POLITECNICO DI TORINO
Repository ISTITUZIONALE

Restoring Application Traffic of Latency-Sensitive Networked Systems using Adversarial Autoencoders

Original
Restoring Application Traffic of Latency-Sensitive Networked Systems using Adversarial Autoencoders / Sacco, A., Esposito, F., Marchetto, G.. - In: IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT. - ISSN 1932-4537. - ELETTRONICO. - 19:3(2022), pp. 2521-2535. [10.1109/TNSM.2022.3192305]

Availability:
This version is available at: 11583/2970939 since: 2022-10-21T09:02:42Z

Publisher:
IEEE

Published
DOI:10.1109/TNSM.2022.3192305

Terms of use:

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright
IEEE postprint/Author's Accepted Manuscript

©2022 IEEE. Personal use of this material is permitted. Permission from IEEE must be obtained for all other uses, in any current or future media, including reprinting/republishing this material for advertising or promotional purposes, creating new collecting works, for resale or lists, or reuse of any copyrighted component of this work in other works.

(Article begins on next page)

10 November 2026

Restoring Application Traffic of Latency-Sensitive Networked Systems using Adversarial Autoencoders

Alessio Sacco *Student Member, IEEE,* Flavio Esposito *Member, IEEE,* Guido Marchetto *Senior Member, IEEE*

***Abstract*—The Internet of Things (IoT), coupled with the edge computing paradigm, is enabling several pervasive networked applications with stringent real-time requirements, such as telemedicine and haptic telecommunications. Recent advances in network virtualization and artificial intelligence are helping solve network latency and capacity problems, learning from several states of the network stack. However, despite such advances, a network architecture able to meet the demands of next-generation networked applications with stringent real-time requirements still has untackled challenges. In this paper, we argue that only using network (or transport) layer information to predict traffic evolution and other network states may be insufficient, and a more holistic approach that considers predictions of application-layer states is needed to repair the inefficiencies of the TCP/IP architecture. Based on this intuition, we present the design and implementation of Reparo. At its core, the design of our solution is based on the detection of a packet loss and its restoration using a Hidden Markov Model (HMM) empowered with adversarial autoencoders. In our evaluation, we considered a telemedicine use case, specifically a telepathology session, in which a microscope is controlled remotely in real-time to assess histological imagery. Our results confirm that the use of adversarial autoencoders enhances the accuracy of the prediction method satisfying our telemedicine application's requirements with a notable improvement in terms of throughput and latency perceived by the user.**

***Index Terms*—Hidden markov model, edge computing, machine learning**

I. Introduction

The ongoing pandemic has highlighted the challenges of building remote and responsive communication systems. One major challenge in implementing such remote interactive services is the design of an architecture that could support very low latency and high reliability, labeled almost a decade ago as "Tactile Internet" [1]. Such paradigm has also led to the definition of the ultra-reliable and low-latency communication (URLLC) services [2], [3], considered to be among the most challenging applications in future networked systems. Their typical use cases include haptic telecommunications, telemedicine, and immersive virtual reality services, to name a few [4], [5]. The networking community proposed the use of several Artificial Intelligence and Machine Learning (AI/ML) techniques, sometimes in combination with network virtualization, to solve some of these problems and provide URLLC services. Some of the most recent advances in AI/ML have favored the design of reactive systems in which states from the TCP/IP network stack are used as input of predictors to foresee future network conditions and react appropriately.

Although we believe those knowledge-defined networking approaches [6] have merit [7]–[10], and indeed they brought significant improvements, URLLC services opened up new challenges, that are exacerbated in networks where time-varying delays and packet losses are frequent. To cope with such conditions, we argue that learning from the TCP/IP stack may be insufficient and a more holistic approach is needed.

Just like error correction codes try to repair the communication at the physical layer, we believe that, in many cases, networked applications can self-repair to improve their performance. The intuition behind such a design principle is based on the notion that predicting network conditions is insufficient to deliver an acceptable level of performance in many applications. Our design is inspired by financial market predictors, that use several indicators to forecast the future price of a stock or the implied volatility of an option contract, and from recent video streaming solutions that learn the optimal video bitrate combining network statistics, e.g., throughput, with application-specific metrics, e.g., buffer occupancy [11]–[13]. So latency-sensitive applications should also predict, when feasible, what the *application-level* data is going to do, not merely what the network data are.

We are not the first to propose application-data prediction, although not for networking applications. For example, researchers have leveraged distributed machine learning models to predict and suggest next actions at the application layer, for next-word prediction, content suggestion, and e-commerce recommendations, offering valid results [14]–[16]. In this paper, we argue for a similar approach in which the network architecture and the application work together to address the following research challenge: *to what extent network information can help application-layer predictions for a more immersive user experience?*

Our Contribution. To solve such research question, in this paper we present *Reparo*, a solution that copes with network suboptimalities by reconstructing, when possible, the content of lost packets at the edge of the network with the help of a novel predictive model. Our system detects when a packet gets lost or is too delayed because of a congestion, and if this is the case, it restores synthetically its content with a new predicted packet generated at the edge. To reproduce the application-layer payload, our model is based on a Hidden Markov Model (HMM), where the traditionally employed Viterbi algorithm [17] is replaced with an Adversarial AutoEn-

This work has been partially supported by NSF Awards 1647084 and 2201536.

Alessio Sacco and Guido Marchetto are with DAUIN, Politecnico di Torino, 10129 Turin, Italy (e-mail: alessio_sacco@polito.it, guido.marchetto@polito.it).

Flavio Esposito is with the Department of Computer Science, Saint Louis University, St. Louis, MO 63103 USA (e-mail: flavio.esposito@slu.edu).

coder (AAE)-based approach [18]. The AAE belongs to the family of Generative Adversarial Network (GAN), a machine learning technique in which two neural networks compete with each other to become more accurate in their predictions. In our context, the adversarial autoencoder is trained to learn the emission probability of the Markov Model and helps the HMM decode the network layer information. Thus, the resulting HMM model keeps track of the communication by monitoring the network state (*observed state*), and, when a packet gets lost or experiences long delays, it predicts the next application-layer payload (*hidden state*). As such, by leveraging this model, we can efficiently map network statistics to application information, enabling statistically and logically similar action to be taken autonomously while the actual action is on its transmission way via the network. The Adversarial AutoEncoder raises the HMM model accuracy to 94%, outperforming other benchmark predictors.

We implemented Reparo on a real medical application: an edge-computing empowered microscope for telepathology remote consultations. Pathologists have the need to operate a microscope remotely for emergency or second consultations, even during surgery, as their opinion can change the course of action of the intervention, e.g., tumor removal surgery. Our results, obtained both in a trace-driven manner and in the real-world scenario, show how an HMM model augmented by an adversarial autoencoder (AAE) is able to obtain a 30% increase in accuracy with respect to a traditional HMM. Besides, Reparo agents can overcome the network loss or delays and provide acceptable reliability of a remotely controlled microscope, and we found that the average latency perceived by the user with *Reparo* is 4 times less than the one attainable without our predictor.

In the rest of the paper, we first review the literature about predictive systems and applications that would benefit from this solution in Section II. Section III depicts the design of our presented solution and overall algorithm. Later, in Section IV we describe the model utilized for the application-level message prediction. We then outlines the implementation details of *Reparo* and experimental results in Section VI. Finally, we conclude the paper in Section VII with some final considerations.

II. Related Work

As tele-operated tasks have received ample attention recently, with an increment of remote critical applications, modeling the corresponding profiles and parameters is becoming more and more important. In this section, we first consider solutions advocating AI/ML to solve this problem; then, we review limitations in current telepathology systems, which refer to our considered use case.

Message and packet prediction. An example of successful use of the ML approach can be found in [19], where delayed or lost feedback information was successfully predicted via Gaussian Mixture Model (GMM) regression problem in a local area network setting. In the context of haptic interaction, the model aims to enable visual feedback, providing the illusion of remotely "touching" something. The same model, GMM, has been used similarly to retrieve a haptic reference trajectory on-the-fly for a peg transfer task [20]. Another common method used in this context is the Hidden Markov Model (HMM), given its ability to model and recognize particular structures [21]. It is often followed by a Gaussian Mixture Regression (GMR) for the online retrieval of a generalized profile. In [22] it has been used to encode a set of force/torque profiles to reproduce the generalized version of force/torque profile for grasping an object in the virtual environment. In the context of mobile apps traffic, [23] proposed the use of HMM to predict the network traffic at both packet and message levels. At the same time, [24], [25] proved a way to reconstruct message-level content on encrypted traffic using Markov models. The great benefit brought by these predictors is their ability to execute predictions in a very short time [26]. However, they come with a high computational cost when the number of Gaussian components increases. Therefore, to meet the latency requirement of URRLC, other improvements are needed. For example, [27] selects a low number of states in order to execute prediction with minimal error in remote robotic surgery applications. In [28] two recurrent neural network (RNN)-based models taught a robot how to draw a line along a ruler exploiting position and force information. The time-consuming prediction, however, indicates that RNN is a bad choice for use as the predictor in remote microscopy. A similar ML predictor is presented in [29] which addresses the problem of viewport prediction (VP) in a networked VR system, i.e., predicting what a viewer is about to consume in the near term. However, none of these predictors has been applied in a real system, neither has been proved to meet critical thresholds in latency and throughput.

Telepathology solutions. One of the interactive services demanding low latency but whose performance is strongly affected by network congestion is telepathology. Current telepathology solutions are limited by the technology, the scale, and the (best-effort) performance of the underlying telecommunication media on which they rely, i.e., the Internet, or, at best, a virtual private network for non-real-time (offline) consultations [30]. These solutions only focus on the transmission of histological images, for asynchronous analysis [31], [32], or on collaborative image viewers [33], but without the ability to control a microscope remotely, and without the ability to cope with suboptimal network conditions. Remotely controlling a microscope (for pathology or other microscopy applications) requires high bandwidth and low delay, along with fast image processing. Nevertheless, these current solutions fail to keep packet losses to a minimum and guarantee a fast and reliable communication. The aforementioned studies on action prediction motivate the need for the telepathology field to receive the same attention, deploying a system able to fetch the content before it is displayed to viewers. When empowered with this feature, our solution can accommodate the network latency and content processing delay.

