Flexible Virtual Network Embeddings:
It’s About Time!
Technical Report

Matthias Rost, Stefan Schmid
{mrost, stefan}@net.t-labs.tu-berlin.de
Telekom Innovation Laboratories & TU Berlin, Germany

Abstract. Network virtualization is a new networking paradigm which envisions an Internet where arbitrarily specified virtual networks (VNets) can be requested and embedded on a shared physical infrastructure for a desired time period. These V Nets can have different characteristics. For example, a QoS-critical VNet (short: QNet) may impose requirements on the minimal QoS parameters during the entire realization of the VNet (e.g., a VNet for live multi-media conferencing), whereas a more deadline-driven VNet (short: DNet) may only require a certain aggregated amount of resources by some deadline (e.g., a grid-like computational VNet). At the heart of network virtualization lies the question of where and when to embed the different V Nets optimally: Any embedding of the VNet must meet all the required specifications of the VNet, but flexibility in terms of placement location and time can be exploited to optimize the embedding.

This paper focuses on the temporal aspects of the VNet embedding problem. We propose a very general and flexible mathematical programming approach for solving the VNet embedding problem. By using a continuous-time model we are able to compute both optimal and approximative solutions. We analyze our model and show how performance can be improved by user cuts. To complement the theoretical contributions, we show how to use our approach to evaluate the benefit of time flexibilities in the VNet specification.

1 Introduction

Network virtualization promises new innovative Internet services where arbitrary virtual networks (short V Nets) can be requested and embedded at short notice. A VNet is a flexibly specified network connecting virtual node resources (e.g., storage or computation) by virtual links (e.g., with latency or bandwidth guarantees) during a certain time period. Such virtual networks may for example be implemented using VMware technologies (for node virtualization), and VLANs or software-defined networking techniques such as OpenFlow (for link virtualization). Different V Nets co-habit a given physical (or again virtual!) network called the substrate, sharing the given resources while providing isolation and QoS guarantees if needed. The substrate may for example be a data center network, but also a single or multi-ISP wide-area network.

This paper attends to the problem of embedding V Nets on a given substrate. Depending on the application, V Nets can have different requirements where and when they are realized or embedded (i.e., mapped to the substrate). For instance, some VNet requests may come with detailed time or resource requirements: e.g., the time period of a multi-media conference typically cannot be negotiated, and the services offered by a VNet for monitoring and controlling Smart Grid electricity distribution networks are time-critical and require a constant resource reservation throughout the monitoring periods. However, other V Nets can be more flexible: a bulk data transfer or a scientific computation can happen during the night where more resources are available at lower prices. Moreover, the execution of a VNet may be distributed across multiple disjoint time intervals (e.g., over a week but avoiding the high energy prices over lunch time) and a VNet may even be migrated to a different location (within the specification constraints) if resource prices change over time.

1.1 Our Contribution

To the best of our knowledge, this is the first paper on the VNet embedding problem supporting both virtual node mapping and time flexibilities. The contribution is twofold. First, we describe the $\Delta$-approach: a flexible temporal
Mixed Integer Program (MIP) formulation that allows to embed two different types of VNets: QNets (short for Quality-of-Service VNets) which require a constant resource rate during their realization, and DNets (short for Deadline-driven VNets) which only require a certain accumulated amount of resources by some deadline. The MIP approach is motivated by its generality, e.g., in terms of objective functions, and the numerous optimized software solvers (e.g., CPLEX [22]).

While most of the work on virtual network embedding focuses on algorithms solving the static embedding problem, i.e. without any notion of time, we consider in this paper a scenario where requests come with desired durations for their realization, distinct start and end times (or just deadlines), or even a set of flexible intervals (possibly weighted with preferences or prices) in which the VNet embedding is intended. The goal then is to exploit the specification flexibility to increase, e.g., the number of embeddable VNet requests or to minimize load. By incorporating the notion of time, it is possible to give time-dependent objectives as e.g., minimizing the makespan or minimizing the load within an interval for which many succeeding requests are assumed to arrive. Thus, our $\Delta$-approach can be used for short- or mid-term scheduling of requests, and re-embedding of long-lived VNets. Although the focus of this paper is on offline optimization, online optimizations are supported by objective functions prioritizing shorter makespans.

Most notably, our $\Delta$-approach uses a continuous-time model (Sections 2 and 3) which avoids discretization errors, making it possible to model fine grained processes like node migration; a discrete model would result in large approximation errors or could not be used to model the situation at all. In addition to giving optimal solutions for linearized settings, our approach allows the efficient approximation of non-linearities due to under-specified time aspects of VNet requests. Particularly, we present a simple algorithm that allows to $\varepsilon$-approximate an arbitrary objective function using $\sqrt{1/\varepsilon}$ many linearization variables. This paper also discusses performance optimization aspects (Section 5), and to demonstrate the feasibility of our approach, we report on a small experimental study with time-flexible embeddings. This analysis sheds light on the ‘benefit of flexibility’ (Section 6), i.e., how the objective’s value depends on how flexibly the VNet is specified.

1.2 Paper Organization

The remainder of this paper is organized as follows. In Section 2 we introduce the temporal embedding problem and discuss specification opportunities that arise when considering time in the specification of requests. There, we will also give the basic idea for our $\Delta$-approach which will be presented fully in Section 3. In Section 4 we discuss how arising non-linearities can be efficiently resolved by presenting an approximation scheme for DNets. In Section 5 we analyze our MIP formulation and describe how performance can be increased by adding further constraints. In Section 6 we investigate the impact of time flexibilities on the objective function. After reviewing related work in Section 7, the paper concludes in Section 8.

2 Embedding Temporal VNets

In this section we first shortly review the static embedding problem. We proceed by discussing which requirements may arise when considering embeddings over time. Finally, we discuss the two most prominent approaches to incorporate time, namely continuous time and time discretization.

2.1 The Static Embedding Problem

Generally the static VNet embedding problem consists of a set of virtual networks that shall be embedded by a provider on its substrate network. The virtual networks specify resources for both nodes and links. Usually, requested node resources are memory, disk storage and CPU while links are described by means of bandwidth and e.g., the modus operandi like duplex or broadcast. The embedding of a VNet entails the mapping of virtual nodes to substrate nodes and realizing virtual links as flows, possibly consisting of multiple paths, according to the node mapping. Commonly, the task of the provider is then to find a feasible embedding of the requests at hand, such that the costs of the provider itself are minimized. For example, objective functions might be used to maximize energy savings by disabling a set of substrate nodes, or to reduce overall load on the substrate links [20]. Note that already static embedding algorithms can be used to support a limited form of migration: by iteratively solving the embedding problem at different points in time.
2.2 Temporal Specification Opportunities

We will now introduce time aspects in the request specification, and will explore opportunities for flexible specifications.

**Time-varying requirements** A VNet’s state may change within a prescribed realization period. For example a link’s required bandwidth might need to be higher during the daytime than during the night.

**QoS-driven allocation** During the realization of a VNet, the requested amount of resources is constantly reserved at any time, i.e., we have a fixed resource rate (e.g., 10 MFLOPS).

**Deadline-driven allocation** Only aggregate resource quantities may be specified, e.g., the amount of data transferred by some deadline. Deadline-driven allocations allow for a much higher flexibility since the provider may choose the embedding period at its will.

**Supporting flexibility** Independently of whether resources are allocated at constant rates or only by some deadline, a customer may want to specify a set of time intervals during which a VNet is realized. The individually specified intervals may be much shorter or much longer than the actual realization of the VNet, and the VNet may be realized in (parts of) multiple intervals. For instance, in a QoS scenario, a customer may specify that she needs 12 hours of 100 MFLOPS, allocated somewhere between 10pm and 5am, between 9am and 11am, and between 2pm and 5pm, to avoid high energy prices.

**Temporal dependencies** As we envision requests consisting of several parts that may be embedded during different time periods, it is important to allow modeling of temporal dependencies. Consider for example a compute job that is distributed on multiple nodes and some master node that aggregates the results. Obviously the master may not start to aggregate the results before all data was transferred to it.

2.3 A Consistent Specification Approach

As we wish to enable modeling of complex requests, we need a more powerful specification method. Consider e.g. the scenario in which a customer specifies the usage of two nodes with repeated data exchange every hour, such that the volume of the data transfers increases over time but the required computing power stays fixed. An approach for modeling such a scenario would be to give a single virtual topology consisting of two nodes and two links between them. Since the volume of the links varies over time, it becomes a necessity to describe the links behavior for every hour separately. Following this approach, the fact that both nodes need fixed computational power between data exchanges introduces redundancy into the specification, as both nodes are then specified separately.

This leads to the idea to aggregate different network elements into a single request group, if and only if they share the same temporal characteristics, as e.g. the allowed realization period or duration. As parts of a request are now grouped according to their temporal specification, the question arises, how these separate request groups and their underlying virtual graphs shall be related to each other. To this end, we introduce the virtual request graph that gives the whole topology that a customer requests, ignoring temporal aspects and node or link specifications totally. Each request group belonging to the same request is then only allowed to specify a QNet / DNet whose underlying graph is a subgraph of the virtual request graph. We thereby enforce that although changes in specification characteristics of a request might change over time, all request groups are logically related by using the same topology.

As multiple request groups may specify characteristics of the same virtual node at the same time, it is imperative that at any point in time, the mapping of virtual nodes to substrate nodes is consistent, i.e. a virtual node is at any time mapped on exactly one substrate node. Summarizing, we embrace the usage of a virtual request graph to describe the customer’s topology and request groups that specify virtual allocations for parts of this virtual topology with the same temporal specification (see Figure[1] for an exemplary specification).

2.4 Deriving the Continuous $\Delta$-Approach

The statically embedding of VNets consists of three main parts:

1. Map virtual nodes on substrate nodes.
2. Based on this mapping, calculate multi-commodity flows in the substrate to embed links.
Virtual Request Graph

Fig. 1: Informal example for a request and its subdivision into request groups.

3. Aggregate allocated substrate resources over all requests and check that no over-allocation occurs.

When considering temporal VNets, two major differences need to be accounted for. Firstly, additionally to deciding where (node placements) and how (link realization) the request shall be realized, feasible times for the start and end time of the request must be found. Secondly, checking feasibility of allocations cannot be done globally but must be ensured for every point in time.

Assuming that request’s allocations on the substrate do not change during the realization period, the task of checking feasibility for infinitely many points in time (for the continuous model) can be reduced to checking feasibility immediately after the start of a request, since feasibility for this instance of time implies feasibility until the next substrate’s state modification. Thus, the intricate part of the mathematical formulation boils down to making the substrate’s state apparent for each instance in time at which a request gets embedded.

We solve this pivotal problem in the following way. Assume that for each request group a feasible start and end time is given. Neglecting the case in which request groups’ start or end times may coincide, these events define a linear order on the substrate changes. Given that linear order, the substrate’s state can be inductively derived by giving an initial state and at each event, adding or subtracting the local allocations of the request that corresponds to the current event point. Naming our approach $\Delta$-approach therefore reflects that state allocations are computed by means of incremental local changes. Figure 2 visualizes this main idea.

