A High-Density, Puzzle-Based System for Rail-Rail Container Transfers

Kevin R. Gue and Gang Hao

Department of Industrial Engineering
University of Louisville, Louisville, KY, USA

Abstract

We describe a high-density, puzzle-based storage and transfer system for containers in a rail-to-rail hub for the Physical Internet. The system uses a new algorithm called GridHub, which is able to transfer items in all four cardinal directions simultaneously within a grid. We show how the GridHub system might be used in a rail-rail transfer hub to transfer containers between one side of the grid and a train.

1 Introduction

The Physical Internet (PI or π) is a new concept in global logistics that, among other things, specifies standards for container sizes and protocols for transfers at hubs [12]. So-called π-containers are transferred between vehicles (π-movers) via π-hubs, which are similar metaphorically to routers of the digital internet [1, 11]: π-containers (data packages) are unloaded from inbound vehicles (data senders) and moved to outbound vehicles with different destinations (data receivers). Because π-containers are handled several times in multiple π-hubs before arriving at their destinations, quick handling at hubs is essential to a plausible, future Physical Internet.

We consider the case of a rail-to-rail (rail-rail) hub at which trains offload containers for transfer to future, arriving trains, and load containers from trains that arrived earlier. The system we described is based on a conceptual design presented in Ballot et al. [1]: trains arrive to a grid of conveyor modules (one side only) that spans the length of two rail cars. Containers are offloaded from one car, while the second, newly emptied car loads new containers. The internal workings of the grid are similar to the GridStore system [7].

2 Literature Review

The most widely used method of transferring containers between trains in railway hubs is by cranes. Several studies have been done in this area, such as Bostel and Dejax [2], Boysen et al. [4], and a survey by Boysen et al. [3]. Because the quantity of cranes in these papers is limited to a very low level to avoid conflicts, performance of container stations that operate in this way is usually low. Furthermore, this conventional method

of controlling cranes employs complex computational models, so solving these models is itself a challenge, making their use in real world settings questionable.

Recent advances in high density storage systems [6, 9] provide a theoretical basis for rail-rail transfer systems, including π-hubs. The term "GridFlow" was coined as a way to refer to material handling systems using a grid of conveyor modules that pass items between them [13]. GridFlow systems have very high density because there is no wasted space devoted to pre-defined aisles, and yet the ability of each storage location (conveyor module) to move items independently also allows a very high throughput [7].

Two approaches to material movement within a grid have emerged. In the first approach, the travel path of an item entering the grid is completely or partially specified after a series of decentralized negotiations processed by the conveyors prior to the actual movements. This method was pioneered by Mayer [10] with the Flexconveyor, which uses decentralized control to convey items in a network of conveyor modules. This method was extended by Seibold et al. [14] with the GridSorter system.

A second approach routes items in an ad hoc, decentralized manner based on the current conditions of conveyor module and its immediate neighbors. This approach was introduced by Gue et al. [7] in a storage and retrieval system, and later extended by Uludag [15] and Gue et al. [8] in a picking and item sequencing system. The case of encountering broken conveyors in grid-based systems was studied by Furmans et al. [5]. Conveyors in all of these systems use an access-negotiation-convey protocol to execute movement in the system.

3 GridHub

The authors are developing the GridHub system, which will be fully described elsewhere, in order to address a number of weaknesses of systems in the existing literature:

- Movement of requested items in existing systems (GridStore, GridPick, GridSequence) is only north and south, never east and west. Consequently, only two sides of a grid have been available to receive or deliver items.
- Existing systems are not able to deliver a requested item to a specified module on the perimeter of the grid, only to a specified *side* of the grid.
- Existing systems are not able to sequence retrievals to a specified boundary module. (GridPick does sequence deliveries by *batch* or order, but not to a specified module).

The present paper presents a limited version of GridHub, in which containers are received and delivered from and to the bottom of the grid only, and in which containers flow up from bottom-left, then right to the shipping side, then down to the bottom-right face, where they are loaded into specified slots on specified cars.

The GridHub system has three key components: a group of conveyors in a grid layout with a central controller, railway cars, and π-containers (Figure 1). Railway cars stop along

the bottom row of conveyors to load and unload containers. The row of conveyors closest to the railway cars is the loading and unloading row. Other conveyors form the storage area, and all of the containers are stored and transferred on these conveyors. Since the system is grid shaped, the location of a conveyor corresponds to its column and row, and we use the columns and the rows of conveyors in the rest of this paper to describe groups of conveyors.

