Decentralized Routing Algorithm with Physical Time Windows for Modular Conveyors

Simon Sohrt and Ludger Overmeyer

***Abstract*—We describe a decentralized routing algorithm with physical time windows for modular conveying systems. Existing routing algorithms for modular conveyors are already capable of bi-directional conveying while avoiding conflicts such as collisions, deadlocks, livelocks and starvation effects. In addition to avoiding conflicts, routing algorithms must also select routes that reduce the travel time. No existing algorithm for modular conveyors bases this decision on the expected physical lead time, even though physical lead time directly affects the system throughput. In this publication, we present an algorithm that uses the physical lead time to select routes while avoiding conflicts.**

***Note to Practitioners*—Conventional automated conveying systems are inflexible to layout changes. Modular conveying systems have been developed that address this issue. Their layout can be changed within minutes with no manual configuration necessary. This flexibility is enabled by decentralized routing algorithms. In this publication, we present a novel decentralized routing algorithm that selects routes based on the physical lead time. Since the throughput is a result of the physical lead time, the throughput is optimized.**

***Index Terms*—Modular conveyors, Multi-agent system, Decentralized control, Conflict-free, Bi-directional routing**

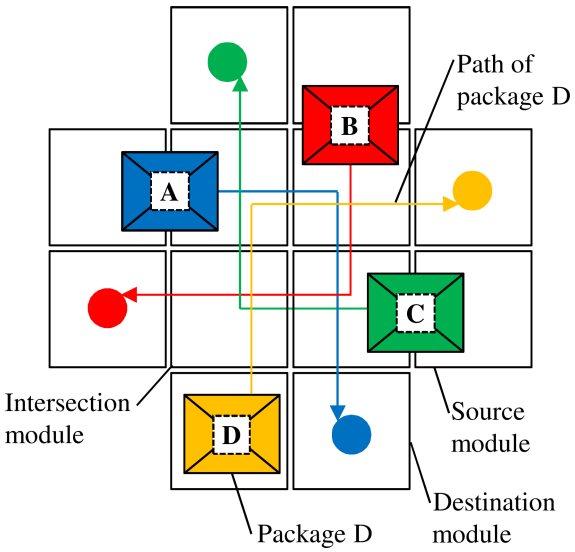


Fig. 1. Intersection consisting of four modular conveyors (based on [4])

I. Introduction

TODAY'S material flow is handled by a combination of conventional automated conveying systems and manual labor. On one hand, conventional conveying systems have a high throughput but are inflexible to layout changes. On the other hand, manual labor has the opposite characteristics.

Flexible automated conveying systems combine the best of both worlds: having a high throughput *and* having high layout flexibility. According to Furmans et al. [1] a system has a high layout flexibility, if the layout can be changed within minutes or at most a few hours. Furmans et al. [1] also identified design patterns of flexible conveying systems – the conveying system must consist of identical modules with decentralized controls. These modules have wheels underneath, and thus, they can be quickly rearranged into a new layout. When the modules have been rearranged, the decentralized controls configure themselves by exchanging messages with their neighboring modules. Thus, no manual configuration of the system is necessary. In addition to having a modular design and decentralized control, all conveying systems must avoid conflicts to ensure the conveying of packages. The most critical conflicts are:

1) **Collisions** occur when two packages collide with each other and interlock, and thus, are no longer conveyed.
2) **Deadlocks** occur when at least two occupied conveyors are cyclically waiting on each other to become available [2].
3) **Livelocks** occur when a package performs the same cycle of movements over and over again [3].
4) **Starvation** occurs when two packages from different directions compete at an intersection for right of way. If one direction has high throughput and is always prioritized, the packages from the other direction will never get closer to their destination [4].

Conflict-avoidance is handled by routing algorithms. The need for routing algorithms is depicted in Fig. 1. In this example, four modular conveyors form an intersection. The intersection is surrounded by source and destination modules through which packages may enter and leave. In this example, the conveying of packages A, B, and C has already begun, while package D just entered the system. The shortest path for every package is depicted by arrows. If package D enters the intersection immediately and follows the shortest path, a conflict occurs (either a collision or a deadlock). Several routing algorithms have been developed that prevent conflicts (see Section II).

In addition to avoiding conflicts, routing algorithms must also select routes that positively affect the throughput. Existing routing algorithms have used different methods of selecting routes, but no algorithm has selected routes based on the expected travel time. In this paper, we propose a novel routing

S. Sohrt and L. Overmeyer are with the Institute for Transport and Automation, Leibniz University Hannover, Hannover, Germany (simon.sohrt@ita.uni-hannover.de; ludger.overmeyer@ita.uni-hannover.de)

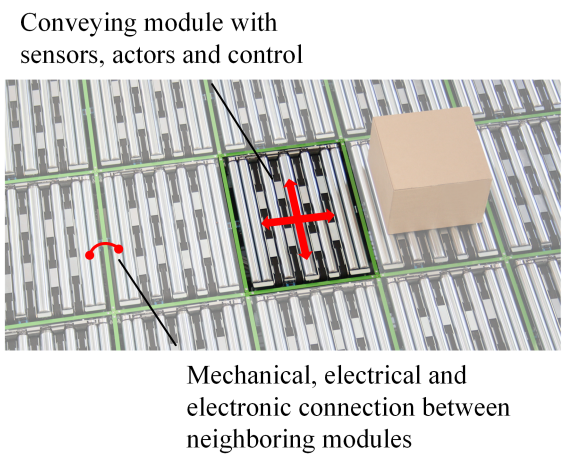


Fig. 2. The GridSorter systems is made up of several large-scaled conveying modules [4]

algorithm that selects routes based on the expected travel time by utilizing physical time windows. Since the throughput is a result of the travel time, the throughput is optimized by basing the selection of routes on the expected travel times.

Our proposed routing algorithm is designed for decentralized controlled modular conveyors. Every conveyor has a schedule and a clock which is synchronized through the network. The synchronized clocks are needed to reserve physical time windows on the schedules. The conveyors reserve routes by exchanging messages: when a conveyor receives a request, it checks its schedule. If the request can be accepted, it sends a request to the next conveyor. If the request can not be accepted, a new time is proposed to the requesting conveyor.

The rest of this paper is organized as follows. In Section II, we present state-of-the-art routing algorithms for modular conveying systems and for automated guided vehicles (AGV). We chose to include AGV routing algorithms because they can be modified to work in modular conveyors. In Section III, we describe our algorithm, and subsequently, in Section IV, we give an application example of the system behavior. Finally, we present our conclusion and outlook in Section V.

II. Related Research

A. Modular Conveying Systems

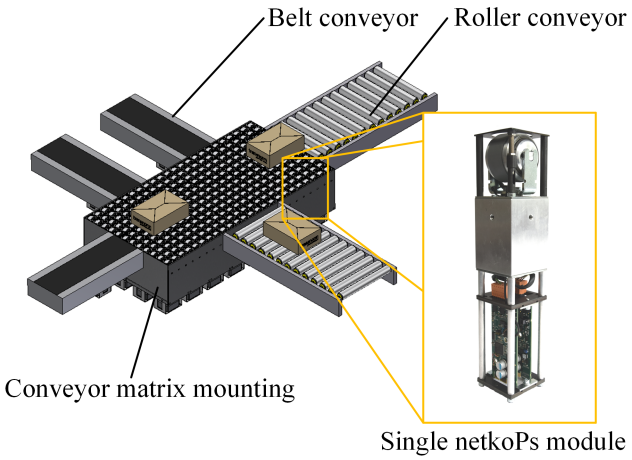
We divide modular conveying systems into two classes – large-scaled modules and small-scaled modules. If a single module can carry a package, the module is large-scaled in relation to the package size. If multiple modules are necessary to carry a package, it is small-scaled in relation to the package size. We will first present the state-of-the-art of large-scaled modules followed by small-scaled modules.