3 The MIP Formulation

In this section we present our main body of work, namely the continuous $\Delta$-approach for computing optimal temporal embeddings. Key features of our MIP formulation are:
Fig. 2: Simplified example for embedding two QNets \( r_{qi} \), \( i \in \{1, 2\} \). On the top, the temporal specification of both requests is given. Given a cyclic substrate with three nodes, we assume that both requests can be figuratively embedded as depicted by \( \Delta(\text{req}_i) \). Note that the allocations \( \Delta(\text{req}_i) \) are invariant during the realization period. A filled node and a solid line mark a node or link on which resources are allocated. In contrast, an unfilled node or a dotted line mark nodes or links that are not used. By assuming valid realization periods for both \( \text{req}_i \), a linear order is derived by associating the start and end of the request groups’ with events. Given this correspondence the state of the substrate can be inductively calculated by adding or subtracting the local allocations of the respective requests.

**Flexibility for Customer** Based on the decomposition of requests into multiple request groups, our formulation allows for a wide variety of customer scenarios. Within a request, we allow the customer to give multiple possible realization periods for its request groups. The customer might specify temporal dependencies between request groups. Furthermore, by using DNets, the customer can specify aggregate resource allocations over time.

**Flexibility for Provider** Our model features access control, i.e. possibly rejecting requests, and can be extended easily for various objective functions.

**Accuracy** In contrast to discrete models our continuous approach allows for computing very accurate solutions as no discretization errors are introduced. The only source for errors is the linearization of DRGs (short for deadline-driven request group). However, we derive a linearization scheme such that the provider can control this error (see Section 4).

### 3.1 Notation

We use **boldface** to denote the input comprising sets and parameters. For directed edges \( e = (s, t) \in E \subseteq V^2 \) we use head : \( E \to V \), head(\( e \)) = s and analogue tail : \( E \to V \), tail(\( e \)) = t to denote the head and respectively the tail of an edge. If \( E \subseteq V^2 \) is a set of directed edges, then \( \delta^+_E(v) = (\{v\} \times V) \cap E \) and \( \delta^-_E(v) = (V \times \{v\}) \cap E \) denote the outgoing and the incoming edges respectively.

We set \( \Gamma = \{[a, b] \in \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} | a < b \} \) to denote all closed time intervals and use \( \mathbb{B} = \{0, 1\} \) to denote binary values. If \( \chi \in \mathbb{B} \) is a binary variable, then we use the notation \( \overline{\chi} \) to denote logical negation, i.e. \( \overline{\chi} := (1 - \chi) \).
3.2 Input Parameter

The substrate graph is modeled as directed graph. For each node and each link a capacity is specified. The goal is to find a temporal embedding such that at any time, the load on the node or the link must be less than or equal to this specified capacity. Timehorizon is the upper bound on time values we will be able to represent in our model (see Table 1).

<table>
<thead>
<tr>
<th>Table 1: Definition and specification of the substrate</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Timehorizon</strong> ∈ ( \mathbb{R} \geq 0 )</td>
</tr>
<tr>
<td><strong>( V_S )</strong></td>
</tr>
<tr>
<td><strong>( E_S )</strong> ⊆ ( V_S \times V_S )</td>
</tr>
<tr>
<td><strong>( sse )</strong> : ( E_S \rightarrow \mathbb{R}^+ )</td>
</tr>
<tr>
<td><strong>( ssV )</strong> : ( V_S \rightarrow \mathbb{R}^+ )</td>
</tr>
</tbody>
</table>

As discussed in Section 2 each customer’s request consists of a virtual request graph \((V_{req}, E_{req})\) and possibly several request groups. We allow two types of request groups, namely QRGs corresponding to QNets and DRGs corresponding to DNets. \(\Gamma_{rg}\) defines a non-empty set of intervals for each request group, such that a request group’s realization period must lie within at least one of these given intervals. Using the set of interval identifiers \(IID_{req}\) to denote the start or the end of a request group, we allow for temporal dependencies \(dep_{req}\) between request groups of the same request. Finally, as we support access control and therefore requests may not be embedded, **must_embed** allows for requiring an request to be embedded at all costs (see Table 2).

<table>
<thead>
<tr>
<th>Table 2: General Request Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Reqs</strong></td>
</tr>
<tr>
<td><strong>RG_{req}</strong> = ( QRG_{req} \cup DRG_{req} ) ( \forall req \in \text{Reqs.} )</td>
</tr>
<tr>
<td><strong>RG</strong> = ( \bigcup \forall req \in \text{Reqs} )</td>
</tr>
<tr>
<td>( V_{req} ) ( \forall req \in \text{Reqs.} )</td>
</tr>
<tr>
<td>( V_{rg} \subseteq V_{req} ) ( \forall req \in \text{Reqs.} \forall rg \in \text{RG}_{req} )</td>
</tr>
<tr>
<td>( E_{rg} \subseteq V_{rg} \times V_{rg} ) ( \forall rg \in \text{RG} )</td>
</tr>
<tr>
<td>( \Gamma_{rg} \subseteq \Gamma, \neq \emptyset ) ( \forall rg \in \text{RG} )</td>
</tr>
<tr>
<td>( IID_{req} = RG_{req} \times {\text{start}, \text{end}} ) ( \forall req \in \text{Reqs.} )</td>
</tr>
<tr>
<td>( dep_{req} \subseteq IID_{req} \times IID_{req} \times \mathbb{R} ) ( \forall req \in \text{Reqs.} )</td>
</tr>
<tr>
<td><strong>must_embed</strong>: ( \text{Reqs} \rightarrow \mathbb{B} )</td>
</tr>
</tbody>
</table>

A QRG specifies allocation rates for each of the nodes and links contained within its sub-graph \((V_{qrg}, E_{qrg}) \subseteq (V_{req}, E_{req})\). Additionally, a fixed duration is given. The provider must embed the given QRG for exactly this duration (see Table 3).

<table>
<thead>
<tr>
<th>Table 3: Specification of QRGs</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>( qsdur )</strong> : ( QRG_{req} \rightarrow \mathbb{R}^+ \setminus {0} ) ( \forall req \in \text{Reqs.} )</td>
</tr>
<tr>
<td><strong>( qsv )</strong> : ( V_{qrg} \rightarrow \mathbb{R}^+ ) ( \forall req \in \text{Reqs.} \forall qrg \in \text{QRG}_{req} )</td>
</tr>
<tr>
<td><strong>( qsl )</strong> : ( E_{qrg} \rightarrow \mathbb{R}^+ ) ( \forall req \in \text{Reqs.} \forall qrg \in \text{QRG}_{req} )</td>
</tr>
</tbody>
</table>
In contrast to QRGs, DRGs only specify an aggregate resource allocation over time. Thus, DRGs differ from QRGs in that they do not specify a fixed duration but an interval \( ds_{\text{dur}} \) in which the duration must lie. Therefore, the provider may vary both the duration and the rate of allocations such that the product of duration and rate is higher than the requested aggregate amount of resources. This non-linearity must be linearized in our MIP approach.

Assuming that the customer wants to realize a data transfer of 5 GB and specifies that this transfer shall take no longer than 10 hours but also not less than half an hour. We approximate the resulting function \( r : [0.5, 10] \rightarrow \mathbb{R} \geq 0, d \mapsto \frac{5}{d} \) by a piecewise linear function, which can be easily implemented in a MIP. Importantly, as the provider must guarantee that at least 5 GB are transferred, the approximation must always lie above (the optimal) function \( r \) (see Figure 3 for an example).

In Section 4 we will describe in detail how such piecewise linear functions can be determined to gain a constant error approximation. For now we assume that the piecewise approximation is already given by duration sampling points called Duration Linearization Points (see Table 4).

Table 4: Specification of DRGs

<table>
<thead>
<tr>
<th>( ds_{\text{dur}} ) : DRG(_{\text{req}} \rightarrow \Gamma )</th>
<th>( \forall \text{req} \in \text{Reqs} ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>DLP(<em>{\text{req}} = {d</em>{\text{lp}}<em>1, \ldots, d</em>{\text{lp}}_{</td>
<td>\text{DLP}_{\text{req}}</td>
</tr>
<tr>
<td>( d_{\text{SV}} ) : V(_{\text{req}} \rightarrow \mathbb{R}^+ )</td>
<td>( \forall \text{req} \in \text{Reqs}. \forall \text{drg} \in \text{DRG}_{\text{req}} ). Aggregate node resource specification over time (e.g. 10 GFLOP hours)</td>
</tr>
<tr>
<td>( d_{\text{SL}} ) : E(_{\text{req}} \rightarrow \mathbb{R}^+ )</td>
<td>( \forall \text{req} \in \text{Reqs}. \forall \text{drg} \in \text{DRG}_{\text{req}} ). Aggregate link resource specification over time (e.g. 10 GB)</td>
</tr>
</tbody>
</table>

For the sake of linearly ordering the request groups according to their start and end times, we introduce the abstract set of Events such that each request group start and end will be assigned to exactly one event. For describing substrate allocations between events, we introduce the set of States (see Table 5).

Table 5: Definition of abstract event point and state sets

| Events = \( \{e_1, \ldots, e_{|\text{RG}|}\} \) | Set of abstract events |
| States = \( \{s_{0,1}, s_{1,2}, \ldots, s_{2,|\text{RG}|+2}, |\text{RG}|+1\} \) | Set of states between events |
3.3 Variables

We will now introduce the variables used in our MIP formulation. As we consider scenarios in which access control, i.e. the possibility to reject requests, is supported, we use a binary variable embedded to indicate whether a request is embedded or not. If a request is not embedded, then the request may not acquire substrate resources, otherwise all its request groups must be embedded. Binary variable $\chi_V$ indicates on which substrate node a virtual node is mapped. Note that the mapping of virtual nodes is invariant for all request groups as it is specified for the request itself. Thereby, all request groups will argue about the same substrate nodes via their virtual aliases. Virtual links are embedded as (multi-commodity) flows. For that purpose, we introduce the variable flow$_{rg}$ which gives for each virtual link the allocations made on the substrate links (see Table 6).

Table 6: Local mapping variables and the flow allocation in the substrate

<table>
<thead>
<tr>
<th>embedded: Reqs $\rightarrow$ $\mathbb{B}$</th>
<th>Indicated whether the request is embedded or not</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\chi_V$ Vreq $\times$ VS $\rightarrow$ $\mathbb{B}$ $\forall$req $\in$ Reqs.</td>
<td>Mapping of virtual nodes on substrate nodes</td>
</tr>
<tr>
<td>flow$<em>{rg}$ E$</em>{req}$ $\times$ E$<em>S$ $\rightarrow$ $\mathbb{R}</em>{\geq 0}$ $\forall$req $\in$ Reqs.$\forall$rg $\in$ RG$_{req}$.</td>
<td>Flow allocations on substrate link for each virtual link</td>
</tr>
</tbody>
</table>

Additional variables are needed for realizing allocations of DRGs. Following the linearization approach presented above, we use $\lambda_{drg}$ and $\chi_{drg}$ for selecting a convex combination of neighboring duration linearization points, yielding the pair of durations and corresponding rates. As the rates are computed in this fashion, we introduce further variables for assigning those selected rates to nodes and links respectively (see Table 7).

