Subproject A1:
Capabilities and Limitations of Local Strategies in Dynamic Networks

Thorsten Götte[1], Till Knollmann[2], Friedhelm Meyer auf der Heide[2],
Christian Scheideler[1], Julian Werthmann[1]

1 Department of Computer Science, Paderborn University,
Paderborn, Germany
2 Heinz Nixdorf Institute and Department of Computer
Science, Paderborn University, Paderborn, Germany

1 Introduction

The On-The-Fly (OTF) market communication infrastructure plays a central role in our envisioned OTF ecosystem. At its core, it must ensure that the participants can communicate efficiently and reliably. During our efforts, we are facing many fundamental problems and challenges: in particular because the size and the dynamics of these systems makes it unrealistic to control and optimize them using classical centralized strategies executed by an entity that has full information about the current state of the system. Therefore, we explored the capabilities and limitations of local algorithms. An algorithm is *local* if each node in a distributed system only needs to interact with its neighbors in order to solve a given task. In large distributed systems, these neighborhoods continuously change, which will also affect the communication.

An especially challenging aspect is that of *external dynamics*, i.e., the network changes due to events that are outside of its control. External dynamics can happen for a variety of reasons. Most notably, they can be caused by attacks, faults, or changes in the membership. However, since the participants operate under ever changing economic or legal conditions, they might re-evaluate their communication interests and therefore change their interaction patterns.

Once the market reaches a certain size, centralized management approaches do not work anymore for handling external dynamics efficiently. For instance, a centralized controller that serves as an entry point to the market can quickly become a bottleneck or even a single point of failure if many actors try to join at once. Further, a malicious controller could even influence the market in its favor by denying certain providers from entering the market. Hence, distributed and ideally local strategies are needed. The goal of these strategies should not just be to maintain a fixed topology but to also adapt the topology to the needs of the market participants.

One of the topics that we focused on was self-stabilizing overlay networks, i.e., networks that can recover their topology from any weakly connected state. Many overlay networks

thgoette@mail.upb.de (Thorsten Götte), tillk@mail.upb.de (Till Knollmann), fmadh@upb.de (Friedhelm Meyer auf der Heide), scheideler@upb.de (Christian Scheideler), jwerth@mail.upb.de (Julian Werthmann)

have been proposed before, among them pioneering works such as Chord, CAN, Pastry, and Tapestry. However, these were not self-stabilizing. Later, various self-stabilizing overlay networks have been proposed, including self-stabilizing lists, skip graphs, de Bruijn graphs and Delaunay graphs (see, e.g., [FSS21] for an overview of stabilizing and non-stabilizing networks). However, they do not necessarily ensure monotonic self-stabilization in a sense that all parts of the network that are already in a desired state will remain in a desired state during self-stabilization. Since this property is critical for a high availability, we particularly focused on monotonically self-stabilizing overlay networks. An overview of our solutions in given in **Section 2.1**. *Monotone Searchability: A New Paradigm for Self-stabilizing Algorithms*.

Developing scalable solutions for distributed data structures was another topic of our research. Many solutions have already been proposed for distributed hash tables before our research, but no solutions were known before that scale well for inherently sequential data structures such as stacks, queues, and heaps. We did not just come up with highly scalable solutions that work under the very general asynchronous message passing model. We also devised ones that ensure strict consistency requirements such as sequential consistency. We describe our contributions in **Section 2.2**. *Distributed Data Structures: Scalability Meets Consistency*.

During the last years we initiated the study of hybrid communication networks. These are networks that support two different communication modes: a local mode that only allows the nodes to exchange information with some fixed, predetermined neighborhood, and a global mode where the nodes are able to change the network topology over time arbitrarily. Hybrid networks have already been used in various contexts such as data centers in which servers exchange information via some wired infrastructure as well as wireless or optical communication. Another prominent example is that of hybrid multi-point VPNs, where connections between their participants are established via leased lines as well as best-effort connections via TCP/IP. Despite their importance in practice, theoretical research on hybrid networks did not exist before our work. We particularly focused on solving graph problems via hybrid networks, such as finding a minimum spanning tree or shortest paths in the local network. We present more details of this research in **Section 2.3**. *Hybrid Networks: Exploiting the Heterogeneity of Modern Communication Infrastructures*.

Besides our research about models of networks and their dynamics, we have investigated resource allocation problems: Where should resources in a network be placed so that both the cost for placement and access are minimized? A prominent example is the facility location problem. In this problem, we are given a space with locations and distances between them, a function defining the cost of opening a facility at each location, and a sequence of requests at locations. Every request of the sequence must be served by a facility open when the request arrives for a price of the distance between the request and the respective facility. The goal is to open up facilities so as to minimize the total opening cost plus the total cost for serving all requests. Our contributions extend previous work in several directions: We developed efficient approximation algorithms in a dynamic settings by distributed algorithms. Further, we considered the online case where requests are not known in advance but instead arrive over time. An algorithm here needs to serve arriving requests immediately while not knowing future ones. Among others, we have analyzed leasing approaches, where resources are not bought and then exist “forever” but can be leased for different time periods and for different prices. We have investigated

the potential of allowing resources to be moved in the network (mobile resources), and we have generalized facility location and other resource allocation problems to take the heterogeneity of the resources into account. We describe our contribution to resource allocation in **Section 2.4**. *Online Allocation of Resources: Providing Services Without Knowledge on Future Demands*.

2 Main Contributions

In the following, we summarize the main contributions of our subproject.

2.1 Monotone Searchability: A New Paradigm for Self-Stabilizing Algorithms

Like any large-scale distributed system, our OTF market infrastructure will undergo frequent changes during its operation. The exact nature of these changes can be manifold. On the one hand, there are benign changes of the membership, e.g., participants joining or leaving the market in a coordinated way, or changes in the communication structure. On the other hand, there might also be actively harmful changes caused by malicious attackers. These attacks could even be aided by insiders in the market that benefit from failures of their competitors. For example, ill-intentioned service providers could try to exclude their competitors from the composition algorithm. They can achieve this by attacking the network in a way such that the corresponding composition requests do not reach other providers anymore and their services are not considered. Therefore, a key requirement for a robust OTF market infrastructure is *resilience* against these attacks as long as this is possible and *fast recovery* once the attack is over. In this subproject, we focused on solutions for fast recovery. As the global state of the system after a failure or attack might be arbitrarily corrupted, our protocols must potentially be able to recover the infrastructure from scratch. In particular, we want our system to reach a desired configuration from *any* initial configuration in a *finite* number of steps. This paradigm is commonly referred to as *self-stabilization* and has sparked a vast amount of research in the last 50 years. In his seminal paper, Esger W. Dijkstra defined self-stabilization as follows:

Definition 1 (Self-stabilization (cf. [Dij74])) *Let $\mathfrak{C}$ be the set of all possible states of a system. Then, a protocol is self-stabilizing w.r.t. to a set of legitimate states $\mathcal{L} \subseteq \mathfrak{C}$ if it satisfies the following two properties.*

- ***Convergence:*** *starting from an arbitrary system state $S_0 \in \mathfrak{C}$, the protocol is guaranteed to arrive at a state $S_t \in \mathcal{L}$ in a finite number of steps.*
- ***Closure:*** *starting from a legitimate state the protocol remains in legitimate states thereafter.*

A self-stabilizing protocol is thus able to recover from transient faults regardless of their nature. Moreover, a self-stabilizing protocol does not have to be initialized as it eventually starts to behave correctly regardless of its initial state.