An example for a conveying system that is made up of large-scaled modules is shown in Fig. 2 – the GridSorter system. The prototypes of the GridSorter modules have been described in [2] and [4] and are now manufactured by *flexlog GmbH* [5]. The footprint of a GridSorter module is 500 mm x 500 mm allowing packages to be conveyed into all four cardinal directions. Every module has four sensors at the sides to detect incoming packages and a dedicated control. Communication is only possible with its four direct neighbors. All modules have wheels underneath and four standardized connections on every side, which are used to transmit information and power from one module to another. Due to the wheels and the standardized connections, the modules can be rearranged into a new layout within minutes. If multiple modules are combined to form a layout, no single control has an overview of the whole system. Instead, the modules exchange messages and decide the conveying of packages on their own. Thus, the modules are software-agents as defined by Franklin and Graesser [6] with decentralized and distributed controls. Large-scaled modules have also been manufactured by other companies. Two example systems are the XPlanar system [7] and the Motion Cube system [8], but no control algorithms have been published for these systems.

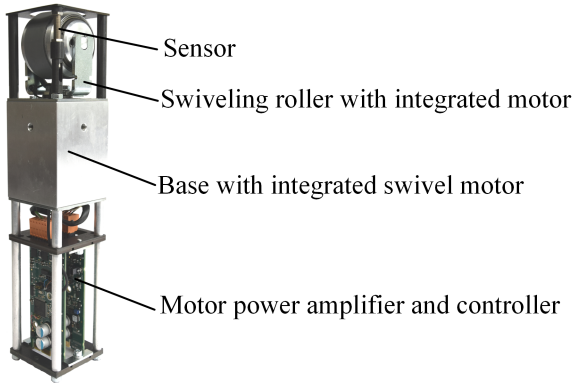
We will only consider algorithms that can potentially fulfill our requirements, which are: conflict-free, work for any bi-directional conveyor layout, and select routes based on the expected travel time. We use a classification to filter out all algorithms that do not fulfill our requirements. Different classifications are available (see [9]–[11]), but in this publication, we use the classification by Seibold [4]. Only algorithms from Seibold's class of "Time-window based Route Reservation" are able to fulfill all our requirements. Algorithms from this class reserve routes, from source modules to destination modules, for individual packages. Every module keeps a schedule, which are used to reserve time windows for the individual packages. Even though we will not consider algorithms from other classes, we still want to mention the algorithms for modular conveying systems by Gue et al. [12], [13], by Mayer et al. [2], [14] and by Krühn et al. [15], [16].

Seibold described a routing algorithm [4] that belongs to the class of "Time-window-based Route Reservation". This algorithm is conflict-free and works for any bi-directional layout, selecting routes based on the expected logical lead time. One downside of utilizing logical time is that it does not progress unless a package is conveyed from one module to the next. The route with the shortest logical lead time is not necessarily the route with the shortest physical lead time.

After presenting the algorithms for large-scaled modules, we will now present state-of-the-art small-scaled modules. An example depicting a conveying system that is made up of small-scaled modules is shown in Fig. 3 – the netkoPs system. The prototypes of the netkoPs modules are described in greater detail in [17]–[19]. The netkoPs modules are based on the cogniLog modules that have been described in [15], [16], [20]. Every netkoPs module has a sensor to detect packages, can convey in any direction, and has its own dedicated control. The footprint of a module is 60 mm x 60 mm. These modules can only communicate with their four direct neighbors and can be freely arranged into the slots of a matrix mounting. Prototypes of matrix mountings with different dimensions have been manufactured, and slots can be left unoccupied. Every matrix mounting has wheels underneath, and thus, can be freely positioned within a warehouse. Another example of a small-scaled conveyor system is the Celluveyor from Uriarte et al. [21]–[23]. None of the published control algorithms for



a) Conveyor matrix with bordering conveyors



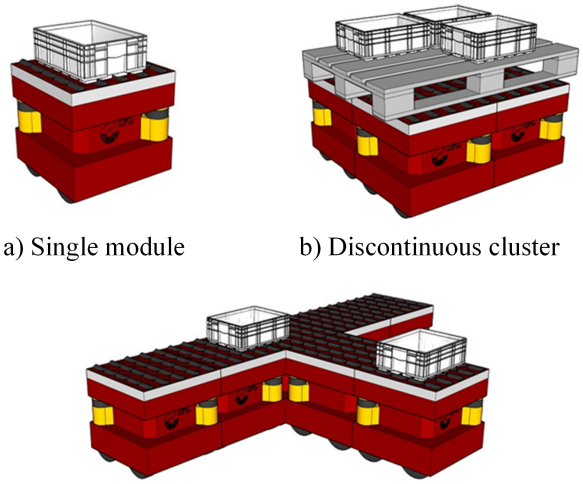
b) Detailed view of a single netkoPs module

Fig. 3. The netkoPs system is made up of several small-scaled conveying modules (based on [17])

small-scaled modules belong to the class of "Time-windows based Route Reservation", and therefore, are not presented. Krühn has shown that an algorithm that works for large-scaled modules can be modified to also work for small-scaled modules [15], [16].

B. Automated Guided Vehicles

Both the hardware and the algorithms for automated guided vehicles (AGV) are much more advanced than their modular counterparts. The following overview publications describe state-of-the-art AGV routing algorithms: [9]–[11], [24]. As with the modular conveying systems, we will only focus on algorithms from the class of "Time-window-based Route Reservation." Most algorithms from this class are based on the algorithm described by Kim and Tanchoco [25]. This algorithm is conflict-free, works for any bi-directional layout, and selects routes based on the physical travel time. Unfortunately, all routes are sequentially planned by a central control that has a complete overview of the system. After a route has been planned, it is uploaded to the respective AGV. Kim and Tanchoco even explicitly stated that the algorithm cannot be used in a decentralized controlled system: "Parallel



a) Single module b) Discontinuous cluster

c) Continuous cluster

Fig. 4. The KARIS system can operate both as a single module, as a discontinuous cluster or as a continuous cluster [28]

execution of the routeing algorithm may result in conflicting travel schedules or gridlocks in a bidirectional network." [25]

We want to highlight two publications regarding Kim and Tanchoco's algorithm. The original algorithm neglected the problem of unexpected delays that can occur in real-world systems, during transport execution. Maza and Castagna [26] presented a modification that prevents conflicts even when the previously reserved time windows are missed. This is accomplished by monitoring intersections: if a package misses its time window on the intersection, no other AGV must enter the intersection until the delayed AGV has passed the intersection.

Möhring et al. [27] tested the practicality of the algorithm for real-world use-cases. Their algorithm is implemented to route vehicles at a Container Terminal in the Hamburg Harbor. Beforehand, they simulated different scenarios with up to 48 vehicles on a computer with an AMD-Athlon 2100+ (1,7 Mhz) processor and 512 MB Ram. Even for worst-case scenarios (new routes must be calculated for all vehicles at the same time), the computation took less than half a second. We believe that these fast computation times are the reason why Kim and Tanchoco's algorithm was never modified extensively. It is already efficient enough for real-world use-cases even in its basic form.

Before we conclude the related research section, we want to mention the KARIS vehicles [28], [29]. The vehicles cannot only individually transport single items, but are also able to form two different functional clusters, as shown in Fig. 4. As a discontinuous cluster, KARIS vehicles connect to each other to transport items that are larger than a single module. As a continuous cluster, several KARIS vehicles form a conveyor line to realize high throughput of goods. To our knowledge, no detailed routing algorithms for KARIS vehicles have been published.