Table 7: Additional variables for setting DRG allocations.

| $\lambda_{drg}$ DLP$_{drg} \rightarrow [0,1]$ $\forall$req $\in$ Reqs.$\forall$drg $\in$ DRG$_{req}$. | Give a convex combination of neighboring duration linearization points to yield the chosen duration. |
| $\chi_{drg}$ $\{i | 1 \leq i < |DLP_{drg}|\} \rightarrow \mathbb{B}$ $\forall$req $\in$ Reqs.$\forall$drg $\in$ DRG$_{req}$. | Assume that $|DLP_{drg}|$ many duration linearization points are given. Any two neighboring sampling points induce a line. The following variables are used to select exactly one piece of the $|DLP_{drg}| - 1$ many line segments. |

| node alloc$_{c}$ V$_{drg}$ $\times$ VS $\rightarrow \mathbb{R}_{\geq 0}$ $\forall$req $\in$ Reqs.$\forall$drg $\in$ DRG$_{req}$. | Allocated node resources (rate) of virtual node on substrate node |
| node flow$^+$ E$_{drg}$ $\times$ VS $\rightarrow \mathbb{R}_{\geq 0}$ $\forall$req $\in$ Reqs.$\forall$drg $\in$ DRG$_{req}$. | Outgoing flow (rate) at substrate node for each virtual link |
| node flow$^-$ E$_{drg}$ $\times$ VS $\rightarrow \mathbb{R}_{\leq 0}$ $\forall$req $\in$ Reqs.$\forall$drg $\in$ DRG$_{req}$. | Incoming flow (rate) at substrate node for each virtual link |

Using the mapping variables $\chi_+$ and $\chi_-$ the start and end of request groups are assigned to events. Events are attributed with a point in time, given by $t_{\text{event}}$. The realization period of a request group rg is defined by the interval $[t_+(rg), t_-(rg)]$. The variables $\chi_{rg}$ will be used for deciding whether each request group lies in one of the predefined intervals. For each event the variables $\Delta_E$, $\Delta_V$ give the change of resource allocation (rates) implied by the assignment of a request group start or end. Note that these variables are unbounded. Positive changes reflect a request group start while negative changes reflect that a request group end occurs. Each event is naturally attributed with a time value corresponding to either $t_+$ or $t_-$ of some request group. Using $\Delta_E$ and $\Delta_V$ the substrate’s allocations at every state will be stored using link alloc and node alloc. (see Table 8).

3.4 Constraints

In this section we will give all constraints needed to describe feasible temporal embeddings. The constraints will be introduced stepwise. We begin by discussing access control in Section 3.4.1. To derive local allocations in Section 3.4.2 we firstly discuss how the duration, and thereby the rate, of DRGs can be calculated in Section 3.4.3. In Section 3.4.4 we then map the request groups to events to derive a linear order on the request group start and end events. Based on
Table 8: Event Point, State and Time Variables

<table>
<thead>
<tr>
<th>Variable</th>
<th>Domain</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{\text{event}} )</td>
<td>( \text{Events} \rightarrow \mathbb{R}_{\geq 0} )</td>
<td>Gives the point in time for events.</td>
</tr>
<tr>
<td>( t_+ )</td>
<td>( \text{RG} \rightarrow \mathbb{R}_{\geq 0} )</td>
<td>Point in time at which the request group is embedded</td>
</tr>
<tr>
<td>( t_- )</td>
<td>( \text{RG} \rightarrow \mathbb{R}_{\geq 0} )</td>
<td>Point in time at which the request group’s embedding is ended</td>
</tr>
<tr>
<td>( \chi_{+,\text{rg}} )</td>
<td>( \Gamma</td>
<td></td>
</tr>
</tbody>
</table><p>ightarrow \mathbb{B} ) | Binary variable deciding whether the request group lies within a specified time interval. |
| ( \chi_+ ) | ( \text{Events} \times \text{RG} \rightarrow \mathbb{B} ) | Variable indicating whether the start of a request group is mapped on an event. |
| ( \chi_- ) | ( \text{Events} \times \text{RG} \rightarrow \mathbb{B} ) | Variable indicating whether the end of a request group is mapped on an event. |
| ( \Delta_E ) | ( \text{Events} \times \text{ES} \rightarrow \mathbb{R} ) | Substrate link allocation changes at event. If positive, then allocations are added. If negative, then allocations are freed. |
| ( \Delta_V ) | ( \text{Events} \times \text{VS} \rightarrow \mathbb{R} ) | Substrate node allocation changes at event. If positive, then allocations are added. If negative, then allocations are freed. |
| link_alloc | ( \text{States} \times \text{ES} \rightarrow \mathbb{R}<em>{\geq 0} ) | Global substrate link allocations at each state (i.e., between two event points) |
| node_alloc | ( \text{States} \times \text{VS} \rightarrow \mathbb{R}</em>{\geq 0} ) | Global substrate node allocations at each state (i.e., between two event points) |</p>

For this mapping to abstract event points, we show how to calculate time points for the start and end of request groups to allow checking of temporal feasibility in Section 3.4.5. Finally, we will derive the state changes at events based on the local allocations in Section 3.4.6 to then check feasibility of the allocations in the substrate in Section 3.4.7.

3.4.1 Access Control

As mentioned earlier, our model features access control, i.e. the ability to reject a customer’s request. Naturally, the constraints dealing with the embedding depend crucially on whether the request is embedded or not. Following the methodology introduced in Section 2.4, the embedding is calculated based on the mapping to nodes. Therefore, if a request is not embedded nodes are not mapped (see Table 9).

Table 9: Access control and mapping of nodes

_\text{SET \ REQUESTS WHICH MUST BE EMBEDDED}_

\( \forall \text{req} \in \text{Reqs}. \quad \text{embedded}(\text{req}) \geq \text{must\_embed}(\text{req}) \)

_\text{MAP \ NODES}_

\( \forall \text{req} \in \text{Reqs}. \forall \text{N}_v \in \text{V}_{\text{req}}. \quad \text{embedded}(\text{req}) = \sum_{\text{N}_s \in \text{VS}} \chi_V(\text{N}_v, \text{N}_s) \)

_\text{NON-MAPPED REQUESTS DO NOT ACQUIRE LINK RESOURCES}_

\( \forall \text{req} \in \text{Reqs}. \forall \text{rg} \in \text{RG}_{\text{req}}. \forall \text{L}_s \in \text{ES}. \quad \sum_{\text{L}_v \in \text{L}_{\text{rg}}} \text{flow}_{\text{rg}}(\text{L}_v, \text{L}_s) \leq \text{embedded}(\text{req}) \cdot \text{SS}_{\text{E}}(\text{L}_s) \)

3.4.2 Choosing Durations for DRGs

The allocation rate of DRGs for nodes and links depend on the chosen duration. Using the variables \( \lambda_{\text{drg}} \) a convex combination corresponding to the duration linearization points is constructed. However, if the request is not embedded, then \( \sum_{\text{drg} \in \text{DLP}_{\text{drg}}} \lambda_{\text{drg}} = 0 \), which will be used in later constraints to set resource allocations to zero. The convex combination is restricted in such a way that maximal two adjacent duration linearization points may be used for calculating the duration, thereby describing a piecewise linear function (see Table 10).
Table 10: Selecting a duration using convex combinations of the duration linearization points

**REQUIRE_CONVEX_COMBINATION**

**General:** If the request at hand is embedded, then the \( \lambda_{drg} \) variables must constitute a convex combination. If the request is not embedded, then all \( \lambda_{drg} \) are set to zero.

\[
\forall \text{req} \in \text{Reqs}. \forall \text{drg} \in \text{Drg}_{\text{req}}. \sum_{\text{dlp}_i \in \text{Dlp}_{\text{drg}}} \lambda_{drg}(\text{dlp}_i) = \text{embedded}(\text{req})
\]

**SELECT_EXACTLY_ONE_PIECE**

**General:** If the request is embedded, then exactly one of the \( \chi_{drg} \) variables is set to one.

\[
\forall \text{req} \in \text{Reqs}. \forall \text{drg} \in \text{Drg}_{\text{req}}. \sum_{|\text{Dlp}_{\text{drg}}| - 1}^{i=1} \chi_{drg}(i) = \text{embedded}(\text{req})
\]

**RESTRICT_TO_NEIGHBORING_POINTS_1**

**General:** Guarantees that the first duration linearization point may only be used in the convex combination if the first line segment is selected.

\[
\forall \text{req} \in \text{Reqs}. \forall \text{drg} \in \text{Drg}_{\text{req}}. \lambda_{drg}(\text{dlp}_1) \leq \chi_{drg}(1)
\]

**RESTRICT_TO_NEIGHBORING_POINTS_2**

**General:** Guarantees that an intermediate duration linearization point may only be used in the convex combination if either its right or its left line segment is selected.

\[
\forall \text{req} \in \text{Reqs}. \forall \text{drg} \in \text{Drg}_{\text{req}}. \forall \text{dlp}_i \in \text{Dlp}_{\text{drg}} \text{ with } 1 < i < |\text{Dlp}_{\text{drg}}|. \lambda_{drg}(\text{dlp}_i) \leq \chi_{drg}(i - 1) + \chi_{drg}(i)
\]

**RESTRICT_TO_NEIGHBORING_POINTS_3**

**General:** Guarantees that the last duration linearization point may only be used in the convex combination if the first line segment is selected.

\[
\forall \text{req} \in \text{Reqs}. \forall \text{drg} \in \text{Drg}_{\text{req}}. \lambda_{drg}(\text{dlp}_{|\text{Dlp}_{\text{drg}}|}) \leq \chi_{drg}(|\text{Dlp}_{\text{drg}}| - 1)
\]

3.4.3 Local Allocations In the following we show how local allocations can be computed for both QRGs and DRGs. For this, assume that the embedded and \( \chi_V \) variables are fixed. Let us first consider local allocations of QRGs. Firstly, note that node allocations are already fixed by mapping the virtual nodes on substrate nodes and therefore do not need to be calculated. To realize virtual links, we must induce corresponding flows in the substrate (see Table 11).

Table 11: Inducing multi-commodity flows in the substrate (QRG)

**GUARANTEE_FLOW_PRESERVATION_Q**

**General:** Induces the appropriate amount of flow in the local flow allocation variables \( \text{flow}_{\text{qrg}} \).

**Technical:** If \( N_s \) is neither used as source nor sink, then the node flow must be zero (preservation). If \( N_s \) is both source and sink, then the rhs is zero and therefore no flow is created. If \( N_s \) is either the source or the sink then it will send or receive the amount of flow necessary.

\[
\forall \text{req} \in \text{Reqs}. \forall \text{qrg} \in \text{Qrg}_{\text{req}}. \forall L_v \in \text{E}_{\text{qrg}}. \forall N_s \in \text{V}_{\text{S}} \sum_{L_e \in \text{E}_{\text{qrg}}(N_s)} \text{flow}_{\text{qrg}}(L_v, L_s) - \sum_{L_e \in \text{E}_{\text{qrg}}(N_s)} \text{flow}_{\text{qrg}}(L_v, L_s) = \text{qs}_{E}(L_v) \cdot (\chi_V(\text{tail}(L_v), N_s) - \chi_V(\text{head}(L_v), N_s))
\]

Constructing the appropriate allocations for DRGs is slightly more difficult. First of all, we consider node allocations. Using the variable \( \text{node}_{\text{alloc}_{\text{drg}}} \) the allocation rates of each virtual node on each substrate node is explicitly calculated (see Table 12),
we first derive the flow balance of each virtual link on each substrate node. Incoming and outgoing flow must be determined by \( \forall \) and technical:

\[
\text{FORBID_NODE_ALLOC_FOR_NON_MAPPED_D}
\]

General: If a virtual node is not mapped on a substrate node it may not acquire resources.