For our particular case, we model the system as dynamic graph, more precisely, as an overlay network. Each node in the graph represents a market participant and each edge

represents a TCP/IP connection between the participants. We assume that nodes can delegate their edges by sending the corresponding IP addresses. Thus, a state might be modeled by a set of nodes V, their internal variables, and the messages in transit to each node. A legitimate state might then include a particular graph topology or a family of graph topologies. For each state S_t, we can define a graph $G(S_t) := G_t := (V, E_t^e \cup E_t^i)$ with two kinds of edges. First, there are *explicit* edges E_t^e with $(u, v) \in E_t^e$ if and only if u stores a reference to v in its local memory in step t. Second, we call an edge $(u, v) \in E_t^i$ *implicit* if and only if there is a reference to v in the channel of u in step t (and will eventually be received). Arguably, the most important operation in this model is the so-called *delegation*. Here, a node sends of its stored references to another node and thereby creates a new implicit edge.

It is well understood how one can build a plethora of useful network topologies in this model by delegating the edges. This research area is commonly referred to topological self-stabilization and it has seen a large number of impactful results in the last two decades. The investigated topologies range from simple line graphs, over sophisticated topologies of low degree and diameter, to spanners for a given metric space. A recent survey paper provides a comprehensive overview of the state-of-the-art [FSS21] of the techniques and algorithms for topological self-stabilization. Many of these protocols were developed during the first funding period of the CRC and/or by key researchers of this subproject. However, the specifics of these protocols are not the focus here. Instead, we consider a more structural problem shared by all these protocols: None of them provide any guarantees about the stabilization process (other than its eventual convergence). In particular, some desired functionalities or properties that hold in some configuration $S_\tau \notin \mathcal{L}$ may be violated in later state $S_{\tau'}$ with $\tau' > \tau$ that can be reached be the protocol. We argue that in many cases this is not the desired behavior of a resilient protocol. Intuitively, we want a stabilization protocol to *improve* in every time step, i.e., once parts of the system are operational, they should remain operational to ensure the best possible service.

One example of such functionality is *searching*, i.e., the ability to route a message to a specific node in the system. This is arguably one of the most frequently used functionality of any distributed system and — in the context of the CRC — crucial for the composition process. If we can find a node $v \in V$ from another node $w \in V$ in some configuration S_t, we want to be able to do so in all subsequent configurations. In this case, we say the protocol satisfies *monotone searchability*

Seeking a formal definition of the problem, let us first clarify what we mean by *searching*. We consider search requests, i.e. , `search`$(v, destID)$ messages that are routed according to a given routing protocol R, where v is the sender of the message and $destID$ is the identifier of a node we are looking for. Note that $destID$ does not necessarily belong to an existing node w but rather represents a virtual address that may or may not be occupied by a node. A search request can either *succeed* or *fail*. We say it *succeeds* if a `search`$(v, destID)$ message reaches a node w with $id(w) = destID$. Otherwise, if the message reaches some node u with $id(u) \neq destID$ **and** cannot be forwarded anymore according to R, the search request *fails*. We assume that nodes themselves initiate `search`() requests at will. Given these preliminary thoughts and definitions, we formally define *monotone searchability* as follows:

Definition 2 (Monotone Searchability (cf. [SSS15])) *We say a (self-stabilizing) proto-*

col P with legitimate states $\mathcal{L}$ satisfies monotonic searchability *according to some routing protocol R if it holds for any pair of nodes v, w that once a* `search`$(v, id(w))$ *request (that is routed according to R) initiated at time t succeeds, any* `search`$(v, id(w))$ *request initiated at a time $t' > t$ will succeed.*

In the following, we will slightly abuse notation and refer to a tuple $(\mathcal{P}, \mathcal{L}, R)$ of a self-stabilizing protocol $\mathcal{P}$, its set of legitimate states $\mathcal{L}$, and a routing algorithm R simply as a *protocol*. The research on monotone searchability went in two directions. First, the goal was to find efficient protocols that satisfy monotone searchability given a concrete set of legitimate states $\mathcal{L}$ and a concrete routing algorithm R. In this area, our researchers could develop protocols for constructing a sorted list, a skip list, and quadtrees embedded in the Euclidean plane. For more details on these protocols, we again refer to the aforementioned survey [FSS21]. The second (and arguably more interesting) direction considered the question of which properties must be fulfilled by a protocol to support monotone searchability, or, on a related note, which classes of protocols can be transformed into protocols that satisfy monotone searchability. Surprisingly, in [SSS16] Setzer, Scheideler, and Strothmann could show that a broad class of existing self-stabilizing protocols containing virtually all protocols developed in the CRC can be transformed to satisfy monotonic searchability. In the remainder of this section, we will quickly outline their approach. We begin by considering the two preconditions a protocol must fulfill to be suitable for the transformation, a generic distance metric `dist`$(\cdot, \cdot)$ and the so-called **MDL** property.

Distance oracle `dist`$(\cdot, \cdot)$: First, all nodes need to have access to a distance oracle `dist` : $V \times V \to \mathbb{R}$ that takes two identifiers as input and output their distance in some shortest path metric. Further, the nodes must be able to evaluate the oracle locally. While this might sound like a hard and unhandy assumption at first, a closer look into existing protocols reveals this is indeed fulfilled by almost all of them. For example, many topologies contain a sorted list of its nodes. Assuming that identifiers are integers in $[1, n]$, one can simply define `dist`$((v, w)) := |v - w|$ and obtain an oracle with all desired properties.

MDL property: The next property (or rather set of properties) is dubbed **MDL** property in [SSS16]. It encapsulates four basic features that ensure that a protocol gets monotonically closer to the legitimate states. More precisely, a deterministic protocol $(\mathcal{P}, \mathcal{L}, R)$ fulfills the **MDL** property if for any action a of the protocol it holds that:

1. An edge $(u, v) \in E_{\mathcal{L}}$ will **never** be delegated by $u \in V$.
2. If an edge $(u, v) \notin E_{\mathcal{L}}$ is delegated in step τ, it will be delegated in all steps $\tau' > \tau$ (if u receives another reference of v).
3. Any stable edge that may be traversed by the search protocol (and any implicit edge that results from it delegation) is only delegated to a node whose distance (in the underlying distance metric) is closer to the target than the current node.

Informally speaking, the first two properties imply that the protocol *monotonically* converges to its desired topology, since edges of the topology are always kept and edges that are not part of the topology are obviated over time. The last property states that the delegation of edges can only improve the routing. Further, if the legitimate states contain implicit edges, we additionally require the following:

4. In every legitimate state, for any implicit edge (u, v), there are fixed cycle-free paths $(u = u_1, u_2, \ldots, u_k)$ such that u_i sends the reference of v to u_{i+1}, and u_k has an explicit edge (u_k, v), i.e., the reference of v is forwarded along fixed paths until it finally fuses with an existing reference.

This property implies that all implicit edges will eventually merge with explicit ones in legitimate states. Note that the MDL property is generally not a severe restriction as almost all topological self-stabilization algorithm fulfill it naturally.

Given all these two preconditions and constructions, the main result was the following:

Theorem 1 (Main Result of [SSS16]) *Any self-stabilizing protocol $(\mathcal{P}, \mathcal{L}, R)$ that satisfies the* MDL *property can be turned into a protocol $(\mathcal{P}', \mathcal{L}, R')$ that satisfies monotone searchability.*

The actual construction of Setzer, Scheideler, and Strothmann has two building blocks. First, they adapt the mechanisms how edges are delegated by the protocol. Second, they introduce a generic routing protocol that exploits the distance metric and the MDL property. Again, we give a brief overview over both approaches.