To summarize, we identified two algorithms that are

conflict-free and work in any bi-directional layout – Seibold's algorithm and Kim and Tanchoco's algorithm. Seibold's algorithm is decentralized, but routes are selected by using logical lead time and not physical lead time. Kim and Tanchoco's algorithm, and its modifications, use physical lead time to select routes but are designed for a centrally controlled system. As a result of our literature review, we identified the following **research question:**

Is it possible to create a decentralized routing algorithm that selects routes based on the physical lead time in any bi-directional layout while avoiding conflicts? We believe that by utilizing physical time windows we can answer this question.

III. The Algorithm

A. Clock Synchronization

A prerequisite for our algorithm is the synchronization of clocks in all conveyors. The clocks need not be synchronized perfectly, but the timing differences of neighboring conveyors must be small enough that the conveying process is not affected in a meaningful way.

The synchronization of clocks is a well-researched field [30], and many cheap technical solutions exist. We would like to point out the Precision Time Protocol (PTP), which was defined in the IEEE 1588-2008 standard [31]. PTP was designed for local systems requiring high accuracy that can not bear the cost of a GPS receiver at each node, or for which GPS signals are inaccessible [32]. It can be used within a standard ethernet-network, and thus, is cost-effective. The accuracy of PTP time signals is in the sub-microsecond range, and thus, is sufficient for the conveying of packages which rarely exceeds 10 m/s.

B. Algorithm for Large-scaled Modules

We will first present our algorithm for large-scaled modules, and subsequently, a modification for small-scaled modules. As previously mentioned in Section III-A, we assume that the clocks of every module are sufficiently synchronized. The algorithm is depicted in flowchart-form in Fig. 5, and an application example for further clarification is given in Section IV.

A short overview of the algorithm: the modules reserve a route from source to destination by exchanging messages. Every module along a route stores a routing entry on its schedule. Routing entries include an arrival and a departure time. Every route reservation has two phases: the request phase and the confirmation phase. In the request phase the route is planned from source to destination. During this phase, the modules have to select a path, and they have to negotiate the arrival and the departure times. After the request has reached the destination module, the route is confirmed from destination to source.

The presented algorithm ignores the effects of acceleration, deceleration, and slippage that occur in real-world conveying systems. All three effects can be accounted for by adding a safety allowance to the time windows. In the following paragraphs, we will not explicitly mention this safety allowance for the sake of simplicity.

Before route reservations can be created, the modules must be initialized. During this initialization, every module generates a routing table by communicating with its direct neighbors. Hereupon, this information is propagated through the system. This process is similar to creating metric tables in computer networks using the distance-vector routing protocol [33]. For every destination, the modules determine the ID of the neighboring module that is closest to this destination, the distance to the destination, and the number of intersections between the source module and the destination.

Routing entries store the following data: unique request ID, physical time window (arrival time and departure time), priority, time-to-live (TTL), ID of the preceding module, ID of the succeeding module, and state (the two states are "requested" and "confirmed"). The TTL is the maximum time a routing request may "live" before it is discarded and a new request is created by the source. Consequently, every module periodically checks the TTL for routing entries and deletes expired ones. The introduction of TTL was necessary to take into account the uncertainty of a decentralized controlled system with physical time windows. Since no module has a complete overview of the system, the source module can only estimate how long the route reservation process takes. Estimating the TTL, choosing time windows, and selecting routes are intertwined problems we will explain in detail after we overview the algorithm.

Modules can send the following four messages types which must be processed in the following sequence: OBSOLETE, DENIAL, REQUEST, CONFIRMATION. The processing of messages in this sequence positively affects the throughput. Both OBSOLETE and DENIAL messages delete routing entries on the schedule. By processing them first, the schedule opens up. Then, the REQUEST and CONFIRMATION messages are processed, which either create or change the state of routing entries. By processing the REQUEST and CONFIRMATION messages, previously unavailable time windows can be reserved, and thus, the chance of reserving a well-fitting time window is increased, positively affecting the throughput.

In the following paragraphs, we will explain how an initial request is started, how each message type is processed, how time windows are chosen, how the TTL is calculated, and how routes are selected.

Initial creation of a REQUEST message: When a package enters the conveying system on a source module, the schedule of this source module becomes blocked indefinitely until the route reservation is confirmed. This source module estimates the TTL and the desired arrival time, and subsequently, sends a REQUEST message to the neighboring module with the shortest lead time. Every REQUEST message includes a unique request ID, the priority, the destination ID, the desired arrival time, and the TTL.

Receiving a REQUEST message: When a module receives a REQUEST message, it tries to reserve an entry with the requested arrival time. If the newly created time window does not overlap with any previously created time window, an entry is created on the schedule, and a REQUEST message is sent to the neighboring module with the shortest lead time. Livelocks are prevented by excluding the possibility of sending

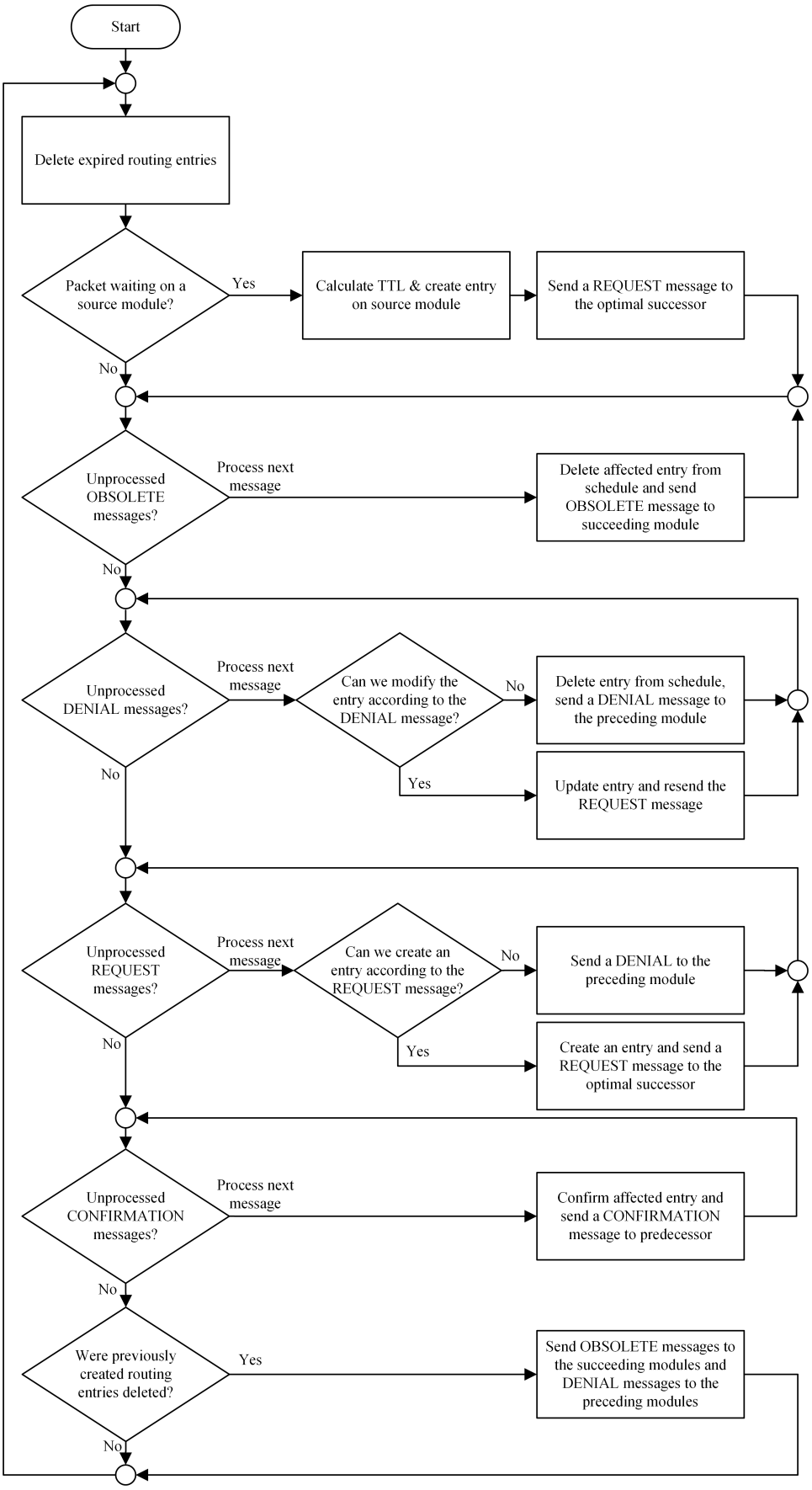


Fig. 5. Flowchart of the algorithm

a REQUEST message to the predecessor.

