On Fair Traffic allocation and Efficient Utilization of Network Resources based on MARL

Preprint · May 2023

CITATIONS
0

READS
105

3 authors:

Evgeniy Stepanov
Lomonosov Moscow State University
7 PUBLICATIONS **11** CITATIONS
SEE PROFILE

Smelyasnskiy Ruslan
Lomonosov Moscow State University
3 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Ruslan. L. Smeliansky
Lomonosov Moscow State University
122 PUBLICATIONS **847** CITATIONS
SEE PROFILE

On Fair Traffic allocation and Efficient Utilization of Network Resources based on MARL

Stepanov E.P.[a,*], Smeliansky R.L.[a], Plakunov A.V.[b], Borisov A.V.[c], Xia Zhu[d], Jianing Pei[d], Zhen Yao[d]

[a] *Lomonosov Moscow State University Russia*
[b] *Applied Research Center for Computer Networks Russia*
[c] *Federal Research Center Computer Science and Control of the Russian Academy of Sciences Russia*
[d] *Huawei Technologies Co. Ltd. China*

Abstract

The problem of traffic balancing in a data network environment is considered in relation to Network Powered by Computing (NPC) - a new generation of computational infrastructure, where computational resources and data transmission resources form a network - "Network is a Computer". The key contribution of this work is Multi-Agent Routing using Hashing (MAROH) method that provides fast and fair bandwidth distribution inside an overlay network. The fast operation is guaranteed by multi-agent reinforcement learning methods, while the fair distribution is achieved by consistent hashing scheme. The tandem of these approaches ensures efficient use of resources in the overlay network. This paper presents MAROH detailed description and experimental results that demonstrate advantage over ECMP, convergence rate and robustness of MAROH method.

Keywords: MARL, traffic engineering, NPC

1. Introduction

Recently, the landscape of computational infrastructure is in dramatic changes under the pressure of application requirements [1, 3]. The suit of the properties of modern applications can be summarized as follows: distributed, self-sufficient, work in real time, elastic, cross-platform, actively interact and synchronize, and are easy to update. The definitions of these terms are in [1]. For further understanding, it is important to recognize that an application is made up of interrelated components, which we will refer to as application functions. The analysis of requirements of modern application to the computational infrastructure presented in [2, 3, 4] shows the trend of ubiquitous application deployment. We are moving to the era when data processing resources and data transmission resources form a single space for computing - computational infrastructure. In other words, the time has come for the implementation of the slogan "Network is a Computer". Further we will call such computational infrastructure Network Powered by Computing (NPC). It is should be noted that similar concept was proposed under the name Computing Power Network [3].

Several versions of Functional Architecture for such new generation of computational infrastructure were proposed [3, 4]. Briefly NPC functional architecture can be described as following. It consists of data processing (DP) plane, data transmission (DT) plane, data processing control (DPC) plane, data transmission control (DTC) plane, administration, orchestration and management plane (AOM plane). DP plane covers all computational resources of NPC. DT plane is an overlay network over underlying physical network. Actually data transmission plane is data transmission network (DTN). DPC plane is responsible for preparation of the application for execution, planning the placement of application components, calculation of the quality of service (QoS) requirements based on the service level agreement (SLA) specified by the user, generation of DTN control plane instructions for setting up overlay tunnels in accordance with the application function interaction topology of the application. DTC plane is responsible for control and monitoring of DTN. AOM plane orchestrates interactions between application components in accordance with application topology, collects NPC resource consumption statistics by every application component, secures management and administration of NPC.

In this architecture there is an important point, which is weakly explored so far – the facility that is responsible for the integration of every data processing resource (computational node - CN) with data transmission network (DTN) and external sources of computational service requests. Call this facility NPC router (NPCR). NPCR replaces several devices at once - a task manager, a traffic and task router, VPN-gateway, CPE, and supplies the following functionality:

- distribution of application functions (ApF)/ virtual network functions (VNF) across computational nodes (CN) of DP plane;

*Corresponding author

- decision making: is it worth to execute the certain ApF/VNF on the CN connected to this current NPCR or not;
- forwarding ApF/VNF that was not accepted by the current facility under some reason to other facilities where their computational resources are much more promising from the point of Application execution efficiency as a whole;
- optimal data traffic routing as between ApF as between corresponding VNF;
- provision of the transport connection that meets the required Service Level Agreement (SLA).

In those cases when the NPCR provides the input for external sources of data, applications, computational service requests to the NPC resources, we will call such NPCR as pole.

This paper has been dedicated only to one problem from the list above – the optimal traffic routing in overlay DTN of NPC. It should be noted that the solution proposed here for optimal traffic engineering is also applicable in traditional data network.

Recent surveys [5, 6] on load balancing methods showed focus on Machine Learning (ML) methods, especially multi-agent reinforcement learning (MARL). The scale and rate of data traffic in our network do not let us get the proper solution for speed and accuracy of traffic balancing by traditional centralized methods. That is why the research keeps focus on ML. There are three approaches to MARL optimization: centralized, decentralized with communication and fully decentralized. It is assumed in all these approaches that agents construct their local state based on the environment observations. However the agents behavior in these approaches are different:

- centralized approach: a single entity named control center is responsible for the operation of every agent. This center collects the local states from all agents and defines the actions for every agentby solving an optimization problem.
- decentralized approach with communication (or just decentralized approach): assumes that the local state of every agent can be broadcasted to all neighboring agents. Then the agent by himself chooses an optimal action based on the knowledge of local and neighboring agent states.
- the fully decentralized approach: do not suppose any communication between agents. Decision should be made based only on the history of the agent local behavior observations.

The main problems of MA methods for traffic control are poor scalability; there are no mathematical models that guarantee convergence to the optimal solution; it is difficult to mathematically frame the optimization functional; the extent of deviation from the optimal solution is unknown.

In this paper we focus on the decentralized approach as NPCRs are able to communicate with each other and there is no single control center in NPC. Based on examining of the works related to ML methods for load balancing a new method is proposed called MultiAgent Routing with Hashing (MAROH) to surmount the problems listed above. This method combines multi-agent reinforcement learning and hashing algorithms to achieve fast and fair bandwidth distribution.

The paper is organized as follows. In the section 2 we describe the traffic engineering optimization problem related to NPC. The works that deal with ML methods for load balancing are discussed in the section 3. The proposed new method MAROH is presented in the section 4. The results of the experiments are presented in the section 5.

2. Traffic Engineering Problem Statement

Let us formulate the load balancing problems in term of NPC. As it was mentioned above new proposed MAROH method is applicable to both NPC networks and traditional data networks. It suffices to imagine instead of NPCR and an overlay channel, a network device such as router/switch and an ordinary line, respectively. The notations are used in the paper are shown in table 1.

Consider DT plane $G = (N, E)$, represented by the directed graph G, where N is a set of NPCR facilities and E is a set of overlay channels, connecting NPCRs. According to functional NPC architecture [1], NPCR monitors the state of adjacent overlay channels: it collects bandwidth capacity $c_{u,v}$ and occupied bandwidth $b_{u,v}$ for any pair of neighboring NPCRs (u, v).

The set of flows generated by ApF/VNF interactions is represented by the traffic matrix TM. Although we use traffic matrix to describe load balancing problem, individual NPCRs only know about occupied bandwidth of their adjacent overlay channels. Each flow is represented by triplet $< s_f, d_f, r_f >$, where s_f is a source NPCR, d_f is a destination NPCR and r_f is a flow rate. We consider the discrete time model: at each time t_i there are changes in traffic matrix TM or in DT plane G.

Our goal is to develop a method that will provide a balanced load of the channels on every network node, e.g. NPCR. We define the balanced load as flow distribution with optimization criterion: minimization of the function Φ:

$$\Phi = \frac{1}{|E|}\Sigma_{(u,v)\in E}(\frac{b_{u,v}}{c_{u,v}} - \mu')^2, \quad (1)$$

where $|E|$ is the number of overlay channels, μ' is a mean channel load equal to $\frac{1}{|E|}\Sigma_{(u,v)\in E}\frac{b_{u,v}}{c_{u,v}}$. In terms of MARL optimization, Φ is a goal function.