Safe delegations: Recall that our model does not assume FIFO delivery, so messages may be received out of sequence. This creates problems if a node v forwards a reference to another node and a `search`$(\cdot, \cdot)$ message that needs to be sent to that reference. With non-FIFO delivery, the `search`$(\cdot, \cdot)$ message could *overtake* the reference and the search fails although it worked when v still stored the reference. Thus, our protocols need to cope with the non-FIFO message delivery. To address this issue, Setzer, Scheideler, and Strothmann developed a set of extensions for the standard topology manipulation primitives. On a high level, these extended primitives ensure that edges delegated away by some node $v \in V$ are not integral to the routing. Their key technique to achieve this is to *warn* neighboring nodes that there will be a delegation and then wait for their acknowledgment that this delegation is indeed safe. This, however, requires the use of sequence numbers to match warnings and their acknowledgments. Thus, the system only stabilizes if the initial state does not contain corrupted sequence numbers. Although this goes against the main idea of self-stabilization, it is (in some sense) unavoidable. The authors proved that under non-FIFO message delivery, no protocol could maintain monotone searchability from arbitrary initial states. Therefore, either the initial set of states must be restricted or one needs to assume FIFO delivery.

Generic routing: Finally, the authors need to account for the fact that the routing R takes paths that are not guaranteed to persist. To cope with this problem, they introduce a *generic search protocol* R' that exploits the distance oracle `dist`$(\cdot, \cdot)$ and the MDL property. Informally speaking, the message always takes the *smallest possible* step towards the target with respect to `dist`(v, w) that it has seen so far. To this end, all identifiers *seen* along the path are also stored in the message. If the search gets stuck in some node, it can use these to *backtrack* and try another path.

2.2 Distributed Data Structures: Scalability Meets Consistency

Like in the sequential world, efficient distributed data structures are important in order to realize efficient distributed applications. The most prominent type of distributed data

structure is the distributed hash table (DHT). Many distributed data stores employ some form of DHT for lookup. Important applications include file sharing (e.g., BitTorrent), distributed file systems (e.g., PAST), publish-subscribe systems (e.g., SCRIBE), and distributed databases (e.g., Apache Cassandra). However, other distributed forms of well-known data structures, such as queues, stacks, and heaps, have received much less attention although queues, for example, have a number of interesting applications as well. The main challenge of coming up with scalable distributed solutions of queues, stacks, and heaps is that they are inherently sequential, i.e., the challenge is to execute the operations in a distributed fashion so that the outcome of the execution is consistent with a sequential execution. We considered the following forms of consistency.

Definition 3 *Let $\mathcal{DS}$ be a distributed data structure on a node set V and let OP be set of requests issued to the data structure. Further, each request $op(u, i) \in OP$ is associated with a node $u \in V$ and sequence number $i \in \mathbb{N}$. Then, we say*

1. *$\mathcal{DS}$ is **serializable** if and only if there exists an ordering $\prec$ on the set OP so that the distributed execution of all requests in OP on the data structure is equivalent to the serial execution w.r.t. $\prec$.*
2. *$\mathcal{DS}$ is **locally consistent** if and only if there exists an ordering $\prec$ on the set S so that for all u and i it holds that $op(u, i) \prec op(u, i + 1)$.*
3. *$\mathcal{DS}$ is **sequentially consistent** if and only if it is serializable and locally consistent w.r.t. the same ordering.*

Intuitively, local consistency means that for each node v, the requests issued by v have to come up in $\prec$ in the order they were issued by that node. Thus, it is somewhat independent of the data structure's semantics and may always be achieved by a trivial lexicographic order. The same cannot be said of serializability, which is greatly affected by the data structure's semantics and, moreover, is highly non-trivial to be achieved efficiently in a distributed system.

For example, consider the classical (sequential) stack data structure where `Push(i)` adds a data item i and `Pop()` returns the data item that was added last. In the distributed setting, each node is able to invoke `Push(i)` and `Pop()`. The order relation $\prec$ must ensure that the request can mapped to a sequential execution. In particular, this means that any `Pop()` operation that returns i is preceded by a `Push(i)` operation (by some node). Moreover, there must be no `Push(j)` operation with `i` $\neq$ `j` ordered in between the operation (unless there is another `Pop()` that removes `j`). A straw-man distributed solution could simply see one node responsible for the stack and let it handle all `Push(i)` and `Pop()` requests. However, this would hardly be scalable as a single node needs to process the entire system's traffic. Thus, an efficient solutions must carefully distribute the responsibilities for each requests while still ensuring serializability. The feasibility greatly depends on the data structure's concrete semantics. Besides the stack, we were able to develop solutions for several different well-known and established data structures, which at first glance might seem inherently sequential. These are presented in the remainder of this chapter.

Distributed queue: A distributed queue can be used to come up with a unique ordering of messages, transactions or jobs, and it can be used to realize fair work stealing since tasks available in the system would be fetched in FIFO order. Other applications are distributed

mutual exclusion, distributed counting, or distributed implementations of synchronization primitives. Server-based approaches of realizing a queue in a distributed system already exist, such as Apache ActiveMQ, IBM MQ, or JMS queues. Many other implementations of message and job queues can be found at http://queues.io/. However, none of these implementations provides a queue that allows massively parallel accesses without requiring powerful servers. The major problem of coming up with a fully distributed version of a queue is that its semantics are inherently sequential. Nevertheless, we were able to come up with a distributed protocol for a queue ensuring sequential consistency that fairly distributes the communication and storage load among all members of the distributed system and that can efficiently process even massive amounts of `Enqueue()` and `Dequeue()` requests. Our protocol works in the asynchronous message passing model and can also handle massive amounts of join and leave requests efficiently.

A distributed queue has to implement four operations: `Enqueue()`, `Dequeue()`, `Join()` and `Leave()`. `Enqueue()` adds an element to the queue and `Dequeue()` removes an element from the queue so that the FIFO requirement is satisfied. `Join()` allows a process to enter the system while `Leave()` allows a process to leave the system.

We presented a distributed queue ensuring sequential consistency under the asynchronous message passing model, which also ensures a high scalability. More precisely, when assuming synchronous message passing, our `Enqueue()` and `Dequeue()` operations are processed in $O(\log n)$ communication rounds w.h.p., where n is the number of nodes. Furthermore, we show that we can process n `Join()` or $n/2$ `Leave()` operations in $O(\log n)$ rounds, w.h.p. Through the use of a distributed hash table, our distributed queue allocates its elements equally among all processes such that no process stores significantly more elements than the rest [FSS18a]. In the arXiv version of our paper, we also showed how to use the techniques for our distributed queue to come up with a highly scalable distributed stack ensuring sequential consistency [FSS18b].

Distributed priority queue: We also presented a highly scalable distributed priority queue (or heap). A distributed heap might be useful in scheduling, for example, where one might insert jobs that have been assigned priorities and workers might pull these jobs from the heap based on their priority. Another application for a distributed heap is distributed sorting. A distributed heap supports the following operations:

- `Insert(e,p(e))`: Inserts the element `e` with priority `p(e)` into the heap.
- `DeleteMin()`: Retrieves the element with minimum priority from the heap or returns $\perp$ if the heap is empty.
- `Join()`: The node v issuing the operation wants to join the system.
- `Leave()`: The node v issuing the operation wants to leave the system.

We distinguished between settings that only allow a constant amount of priorities and settings for arbitrary amounts and presented two novel distributed protocols for these scenarios – Skeap and Seap. Both protocols support insertions and deletions of elements in time $O(\log n)$ w.h.p., where n is the number of processes participating in the heap. Furthermore, we provided some guarantees on the semantics, by having Skeap guarantee sequential consistency and Seap guarantee serializability. For part of Seap, we obtained a novel protocol KSelect for distributed k-selection that runs in $O(\log n)$ rounds w.h.p. Both Skeap and Seap work in the asynchronous message passing model. To provide an

additional feature we can handle join and leave requests of processes in time $O(\log n)$ w.h.p. without violating the heap semantics or losing important data. Even though Seap comes with slightly weaker semantics than Skeap, it only uses $O(\log n)$ bit messages for its operations, while the message size in Skeap partially depends on the rate with which processes generate new operations [FS19].

