AMAC: attention-based multi-agent cooperation for smart load balancing

Omar Houidi, Sihem Bakri, Djamal Zeghlache, Julien Lesca, Pham Tran Anh Quang, Jérémie Leguay, Paolo Medagliani

AMAC: Attention-based Multi-Agent Cooperation for Smart Load Balancing

Omar Houidi*, Sihem Bakri*, Djamal Zeghlache*,
Julien Lesca†, Pham Tran Anh Quang†, Jérémie Leguay†, Paolo Medagliani†
*Telecom SudParis, Samovar-Lab, Institut Polytechnique de Paris, France
†Huawei Technologies Ltd., Paris Research Center, France

***Abstract*—In cooperative multi-agent reinforcement learning (MARL), efficient communication among agents requires the reduction of excessive message exchange at run-time to make it practical for real-world applications. This paper proposes a novel communication scheme, *Attention-based Multi-Agent Cooperation* (AMAC), that reduces overhead and shared information by exchanging only relevant messages across agents to coordinate decision-making and improve load balancing in networks. Experiments show that AMAC can significantly lower inter-agent communications overhead and learning complexity at the network controller level without degrading performance. The results demonstrate that our method actually outperforms multiple MARL benchmarks in Key Performance Indicators KPIs (such as throughput, delay, jitter), and Key Quality Indicators KQIs (such as QoE, average video bitrate, stalling).**

***Index Terms*—Multi-Agent, Reinforcement Learning, Smart Load Balancing, QoE Optimization, Overhead reduction.**

I. Introduction

Load balancing (LB) plays a key role in enhancing Quality of Service (QoS) by using more efficiently network resources [1]. The most popular load balancing mechanism, i.e. Unequal Cost Multipath routing (UCMP), splits traffic aggregates evenly over candidate paths between source and destination pairs. In Software-Defined Networking [2] architectures, a network controller is responsible for adjusting load balancing weights over time. Centralized methods such as Niagara [3] have been proposed to control the split of traffic so as to minimize a linear routing cost or the Maximum Link Utilization (MLU). However, these proposals are not explicitly trying to improve QoS metrics such as the end-to-end latency. An alternative to these traditional approaches consists in tackling the load balancing problem through a model-free approach that can learn and reinforce its decisions over time. Even if network calculus or queuing models can estimate latency, packet loss, and jitter, the associated models remain quite complex and difficult to integrate in global load balancing solutions. Model-free reinforcement learning approaches have been proposed as a promising solution to optimize QoS [4]. However, the presence of a centralized control entity may not be desirable for scalability reasons and can yield a high overhead. Indeed, the execution of a unique reinforcement learning model requires the exchange of a large amount of data in a timely manner.

To address this limitation on scalability, this paper studies semi-distributed solutions based on cooperative Multi-Agent Reinforcement Learning (MARL) that has achieved remarkable success in a variety of challenging areas including intelligent traffic signal control, autonomous driving, and sensor networks [5]. In MARL, multiple agents interact at run-time in a shared environment to realize distributed and cooperative decision-making. Each agent has access to partial environment observations and makes local decisions based on messages received from other agents. This work aims to reduce the overhead of semi-distributed multi-agent RL-based load balancing solutions without degrading accuracy and performance. This is realized by using attention mechanisms and limiting agent cooperation to most relevant neighbors so that agents can learn to select valuable observations and exclude others. Our goal is to show that focusing only on what contributes most to learning, convergence, and stability and ignoring less useful data leads to significant overhead reduction and complexity reduction without sacrificing performance.

A widely used MARL paradigm known as *Centralized Training and Decentralized Execution* (CTDE) [6], [7], learns agent policies centrally and executes the derived policies decentrally. Several CTDE learning approaches have been proposed, including both policy gradient and value-based methods [8] shown to achieve reasonably good performance in challenging tasks. Despite its success, the execution of fully decentralized policies suffers from limitations, particularly when agents have partial observability in a stochastic environment. In fact, during decentralized sequential execution, an agent's uncertainty about the states and actions of other agents can be aggravated and lead to sub-optimal policies. To address this limitation, some works enable inter-agent communication [9], [10]. This exchange of messages between agents enhances local observations, and improves decision policies leading to better actions. However, in complex settings, frequent inter-agent communication leads to poor performance as it causes higher learning complexity, higher overhead, and expands the agents' local policy spaces.

Motivated by these observations, we design AMAC, a novel deep MARL architecture that can significantly improve inter-agent communication efficiency at the agent-level by exchanging only relevant messages for cooperation without impacting accuracy. AMAC combines multi-head attention [11] and a QMIX [12] adapted mixing network that results in an attention-based value-mixing network that transforms the individual agent Q-values Q_i into a global value Q_{tot}, instead of directly summing up all the local Q-values. AMAC makes decisions on load balancing weights in a semi-distributed fashion, by learning a centralized joint action-value function Q_{tot} and by using it to guide the optimization of semi-decentralized policies at each agent. The agents select actions locally (in our case selecting a path among candidate paths at an origin node for improved load balancing), based on Q_{tot}, Q_i and their individual local observations. The controller takes as input the most important summarized information returned by agents to produce the joint and global value function Q_{tot}.

This paper addresses the problem of finding the relevant

subset of neighbors an agent can cooperate with and identifying the most useful information to exchange between agents. The idea is to avoid using a fixed adjacency matrix representing the allowed communication between agents. In addition, we avoid using a fixed number of neighboring nodes and instead select the subset of neighboring agents most conducive to successful training steps. In summary we propose a new smart load balancing approach using multi-head attention mechanisms combined with a multi-agent *value decomposition* [13] that decomposes the global shared multi-agent Q-value Q_{tot} into individual Q-values Q_i to guide the behavior of the local agents. We intend to adopt a semi-distributed (i.e., where decentralized policies are learned centrally) and cooperative multi-agent approach that generates less overhead than purely centralized and semi-distributed approaches, by reducing the amount of data treated by the controller and at the same time does not suffer from the performance limitation of fully distributed approaches. Our objective is to strike a balance between cooperation overhead and performance by conducting most of the reinforcement learning locally at the agent-level and sending only the relevant local Q-value updates to the controller. In parallel, we rely on a scoped and circumscribed cooperation between the distributed agents to reduce the amount of data shared with neighboring agents. This is realized by using the attention mechanism and the identification of significant neighbors, as well as the pruning out of less influential agents in the local decisions. The net result is the clustering of neighbors (using the neighbor matrix weight values) in subgroups that have a significant impact on the local decisions. Furthermore, using an attention mechanism, we reduce the complexity of the controller by transforming the agent Q-values into a weighted Q-value that simplifies the processing of a mixing network.