III. System Design

In this section, we analyze our solution design, highlighting how burdens of interactive communication are shared between

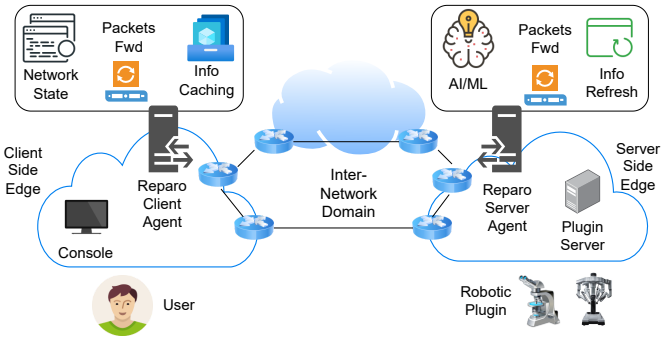


Fig. 1: System Overview. The remote users can operate and use specific resources even if located in a different site. Edge computations are present in both sites, with the aim of speedup message restoration.

both client and server edges. Then, we analyze the functionalities offered by these new components, and finally, we overview the overall algorithm that defines our system.

A. Reparo Overview

Our general system design, whose overview is presented in Figure 1, can help latency-sensitive networked systems to work efficiently. A user establishes a communication, asking for control of a remote resource, in our case the control of the microscope stage or camera operations. These services are hosted on a *Plugin Server* (in the following often referred to as plugin) that is directly connected to the machinery, e.g., microscope or haptic robot. While in this paper, our implementation and evaluation focus on controlling a microscope remotely, our design can also be deployed on other systems whose purpose is to enable interactive communications, for example, commanding a robotic agent, haptic applications, or even virtual reality systems. In the case of holographic-type communications, for example, the user changes position or viewing angle, and the application is supposed to rapidly adapt the streamed contents. Similarly, in our telemedicine application, a pathologist moves the stage mechanically to shift the position of the glass slide containing human histological tissue. Regardless of the use case, we design the solution so that our Plugin Server responds by sending a stream of data that the user can view on a web portal. While viewing the image and the video stream, the user can *interactively* operate over such stream.

Challenges raised by these interactive communications go beyond the ones in traditional video streaming [34]. While conventional video and voice applications naturally allow for graceful degradation of network performance (e.g., adaptive video coding), this context requires handling both input and output of remote machinery for smooth user interaction (e.g., a microscope zoom or pan operation for remote diagnosis), treating user feedback as sensitive application traffic. To address these challenges and speed up such communications, we move the intelligence to the edge of the network by equipping both client and server sites with edge computation for the processing of network and application information. In between sits the network, which we assume is fully or partially out of control.

The edge agents monitor the ongoing communications and provide a fast response to the client and server, assuring a smooth user interaction and guaranteeing an acceptable level of user experience. The *Reparo client agent* is responsible for improving the interaction with the client; as such, it replies to requests coming from the user and provides cached content such as known portions of images. Likewise, the *Reparo server agent* predicts the client requests at the application-level and maintains the server transmission active. These agents can be viewed as proxies that analyze the traffic and take appropriate actions before the occurrence of undesirable behavior, such as network congestion. Due to this functionality, in the following, we often refer to client and server proxies.

During our solution design, we devise an architecture that assures being as minimally invasive as possible. In fact, the client is oblivious of the complexity and keeps sending packets as normal. All the complexity is managed by the edge components. In particular, the server must accept application requests coming from the original client, but implicitly from the client proxy, which intercepts all the packets for/from the user.

B. Reparo Edge Network Components

To productively exploit both network edges, we delegate network processing operations to the two edge proxies present in our system. However, the presence of two nodes acting on the client-server transmission allows providing appropriate services but requires a clear and sound tasks division so that actions are not overlapped.

Our *Reparo client agent* acts as a reverse and caching proxy at the client site, keeps track of the ongoing TCP connections that entails the plugin server. It forwards the packets for the plugin and, if the subsequent response is received, no further operations are demanded. Otherwise, this component sends the most appropriate response to the user: plain positive reply, if the requested action does not impact image or video view, e.g., plugin configuration or setting; or if an update of the image view is necessary, in response to panning or zooming of the sample, it searches if this image portion was cached. If present, the proxy sends this piece of information without contacting the server, so to reduce the response time.

The *Reparo server agent*, being at the server site, is where the actual prediction takes place. In this node resides the HMM boosted model, which outputs the most likely client message request in case the packet is undergoing network congestion. Since the client message in latency-sensitive applications likely contains the action requested, in the following, we often refer to it simply as action. Aside from this ML component, the server proxy includes the application logic in order to adapt and optimize the prediction to the application messages. In fact, two are the main tasks to execute. First, gathering the statistics to improve the HMM model at run-time and providing the final known state on which prediction occurs. Second, restoring the message content that is thought to be lost or very delayed. This situation is spotted by means of a timeout that would help not exceed a maximum application latency. These tasks, i.e., additional network processing and predictive

component, run asynchronously to the communication process, so that packets are then sent to the actual destination without incurring in excessive overhead. We provide further details about the prediction model and our proposed variant in the following (Section IV), after having formalized algorithms dictating the logic in message restoration.

C. Overall Algorithm

Given these considerations, we are now ready to formulate and overview the global algorithm underpinning Reparo in Algorithm 1.

Algorithm 1 Edge proxies logic with packet loss recovery

Let T_s and T_c be the timeout intervals on server and client
Let T_t be the HMM re-training interval
Initialize the HMM and AAE models
Function at the client side
    Set timer T_c for incoming packets *from* plugin
    upon receive packet p **do**
        if $p.dst$=plugin.ip **then**
            $req \leftarrow$ p.req
        if $p.src$=plugin.ip **then**
            If p carried an image, save it
    upon timer T_c expiration **do**
        $sendResponse(req)$
End function
Function at the server side
    Set timer T_s for incoming packets *for* plugin
    upon receive packet p **do**
        if $p.dst$=plugin.ip **then**
            $size \leftarrow$ p.size
        if $p.src$=plugin.ip **then**
            $obs \leftarrow [size$, remaining features of Table I]
            if T_t is elapsed since last model update **then**
                Re-train HMM using last values of obs
    upon timer T_s expiration **do**
        $a \leftarrow predict(obs)$
        Take action based on predicted user action a
        If necessary, $sendClientResponse(a)$
End function

The communication in the server-client direction is mainly characterized by video and image transmission and is thus critical to our interactive system. However, the literature about video streaming is broad and continuously advancing [35], [36], and we decide not to add more ad-hoc mechanisms and leverage the current solutions. For example, to mitigate network congestion, image and video super-resolution is a valuable approach in using ML to recover a high-resolution image from a low-resolution media [37]. Alternatively, many studies suggest deep learning is also effective in predicting future frames that look natural to human eyes [38]. On the other hand, we look with particular interest to the requests from client to server, where our predictive model is applied. Recent studies have shown how the prediction performs very nicely when in this direction [29] (see Section II for a more complete discussion). That said, we can observe the presence of two functions: one running at the client site and one for the server site. They replicate the functionalities aforementioned, where the *Reparo server agent* is the core of our predictive system, and the *Reparo client agent* is fundamental to provide fast feedback.

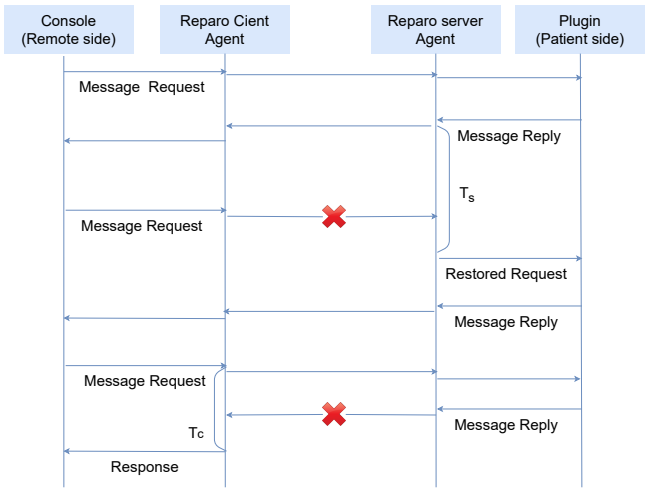


Fig. 2: Signaling and interaction of our Reparo procedure. We highlight the intervention of the two timers T_s and T_c, which acts respectively on the server and client agent.

An important aspect to consider is the presence of three time intervals, which serve to establish the occurrence of relevant events. Specifically, we re-train the HMM periodically on the server proxy, with a time interval named T_t. By doing so, we can improve the model even at run-time by considering fresh data. We then consider one timer for the client proxy, T_c, and one for the server proxy, T_s. They are triggered if no message is received from the server or the client within the defined time interval, respectively. The expiration of T_c indicates the absence of a coming reply, and in response, the appropriate feedback is sent to the waiting user. Similarly, when T_s expires, the server proxy sends the predicted request to the server, since it indicates that the packet is likely to be lost or just very delayed. The subsequent response coming from the server and directed to the client is forwarded as normal. It must be noted that, despite restored messages can be generated either from the client agent or the server agent, we design the solution in order to not overlap the actions and no not deliver duplicate messages to the client console. If the client agent receives a message in response to an event already managed by itself, i.e., the TCP state has already evolved, it discards the message.