Technical: Guarantees that node alloc \( \forall \varepsilon \in \text{DRG}_{\text{req}}, \forall N_v \in V_{\text{drg}} \cdot \forall N_s \in V_{\text{S}} \) node alloc \( \forall \varepsilon \in \text{DRG}_{\text{req}}, \forall N_v \in V_{\text{drg}} \cdot \forall N_s \in V_{\text{S}} \)

\[
\text{node alloc}_d (N_v, N_s) \leq \chi_V(N_v, N_s) \cdot \text{ss}_V(N_s)
\]

\[
\text{GUARANTEE_NODE_ALLOCATIONS_D}
\]

General: Induces the allocation of node resources for all virtual nodes on substrate nodes corresponding to the convex combination of the chosen duration.

Technical: By the above constraint FORBID_NODE_ALLOC_FOR_NON_MAPPED_D we have specified that the allocations must be zero for non-mapped nodes. As a virtual node can only be mapped on at most one node, the whole resources are allocated on this single substrate node. Note that if the request is not embedded at all, then the convex combination will be zero. Thus, no allocation will be incurred in this case.

\[
\forall \varepsilon \in \text{Reqs}, \forall \varepsilon \in \text{DRG}_{\text{req}}, \forall N_v \in V_{\text{drg}} \cdot \forall N_s \in V_{\text{S}} \cdot \sum_{N_s \in V_{\text{S}}} \text{node alloc}_d (N_v, N_s) = \sum_{\lambda_{\varepsilon} \in \text{DLP}_{\text{ag}}} \lambda_{\varepsilon} \cdot \frac{\text{ds}_V(L_s)}{\text{dl}_p}
\]

We consider now how virtual links of DRGs can be realized. As link rates are subject to the selected durations, we first derive the flow balance of each virtual link on each substrate node. Incoming and outgoing flow must be considered separately to allow the case in which both head and tail of a virtual link are mapped on the same substrate node, as then no flow shall be incurred (see Table 13).

\[
\text{SET_FLOW_BALANCE_NON_MAPPED_NODES_D}
\]

General: If the head (or tail) of a virtual link is not mapped on a substrate node, then it will not absorb (or emit) flow.

Technical: The constant \( \sum_{L_s \in E_{\text{S}} \text{ with head}(L_s) = N_s} \text{ss}_E(L_s) \) can be replaced by \( \text{ds}_E / \text{dl}_p \), if this yields a better bound.

\[
\forall \varepsilon \in \text{Reqs}, \forall \varepsilon \in \text{DRG}_{\text{req}}, \forall L_v \in E_{\text{drg}} \cdot \forall N_s \in V_{\text{S}} \cdot \text{nodeflow}_{\varepsilon}^+(L_v, N_s) \leq \chi_V(\text{head}(L_v), N_s) \cdot \sum_{L_s \in E_{\text{S}} \text{ with head}(L_s) = N_s} \text{ss}_E(L_s)
\]

\[
\forall \varepsilon \in \text{Reqs}, \forall \varepsilon \in \text{DRG}_{\text{req}}, \forall L_v \in E_{\text{drg}} \cdot \forall N_s \in V_{\text{S}} \cdot \text{nodeflow}_{\varepsilon}^-(L_v, N_s) \geq -\chi_V(\text{tail}(L_v), N_s) \cdot \sum_{L_s \in E_{\text{S}} \text{ with tail}(L_s) = N_s} \text{ss}_E(L_s)
\]

\[
\text{SET_FLOW_BALANCE_MAPPED_NODES_D}
\]

Technical: See constraint GUARANTEE_NODE_ALLOCATIONS_D in Table 12.

\[
\forall \varepsilon \in \text{Reqs}, \forall \varepsilon \in \text{DRG}_{\text{req}}, \forall L_v \in E_{\text{drg}}, \forall N_s \in V_{\text{S}} \cdot \text{nodeflow}_{\varepsilon}^+(L_v, N_s) = \sum_{N_s \in V_{\text{S}}} \text{nodeflow}_{\varepsilon}^+(L_v, N_s) + \sum_{\lambda_{\varepsilon} \in \text{DLP}_{\text{ag}}} \lambda_{\varepsilon} \cdot \frac{\text{ds}_V(L_s)}{\text{dl}_p}
\]

\[
\text{GUARANTEE_FLOW_PRESEvation_D}
\]

General: Induces the flow determined by \( \text{nodeflow}_{\varepsilon}^+ \) and \( \text{nodeflow}_{\varepsilon}^- \) in the substrate.

Technical: Note that outgoing flow is always positive, while incoming flow is always less or equal to zero.

\[
\forall \varepsilon \in \text{Reqs}, \forall \varepsilon \in \text{DRG}_{\text{req}}, \forall L_v \in E_{\text{drg}} \cdot \forall N_s \in V_{\text{S}} \cdot \text{nodeflow}_{\varepsilon}^+(L_v, N_s) + \text{nodeflow}_{\varepsilon}^-(L_v, N_s) = \sum_{L_s \in E_{\text{S}}(N_s)} \text{flow}_{\varepsilon}(L_v, L_s) - \sum_{L_s \in E_{\text{S}}(N_s)} \text{flow}_{\varepsilon}(L_v, L_s)
\]
3.4.4 Mapping of Request Groups to Events  Each request group’s start and end must be mapped on some event point. We require this, for reasons explained later on, even if the request itself is not embedded at all. Constraint \text{CALCULATE\_TIME\_POINTS\_FOR\_RG} relates the time of events and the time or the corresponding request group start or end time by giving a tight bound if $\chi_+$ or respectively $\chi_-$ is set to 1. Otherwise, by adding or subtracting Timehorizon, the constraints do not actually constrain the set of solutions (see Table [14]).

Table 14: Associating request groups with events

\begin{align*}
\text{ASSOCIATE\_REQUEST\_GROUPS\_WITH\_EVENTS} \\
\text{General: Each start and end of a request group is associated with exactly one event point. Note that request groups must be mapped to events, even though they might not be embedded.} \\
\forall \text{rg} \in \text{RG}, \\
\sum_{e_i \in \text{Events}} \chi_+(e_i, \text{rg}) = 1 \\
\sum_{e_i \in \text{Events}} \chi_-(e_i, \text{rg}) = 1
\end{align*}

\begin{align*}
\text{ASSOCIATE\_ONLY\_ONE\_REQUEST\_GROUP\_WITH\_EVENT} \\
\text{General: Each event shall only be associated with one request group.} \\
\forall e_i \in \text{Events}, \\
\sum_{\text{rg} \in \text{RG}} (\chi_+(e_i, \text{rg}) + \chi_-(e_i, \text{rg})) = 1
\end{align*}

3.4.5 Guaranteeing Temporal Feasibility As we use the mapping to events to gain a linear order, the point of time of the events must be ordered monotonically. Using the assignment of request groups to events, we can derive the actual start and end times for the request groups (see Table [15]).

Table 15: Guarantee temporal ordering of event points and derive start and end times for request groups

\begin{align*}
\text{EVENT\_POINTS\_ARE\_ORDERED\_MONOTONICALLY} \\
\text{Technical: We enforce the monotonical ordering of the event points’ time points according to the events’ ordering.} \\
\forall i \in \{1, 2, ..., |\text{Events}| - 1\}, \\
t_{\text{event}}(e_i) \leq t_{\text{event}}(e_{i+1})
\end{align*}

\begin{align*}
\text{CALCULATE\_TIME\_POINTS\_FOR\_RG} \\
\text{General: Using the assignment of request groups to events, we can derive the start and end times.} \\
\forall \text{rg} \in \text{RG}, \forall e_i \in \text{Events}, \\
t_+ (\text{rg}) \leq t_{\text{event}}(e_i) + \chi_-(e_i, \text{rg}) \cdot \text{Timehorizon} \\
t_+ (\text{rg}) \geq t_{\text{event}}(e_i) - \chi_+(e_i, \text{rg}) \cdot \text{Timehorizon} \\
t_- (\text{rg}) \leq t_{\text{event}}(e_i) + \chi_+(e_i, \text{rg}) \cdot \text{Timehorizon} \\
t_- (\text{rg}) \geq t_{\text{event}}(e_i) - \chi_-(e_i, \text{rg}) \cdot \text{Timehorizon}
\end{align*}

Next we enforce that the realization period of QRGs corresponds to the specification $q_{\text{dur}}$ and that the chosen duration of DRGs is correctly reflected by their corresponding event points’ times (see Table [16]).

Table 16: Relate realization periods of request groups to the specification

\begin{align*}
\text{CALCULATE\_REALIZATION\_PERIOD\_Q} \\
\text{General: Guarantee that QRGs are embedded for the specified duration.} \\
\forall \text{req} \in \text{Reqs}, \forall \text{qrg} \in \text{QRG}_\text{req}, \\
q_{\text{dur}}(\text{qrg}) = t_- (\text{qrg}) - t_+ (\text{qrg}) \\
\text{(continues on next page)}
\end{align*}
Table 16: Relate realization periods of request groups to the specification (continued)

**CALCULATE_REALIZATION_PERIOD_D**

**General:** Guarantee that the realization period of DRGs corresponds to the duration chosen by the convex combination of the duration linearization points.

**Technical:** If the request at hand is not embedded, then $\sum_{dlp, i \in DLP_{drg}} \lambda_{drg}(dlp_i) \cdot dlp_i = 0$ holds, since each $\lambda_{drg}(dlp_i)$ is set to zero. Therefore, in this case the constraints are disabled by either adding or subtracting $Timehorizon$.

$\forall req \in Reqs. \forall drg \in DRG_{req}$,

$$\sum_{dlp, i \in DLP_{drg}} \lambda_{drg}(dlp_i) \cdot dlp_i + \text{embedded}(req) \cdot Timehorizon \geq t_-(drg) - t_+(drg)$$

$$\sum_{dlp, i \in DLP_{drg}} \lambda_{drg}(dlp_i) \cdot dlp_i - \text{embedded}(req) \cdot Timehorizon \leq t_-(drg) - t_+(drg)$$

Having guaranteed that the request groups realization periods are correct, we can now check whether each request group lies in at least one of the predefined intervals $\Gamma_{tg}$ (see Table 17).

**Table 17:** Guarantee lying in specified intervals

**FORBID_ERRONEOUS_ASSIGNMENT_TO_INTERVAL**

**Technical:** A request group does not lie within an interval $\iota = [a, b]$ if either $t_+(rg) < a$ or $t_-(rg) > b$. In both cases we enforce, that the indicator variable for lying in interval $\iota$ is set to zero. On the other hand, if neither one of the above cases holds, then the indicator variable may be set to one.