To evaluate the performance of our approach, we consider networks conveying video flows whose QoE needs to be optimized. This use case is relevant for RL-based methods since the impact of load balancing decisions on QoS and QoE has to be learned. In addition, since collecting QoE at the clients is often unfeasible, we instead rely only on network-level collected KPIs to make decisions and derive load balancing policies.

Section II provides an overview of traditional LB and MARL methods. Section III formulates the problem and section V describes the proposed model. Section VI reports the performance evaluation results in simulations and section VII summarizes the main findings.

II. Related Work

Equal-Cost Multi-Path routing (ECMP) [14] or UCMP have been extended to perform path selection based on the least utilized path so as to better balance traffic according to target load balancing weights. However, these traditional schemes base decision only on their knowledge of network conditions for a given short period of time when long-term variation knowledge is actually required to provide more relevant solutions. They also cannot optimize end-to-end QoS parameters as they cannot predict the impact of their decisions. To address these issues, model-free approaches based on Machine Learning (ML) and Artificial Intelligence (AI) have recently emerged as candidate solutions to optimize *Quality of Experience* (QoE) as well as correctly predicting future conditions [4].Deep multi-agent reinforcement learning (Deep-MARL) [8] approaches have achieved outstanding results in recent years and shown superior performance in many real-world tasks where multiple agents learn to cooperate despite their private observations and limited communication ability. Therefore, the cooperation between agents in a MARL environment has been adopted by several research papers including DGN [15], RILNET [16], and QMIX [12] to improve the overall performance of value-based multi-agent methods.

A relevant MARL-based approach is DGN [15], which primarily tackled the more general routing problem in networks. DGN proposes a multi-agent cooperation algorithm based on convolutional graph reinforcement learning suitable for capturing the dynamics of the multi-agent underlying environment connectivity graph. DGN uses a deep-Q-network and shares weights among all agents. DGN abstracts the mutual interplay between agents by relation kernels, extracts latent features by convolution, and induces consistent cooperation by temporal relation regularization. Unlike DGN, which uses a fixed number of neighboring nodes and a fixed set of adjacency matrices, our proposed algorithm AMAC finds the relevant subset of neighbors an agent can cooperate with and identifies the most useful information to exchange between agents. Here, we also solve a different problem from the routing in packet switching networks.

Some previous works such as RILNET (*ReInforcement Learning NETworking*) [16] integrate Reinforcement Learning (RL) in their approach, aiming at load balancing for data-center networks. The RL algorithm used in RILNET is *Deep Deterministic Policy Gradient* (DDPG) [17], which is a model-free, off-policy actor-critic algorithm that uses deep function approximations that can learn policies in continuous action spaces. The reward used in the agent learning phase is the Maximum Link Utilization (MLU). The used architecture is based on a centralized framework, where the critic and the agent control and know the entire network state. However, these methods do not include communication between agents nor any reduction of overhead in their policy optimization.

Given the success of the centralized training and decentralized execution CTDE paradigm, the value function decomposition method has been proposed to improve agent learning performance. In *Value Decomposition Network* (VDN) [13], the joint Q-value is defined as the sum of the individual agent Q-values. VDN does not make use of additional state information during training and can represent only a limited class of centralized action-value functions. QMIX [12] sheds some light on VDN, and uses a neural network with non-negative weights to estimate the joint Q-value using both individual Q-values and the global state information.

All the aforementioned methods assume no communication between the agents during the execution. However, when the environment is stochastic and the agent observation is partial, the fully decentralized execution is considerably less efficient. As a result, recent studies have shown that introducing communication between agents can significantly improve overall performance in cooperative multi-agent reinforcement learning. Authors of TMC [18] propose a *Temporal Message Control* (TMC) framework that applies a temporal smoothing technique to drastically reduce the amount of information exchanged between agents. In [19], the authors propose *Variance Based Control* (VBC), a simple approach to achieve efficient

communication among agents in MARL, by constraining the variance of the exchanged messages during the training phase. In [10], the authors propose a framework named *Multi-Agent Incentive Communication* (MAIC), which adopts incentive communication to enhance coordination.

In our previously published work [4], a QoS-aware load balancing approach was proposed. It uses a *Deep Reinforcement Learning* (DRL) algorithm with a constrained policy optimization that learns the reward parameters to meet QoS requirements with the main objective of controlling load balancing weights over a set of pre-computed paths for the multiple access devices. We resort to an actor-critic approach that uses CTDE and relies on a *Multi-Agent Deep Deterministic Policy Gradient* (MADDPG) [20] philosophy to realize robust and improved load balancing performance. For cooperation, the MARL framework has been improved by introducing *Differentiable Inter-Agent Learning* (DIAL) [9] mechanism, a technique to pass messages through pre-established communication channels. DIAL passes messages to improve training and execution. While it use useful to have trainable message protocols. However, it does not provide much insight or useful interpretation on how agents cooperatively ought to perform tasks. AMAC shows that applying an attention mechanism and using teammate modeling to enhance coordination without enlarging policy spaces, can significantly improve the performance of multi-agent systems and realize efficient and selective cooperation.

III. Problem Formulation

To model the cooperative multi-agent environment, we use the Decentralized Partially Observable Markov Decision Process (Dec-POMDP) [21] allowing the coordination and decision-making among the multiple agents.

A Dec-POMDP can be defined by a tuple expressed as: $\mathcal{M} = \langle \mathcal{N}, \mathcal{S}, \mathcal{A}, T, R, \Omega, O, \gamma \rangle$, where:

- $\mathcal{N} = \{1, ..., n\}$ is the set of agents,
- $\mathcal{S}$ is a set of global states,
- $\mathcal{A}$ is the set of actions,
- T is a set of transition probabilities between states, $T(s, a, s') = P(s' \mid s, a)$, which defines the probability distribution over possible next states,
- $R(s, a) : S \times \mathcal{A} \rightarrow \mathbb{R}$ is the reward function,
- We consider a *partially observable* setting, where each agent i receives an individual partial observation $o_i \in \Omega$ according to the observation probability function $O(o_i \mid s, a_i)$, where $s \in \mathcal{S}$ is the global state, and a_i is an action made by agent i.
- $\gamma \in$ [0,1) is the discount factor.