We summarize the exchange of messages entailed in the communication between the plugin and the connected user in Figure 2. If no (i) packet losses or (ii) excessive delays occur, the messages exchange follows as normal. Yet, at the occurrence of one of the two events (detected by the server agent by means of a timer T_s), the system activates the restoration process. This allows the plugin to receive the message despite the adversarial network conditions. Similarly, if the response from the server gets lost or delays more than T_c, the Reparo client agent can generate a response containing the

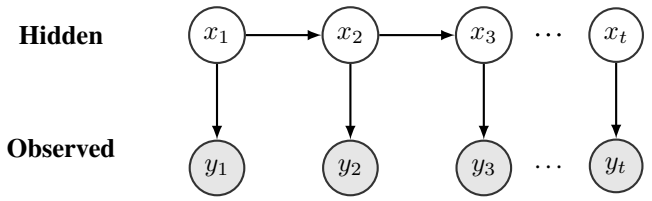


Fig. 3: Schematic view of a general HMM model, composed of temporarily-ordered metrics, in turn categorized as hidden or observed states.

cached image (if a simple image is requested) or an affirmative response (if a plugin operation is requested).

Our insight is that the time required to detect a loss, added to the retransmission time, may lead to intolerable delays. Predicting the application payload would eliminate the retransmission process overhead, demanding only for loss identification. This mechanism is thus crucial in real-time applications. The timeouts are hence crucial to assess when a packet gets lost. For this reason, we set such default values only after a sensitivity analysis presented in Section VI-C. Lastly, specific actions depend on the considered application. As such, more details about these optimizations are thus presented in Section V.

IV. Predictive Model Design

In this section, we briefly describe the model used to predict the transmitted messages. Such a predictor consists of a Hidden Markov Model (HMM) that dictates the way of modeling the environment and an autoencoder used to increase the performance of the HMM. We start by describing the latter component and its parameters; then, we overview the application of the autoencoder in the general HMM model; we finish outlining our model and how it is employed in the prediction.

A. Modeling the System Dynamic with Hidden Markov Model

Because of their ability to capture important traffic statistical properties with a relatively small number of states, HMMs have gained popularity as the traffic model of choice in years states [21], [39]. In these studies, HMM has been applied to model the packet flow either generated by an individual application or that of the aggregate traffic on a single channel. In addition, HMM has been shown to be effective in capturing the dynamic behavior of losses and delays on end-to-end packet channels [40], [41]. Motivated by these analyses and successful results, we model the TCP channel between the client and the plugin by means of the HMM.

In a HMM problem, the time invariant state-space models are typically defined as follows:

$$p(X,Y) = \pi(x_0) \prod_{i=0}^{T} p(y_i \,|\, x_i) \prod_{i=0}^{T-1} p(x_{i+1} \,|\, x_i)$$

where x_i represents the *hidden* variable and y_i is the *observed* variable, $p(x_{i+1} \mid x_i)$ denotes the transition probability which describes the dynamic behavior of the system, and $p(y_i \mid x_i)$ is the emission probability which identifies how the system generates the observation based on the hidden variable. $\pi(x_0)$ constitutes the initial state distribution, that is required to the model to start the process.

The model can be represented as in Figure 3, which explains the relationship between hidden states, observed states, and their probabilities to occur. The evolution of states in HMM is based on the key assumption that the state evolves as a Markov process where the probability distribution of the current state only depends on the state of the previous epoch, i.e., $p(x_i \,|\, x_{i-1}, ..., x_1) = p(x_i \,|\, x_{i-1})$. Despite its simplicity, this first-order Markov process is sufficient for modeling temporal characteristics of the network channel, as outlined in [41], [42].

It is convenient (and hence common) to describe the state evolution over time by means of a transition matrix (TM). Such a matrix is filled with all the transition probabilities $p(x_{i+1} \,|\, x_i), \forall x_i \in \chi$. The probabilities of moving from hidden state to the observed state, $p(y_i \,|\, x_i = j), \forall x_i \in \chi$ are saved in the emission probability matrix. Furthermore, each Hidden Markov Model can also be represented as $\lambda = (A, B, \pi)$, where A refers to the transition probability matrix (TM), B is to the emission probability matrix, and π denotes the vector containing the initial states probabilities.

Generally, these three variables (TM, $\pi(x_0)$, and emission probabilities) are unknown, and they are estimated either using some parametric or data-driven approaches. In particular, three are the main basic problems which need to be solved to characterize the HMMs: *training*, *likelihood*, and *decoding*.

The first problem, *training*, is common to other ML algorithms and is formally defined as the problem of, given the observation sequence in time Y, finding the model $\lambda = (A, B, \pi)$ that maximizes the probability of Y. This problem is crucial for any HMM application because it allows model parameters to be optimally adapted to the training observation sequence, i.e., to learn the best models for real phenomena. This problem is generally solved via expectation-maximization (EM) algorithms, where a particularly successful instance is the *Baum-Welch* algorithm (forward-backward algorithm) [43]. The Baum-Welch algorithm starts by iteratively estimating the initial transition, emission, and state transition probabilities, repeating estimations until the resulting probabilities converge satisfactorily. Being a special case of the EM algorithms, such an algorithm consists of two main steps: the E-step, which computes posteriors over the states, i.e., the probabilities of being at state s at time t, and the M-step, which performs re-estimation of the model parameters in order to maximize the likelihood of posteriors found in the previous E-step.

The second problem, the *likelihood*, attempts to compute the likelihood of a particular observation sequence. More formally, given the observation sequence over time Y and the HMM model $\lambda = (A, B, \pi)$, a solution algorithm should determine the likelihood $P(Y|\lambda)$, i.e., the probability that the observed sequence was produced by the model. This problem can be solved via the recursive forward algorithm, which computes the joint probability of observing the sequence up to time t and the Markov process being in state s_t. The likelihood values

$P(Y|\lambda)$ are then calculated using these joint probabilities.

Thirdly and finally, the *decoding* phase is responsible for finding the hidden state sequence X that is most likely to obtain, given in input the observation sequence over time Y and the HMM model $\lambda = (A, B, \pi)$. The problem is a very common situation and is usually solved by means of the *Viterbi* [17] algorithm for hidden state estimation. This algorithm is a two-step process based on a dynamic programming approach that maximizes the likelihood of the whole generating state sequence. In the first step, it obtains the most likely state s_t at time t through the utilization of a γ_t parameter, while during the second step, the γ parameter is calculated using the forward-backward method. This problem of finding the most likely state sequence can also been summarized as: given a sequence of observed values $(\widetilde{y}_0, \widetilde{y}_1, ..., \widetilde{y}_n)$, the decoding algorithm infers the corresponding hidden variable $\widetilde{x}_t$, i.e.,

$$\widetilde{x}_t \sim p(x_t \,|\, \widetilde{y}_t, ..., \widetilde{y}_0).$$

In this paper we improve traditional HMMs by making use of adversarial autoencoder as an alternative method for the decoding problem. Empirically we have observed how this learner can be helpful and effective in empowering HMM because of its capacity to encode and decode information between different spaces (see results in Section VI).

B. Decoding the Information via Adversarial Autoencoder

Adversarial AutoEncoder (AAE) is based on the idea of blending the autoencoder architecture with the adversarial loss concept introduced by generative adversarial networks (GAN), to reproduce a generative model [18]. An autoencoder model is generally an artificial neural network that learns how to efficiently compress and encode data and then uses this reduced encoded representation to reconstruct a representation that is as close to the original input as possible. AAE simply turns an autoencoder into a generative model by combining the traditional autoencoder with the more general approach of GAN. As in any GAN-based model, the main idea at the basis of AAE is the match of an aggregated posterior of the model (in this case autoencoder) with an arbitrary prior distribution.

In recent years GAN has become one of the most utilized approaches in deep generative modelling, giving rise to a large number of GAN-based models, such as Super-Resolution GAN [44], CycleGAN [45], and BiGAN [46], to cite a few. GAN and its variation models are typically applied over bits of an image, to generate a new synthetic image. Any GAN model is made up of at least two neural networks: a generator and a discriminator. The former accepts an input vector of randomly generated noise and produces an output "imitation" image that looks similar, if not identical, to the authentic image. On the other hand, the latter network attempts to determine if a given output image is "authentic" or "fake". The purpose of GAN is to take a random vector as an input and transform it to follow the pixel distribution of a desired output. The similarity with GAN in AAE is that the autoencoder is trained with dual objectives– a traditional reconstruction error criterion and an adversarial training criterion. At the same time, the presence of a discriminator network and a different training process, make AAE different from traditional autoencoders, i.e., the well-known Variational autoencoder (VAE). While VAE regularizes the latent space with KL-divergence, AAE uses the adversarial training for the regularization.

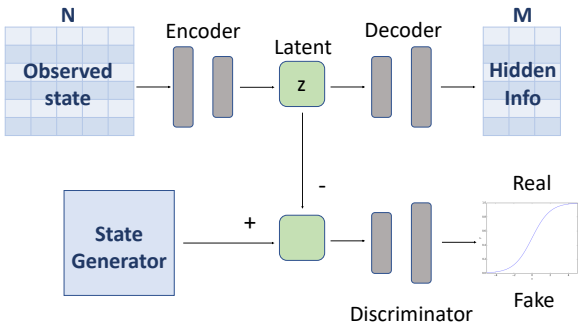


Fig. 4: Autoencoder architecture. In the top row the autoencoder reconstructs the output space (hidden info) from the input data (observed state), using a latent code z. In the bottom row a second network discriminatively predicts whether given latent code is generated by the autoencoder (fake) or a random vector sampled from the normal distribution (real).