When a REQUEST message reaches the destination module, a CONFIRMATION message is sent back to the requesting module.

If a module receives a REQUEST message with an arrival time that overlaps with a previously created routing entry, it first checks the state of the previously created entry. If the previously created entry is already in the state confirmed, the module determines the next available time window and sends back a DENIAL message to the requesting module. If the previously created entry is in the state requested, the module compares the priority of the REQUEST message with the previously created entry. If the priority of the REQUEST message is lower, a DENIAL message is sent to the requesting module. If the priority of the REQUEST message is higher, the previously created time window is deleted, and a new routing entry is created. A DENIAL message is sent to the preceding module of the deleted routing entry, and an OBSOLETE message is sent to the succeeding module.

Receiving a CONFIRMATION message: When a module receives a CONFIRMATION message, it changes the state of the corresponding routing entry to the state confirmed and sends a CONFIRMATION message to its preceding module. When a CONFIRMATION message reaches the source module, the actual conveying of the package can start.

Receiving a DENIAL message: A DENIAL message includes the unique request ID and the proposed time. When a DENIAL message is received, the receiving module attempts to alter the departure time of the corresponding routing entry. If this altered entry does not overlap with other entries, a REQUEST message is sent to the succeeding module. If overlap occurs, the same overlap resolution is used when receiving a REQUEST message.

Receiving an OBSOLETE message: When a module receives an OBSOLETE message, the corresponding routing entry on is deleted even when it is already in the state confirmed.

Choosing time windows: The goal of the algorithm is to choose time windows that are as short as possible (as soon as a package leaves a module, another one can enter). If the time windows are as short as possible, the system utilization is higher, and thus, the throughput is positively affected. The length of the time windows is determined by arrival and departure time. The arrival time is based on the arrival of the package: when the package enters, the source is blocked indefinitely until the route reservation process has been completed. The blockage of the source does not negatively affect the throughput, because we only allow sources to exist at the border of our system (only one neighboring module is allowed).

The reservation of the time window on the succeeding module is more complex: when the source sends a REQUEST message, it must include a desired arrival time. This arrival time must be far enough into the future that the route reservation process is most likely completed by then. The completion time depends on the number of modules between the source and the destination and the number of overlaps that will likely occur, which need to be resolved. Since the source has no complete system overview, it can only estimate the number of overlaps. Because the number of overlaps can only be estimated, the arrival time can only be estimated.

When a module receives a REQUEST message, it computes the earliest possible departure time, based on the arrival time and the conveying speed of the module. The routing entry is created, and a new REQUEST message is sent to the succeeding module. The arrival time of this new REQUEST message is based on the departure time that has just been computed. If the succeeding module sends a DENIAL message back, the departure time is altered to the proposed time of the DENIAL message. If this altered time window overlaps with other time windows, the overlap is resolved according to the previously presented overlap resolution method.

If more overlaps occur than estimated, the CONFIRMATION message will reach the source after the estimated arrival time. Thus, it would become physically impossible for the package to arrive at the estimated arrival time. Because the first time window is missed, all subsequent time windows may also be missed. A mismatch would have occurred between the projected movement of the package and the actual movement. As already mentioned in Section II-B Maza and Castagna [26] have proven that even when a mismatch occurs, conflicts can still be avoided by following the sequence of the previously negotiated time windows.

Even though no conflict occurs due to the mismatch, it is still advantageous to avoid such a mismatch because the selection of routes depends on accurate schedules. The selection of routes is based on the lead time, which itself, is based on the projected movement. If the mismatch becomes too great, sub-optimal routes might be chosen.

To avoid a mismatch between projected movement and actual movement, we introduce the concept of TTL. Before a source sends the first REQUEST message, it estimates how long the route reservation will most likely take. A safety margin is added to increase the chance of completing the route reservation within the TTL. If the TTL is estimated too conservatively and the route reservation is confirmed faster than estimated, the packet *cannot* start earlier, as this would also cause a mismatch between projected movement and actual movement. Therefore, the packet has to wait, and thus, the throughput is negatively affected. If the estimate is too optimistic and the route reservation is not confirmed within the TTL, the source module has to start a new request. Since the packet has to wait until this new request has been confirmed, the throughput is once again negatively affected. Thus, the next paragraphs, we present our approaches for estimating the TTL.

Estimating the TTL: We developed two approaches for estimating the TTL. If the operating conditions remain approximately constant, the first approach estimates the TTL optimally. Operating conditions in this context are the rates at which packages enter and leave the system. When using the first approach, the TTL is calculated as follows:

$$TTL = t_{avg,dest} \cdot n_{safe} \tag{1}$$

Every source module stores how long each route reservation took to reach its destination, and an average is calculated. This

average is multiplied by a safety factor – n_{safe}. This safety factor takes into account that the operating conditions remain only *approximately* constant. While testing the first approach on different layouts, we achieved satisfactory throughputs by picking a value in a range from 1 to 2 for n_{safe}. If the operating conditions are constant, n_{safe} can be set to 1. Since route reservations which were not confirmed within the TTL, increase the value of $t_{avg,dest}$, this approach is self-correcting.

If the layout of the conveying system changes, all previously calculated averages become invalid. Since packages still need to be conveyed during this start-up phase, a second approach for estimating the TTL is needed:

$$TTL = n_{module} \cdot t_{com} \cdot 2 + n_{inter} \cdot n_{denial} \cdot t_{com} \quad (2)$$

The first term $n_{module} \cdot t_{com} \cdot 2$ takes into account the communication delay between source and destination module: n_{module} is the number of modules between source and destination and t_{com} is the time for one module to receive and process a message. The 2 is needed because all modules along a route have to receive and process at least two messages. First, a REQUEST message and a subsequent CONFIRMATION message. The second term $n_{inter} \cdot n_{denial} \cdot t_{com}$ takes into account the possibility that an intersection rejects the first REQUEST message by sending a DENIAL message. n_{inter} is the number of intersections between source and destination and n_{denial} is the number of denials that are likely sent before the request is accepted. When we tested our algorithm on rectangular layouts, we discovered that $n_{denial} = 2$ leads to satisfactory throughputs.

Selecting a route: Routes are selected based on the physical lead time. The base lead time for every neighboring module is computed by dividing the destination's distance (which was determined during the initialization phase) by the uniform conveying speed of the modules. If a module received a DENIAL message beforehand, it adds the proposed delay to the base lead time. Thus, the modules do not select the neighboring module with the shortest distance to the destination, but the neighboring module with the shortest projected lead time. Livelocks are avoided by never choosing the preceding module, even if it has the lowest lead time.