$\forall req \in Reqs. \forall rg \in RG_{req} \forall \iota = [a, b] \subseteq \Gamma_{tg}$,

$$t_-(rg) - b \leq \chi_{\iota, rg}(t) \cdot (Timehorizon - b)$$

$$t_+(rg) - a \geq -\chi_{\iota, rg}(t) \cdot (Timehorizon - a)$$

**GUARANTEE_LYING_IN_INTERVAL**

**General:** There exists at least one of the specified intervals, in which the request group lies.

**Technical:** Note that we enforce this even if a request is not embedded.

$\forall req \in Reqs. \forall rg \in RG_{req}$,

$$\sum_{\iota \in \Gamma_{tg}} \chi_{\iota, rg}(t) \geq 1$$

The dependencies $dep_{req}$ between request groups of the same request are easily enforced by introducing a constraint for each such dependency (see Table 18). For replacing the interval identifier of request groups we introduce the macro

$$\text{time}((rg, t) \in IID_{req}) = \begin{cases} t_+(rg) & \text{if } t = \text{start} \\ t_-(rg) & \text{if } t = \text{end} \end{cases}$$

**Table 18:** Guarantee temporal dependencies between request groups

**GUARANTEE_REQUEST_GROUP_DEPENDENCIES**

**General:** Inter-request group dependencies are replaced syntactically by the appropriate variables.

$\forall req \in Reqs. \forall (iid_1, iid_2, c) \in dep_{req}$

$$\text{time}(iid_1) + c \leq \text{time}(iid_2)$$

3.4.6 State Changes at Events

Depending on the request group that is mapped on an event point and whether the embedding is beginning or ending, we derive the state changes induced for each event point. As we do not know which request group is mapped on which event points, we have to argue about all possible
combinations of request groups and event points. The following constraints then enforce that the state changes equal exactly the changes induced by the assigned request group (see Table 19).

| Table 19: Constraints setting the $\Delta V, \Delta E$ appropriately |
|-------------------------|-------------------------|
| **CALCULATE_DELTA_CHANGES_LINK** | **CALCULATE_DELTA_CHANGES_NODE_Q** |
| **General:** Variable $\Delta E(e_i, L_s)$ represents the change in substrate link allocations for each event. The following constraints set these changes according to the request group that has been mapped on this event. **Technical:** Note that in the second and third constraint we need $2 \cdot s_{SS}(L_s)$ as the value may be either positive or negative. |
| $\forall e_i \in \text{Events}, \forall L_s \in \text{ES}, \forall \text{rg} \in \text{RG}$ |
| $\Delta E(e_i, L_s) \leq \chi_{+}(e_i, \text{rg}) \cdot s_{SS}(L_s) + \sum_{L_v \in \text{Ereq}} \text{flow}_{\text{rg}}(L_v, L_s)$ |
| $\Delta E(e_i, L_s) \geq -2\chi_{+}(e_i, \text{rg}) \cdot s_{SS}(L_s) + \sum_{L_v \in \text{Ereq}} \text{flow}_{\text{rg}}(L_v, L_s)$ |
| $\Delta E(e_i, L_s) \leq 2\chi_{-}(e_i, \text{rg}) \cdot s_{SS}(L_s) - \sum_{L_v \in \text{Ereq}} \text{flow}_{\text{rg}}(L_v, L_s)$ |
| $\Delta E(e_i, L_s) \geq -\chi_{-}(e_i, \text{rg}) \cdot s_{SS}(L_s) - \sum_{L_v \in \text{Ereq}} \text{flow}_{\text{rg}}(L_v, L_s)$ |

$\forall e_i \in \text{Events}, \forall N_s \in \text{VS}, \forall \text{req} \in \text{Reqs}, \forall \text{rg} \in \text{QRG}_{\text{req}}$

$\Delta V(e_i, N_s) \leq \chi_{+}(e_i, \text{rg}) \cdot s_{SS}(N_s) + \sum_{N_v \in \text{Vsg}} \chi_{V}(N_v, N_s) \cdot q_{S_{V}}(N_v)$

$\Delta V(e_i, N_s) \geq -2\chi_{+}(e_i, \text{rg}) \cdot s_{SS}(N_s) + \sum_{N_v \in \text{Vsg}} \chi_{V}(N_v, N_s) \cdot q_{S_{V}}(N_v)$

$\Delta V(e_i, N_s) \leq 2\chi_{-}(e_i, \text{rg}) \cdot s_{SS}(N_s) - \sum_{N_v \in \text{Vsg}} \chi_{V}(N_v, N_s) \cdot q_{S_{V}}(N_v)$

$\Delta V(e_i, N_s) \geq -\chi_{-}(e_i, \text{rg}) \cdot s_{SS}(N_s) - \sum_{N_v \in \text{Vsg}} \chi_{V}(N_v, N_s) \cdot q_{S_{V}}(N_v)$

**CALCULATE_DELTA_CHANGES_NODE_D**

| **General:** Same as CALCULATE_DELTA_CHANGES_NODE_Q (see above) but using the local assignments of the DRGs. |
| $\forall e_i \in \text{Events}, \forall N_s \in \text{VS}, \forall \text{req} \in \text{Reqs}, \forall \text{drg} \in \text{DRG}_{\text{req}}$

$\Delta V(e_i, N_s) \leq \left( \sum_{N_v \in \text{Vsg}} \text{node}_{\text{alloc}_{\text{drg}}}(N_v, N_s) \right) + \chi_{+}(e_i, \text{rg}) \cdot s_{SS}(N_s)$

$\Delta V(e_i, N_s) \geq \left( \sum_{N_v \in \text{Vsg}} \text{node}_{\text{alloc}_{\text{drg}}}(N_v, N_s) \right) - \chi_{+}(e_i, \text{rg}) \cdot 2 \cdot s_{SS}(N_s)$

$\Delta V(e_i, N_s) \leq \left( \sum_{N_v \in \text{Vsg}} \text{node}_{\text{alloc}_{\text{drg}}}(N_v, N_s) \right) + \chi_{-}(e_i, \text{rg}) \cdot 2 \cdot s_{SS}(N_s)$

$\Delta V(e_i, N_s) \geq \left( \sum_{N_v \in \text{Vsg}} \text{node}_{\text{alloc}_{\text{drg}}}(N_v, N_s) \right) - \chi_{-}(e_i, \text{rg}) \cdot s_{SS}(N_s)$

3.4.7 Guaranteeing Feasible Allocations in the Substrate Finally, after having calculated the changes in the substrate, we can inductively construct the substrate’s states (see Table 21) based on some initial state (see Table 20) and require them to be feasible with respect to the given substrate’s resources (see Table 22).
Table 20: Initial state of substrate

**SET_INITIAL_SUBSTRATE_NODE_ALLOCATIONS**

*General*: We set the initial node allocations to zero.

\[ \forall N_s \in V_S \]

\[ \text{node}\_alloc(s_{0,1}, N_s) = 0 \]

**SET_INITIAL_SUBSTRATE_LINK_ALLOCATIONS**

*General*: We set the initial link allocations to zero.

\[ \forall L_s \in E_S \]

\[ \text{link}\_alloc(s_{0,1}, L_s) = 0 \]

Table 21: Inductively define the substrate’s state between each event

**RELATE_STATE_ALLOCATIONS_WITH_DELTAS_LINKS**

*General*: The substrate’s state can be inductively derived using the initial state and the changes at events.

\[ \forall i \in \{1, 2, \ldots, |\text{Events}|\}. \forall L_s \in E_S. \]

\[ \text{link}\_alloc(s_{i,i+1}, L_s) = \text{link}\_alloc(s_{i-1,i}, L_s) + \Delta E(e_i, L_s) \]

**RELATE_STATE_ALLOCATIONS_WITH_DELTAS_NODES**

*General*: The substrate’s state can be inductively derived using the initial state and the changes at events.

\[ \forall i \in \{1, 2, \ldots, |\text{Events}|\}. \forall N_s \in V_S. \]

\[ \text{node}\_alloc(s_{i,i+1}, N_s) = \text{node}\_alloc(s_{i-1,i}, N_s) + \Delta V(e_i, N_s) \]

Table 22: Guarantee feasibility of allocations

**GUARANTEE_FEASIBILITY_AT_EACH_STATE_LINKS**

*General*: For each state the embedding must be feasible, i.e. allocations do not exceed the substrate’s capacity.

\[ \forall s_{i,i+1} \in \text{States}. \forall L_s \in E_S. \]

\[ \text{link}\_alloc(s_{i,i+1}, L_s) \leq \text{ss}_E(L_s) \]

**GUARANTEE_FEASIBILITY_AT_EACH_STATE_NODES**

*General*: For each state the embedding must be feasible, i.e. allocations do not exceed the substrate’s capacity.

\[ \forall s_{i,i+1} \in \text{States}. \forall N_s \in V_S. \]

\[ \text{node}\_alloc(s_{i,i+1}, N_s) \leq \text{ss}_V(N_s) \]

### 3.5 Objective Functions

The formulation of the MIP so far constitutes the core of our approach, and facilitates the calculation of feasible solutions to the embedding task. A great advantage of using mathematical programming is the simple support for different objective functions. Here, we will first present some basic, general objective functions; subsequently, we will discuss the rather complex objective of minimizing energy consumption.

#### 3.5.1 Maximize Profit

One of the most basic objective functions is to aim to embed the maximal number of requests, weighted by some given parameter \( \text{profit} : \text{Reqs} \rightarrow \mathbb{R}_{\geq 0} \). This leads to the following formulation:

\[ \max \ \text{Profit} \sum_{\text{req} \in \text{Reqs}} \text{embedded(req)} \cdot \text{profit(req)} \]

As an interesting practical use-case, we envision that a provider first determines the maximal set of requests that can be embedded using the above objective, and then continues with this set of requests to optimize another objective as maximizing energy savings.
3.5.2 Minimizing Flow  Assuming that no DRGs shall be embedded and that all requests must be embedded, we can simply minimize the overall induced flow (measured then in e.g., GB).

\[
\text{min DATA TRANSMISSIONS : } \sum_{\text{req} \in \text{Reqs}} \sum_{\text{qrg} \in \text{QRG}_{\text{req}}} \left( q_{\text{dur}}(\text{qrg}) \cdot \sum_{L_s \in \mathcal{E}_S} \sum_{L_v \in \mathcal{V}_S} \text{flow}_{\text{qrg}}(L_s, L_v) \right)
\]

The above objective function cannot be formulated as easily for DRGs due to non-linearities, as the transferred amount of data is equal to:

\[
\sum_{\text{req} \in \text{Reqs}} \sum_{\text{drg} \in \text{DRG}_{\text{req}}} \left( \sum_{\text{dlp}_i \in \mathcal{DLP}_{\text{drg}}} \lambda_{\text{drg}}(\text{dlp}_i) \cdot \text{dlp}_i \right) \cdot \sum_{L_s \in \mathcal{E}_S} \sum_{L_v \in \mathcal{V}_S} \text{flow}_{\text{qrg}}(L_s, L_v)
\]

3.5.3 Maximize Earliness  As additional requests may be received by clients after an embedding was calculated, the provider has an incentive to pack the embeddings as early as possible, maximizing the ‘chance’ for allowing to embed later requests. Given the parameter \( \text{weight : } \text{RG} \rightarrow \mathbb{R}_{\geq 0} \) to measure e.g. the induced load, this constraint can be formulated as follows:

\[
\text{min LATENESS : } \sum_{\text{rg} \in \text{RG}} \text{weight}(\text{rg}) \cdot t_{\text{rg}}
\]

3.5.4 Maximizing Energy Savings  Assuming that all request shall be embedded then the substrate network provider has the incentive to reduce the cost. One possibility to do so, is to maximize energy savings by powering off inactive substrate nodes, thus saving energy. A substrate node is inactive, if no resources are allocated on it and it does not send or receive flow. Importantly, we will consider inactivity of nodes for each state and not consider deactivating nodes for the whole scenario (which could be easily implemented as well). We introduce further variables for deciding whether a node is inactive and measuring its inactivity period (see Table 23).