Figure 4 summarizes the main components of an AAE model: *(i) the encoder*, a neural network transforming the observed state (input composed of N elements), into a lower dimension, the latent code z; *(ii) the decoder*, a neural network that takes the latent code z and transforms it into the hidden info (output composed of M elements); *(iii) the discriminator*, responsible for checking whether the input is real or not. The encoder learns to convert the data distribution to the prior distribution (of the latent space) after a necessary training period, while the decoder learns the best deep generative model that maps such a prior distribution to the actual data distribution. This matching is guided by the discriminator and generator attached to the latent code vector of the autoencoder (encoder + decoder), creating the adversarial aspect of AAE. The purpose of the autoencoder is also to minimize the reconstruction error. In other words, the encoder, acting as the generator, creates a latent code, z, and tries to fool the discriminator into believing that the latent code is sampled from the chosen distribution. The discriminator, instead, detects if the given latent code is generated by the autoencoder or is sampled from the normal distribution created by the state generator. Once the training procedure is done, the autoencoder produces a minimal error in mapping the inputs to the desired output. In particular, this output represents the missing traffic matrix entries, thus constituting the hidden information.

Although generative models have been frequently applied to generate a synthetic version of audio, images, or video, there is now a recent and mostly unexplored trend of applying these models in diverse domains [47]. In this paper we consider this class of problems for Markovian environments, leveraging the autoencoder to map some collected metrics (observed state) to application-specific values (hidden state), following the HMM modelization.

C. Reparo State Variables

We can now describe how the observed and hidden states are defined in our specific model. In our system, the current evidence y_i is modeled with a vector of metrics at timestamp i; similarly for the corresponding hidden state value x_i. To order the information chronologically, we collect and exploit the timestamp, which is then disregarded during the learning phase since it is worthwhile only in data preparation. Table I outlines the metrics collected from the edge nodes and used as observations in the HMM model.

TABLE I: The communication statistics gathered as observed states of HMM.

y	Features retrieved for each request-response pair
1	Request size [bytes]
2	Ratio request/response size [bytes]
3	Response time [ms]
x	Hidden state observation
1	Request message content

Clearly, the number of observed states y, N, as well as the cardinality of the hidden x, M, are crucial parameters that must be specified when designing the model and that also affect the solution performance. While modeling a single hidden ($M = 1$) is a de-facto standard in HMM [17], there is a tradeoff here in choosing suitable N. Smaller N yields simpler models, but it may be inadequate to capture the space of possible behaviors. On the other hand, a large N leads to a more exhaustive model with more parameters, but may cause overfitting issues and deployability problems. We converged to these metrics in Table I and this cardinality ($N = 3$), as a result of feature importance study and cross-validation to learn this critical parameter. In particular, the three metrics are: *(i)* the size of the application request content sent by the user, *(ii)* the ratio between the size of the request and the corresponding response, and *(iii)* time elapsed between the sending of the request and the reception of corresponding response.

The hidden state x denotes the application-level content of the user request. Predicting this value in the near future means forecasting the most likely operation that the client will send to the remote site. Among the application operations, we also include a "no action" operation that constitutes the case when, at that time interval, the user does not transmit any message. As explained in Section III-C, our prediction occurs periodically in order to guarantee a continuous flow perception. As such, we need to model the possibility of no message received within this interval.

As a best practice for any ML model, before using designated metrics, they must be prepared. Therefore, we also apply data preparation for our metrics, using normalization and standardization techniques to re-scale input and output variables before training the ML model. In fact, differences in the scales across input variables may lead to a problem difficult to being modeled [48]. In addition to this standard normalization approach to scale input values into $0 - 1$ range, we improved this process with the aim of making the model more general and transferable over different networks. Specifically, before such a normalization, the response time is divided by the latency encountered by the *ping* command in the case of an unloaded network. This quantity should represent the minimal RTT and can then be used to scale all values accordingly. This assures a more transferable model that adapts to different environments, even though further actions are required to obtain a global and general model [7], [49]–[51].

Furthermore, as explained in Section III-B, these metrics can be collected at either the client or the server site. The location where these features are extrapolated mainly impacts the "response time", which represents the time elapsed since the request sending and the reception of the response. As shown in Section VI, data pre-processing, in conjunction with the model selected, attenuates such differences originating from the site of collection.

D. Message Predictions via HMM

Taking the above into account, we can now describe the procedure dictating the Reparo behavior when demanded to predict the next event. After having gathered the metrics at time t, if requested by the restoration algorithm (Algorithm 1), it may be required the prediction of the future state at time $t + 1$. We convert this task to the HMM task of computing posterior distribution over the future states given evidence till now. We derive the predicted next event as the future hidden state with the highest probability, according to:

$$\widetilde{x}_{t+1} = \arg\max_{x_{t+1}} p(x_{t+1} \mid \widetilde{y}_t, ..., \widetilde{y}_0). \tag{1}$$

We limit our focus in this paper to the one-step ahead prediction, since we have experienced that predicting for more steps ahead drastically degrades the accuracy, as motivated in Section VI. In conclusion, our prediction procedure is composed of two steps. First, we decode the observations at time t in input into the most likely hidden state using AAE. Then, we forecast the next (one-step ahead) hidden variable using the maximization of posterior probabilities.

V. Reparo Implementation for Remote Microscopy

Even though our system has general applicability, our primary considered use case is telemedicine. In particular, *remote microscopy*. In this section, we define our deployment use case, a telepathology system, the motivation for its responsiveness requirements, and we provide some implementation details.

Use case: telepathology. Pathology is the study of disease and underpins every aspect of patient care. Telepathology, the practice of pathology at a distance, focuses on transmitting images of specimens by a telecommunication linkage to a diagnostic hub where a telephatologist views the images on a video monitor. The transmitted medical images may be used for primary diagnosis, proficiency testing, consultation, quality assurance, and distance learning. A telepathology or more generally a telemedicine session involves the transmission of delay-sensitive and bandwidth-sensitive data that must be

processed and shared with a remote medical doctor. Thus, our system ensures that these requested functionalities are provided.

Why Telepathology? Patient care relies on rapid and accurate diagnosis in the pathology laboratory, which can be made possible thanks to a video-assisted procedure. In critical situations, $e.g.$, timely recognition of bacterial organisms in cerebrospinal fluid, the time to diagnosis and its accuracy can literally be life-saving. Often, these diagnoses are made by a pathologist using a microscope to analyze biological material (cells or tissue) on a glass slide. The speed of the diagnosis depends on the experience of the on-call pathologist, who often covers a broad range of subspecialties. Similarly during surgeries, it is common for surgeons to ask the pathologist for rapid assessment of tissue, e.g., to assess whether a tumor has been completely removed. In the majority of non-trivial cases, pathologists seek second opinions from colleagues by physically transporting privacy-protected [52] glass specimens or asking for an additional opinion of the frozen section area. These consultations must occur rapidly to minimize the patient's time under anesthesia. Critical quality assurance measure is painfully limited, particularly when travel is restricted by special circumstance such as the COVID-19 pandemic, or in rural and underserved community hospitals, as it is dependent upon physical availability of on-site personnel and microscopic review.

Telepathology system implementation details. Inspired by recent studies on real-time telepathology [53], [54], we deployed a real system composed of a Java program that the user can utilize to control the microscope remotely. The microscope is emulated through Micro-Manager [55], which we used to create a modified plugin for providing microscope control and image and frames acquisition. We also modify the original Micro-Manager in a program that, supporting network connectivity, handles data marshaling between the network and the microscope firmware. The client program sends commands for the plugin based on gRPC protocol, demonstrated to be extremely lightweight [56]. The Reparo client and server agents work as a reverse proxy, using the NetfilterQueue library to intercept incoming packets [57]. Clearly, this proxy acts as a reverse proxy for the ingress traffic but as a forward proxy for the egress traffic. These nodes collect all incoming packets, keep track of the ongoing flows, and, by caching static contents, they can also work as accelerators. In our specific case, they are required to reconstruct the TCP status machine so that when a predicted message needs to be sent, the appropriate parameters can be set according to the last TCP state. Besides, often the transmitted data carry sensitive material, so they are able to encrypt/decrypt traffic in order to observe messages. This setting is also very secure, where protected devices, i.e., server proxy, host security functions, releasing the back-end, i.e., plugin, from the burden of implementing SSL logic, which is orthogonal to the application logic. We then opportunely implemented Reparo's logic on these nodes in Python programs, and specifically, the AAE agent on top of Keras [58], while the HMM model is built upon hmmlearn library [59].

Live video streaming management. Since we acknowledge that a fluid video experience is key for such an interaction, our decision is to exploit well-known and studied protocols. Empirically, we experienced how encoding new frames via *ffmpeg* into MPEG-TS data results in an extremely low-latency stream with high frame rates, and acceptable quality and bitrates. Compared to other alternative solutions, such as HTTP Live Streaming (HLS), MPEG-DASH, WebRTC, encoding with *ffmpeg* and decoding in a web page via WebSocket, is tremendously simpler, with reduced overhead and reduced latency. These characteristics are indeed precious and key for live video streaming. For this reason, in the following, we leverage this approach for the live video transfer part, which constitutes the main transmission in the server to client direction. At the same time, we mainly investigate the effects caused by network congestion in the client to server direction, where the predictive model in Section IV is responsible for restoring application messages telepathology-specific.

(Offline) image Viewer. Aside from live services offered, the user can also access the application server and demand an offline image view. In such a case, we leverage the OpenSeadragon JavaScript library [60] to visualize the correct slide in an efficient way, reducing the latency. Since it occurs offline, however, there are no strict requirements of latency, and the different tiles are downloaded while loading the page. Therefore, no further mechanisms (aside from the optimizations offered by the library) are required to improve the user experience. Although the Reparo's focus is mainly for the online interaction, where interaction takes place, this service is fundamental for any telepathology system.

Bounded latency. Another function of potential interest that we implemented was the assurance of a delay bounded within an interval, i.e., both lower and upper bound. This means that our proxy, if desired by the application, also assures that the delay between the request-response pair is always higher than a certain minimum delay. This requirement is often necessary for many contexts [61]. Examples include applications that require fairness, such as for gaming or for trading, and/or whose endpoints do not possess sufficient buffer memory to temporarily store packets that are being received early but require a later playout. The minimum target latency can also serve as an equalizer for cases where traffic may flow across paths of different lengths, limiting jitter differences among packets. For this reason, we also provide this application extension in our telepathology-oriented implementation.