In practice, each agent is in charge of load balancing traffic between an origin-destination pair. The states are the amount of traffic (average throughput, number of flows) over each path. The local agent action a_i is about selecting load balancing weights over a set of candidate paths.

At each time step, each agent $i \in \mathcal{N}$ chooses an action $a_i \in \mathcal{A}$, and can only acquire the individual partial observation $o_i \in \Omega$ according to the observation probability function $O(o_i \mid s, a_i)$ where $s \in \mathcal{S}$. The joint action $a = \langle a_1, ..., a_n \rangle$ leads to next state $s' \sim P(s'|s, a)$ and the global reward $R(s, a)$. The formal objective is to find a joint policy $\pi(\tau, a)$, where τ is the joint agent action-observation history, to maximize the joint/global value function: $Q^{\pi}_{tot}(\tau, a) = \mathbb{E}_{s,a}[\sum_{t=0}^{\infty} \gamma^t R(s^t, a^t) \mid s_0 = s^t, a_0 = a^t, \pi]$.

Although training is centralized, execution is decentralized, i.e., the learning algorithm has access to all local action-observation histories $\tau \in \mathcal{T}$ and global state $s \in \mathcal{S}$, but each agent's learnt policy can condition only on its own action-observation history τ_i, where τ_i represents action-observation history of agent i, and $\mathcal{T}$ is the action-observation history of all the agents.

IV. Background

A. QMIX-based Value Decomposition Networks

QMIX [12] is a popular Q-learning algorithm for cooperative MARL in the centralized training and decentralized execution paradigm. QMIX methodology relies on the Value Decomposition Networks (VDN) [13] approach but with a much enhanced class of action-value functions. *Value Decomposition Networks (VDN)* [13] aim to learn an optimal linear value-decomposition by back-propagating the total Q-gradient through deep neural networks representing the individual component value functions. Since the implicit value function learned by each agent depends only on local observations, it is more easily learned. *Value decomposition* consists in decomposing the value function for each agent and measuring their impact on the observed joint reward. The joint action-value function Q_{tot} is (additively) decomposed into N Q-functions for N agents, in which each Q-function only relies on the local state-action history. Each agent observes its local state, obtains the Q-values for its action, and selects an action. The sum of Q-values for the selected actions of all agents then provides the total problem Q-value. Using the shared reward and the total Q-value, the loss is calculated and the gradients are then back propagated into the networks of all agents. *QMIX* [12] aims at overcoming VDN's limitations. QMIX uses a gradient mixing network to estimate joint action-values as a non-linear combination of per-agent values that depend on local observations. Besides, QMIX imposes a monotonicity constraint on the relationship between Q_{tot} and each Q_i, which allows computationally tractable maximization of the joint action-value in off-policy learning. But QMIX performs an implicit mixing of Q_i while regarding the mixing process as a black-box. Besides, when mixing individual Q_is to Q_{tot}, QMIX uses weights directly produced from global features instead of accurately modeling the individual impact to the whole system at a per-agent level.

One of the key benefits of QMIX is the insight that the full factorization of VDN is not necessary to extract decentralized policies fully consistent with their centralized counterpart. Instead, for consistency, it only needs to ensure that a global $argmax$ performed on Q_{tot} leads to the same result as a set of individual $argmax$ operations performed on each Q_i.

B. Attention Mechanism

In recent years, the attention mechanism [11] has been widely used in various research fields and more and more works rely on the idea of the attention to deal with challenges in MARL. An attention function can be described as mapping a query and a set of key-value pairs to an output, where the queries V_Q, the keys V_K^j, the values, and the output are all vectors. In our case, the queries, keys, and values refer to vectors that represent the states, the actions, and the Q-values, respectively. j refers to the index of the head which represents

the attention layers. The output is computed as a weighted sum of values, where the weight w_j assigned to each value is computed by a compatibility function $f(V_Q, V_K^j)$ of the query with the corresponding key. In our case, the attention aims to measure the importance (i.e., weight), given to relevant neighbors for cooperation.

$$w_j = \frac{exp(f(V_Q, V_K^j))}{\sum_k exp(f(V_Q, V_K^k)}, \tag{1}$$

where $f(V_Q, V_K^j)$ is the compatibility function to measure the importance of the corresponding value. The mechanism of scaled, multiplicative, dot-product attention is just a matter of concretely calculating the attentions and re-weighting the values. In practice, multi-head attention is usually employed to allow the model to jointly handle information (queries, keys and values) from different representation sub-spaces at different positions. With a single-head attention, averaging inhibits this joint management process.

V. System model

As mentioned earlier, the proposed smart LB mechanism is based on the MARL paradigm. The applied policy is Centralized Training and Decentralized Execution (CTDE). Our proposed methodology is structured on two main levels, namely (i) ***the controller-level***, where the centralized learning occurs. The controller properly captures the effects of the agents' actions, using a centralized action-value function Q_{tot} (using the mixing network) that depends on the global state s_t and the joint action a_t. When the Q_{tot} function is learned, it offers a way to extract decentralized policies that allow each agent to select only an individual action based on an individual observation with respect to its local Q_i, and (ii) ***the agent-level***, where intelligently scoped communication between only relevant cooperating agents enhances coordination and addresses fully decentralized execution limitations.

AMAC guarantees consistency between the centralized learned policies (derived by the controller based on Q_{tot}) and the decentralized executed policies (where agents select actions according to their Q_i). Fig. 1 summarises the proposed method with the two levels.

A. Controller-level

The purpose of the controller-level is to decide the estimated weights assigned to the candidate routing paths and derive the best load balancing policies. Hence, the output of the controller-level is the estimated weight of each path. The inputs are the agent-level Q-values as described in more details in the ensuing subsection. The controller uses the mixing network of the QMIX to combine the Q-values of each agent in a non-linear fashion into Q_{tot}. To reduce the processing complexity at this level we adopt the dot-product attention [11]. By adding, the dot-product attention, the mixing network mixes the relevant features instead of mixing the global features, this reduces information exchange, and mostly complexity at the controller.