2.3 Hybrid Networks: Exploiting the Heterogeneity of Modern Communication Infrastructures

In our study of hybrid networks, we consider systems with multiple modes of communication. Specifically, we allow our nodes to employ *local* communication with a fixed set of neighbors and global communication with any node in the network. We find many examples of such networks in the context of the CRC. Consider, for example, an execution of a composed service. Since the composition consists of multiple services from multiple providers that possibly have restrictions on where and how their services are initiated, it is unlikely to be executed by single compute node at a single location. Rather, it involves several servers at multiple compute centers at different geographical locations. Such a setup may be seen as a hybrid network. A single server can easily access data stored by servers in the same compute center as are they equipped with capable networking hardware. These would be the local connections, which can be used for large data transfers. On the other hand, large-scale data transmissions between different compute centers are possible in principle, but much slower. Therefore, a connection with a server in another center can be seen as a global connection. Due to their limitations, these global connections should rather be used for metadata, control messages, or intermediate results. Moreover, there are several avenues for hybrid networks beyond the scope of the CRC. For example, they can also be motivated by the ability of modern smart phones to employ device-to-device communication (local) as well as their internet connection (global). As the latter is usually connected to a cost, we tend to impose harsher limits on the number of messages each device is allowed to send via this connection in a given amount of time.

In our research, we focused on efficient algorithms for these hybrid networks that cleverly exploit these different communication capabilities. In the remainder of this section, we will present a more formal treatment of the model and give an overview of our and related results.

The Hybrid communication model (cf. [AHK+20]): In the HYBRID communication model, we consider a fixed set of nodes V with identifiers of length $O(\log n)$. Time is synchronous, i.e., divided into rounds. At the beginning of a round, each node receives messages sent to it in the last round. Afterwards it may perform an arbitrary amount of local computation and then send messages to other nodes that will be delivered in the next round. Specifically, these nodes can communicate in two modes and we parameterize the model by the messaging capacities allowed in each of them. For the so-called local mode, we are given a fixed set of edges and each node may send λ messages of size $O(\log n)$ to each of its neighbors in every round of communication. For the so-called global mode, we assume the nodes to be connected as a clique. However, each node may only send and receive γ messages of size $O(\log n)$ in total in every round of communication.

Special non-hybrid cases of the HYBRID model are the LOCAL model ($\lambda = \infty$, $\gamma = 0$),

the CONGEST model ($\lambda = O(1)$, $\gamma = 0$) and the node-capacitated clique (NCC, $\lambda = 0$, $\gamma = O(\log n)$). In some cases, a more restricted version of the model is studied, where initially no global edges exist. In this case, each node may send any identifier of a node it knows via a message to have the target node learn about the node in the message, thereby establishing a global edge between them. This model of global communication is called NCC_0 for $\gamma = O(\log n)$. Sometimes λ and γ denote the amount of bits that are allowed to be sent over local or global edges, respectively. Finally, we allow the edges of the HYBRID model to be weighted by a weight function w assigning a natural numbered weight to each of the local edges.

Hybrid shortest paths algorithms: An interesting branch of our research on hybrid networks focused on the investigation of shortest paths algorithms. We initiated this research with the paper *Shortest Paths in a Hybrid Network model* by Augustine et al. ([AHK+20]), where we studied how the addition of global edges affects the runtime of SSSP and APSP compared to classical model. To this end, the very powerful LOCAL model was picked for the local edges and the rather restrictive NCC was picked for the global edges, i.e., $\lambda = \infty$ (though $\lambda = n$ suffices for our algorithms) and $\gamma = O(1)$. The main contributions are presented in Table 1 for APSP and Table 2 for SSSP. We want to specifically mention the APSP lower bound of $\widetilde{\Omega}(\sqrt{n})$ for $\widetilde{O}(\sqrt{n})$-approximations. Additionally, we stress that the runtimes beat the non-hybrid lower bound of $\Omega(D)$ for many graphs.

APSP	Approx	Weights	Complexity	Local Capacity
	Exact	✓	$\widetilde{O}(n^{2/3})$	$O(n)$
	$(1+\varepsilon)$	-	$\widetilde{O}(\sqrt{n/\varepsilon})$	$O(n)$
	3	✓	$\widetilde{O}(\sqrt{n})$	$O(n)$
	$\widetilde{O}(\sqrt{n})$	-	$\widetilde{\Omega}(\sqrt{n})$	∞

Table 1: *Overview of the APSP contributions of [AHK+20].*

SSSP	Approx	Weights	Complexity	Local Capacity
	Exact	✓	$\widetilde{O}(\sqrt{\mathsf{SPD}})$	$\widetilde{O}(n^2)$
	$(1+\varepsilon)$	✓	$\widetilde{O}(n^{1/3}/\varepsilon^6)$	$\widetilde{O}(n^{2/3}\varepsilon^6)$
	$(1/\varepsilon)^{O(1/\varepsilon)}$	✓	$\widetilde{O}(n^{\varepsilon})$	$O(1)$
	$2^{O\left(\sqrt{\log n \log\log n}\right)}$	✓	$2^{O\left(\sqrt{\log n \log\log n}\right)}$	$O(1)$

Table 2: *Overview of the SSSP contributions of [AHK+20].*

In the paper *Fast Hybrid Network Algorithms for Shortest Paths in Sparse Graphs* by Feldmann et al. ([FHS20]), we further study the SSSP problem in graphs with only a few edges. Here, we parameterize the hybrid model with $\lambda = O(1)$ and $\gamma = O(\log n)$, restricting the local edges to the CONGEST model. Hence, both local and global is limited, resulting in a more realistic setting. We present deterministic $O(\log n)$ time SSSP algorithms for path graphs, cycle graphs, trees and pseudotrees (i.e., graphs with exactly one cycle). Additionally, we present a randomized $O(\log n)$ time SSSP algorithm for cactus graphs (i.e., graphs where any two cycles share at most one node) and a randomized $O(\log^2 n)$ time algorithm for any graph with at most $n + O(n^{1/3})$ edges and arboricity $O(\log n)$ (cf.

Table 3). As the lower bound for the CONGEST model is $\Omega(D)$, the logarithmic runtimes correspond to an exponential speedup caused by the addition of global edges. In addition to this, the paper presents many useful techniques for HYBRID algorithms such as multiple aggregation techniques and a technique allowing the simulation of algorithms for the well-established PRAM model.

SSSP	Class	Approx	Weights	Complexity
	Path Graphs	Exact	✓	$O(\log n)$
	Cycle Graphs	Exact	✓	$O(\log n)$
	Trees	Exact	✓	$O(\log n)$
	Pseudotrees	Exact	✓	$O(\log n)$
	Cactus Graphs	Exact	✓	$O(\log n)$ w.h.p.
	$*$	3	✓	$O(\log^2 n)$ w.h.p.

$*$: $n + O(n^{1/3})$ edges and arboricity $O(\log n)$

Table 3: *Overview of the contributions of [FHS20].*

In the paper *Near-Shortest Path Routing in Hybrid Communication Networks* by Coy et al ([CCF+22]) and its successor paper [CCS+23], we shift our attention from SSSP to routing. Specifically, we consider how we can construct routing tables and node labels of size $O(\log n)$ that allow us to forward packets from a source to a target such that the path taken is worse than the shortest path by a constant factor only. We restrict our research on *unit disk graphs* (UDGs) that have nodes embedded in $\mathbb{R}^2$ and an edge between two nodes iff their Euclidean distance is at most 1. To this end, we show how to transform any routing scheme for grid graphs, i.e., graphs with nodes embedded in $\mathbb{Z}^2$ and edges between nodes that have distance of 1, to a routing scheme for UDGs in constant time while only losing a constant factor in the distances traveled. This allows us to present an exact routing scheme for grid graphs and obtain the desired routing scheme for UDGs.