Multiple losses. An extreme case to consider is the reaction to multiple losses. In this case, the server proxy could start predicting and sending packets over the congested network, exacerbating the congestion and resulting in performance degradation. To avoid this case, the server proxy only reacts to the first identified loss and neglects all further timeouts. In such a way, the network is not overloaded, and it avoids predicting many future actions, i.e., for a high time horizon, which is acknowledged to be less accurate [62]. The client proxy, conversely, takes action against these losses on the basis of application requests. When the number of losses becomes too high, a warning message is sent to the user to notify an ineluctable network problem. However, most of the network congestions are transient [63], [64], and client proxy attempts

to mitigate this effect by providing a fast response waiting for the network to be fully available.

Application request: an example. As an example, if the medical doctor changes the snipped area of the histological sample while the network is congested, the client proxy intervenes and sends a cached version of the image of the interested region (ROI), if present. Thus, the client agent is an active part and can help mitigate the network congestion. It must be noted that the pathologist is used to visiting multiple times the same ROI in search of different details, and the proxy can save these tiles once for all future requests. But the request may also refer to a change of the microscope settings or to a pre-process of the sample image. In these cases, after the timeout T_s, the agent will predict this request and send it to the plugin. This node will reply with the proper response. In the case of a configuration request, the response is overlooked and, consequently, not sent, since such a response represents only a positive answer and can be generated by the client proxy. In such a way, we avoid duplication of messages and we achieve an efficient use of network resources. On the other hand, for a request of processing the image, the response is crucial and, consequently, the image is sent. These examples of requests highlight the importance of a proper and synchronized division of concerns, like the one presented in Reparo.

VI. Evaluation Results

To evaluate the performance of our developed system, we implemented Reparo in a real scenario of a telepathology session where network conditions are often emulated to replicate critical environments. After an initial phase where a team of pathologists used the program to learn typical workflow, we also replicated these requests programmatically, without user intervention, to make evaluation repeatable and validate the solution at varying network conditions. In this section, we first present the evaluation of our model and the comparison with other ML models. Then, we expose the benefits that our novel model brings to the application when compared to similar solutions. Lastly, we present the sensitivity analysis that we carried out to select the timeout settings utilized in the experimental campaign.

A. Model Prediction Accuracy

In this first part, we study the efficacy of an ML model to predict the correct messages. We thus experienced different activities of pathologists over this tool and saved the collection of messages exchanged during this interaction user-microscope.

Traffic workload. The prediction accuracy and effectiveness of Reparo severely depend on the traffic patterns. For this reason, we evaluate the behavior of considered solutions in a variety of traffic demands. Specifically, we saved four different workload patterns examined during real telepathology sessions that were undertaken by our pathologists' team. These traces differ for the type of requests, e.g., activation of a video stream, image processing algorithms, and operations sequence. We then combine these collections in a single dataset that consists of more than $40,000$ samples, and we (offline) test the accuracy of predictors on it. We leave the performance assessment for these predictors when applied in real-time with different (partially randomized) patterns to the next subsection.

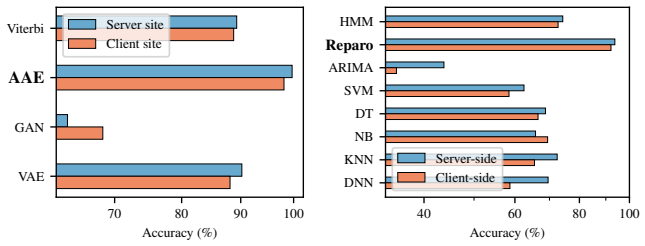
Since the metrics exposed in Section IV-A can be aggregated either at the server edge or client edge, we generated two datasets accounting for these two options, considered as *client-side* and *server-side*. Both scenarios represent the set of packets exchanged during a telepathology session over our system, with the only difference regarding the response time feature. We then separately performed training and testing phases for all the considered algorithms in both scenarios, by splitting the dataset into train (80%) and test (20%) sets. In our application we consider 11 different actions, counting the "no action" option, which must be accounted to represent the case of no packet sent.

Accuracy. First, we evaluate the fidelity of decoding methods, as part of the HMM method. Recalling how this algorithm serves to find the best hidden state (application message) given the observations (network features), we evaluate: *(i)* the traditional algorithm used in HMM, i.e., Viterbi, *(ii)* an alternative Generative adversarial networks (GAN)-based method, referred herein as GAN, whose implementation is inspired by [65], *(iii)* Variational autoencoder (VAE) [66], and *(iv)* our choice, i.e., AAE. We report the results in Figure 5a. For all the graphs in this subsection, the confidence intervals are negligible and do not appear in the graphs since randomicity takes place only in a few algorithms with limited impacts.

It can be observed how AAE is able to reach the maximum accuracy, with a notable 99.7%, which makes it outperforming the default Viterbi of more than 10%. Compared to VAE, we can also conclude how the presence of the adversarial loss similar to GANs makes the model more accurate. In fact, the loss computed on the latent representation of AAE can notably increase accuracy at the cost of a higher training time. The standard GAN model, instead, exhibits a poor ability to encode the observations into hidden states. Obtained results support our choice of replacing Viterbi with AAE as the default decoding algorithm.

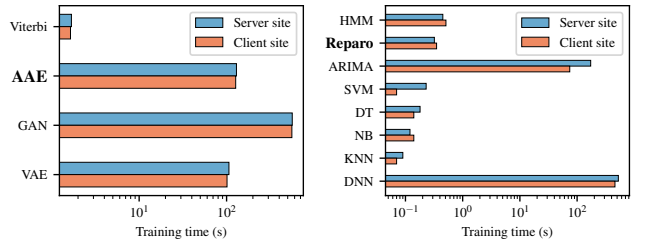
Second, we compare our overall prediction method presented in Section IV-D against other regressors and classifiers. Specifically, in the former class resides Autoregressive Integrated Moving Average (ARIMA), Hidden Markov Model (HMM), whilst in the latter remain Decision-Tree (DT), K-Nearest Neighbors (KNN), Naive-Bayes (NB), Deep Neural Networks (DNN), Support Machine Vectors (SVM). In the case of our HMM model, the next action constitutes the next hidden value; for a regression problem, this value represents the value at the time interval $t+1$ given the last actions; finally, for the classifiers, the action is the output given the past values in inputs. The difference among these models results in how the inputs and outputs are decided and interpreted, which in turn affects the accuracy of the next action prediction.

The results in Figure 5b confirm the hypothesis of the HMM-based algorithm ability to interpret network signals and model the channel. We can notice, indeed, how both our algorithm and HMM outperform the alternatives. Between these two models, however, Reparo raises the accuracy in predicting future values of more than 15%. An improvement of even a few percentage points leads to significant improvements



(a) (b)

Fig. 5: **Accuracy.** (a) Encoding methods performance evaluation. (b) Accuracy of different methods for future payload prediction. Our Reparo system benefits from the autoencoder application and outperforms the benchmarks.



(a) (b)

Fig. 6: **Training time.** (a) Time required for the encoding methods to be trained. (b) Time spent for solving the training problem of predictor algorithms.

in application performance, since it means reduced erroneous usage of network resources (see Section VI-B). Moreover, since the obtained accuracy is close to the (unfeasible) perfect oracle, it results in a trusted predictor that can tremendously enhance the traffic restoration process.

Regarding the location where the model is applied, we can also observe that, even if moderately, running these algorithms at the *server-side* leads to a better outcome. This confirms our design of employing the HMM model on the server proxy.

Training time. We also measure the required time to train these models. This information is precious to assess the cost of an improved prediction and to, subsequently, set the timing logic in our algorithm. We first consider the algorithms addressing the decoding problem, displaying results in Figure 6a. As expected, GAN is the most time-consuming method, with more than 560 seconds, and Viterbi the fastest, with nearly 2 seconds. In between, there are AAE and VAE; the AAE process is slightly longer, but with only 20 seconds of difference. Given the nearly 2 minutes to be trained for our AAE model, we imposed an offline pre-training phase that can also take place over a possible large dataset.

Considering the time to train the predictor model (Figure 6b), the time-series procedure (ARIMA) and the deep-learning model (DNN) need the most considerable amount of time. The remaining algorithms produce a comparable training time, with negligible differences. Notably, for the training problem, our version is also more rapid than the traditional HMM. However, since Reparo's time is less than a second, we opt for an online training based on a rolling window; that is, every period T_t we consider the last obtained data and, on these metrics, perform a re-training process.

Having pre-processed data, the AAE model is application-specific and can be reused for other network scenarios. This motivates our choice of training offline the AAE generator network with a large quantity of data to make it the most accurate as possible. On the other hand, HMM, which models the user experience and the consequent network evolution over time, can be online trained given the shorter training time. Learning data in an online manner, i.e., "on-the-fly", enables continuous model adaptation, and this characteristic is particularly useful for live systems where latency has a significant impact on users.

B. Application Benefits

Besides the validity of the proposed predictor, we evaluate the benefits it can give to a telepathology application. In particular, we deploy some different environments to assess the performance of Reparo in multiple conditions.

Application requirements. Our telepathology system has performance requirements in terms of latency, bandwidth, security, etc., that can be enumerated as follows: *latency* ≤ 100 *ms; throughput* 1 *Mbps; security "High"; reliability "Very High"*, which are in line with the recent studies [67]–[69]. In the following, we evaluate mostly the latency and throughput, since they represent the predominant metrics from the user point of view and the key for assuring real-time control of the microscope.

Benchmark algorithms. To validate our solution, we compare against other two alternative solutions: (*i*) a telepathology application not equipped with any predictor for lost packets, named LiveMicro [33]; (*ii*) a proper adaptation to our use case of the solution proposed in [27], where an HMM/GMR model is used to predict actions in haptic communication, herein referred to as HMM/GMR due to its method.