C. Modified Algorithm for Small-scaled Modules

When the algorithm is modified for small-scaled modules, the concept of module neighborhoods is used [15]: the REQUEST message must be extended by adding the package dimensions. If a module receives a REQUEST message, it becomes the master module of the neighborhood for this time step. The master module uses the package dimensions from the REQUEST message to check the schedules of every slave module within the neighborhood. If every module within the neighborhood can reserve a time window for the REQUEST, a REQUEST message is sent to the optimal neighbor by the master module. If a single module within the neighborhood cannot accept the routing entry, a DENIAL message is sent to the preceding module. The process is depicted in Fig. 6 for a package that has the dimensions of 3 x 3 modules. Successfully reserved neighborhoods are shown for t=10 and t=11.

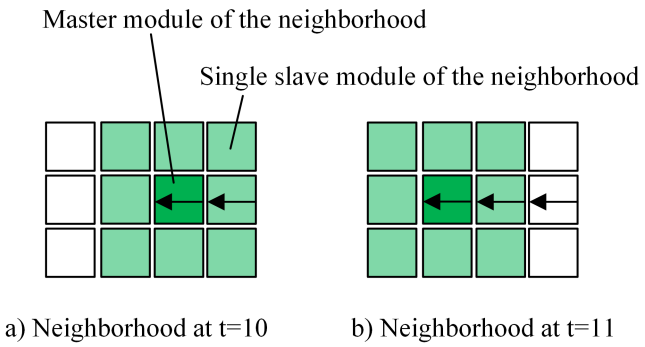


Fig. 6. Neighborhoods during route reservation (based on [15])

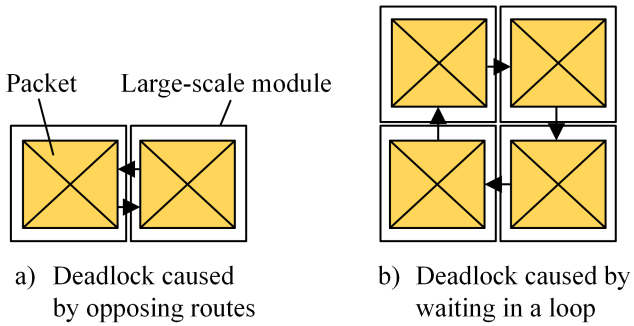


Fig. 7. Deadlock situations that must not occur [4]

D. Proof of Conflict Absence

Conflicts include collisions, livelocks, starvation effects and deadlocks. We avoid livelocks by omitting the requesting module, when searching for a new optimal module. Starvation is avoided by partly basing the priority of requests on the creation time. In the following paragraphs, we will prove the absence of collisions and deadlocks. Seibold identified two possible deadlock situations that can occur in large-scaled modular conveying systems [4]: deadlocks caused by opposing routes and deadlocks caused by packages waiting in a loop (see Fig. 7). In both cases, the packages are deadlocked because the time window on the succeeding module overlaps with the time window of another package. Thus, deadlocks and collisions are avoided by guaranteeing that no time windows overlap. Since the arrival and departure times of packages on every module are solely determined by either accepting or denying REQUEST messages, we must only examine this phase. The following proof works for both small-scaled and large-scaled modules. In the case of large-scaled modules, only the schedule of the neighboring module must be checked for overlaps. In the case of small-scaled modules, an additional step is needed: When a module receives a REQUEST message, it must check the schedules of all modules within the neighborhood before either a REQUEST or a DENIAL can be sent (see also Section III-C). To keep this publication concise, we omit this additional step in the following paragraphs. We first prove the absence of overlapping time windows for the case of opposing routes by proof of exhaustion. The following cases can occur:

1. Both entries are in the state requested: Both modules

send REQUEST messages to the other module. The requested time window will overlap on both modules with the already existing routing entries. Therefore both modules will compare the priority of the entry with the priority of the REQUEST message and delete the one with the lower priority. As a result, both modules will end up with only the entry for the package with the higher priority and thus no deadlock occurs.

2. One entry is in the state confirmed, the other is in the state requested: Only the module with the requested entry sends a message. The module with the already confirmed entry receives the REQUEST, but immediately sends back a DENIAL, because already confirmed requests must not be deleted by a REQUEST message (even if it has a higher priority). Thus, no deadlock occurs.

3. Both entries are in the state confirmed: This case cannot occur, and thus, no deadlock occurs. Prior to having confirmed entries, every entry has to be in the requested state. If both neighboring modules were in the state requested at the same time, the rules for case 1 avoid the creation of two opposing entries with the state confirmed. If one of the entries was in the state confirmed and the other was in the state requested, the rules from case 2 avoid the creation of two opposing entries with the state confirmed.

In contrast to deadlocks caused by opposing routes, deadlocks in loops are caused by non-opposing routes. Once again, we will prove the absence of overlapping time windows by proof of exhaustion. In the following, we examine a module in a loop that receives a request from a module outside the loop. The module within the loop can be in two different states:

1. The entry of the receiving module is in the state confirmed: The receiving module rejects the REQUEST and sends back a DENIAL with a new proposed time, and thus, no deadlock occurs.

2. The entry of the receiving module is in the state requested: The module that receives the request compares the priorities of the REQUEST message with the priority of the routing entry, deleting those with lower priority. Thus, no deadlock occurs.

IV. Application Example

In the following figures, the system behavior is shown for large-scaled modules. In every figure, the layout is shown on the top, and the schedules of the modules are shown on the bottom. No module has this complete overview of the system - every module can only access their own schedules.

The layout consists of five modules. The unique identification number (ID) of every module is in their top left corner. Two source modules exist in this layout: source I has the ID 2 and is at the left-most of the layout, and source II has the ID 5 and is at the bottom-most of the layout. There is only one destination module in the layout: it has the ID 1 and is at the top-most of the layout. Routing entries that are in the state requested are displayed as a single arrow on a conveyor. Routing entries in the state confirmed are displayed with two arrows on the conveyor. Routing entries are assigned to their packages in two ways: the first arrow originates at their respective package, and the arrows have the same color as the packages.

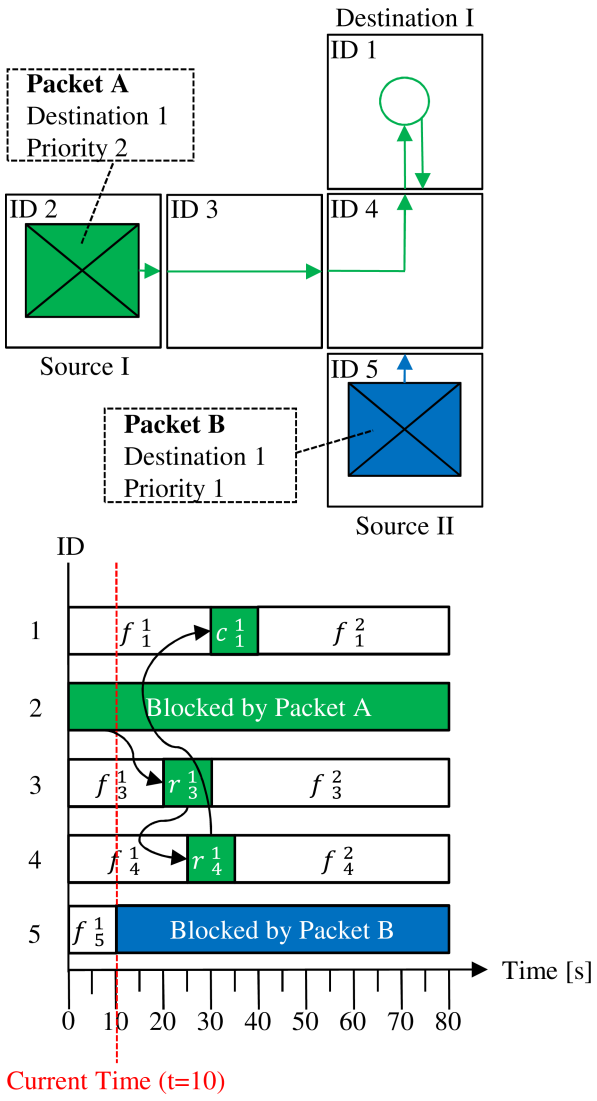


Fig. 8. Application example at t=10

The schedules of all modules are depicted on the right side: the horizontal axis displays the time, while the schedules of the individual modules are arranged on the vertical axis. The states of the time windows are denoted by one of three possible letters: f meaning the time window is not assigned to any routing entry, r meaning the time window is assigned to a routing entry that is in the state requested and c meaning the time windows is assigned to a routing entry that is in the state confirmed. Every time window has two indexes: the lower index is used to denote the module ID, and the upper index is used to number the time windows.