1) ***Multi-head attention****:* In fact, all states, actions, and Q-values are not directly integrated into the mixing network, the Dot-Product transforms them into one weighted value Q^h thereby reducing also the complexity of the mixing network at the controller-level during the training phase.

$$Q^h = \sum_{i=1}^{N} \mu_i \times Q_i(\tau_i, a_i) \tag{2}$$

where μ_i is the head coefficient. For each h, the inner weighted sum operation can be implemented using the differentiable key-value memory model to approximate the coefficients and establish the relations from the individuals to the global. The differentiable key-value memory paradigm provides powerful function approximation capability. We use multiple attention heads to implement the approximations of different orders of partial derivatives. The multi-head attention structure allows the model to jointly attend to information from different representation sub-spaces at different positions. By summing up the head Q-values Q^h from different heads, we get:

$$Q_{tot} = c(s) + \sum_{h=1}^{H} Q^h \tag{3}$$

where H is the number of attention heads, and the first term $c(s)$ in Eq. 3 could be learned by a neural network with the global state s as the input. QMIX [12] uses weights directly produced from global features instead of accurately modeling the agent's individual impact on the whole system at a per-agent level. In our model, Q^h is fed then into the mixing network to produce Q_{tot}. AMAC mixes the weighted Q_i (Q^h) produced from the attention model. In this way, only relevant features are mixed instead of mixing all features.

2) ***Mixing network****:* Our mixing network is a feed-forward neural network that mixes the Q^h monotonically, producing the values of Q_{tot}. The weights of the mixing network are produced by separate hyper-networks. Each hyper-network takes the state s as input and generates the weights of one layer of the mixing network. Each hyper-network consists of a single linear layer, followed by an absolute activation function, to ensure that the mixing network weights are restricted to be non-negative.

Besides, QMIX does not explicitly consider the agent-level impact of individuals on the whole system when transforming individual Q_is into Q_{tot}. In our case, we derive a general formula of Q_{tot} in terms of Q_i, which allows us to implement a multi-head attention formation to approximate Q_{tot}, leading to not only a refined representation of Q_{tot} with an agent-level attention mechanism, but also a tractable maximization algorithm of decentralized policies. When converting individual Q_is to Q_{tot} within a multi-head attention structure, AMAC uses the key-value memory operation to explicitly measure the relevance of each agent to the entire system. Using attention, AMAC measures agents with different weights for mixing Q_is into Q_{tot} according to each agent's individual properties.

B. Agent-level

The agent-level, with relevant communication between agents, sets for each agent the Q-values used then as input to the previously described controller-level. The Q-value is the maximum expected reward an agent can reach by taking a given action from the state. After an agent has learned the Q-value of each state-action pair, the agent at a given state maximizes its expected reward by choosing the action with the highest expected reward. For decentralized decision-making, each agent takes its current obtained observation and last

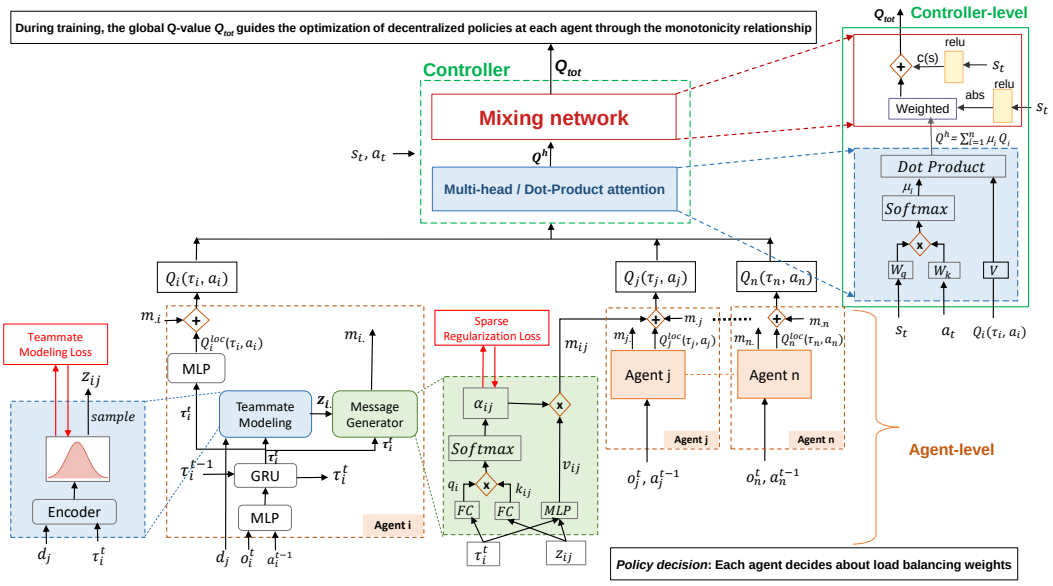

Fig. 1: AMAC structure diagram.

action as input and feeds them into a gating mechanism called *Gated Recurrent Units* (GRU) [22] cell to get a representation of historical information that will help improve the current system decision-making.

We further use this historical representation to generate local Q-values $Q_i^{loc}(\tau_i, a_i)$ via a *Multi-Layer Perceptron* (MLP). The local network of each agent also contains a teammate modeling network (i.e., neighbor relevance assessment), where a teammate is a selected neighbor/retained relevant neighbor.

The agent uses a sampled representation from teammate models to generate sparse communication weights and tailored message contents. The processed messages are then fed to the policies of other agents in an incentive manner, resulting in efficient and sparse/infrequent communication. As depicted in Fig. 1, the agent-level system consists of the following blocks:

1) ***MLP network block****:* A multi-layer perceptron to obtain a local Q-value $Q_i^{loc}(\tau_i, a_i)$ of agent a_i.