Resulting related work: Our research on shortest paths in the HYBRID model sparked interesting research by other authors that has been published at established and competitive conferences. In their paper *Distance Computations in the Hybrid Network Model via Oracle Simulations* Censor-Hillel, Leitersdorf, and Polosukhin provide an exact weighted shortest path algorithm that runs in $\tilde{O}(n^{1/3})$ rounds, w.h.p. [CLP21]. To achieve this, they exploit the ability of nodes with many local connections to distribute data much quicker than nodes that are not as well connected. Specifically, their goal is to simulate *oracles* that are able to receive $\deg(v)$ messages from each node v per simulated round. Note that this would enable them to learn the entire graph, which is known to be difficult in HYBRID and would therefore require a large overhead in runtime. Hence, they restrict the simulation to a skeleton graph roughly representing the entire graph. This allows the simulation of one round of oracle communication in $\tilde{O}(n^{1/3})$ rounds, w.h.p. Further, the paper *Routing Schemes and Distance Oracles in the Hybrid Model* by Schneider and Kuhn [KS22] investigated the lower bounds in the HYBRID model. The authors show that constant factor approximations of APSP require a runtime polynomial in n on general graphs. More precisely, they present a worst-case graph where computing an α-approximation of all shortest paths takes $\Omega(n^{O(1/\alpha)})$ time. This shows that our results on general graphs cannot fundamentally be improved to, say, polylogarithmic runtimes (unless $\alpha \in \Omega(\log(n))$). Further, the lower bound of [KS22] is accompanied by algorithms that (almost) match this

bound.

Outlook: Although [CLP21] and [KS22] provide strong results, this does not mean that the topic of (approximate) APSP is exhausted. Recall that the lower bound from [KS22] only holds for general graphs as the worst-case instance has a quite unique and pathological topology. As a consequence, we will focus our attention on restricted graph classes that often appear in practice. We have already made some progress here, as evidenced by [FHS20] and [CCF+22], but strive to extend this with even more practically motivated graphs classes such as planar graphs or graphs embedded in the Euclidean plane.

2.4 Online Allocation of Resources: Providing Services without Knowledge on Future Demands

Within the scenario of our CRC not only are the clients interested in an automatic composition of software services, they also desire to use the created services. To this end, services are executed in a distributed system and the system itself should manage how and where the services are deployed while the clients access them. Nowadays, deployment of software in distributed systems is done virtually such that the services can be managed widely independent of the physical infrastructure. One common approach is to run services in virtual machines that simulate a physical machine by software. The main advantage of such virtual machines for our system is that they allow us to manage the deployed services dynamically. For example, they can be migrated (moved to another participant) if one participant receives too many requests, or re-configured if the specifications of services or requirements of clients change. Our system benefits from these properties a lot as it further allows us to design algorithms that optimize the placement of the services.

Resource allocation problems: On the one hand, clients are interested in having their desired services at network participants that are close, as this reduces the delay while utilizing the services. On the other hand, the deployment of services itself requires time, energy, and available local resources (computation time, memory), so we should, for example, avoid deploying too many copies of services to ensure that our system scales. Such a setting can be modeled by the well-established field of research around *resource allocation* problems. Here, a set of *resources* (virtual machines containing services) is managed by an algorithm to serve a sequence of *requests*. The goal is to minimize a cost function. Costs are given by the actions of the algorithm such as deployment cost or migration cost and also by the cost to serve requests (for example, given by the distance of a request to the nearest resource).

Within the broad area, there are several models for resource allocation that consider different possible scenarios. The facility location problem, for example, considers the setting where the algorithm is only allowed to deploy new resources and serves requests by connecting them to the nearest one. Deploying and connecting incurs the cost here. Other instances of resource allocation problems are the file migration problem, the k-server problem, and the set cover problem. In file migration, the algorithm cannot deploy a new resource but it can migrate the existing one. Similarly, the algorithm cannot deploy new resources but migrate existing ones in the k-server problem. In comparison to file migration, the algorithm must place one copy (of k existing ones) on the location of each arriving request to serve it (it cannot serve a request by connecting it). In the set cover

problem, each request asks for an element of a given ground set. The algorithm serves a request by offering the requested element, which can be accomplished by buying subsets of the ground set. These subsets are also fixed and given as part of the input.

Approximation algorithms: Many resource allocation problems are difficult to compute. The facility location problem, for example, is known to be NP-hard and motivates the research on *approximation algorithms*. Here, the aim is to compute an approximate solution having at most p times the cost of the optimal solution, where p is the *approximation factor*. In turn, the computation time of such an approximate solution is polynomial in the input. For example, one of the early approximation algorithms for the facility location problem achieves an approximation factor of 3.16 [STA97]. In our research, we concentrated on distributed approximation algorithms for resource allocation problems. On top of the distributed setting, we considered a network that is under *external dynamics*, i.e., that changes over time uncontrollably. In [ACD+11], we presented two *local* approximation algorithms for the metric facility location problem. Both have a poly-logarithmic runtime in the number of peers and a constant approximation factor (one of the two algorithms is slightly faster than the other but may provide a slightly worse solution). It is especially interesting that both algorithms can deal very well with the aforementioned external dynamics.

Online algorithms: Besides the difficulty to compute solutions even when all requests are known, resource allocation offers the following additional challenge. On top of the task to manage services optimally, the requests of clients are usually not known beforehand in our system. In contrast, they arrive over time and future requests are usually unknown. Therefore, the main difficulty that our system has to deal with is maintaining a good placement of the services while adapting the placement for future unknown requests. This motivates us to consider resource allocation problems in a setting where requests arrive *over time* while the algorithm has to take irrevocable actions to guarantee that each request gets served before the next arrives. Such problems are denoted as *online problems*. The criterion of actions being *irrevocable* captures a natural rule: When the algorithm decides to execute an action, the cost of it is paid immediately. The decision cannot later be taken back to reduce the cost when knowledge about future requests is obtained. Of course, an algorithm following such a strong restriction might not be able to perform as well as one that knows all requests beforehand. To measure the loss in performance due to considering a problem online, the *competitive ratio* was introduced and evolved into a standard measure (as amortized efficiency in [ST85]):

Definition 4 (Competitive Ratio) *Let P be a problem with a set of instances I. Let* Alg *be an online algorithm and* Opt *be an optimal offline algorithm for P. Denote by* $\mathrm{Cost}(A, i)$ *the total cost of an algorithm A on an instance $i \in I$. Then* Alg *is called c-competitive if for all instances $i \in I$ for some constant a independent of i it holds that*

$$\mathrm{Cost}(\textsc{Alg}, i) \leq c \cdot \mathrm{Cost}(\textsc{Opt}, i) + a.$$

The competitive ratio allows us to measure how well an algorithm (Alg) performs in terms of the best way it could perform had it known all requests beforehand (Opt). Note that the ratio is a worst-case ratio, i.e., it expresses a guarantee of the performance of an online algorithm regarding every possible valid input sequence for the respective problem.

As an example, consider the facility location problem mentioned already above. Here, the lower bound on the competitive ratio is $\Omega(\frac{\log n}{\log\log n})$ where n is the number of arriving requests [Fot08]. On the other hand, there are several algorithms with a competitive ratio close to this bound, e.g., a randomized algorithm by Meyerson with an expected competitive ratio of $O(\frac{\log n}{\log\log n})$ [Mey01; Fot08]. Regarding online resource allocation, we considered three directions; leasing of resources, mobility, and heterogeneity.