Application performance. First of all, we consider a local deployment, where the user is located in the same geographical area of the plugin but different access networks. In such a way, we have full control over the two edge networks, but the impact of the network in between, which is however beyond our control, is reduced and allows us to limit the series of unexpected events. We run the telepathology application for 15-minutes and we evaluate the throughput and latency during its execution (Figure 7a). Metrics are collected on the client program and refer to packets sent/received. We can notice how Reparo provides the lowest perceived latency in conjunction with the highest throughput, showing a particularly significant advantage in latency. In fact, as LiveMicro does not utilize any predictive mechanism, it is unable to provide a shorter delay. Not only Reparo has a lower latency on average, but this value is also more stable. This feature is clearly one of the differences from the HMM/GMR solution. This outcome derives from the algorithm that benefits from adopting more periodic actions based on diverse timers, online training, and wise response to multiple consecutive losses.

Congested network: losses. We then consider how the network affects our solution, studying in particular the impact of packet loss rate. We simulate this experience using the

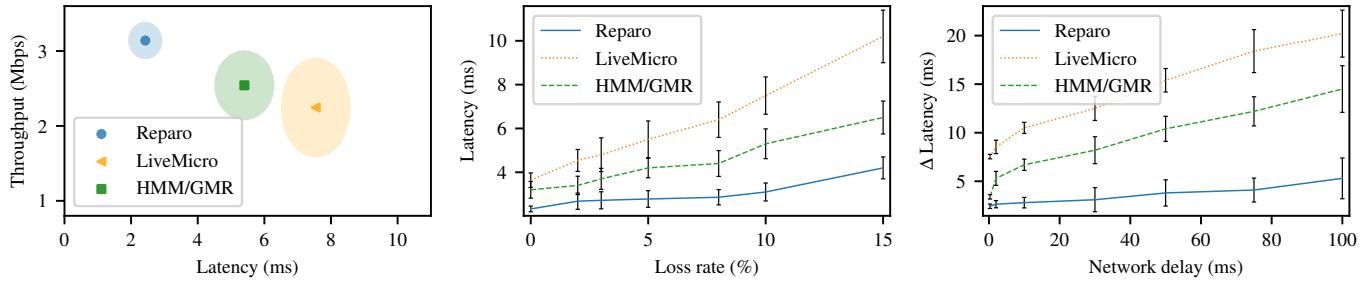


Fig. 7: **Edge Prediction advantages.** (a) Throughput and latency of a session performed locally, in the same campus. Application performance for challenging network conditions: (b) for increasing loss rate in the network, (c) for increasing network delay. The prediction in Reparo enables to drastically shorten the encountered latency. All bars or colored area denote 95% C.I.

Mahimahi emulator [70]. As can be seen in Figure 7b, the latency of the application is bounded even in the case of a high packet loss rate in the network. As the number of lost packets increases, our solution can take an appropriate response to limit the latency to acceptable levels, while also limiting the variability. It must be noted how the network conditions are exacerbated in order to better assess the behavior of a traffic restoration process. For example, a loss rate of 10% is very rare and only occurs in some cellular networks. Nevertheless, we can observe how Reparo, when the loss caused by the network raises, can tackle this lack of messages by restoring them and can provide reliability and continuity in the interactive telepathology session.

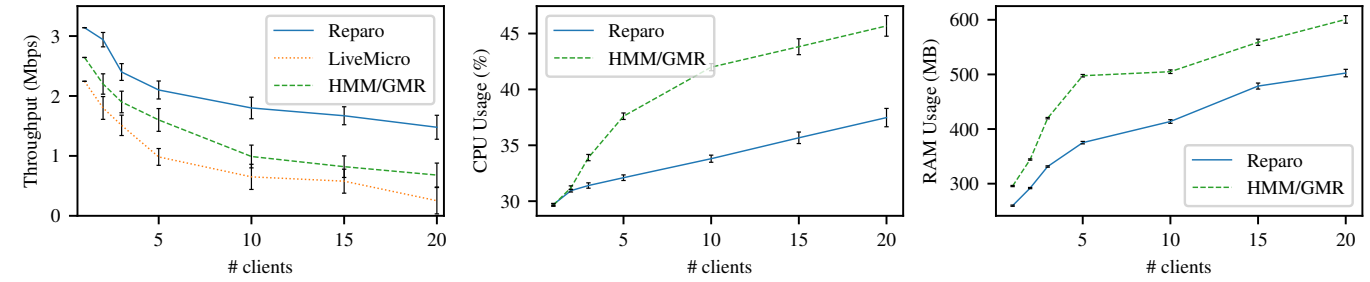
Congested network: delay. Along with the packet loss, we then consider the effects of an increasing network delay, and we compare the resulting perceived latency (Figure 7c). A larger delay may occur either because a microscope is farther away or due to congestion that increases the switching queue delay. In the graph we report the difference between the application latency and network delay, referring to this quantity as $\Delta latency$. Firstly, we can observe the reduced variance in latency for our solution, similar to the previous figure. Secondly, we can notice how Reparo can further reduce the viewed application latency, reacting to the increasing delay and providing a bounded increment in the latency. By predicting actions, the solution can then achieve a reduced latency on average and tolerate extremely delayed packets. Clearly, this result is a consequence of both message predictor and services offered by the two proxies to reduce the number of messages over a wider network. Hence, our solution can drastically help towards a latency that can enable interactivity.

Multiple clients. In Figure 8a we quantify the application throughput when the number of concurrent clients increases. The reported throughput refers to the mean among the users. When more clients are connected simultaneously, our edge components have to keep more models as well, with the result of a greater overhead. However, we can observe how this does not severely affect the behavior of the proxies. On the contrary, Reparo allows to reduce the impact of the inevitable bottleneck and to efficiently handle more ongoing transmissions. HMM/GMR dramatically suffers from the increment of users exhibiting an inability to share resources when they are reduced. As expected, when no mechanism is employed as in LiveMicro, network congestion becomes dominant and degrades the application performance.

Moreover, we measure the amount of RAM and CPU required to run the *Reparo server agent*, which manages all the active connections and where reside our predictive component. To analyze the resource consumption, we execute the agent over an Intel(R) Core(TM) i7-7500U CPU @ 2.70GHz, and limit our focus to solutions restoring message content: we neglect LiveMicro as it only offers the mere stream of data. Considering the amount of CPU required for an increasing number of clients (Figure 8b), we can observe how Reparo reduces the CPU usage. Clearly, as the number of clients increases, so does the CPU. Nevertheless, this consumption is drastically lower than with the HMM/GMR solution. This is due to the implementation of a restoration process only for some events and only when the timer T_s expires.

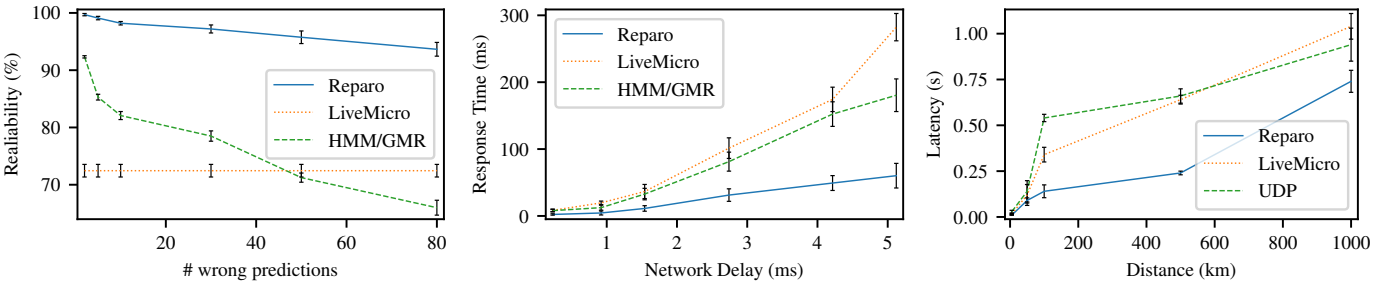
Similar conclusions hold for the use of RAM (Figure 8c), where for more clients, Reparo needs to create more data structures to store the state of the multiple connections. However, these resources utilized are less than the other benchmark HMM/GMR. These results are important to validate how Reparo can manage a high number of concurrent connections without incurring in excessive resource consumption.

Incorrect predictions. Another issue to take into account is how a wrong prediction damages the application and to what extent. To this end, we consider a highly lossy network that can fool the predictor and report the obtained reliability in Figure 9a. We collect results by running the three different solutions in a network with a loss rate = 12%, until the reported number of wrong predictions is obtained, then we stop the execution. In the case of Reparo and HMM/GMR, the reliability refers only to the actual desired application traffic that is correctly received (often referred in the literature as goodput). Since LiveMicro does not forecast, its reliability value is just determined by the packet lost in the network and, thus, it is constant in the graph and is used as the baseline. As shown in the graph, we can note how the more accurate model of Reparo leads to fewer and more sparse errors over time.As mentioned in Section V, our algorithm reduces the number of consecutive predictions, which, if wrong, would harm exponentially. This design choice led the reliability of Reparo more stable and only partially affected by more mistakes. Conversely, HMM/GMR drastically reduce



(a) (b) (c)

Fig. 8: **Multiple clients.** (a) System performance when the more clients access the same microscope and share network resources. (b) CPU and (c) Memory resources consumed on board of the *Reparo server agent*. All bars or colored area denote 95% C.I.



(a) (b) (c)

Fig. 9: **Incorrect predictions**. (a) Impact over the actual communication reliability of wrong predictions. **Wide area communication**. (b) Measuring the application latency for diverse transmission technologies, Reparo can stably lead to lower latency. The comparison also includes an interaction based on UDP. (c) Impact of the physical distance between the user (client) and the microscope (server).

the offered performance when the predictor misinterprets the network signals. Even with a considerable number of wrong predictions (around 80), our system considerably outperforms a prediction-less solution. Moreover, we realize that the worst case occurs when the model predicts a request for a time-consuming action, but in reality the user has not requested any action. However, the high reliability observed in the results, along with the high prediction accuracy, suggests that our solution can well mitigate this circumstance.