$G = (N, E)$	directed graph, describing NPC
N	NPCR entities
E	overlay channels between NPCR
$c_{u,v}$	channel bandwidth capacity
$b_{u,v}$	occupied channel bandwidth
$TM = \{s_f, d_f, r_f\}$	traffic matrix
s_f	source NPCR of flow f
d_f	destination NPCR of flow f
r_f	rate of flow f
Φ	goal function
μ'	mean channel load
$r_{u,v}$	channel weight

Table 1: Notations

Balanced channel load can be achieved in different ways: packet balancing, flow balancing and flowlet balancing. Here we will use the following way: on every t_i every NPCR egress port/channel will be assigned a certain weight $r_{u,v}$. Special hash-function will use these weights to choose an outgoing channel for every flow. So the key problems are how to calculate and how to assign weights $r_{u,v}$ to an egress port and how to construct the proper hash-function.

Note, that elephant-flows and mice-flows cannot be distinguished by hash-function. In the paper we do not deal with this problem. In case such distinction is necessary, one should add an external mechanism of flow classification into elephant and mice flows, and apply the proposed method MAROH to each of the two sets separately. In this way elephant flows will compete with elephant flows for resources and mice flows will compete with mice flows.

Let us summarize the problem statement. Each NPCR $u \in N$ needs to choose channel weights $r_{u,v}$ based on information about channel occupied bandwidth $b_{u,v}$, collected from neighbors. $r_{u,v}$ should be selected in such way that function Φ from (1) reaches its minimum. We assume that NPCRs periodically broadcast their local state to all other NPCRs to evaluate Φ, like IGP routers maintain Link State Database (LSDB).

3. Related Works

In this section, we will look at how we could choose $r_{u,v}$ based on already known methods. Recently reinforcement learning (RL) algorithms for traffic balancing in networks are based on multi-agent reinforcement learning (MARL) with decentralized network information to easily scale the solution. This survey covers only the works that mostly fit our problem statement. The key features of every observed RL methods are collected in table 2. The distinguishing points are: single or multiple agents, centralized or decentralized network state information, state, action and reward spaces of an agent.

The main idea of AuTO method [7] is based on animals Peripheral & Central Nervous Systems analogy. Deep Reinforcement Learning system design has two parts to solve the scalability problem that many RL systems struggle with. Peripheral Systems (PS) are located at end-host devices of the network, collecting flow information to optimize the flow balancing. The short flow traffic is optimized locally to minimize delay for network information distribution. Central System (CS) aggregates the information from all the PS's and processes the global traffic information for long flows optimization. This makes AuTO an end-to-end automatic traffic optimization system that collects network information, learns from past decisions, and performs actions to achieve operator-defined goals. The usage of AuTO method implies the ability to recognize short-term flows on PS. This is not an easy problem to solve.

Multi-agent Actor-Critic Reinforcement Learning Based In-network Load Balance [8] proposes a multi-agent actor-critic reinforcement learning algorithm that uses "centralized learning – distributed execution" (CL-DE) model. This means that the forwarding elements like switches get advice from a centralized network "critic" that helps them get coordination in their acting by updating each agent's policy, while the agents take actions relying on local observations. The authors also introduce a Target Network function that helps keep the Q-function (the action-value function) more stable by learning on a fixed number of timesteps, and Experience Replay that randomly trains the policy on mini-batches of "experiences" for the algorithm to converge faster.

RILNET: A Reinforcement Learning Based Load Balancing Approach for Datacenter Networks [9]. DCN suffers from various problems, for example, highly dynamic workloads, congestions and topology asymmetry. ECMP, as a traditional load balancing mechanism that is widely used recently in DC, demonstrates a poor balance and leads to overloads. Many load balancing schemes have been proposed to surmount ECMP problems. However, these traditional schemes usually balance a load based only on knowledge about the network from a snapshot or a network state taken within a short period of time. The cited paper proposes an approach based on RL technique, called RILNET (Reinforcement Learning NETworking), aimed at load balancing for DCNs. RILNET adapts its operation for the current load by RL approach. To achieve a higher granularity control, instead of per flow routing, RILNET routes aggregated flows, which is a set of flows that includes all flows going from one source boundary switch to one destination boundary switch. Performance of RILNET implementation was tested on flow and packet level simulation. In both cases the results showed that RILNET can balance the traffic load much more efficiently than ECMP and DRILL (another load balancing technique which employs per-packet decisions at each switch based on local queue occupancies and randomized algorithms to distribute load). RILNET is superior to DRILL in terms of data loss and maximum connection latency. The main RILNET disadvantages for our purpose are: Single-agent RL method with Centralized way of obtaining data.

Paper	Single/ multiple agents	Centralized/ Decentralized	State space	Action Space	Reward
[7]	+−	centralized	Two kinds of states for short and long flows: the set of all finished flows and the set of all active flows with finished flows respectively	Two kinds of actions for short and long flows: the set of MLFQ threshold values and the tuple (flow priority, rate limit, flow path)	ratio between goal functions of two consecutive time steps
[8]	+	centralized	- 10-episode history of link utilization - current utilization - switch buffer state	fraction of flows sent on a path	negative of the maximum link utilization
[9]	−	centralized	snapshot of all throughput between all pairs of edge switches	routing proportions for all pairs of edge nodes	variance vector of rewards for all paths between each pair of edges
[10]	−	centralized	traffic demand matrix	the set of link weights of all nodes	average delay of packets within a time slot
[11]	−	centralized	traffic demand matrix	link-weight setting that implicitly determines the shortest paths between any source-destination pair	negative variance of network links utilization
[13]	+	centralized /decentralized	vector of the agent in region consisting of current link utilization in region	T-agent: set of fractions of traffic amount delivered from ingress nodes to egress nodes on a certain path O-agent: set of splitting ratios of the outgoing demands from the ingress nodes to the neighboring regions on a certain path	T-agent: negative of the maximum edge cost of T-agent's region O-Agent: negative of the maximum edge cost of T-agent's region
[12]	+	decentralized	link hidden representation consisting of current weight and link utilization for each agent	modification of the weight of agents' associated links	difference of the global maximum link utilization between consecutive steps
[14]	+	decentralized	- destination of the current packet - extra information related to agent - information shared from the neighbor nodes of agent	choice of neighbor node to deliver packet	sum of queueing time and transmission time
[15]	+	decentralized	- current node - neighboring nodes of the current node - destination node of the first packet in the current node's queue	sending packet to a neighboring node according to a greedy algorithm	negative sum of the estimated queue delay for the packet in the next agent and the transmission time between the two routers

Table 2: RL methods in TE

Cognitive Routing based on Deep Reinforcement Learning [10] relies on historical data analysis engine for optimal routing decisions by considering the inference of network quality state. The proposed Deep Reinforcement Learning (DRL) Single-Agent system uses Traffic Matrices to show the state of the whole system. The optimal policy is modified by using Deep Deterministic Policy Gradient (DDPG, a model-free, off-policy, actor-critic algorithm) and Temporal Difference (TD) Learning.

DeepRLB: A deep reinforcement learning-based load balancing in DCN [11] is a DRL-based load balancing approach for SDN DCN, which uses the DDPG algorithm to adaptively learn the link-weight values by observing the traffic flow characteristics. It exploits the flow statistics periodically collected by the controller polling or switch pushing mechanisms. At the next step, the statistics manager returns the acquired load balancing performance as a reward to the actor-network. The agent appends the obtained experiences at each step to the replay buffer, and the parameters of actor and critic networks are updated as soon as the buffer size exceeds the batch size. DeepRLB dynamically updates its knowledge from the environment (RL parameters) by repeating this process over the successive traffic demands in an SDN-based DC to improve the link-weight setting.