In this application example, the communication speed is unrealistically slow - sending a message from one module to another module takes one full second. We have chosen to use this slow communication speed in this example for two reasons. Firstly, the conveying speed and the communication speed are now within the same magnitude, which makes it possible to display both in the same figures. Secondly, we wanted to show that even when the communication speed becomes low, the algorithm still works.

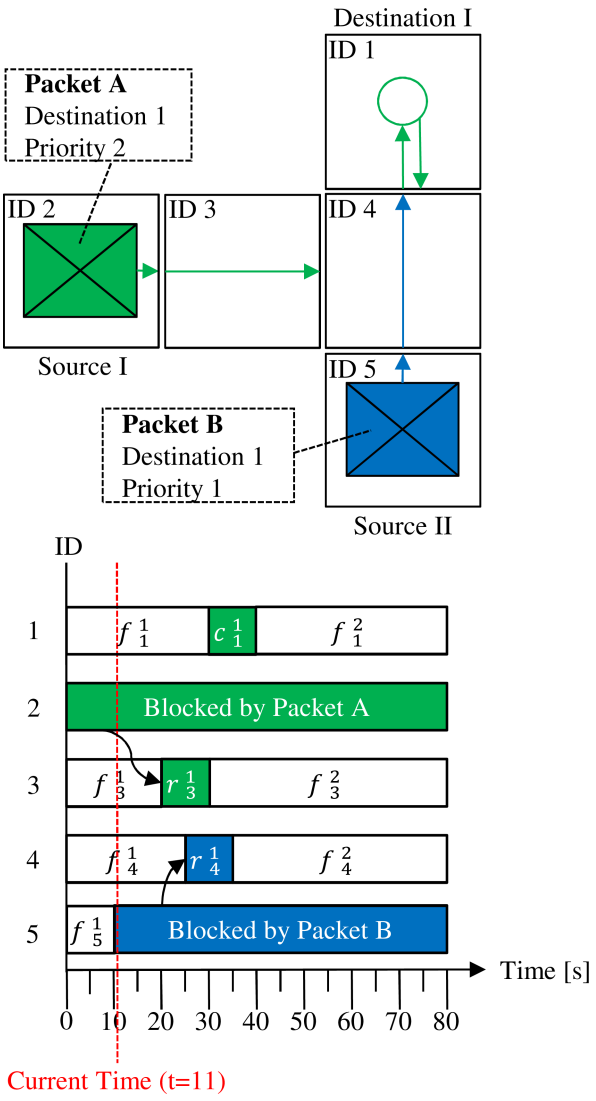


Fig. 9. Application example at t=11

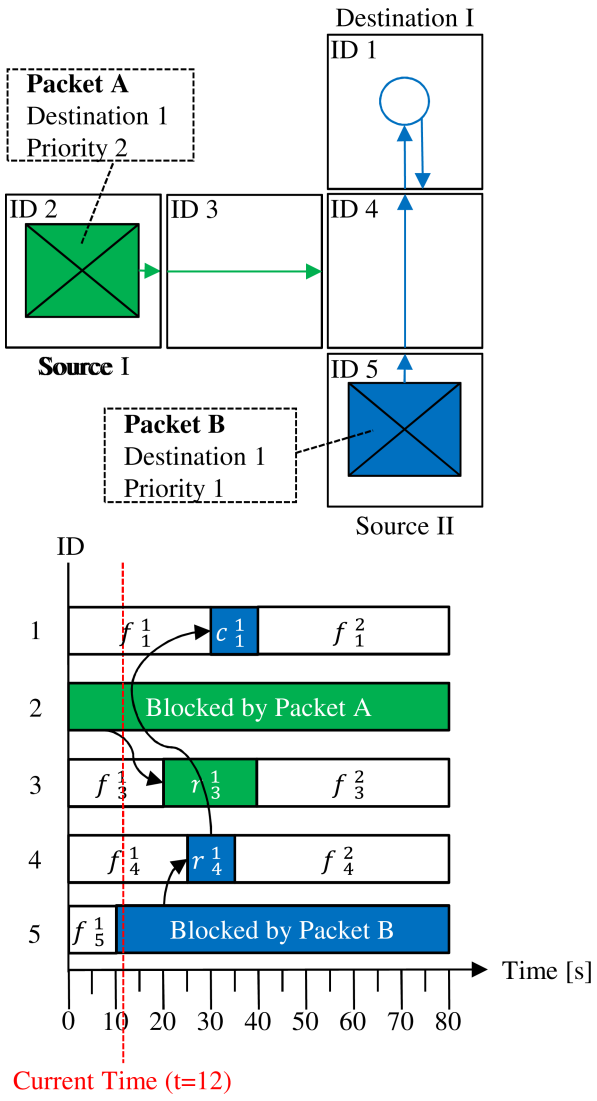


Fig. 10. Application example at t=12

At t=10, package A has entered the system through source I, and its request has reached destination I, which sends back a CONFIRMATION message to module 4. At the same time, package B enters the system through source II. A REQUEST message for package B is sent to module 4. Package B has a higher priority than package A.

At t=11, module 4 receives both the CONFIRMATION message and the REQUEST message. The REQUEST message is processed first: since the requested arrival times for package A and package B overlap, the module compares the priorities, and subsequently, deletes the routing entry for package A before creating a new entry for package B. This causes the creation of the following three messages. Firstly, an OBSOLETE message is sent to module 1 to inform it that the already confirmed routing entry for package A must be deleted. Secondly, a DENIAL message is sent to module 3 with a new proposed arrival time. Finally, a new REQUEST message for package B is sent to module 1. After processing the REQUEST message, the CONFIRMATION message is processed. Since the routing entry for package A no longer exists, the CONFIRMATION message is discarded.

At t=12, module 1 receives the OBSOLETE message for package A and the REQUEST message for package B. The reception of the OBSOLETE message causes the deletion of the previously confirmed time window for package A. Caused by the reception of the REQUEST message, module 1 creates a routing entry for package A. This routing entry is in the state confirmed, because the request has reached its destination. Subsequently, a CONFIRMATION message is sent back to module 4. At the same time, module 3 receives the DENIAL message from module 4, and subsequently, alters the time windows for the routing entry for package A. A new REQUEST message with an altered arrival time is sent to module 4.