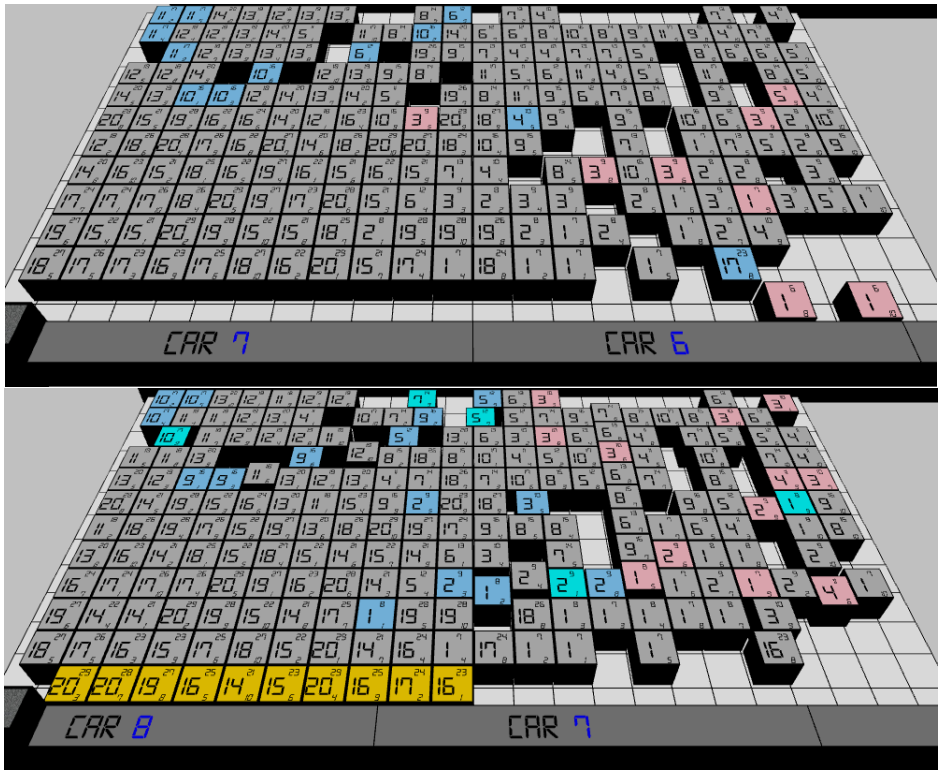


Figure 1: Overview of GridHub

Railway cars are numbered ascending from 1 according to their arrival sequence. In each railway car, there are slots in which to place π-containers, and each slot can hold only one π-container. Slots in one railway car are numbered from the left to the right side of the car starting with number 1.

Every π-container has departure information, which consists of two parts: the slot in which it should be loaded in its destination car (*slot number*) and the rail car onto which it will be loaded (*car number*). π-containers are sorted into different groups according to their slot numbers, and all containers in a same group have the same slot number. Each group of containers is sorted by car number to form a "virtual queue." The *queue number* of the container indicates the position of that container in its virtual queue. In brief, this number of one container is generated according to its car number. In other words, a container with a smaller car number consequently has a smaller queue number. When containers leave or enter the system, the grouping and sorting of containers will be performed, hence the queue

numbers will be changed several times during the period when the containers stay in the system.

Figure 2 illustrates an example for containers with departure information displayed. In the farthest right column of containers, the farthest bottom container is scheduled to be loaded into the 9th slot (shown by the small text in the bottom side of the box) of car number 10 (shown by the small text in the top side of the box), and it is the 1st position (the big text) of the virtual queue formed by containers, and all containers in this queue will be loaded into the 9th slot in all future railway cars. After this container leaves the system, the container with car number 11 and slot number 9 changes its queue number from 2 to 1.

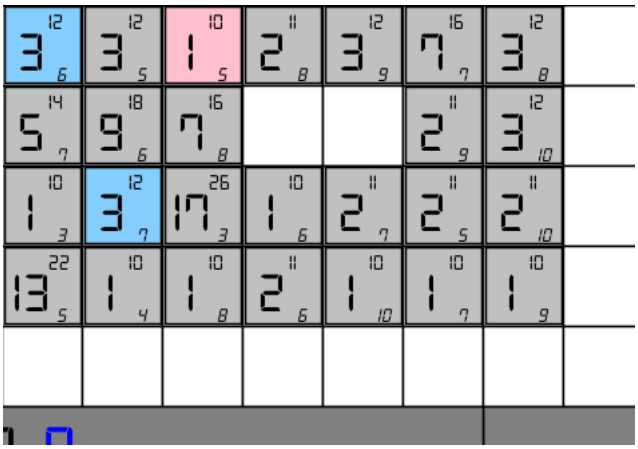


Figure 2: Departure information of containers

Additional settings and assumptions of this paper are as follows:

- All π-containers are of unit size.
- Each conveyor can only hold one π-container.
- Every railway car contains the same number of containers.
- Railway cars are fully loaded when they enter and leave the system, and they follow the first-in-first-out rule.
- All departure information of all π-containers is known while they enter the system, and this information will not be changed during the period of staying in the system.
- All containers' destination cars arrive later than the railway cars that carried these containers into the system.
- Each π-container can only communicate with the conveyor that holds it via some wireless technology such as Bluetooth, Wi-Fi, or RFID.
- Conveyors are in a grid-like configuration, and each knows its location in the grid.