A Multi-agent Reinforcement Learning Perspective on Distributed Traffic Engineering was proposed in [13] as a data-driven framework for multi-region TE problems which involves the use of multi-agent deep reinforcement learning technique. There are two reinforcement learning agents that control the terminal traffic (T-agent) and outgoing traffic (O-agent) in each region. These agents collect local link utilization statistics each in their regions, optimize local routing decisions, and observe the resulting congestion-related reward. The operation of the two agents in every region adheres to the following scheme: the T-agent observes local region network status and routes terminal traffic for optimizing a local TE objective, while the O-agent solicits reward feedback from neighboring regions and routes outgoing traffic for optimizing a cooperative TE objective. The agents are first trained offline on test-bed simulating the network. During the online stage, the system can continue to improve and make suboptimal routing decisions quickly.

Machine Learning Ready for Traffic Engineering Optimization from [12] has proposed a new distributed system for TE, which uses the latest achievements in ML techniques. The proposed system architecture combines multi-agent reinforcement learning (MARL) method and graph neural networks (GNN) to minimize congestions in a network. The paper presents the empirical comparing the proposed MARL+GNN approach system with DEFO (a centralized network optimizer that translates high-level goals of operators into network configurations in real-time), a network optimizer based on Constraint Programming, which represents the state of the art in TE. Experimental results show that the proposed MARL+GNN solution has demonstrated almost the same performance as DEFO in a wide range of networks, including three real network topologies. At the same time, MARL+GNN solution significantly reduces execution time (from a few minutes in DEFO to a few seconds in MARL+GNN). The MARL+GNN method assumes a multi-agent approach with a decentralized way of interaction between agents (each agent is responsible for communication between network nodes). This feature of the method meets the requirements of our problem.

Packet Routing With Fully Distributed Multiagent Deep Reinforcement Learning method was proposed in [14]. This method was intended to solve one of the problems of the RL algorithms: the dimension of the network state space. The size of this space limits the possibilities for comprehensive representation of this space. Consequently its potential benefit is limited. This paper proposes a solution to this problem using a multi-agent DRL approach. In the proposed solution, each agent uses a recurrent neural network (RNN) with long short-term memory (LSTM). Network training and decision making are decentralized. The LSTM RNN extracts routing features from rich information regarding backlogged packets and past actions, and effectively approximates the value function of Q-learning. Each switch can periodically communicate with its direct neighbors, helping better define network states. The deep RNN uses three fully connected layers and one LSTM layer. The input of the neural network consists of Current Destination, Action History, Future Destinations and Max-Queue Node. LSTM maintains an internal state and aggregates observations over time. Activation function and optimization algorithm of the neural network are ReLU and RMSProp, respectively.

Packet Routing with Graph Attention Multi-agent Reinforcement Learning method was published in [15]. The paper presents a solution of TE problem for three different cases: centralized, federated and cooperative learning. Each router/node is considered a separate agent. A feature matrix (current node, neighboring node, or destination node) is transformed using a matrix of weights, attention coefficients, and LeackyReLu, which gives more "significant" features at the output. These features are needed to calculate Q-value in a neural network (each agent has its own, and the parameters are also different). As a result, it is shown that training without a central controller (as in centralized and federated training) shows the best results.

Output. We have considered various methods of RL based on single-agent and multi-agent learning approaches for network traffic TE. The main technique in RL methods was the Actor-Critic model. Training and evaluation processes can be either centralized or decentralized. In the centralized case the state of the whole network is supposed to be known. This can induce a significant delay before the balancing decision could be made. So, the decentralized approach is preferable, but nodes in the network should be able to exchange local information with each other to get

a consistent decentralized learning. None of the considered above methods provide the proper solutions for the problems listed in section 1.

4. MAROH method for TE

Here the new proposed traffic balancing method will be described. Repeat again that the goal of our method is to combine multi-agent reinforcement learning with hashing algorithms to provide a fair traffic allocation and maintain efficient network resource utilization. For that purpose on every t_i each NPCR egress port/channel will be assigned a certain weight, which will be used for allocation flows to channels by special hash-function. So the key problems are how to assign weights to egress ports and how to construct the proper hash-function.

In our scheme we will consider more options than just shortest paths to destination when choosing egress ports. However doing this might cause a loop to appear in the flow path. To avoid loops we have developed a special algorithm called next hop selection (NHS). It restricts the set of egress ports based on NPC topology subgraphs, explained in detail further.

To summarize the above, MAROH method consists of three parts, illustrated on figure 1:

1. The set of next hops (egress ports) is calculated by NHS algorithm in advance.
2. MARL algorithm calculates the weights for these ports.
3. Hashing algorithms distribute flows between channels according to calculated weights.

The following subsections describe each part in details.

4.1. Next hop selection algorithm

The proposed algorithm consists of two parts: the auxiliary one where some data structures will be generated and the main one where the next hop for every active flow on each router will be identified. The auxiliary part consists of the following steps:

1. Transform the original graph G to the undirected graph U without parallel edges[1] (if there is at least one arc between vertices in G, then U will have only one edge between the same pair of vertices).
2. Choose the root vertex $r \in U$ (for example, the vertex with the longest DFS path).
3. Obtain a directed acyclic graph $DAG(U)$ from U by DFS. Arcs in $DAG(U)$ are directed from the vertex with lower DFS number to the vertex with higher DFS number.
4. Construct a directed subgraph T_1 of G: any arc in T_1 exists if and only if there is the arc with the same direction between the same pair of vertices in $DAG(U)$.
5. Construct a directed subgraph T_2 of G: any arc in T_2 exists if and only if there is the arc with the opposite direction between the same pair of vertices in $DAG(U)$.

Now consider the algorithm for next hop port selection (see algorithm 1) for some vertex v from the original graph G. Assume the packet was received from vertex $p \in \{G \cup \varnothing\}$ and the packet destination is the vertex $d \in G$. The symbol $\varnothing$ denotes the case when the packet first arrived at the NPCR pole.

Algorithm 1 Next hop selection algorithm

```
next_hops = {}
if p ≠ ∅ then
    cg ← T1 if dfs(p) < dfs(v) else T2
    rg ← T1 if dfs(p) > dfs(v) else T2
    if v →cg d then
        for each n ∈ NBcg(v) do
            if n →cg d then
                next_hops+ = n
            end if
        end for
    else if v →rg d then
        for each n ∈ NBrg(v) do
            if n →rg d then
                next_hops+ = n
            end if
        end for
    else
        for each n ∈ NBcg(v) do
            if n →cg k and k →rg d then
                next_hops+ = n
            end if
        end for
    end if
else
    cg ← T1 if dfs(v) < dfs(d) else T2
    rg ← T1 if dfs(v) > dfs(d) else T2
    if v →cg d then
        for each n ∈ NBcg(v) do
            if n →cg d then
                next_hops+ = n
            end if
        end for
    end if
    for each n ∈ NBrg(v) do
        if n →rg k and k →cg d then
            next_hops+ = n
        end if
    end for
end if
```

[1] Everywhere below, the term edge will be used for undirected arcs in the graph, and the term arc will be used only for directed one.

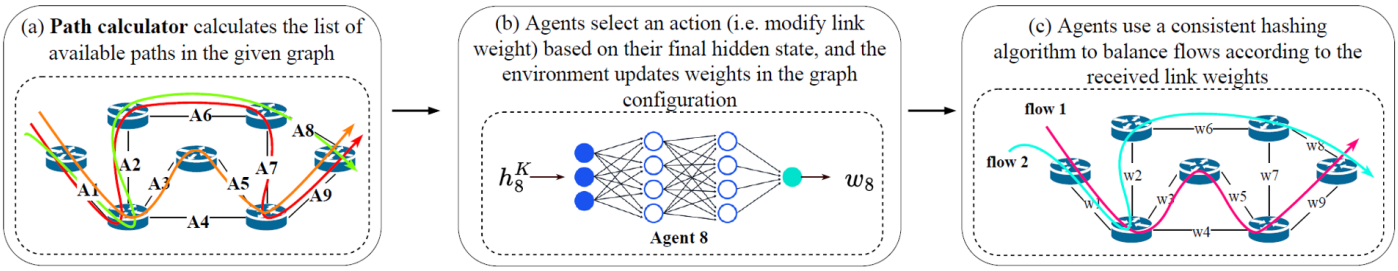

Figure 1: MAROH general operation scheme: (a) NHS algorithm produces feasible set of egress ports; (b) MARL algorithms produces weights for these ports; (c) hash-function distributes flows accordingly to these weights.