**Table 23:** Variables for measuring the inactivity of substrate nodes

<table>
<thead>
<tr>
<th>node_inactive</th>
<th>$\text{States} \times \mathcal{V}_S \rightarrow \mathbb{R}$</th>
<th>Decides whether a substrate node is inactive at some state.</th>
</tr>
</thead>
<tbody>
<tr>
<td>node_inactivity_duration</td>
<td>$\text{States} \times \mathcal{V}<em>S \rightarrow \mathbb{R}</em>{\geq 0}$</td>
<td>If a substrate node is inactive in some state, then this variable gives the time of inactivity during the state.</td>
</tr>
</tbody>
</table>

Using the variables introduced (see Table 23), we maximize the inactivity duration of nodes weighted by their capacity, which we assume to be related to its power consumption:

\[
\text{max ENERGY SAVINGS : } \sum_{s_i, i+1 \in \text{States}} \sum_{N_s \in \mathcal{V}_S} \text{ss}_V(N_s) \cdot \text{node_inactivity\_duration}(s_i, i+1, N_s)
\]

Using the constraints in Table 24, we enforce that inactive nodes do in fact not consume resources.

**Table 24:** Constrain inactive nodes to allocate no resources and send no flow

**General:** If a substrate node is inactive at a given state, then no resources may be allocated on it.

\[
\forall s_{i, i+1} \in \text{States}. \forall N_s \in \mathcal{V}_S, \quad \text{node\_alloc}(s_{i, i+1}, N_s) \leq \text{node\_inactive}(s_{i, i+1}, N_s) \cdot \text{ss}_V(N_s)
\]

(continues on next page)
Table 24: Constrain inactive nodes to allocate no resources and send no flow (continued)

INACTIVE_SNODE IMPLIES NO_OUTGOING_FLOW

General: If a substrate node is inactive at a given state, then it may neither send or receive any flow.

Technical: As inactive nodes may not send any flow, it may only receive flow destined for it as otherwise he would need to forward it. However, if a node receives flow, then some virtual node must be mapped on it, thereby allocating resources on it. This contradicts its inactivity. Therefore, the following constraint suffices for disabling both sending and receiving flow:

∀s_{i,i+1} \in \text{States}. \forall L_s \in E_S.
\quad \text{link}_\text{alloc}(s_{i,i+1}, L_s) \leq \text{node}_\text{inactive}(s_{i,i+1}, N_s) \cdot \text{ss}_E(L_s)

The inactivity duration of a substrate node is bounded from above by the duration of the corresponding state. We therefore introduce the macro to denote the duration of a state:

$$\text{duration}_\text{State}(s_{i,i+1} \in \text{States}) = \begin{cases} \text{Time}_\text{horizon} - t_{\text{event}}(e_2|\text{RG}) & s_{i,i+1} = s_2|\text{RG}|, s_2 \in \text{RG} + 1 \\ t_{\text{event}}(e_1) & s_{i,i+1} = s_0, 1 \\ t_{\text{event}}(e_{i+1}) - t_{\text{event}}(e_i) & \text{else} \end{cases}$$

To calculate the inactivity duration per state, it suffices to note that this duration is zero for active nodes and is bounded by the state’s duration for inactive nodes (see Table 25).

Table 25: Bound the inactivity duration of substrate nodes

BOUND_INACTIVITY_DURATION_FROM_ABOVE_BY_INACTIVITY

General: The inactivity duration of a node can only be greater than zero, if the substrate node is really inactive.

∀s_{i,i+1} \in \text{States}. \forall N_s \in V_S.
\quad \text{node}_\text{inactivity}_\text{duration}(s_{i,i+1}, N_s) \leq \text{node}_\text{inactive}(s_{i,i+1}, N_s) \cdot \text{Time}_\text{horizon}

BOUND_INACTIVITY_DURATION_FROM_ABOVE_BY_DURATION

General: The inactivity duration of a substrate node with respect to a state is bounded by the corresponding state’s duration.

∀s_{i,i+1} \in \text{States}. \forall N_s \in V_S.
\quad \text{node}_\text{inactivity}_\text{duration}(s_{i,i+1}, N_s) \leq \text{duration}_\text{State}(s_{i,i+1})

In preliminary experiments using the solver CPLEX \cite{22} could not establish a good upper bound on this objective. However, we can derive such a bound on our own as presented in Table 26. This upper bound can then be used by the MIP solver to establish an optimality gap.

Table 26: Bound the objective function by the allocations that need to be placed in the substrate

BOUND_OBJECTIVE_FROM_ABOVE

General: The following constraint gives an explicit upper bound on the objective.

$$\text{ENERGY SAVINGS} \leq \text{Time}_\text{horizon} \cdot \sum_{N_s \in V_S} \text{ss}_V(N_s) - \left( \sum_{\text{req} \in \text{Reqs}} \left( \sum_{\text{qrg} \in \text{QRG}} \text{qs}_\text{dur}(\text{qrg}) \sum_{N_v \in V_{\text{qrg}}} \text{qs}_V(N_v) + \sum_{\text{drg} \in \text{DRG}} \sum_{N_v \in V_{\text{drg}}} \text{ds}_V(N_v) \right) \right)$$

17
4 Approximation of Time-Dependent Variables

While QNets are solved optimally by our \( \Delta \)-approach by definition, time-dependent variables introduce non-linearities (see Section 3.2 for a short discussion). In the following we will show how to approximate the rate as a function of the duration for any constant absolute value by means of piecewise linear functions (see Figure 3 for an example) to yield the duration linearization points used in our MIP formulation (see Table 4).

Let \( D = [d_{\text{min}}, d_{\text{max}}] \) denote the range of the time-varying duration and let \( \text{load} \) denote the amount of resources that shall be allocated over time. We set \( r(d) : D \rightarrow \mathbb{R}_{\geq 0}, d \mapsto \frac{\text{load}}{d} \), such that for a given duration the lowest rate possible is selected.

Our aim is to find a linear approximation \( \tilde{r} : D \rightarrow \mathbb{R}_{\geq 0} \) such that the following two conditions hold:

1. \( \| \tilde{r} - r \|_{\infty, D} = \max_{d \in D} |\tilde{r}(d) - r(d)| \leq \varepsilon \)
2. \( \tilde{r}(d) \geq r(d) \) for all \( d \in D \)

The second condition here guarantees, that the customer will always obtain at least the resources requested as the provider will only overallocate resources.

4.1 Iterative Algorithm for Calculating Duration Linearization Point

We show how to construct a piecewise linear function \( \tilde{r} \) subject to \( \| \tilde{r} - r \|_{\infty} \leq \varepsilon \) and \( \tilde{r} \geq r \) by iteratively calculating duration linearization points. Let \( G(r) = \{(d, r(d))|d \in D\} \) denote the graph of \( r \). Given a point \( P = (x, y) \in G(r) \) we construct the rightmost point \( P' = (x', y') \in G(r) \) such that for the line segment \( \|PP' - r\|_{\infty,[x,x']} \leq \varepsilon \) holds.

Assume for now that both points \( P, P' \in G(r) \) are given. The line segment has the following functional form \( \overline{PP'} : [x, x'] \rightarrow \mathbb{R}_{\geq 0}, \overline{PP'}(d) = y + \frac{y' - y}{x' - x} \cdot (d - x) \). As \( P, P' \in G(r) \) we have \( y = \frac{\text{load}}{x} \) and \( y' = \frac{\text{load}}{x'} \). Using this fact, transformations yield

\[
\overline{PP'}(d) = \text{load} \cdot \left( \frac{x + x' - d}{x \cdot x'} \right).
\]

Due to the convexity of \( r \) the function \( \overline{PP'} \) lies above \( r \) on \([x, x']\). The error at any point can therefore be written as \( \text{err} : [x, x'] \rightarrow \mathbb{R}_{\geq 0} \)

\[
\text{err}(d) = \text{load} \cdot \left( \frac{x + x' - d}{x \cdot x'} \right) - \frac{\text{load}}{d}.
\]

Differentiating \( \text{err} \) and simplifying the terms gives:

\[
\text{err}'(d) = -\text{load} \cdot \left( \frac{1}{x \cdot x'} - \frac{1}{d^2} \right)
\]

The solutions of \( \text{err}'(d) = 0 \) are therefore \( d = \pm \sqrt{x \cdot x'} \). As both \( x \) and \( x' \) are positive we can restrict ourselves to \( d' = \sqrt{x \cdot x'} \). Due to the convexity of \( r \) the function \( \text{err} \) attains at \( d' \) its unique global maximum within the interval \([x, x']\). This yields:

\[
\|\overline{PP'} - r\|_{\infty,[x,x']} = \text{err}(d') = \text{load} \cdot \left( \frac{x + x' - d'}{x \cdot x'} - \frac{1}{d'} \right).
\]

By substituting \( d' = \sqrt{x \cdot x'} \) and decomposing the fraction we obtain

\[
\|\overline{PP'} - r\|_{\infty,[x,x']} = \text{load} \cdot \left( \frac{1}{x'} + \frac{1}{x} - \frac{2}{\sqrt{x \cdot x'}} \right)
\]

To determine the next sampling point \( P' \) we set \( \|\overline{PP'} - r\|_{\infty,[x,x']} = \varepsilon \). By using the substitution \( z = \frac{1}{\sqrt{x}} \) we obtain

\[
\text{load} \cdot \left( z^2 + \frac{1}{x} - \frac{2 \cdot z}{\sqrt{x}} \right) = \varepsilon
\]

18
with solutions

\[ z = \frac{1}{x} \pm \sqrt{\frac{\varepsilon}{\text{load}}}. \]

By resubstitution we can derive

\[ x' = x \cdot \frac{1}{(1 \pm \sqrt{x \cdot \frac{\text{load}}{\varepsilon}})^2}. \]

As \((1 + \sqrt{x \cdot \frac{\text{load}}{\varepsilon}})^2 > 1\) implies \(x' < x\), we can restrict ourselves to the case

\[ x' = x \cdot \frac{1}{(1 - \sqrt{x \cdot \frac{\text{load}}{\varepsilon}})^2}. \]

Note that the above assignment is only feasible if \(x \neq \frac{\text{load}}{\varepsilon}\). In fact, we can require that \(x < \frac{\text{load}}{\varepsilon}\) holds, since for all \(d \geq \frac{\text{load}}{\varepsilon}\) the value of \(r(d)\) is less than \(\varepsilon\). Thus, any point will realize a maximal error less than \(\varepsilon\). In our algorithm, we choose in this case \(x' = d_{\text{max}}\). Concluding, we present our algorithm \textsc{LINAP} (see Algorithm 1) for approximating \(r\) within any absolute error \(\varepsilon\).

\begin{algorithm}

\textbf{Algorithm 1 LINAP: Linear approximation of} \(r: D = [d_{\text{min}}, d_{\text{max}}] \rightarrow \mathbb{R}_{\geq 0}\), \(r(d) = \frac{\text{load}}{d}\) \textit{within a constant maximal error of} \(\varepsilon\)

\begin{itemize}
  \item \textbf{Require:} \(d_{\text{min}} > 0 \land d_{\text{max}} > d_{\text{min}} \land \varepsilon > 0 \land \text{load} > 0\)
  \item \textbf{Ensure:} \(||\hat{r} - r||_{\infty, D} \leq \varepsilon \land \hat{r} \geq r\)
  \item \(x_0 = d_{\text{min}}\)
  \item \(i \leftarrow 0\)
  \item \(x_i < \frac{\text{load}}{\varepsilon} \land x_i < d_{\text{max}}\) \textbf{do}
    \item \(x_{i+1} \leftarrow x_i \cdot \frac{1}{(1 - \sqrt{x_i \cdot \varepsilon})^2}\)
    \item \(i \leftarrow i + 1\)
  \item \textbf{end while}
  \item \(x_i \leftarrow d_{\text{max}}\)
  \item \(\hat{r} = \left\{ x_i + \frac{r(x_{i+1}) - r(x_i)}{x_{i+1} - x_i} \cdot (d - x_i) \right\} \land x_i \leq d \leq x_{i+1}\)
  \item \textbf{return} \(\hat{r}\)
\end{itemize}

\end{algorithm}

4.2 Analysis of Algorithm \textsc{LINAP}

We show now that the approximation given by Algorithm 1 is efficient.