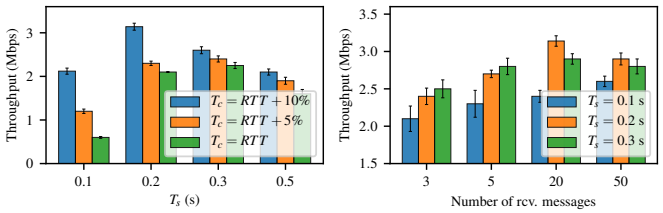
Geographical distance. At the same time, the benefits of Reparo are also valid when the distance between the client and the server increases. Figure 9b displays the average time between the client sent the request and its received response, varying the distance between the client and the microscope from a few meters to a few kilometers. To obtain results, we simply leave the plugin in the same campus, but we reproduce a new client in a new location. The figure reports the network delay, measured through the *ping* tool and indicating the time required to send a (small) packet over the network when it is uncongested. We compare our Reparo against solutions not equipped with a restoration process. From the graph, we can assess that our timer-based algorithm is particularly effective at limiting response time, even when network delay, along with geographic distance, increases.

We then compare the latency attained by the application requests when the server is placed at hundreds of kilometers away (Figure 9c). To further validate our approach, we compare Reparo against a UDP-based protocol to contemplate differences between reliable and unreliable communication. Similar conclusions to the response time hold. Our solution can efficiently reduce transmission latency when the distance is significant. Therefore, the impacts of different and multiple ISP networks are only partially felt when a solution such as ours is deployed, and interactivity is thus preserved.

C. Sensitivity Analysis

Lastly, we discuss our sensitivity analysis, performed to configure timers on the client and server proxies of our system. These values are important as they define when a packet is imagined to struggle in the network congestion storm and can be considered lost. Another timer T_t determines the frequency in re-training the predictor and impacts the amount of fresh information to use.

Firstly, we analyze the best combination of timers running on client and server proxies, respectively T_c and T_s. Rather than focusing on a single value, we study the combination of these timers, since they ensure that the procedures running in the two edge components operate in a synchronized way, and we aim to guarantee the unicity and validity of the actions performed. Figure 10a shows the performance (throughput) at varying the values and relationship between these two values. In particular, we perform tests in a network whose RTT between the user and the plugin is of 0.0274s. We can observe how in this scenario the combination $T_s = 0.2$s



(a) (b)

Fig. 10: **Effects of timers over the application**. (a) Combination of the timer on the client proxy (T_c) and the server proxy (T_s). (b) Relationship between the model train interval, occurring every x received messages, and timer on the server side (T_s).

and $T_c = 0.03$s ($RTT + 10\%$) results in highest application throughput.

Secondly, we consider the interval for re-training the predictor in terms of the number of messages received by the server proxy. We report in Figure 10b the resulting throughput for diverse model update intervals, changing also the T_s timer. We note that the throughput reaches its maximum when the network model is re-trained every 20 messages and T_s is 0.2s. This value also assures the usage of fresh data but without incurring in too frequent updates.

In light of these findings, we set the following default values in our evaluation campaign, $T_s = 0.2$s, $T_c = 0.4$s, $T_t = 20$ msg. However, these values are parameters of the algorithm that can be set by the user in accordance with the application requirements. Since these timers are application-specific and highly depend on the RTT of the network, it is highly recommended that they are set depending on the application and network environment where the solution run.

VII. Conclusion

In this paper, we presented Reparo, an edge computing based system designed for latency-critical applications. To achieve the required low-latency interactive sessions even when a session experiences delays or losses due to a congested path, Reparo predicts the next application-layer message using a Hidden Markov Model (HMM) empowered with adversarial autoencoders. We have implemented Reparo over a telepathology application, providing the ability to control a microscope remotely. Experimental results demonstrate the high accuracy of the model and the validity of our solution in lowering the experienced delay and increasing the throughput and reliability of the application. While the results focus on a telemedicine scenario, we can argue that our message restoration process can be transferred to other haptic programs or in industry 4.0 applications, given the similarity of requirements. To this end, in the future, we plan to transfer the predictive model and apply our approach even in other critical applications to evaluate its potential and assess the benefits brought. We envision our solution to constitute a portion of a possible broader system aiming to guarantee a specific Service Level Objective (SLO), where Reparo can tremendously help mitigate the impact of packet losses.

References

[1] G. P. Fettweis, "The tactile internet: Applications and challenges," *IEEE Vehicular Technology Magazine*, vol. 9, no. 1, pp. 64–70, 2014.

[2] A. Aijaz, M. Dohler, A. H. Aghvami, V. Friderikos, and M. Frodigh, "Realizing the tactile internet: Haptic communications over next generation 5g cellular networks," *IEEE Wireless Communications*, vol. 24, no. 2, pp. 82–89, 2016.

[3] M. Shafi, A. F. Molisch, P. J. Smith, T. Haustein, P. Zhu, P. De Silva, F. Tufvesson, A. Benjebbour, and G. Wunder, "5g: A tutorial overview of standards, trials, challenges, deployment, and practice," *IEEE journal on selected areas in communications*, vol. 35, no. 6, pp. 1201–1221, 2017.

[4] M. Luvisotto, Z. Pang, and D. Dzung, "Ultra high performance wireless control for critical applications: Challenges and directions," *IEEE Transactions on Industrial Informatics*, vol. 13, no. 3, pp. 1448–1459, 2016.

[5] R. Ali, Y. B. Zikria, A. K. Bashir, S. Garg, and H. S. Kim, "Urllc for 5g and beyond: Requirements, enabling incumbent technologies and network intelligence," *IEEE Access*, vol. 9, pp. 67 064–67 095, 2021.

[6] A. Mestres, A. Rodriguez-Natal, J. Carner, P. Barlet-Ros, E. Alarcón, M. Solé, V. Muntés-Mulero, D. Meyer, S. Barkai, M. J. Hibbett *et al.*, "Knowledge-defined networking," *ACM SIGCOMM Computer Communication Review*, vol. 47, no. 3, pp. 2–10, 2017.

[7] R. Boutaba, M. Salahuddin, N. Limam, S. Ayoubi, N. Shahriar, F. Estrada-Solano, and O. Caicedo Rendon, "A comprehensive survey on machine learning for networking: Evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, 05 2018.

[8] A. Sacco, F. Esposito, and G. Marchetto, "Supporting sustainable virtual network mutations with mystique," *IEEE Transactions on Network and Service Management*, vol. 18, no. 3, pp. 2714–2727, 2021.

[9] C. Zhang, P. Patras, and H. Haddadi, "Deep learning in mobile and wireless networking: A survey," *IEEE Communications surveys & tutorials*, vol. 21, no. 3, pp. 2224–2287, 2019.

[10] A. Sacco, M. Flocco, F. Esposito, and G. Marchetto, "Owl: Congestion control with partially invisible networks via reinforcement learning," in *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021.

[11] Z. Akhtar, Y. S. Nam, R. Govindan, S. Rao, J. Chen, E. Katz-Bassett, B. Ribeiro, J. Zhan, and H. Zhang, "Oboe: auto-tuning video abr algorithms to network conditions," in *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '18, 2018, pp. 44–58.

[12] H. Mao, R. Netravali, and M. Alizadeh, "Neural adaptive video streaming with pensieve," in *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, ser. SIGCOMM '17, 2017, pp. 197–210.

[13] F. Y. Yan, H. Ayers, C. Zhu, S. Fouladi, J. Hong, K. Zhang, P. Levis, and K. Winstein, "Learning in situ: a randomized experiment in video streaming," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, 2020, pp. 495–511.

[14] A. Hard, K. Rao, R. Mathews, S. Ramaswamy, F. Beaufays, S. Augenstein, H. Eichner, C. Kiddon, and D. Ramage, "Federated learning for mobile keyboard prediction," *arXiv preprint arXiv:1811.03604*, 2018.

[15] C. Niu, F. Wu, S. Tang, L. Hua, R. Jia, C. Lv, Z. Wu, and G. Chen, "Billion-scale federated learning on mobile clients: A submodel design with tunable privacy," in *Proceedings of the 26th Annual International Conference on Mobile Computing and Networking (MobiCom '20)*, 2020, pp. 1–14.

[16] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving google keyboard query suggestions," *arXiv preprint arXiv:1812.02903*, 2018.

[17] L. R. Rabiner, "A tutorial on hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[18] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey, "Adversarial autoencoders," *arXiv preprint arXiv:1511.05644*, 2015.

[19] A. Chowriappa, R. Wirz, A. R. Ashammagari, and Y. W. Seo, "Prediction from expert demonstrations for safe tele-surgery," *International Journal of Automation and Computing*, vol. 10, no. 6, pp. 487–497, 2013.

[20] C. J. Pérez-del Pulgar, J. Smisek, V. F. Munoz, and A. Schiele, "Using learning from demonstration to generate real-time guidance for haptic shared control," in *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*. IEEE, 2016, pp. 003 205–003 210.

[21] K. Salamatian and S. Vaton, "Hidden markov modeling for network communication channels," *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 1, pp. 92–101, 2001.

[22] A. M. Schmidts, D. Lee, and A. Peer, "Imitation learning of human grasping skills from motion and force data," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2011, pp. 1002–1007.

[23] G. Aceto, G. Bovenzi, D. Ciuonzo, A. Montieri, V. Persico, and A. Pescapé, "Characterization and prediction of mobile-app traffic using markov modeling," *IEEE Transactions on Network and Service Management*, vol. 18, no. 1, pp. 907–925, 2021.

[24] C. Liu, Z. Cao, G. Xiong, G. Gou, S.-M. Yiu, and L. He, "Mampf: Encrypted traffic classification based on multi-attribute markov probability fingerprints," in *2018 IEEE/ACM 26th International Symposium on Quality of Service (IWQoS)*. IEEE, 2018, pp. 1–10.