We used the following notations in algorithm 1:

- cg means *current graph* (either T_1 or T_2);
- rg means *reversed graph* (T_2 or T_1 respectively);
- $v \rightarrow_G d$ means that vertex d is reachable from vertex v in graph G;
- $NB_G(v)$ means the set of neighbors of vertex v in graph G.

Lines 2-23 correspond to the case when the packet is on the node within the path and lines 24-39 correspond to the case when the packet is on the first node of the path.

Lines 5-10 correspond to the case when we can reach our destination in the current subgraph (either T_1 or T_2). Therefore we don't need to switch to the other subgraph. Thus it guarantees there will be no cycles on the path because both T_1 and T_2 are DAG.

Lines 11-16 correspond to switching from one subgraph to other, while lines 18-23 stand for the search of path to the node, where we can switch between subgraphs. The condition "is reachable" could be calculated beforehand and can be limited by an additional parameter meaning the maximum path length.

In a case, when v is the first vertex on the path, we try to find the path in both subgraphs: lines 27-33 for searching in the first subgraph and lines 34-38 for the second one.

4.2. MARL algorithm

The proposed solution for traffic TE in NPC is an algorithm that runs on each NPCR. By this algorithm we have tried to combine two algorithms MARL and Graph Neural Network (GNN) follow to work [12] in such way that it will solve the problems mentioned in the introduction.

The general scheme of agent operation in the proposed method is illustrated on figure 2. GNN represents data network that transmits messages between network agents. For ordinary TCP/IP network these messages will include both the IGP (e.g. OSPF) messages with updated network state and agent messages with its hidden state. For NPC data plane the existed IGP protocols could be adapted.

We will start with special kind of GNN, named the Message Passing Neural Network (MPNN). Then MARL framework will be described. After that the proposed algorithm will be described in detail. Used notations are given in table 3.

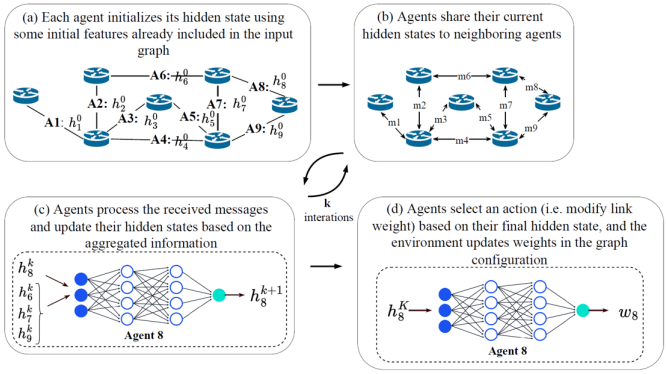

Figure 2: MAROH MARL operation scheme: (a) agent hidden state initialization, (b) - (c) - message exchanges and hidden state update, (d) MPNN evaluation of actions

4.2.1. MPNN organization

Message Passing Neural Network (MPNN) [16] is designed to learn the optimal message transmission between agents in the network represented by undirected graph. MPNN is based on an iterative message passing algorithm that passes information between selected graph elements – in our case we treat the agents as graph elements. Transmitted messages update the agent's state, which called *hidden state* h_v^0 in MPNN. In our case the hidden state is represented by occupied channel bandwidth $b_{i,v}$, channel capacity $c_{i,v}$ and current weights $r_{u,v}$. The hidden state can be initialized by random values or based on already collected statistics.

In our implementation hidden states h_v^0 are represented by 16 element vectors that include the information from the neighbors. As the hidden state size is limited, transmitted messages should be compressed and received messages should be aggregated. These operations are formalized by message function $m(\cdot)$ and aggregation function $a(\cdot)$.

It is assumed in MPNN approach that all agents exchange messages with their neighbors within the same time slot. It takes several time slots to distribute information

V	set of agents
v	specific agent in V
K	number of exchanges in MPNN
$B(v)$	all nodes in neighborhood of agent v
$m(\cdot)$	message function between two nodes
$a(\cdot)$	aggregation function in MPNN
$u(\cdot)$	update function applied to each agent v
$r(\cdot)$	readout function of agent hidden state
t	Reinforcement Learning step
s_t	state at step t
a_t	action at step t
r_t	reward at step t
$\gamma \in [0, 1]$	discount reward factor
S	state space
A	action space
$logit$	unnormalized log probability
h_v^t	hidden state of agent v at step t
$M_v^t = a(m(h_v^t, h_i^t)_{i \in B(v)}$	combined messages of MPNN
$h_v^{t+1} = u(h_v^t, M_v^t)$	new hidden state at step $t+1$
T	the step number of the episode end
$G_t = \sum_{t=o}^{T} \gamma^t r_t$	discounted cumulative reward
$\pi : S \to A$	policy, the agent behavior
$V^{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}[G_t \vert s_t = s]$	state value function

Table 3: MARL notations

to more agents in the network At each k-th message exchange, each agent v receives through messages the current hidden states of all nodes in its neighborhood $B(v)$ and processes them separately, applying the message function $m(\cdot)$, together with its own internal state h_v^k. The processed messages are then combined by aggregation function $a(\cdot)$:

$$M_v^k = a(\{m(h_v^k, h_i^k)\}_{i \in B(v)}) \tag{2}$$

At the end of k-th message exchange, every agent v applies an update function $u(\cdot)$ to the aggregated messages M_v^k and its current hidden state h_v^k. The output of $u(\cdot)$ is a new hidden state for the next step $(k+1)$:

$$h_v^{k+1} = u(h_v^k, M_v^k) \tag{3}$$

After a certain K message exchanges (configuration parameter of MAROH method), a readout function $r(\cdot)$ produces the final output of the MPNN using final node states h_v^K as input. This readout function will evaluate the actions of weight changes.

Note that the MPNN model generates a single set of message function, aggregation function, update function and readout functions that are replicated in each NPCR. This means that these functions must be versatile and flexible enough to adapt their behavior to different scenarios, so they are usually modeled as multi-layer perceptron, with the sole exception of the aggregation function, which is usually a piecewise summation. Neural networks are described by weights $\theta = \{\theta_i\}_{i \in m,a,u,r}$. So MPNN becomes the net that consists of multi-layer perceptrons.

4.2.2. MARL framework

In the standard reinforcement learning setting, an agent interacts with the environment in the following way: at each step t, the agent selects an action a_t based on its current state s_t, to which the environment responds with a reward r_t and then moves to the next state s_{t+1}.

This interaction is modeled as an episodic, time homogeneous Markov decision process (MDP) (S, A, r, P, γ). S and A are state and action spaces, respectively. P is the transition kernel: $s_{t+1} \sim P(\cdot|s_t, a_t)$. r_t represents the immediate reward provided by the environment after performing an action when t is in state s_t. $\gamma \in (0, 1]$ is the discount factor used to compute the return G_t, defined as the discounted-cumulative reward from a given time step t to the end of episode T (a preset value for the time interval, the number of time steps t): $G_t = \sum_{t=0}^{T} \gamma^t r_t$.