3.1 Control Architecture and Operations

The control architecture of GridHub consists of two layers (Figure 3).

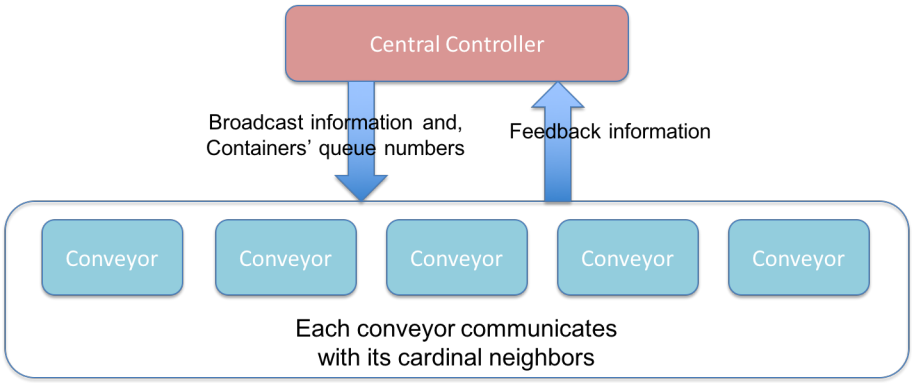


Figure 3: Control architecture of system

The top layer is a central controller, which has the following functions:

- Coordinates traffic moving, for example, when the railway car which is loading containers is full, it asks the train to move forward.
- Checks status of components and broadcasts operational information to conveyors. For example, it surveys all containers and railway cars in the system; if some containers are scheduled to be loaded into the car waiting for containers, it will broadcast this information to all conveyors.
- Sorts containers and assigns queue numbers to containers according to their departure information.

The second layer consists of all the conveyors. Every conveyor is loaded with the same rules, and each can perform the following activities:

- Assigns transferring tasks by marking the container as "requested," and gives feedback to the central controller if necessary.
- Negotiates with its four cardinal neighbors, and makes decisions to transfer a container based on the container's status and its neighbors' statuses.
- Carries out the negotiation decision by physically moving its container.

The chart in Figure 4 presents the operation flows of GridHub.

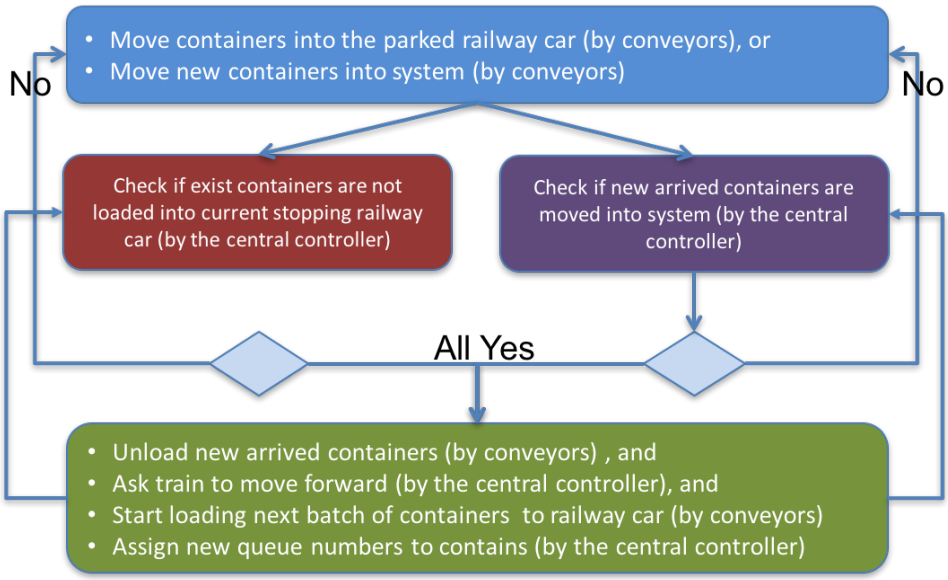


Figure 4: Overall operation steps

3.2 Control Algorithm

The control algorithm of GridHub extends the GridStore [7] and the GridPick [15] systems by adding additional negotiations. The approach is still divided into three phases (access-negotiate-convey), and the conveyors have several states to represent negotiation status or carried containers' status. These three phases of activities are taken by every conveyor simultaneously and cyclically, and each iteration performed by these conveyors can also be called an operation cycle.