2) ***The decentralized teammate modeling block****:* To guide the intentions of certain agents, this block learns targeted teammates models, which can infer the action selection of these agents in a partially observable manner. To obtain a teammate model from agent i to agent j, the system takes as input local information τ_i that represents the action-observation history of agent i, and a specific teammate ID d_j. The teammate model is represented by a multivariate Gaussian distribution whose parameters are computed by an encoder with multiple fully-connected layers. The output of this part is a sampled representation of the teammate z_{ij} for agent j, which will be fed afterward into the message generator block to guide the message generation process of agent i. As the simple MLP cannot guarantee the anticipation of teammate models, an explicit regularization mechanism has been introduced at this block to guide the teammate modeling.

We expect the learned teammate models to be responsive to the action selection of every specific teammate, as the selected actions can intuitively exhibit the coordination relation among agents. We optimize the teammate models by maximizing the *Mutual Information* (MI) between the action a_j taken by agent j and the random variable z_{ij} of the teammate model distribution conditioned on agent i's local action-observation history τ_i (that contains the current local observations o_i^t, and the last action a_i^{t-1}), and the specific teammate ID d_j. We can express MI via the entropy $H(z_{ij}|\tau_i, d_j)$ and the conditional entropy $H(z_{ij}|\tau_i, a_j, d_j)$ as follows:

$$I(z_{ij}, a_j|\tau_i, d_j) = H(z_{ij}|\tau_i, d_j) - H(z_{ij}|\tau_i, a_j, d_j) \quad (4)$$

Maximizing MI between each pair of agents is equivalent to minimizing the uncertainty about the learned teammate model conditioned on the agent's local information, enabling agent i to acquire a powerful teammate representation z_{ij} when modeling agent j.

The agent learns a targeted teammate model for every other agent from its local observation. This teammate model, which maximizes a Mutual Information (MI) regularizer to associate learned models with opponent intention anticipation, can help each agent dynamically generate tailored incentive messages to specific teammates without enlarging policy spaces.

The teammate model is used to design a novel message generator to produce messages with prominent communication weights. The emergence of significant communication weights addresses interaction sparsity, a commonly existing and useful structure in multi-agent systems, realized by pruning/removing messages with minor communication weights.

3) ***The message generator block****:* As the agent can extract targeted teammate information from learned teammate models, this representation can be utilized to generate tailored messages for different agents. Especially, the system maintains a local Q-network to compute the local Q-values $Q_i^{loc}(\tau_i, a_i)$ for each action $a_i \in \mathcal{A}$. The message generator contains an MLP value $v_{ij} = F_m(\tau_i, z_{ij})$, whose input includes the local action-observation history τ_i of agent i as well as the teammate representation z_{ij} for agent j. The output v_{ij} has the same dimension as the local Q-network, which is the dimension of the action space. The teammate representation can also help communicate in a more targeted manner, as unnecessary

information may confuse the receiver. The agent will calculate communication weights for all other agents by taking the teammate representation as input.

At the same time, for a better combination of the representation and the agent's own information, a Dot-Product attention mechanism [11] is applied to compute the query q_i from the agent's action-observation history τ_i and the key k_{ij} from the representation of teammate j. Both q_i and k_{ij} are computed by simple linear functions. Then, by applying a *Softmax* function, the query q_i and the key k_{ij} are normalized and used to provide the communication weight α_{ij} between agents i and j. The communication weight α_{ij}, which defines the weight to agent j for agent i's communication, is normalized to make all α_{ij} from agent i produce an explicit categorical distribution:

$$\alpha_{ij} = \frac{exp(-\sigma \times q_i \times k_{ij})}{\sum_{m \neq i} exp(-\sigma \times q_i \times k_{im})} \tag{5}$$

Where $\sigma \in \mathbb{R}^+$ is the temperature parameter to scale the magnitude of the input. We select the best value for σ (i.e., $\sigma*$) empirically. Equation Eq.5 is related to Eq.1. The final message from agent i to agent j is calculated by the product:

$$m_{ij} = \alpha_{ij} \times v_{ij} \tag{6}$$

As the message is already represented in an effective and targeted way, the final message will be used to favor the other agent's Q-values as an incentive, because this approach would not explicitly enlarge the policy space of every agent.

The Q-value of each agent $Q_i(\tau_i, a_i)$ is then formulated as:

$$Q_i(\tau_i, a_i) = Q_i^{loc}(\tau_i, a_i) + \sum_{j \neq i} m_{ji} \tag{7}$$

where $Q_i(\tau_i, a_i)$ and $Q_i^{loc}(\tau_i, a_i)$ indicate the corresponding Q-value vector for every action a_i.

*4) **Sparse and infrequent communication between agents**:* Despite the fact that we can generate communication weights for different teammates, the neural network itself cannot force agents to produce sparse weights, resulting in uniform communication weights for different agents. In order to learn a sparse but effective communication, we further introduce a sparsity regularization, which optimizes the entropy of the category distribution formed by communication weights:

$$\mathcal{L}_c(\theta_c) = \sum_{j=1}^{n} \mathcal{H}(\alpha_{ij}) = -\sum_{j=1}^{n} \alpha_{ij} \times \log \alpha_{ij} \tag{8}$$

where θ_c are the parameters of the message generator and $\mathcal{H}$ is the Shannon entropy, as defined in [23]. By minimizing this entropy loss, we can obtain communication weights with lower uncertainty. Furthermore, it is also possible to eliminate useless and ineffective communication links and reduce redundant information through the communication weight. The regularization facilitates action taking by adopting a cooperative behavior providing coherent actions in the long-term in order to maximize the final reward.

C. States, actions and rewards

We use 1 agent for each Origin-Destination (OD) pair. Each agent is in charge of selecting paths for incoming flows over a set of pre-computed paths. The path selection policies are derived in our case to optimize the QoE in a video network using HTTP adaptive streaming.

*1) **States**:* The states are the amount of traffic (average throughput) and the number of active flows on each outgoing path (route).

*2) **Actions**:* For the policy decisions, each agent positioned at the source node decides about load balancing weights for the $\mathcal{P}$ paths towards the destination. Each time a new flow arrives, the path is selected accordingly. Due to the discrete nature of RL algorithms, we choose to discretize the decision-making process for the agents: at each time step/period P a new policy is selected and new flows arriving between the current time t and $t + P$ will follow the same policy (i.e. set of load balancing weights).