At t=13, module 4 receives two messages: the CONFIRMATION message for package B and the new REQUEST message for package A. Since the requested arrival time for package A does not overlap with the time window of package B, a routing entry for package A is created. Subsequently, a REQUEST message is sent to module 1. Next, the CONFIRMATON mes-

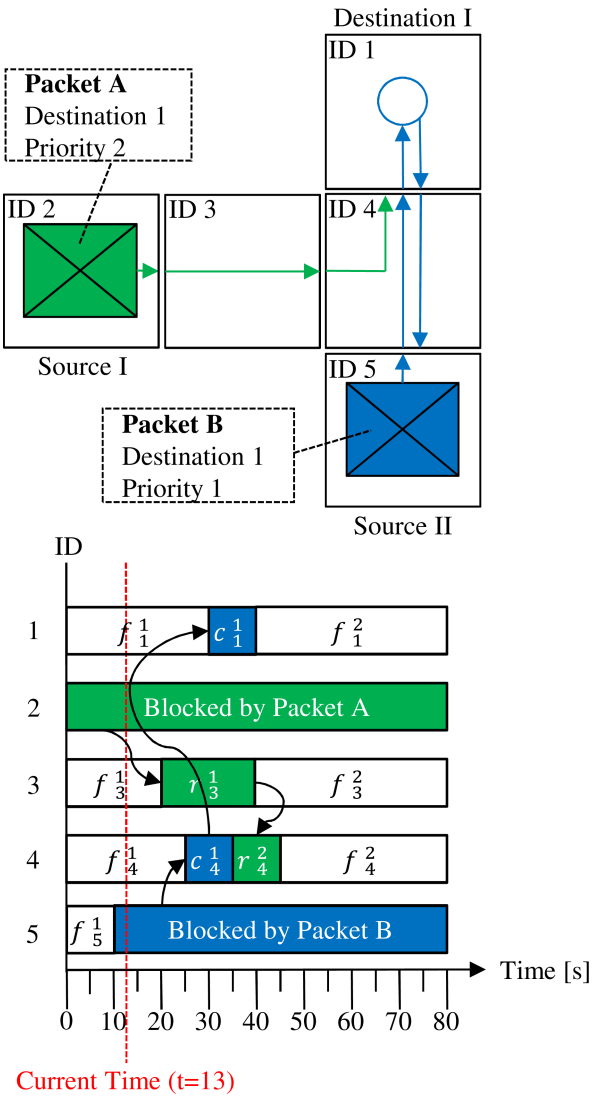


Fig. 11. Application example at t=13

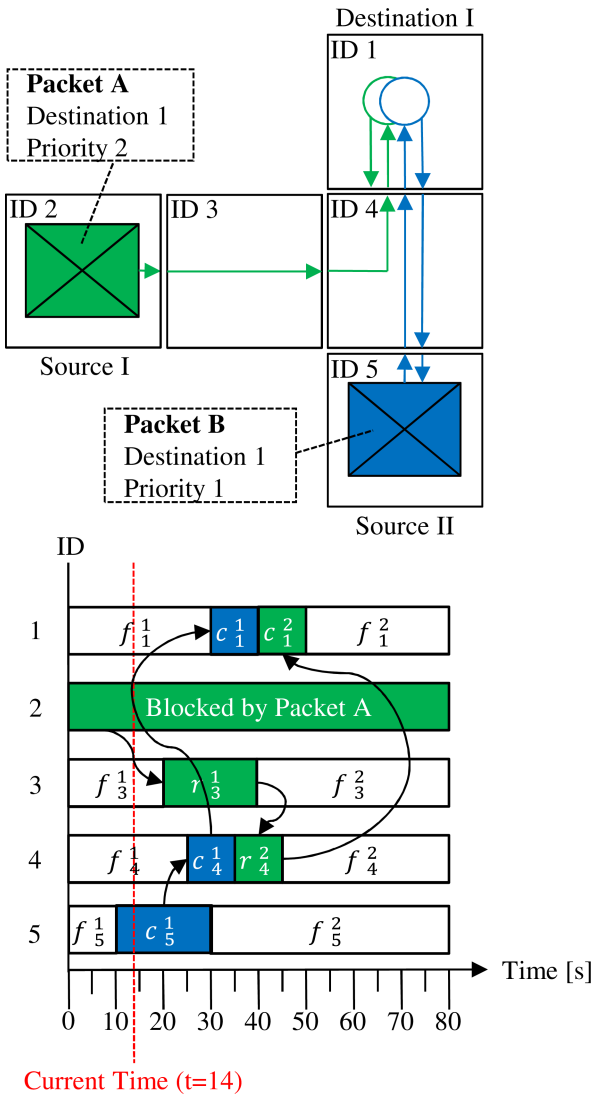


Fig. 12. Application example at t=14

sage is processed: the state for the routing entry for package B is set to confirmed, and a CONFIRMATION message is sent to module 5.

At t=14, module 1 receives a REQUEST message for package A: it checks its schedule and creates a routing entry for package A. This routing entry is in the state confirmed, because the request has reached its destination. Subsequently, a CONFIRMATION message is sent back to module 4. At the same time, module 5 receives the CONFIRMATION message for package B. The state of the corresponding is altered to confirmed and the reservation process for package B has successfully ended. The schedule of module 5 is no longer blocked indefinitely by package B.

To show the effects of a sub-optimal calculation of the TTL, the TTL is calculated conservatively in this example. Every source module calculates the TTL by adding 10 seconds to the creation time of a request. Because of this sub-optimal TTL calculation, package B has to wait for 6 more seconds after the CONFIRMATION message has reached source module 2 before the conveying can start.

V. Conclusion and Outlook

In this publication, we have presented an algorithm that can be used for both small-scaled and large-scaled modular conveyors. The algorithm fulfills the flexibility properties that have been defined by Furmans et al. [1]. We have also proven the absence of any conflicts in our algorithm. Unlike all previously published algorithms for modular conveyors, physical lead time is used to select routes.

Two important aspects of our algorithm still need to be investigated further: the first aspect is determining the optimal operating parameters for different layouts, and the second is conducting a throughput analysis. Since both aspects are intertwined, the investigation of the optimal operating parameters must be conducted first. Two parameters must be investigated: n_{denial} as defined in formula 2 and n_{safe} as defined in formula 1. Both parameters can be determined by simulating different layouts. The following parameters will have the biggest influence on n_{denial} and n_{safe}: number of modules within the system, number of source modules, number of intersections, and package arrival rate at source

modules. After determining optimal operating parameters, a throughput analysis must be conducted. We are currently investigating both aspects and will publish the results at a later time.

ACKNOWLEDGMENT

This publication was done within the scope of the research project "Vernetzte, kognitive Produktionssysteme (netkoPs)", which was funded by the German Federal Ministry of Education and Research. The authors would like to thank the staff of the Logistics and Distribution Institute (LoDI) at the University of Louisville for their constructive comments on the manuscript.

REFERENCES

[1] K. Furmans, F. Schönung, and K. R. Gue, "Plug-and-work material handling systems," *Progress in Material Handling Research*, pp. 132–142, 2010. [Online]. Available: https://digitalcommons.georgiasouthern.edu/pmhr_2010/1

[2] S. H. Mayer, "Development of a completely decentralized control system for modular continuous conveyor systems," Dissertation, Karlsruhe Institute of Technology, Karlsruhe, 17.06.2016. [Online]. Available: https://publikationen.bibliothek.kit.edu/1000011463

[3] M. Schwab, "A decentralized control strategy for high density material flow systems with automated guided vehicles," Dissertation, Karlsruher Institut für Technologie (KIT), Karlsruhe, 24.04.2015. [Online]. Available: http://dx.doi.org/10.5445/KSP/1000047227

[4] Z. Seibold, "Logical time for decentralized control of material handling systems," Dissertation, Karlsruher Institut für Technologie (KIT), Karlsruhe, 2016. [Online]. Available: http://dx.doi.org/10.5445/KSP/1000057838

[5] flexlog GmbH, "Dezentral steuerbar – Modulbaukasten mit FlexTechnology," 2019. [Online]. Available: www.flexlog.de/

[6] S. Franklin and A. Graesser, "Is It an agent, or just a program?: A taxonomy for autonomous agents," in *Intelligent Agents III Agent Theories, Architectures, and Languages*, J. P. Müller, M. J. Wooldridge, and N. R. Jennings, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 1997, pp. 21–35.