Leasing problems: In *leasing*, the classical problems are extended by assuming that each resource that is otherwise bought is now *leased* for a fixed time only. The possible lease lengths and their costs are given as part of the input. When the lease for a resource runs out, it has to be leased again if needed. Leasing captures the intuition that a deployment of a service should not remain forever but is only required as long as clients using the service arrive. The leasing model above became famous as the parking permit problem considered by Meyerson [Mey05], which itself generalizes the ski rental problem. Regarding the parking permit problem, Meyerson showed a deterministic lower bound of $\Omega(k)$ and a randomized lower bound of $\Omega(\log k)$ where k is the number of available leases. For both cases — randomized and deterministic — asymptotically optimal algorithms are known. The parking permit problem grasps the difficulty of leasing very precisely and has been applied to other resource allocation problems to extend them. For example, Nagarajan and Williamson used it to extend the aforementioned online facility location problem by leasing in [NW13]. Here, requests can only be served by resources available at the point in time the request arrives. They presented an algorithm achieving a competitive ratio of $O(k \log n)$, i.e., with an additional factor of k in the competitive ratio due to the leasing in comparison to the classic facility location problem.

Leasing introduces the additional difficulty of not only deciding on *where* and *when* resources are deployed, but also *how long* they are leased. In [KMP12; AKM+16], we presented the first algorithm for online facility leasing that is independent of the length of the request sequence n. In general, the algorithm has a competitive ratio of $O(\ell_{\max} \log \ell_{\max})$ where $\ell_{\max}$ represents the length of the longest lease. Additionally, for many natural input sequences where the number of arriving requests per time step does not vary too much in consecutive steps, the algorithm achieves a better competitive ratio of $O(\log^2 \ell_{\max})$. Leasing can also be applied to other resource allocation problems such as the set cover problem. The leasing variant of the set cover problem has previously been studied exclusively as an offline problem (cf. [AG07]). Here, the model extension is similar to the online facility location problem, i.e., sets are no longer bought but have to be leased following the given available leases. We have presented the first online algorithms for set cover leasing in [AMM14]. Our randomized algorithms also work for the variant of set cover with repetitions presented by Alon et al. [AAG09], where elements appear multiple times and must be served by a different set each time. Our results improve their competitive ratio of $O(\log^2(m \cdot n))$ to $O(\log m \log(m \cdot n))$, where n is the size of the ground set and m is the number of available sets that can be leased.

Mobile resource augmentation: Regarding *mobility*, we have addressed the question of whether and, if so, to what extent we can improve resource allocation through resource mobility. This involves a model that allows moving resources in Euclidean space to make them more usable. Our models capture the intuition that a slight adaption of the location a service is deployed at has a negligible cost, while it might significantly improve the overall cost. As a basis, we have considered a simple model [FM19] similar to the

file migration problem, in which a single resource can be moved over a finite distance at each time step to better serve future requests. Recapitulate that in the file migration problem [BS89], the movement of the algorithm is unlimited although the cost increases with the moved distance. If both the optimal solution and the online algorithm are allowed to move resources, the competitive ratio is immediately dependent on the length of the request sequence. Therefore, we consider mobility as a *resource augmentation*. Resource augmentation gives additional actions to the online algorithm while still comparing its performance to an optimal solution that does not have these actions.

For mobility, this means that the online algorithm is allowed to move its resources in a very limited way while the optimal solution cannot move its resources. For the base problem mentioned above, we have shown that a simple algorithm is sufficient to achieve an optimal competitive factor. In [FKMM21], we extended the model to deal with k mobile resources. Here, the additional decision an algorithm has to make is which resource to move. In general, no online algorithm can even have a bounded competitive ratio in our model. To achieve a bounded competitive ratio, one has to restrict the power of the adversary by enforcing *locality of requests*, i.e., consecutive requests are only allowed to arrive at a bounded distance to the previous request. We present a general algorithm with a bounded competitive ratio in this setting in [FKMM21].

Further, we applied mobility to the facility location problem [FKM22]. More in detail, we allow an online algorithm to move opened facilities between serving requests at an appropriate cost. We compare the cost of such an algorithm with an optimal solution that knows all requests in advance but generates a static solution in which the facilities are not shifted. In [FKM22], we showed that the lower bound for the classical online facility location problem of $\Omega(\frac{\log n}{\log \log n})$ [Fot08] can be beaten in this model. The lower bound for the classical model implies a dependence on the competitive factor on the number of requests. We were able to remove this dependence completely, leaving the competitive factor to depend only on the parameters of the cost function. Our results are asymptotically optimal.

Heterogeneous requests and resources: Our most recent focus lies on *heterogeneity*, i.e., systems in which the offered resources are not all the same but offer different commodities. Requests in such a system can specify which commodities they are interested in as well. Our extensions capture that the set of services (commodities) managed in our system is heterogeneous, while a joint management often implies lower costs compared to managing each kind of service on its own. For example, deploying multiple different services (commodities) at the same location is usually cheaper than deploying them all on their own, as the common deployment overhead is only paid once. One problem we generalized by commodities is the online facility location problem in [CFK+20]. Formerly, the multi-commodity facility location problem has only been researched in the offline case [RS04]. Here, whenever the algorithm deploys a resource, it has to decide on which commodities to offer at the resource. The offered commodities influence the deployment cost such that offering a combination of commodities in one facility is cheaper than offering the same set of commodities by multiple facilities. Here, the additional difficulty for an online algorithm is to decide on which set of commodities to offer when constructing a facility. In general, our model extension increases the competitive ratio in the lower bound by an additive $\sqrt{|S|}$, where S is the set of commodities. Intuitively, the bound comes from the observation that no online algorithm can efficiently guess the correct set of commodities that is required

by future requests even at a single point. On the algorithmic side, we presented online algorithms, a deterministic and a randomized one, that gain at most an additional factor of $\sqrt{|S|}$ in the competitive ratio for many cost functions. The increase in the competitive ratio however heavily depends on the function describing the cost of the algorithm.

Heterogeneous k-server problems: Another perspective on heterogeneity in resource allocation can be seen in [CFK+22], where we extended the k-server problem. Next, we would like to go into more technical details regarding this publication. The k-server problem was first introduced in 1988 [MMS88], where a lower bound of k for deterministic algorithms on uniform metrics was shown. In uniform metrics, the k-server problem reduces to the paging problem. It was famously conjectured (and not proven up to today) that there is a deterministic algorithm achieving a competitive ratio of k for any metric. In our work in [CFK+22], we assume that the servers are all distinct and any arriving request can either demand to be served by any server (*general requests*) or a specified one (*specific requests*). This extension yields some interesting effects. One might be tempted to assume that the competitive ratio decreases when specific requests appear because a request leaving no choice on which server has to move can be trivially served. Especially, when all requests are specific, a competitive ratio of 1 is easy to achieve by simply moving the servers as dictated by the input. Perhaps surprisingly, instances in which both general and specific requests arrive yield a higher lower bound of $2k - 1$.