*3) **Rewards**:* We explore the use of QoS metrics collected in the network as an alternative to QoE-based DRL methods as QoE factors are difficult to retrieve at the network layer (or at the clients) in practice. Equation 9 presents the reward expression used for comparisons in the performance evaluation.

$$\mathcal{R} = Throughput - \lambda \times c(s, a) \tag{9}$$

where $c(s, a) = |Reference_{Delay} - Current_{Delay}(s, a)|$

As shown in Eq. 9, we introduce constrained reward expressions to penalize decisions if the experienced flow delivery delay violates a tolerable bound, called the *Reference Delay* (arbitrarily set to 0.5s in our case). Recall that our goal is to use only network KPIs in the smart load balancing algorithms to intelligently distribute traffic to optimize QoE. The reward expression is well known from previous work [24], and embeds throughput that has a direct impact on the average video bitrate and also latency that impacts stalling. The constrained rewards use parameter λ to penalize actions violating the flow delivery delay limit. While algorithms like RCPO [25] can be implemented to find optimal λ values (i.e. Lagrangian multipliers), in our case, we performed an iterative search for simpler implementation. The next section presents the evaluation methodology and performance assessment results.

VI. Performance Evaluation

A. Simulation environment

The topology used for the performance evaluation is the Abilene [26] topology and for this set of results the clients generate video flows using the LibDASH library [27] in the ns-3 simulator [28] coupled with a PyTorch environment for the machine learning framework where the approaches have been implemented and used to conduct the evaluation.

Our simulation platform, from [4], is based on the Adaptive Multimedia Streaming Simulator Framework (AMust) [29] in ns-3, which implements an HTTP client and server for LibDASH, one of the reference software of ISO/IEC MPEG-DASH standard. As streaming content, we have chosen a representative open movie commonly used for testing video codecs and streaming protocols: Big Buck Bunny (BBB). More specifically, the following simulation scenario have been used for the reported results:

- **Topology**: Abilene [26] (11 nodes, 14 links).
- **Traffic type**: video streams use DASH [27] that adjusts video encoding according to perceived bearer conditions sensed at the application level in the clients.

- **The selected MLP configuration**: consists of 1 hidden layer with 16 neurons, a batch size for experience replay sampling set to 32.
- **Test duration**: 300 time units (seconds).
- **Training duration**: 10 000 steps.
- **Medium and High load scenarios**: for the medium load scenario, the link bandwidth capacities are randomly generated between 5 Mbps and 20 Mbps. For the high load scenario, the link bandwidth capacities are within 1 Mbps and 4 Mbps. We stress the system by setting less total available bandwidth.

We generated 3 paths for each source/destination pair, using a k-maximally disjoint shortest path algorithm. The traffic is randomly generated at the beginning of the experiment following these characteristics:

- The duration of flows is 40s with 10 Origin-Destination (OD) pairs active per simulation.
- The inter-arrival time of flows follows a Poisson distribution with heterogeneous parameters for each origin-destination pair. The global average arrival rate is 1.1 new streams per second.

*1) **Metrics and performance indicators**:* The metrics to assess performance are illustrated in the following paragraph:

- **Delay (s)**: Average end-to-end delay at network layer (constrained).
- **Throughput (Mbps)**: Average throughput of video flows (to maximize).
- **QoE**: A normalized index of the average video bitrate of DASH representations. Depends on screen resolution adaptation by DASH. Average on all downloaded chunks of a normalized quality index indicating to which representation they belong. It evolves between 0 and 1 for r^{min} (min video bitrate of DASH representations) and r^{max} (max representations bitrate), respectively (see [30] for more details) (to maximize).
- **Video Stalling (ms)**: Average duration of freezing times over each video session duration (to minimize).
- **Jitter (s)**: The variation of the video segment retrieval time (the time it takes to download a video segment) (should be kept as close to zero as possible).

TABLE I: AMAC parameters

Learning rate	10^{-2}
Batch size	32
τ delayed network update rate	0.1
γ discount factor	0.9
Number of hidden layers	1
Dimension of hidden layers	16
Policy refreshing period (time period)	$10s$

*2) **Simulation conditions and settings**:* The simulation conditions and hyper-parameters used for performance evaluation of all network load scenarios are specified in Table I. The scenario served for comparison with relevant state-of-the-art solutions with similar hyper-parameters. The policy (load balancing weights decision) is updated every 10s.

B. Training Loss

We analyze the training loss to assess the performance of AMAC by comparing with other MARL based algorithms that address smart load balancing using centralized training and distributed policy execution. Fig. 2 depicts the loss function for the AMAC algorithm that stabilizes after 1000 training episodes and converges to a state leading to a stable output policy for smart load balancing. AMAC is faster than the semi-distributed MADDPG actor-critic-based approach "Semi-Th+D" [4] that uses DIAL [9] for the communication and cooperation across agents. AMAC also outperforms the other semi-distributed algorithms such as DGN [15] and QMIX [12]. Besides confirming that AMAC converges towards stable policies, we will also show that it outperforms all other algorithms in key metrics in the evaluation sections.

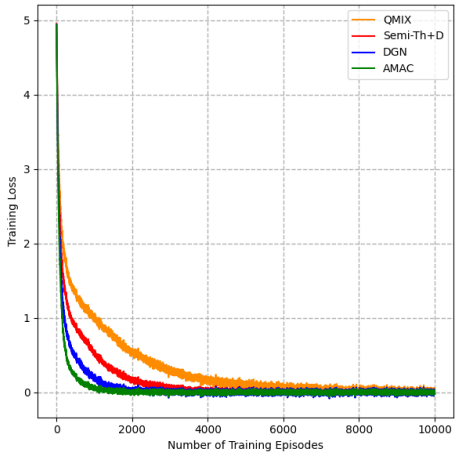

Fig. 2: The loss as a function of training steps.