\textbf{Theorem 1.} Every function \(r: D = [d_{\text{min}}, d_{\text{max}}] \rightarrow \mathbb{R}_{\geq 0}\), \(r(d) = \frac{\text{load}}{d}\) can be approximated within the absolute error \(\varepsilon > 0\) using \(O(1/\sqrt{\varepsilon})\) many line segments.

\textbf{Proof.} We use Algorithm 1 to calculate \(\hat{r}\). We consider the scaling of two adjacent sampling points \(\frac{x_{i+1}}{x_i} = 1/(1 - \sqrt{x_i \cdot \frac{\text{load}}{\varepsilon}})^2\). As \(x_i < \frac{\text{load}}{\varepsilon}\) holds for all but the last sampling point, \(\frac{x_{i+1}}{x_i}\) is a strictly monotonically increasing sequence with \(\frac{x_{i+1}}{x_i} \geq \frac{x_1}{x_0} = 1/(1 - \sqrt{d_{\text{min}} \cdot \frac{\text{load}}{\varepsilon}})^2\). By the observation \(\frac{x_i}{x_0} = \frac{x_1}{x_{i-1}} \cdot \frac{x_2}{x_{i-2}} \cdots \cdot \frac{x_i}{x_0} \geq 1/(1 - \sqrt{d_{\text{min}} \cdot \frac{\text{load}}{\varepsilon}})^{2i}\) and since \(x_0\) equals \(d_{\text{min}}\) we obtain

\[ x_i \geq d_{\text{min}}/\left(1 - \sqrt{d_{\text{min}} \cdot \frac{\varepsilon}{\text{load}}}\right)^{2i}. \]

As the algorithm must terminate when the sampling point exceeds \(d_{\text{max}}\), we can bound the number of needed intervals as follows (where \(\log\) denotes the binary logarithm):

\[ d_{\text{max}} \leq d_{\text{min}}/\left(1 - \sqrt{d_{\text{min}} \cdot \frac{\varepsilon}{\text{load}}}\right)^{2i} \iff i \geq \frac{\log d_{\text{min}}}{2 \cdot \log(1 - \sqrt{\varepsilon \cdot d_{\text{min}} / \text{load}})}. \]
By setting $c = \frac{d_{\min}}{d_{\text{load}}}$ and $d = \log \frac{d_{\min}}{d_{\max}}$, the number of needed sampling points can be bound by $i(\varepsilon) = \lceil \frac{d}{2\log(1 - \sqrt{c\varepsilon})} \rceil$.

Using only a single line segment between $d_{\min}$ and $d_{\max}$ the approximation error computes to $\frac{\varepsilon_{\max}}{\sqrt{d_{\min} d_{\max}}} = \varepsilon_{\max}$. Thus, we restrict the domain of $\varepsilon$ to $(0, \varepsilon_{\max})$ as otherwise a one line segment approximation will always suffice.

We proceed by showing that $\tilde{i}(\varepsilon) = -\frac{d}{2\sqrt{c\varepsilon}}$ is an upper bound on $i(\varepsilon)$.

$$
\tilde{i}(\varepsilon) \geq i(\varepsilon) \iff -\frac{d}{2\sqrt{c\varepsilon}} \geq \frac{d}{2\log(1 - \sqrt{c\varepsilon})} \\
\iff -\sqrt{c\varepsilon} \geq \log(1 - \sqrt{c\varepsilon}) \\
\iff 0 \geq \log(1 - \sqrt{c\varepsilon}) + \sqrt{c\varepsilon}
$$

Let $g(\varepsilon) = \log(1 - \sqrt{c\varepsilon}) + \sqrt{c\varepsilon}$ denote the right-hand side of the above formula. We show that $g(\varepsilon) \leq 0$ holds for all $\varepsilon \in (0, \varepsilon_{\max})$ which yields the bound. Consider at first $g'(\varepsilon) = \frac{\sqrt{c}}{\sqrt{c\varepsilon} - \sqrt{\varepsilon}} + \frac{\sqrt{\varepsilon}}{\sqrt{\varepsilon}} = \frac{1}{\sqrt{\varepsilon}} \cdot \left( \frac{\sqrt{c}}{\sqrt{c\varepsilon} - \sqrt{\varepsilon}} + \sqrt{\varepsilon} \right)$. We show that $g$ is strictly monotonically decreasing by showing that $g' < 0$ holds for all $\varepsilon \in (0, \varepsilon_{\max})$.

$$
g'(\varepsilon) < 0 \iff \frac{\sqrt{c}}{\sqrt{c\varepsilon} - \sqrt{\varepsilon}} + \sqrt{\varepsilon} < 0 \\
\iff 1 < \frac{1}{1 - \sqrt{c\varepsilon}} \\
\iff 1 - \sqrt{c\varepsilon} < 1 \\
\iff 0 < \varepsilon
$$

Here, the last equivalence holds, since $c = \frac{d_{\min}}{d_{\text{load}}} > 0$ holds. Therefore, for all $\varepsilon \in (0, \varepsilon_{\max})$ the function $g$ is strictly monotonically decreasing. Finally, we have to check whether $\lim_{\varepsilon \to 0} g(\varepsilon) \leq 0$ such that indeed $g(\varepsilon) \leq 0$ holds for all $\varepsilon \in (0, \varepsilon_{\max})$.

$$
\lim_{\varepsilon \to 0} g(\varepsilon) \leq 0 \iff \lim_{\varepsilon \to 0} \log(1 - \sqrt{c\varepsilon}) + \sqrt{c\varepsilon} \leq 0 \\
\iff \lim_{\varepsilon \to 0} \log(1 - \sqrt{c\varepsilon}) \leq \lim_{\varepsilon \to 0} -\sqrt{c\varepsilon} \\
\iff \lim_{\varepsilon \to 0} \left( \frac{\log(1 - \sqrt{c\varepsilon})}{-\sqrt{c\varepsilon}} \right) \geq 1 \\
\iff \lim_{\varepsilon \to 0} \left( \frac{\frac{d}{d\varepsilon} \left( \log(1 - \sqrt{c\varepsilon}) \right)}{\frac{d}{d\varepsilon} \left( -\sqrt{c\varepsilon} \right)} \right) \geq 1 \\
\iff \lim_{\varepsilon \to 0} \left( \frac{\sqrt{c\varepsilon} - \sqrt{\varepsilon}}{\sqrt{c\varepsilon} - \sqrt{\varepsilon}} \right) \geq 1 \\
\iff \lim_{\varepsilon \to 0} \left( \frac{1}{1 - \sqrt{c\varepsilon}} \right) \geq 1 \\
\iff 1 \geq 1
$$

The equivalence in the line marked by $\ast$ holds by L'Hôpital's rule as the limit $\lim_{\varepsilon \to 0} \left( \frac{\frac{d}{d\varepsilon} \left( \log(1 - \sqrt{c\varepsilon}) \right)}{\frac{d}{d\varepsilon} \left( -\sqrt{c\varepsilon} \right)} \right)$ is 1 and therefore exists. Thus, we have shown that $\lim_{\varepsilon \to 0} \leq 0$ holds. Combined with the fact that $g'$ is strictly monotonically
decreasing we conclude that \( g(\varepsilon) \leq 0 \) holds for all \( \varepsilon \in (0, \varepsilon_{\text{max}}) \). This completes the analysis of the number of intervals needed for a constant \( \varepsilon > 0 \). As \( i(\varepsilon) = O(1/\sqrt{\varepsilon}) \) holds, we have proven our claim. \( \square \)

Note that the Algorithm 1 approximates the rate with a constant absolute error \( \varepsilon \). The relative amount of overallocations is then less or equal to \( \varepsilon \cdot d_{\text{max}} \) for one DRG. Thus, we obtain the following corollary.

**Corollary 1.** The relative overallocation error \( \varepsilon_{\text{overallocation}} \) is bounded by \( \varepsilon \cdot \text{Timehorizon} \) for some fixed \( \varepsilon \). As \( \text{Timehorizon} \) is a constant, using the approximation algorithm LIPA we have a relative approximation error of the overallocation of \( \varepsilon_{\text{overallocation}} \) using \( O(1/\sqrt{\varepsilon_{\text{overallocation}}} \) many line segments.

5 Analysis of the \( \Delta \)-Approach

While our approach allows to compute optimal embeddings for QNets and very good approximations for DNets, computing these solutions may be costly (see Section 6). In this section we analyze shortcomings of our MIP core, to then discuss how execution runtimes can be greatly improved by adding further so-called user cuts that tighten the LP-relaxations.

5.1 Tightness of the LP-Relaxation

For the branch and bound procedure of MIP solvers to be effective, it is crucial that the LP-relaxations give good bounds on the objective value of the MIP itself. As we do not make use of so-called big-M constants (“infinity constraints”) in our MIP, the constraints themselves are tight. Nonetheless, as we will analyze below, constraints of the following form are problematic: \( \var_1 \leq \var_2 + \chi \cdot u(\var_1) \land \var_1 \geq \var_2 - \chi \cdot u(\var_1) \).

Here, \( \chi \) shall denote a binary variable and we assume \( \var_1 \in [0, u(\var_1)] \). Clearly, this constraint performs the assignment \( \var_1 \leftarrow \var_2 \) if \( \chi = 1 \). If \( \chi = 0 \) holds, then this constraint does not bound \( \var_1 \) in any way. Consider now the case that multiple request groups have not been mapped on events yet. In the LP-relaxation the corresponding unfixed mapping variables \( \chi_+ \) and \( \chi_- \) may then be “smeared”, i.e. the mapping variables do not attain binary values.

Thus the request group starts or end are fractionally assigned to multiple events. This smearing effect can be observed in fact quite often as the local allocations of the request groups can thereby be distributed among multiple states, allowing for feasible embeddings in the relaxation, even though no integral solution might exist.