The Lower bound: To see this, consider the following instance on a uniform metric with $k + 1$ many locations $v_1, \ldots, v_{k+1}$ and any deterministic online algorithm. Let $s_1, \ldots, s_k$ be the algorithm's servers and $o_1, \ldots, o_k$ be the optimal ones. Initially, assume that $s_i = o_i = v_i$ for all $1 \leq i \leq k$, i.e., the algorithm's servers are exactly at the same position as the optimal ones. Since the algorithm acts deterministic, we can assume that in the following sequence the algorithm moves its servers in the order of the indices. Our input sequence uses general requests at the location that is not covered by the algorithm until the algorithm covers the locations $v_1, \ldots, v_{k-1}, v_{k+1}$. Since the algorithm moves its servers in order, this looks as follows: The first request appears at v_{k+1} and the algorithm moves s_1 there. Then, the next request appears at v_1, and the algorithm either moves s_1 back or it moves s_2 there. Due to the deterministic behavior of the algorithm, there is an ordering of the servers such that it moves them in the order of the indices and hence has k movements until it covers all positions except for v_k. Essentially, the optimal solution only needs to move o_k to v_{k+1} and is done. If we stop the sequence here, we have the original lower bound of k for the classical k-server problem. However, due to specific requests, we can increase the cost of the algorithm even further. Note how the optimal solution has all servers o_i at v_i for $1 \leq i \leq k - 1$. Therefore, the optimal solution does not have to move if we simply add specific requests for all these servers at their initial positions. The algorithm however moves all these servers. Either it already moved a server i twice, or it has to move i due to the specific request back to its initial position. Therefore, for all servers except s_k, the algorithm has a movement cost of at least 2. In total, this yields a cost of the algorithm of $2k - 1$ while the optimal solution only needs one movement.

Intuitively, the lower bound shows that by heterogeneity it is no longer sufficient to cover the same positions as the optimal solution. Further, an online algorithm must converge the location of each of its servers to the matching server in the optimal solution.

A Trade-off: The lower bound we just described uses $2k - 1$ requests in total, k of them being general requests and $k - 1$ of them being specific ones. Consider the ratio of

specific requests requiring a movement vs. all requests requiring a movement. We only consider requests requiring a movement of the algorithm, as all others do not influence the competitive ratio. For the lower bound above, the ratio is $\frac{k-1}{2k-1} \approx \frac{1}{2}$. When there are only general requests, the ratio is 0 and the lower bound is only k. On the other hand, when there are only specific requests, the ratio is 1 and the lower bound is simply 1. In our work, we noticed that the ratio plays an important role in the competitive ratio. Therefore, we established a more general lower bound parameterized in the aforementioned ratio. To see a plot of it, consider Figure 2 below, where s expresses the ratio. As we can see, the starting from k at $s = 0$, the lower bound increases up to roughly $\frac{1}{2}$. Thereafter it very quickly decreases to 1 for $s = 1$.

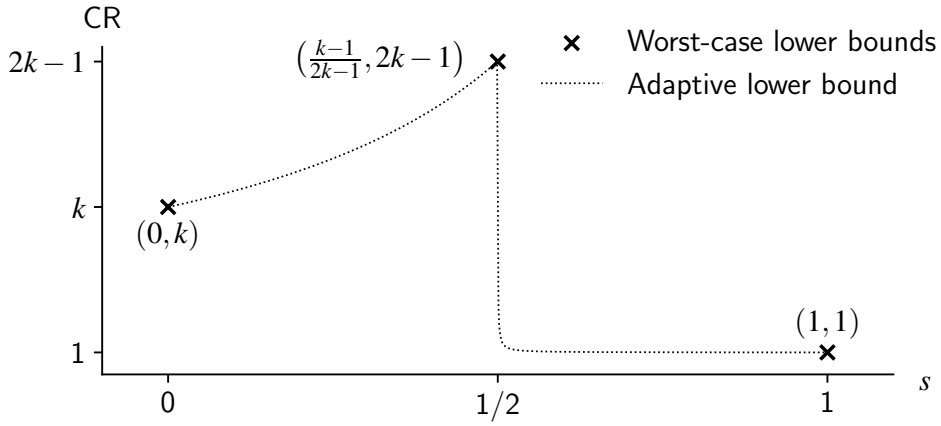

Figure 2: *A plot of our adaptive lower bound of [CFK⁺22]. CR is the competitive ratio and s is the ratio between the number of specific requests requiring a movement and the total number of requests requiring a movement.*

This hints at the following: The adversary in our model only has significant power just until specific requests start to dominate the input sequence. To complement this, we showed algorithms for the uniform metric space that have a tight competitive ratio for $s > \frac{1}{2}$. So, the interesting part in the competitive ratio happens for $s < \frac{k-1}{2k-1}$. Here, the question arises if we can find an algorithm with a tight competitive ratio. We show that no such algorithm can exist. More specifically, no algorithm can have a competitive ratio of k for $s = 0$ and $2k - 1$ for $s \approx \frac{1}{2}$. This is because to achieve a good worst-case competitive ratio close to $2k - 1$, an algorithm has to follow a specific behavioral rule that we establish in [CFK⁺22]. If an algorithm does so, we can show a lower bound in the case of $s = 0$, raising the competitive ratio by one for each server following the rule. If one does not follow the rule, a competitive ratio of k can be achieved for $s = 0$. However, each server not acting according to the rule raises the competitive ratio by one in the worst case.

What we see here is a trade-off that would remain hidden if we did not parameterize in s. In advance, we have to choose if we would like to have a good performance on classical k-server instances, or in the worst case, when specific requests appear. The good news is that we presented two algorithms following this trade-off. The one is optimal for $s = 0$ and achieves a worst-case competitive ratio of $3k - 2$, while the other is close to optimal in the worst-case with a competitive ratio of $2k + 14$ but has at least a competitive ratio of $2k - 1$ if $s = 0$. Both algorithms can be mixed to derive an algorithm that has a fine-tuned competitive ratio dependent on s.

Our research on online resource allocation explored different aspects of the field. We focused on leasing approaches, where resources are leased for a limited time instead of being bought permanently. We then enhanced online algorithms by incorporating limited mobility to test the boundaries when minor adjustments are permitted. Lastly, we expanded classical problems to accommodate heterogeneity, enabling requests and resources to handle multiple commodities.

3 Concluding Remarks

Our research of Subproject A1 of our CRC has significantly extended the state-of-the-art in the area of overlay networks, hybrid networks, and resource allocation problems. We envision hybrid approaches to be an important direction for future research. One particularly interesting direction seems to be hybrid wireless networks. In a hybrid wireless network, the participants can establish local connections via Wi-Fi connections and can also establish arbitrary global connections via some given infrastructure, such as a cellular environment or a satellite. One of the goals we are currently investigating is how to quickly set up routing tables in the nodes with the help of these two communication modes so that messages can be sent along the near-shortest paths in the local network. Solutions to this problem could be used in modern smartphones in order to significantly improve the efficiency of direct interactions between these. Another interesting direction is cloud-assisted overlay networks. The cloud offers a highly scalable solution to classical client-server-based approaches. However, the cloud is not for free. Thus, it would be desirable to use the cloud only to assist an overlay network, instead of taking over its role. Finding the right balance between using the cloud and an overlay network in order to come up with highly scalable and robust solutions for certain applications appears to be a challenging problem.

In our research on heterogeneous resource allocation problems, we explored various extensions. For example, we presented results on uniform metrics for the heterogeneous k-server problem, but further study is needed for more complex metrics. Additionally, considering more complex request types, such as subsets of servers, can significantly increase the problem's difficulty. Similar extensions can be applied to other resource allocation problems like multi-commodity facility location. Our research aims to connect the influence of heterogeneity with classical resource allocation through parameterized competitive ratios. As an example of general ways to step away from a worst-case measure, recently, models became popular where the online algorithm can utilize the advice of a machine learning algorithm to make better choices. Here, the difficulty lies in obtaining improvements when the advice is good while keeping the worst-case guarantees of classical online analysis when the advice is bad. Such techniques allow the research on online algorithms in general to approach the often far better performance observed in practice.