[25] M. Shen, M. Wei, L. Zhu, and M. Wang, "Classification of encrypted traffic with second-order markov chains and application attribute bigrams," *IEEE Transactions on Information Forensics and Security*, vol. 12, no. 8, pp. 1830–1843, 2017.

[26] S. Calinon, "A tutorial on task-parameterized movement learning and retrieval," *Intelligent service robotics*, vol. 9, no. 1, pp. 1–29, 2016.

[27] F. Boabang, R. Glitho, H. Elbiaze, F. Belqami, and O. Alfandi, "A framework for predicting haptic feedback in needle insertion in 5g remote robotic surgery," in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC)*. IEEE, 2020, pp. 1–6.

[28] T. Adachi, K. Fujimoto, S. Sakaino, and T. Tsuji, "Imitation learning for object manipulation based on position/force information using bilateral control," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 3648–3653.

[29] T. Xu, B. Han, and F. Qian, "Analyzing viewport prediction under different vr interactions," in *Proceedings of the 15th International Conference on Emerging Networking Experiments And Technologies*, ser. CoNEXT '19, 2019, pp. 165–171.

[30] R. S. Weinstein, A. R. Graham *et al.*, "Overview of telepathology, virtual microscopy, and whole slide imaging: prospects for the future," *Human pathology*, vol. 40, no. 8, pp. 1057–1069, 2009.

[31] R. Lebre, R. Jesus, P. Nunes, and C. Costa, "Collaborative framework for a whole-slide image viewer," in *2019 IEEE 32nd International Symposium on Computer-Based Medical Systems (CBMS)*. IEEE, 2019, pp. 221–224.

[32] C. Alvarez, G. Corredor, D. Giraldo, and E. Romero, "Tele-pathology: A use case in colombia," in *2019 IEEE 16th International Symposium on Biomedical Imaging (ISBI 2019)*. IEEE, 2019, pp. 1417–1421.

[33] A. Sacco, F. Esposito, P. Okorie, and G. Marchetto, "Livemicro: An edge computing system for collaborative telepathology," in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Edge Computing*, ser. HotEdge '19, 2019.

[34] C. Han, Y. Wu, Z. Chen *et al.*, "Network 2030 a blueprint of technology, applications and market drivers towards the year 2030 and beyond," 2018.

[35] K. Du, A. Pervaiz, X. Yuan, A. Chowdhery, Q. Zhang, H. Hoffmann, and J. Jiang, "Server-driven video streaming for deep learning inference," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, ser. SIGCOMM '20, 2020, pp. 557–570.

[36] J. Kim, Y. Jung, H. Yeo, J. Ye, and D. Han, "Neural-enhanced live streaming: Improving live video ingest via online learning," in *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, ser. SIGCOMM '20, 2020, pp. 107–125.

[37] H. Yeo, S. Do, and D. Han, "How will deep learning change internet video delivery?" in *Proceedings of the 16th ACM Workshop on Hot Topics in Networks (HotNets '17)*, 2017, pp. 57–64.

[38] W. Lotter, G. Kreiman, and D. Cox, "Deep predictive coding networks for video prediction and unsupervised learning," *arXiv preprint arXiv:1605.08104*, 2016.

[39] Y. Sun, X. Yin, J. Jiang, V. Sekar, F. Lin, N. Wang, T. Liu, and B. Sinopoli, "Cs2p: Improving video bitrate selection and adaptation with data-driven throughput prediction," in *Proceedings of the 2016 ACM SIGCOMM Conference*, ser. SIGCOMM '16, 2016, pp. 272–285.

[40] P. S. Rossi, G. Romano, F. Palmieri, and G. Iannello, "A hidden markov model for internet channels," in *Proceedings of the 3rd IEEE International Symposium on Signal Processing and Information Technology (IEEE Cat. No. 03EX795)*. IEEE, 2003, pp. 50–53.

[41] J. Liu, I. Matta, and M. Crovella, "End-to-end inference of loss nature in a hybrid wired/wireless environment," in *Proceedings of WiOpt'03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2003.

[42] S. Tao and R. Guérin, "On-line estimation of internet path performance: an application perspective," in *IEEE INFOCOM 2004-IEEE Conference on Computer Communications*, vol. 3. IEEE, 2004, pp. 1774–1785.

[43] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[44] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang *et al.*, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 4681–4690.

[45] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2223–2232.

[46] J. Donahue, P. Krähenbühl, and T. Darrell, "Adversarial feature learning," *arXiv preprint arXiv:1605.09782*, 2016.

[47] K.-Y. Chen, C.-P. Tsai, D.-R. Liu, H.-Y. Lee, and L.-s. Lee, "Completely unsupervised phoneme recognition by a generative adversarial network harmonized with iteratively refined hidden markov models," in *INTERSPEECH 2019 - Annual Conference of the International Speech Communication Association*, 2019, pp. 1856–1860.

[48] C. M. Bishop *et al.*, *Neural networks for pattern recognition*. Oxford university press, 1995.

[49] Z. Allen-Zhu, Y. Li, and Y. Liang, "Learning and generalization in overparameterized neural networks, going beyond two layers," in *Advances in neural information processing systems (NeurIPS 2019)*, 2019, pp. 6158–6169.

[50] N. Jay, N. Rotman, B. Godfrey, M. Schapira, and A. Tamar, "A deep reinforcement learning perspective on internet congestion control," in *International Conference on Machine Learning (ICML 2019)*. PMLR, 2019, pp. 3050–3059.

[51] H. Wang, S. Zheng, C. Xiong, and R. Socher, "On the generalization gap in reparameterizable reinforcement learning," *arXiv preprint arXiv:1905.12654*, 2019.

[52] Health Insurance Portability and Accountability Act of 1996 (HIPAA), 2021, http://www.hhs.gov/hipaa/.

[53] D. U. Ekong and P. Fontelo, "Prototype telepathology solutions that use the raspberry pi and mobile devices," in *2017 IEEE Global Humanitarian Technology Conference (GHTC)*. IEEE, 2017, pp. 1–4.

[54] A. Sacco, F. Esposito, G. Marchetto, G. Kolar, and K. Schwetye, "On edge computing for remote pathology consultations and computations," *IEEE Journal of Biomedical and Health Informatics*, vol. 24, no. 9, pp. 2523–2534, 2020.

[55] Micro-Manager library, https://micro-manager.org/.

[56] gRPC, A high performance, open-source universal RPC framework., 2021, https://grpc.io/docs/.

[57] NetfilterQueue library, https://pypi.org/project/NetfilterQueue/.

[58] A. Gulli and S. Pal, *Deep learning with Keras*. Packt Publishing Ltd, 2017.

[59] HMM Learn library, 2021, https://github.com/hmmlearn/hmmlearn/.

[60] OpenSeadragon, http://openseadragon.github.io/.

[61] A. Clemm and T. Eckert, "High-precision latency forwarding over packet-programmable networks," in *NOMS 2020-2020 IEEE/IFIP Network Operations and Management Symposium*. IEEE, 2020, pp. 1–8.

[62] G. P. Zhang, "Time series forecasting using a hybrid arima and neural network model," *Neurocomputing*, vol. 50, pp. 159–175, 2003.

[63] A. P. C. da Silva, M. Varela, E. d. S. e Silva, R. M. Leao, and G. Rubino, "Quality assessment of interactive voice applications," *Computer Networks*, vol. 52, no. 6, pp. 1179–1192, 2008.

[64] J.-c. Bolot and H. Crépin, "Analysis and control of audio packet loss over packet-switched networks," in *IEEE Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV)*. Citeseer, 1993.

[65] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros, "Image-to-image translation with conditional adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 1125–1134.

[66] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," *arXiv preprint arXiv:1312.6114*, 2013.

[67] Q. Zhang, F. Liu, and C. Zeng, "Adaptive interference-aware vnf placement for service-customized 5g network slices," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 2449–2457.

[68] A. Sacco, F. Esposito, and G. Marchetto, "Rope: An architecture for adaptive data-driven routing prediction at the edge," *IEEE Transactions on Network and Service Management*, vol. 17, no. 2, pp. 986–999, 2020.
[69] N. Alliance, "5g white paper," *Next generation mobile networks, white paper*, vol. 1, 2015.
[70] R. Netravali, A. Sivaraman, S. Das, A. Goyal, K. Winstein, J. Mickens, and H. Balakrishnan, "Mahimahi: Accurate record-and-replay for http," in *2015 USENIX Annual Technical Conference (USENIX ATC 15)*, 2015, pp. 417–429.

Alessio Sacco received the received the M.Sc. degree (summa cum laude) and the Ph.D. degree (summa cum laude) in computer engineering from the Politecnico di Torino, Torino, Italy, in 2018 and 2022, respectively, His research interests include architecture and protocols for network management; implementation and design of cloud computing applications; algorithms and protocols for service-based architecture, such as Software Defined Networks (SDN), used in conjunction with Machine Learning algorithms.

Flavio Esposito is an Assistant Professor with the Department of Computer Science at Saint Louis University (SLU). He also has an affiliation with the Parks College of Engineering at SLU. He received an M.Sc. degree in Telecommunication Engineering from the University of Florence, Italy, and a Ph.D. in computer science from Boston University in 2013. Flavio worked in the industry for a few years, and his main research interests include network management, network virtualization, and distributed systems. Flavio is the recipient of several awards, including four National Science Foundation awards and two best paper awards, one at IEEE NetSoft 2017 and one at IEEE NFV-SDN 2019.

Guido Marchetto (M'06-SM'21) received the Ph.D. degree in computer engineering from the Politecnico di Torino, in 2008, where he is currently an Associate Professor with the Department of Control and Computer Engineering. In 2009, he visited the Department of Computer Science at Boston University. His research topics cover distributed systems and formal verification of systems and protocols. His interests also include network protocols and network architectures. He is Senior Member of the IEEE and he serves as an Associate Editor of the IEEE Transactions on Vehicular Technology.