[7] Beckhoff Automation GmbH & Co. KG, "Flying Motion: XPlanar," 2019. [Online]. Available: https://www.beckhoff.de/xplanar/

[8] WEKA BUSINESS MEDIEN GmbH, "Festo - Palettiersystem Motion Cube," 2017. [Online]. Available: https://www.handling.de/video---festo---palettiersystem-motion-cube.htm

[9] I. F. Vis, "Survey of research in the design and control of automated guided vehicle systems," *European Journal of Operational Research*, Vol. 170, No. 3, pp. 677–709, 2006.

[10] T. Le-Anh and M. de Koster, "A review of design and control of automated guided vehicle systems," *European Journal of Operational Research*, Vol. 171, No. 1, pp. 1–23, 2006.

[11] L. Qiu, W.-J. Hsu, S.-Y. Huang, and H. Wang, "Scheduling and routing algorithms for AGVs a survey," *International journal of production research : American Institute of Industrial Engineers ; Society of Manufacturing Engineers*, Vol. 40, No. 3, pp. 745–760, 2002.

[12] O. Uludağ, "GridPick: A High Density Puzzle Based Order Picking System with Decentralized Control," Dissertation, Auburn University, Auburn, Alabama, USA, 2014. [Online]. Available: https://etd.auburn.edu/handle/10415/3984

[13] K. R. Gue, K. Furmans, Z. Seibold, and O. Uludağ, "GridStore: A Puzzle-Based Storage System With Decentralized Control," *IEEE Transactions on Automation Science and Engineering*, Vol. 11, No. 2, pp. 429–438, 2014.

[14] S. Mayer and K. Furmans, "Deadlock prevention in a completely decentralized controlled materials flow systems," *Logistics Research*, Vol. 2, No. 3-4, pp. 147–158, 2010.

[15] T. Krühn, "Dezentrale, verteilte Steuerung flächiger Fördersysteme für den innerbetrieblichen Materialfluss," Dissertation, Leibniz Universität Hannover, Hannover, 16.04.2015. [Online]. Available: http://www.tewiss-verlag.de/katalog/details/?isbn=978-3-95900-014-7

[16] T. Krühn, S. Sohrt, and L. Overmeyer, "Mechanical feasibility and decentralized control algorithms of small-scale, multi-directional transport modules," *Logistics Research*, Vol. 9, No. 1, p. 147, 2016. [Online]. Available: https://link.springer.com/article/10.1007/s12159-016-0143-x

[17] L. Overmeyer and H. Stichweh, *Vernetzte, kognitive Produktionssysteme: Abschlussbericht*, ser. Berichte des TEWISS Verlags. Garbsen: TEWISS - Technik und Wissen GmbH, 12.12.2017.

[18] H. Stichweh, M. Theßeling, S. Sohrt, A. Heinke, and L. Overmeyer, "Intelligent routen, fördern und verteilen: Die Conveyor Matrix für die kognitive Produktion der Zukunft." *25. Deutscher Materialfluss-Kongress mit VDI-Konferenz Routenzugsysteme, TU München, Garching, 17. und 18. März 2016*, pp. 127–142, 2016.

[19] S. Sohrt, N. Shchekutin, and L. Overmeyer, "Steuerung von kleinskaligen Fördermodulen," *Logistics Journal Proceedings*, Vol. 2016, No. 10, 2016. [Online]. Available: http://nbn-resolving.de/urn:nbn:de:0009-14-44516

[20] K.-U. Ventz, "Beitrag zur innovativen Gestaltung von Intralogistik durch Kopplung kleinskaliger Systeme," Dissertation, Leibniz Universität Hannover, Garbsen, 2016. [Online]. Available: http://www.tewiss-verlag.de/katalog/details/?isbn=978-3-95900-083-3

[21] C. Uriarte, A.-K. Rohde, and S. Kunaschk, "Celluveyor - Ein hochflexibles und modulares Förder- und Positioniersystem auf Basis omnidirektionaler Antriebstechnik," *18. Magdeburger Logistiktage - Sichere und nachhaltige Logistik*, 2013.

[22] C. Uriarte, H. Thamer, and M. Freitag, "Fördertechnik aus der Zelle," *Hebezeuge und Fördermittel 10*, 2015.

[23] H. Thamer, C. Uriarte, and A. Y. Benggolo, "Intelligente Förderzelle: Start-up aus Bremen entwickelt flexibel nutzbare und raumsparende Module," *Hebezeuge und Fördermittel*, Vol. 1-2, No. ISSN: 0017-9442, pp. 56–59, 2018.

[24] K. C. T. Vivaldini, L. F. Rocha, M. Becker, and A. P. Moreira, "Comprehensive Review of the Dispatching, Scheduling and Routing of AGVs," *A.P. Moreira et al. (eds.), CONTROLO'2014 - Proc. of the 11th Port. Conf. on Autom. Control,*, 2015.

[25] C. W. Kim and J. M. A. Tanchoco, "Conflict-free shortest-time bidirectional AGV routeing," *The International Journal of Production Research*, Vol. 29, No. 12, pp. 2377–2391, 1991.

[26] S. Maza, P. Castagna, and IEEE, "Conflict-free AGV routing in bidirectional network," in *ETFA 2001: 8TH IEEE INTERNATIONAL CONFERENCE ON EMERGING TECHNOLOGIES AND FACTORY AUTOMATION, VOL 2, PROCEEDINGS*, 2001, pp. 761–764.

[27] R. H. Möhring, E. Köhler, E. Gawrilow, and B. Stenzel, "Conflict-free Real-time AGV Routing," *Operations Research Proceedings 2004*, 2005.

[28] T. Stoll, "Dezentral gesteuerter Aufbau von Stetigförderern mittels autonomer Materialflusselemente," Dissertation, Karlsruher Institut für Technologie (KIT), Karlsruhe, 02.05.2012. [Online]. Available: http://dx.doi.org/10.5445/KSP/1000028697

[29] Institut für Fördertechnik und Logistiksysteme, "KARIS PRO – Autonomer Materialtransport für flexible Intralogistik," Karlsruhe. [Online]. Available: http://karispro.de/Abschlussbericht%20KARIS%20PRO.pdf

[30] A. S. Tanenbaum and M. van Steen, *Distributed systems: Principles and paradigms*, 2nd ed. Leiden: Maarten van Steen, 2016.

[31] IEEE, "IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems," 2008. [Online]. Available: https://standards.ieee.org/standard/1588-2008.html

[32] John C. Eidson, *Measurement, Control, and Communication Using IEEE 1588*. Berlin/Heidelberg: Springer-Verlag, 2006.

[33] A. S. Tanenbaum and D. Wetherall, *Computer networks*, 5th ed. Boston, Mass.: Pearson, 2011.

Simon Sohrt received his master's degree in mechanical engineering from the Leibniz University Hannover, Hannover, Germany, in 2013.

He is currently working towards the Ph.D. degree at the Institute for Transport and Automation Technology, Leibniz University Hannover, since 2014. His current research interests include transport technology, multi-agent simulations, control theory and warehousing.

Ludger Overmeyer received the Dipl.-Ing. degree in electrical engineering from the Leibniz University Hannover, Hannover, Germany, in 1991.

He was a Research Associate and the Department Leader of Laser Zentrum, Hanover, from 1991 to 1997. He was the Leader for research and development with Mühlbauer AG, Roding, Germany. Since 2002, he has been a Professor and the Chair of the Institute of Transport and Automation Technology with the Leibniz University Hannover. He is currently the executive director of the Institute of Integrated Production GmbH (IPH) and also a member of the executive board of the Laser Zentrum Hannover. He has authored or coauthored over 100 conference papers, journal papers, and book chapters. His current research interests include transport technology, automation technology, laser materials processing, optoelectronic packaging, and planar optronic sensor systems.