The agent's behavior is defined by a policy $\pi : S \to A$, which maps each state to a probability distribution over the action space. Then the agent's RL goal is to find the optimal policy under which for any considered state $s \in S$ the chosen action maximizes the expected return G_t. The agent learns an explicit policy representation π_θ with parameter θ – the input of MARL framework, which we calculate in MPNN model. In most cases, agents learn also the approximation $V_\phi(s)$ of the state value function $V^{\pi_\theta}(s)$, defined as the expected discounted return from a given state s by following policy π_θ: $V^{\pi_\theta}(s) = \mathbb{E}_{\pi_\theta}[G_t|s_t = s]$. MAROH method uses Actor-Critic policy gradient algorithm, where actions are selected from a function that evaluates the policy (i.e., the actor), and learning such a policy is guided by the value function of the actions consequences (i.e., the critic). Our solution is based on Proximal Policy Optimization (PPO) [17], which offers a favorable balance between reliability, sample complexity, and simplicity.

Each agent is responsible for a single link configuration, controlling the traffic only in one direction. State is defined as link hidden representation consisting of current weight $r_{u,v}$ and occupied channel bandwidth $b_{u,v}$ of each agent. Action is defined as modification of the weight of agents' associated channel (increase the weight's value by one). And reward is defined as difference of the optimization function Φ (1) between consecutive steps. The

obtained weights will be used by hash-function to allocate flows among channels.

Lines 4-8: denote the MPNN part, which produces updated hidden states for the agents. Once agents generate their final hidden representation, a readout function, following the MPNN design [16], is applied to obtain the global policy π_θ.

Line 9: for every possible action in agent's action space, it computes its logit – unnormalized log probability, based on readout function of trained MPNN model. *logit* is understood as ML term that means classification model generates the vector of raw (non-normalized) predictions, passed to a normalization function. Logits typically become an input to the softmax function - normalized exponential function. Then a vector of (normalized) action probabilities is generated by softmax function.

Algorithm 2 MARL-TE agent algorithm

Require: A graph $G = (N, E)$ with a set of agents V, MPNN trained parameters (weights of neural network nodes) $\theta = \{\theta_i\}_{i \in m,a,u,r}$ Initial graph configuration X_G^0 (information about nodes and links with their attributes), episode length T, number of message passing steps K

1: Agents initialize their states s_v^0 based on X_G^0
2: **for** $t \leftarrow 0$ to T **do**
3: $h_v^0 \leftarrow (s_v^t, 0, ..., 0)$
4: **for** $k \leftarrow 0$ to K **do**
5: Agents share their current hidden state h_v^k to neighboring agents $B(v)$
6: Agents process the received messages: $M_v^k \leftarrow a_{\theta_a}(\{m_{\theta_m}(h_v^k, h_\mu^k)\}_{\mu \in B(v)})$
7: Agents update their hidden state $h_v^{k+1} \leftarrow u(h_v^k, M_v^k)$
8: **end for**
9: Agents compute their actions' *logits*: $\{logit_v(a)\}_{a \in A_v} \leftarrow r_{\theta_r}(h_v^K)$
10: Agents receive the actions' logits of the rest of agents and compute the global policy $\pi_\theta \leftarrow CategoricalDist(\{\{logit_v(a)\}_{a \in A_v}\}_{v \in V})$
11: Using the same probabilistic seed, agents select an action $a_t \in A_{v'}, for v' \in V$, from policy π_θ
12: Agent v' executes action a_t
13: Agents update their states s_v^{t+1}
14: Evaluation of $\Phi = \{\frac{1}{N}\sum_{u,v}(\frac{b_{u,v}}{c_{u,v}} - \mu')^2\}$ function
15: **end for**
return Updated graph configuration X_G^* that optimizes some pre-defined objective (i.e., Φ function)

Line 10: a categorical distribution is a discrete probability distribution. It describes the probability that a random variable will take on a value that belongs to one of K categories, where each category has a probability associated with it. So at this step, we know all probabilities to make different actions a_t in state s_t also known as policy π. And categorical distribution of all these probabilities is the global policy π_θ.

Lines 11-12: to ensure that all agents sample the same action along the message-passing process $a_{v'}^t \sim \pi_\theta(\cdot|s_t), v' \in V$, they share a common seed before initiating this process. Consequently, only the agent v' whose action has been selected does execute an action at each time-step t. Each action changes the weight $r_{u,v}$, that leads to the change in the optimization function Φ.

Lines 12/13: the main task of agents is to set channel weights to balance the load, which is under control of hashing algorithm. Agents update new state information for next timesteps by channel monitoring in NPCR.

Line 14: on every time slot t_i each NPCR calculates the optimization function Φ (1).

Formally, the training goal of our RL's algorithm is to optimize the parameters $\{\theta, \Phi\}$ so that:

- The previously described GNN-based actor π_θ becomes a good estimator of the optimal global policy;
- The critic V_Φ learns to approximate the state value function.

The optimal global policy is sought in the global state space, the union of the agent's local states. In particular, the training pipeline operation is following: an episode of length T is generated by following the current policy π_θ, while at the same time the critic's value function V_Φ evaluates each observed global state. Therefore the episode defines a trajectory $\{s_t, a_t, r_t, p_t, V_t, s_{t+1}\}_{t=0}^{T-1}$, where $p_t = \pi_\theta(a_t|s_t)$ and $V_t = V_\Phi(s_t)$. When the episode ends, this trajectory is used to update the model parameters by maximizing the global PPO objective $L^{PPO}(\theta, \Phi)$.

4.2.3. Hash-function

The mission of hash-function in MAROH method is to distribute flow according to weights, provided by MARL. Additionally we require that hash-function preserves egress ports assigned to flows in most cases, even if the weights or number of egress ports have changed. Such hash-function called consistent. Egress port preservation helps to maintain stable state of congestion control algorithm for TCP flows. Otherwise the flow route change may violate packet arrival order, leading to the incorrect congestion recognition.

To provide consistent hashing for flow balancing we chose DxHash hash-function [18]. DxHash uses the Pseudo-Random Sequence to map the keys to the nodes. The implementation of DxHash is based on the Pseudo-Random Generator (PRG) and the Pseudo-Random Sequence (PRS). PRG is a random function that has the property to always return the same result for a fixed seed. And for different seeds, the results are evenly distributed over the range. Denote $R(s)$ as the ideal PRG. PRS is generated by applying PRG a given number of times. $R^2(x)$ denotes the superposition of R - $R(R(x))$. Thus, when the seed s and the PRG $R(x)$ are given, the PRS is $\{R(s), R^2(s), ..., R^n(s)\}$.

Hash-function is determined by lookup and update operations. The lookup operation for a given key in DxHash is as follows. DxHash uses PRS iteratively to find appropriate egress ports in the set of all egress ports. Denote by A the set of all egress ports, W to denote the set of enabled ports, and F to denote the set of disabled ports, $A = W \cup F$. For a given key DxHash chooses port b as the mapped port where $b = R^i(key) mod |A|$ and i is the minimum number that satisfies $b \in W$.

There are two kinds of egress port update operations: adding and removing. When a port becomes enabled, DxHash randomly allocates a failed port ID to the new one. Then DxHash removes the port ID from the set of disabled ports F and adds it into the working set W. When a port becomes disabled, the corresponding port ID is removed from the working set W and inserted into the set F. After the working set is changed, the lookup result for related keys is changed accordingly. When the number of the enabled ports is far less than the number of all ports $|A|$, for example less than $1/4$ of $|A|$, the port set decreases. First, the ports, whose IDs are greater than $(|A|)/2$ are removed. Then, the set of all ports A is halved, and the ports whose IDs are greater than $(|A|)/2$ are removed from F.

Weighted DxHash introduces a weight for each port and a hash-function denoted as H. Both the function and the weight value belong to the interval of[0,1]. In Weighted DxHash the conditions to map a key to a port are more restrictive. Not only the i-th item in PRS is the enabled port, but also the hashing value of the i-th item should be smaller than the weight of the port. Denote the weight of i-th port as $A[i]$. The weight of the disabled port is 0. The port ID which k belongs to is the first item in the PRS which satisfies: $H(R^i(k)) < A[b]$, where $b = R^i(k) mod |A|$.