5.2 Dependency Graph User Cuts

To yield reasonable LP-relaxations, smearing should be reduced as best as possible. As checking feasibility crucially depends on the assignment of request groups to events, we will now enrich our model with so-called user cuts, i.e. constraints deducted from the input, such that the set of feasible solutions is not reduced but the LP-relaxations are tightened. To this end, we introduce the directed dependency graph \( G_{\text{dep}}(\text{Reqs}) = (V_{\text{dep}}, E_{\text{dep}}) \) that will reflect implicit temporal dependencies.

We define the Graph as follows. The set of nodes represents the abstract start and end point for each request group: \( V_{\text{dep}} = \text{RG} \times \{\text{start, end}\} \). Let \( \bigcup_{\iota \in \text{Rg}} I_{\iota} \) denote the union of all intervals in which the request group \( \text{rg} \) may be realized. We define the following functions to calculate the earliest possible start and the latest possible end time.

\[
\text{earliest}(rg, t) \in V_{\text{dep}} = \begin{cases} 
\min \bigcup_{\iota \in \text{Rg}} I_{\iota}, & \text{if } t = \text{start} \\
\min \bigcup_{\iota \in \text{Rg}} I_{\iota} + \text{qs}_{\text{dur}}(\text{rg}) & \text{if } \text{rg} \in \text{QRG} \land t = \text{end} \\
\min \bigcup_{\iota \in \text{Rg}} I_{\iota} + \text{dlp}_1 & \text{if } \text{rg} \in \text{DRG} \land t = \text{end}
\end{cases}
\]

\[
\text{latest}(rg, t) \in V_{\text{dep}} = \begin{cases} 
\max \bigcup_{\iota \in \text{Rg}} I_{\iota} - \text{qs}_{\text{dur}}(\text{rg}) & \text{if } \text{rg} \in \text{QRG} \land t = \text{start} \\
\max \bigcup_{\iota \in \text{Rg}} I_{\iota} - \text{dlp}_1 & \text{if } \text{rg} \in \text{DRG} \land t = \text{start} \\
\max \bigcup_{\iota \in \text{Rg}} I_{\iota} & \text{if } t = \text{end}
\end{cases}
\]

A directed edge \((v, w) \in V_{\text{dep}}^2\) will be contained in \( E_{\text{dep}} \) iff. \( v \) must start before \( w \).
MAPPING

RESTRICTION

GLOBAL

GROUP

REQUEST

DEPENDENCY

and restricted scenario (focusing on temporal aspects), we believe the raised questions to be of general interest. Our costs? Although a rigorous analysis is beyond the scope of this paper and we will only investigate an anecdotal question arising in network virtualization in general: How does the specification of a VNet request affect allocation algorithms? The above observations therefore yield the following constraints presented in Table 27.

The above observations therefore yield the following constraints presented in Table 27.

### Table 27: Dependency Graph User Cuts

<table>
<thead>
<tr>
<th>Event Mapping Restriction</th>
<th>General: We restrict the set of events that a request group start or end can be mapped on by using the reachability information of the dependency graph.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Technical: (</td>
</tr>
</tbody>
</table>

\[ |\text{RG}| - |\text{dist}^\text{min}(v)| \sum_{i=|\text{dist}^\text{max}(v)|+1}^{2|\text{RG}|} \chi_{\text{Event}}(e_i, v) = 1 \]

<table>
<thead>
<tr>
<th>Enforce Global Request Group Dependency</th>
<th>General: This constraint implements each - possibly transitive - relation of the dependency graph as a constraint. Let (v, w \in V_{\text{dep}}) such that (w ) is reachable from (v). Using the maximal distance (\text{dist}^\text{max}(v, w)) we know that if (w) is mapped on an event (e_j) then (v) must have been mapped on an event in the set ({e_1, e_2, \ldots, e_{j-\text{dist}^\text{max}(v, w)}}).</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( \forall v \in V_{\text{dep}}, \forall w \in \text{dist}^\text{max}(v), \forall e_i \in \text{Events}, \text{dist}^\text{max}(v, w) + 1 \leq i \leq 2 \cdot</td>
</tr>
</tbody>
</table>

### 6 On the Benefit of Flexibility

One of the main advantages of our \(\Delta\)-approach is its flexibility. In particular, it allows VNets specifications to be vague on where and when VNets are realized, or even to omit certain specifications (like time aspects) entirely. Our algorithm is able to exploit these flexibilities to optimize the embeddings (according to the given objective function).

Concretely, to demonstrate the benefit of this flexibility support, this section initiates the study of an interesting question arising in network virtualization in general: How does the specification of a VNet request affect allocation costs? Although a rigorous analysis is beyond the scope of this paper and we will only investigate an anecdotal and restricted scenario (focusing on temporal aspects), we believe the raised questions to be of general interest. Our algorithms give us a means to investigate these questions.

\[ E_{\text{dep}} = \{(v, w) \in V_{\text{dep}}^2 \mid \text{late}\text{(v)} < \text{early}\text{(w)}\}. \]
We attend to the following simulation setup. (The full simulation code and result set can be downloaded at [17].) We use a substrate network describing a $5 \times 3$-sized grid topology with unidirectional links in both directions between pairs of nodes (modeling full-duplex links). Substrate nodes have a capacity (e.g., memory or CPU) chosen uniformly at random from the interval $[6, 10]$. The capacity of links (i.e., bandwidth) is chosen uniformly from $[7, 15]$ (a link pair connecting the same node pair has the same capacity). In our scenario, eight VNet requests (concretely: QNets) arrive according to a Poisson process with exponentially distributed inter-arrival times (parameter $\lambda = 2$). The time durations of the VNets (known in advance in our setting) follow a Weibull distribution (with shape parameter 2 and scale 5). The VNet topologies are: undirected cycles (four times), “aggregation” stars with links directed to the center (two times), and “multicast” stars with links directed away from the center (two times). The capacity requirements of virtual nodes are chosen from interval $[1, 5]$ and of virtual links from interval $[2, 6]$.

To study the flexibility, we assume the following model. Without flexibility, a VNet arriving at time $t$ with duration $d$ must be immediately embedded (or, if not possible due to resource constraints, rejected); the request is hence satisfied by time $t + d$. With flexibilities, we use a parameter $s$ to denote a tolerable slack when the execution of the VNet starts. Concretely, a VNet arriving at $t$ may start at any time $t' \in [t - s, t + s]$ and hence terminate at time $t' + d$. (Recall that we focus on QNets only here.) The simulated execution period is one day (i.e., 24 hours).

Figure 4 (left) shows our simulation results. As objective function we maximize the overall system utilization, i.e., the duration $\times$ VNet size (in terms of virtual nodes) product of the accepted VNets. We see that the $\Delta$-approach succeeds to exploit flexibilities to improve the embedding according to the objective function. We limit the computation to one hour and use the best embedding so far. The corresponding optimality percentage plots for $s \in \{0, \ldots, 6\}$ in Figure 4 (right) indicate that this comes without significant quality loss. We also find that execution times (and hence potentially the optimality gaps) are longer under higher flexibilities, which is expected (see 5).

Fig. 4: Impact of temporal specification flexibilities for two independent random request sets. Boxplot is over 25 repetitions where in each repetition the nodes are placed beforehand uniformly on the substrate with the restriction that two virtual nodes may not be placed on the same substrate node. For the experiments, we used CPLEX Version 12.1.0 with parameters emphasis mip 1, threads 8, timelimit 3600, workmem 2000, mip limits cutpasses 1, and mip limits probetime 5. Embeddings were computed on an Intel Xeon CPU (L5420@2.50GHz) with 16 GB RAM and 8 CPU cores.

7 Related Work

For a good introduction and overview of the network virtualization paradigm, the reader is referred to the surveys [5] and [10]. The virtual network embedding problem has been studied intensively over the last years. The general formulation is NP-hard [2], and existing literature falls into four main categories: some works focus on optimal solutions on smaller scale environments (such as a router site), e.g., [18], others propose approximation algorithms (e.g., [4]) with provable quality guarantees or heuristics (e.g., [8][15][25]) which perform well, e.g., in simulations, and some works even propose to study environments which mitigate the computational complexities by imposing certain structures (e.g., [23]). Researchers have also pursued mathematical programming approaches already. For instance, Kumar et al. [12] describe an approach to solve a simpler Virtual Private Network tree computation problem for bandwidth
Fig. 5: Depicted are all 5 request sets of the scenario described in Section 6. Notably, the averaged objective value first increases under higher temporal flexibility (left). At a temporal flexibility of 6 hours the objective value decreases for the two top request sets, as the computation time approaches the limit of one hour (right). Apparently, the runtime is correlated to the overall load (the objective function) of the request sets.

provisioning. Even et al. [7] propose a general access control algorithm for many different routing and traffic models which selects and embeds only the requests of high benefit such that the overall benefit is maximized, but without exploiting placement or time flexibilities. Chowdhury et al. [4] present an integer embedding program supporting flexible node placements for VNet, and propose a relaxation strategy to find approximate solutions more quickly.

Surprisingly, however, while a large body of literature exists on static embeddings, the important time-aspects for short-term embeddings have received much less attention. In this respect, the closest literature to ours are arguably the works on the temporal routing and max flow problems which focus on links only (e.g., [11]). A limited form of temporal embedding support is also contained in the works by Zhang et al. [24] who also introduce a heuristic topology-aware embedding scheme, or in the context of VPN embeddings [7]; however, the opportunities of time flexibilities are not explored.

There are two main models to represent time in the literature, (continuous) points [21] and (discrete) intervals [1]. Temporal information can be represented as a network of constraints on these time variables, and there exists much work on the computational tractability of fundamental properties such as consistency (see e.g., [3]). Often, formal languages and algebras are used to specify such temporal relationships (e.g., [16]), and there are interesting extensions to settings where time periods cannot be precisely described but are rather vague in nature (e.g. [19]). Also the differences between continuous and discrete time approaches have been explored and are well-understood today (e.g., see [9] for a review).

Also in other fields with time-critical applications there exist mathematical programming solutions, such as the spatio-temporal composition of distributed multimedia objects [13], or more remotely, e.g., chemical production planning [14]. Indeed, several concepts introduced in the chemical production literature also apply to the VNet embedding problem. However, the focus in chemical production planning is more on the sequential planning of production processes, where tasks can only be performed after the completion of predecessor tasks whose output is needed as input; time and specification flexibilities play only a secondary role. We are not aware of any results on more complex objective functions with time-dependent variables (e.g., to describe the flexible time-rate product in DNet), whose state (e.g., resource allocations) and state differences needs to be tracked over time, as it is achieved with our ∆-approach.

8 Conclusion

This paper has proposed a rigorous mathematical programming approach for the temporal embedding problem of flexibly specified VNets. We understand the presented ∆-approach as a framework in the sense that many additional features can be added easily. There are two main directions for future research. First, we only provided preliminary insights into the flexibility benefits and tradeoffs computable with our approach, and a more in-depth simulation study needs to be conducted. Second, while we believe that finding solutions with an accuracy and optimality similar to the ∆-approach cannot be computed significantly faster, the corresponding formal time complexity tradeoffs as well as practical means to tune standard solvers to our specific models need to be explored.
References