C. AMAC inter-agent cooperation overhead

AMAC is sufficiently robust to remove useless communications, and by elimination of redundant messages promotes good coordination and reduces overhead. The system searches for and keeps only the optimal weights for all possible communications between two nodes. For a topology with N nodes, in the worst case, if all nodes communicate with each other, we have $N(N-1)$ interconnections corresponding to $O(N^2)$ complexity. Since, the focus on useful and highest weights enables lowering down the number of relevant neighbors. Often the attention matrix and the communication weights of different agent pairs contain many elements with very low entropy for cooperation, only a subset with significant values is relevant and useful for cooperation. By filtering these low values, we reduce the amount of messages exchanged between the agents. In general, the communication load can be evaluated in terms of bandwidth consumption using:

$$\sum_t x_t \times s \times BW[bps] \tag{10}$$

where x_t is the total number of communicating agents at time t $\in$ T where $\alpha_{ij} > \delta$. Variable s is the message size. The threshold δ can be used to disable communications with agents whose attention communication weight α_{ij} (defined in Eq. 5) is below δ, a value that needs to be high enough to contribute usefully to the cooperation and action taken by each agent. This can help filter all agents that do not contribute substantially to the decisions, that is, have little or no influence on the agent's action. This filtering or pruning can help reduce the load on the system due to limited inter-

TABLE II: Medium load scenario

Metric \ Algorithm	ECMP [14]	RILNET [16]	Cen-Th+D [4]	Semi-Th+D [4]	DGN [15]	QMIX [12]	AMAC (δ=0.8)
Delay(s) Reference delay $\leq$ 0.5s	0.359	0.351	0.282	0.278	0.274	0.286	0.261
Throughput (Mbps)	7.03	7.73	9.97	10.21	9.36	9.27	10.46
QoE	0.413	0.442	0.464	0.465	0.463	0.451	0.483
Video Stalling (ms)	711.62	687.76	549.27	520.74	518.13	569.22	415.42
Jitter (s)	0.065	0.051	0.038	0.0375	0.037	0.039	0.019

TABLE III: High load scenario

Metric \ Algorithm	ECMP [14]	RILNET [16]	Cen-Th+D [4]	Semi-Th+D [4]	DGN [15]	QMIX [12]	AMAC (δ=0.8)
Delay(s) Reference delay $\leq$ 0.5s	0.585	0.562	0.507	0.498	0.495	0.504	0.483
Throughput (Mbps)	1.453	1.541	1.788	1.797	1.808	1.798	1.827
QoE	0.267	0.289	0.323	0.325	0.326	0.314	0.342
Video Stalling (ms)	1673.61	1392.24	984.61	976.17	947.12	983.67	886.75
Jitter (s)	0.161	0.145	0.139	0.136	0.115	0.122	0.103

agent communications and cooperation. Note that the α_{ij}'s are normalized and their sum is consequently equal to 1. The threshold $\delta \in [0, 1]$ is known as the message sparsity threshold. We select the best value for δ (i.e., δ*) empirically (i.e., based on observation and experience), but this coefficient can be found formally and mathematically via a Lagrangian or a Bayesian optimization. For the rest of the paper, we kept $\delta = 0.8$. The bandwidth, BW is the amount of bandwidth needed to send 1 bit of message. In practice, one float of a message needs 4 bytes (32 bits) to be stored. There is one exchange of messages between the most relevant neighboring agents every 10 seconds (i.e., policy refreshing period). The message size s measured in bytes is expressed in bits in the expression.

TABLE IV: Inter-agent communication overhead

Algorithms	Overhead (kbps)	AMAC agents exchange gains
AMAC (where $\delta = 0.8$)	4.28	0%
AMAC (where $\delta = 0.7$)	6.41	33.23%
DGN [15]	12.72	66.35%
Semi-Th+D [4] (using DIAL [9])	16.63	74.26%
All-to-all communication	70.4	93.92%

As depicted in Table IV, AMAC produces around an order of magnitude lower communication overhead compared with the all-to-all communication algorithm that does not limit inter agent message transmission (i.e., δ = 0). QMIX [12] is not included in the communication overhead evaluation because agents work independently without direct communication.

D. Performance Evaluation Results

We present results of AMAC, ECMP, RILNET, Constrained Reinforcement Learning (Cen-Th+D and Semi-Th+D from [4]), DGN, and QMIX algorithms in Tables II and III. The tables report the medians of average delay, throughput, QoE, video stalling, and jitter over a test episode of 300 time units (seconds), with a policy refreshing period of 10 seconds (i.e., update frequency of load balancing weights). The comparisons correspond to medium and high load conditions and to scenarios where all links have enough capacity to serve the demand as well as for situations where links are typically congested compared with the load of the end-to-end networking requests. Results depicted in Tables II and III show that the AMAC algorithm achieves best overall trade-off across jointly analyzed performance metrics. Under high load, where links are typically congested, the AMAC algorithm continues to provide good results for all the metrics. Note that AMAC outperforms QMIX and DGN with a QoE of 0.34 versus 0.3 QoE performance and, more significantly, when compared with ECMP, whose QoE is lower with a value of 0.27. Furthermore, AMAC achieves slightly better results in all metrics when compared with throughput-based reward maximization and delay constrained policy optimization algorithms, centralized and semi-distributed (Cen-Th+D and Semi-Th+D, from [4], in Tables II and III). This confirms that AMAC can reduce agent cooperation messages load while not penalizing overall performance. The performance improves slightly by focusing only on key neighbors and neglecting less relevant agents that affect negatively training and convergence to the best policies.

VII. Conclusions

The AMAC algorithm proposed to limit the information exchanged between agents in cooperative MARL approaches for improving load balancing in networks. It is shown not to degrade the performance of cooperative learning. This is achieved by exchanging only information between neighboring agents that usefully contribute to joint learning. Exchanging information between all or too many agents, especially those that do not contribute significantly to joint learning, is not recommended. It is more beneficial to cooperate with agents that contribute most to joint knowledge and learning. AMAC results in the dynamic creation of cooperating agent clusters based on the importance of the communication or cooperation weights (α_{ij}, see Eq. 5) between agents. AMAC outperforms state-of-the-art load balancing techniques and reduces the load induced by cooperation in distributed machine learning algorithms by up to an order of magnitude. This is realized without affecting performance in all key indicators. Further, it outperforms approaches where too many agents cooperate, since a lot of secondary disturbing and spurious information comes into play in the learning process. AMAC confirms that it is essential to cooperate only with the most relevant neighboring agents and focus on identifying them dynamically. To reduce complexity at controller-level, instead of mixing the global features, AMAC mixes only the relevant extracted features for training that also speeds up convergence.