5. Experimental results

In our simulation experiments we have analyzed the different ways of training the algorithm, evaluated how GNN parameters affect the training process, estimated the speed of convergence and the stability of the results when changes in the topology happen.

We consider symmetric 16-node topology (Figure 3). Due to lack of information about the link capacities, all of them are set to 4 GBit/s.

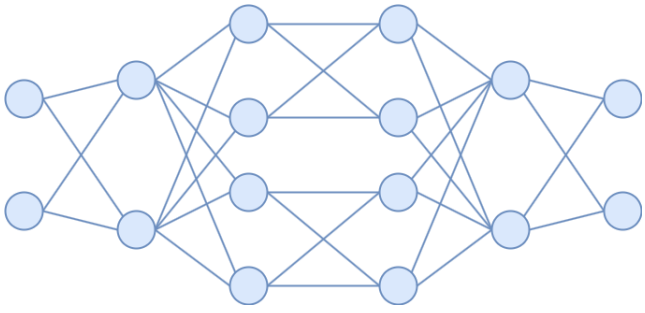

Figure 3: Symmetric 16-node topology

In our simulation experiments for flow generation we use traffic matrices, one per time slot $[t_0, ..., t_L]$. Recall that traffic matrix TM entry $TM[u, v]$ represents the total amount of traffic that should be transferred between of nodes u and v. For TMs generation we use gravity model [20] and its extension for sequence of traffic matrices in [21]. The base value $TM^*[u, v]$ is calculated proportional to the outgoing capacities of the nodes and inversely proportional to the shortest path length between the nodes. We introduce coefficients u_{min} and u_{max} to generate the subsequent values of $TM[u, v]$ for each time slot by following uniform distribution $U[u_{min} * TM^*[u, v], u_{max} * TM^*[u, v]]$. Those coefficients can be used to change the resulting average link load in the topology.

To generate the set of flows from obtained traffic matrices we introduce the following parameters: f_{max} is the maximum amount of flows between any pair of nodes, d_{min}, d_{max} are the minimum and maximum flow duration, P is the length of a time slot. The set of flows for the first time slot is obtained by randomly choosing numbers $n_{u,v} \in [1, f_{max}]$ for $u, v \in [1, N]$ - this will be the number of flows between nodes u and v to be generated. For each flow its duration is chosen from uniform distribution $U[d_{min}, d_{max}]$. The bandwidths of the flows are chosen by uniform sampling so that their bandwidths sum up to $TM[u, v]$. For the following time slots we first examine if the time slot length and the actual duration of any of the flows from previous time slot allow them to transfer into the next time slot. If $r_{u,v}$ is the amount of flows transferred this way between nodes u and v, then $n_{u,v}$ is now randomly chosen from the uniform distribution $U[r_{u,v}, f_{max}]$. This way allows the flows to exist in the topology during multiple time slots, which is essential for hash-function to consistently balance them in the environment where hash weights change dynamically.

To run the simulation experiments, we have implemented the algorithm and a testbed in Python using tensorflow library (version 2.11).

Every simulation experiment has the following steps:

1. Initialize hash weights to be equal to 1 for all links.
2. Get flows for current time slot from TM.
3. Run one iteration of the MARL algorithm to get new hash weights.
4. With the hash weights from step 3, run hash function to calculate currently occupied bandwidth on all the links in the topology.
5. Calculate goal function value Φ (1).
6. If current time slot is not the last one, move to the next time slot and go to step 2.

5.1. Algorithm convergence estimation

For algorithm convergence estimation we do not need several traffic matrices. Just one will be enough ($L = 1$). Experiments with this setup let us conclude that the algorithm achieves convergence and estimate how fast it will get it. From practical standpoint, the results of such

simulation can be useful if there is a way to predict the flow distribution in the network for the next time slot.

One flow set was generated for each of the following average network loads: 20%, 40% and 60%. Each flow set had 50 traffic matrices, $f_{max} = 2, d_{min} = 25s, d_{max} = 700s, p = 30s$. The algorithm runs up to 4500 episodes.

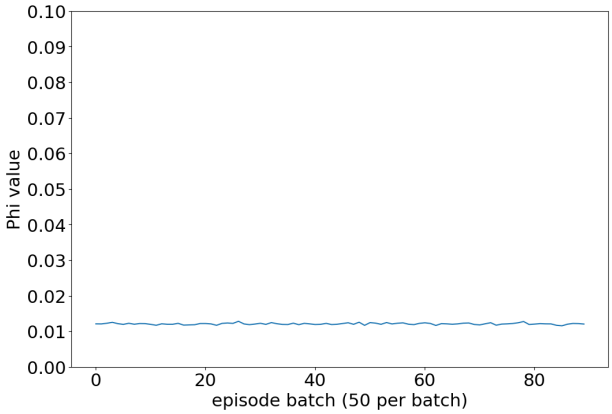

Figure 4: Algorithm convergence for 20% average network load

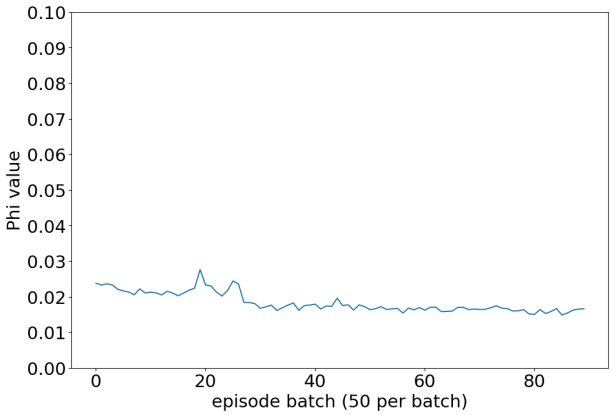

Figure 5: Algorithm convergence for 40% average network load

The results are presented in Figures 4, 5 and 6 for, respectively, 20%, 40% and 60% average network load. Every point of data on the graph corresponds to a phi value averaged from 50 episodes. The lack of convergence for 20% load can be explained by agent reward calculation method. In these experiments reward for an action was calculated as phi value difference from the old value to the new value. In the 20% load dataset the phi values are too small for algorithm to learn on. Another explanation is the values might already be close to optimal values.

In the 40% load dataset the algorithm demonstrates an improvement after 1500 episodes. The phi values after 1500 episodes are about 25% better than in the first 1500 episodes.

In the 60% load dataset the algorithm demonstrates almost 100% improvement after 500 episodes, and the results finally stabilize after 3000 episodes. In can be concluded

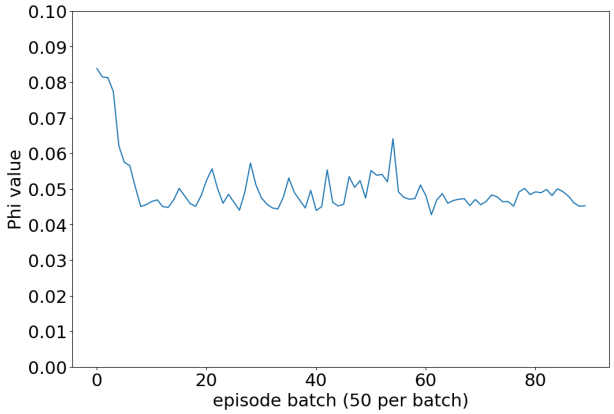

Figure 6: Algorithm convergence for 60% network load

that the algorithm is most effectively used in high network load scenarios.

5.2. Comparison of training methods

In this section we explore three methods to train and to run the algorithm on datasets with multiple traffic matrices: sequential, random, untrained.

Sequential method means that the algorithm is first run on a training dataset. In this dataset traffic matrices are processed in the same order they were generated: TM_0 corresponds to time slot t_0, TM_1 corresponds to time slot t_1 and so on. The algorithm runs fixed amount of episodes on each traffic matrix. The algorithm trained this way is then run on evaluation dataset.