Conveyors have three initial states decided by the items they hold:

- **Empty**: no container placed on it. The conveyor is shown as a blank white square.
- **Occupied**: holds a container that is not marked as "requested." The container on it is in silver.
- **Requested**: the container on it is marked as "requested," to be transferred toward one of the four neighbors. Container in blue is "requested" to be moved right, cyan is "requested" to be moved left, pink is "requested" to be moved down and gold is "requested" to be moved up.

The negotiations proceed according to these states, and during negotiations, conveyors enter or exit additional states with interacting activities among their neighbors. The steps of the algorithm are:

3.2.1 Access Phase

In this phase, each conveyor reads information of the container placed on it, and updates its initial state to one of the three states described above. The information to read includes: the container's state of "requested," the slot number, the car number and the queue number. If the container is marked as requested, its target column or target row will also be read.

3.2.2 Negotiate Phase

Checking fully occupied columns or rows Negotiations start with detecting whether each of the conveyors is located in a fully occupied row or column of conveyors. Two waves of message passing are performed. These messages are initialized by the conveyors in one edge of the system, and sent toward the other edge of the system. Only occupied conveyors can pass this kind of message; in other words, an empty conveyor will stop the message passing. If the messages reach the conveyors in the other edge through a column or row of conveyors, it means that all of the conveyors in this column or row are occupied. Also, the row or column consists of these conveyors are fully occupied. These conveyors are marked as "in full column" or "in full row." An example of fully occupied column is shown in the first picture of Figure 5.

Assigning transfer tasks By reading the broadcast information from the central controller, a conveyor assigns a task to a container according to its location in the grid and the container's departure information. For example, in Figure 2, the conveyor located in the furthest bottom row and the second column from right:

- It is in the right side of the slot which the container is supposed to be loaded into (it faces slot number 10, but the intended one is 9),
- it is in the 1st position of the virtual queue (its queue number is equal to or less than 2), and
- it has not been marked as "requested."

This conveyor will mark the container as "requested" in the left direction and the target column equals 9. Additionally, the conveyor will also change its state to "requested" with appropriate information.

Check "half fully" occupied columns or rows Similar to the above negotiation, after assigning a task, if a conveyor is not in a fully occupied row or column, and it is in the "requested" state, then this conveyor initializes two waves of negotiations toward both edges of the system in order to detect whether it is in a "half-fully" occupied column or row (second picture of Figure 5).

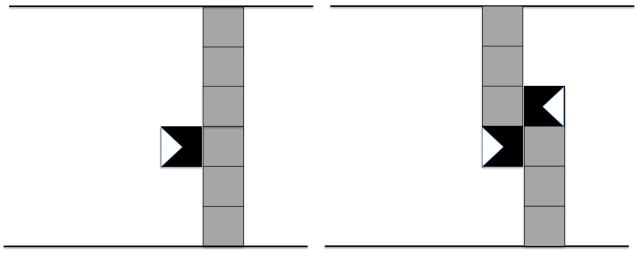


Figure 5: Example of fully occupied columns and half fully occupied columns

Negotiations to remove fully occupied rows or columns According to Gue et al. [7], to keep a GridStore system away from deadlock, at least one empty conveyor has to be in each column or row. Otherwise the cases shown in Figure 5 could cause deadlocks, and transfer tasks cannot be completed. We apply this rule to GridHub as well. To avoid deadlocks, some containers need to be removed from the fully occupied rows or columns. Negotiations include following steps:

- **Method 1**: every "requested" conveyor initializes messages to remove an item from the fully occupied column or row, or the column or row next to it, and this column or row is in its "requested" direction. This idea is similar with "balancing" negotiations of the GridPick system [15].
- **Method 2**: if the above method failed to "break" a fully occupied column or row, some containers meeting certain conditions will be forced to move out of the fully occupied column or row.

Negotiations to move newly arrived containers This wave of negotiations is to move the new arrivals into the system quickly to avoid the congestion near the loading and unloading row. The idea is similar to the "exchange of home row" method in [7]. As a result of successful negotiations, containers will be moved in tandem along a column, and the container in the loading and unloading row will be moved into the system in one cycle.