Bibliography

[AAG09] Alon, N.; Azar, Y.; Gutner, S.: Admission control to minimize rejections and online set cover with repetitions. In: *ACM Transactions on Algorithms* 6 (2009), no. 1, 11:1–11:13

[ACD+11] Abshoff, S.; Cord-Landwehr, A.; Degener, B.; Kempkes, B.; Pietrzyk, P.: Local Approximation Algorithms for the Uncapacitated Metric Facility Location Problem in Power-Aware Sensor Networks. In: *Algorithms for Sensor Systems - Proceedings of the 7th International Symposium on Algorithms for Sensor Systems, Wireless Ad Hoc Networks and Autonomous Mobile Entities (ALGOSENSORS)*. Vol. 7111. 2011, pp. 13–27

[AG07] Anthony, B. M.; Gupta, A.: Infrastructure Leasing Problems. In: *Proceedings of the 12th International Conference on Integer Programming and Combinatorial Optimization (IPCO)*. Vol. 4513. 2007, pp. 424–438

[AHK+20] Augustine, J.; Hinnenthal, K.; Kuhn, F.; Scheideler, C.; Schneider, P.: Shortest Paths in a Hybrid Network Model. In: *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2020, pp. 1280–1299

[AKM+16] Abshoff, S.; Kling, P.; Markarian, C.; Meyer auf der Heide, F.; Pietrzyk, P.: Towards the price of leasing online. In: *Journal of Combinatorial Optimization* 32 (2016), no. 4, pp. 1197–1216

[AMM14] Abshoff, S.; Markarian, C.; Meyer auf der Heide, F.: Randomized Online Algorithms for Set Cover Leasing Problems. In: *Proceedings of the 8th International Conference on Combinatorial Optimization and Applications (COCOA)*. Vol. 8881. 2014, pp. 25–34

[BS89] Black, D.; Sleator, D.: *Competitive Algorithms for Replication and Migration Problems*. Technical Report CMU-CS-89-201. Department of Computer Science, Carnegie-Mellon University, Jan. 1989

[CCF+22] Coy, S.; Czumaj, A.; Feldmann, M.; Hinnenthal, K.; Kuhn, F.; Scheideler, C.; Schneider, P.; Struijs, M.: Near-Shortest Path Routing in Hybrid Communication Networks. In: 217 (2022), 11:1–11:23

[CCS+23] Coy, S.; Czumaj, A.; Scheideler, C.; Schneider, P.; Werthmann, J.: *Routing Schemes for Hybrid Communication Networks*. 2023

[CFK+20] Castenow, J.; Feldkord, B.; Knollmann, T.; Malatyali, M.; Meyer auf der Heide, F.: The Online Multi-Commodity Facility Location Problem. In: *Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. July 2020, pp. 129–139

[CFK+22] Castenow, J.; Feldkord, B.; Knollmann, T.; Malatyali, M.; Meyer auf der Heide, F.: The K-Server with Preferences Problem. In: *Proceedings of the 34th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA)*. July 2022, pp. 345–356

[CLP21] Censor-Hillel, K.; Leitersdorf, D.; Polosukhin, V.: Distance Computations in the Hybrid Network Model via Oracle Simulations. In: *Proceedings of the 38th International Symposium on Theoretical Aspects of Computer Science (STACS)*. Vol. 187. 2021, 21:1–21:19

[Dij74] Dijkstra, E. W.: Self-Stabilizing Systems in Spite of Distributed Control. In: *Communications of the ACM* 17 (Nov. 1974), no. 11, pp. 643–644

[FHS20] Feldmann, M.; Hinnenthal, K.; Scheideler, C.: Fast Hybrid Network Algorithms for Shortest Paths in Sparse Graphs. In: *Proceedings of the 24th International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 184. 2020, 31:1–31:16

[FKM22] Feldkord, B.; Knollmann, T.; Meyer auf der Heide, F.: Online facility location with mobile facilities. In: *Theory of Computer Science* 907 (2022), pp. 45–61

[FKMM21] Feldkord, B.; Knollmann, T.; Malatyali, M.; Meyer auf der Heide, F.: Managing Multiple Mobile Resources. In: *Theory of Computing Systems* 65 (2021), no. 6, pp. 943–984

[FM19] Feldkord, B.; Meyer auf der Heide, F.: The Mobile Server Problem. In: *ACM Transactions on Parallel Computing* 6 (2019), no. 3, 14:1–14:17

[Fot08] Fotakis, D.: On the Competitive Ratio for Online Facility Location. In: *Algorithmica* 50 (2008), no. 1, pp. 1–57

[FS19] Feldmann, M.; Scheideler, C.: Skeap & Seap: Scalable Distributed Priority Queues for Constant and Arbitrary Priorities. In: *Proceedings of the 31st ACM on Symposium on Parallelism in Algorithms and Architectures (SPAA)*. 2019, pp. 287–296

[FSS18a] Feldmann, M.; Scheideler, C.; Setzer, A.: Skueue: A Scalable and Sequentially Consistent Distributed Queue. In: *Proceedings of the 2018 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. 2018, pp. 1040–1049

[FSS18b] Feldmann, M.; Scheideler, C.; Setzer, A.: Skueue: A Scalable and Sequentially Consistent Distributed Queue. In: *CoRR* abs/1802.07504 (2018). arXiv: 1802.07504.

[FSS21] Feldmann, M.; Scheideler, C.; Schmid, S.: Survey on Algorithms for Self-stabilizing Overlay Networks. In: *ACM Computing Surveys* 53 (2021), no. 4, 74:1–74:24

[KMP12] Kling, P.; Meyer auf der Heide, F.; Pietrzyk, P.: An Algorithm for Online Facility Leasing. In: *Proceedings of the 19th International Colloquium on Structural Information and Communication Complexity (SIROCCO)*. Vol. 7355. 2012, pp. 61–72

[KS22] Kuhn, F.; Schneider, P.: Routing Schemes and Distance Oracles in the Hybrid Model. In: *Proceedings of the 36th International Symposium on Distributed Computing (DISC)*. Vol. 246. 2022, 28:1–28:22

[Mey01] Meyerson, A.: Online Facility Location. In: *Proceedings of the 42nd Annual Symposium on Foundations of Computer Science (FOCS)*. 2001, pp. 426–431

[Mey05] Meyerson, A.: The Parking Permit Problem. In: *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS)*. 2005, pp. 274–284

[MMS88] Manasse, M. S.; McGeoch, L. A.; Sleator, D. D.: Competitive Algorithms for On-line Problems. In: *Proceedings of the 20th Annual ACM Symposium on Theory of Computing (STOC)*. 1988, pp. 322–333

[NW13] Nagarajan, C.; Williamson, D. P.: Offline and online facility leasing. In: *Discrete Optimization* 10 (2013), no. 4, pp. 361–370

[RS04] Ravi, R.; Sinha, A.: Multicommodity facility location. In: *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*. 2004, pp. 342–349.

[SSS15] Scheideler, C.; Setzer, A.; Strothmann, T.: Towards Establishing Monotonic Searchability in Self-Stabilizing Data Structures. In: *Proceedings of the 19th International Conference on Principles of Distributed Systems (OPODIS)*. Vol. 46. 2015, 24:1–24:17

[SSS16] Scheideler, C.; Setzer, A.; Strothmann, T.: Towards a Universal Approach for Monotonic Searchability in Self-stabilizing Overlay Networks. In: *Proceedings of the 30th International Symposium on Distributed Computing (DISC)*. Vol. 9888. 2016, pp. 71–84

[ST85] Sleator, D. D.; Tarjan, R. E.: Amortized Efficiency of List Update and Paging Rules. In: *Communications of the ACM* 28 (1985), no. 2, pp. 202–208

[STA97] Shmoys, D. B.; Tardos, É.; Aardal, K.: Approximation Algorithms for Facility Location Problems (Extended Abstract). In: *Proceedings of the 29th Annual ACM Symposium on the Theory of Computing (STOC)*. 1997, pp. 265–274