Random method also means that the algorithm is first run on a training dataset, but opposite to the sequential method, the traffic matrix order is randomly shuffled. The algorithm runs only one episode on each traffic matrix, and when it reaches the last traffic matrix, the traffic matrix order is randomly shuffled again and the process repeats again, until the algorithm reaches predefined number of episodes. The algorithm trained this way is then run on evaluation dataset.

Untrained method means that there was no initial training and the algorithm is run on evaluation dataset immediately and trains on the fly.

The average network load of both datasets was 40%. Generator parameters were set according to section 5.1. The results are shown in Figure 7 by box plot.

As it can be seen from fig. 7 random training shows the best median results, but untrained case has better average value. However sequential method has the least amount of ejections. For the further experiments we use random training method.

5.3. MAROH Advantages

In this section we will demonstrate the advantage obtained by the combination of our NHS, MARL and hash-function. Consider three approaches to compare:

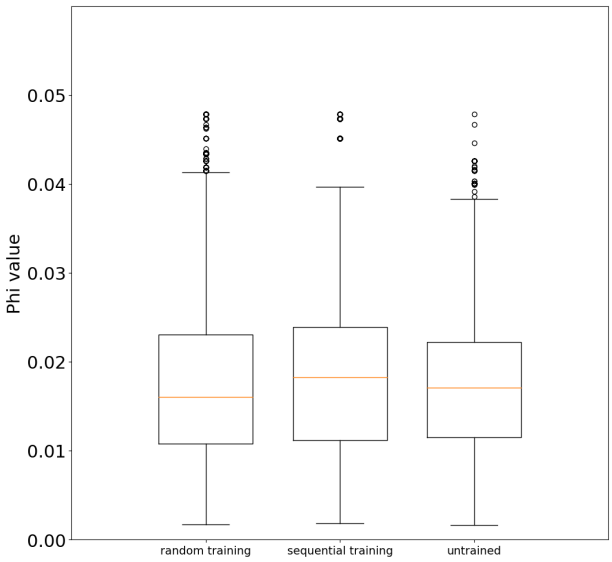

Figure 7: Experimental comparison of training methods

1. All egress port weights are equal to 1. NHS algorithm only considers nexthops than belong to the shortest paths by hops to destination. Hashing is done by DxHash function. Call this approach ECMP-hashing.
2. All egress port weights are equal to 1. NHS algorithm works according to section 4.1. Hashing is done by DxHash function. Let this approach be called Weightless MAROH.
3. MAROH approach as described in section 4.

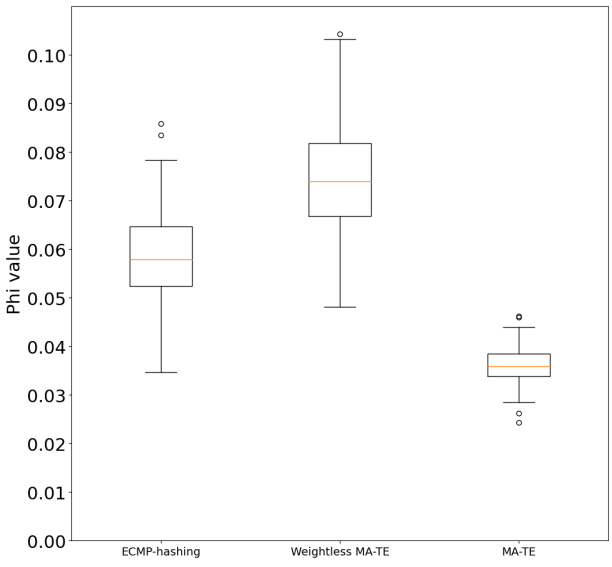

Figure 8: Advantage of the MAROH approach

Figure 8 shows 200 runs of each approach on the same dataset. The dataset was generated at 60% average network load with parameters chosen according to section 5.1. MAROH approach was trained on another dataset generated with the same parameters by performing random training and running 10000 episodes. The figure shows that full MAROH approach has significantly better results compared to approaches where only parts of the full MAROH approach are used.

5.4. Stability under topology changes

In this section we will analyze to what extent the topology changes have influence on the behavior and results of the trained algorithm. This implies that topology changes can be significant compared to the one on which the algorithm was trained.

The algorithm was trained on a single traffic matrix dataset by performing 10000 episodes. The dataset was generated at 40% average network load with parameters chosen according to section 5.1.

In the first experiment the algorithm was run 300 times, and in each run two random connected nodes u and v were chosen. All edges between u and v in both directions were removed from the topology. The algorithm outputted the average goal function value from 10 runs in the updated topology.

The second experiment was performed identically except this time two pairs of connected nodes were chosen: u, v and m, n. All edges between u and v, and all edges between m and n were removed from the topology. Additional condition was $u \neq v \neq m \neq n$ to keep the topology connected at all times.

The results of the experiment are shown in Figure 9. Blue line indicates the average goal function value obtained by running the algorithm on full topology 10 times.

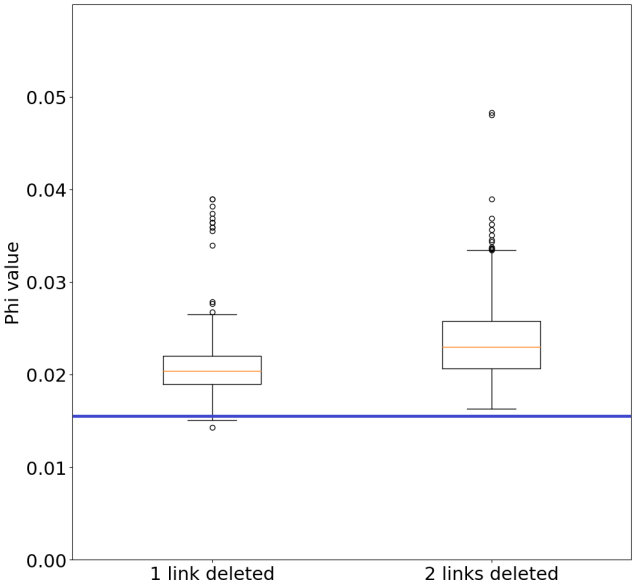

Figure 9: Link removing experiment results.

The figure 9 shows that the median and spread are increased by removing 1 link from a topology with a smaller increase when 2 links are removed. Actually we found

that the result depends on which link was removed. Certain links barely increase the phi value, however there are cases when removing one link can lead to almost tripling the phi function as can be seen on the ejections in the graphs.

5.5. MPNN parameter tuning

So far in the research the MPNN parameters were chosen as follows: the number of exchanges K was equal the the topology graph's diameter, and the length of a single MPNN message M was equal to 16. The value of parameter K was chosen as such because our experiments show any values higher than that demonstrated poor behavior: *logits* produced by the agent were less dependent on s_t and were more and more similar for different agents with rising value of K.

In the final section of the experimental research we will examine how M affects the algorithm's convergence. K will be fixed to topology graph's diameter, and M will vary between 2 and 16.

MAROH convergence speed will be estimated on a single traffic matrix. We define it as number of episodes required for the average phi value from 100 episodes to be less than a fixed value C. We assume $C = 0.015$ and for each value of M the algorithm was run 10 times.