REFERENCES

[1] Ning Wang, Kin Hon Ho, George Pavlou, and Michael Howarth. An overview of routing optimization for internet traffic engineering. *IEEE Communications Surveys & Tutorials*, 10(1):36–56, 2008.

[2] Diego Kreutz, Fernando MV Ramos, Paulo Esteves Verissimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2014.

[3] Nanxi Kang, Monia Ghobadi, John Reumann, Alexander Shraer, and Jennifer Rexford. Efficient traffic splitting on commodity switches. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, CoNEXT '15, New York, NY, USA, 2015. Association for Computing Machinery.

[4] Omar Houidi, Djamal Zeghlache, Victor Perrier, Pham Tran Anh Quang, Nicolas Huin, Jérémie Leguay, and Paolo Medagliani. Constrained Deep Reinforcement Learning for Smart Load Balancing. In *2022 IEEE 19th Annual Consumer Communications & Networking Conference (CCNC)*, pages 207–215. IEEE, 2022.

[5] Sven Gronauer and Klaus Diepold. Multi-agent deep reinforcement learning: a survey. *Artificial Intelligence Review*, 55(2):895–943, 2022.

[6] Frans A Oliehoek, Matthijs TJ Spaan, and Nikos Vlassis. Optimal and approximate Q-value functions for decentralized POMDPs. *Journal of Artificial Intelligence Research*, 32:289–353, 2008.

[7] Landon Kraemer and Bikramjit Banerjee. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82–94, 2016.

[8] Afshin Oroojlooyjadid and Davood Hajinezhad. A review of cooperative multi-agent deep reinforcement learning. *ArXiv*, abs/1908.03963, 2019.

[9] Jakob N. Foerster, Yannis M. Assael, Nando de Freitas, and Shimon Whiteson. Learning to Communicate with Deep Multi-Agent Reinforcement Learning. In *Advances in Neural Information Processing Systems, December 5-10, 2016, Barcelona, Spain*, pages 2137–2145, 2016.

[10] Lei Yuan, Jianhao Wang, Fuxiang Zhang, Chenghe Wang, Zongzhang Zhang, Yang Yu, and Chongjie Zhang. Multi-Agent Incentive Communication via Decentralized Teammate Modeling. 2022.

[11] Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez A N, Kaiser Ł, , and Polosukhin I. Attention is all you need. *NeurIPS'17, pp. 5998-6008*, 2017.

[12] Tabish Rashid, Mikayel Samvelyan, Christian Schröder de Witt, Gregory Farquhar, Jakob N. Foerster, and Shimon Whiteson. QMIX: Monotonic Value Function Factorisation for Deep Multi-Agent Reinforcement Learning. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, volume 80 of *Proceedings of Machine Learning Research*, pages 4292–4301, 2018.

[13] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al. Value-Decomposition Networks For Cooperative Multi-Agent Learning. *arXiv preprint arXiv:1706.05296*, 2017.

[14] Christian Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. Technical report, RFC 2992, November, 2000.

[15] Jiechuan Jiang, Chen Dun, Tiejun Huang, and Zongqing Lu. Graph Convolutional Reinforcement Learning. In *International Conference on Learning Representations*, 2020.

[16] Qinliang Lin, Zhibo Gong, Qiaoling Wang, and Jinlong Li. Rilnet: A reinforcement learning based load balancing approach for datacenter networks. In *MLN*, 2018.

[17] T. Lillicrap, Jonathan J. Hunt, A. Pritzel, N. Heess, T. Erez, Yuval Tassa, D. Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *CoRR*, abs/1509.02971, 2016.

[18] Sai Qian Zhang, Qi Zhang, and Jieyu Lin. Succinct and Robust Multi-Agent Communication With Temporal Message Control. *Advances in Neural Information Processing Systems*, 33:17271–17282, 2020.

[19] Sai Qian Zhang, Qi Zhang, and Jieyu Lin. Efficient Communication in Multi-Agent Reinforcement Learning via Variance Based Control. *Advances in Neural Information Processing Systems*, 32, 2019.

[20] Ryan Lowe, Yi Wu, Aviv Tamar, Jean Harb, Pieter Abbeel, and Igor Mordatch. Multi-Agent Actor-Critic for Mixed Cooperative-Competitive Environments. In *Advances in Neural Information Processing Systems, 4-9 December 2017, Long Beach, CA, USA*, pages 6379–6390, 2017.

[21] Frans A Oliehoek and Christopher Amato. *A concise introduction to decentralized POMDPs*. Springer, 2016.

[22] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar*, pages 1724–1734, 2014.

[23] Claude Elwood Shannon. A mathematical theory of communication. *The Bell system technical journal*, 27(3):379–423, 1948.

[24] Vladislav Vasilev, Jérémie Leguay, Stefano Paris, Lorenzo Maggi, and Mérouane Debbah. Predicting qoe factors with machine learning. In *2018 IEEE International Conference on Communications (ICC)*, pages 1–6. IEEE, 2018.

[25] Chen Tessler, Daniel J. Mankowitz, and Shie Mannor. Reward Constrained Policy Optimization. *CoRR*, abs/1805.11074, 2018.

[26] Anukool Lakhina, Konstantina Papagiannaki, Mark Crovella, Christophe Diot, Eric D Kolaczyk, and Nina Taft. Structural analysis of network traffic flows. In *Proceedings of the joint international conference on Measurement and modeling of computer systems*, pages 61–72, 2004.

[27] Christian Kreuzberger, Daniel Posch, and Hermann Hellwagner. A scalable video coding dataset and toolchain for dynamic adaptive streaming over http. In *Proceedings of the 6th ACM Multimedia Systems Conference*, page 213–218, 2015.

[28] George F Riley and Thomas R Henderson. The ns-3 network simulator. In *Modeling and tools for network simulation*, pages 15–34. Springer, 2010.

[29] C Kreuzberger, D Posch, and H Hellwagner. Amust framework-adaptive multimedia streaming simulation framework for ns-3 and ndnsim, 2016.

[30] Giacomo Calvigioni, Ramon Aparicio-Pardo, Lucile Sassatelli, Jeremie Leguay, Paolo Medagliani, and Stefano Paris. Quality of experience-based routing of video traffic for overlay and ISP networks. In *Proc. of IEEE INFOCOM*, 2018.