Negotiations to eliminate conflicts Since the containers are transferred in four directions, some of the conveyors holding them may compete with an empty conveyor during the transfer process, and we call these cases "conflicts." To solve this problem, we define priorities of containers by their requested directions. Their priorities are listed from high to low by requested directions: left, right, up and down. Every conveyor holding a higher priority container sends a message to its neighbor conveyors holding lower priority containers. By receiving this message, those conveyors holding lower priority containers change their states from "requested" to "occupied." As a result, those conveyors are considered "occupied" in the later negotiations of the current cycle, and conflicts can be solved.

Transfer negotiations After all negotiations prior to this wave of negotiation, all conflicts are solved, and all blocking is ready to be removed. Then, the regular negotiations of moving containers proceed, as defined by other GridFlow based systems [7, 8, 15].

3.2.3 Convey Phase

After all negotiations are done, conveyors know whether or not to move their containers, and they also know the direction of movement. Hence, physical movements are carried out by conveyors. After each container arrives at its destination, identifications will be performed to check whether the container is at its target location. If yes, they are converted to regular stored containers; otherwise, they will continue to be moved in future cycles.

Finally, every conveyor cleans all negotiation states to the initial state, and then it enters a new cycle of negotiation.

3.2.4 Example of Running System

Figure 6 illustrates how GridHub runs. In Figure 6a, the railway car labeled 18 is being moved forward to the parking position which is beside the bottom right part of the GridHub system. This railway car just unloaded π-containers that are located at the bottom left part of the system, and these containers are waiting to be moved into the system. Containers with queue numbers equal to 1 have been marked as "requested" to be moved down.

In Figure 6b, after several cycles of transportation, all arriving containers have been moved into the system. In the same period, containers scheduled to be loaded into the railway car labeled 18 keep moving into the car.

In Figure 6c, the last container belonging to railway car labeled 18 is leaving the system. Other blue containers are either moved to the column facing their slots on railway cars, or marked as "requested" by negotiations described in 3.2.2.

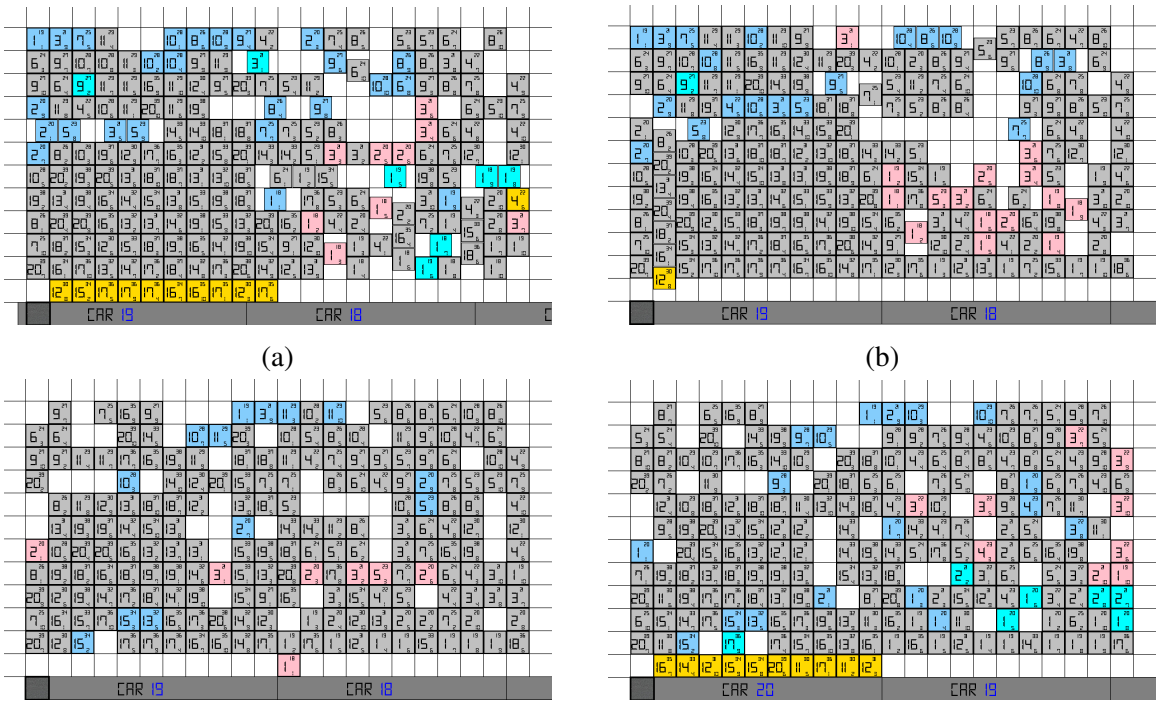
In Figure 6d, the car labeled 18 has left the system, and a new batch of containers from the car labeled 19 are unloaded.