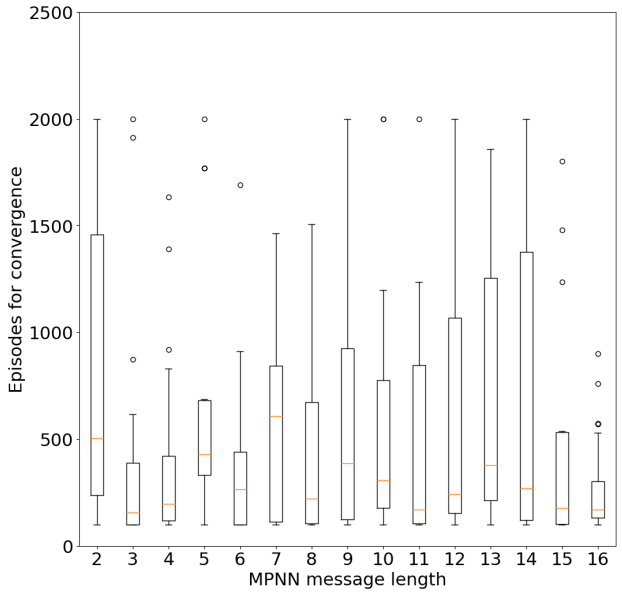

Figure 10: Number of episodes for convergence with varying length of the MPNN message

The dependence of the convergence speed on M is shown on Figure 10. This parameter has a great influence on convergence speed. Low values of M demonstrate unstable convergence: even though with $M = 3$ and $M = 4$ the median value is almost as low as with $M = 16$, there are cases when the algorithm takes from 3 to 20 times the number of episodes to converge. The lowest values of M with more stable convergence speed are $M = 7$ and $M = 8$. The median value and range are higher compared to $M = 16$, but it comes with the benefit of smaller messages. These results are useful to reduce the amount of extra information transferred in the network.

6. Conclusion

The problem of traffic balancing in a data network environment is considered in relation to Network Powered by Computing (NPC) - a new generation of computational infrastructure, where computational resources and data transmission resources form a network. The key contribution of this work is Multi-Agent Routing using Hashing (MAROH) method that provides fast and fair bandwidth distribution inside a data transmission network.

There are three approaches to MARL optimization: centralized, decentralized with communication and fully decentralized. It is assumed in all these approaches that agents construct their local state based on the environment observations. However the agents behavior in these approaches are different:

- centralized approach: a single entity named control center is responsible for the operation of every agent. This center collects the local states from all agents and defines the actions for every agent by solving an optimization problem.
- decentralized approach with communication (or just decentralized approach): assumes that the local state of every agent can be broadcasted to all neighboring agents. Then the agent himself chooses an optimal action based on the knowledge of local and neighboring agent states.
- the fully decentralized approach: do not suppose any communication between agents. Decision should be made based only on the history of the agent local behavior observations.

The main problems of MA methods for traffic control are poor scalability; there are no mathematical models that guarantee convergence to the optimal solution; it is difficult to mathematically frame the optimization functional; the extent of deviation from the optimal solution is unknown.

In this paper we focus on the decentralized approach as NPCRs are able to communicate with each other and there is no single control center in NPC. Based on examining of the works related to ML methods for load balancing, a new method is proposed called MultiAgent Routing with Hashing (MAROH) to surmount the problems listed above. This method combines multi-agent reinforcement learning and hashing algorithms to achieve fast and fair bandwidth distribution.

The experimental results show that proposed MAROH method achieves convergence and is most efficient under high network load. Three ways to train the algorithm were

examined and compared: all three are deemed suitable for different scenarios.

The scope of the research presented in this paper does not cover the point of deviation between the goal optimization function and the optimal one. The space of the paper does not have enough room for that. The result of the research of this point will be topic for the next paper.

7. Acknowledgements

Authors would like to thank Ozerova Daria, Tsvetkova Vera and Morozova Veronika, undergraduate students of Lomonosov Moscow State University for their contribution to the experimental part of this research.

The part of this research regarding studying MPNN organization was supported by the Interdisciplinary Scientific and Educational School of Moscow University "Brain, Cognitive Systems, Artificial Intelligence".

References

[1] R. Smeliansky, "Network Powered by Computing," 2022 International Conference on Modern Network Technologies (MoNeTec), 2022, pp. 1-5, DOI: 10.1109/MoNeTec55448.2022.9960771

[2] L. Geng and P.Willis, "Compute First Networking (CFN) scenarios and requirements," in IETF RTGWG Working Group, 2019.

[3] Yukun Sun, Bo Lei, Junlin Liu, Haonan Huang, Xing Zhang, Jing Peng, Wenbo Wang Computing Power Network: A Survey. arXiv:2210.06080

[4] Smelyanskiy R. Network Powered by Computing: Next Generation of Computational Infrastructure. In Edge Computing - Technology, Management and Integration (ISBN 978-1-83768-862-3)

[5] Tang, Fengxiao, et al. "Survey on machine learning for intelligent end-to-end communication toward 6G: From network access, routing to traffic control and streaming adaption." IEEE Communications Surveys & Tutorials 23.3 (2021): 1578-1598.

[6] Xiao, Yang, et al. "Leveraging deep reinforcement learning for traffic engineering: A survey." IEEE Communications Surveys & Tutorials 23.4 (2021): 2064-2097.

[7] Chen, Li, et al. "Auto: Scaling deep reinforcement learning for datacenter-scale automatic traffic optimization." Proceedings of the 2018 conference of the ACM special interest group on data communication. 2018.

[8] Mai, Tianle, et al. "Multi-agent actor-critic reinforcement learning based in-network load balance." GLOBECOM 2020-2020 IEEE Global Communications Conference. IEEE, 2020.

[9] Lin, Qinliang, et al. "Rilnet: A reinforcement learning based load balancing approach for datacenter networks." Machine Learning for Networking: First International Conference, MLN 2018, Paris, France, November 27–29, 2018, Revised Selected Papers 1. Springer International Publishing, 2019.

[10] Wu, Jiawei, et al. "Towards cognitive routing based on deep reinforcement learning." arXiv preprint arXiv:2003.12439 (2020).

[11] Rikhtegar, Negar, Omid Bushehrian, and Manijeh Keshtgari. "DeepRLB: A deep reinforcement learning-based load balancing in data center networks." International Journal of Communication Systems 34.15 (2021): e4912.

[12] Bernárdez, Guillermo, et al. "Is machine learning ready for traffic engineering optimization?." 2021 IEEE 29th International Conference on Network Protocols (ICNP). IEEE, 2021.

[13] Geng, Nan, et al. "A multi-agent reinforcement learning perspective on distributed traffic engineering." 2020 IEEE 28th International Conference on Network Protocols (ICNP). IEEE, 2020.

[14] You, Xinyu, et al. "Toward packet routing with fully distributed multiagent deep reinforcement learning." IEEE Transactions on Systems, Man, and Cybernetics: Systems 52.2 (2020): 855-868.

[15] Mai, Xuan, Quanzhi Fu, and Yi Chen. "Packet routing with graph attention multi-agent reinforcement learning." 2021 IEEE Global Communications Conference (GLOBECOM). IEEE, 2021.

[16] Gilmer, Justin, et al. "Neural message passing for quantum chemistry." International conference on machine learning. PMLR, 2017.

[17] Schulman, John, et al. "Proximal policy optimization algorithms." arXiv preprint arXiv:1707.06347 (2017).

[18] Dong, Chaos, and Fang Wang. "DxHash: A Scalable Consistent Hash Based on the Pseudo-Random Sequence." arXiv preprint arXiv:2107.07930 (2021).

[19] Simon Knight, Hung X. Nguyen, Nickolas Falkner, Rhys Bowden, Matthew Roughan. "The Internet Topology Zoo", IEEE Journal on Selected Areas in Communications, Vol. 29, No. 9, October 2011

[20] Md. Mostafizur Rahman, Subrata Saha, Usha Chengan, Attahiru Sule Alfa, "IP Traffic Matrix Estimation Methods: Comparisons and Improvements", IEEE International Conference on Communications. IEEE, 2006. doi: 10.1109/icc.2006.254710.

[21] Nan Geng, Tian Lan, Vaneet Aggarwal, Yuan Yang, Mingwei Xu, "A Multi-agent Reinforcement Learning Perspective on Distributed Traffic Engineering", IEEE 28th International Conference on Network Protocols (ICNP). IEEE, Oct. 2020. doi: 0.1109/icnp49622.2020.9259413.