 9: for k ← 1 to s do
10:     Generate distinct random numbers:
        $\mathrm{src}, \mathrm{dest} \in [1, |N|]$
11:     Generate random number: $r \in [\varepsilon, 1]$ with mean $\mu$
12:     $r_{\mathrm{src,dest}} \leftarrow r_{\mathrm{src,dest}} + r$
13:     $r_{\mathrm{dest,src}} \leftarrow r_{\mathrm{dest,src}} + r \cdot 0.05$
14: end for
15: return $\underline{R}$
```

At the beginning, we assign each node pair in each direction a constant communication demand holding a very small positive value ε (Line 5 in Algorithm 6). This represents a small amount of traffic going from each node to each other node (e.g., management traffic).

We use two streams of pseudo-random numbers: One to draw the source (src) and destination nodes (dest), and one to generate the traffic demands $r_{\mathrm{src,dest}}$. We draw the source and destination nodes from a discrete uniform distribution between 0 and $|N|$. The size of the quadratic demand matrix $\underline{R}$ is equal to the number of nodes $|N|$.

The demand generator creates a number s of data streams. In this paper, a stream is simply characterized by an amount of traffic as well as exactly one source and one destination node. We draw the amount of traffic $r_{\mathrm{src,dest}}$ randomly from a negative-exponential distribution with mean value μ. In order to be *more realistic*, we bound the values of the negative-exponential distribution. This means if the random value is smaller than the lower boundary (ε) or greater than the upper boundary (1), we draw-out a new random number. This is done as long as a value between lower and upper boundary is obtained. Since we assume that for each stream is going some traffic in the reverse direction, e.g., acknowledgment and control messages, we add a small amount of traffic, in that case 5%, that goes from the destination to the source node (Line 13 in Algorithm 6).