4 Results and Discussion

Performance of GridHub is obtained by measuring the number of railway cars completing loading and unloading activities in a certain period. All of the simulations are performed in Anylogic.

4.1 Experiment Configurations and Results

The GridHub system for the experiment is set as a grid of conveyors with 22 columns and 11 rows. The number of rows will also be changed to 12 and 13 for further measurements of system performance. In each railway car, the number of slots is set to 10. The number



(a) (b) (c) (d)

Figure 6: Example of a running system

of containers in the system are set from 110 to 200, step 10. Table 1 shows the experiment settings. Each configuration in this table is run for 50 replications. In every replication, 6800 cycles are run. The first 800 cycles are the warm-up period, and the final 6000 cycles are the experimental period.

Table 1: Experiment settings

Number of containers	Number	of	rows
110	11	12	13
...	11	12	13
190	11	12	13
200	11	12	13

The number of railway cars processed is shown in Figure 7. In this figure, the horizontal axis shows the number of containers in the system, and the vertical axis shows the number of cars processed during the 6000 cycles. The performance is changed by changing of the system storage densities: while the system's storage utilization is low, system performance shows small differences among the three utilization settings. The storage utilization is

defined as:

$$U_s = \frac{Number\ of\ containers}{Number\ of\ conveyors(exclude\ the\ border\ conveyors)}$$

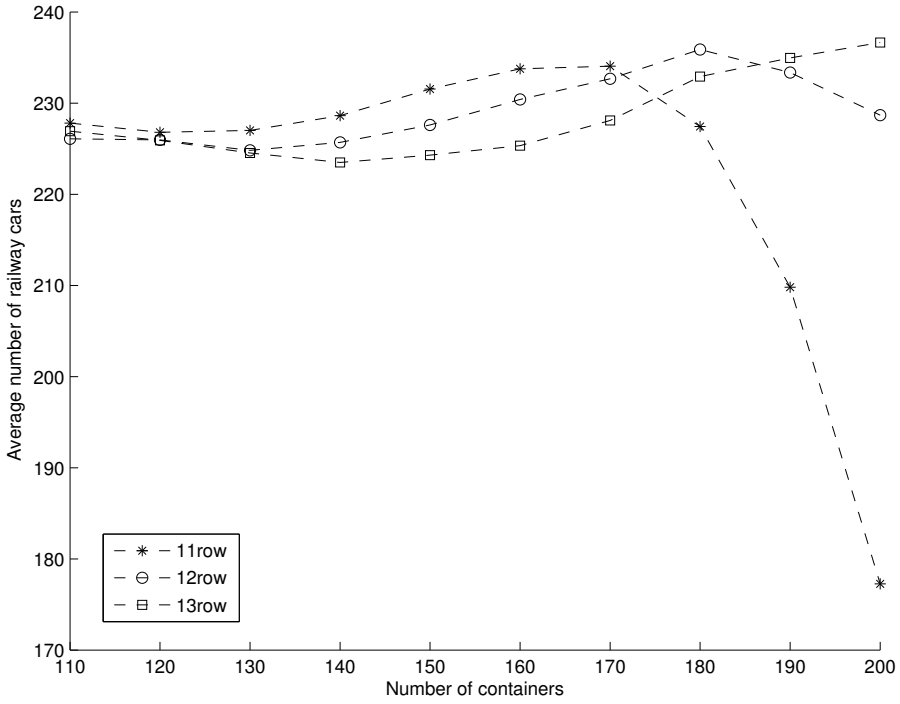


Figure 7: Number of cars processed

4.2 Discussion

If the GridHub is considered as the server of a single-server queuing system, the service times are the waiting intervals between departure events of two adjacent railway cars. The longer intervals require the railway cars to wait longer; then the longer intervals result in lower overall performance (Figure 7 and Figure 8). Hence, the number of railway cars processed is decided by the intervals between two loading and unloading events of adjacent cars.

The length of the intervals affected by multiple factors, and the value of these factors are plotted in Figures 9a to 10b. The mechanism is explained by two cases (low U_s and high U_s).

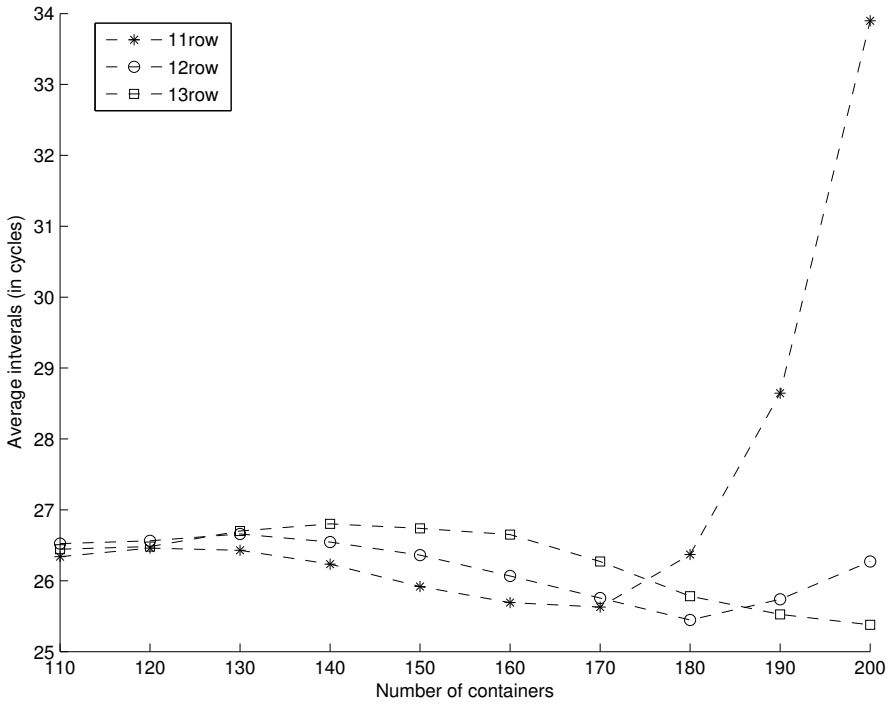


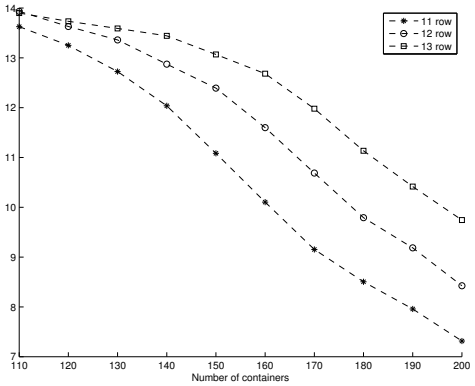
Figure 8: Average length of intervals between loading/unloading of two adjacent cars

When the system is less congested (the number of containers is less than 140), fewer negotiations (Method 2 in Section 3.2.2, Figure 9b) take place. This results in fewer containers being transferred into columns, which are facing their slots on cars in time. For example, a container scheduled to go to slot 10 may start to be moved from some places in the left part of the system when its target railway car has already arrived. As a comparison, in the cases of higher U_s, the container may already stays in the column which is facing its target slot on its target railway cars when the car arrives.

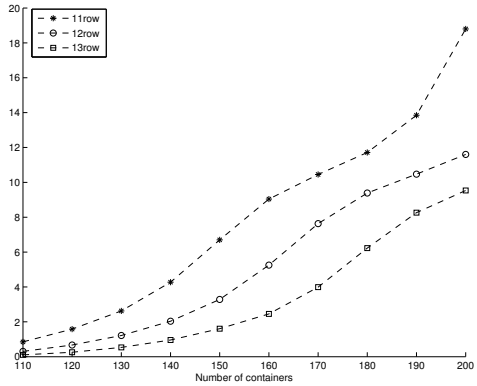
These extended distances increase the overall waiting time of trains (Figure 9a). In Figure 8, the increasing of intervals of the left parts are made in this way.

As the utilization of storage increases (number of containers greater than 140), the effects of the above mechanism get weaker, and the other factors have stronger influence on the system performance.

Though containers can be transferred to railway cars via fewer conveyors (Figure 9a), when the system is more congested (the number of containers greater than 140), these containers have to "pass through" a very crowded area. The crowded area is generated by the negotiation activities to remove fully occupied columns and rows (Figure 10a and Figure 10b). These negotiations make a lots of conveyors in "requested" states, but the



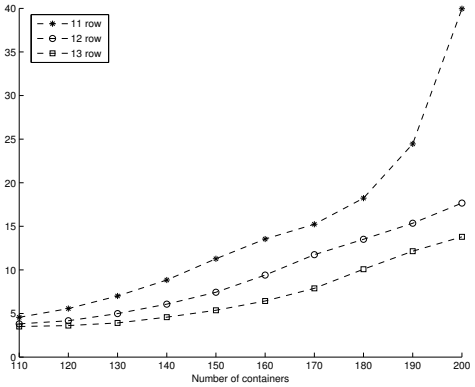
(a) Average longest distance(unit: number of conveyors)



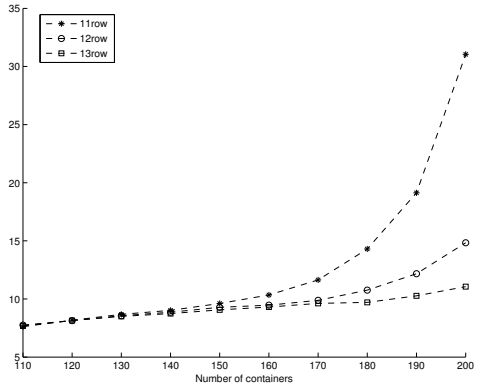
(b) Average number of negotiations of horizontal movement (method-2 in Section 3.2.2)

Figure 9: Factors affect performance while storage utilization is low

amount of empty conveyors are limited. Hence, it takes longer time to complete movements than the cases of lower utilization, so the performance of the system is lower in this case.



(a) Average number of negotiations of horizontal movement (method-1 in Section 3.2.2)



(b) Average number of negotiations of vertical movement (method-1 in Section 3.2.2)

Figure 10: Factors affect performance while storage utilization is high

5 Conclusion

The GridHub system offers a potential solution to the problem of transferring π-containers between rail cars at π-hubs in the Physical Internet. Of course, the algorithms we propose in

this paper would have to be instantiated in real software, running real hardware, and many modifications might be required. Nevertheless, the prototypical system we have simulated suggests high throughput is possible with a very high density of storage.

References

[1] Ballot, E., Montreuil, B., and Thivierge, C. (2012). *Progress in Material Handling Research 2012*. MHIA, Charlote, NC, U.S.A.

[2] Bostel, N. and Dejax, P. (1998). Models and Algorithms for Container Allocation Problems on Trains in a Rapid Transsipment Shunting Yard. *Transportation Science*, 32(4):370–379.

[3] Boysen, N., Fliedner, M., Jaehn, F., and Pesch, E. (2011). A Survey on Container Processing in Railway Yards: Decision Problems, Optimization Procedures and Research Challenges. Technical report, Friedrich-Schiller-Uiversitat Jena.

[4] Boysen, N., Fliedner, M., and Kellner, M. (2010). Determining fixed crane areas in rail-rail transshipment yards. *Transportation Research Part E*, 46:1005–1016.

[5] Furmans, K., Gue, K. R., and Seibold, Z. (2013). *Optimization of failure behavior of a decentralized high-density 2D storage system*. Springer Berline Heidelber.

[6] Gue, K. (2006). Very high density storage systems. *IIE Transactions*, 38:79–90.

[7] Gue, K., Furmans, K., Seibold, Z., and Uludag, O. (2014). GridStore: A Puzzle-Based Storage System With Decentralized Control. *IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING*, 11(2).

[8] Gue, K., Furmans, K., and Uludag, O. (2012). A High-Density System for Carton Sequencing. In *Proceedings of the 6th International BVL Symposium on Logistis*, Hamburg, Germany.

[9] Gue, K. R. and Kim, B. S. (2007). Puzzle-Based Storage Systems. *Naval Research Logistics*, 54(5):556–567.

[10] Mayer, S. H. (2009). *Development of a completely decentralized control system for modular continuous conveyors*. PhD thesis, Institutes fur Fordertechnik und Logistiksysteme der Universitat Karlsruhe (TH), Karlsruhe, Baden-Wrttemberg, Germany.

[11] Meller, R. D., Montreuil, B., Thivierge, C., and Montreuil, Z. (2013). Functional Design of Physical Internet Facilities: A Road-Based Transit Center. Technical Report FSA-2013-001, CIRRELT, Montreal, Canada.

[12] Montreuil, B. (2011). Toward a Physical Internet: meeting the global logistics sustainability grand challenge. *Logist. Res.*, 3:71–87.

[13] Schwab, M. (2015). *A decentralized control strategy for high density material flow systems with automated guided vehicles*. PhD thesis, Karlsruher Instituts fur Technologie (KIT), Karlsruhe, Baden-Wrttemberg, Germany.

[14] Seibold, Z., Stoll, T., and Furmans, K. (2013). Layout-Optimized Sorting of Goods with Decentralized Controlled Conveying Modules. In *Systems Conference (SysCon), 2013 IEEE International*, pages 628–633. IEEE.

[15] Uludag, O. (2014). *GridPick: A High Density Puzzle Based Order Picking System with Decentralized Control*. PhD thesis, Auburn University, Auburn, Alabama.