Journal of Network and Systems Management (2023) 31:64
https://doi.org/10.1007/s10922-023-09755-y

RiskNet: Neural Risk Assessment in Networks of Unreliable Resources

Krzysztof Rusek[1] · Piotr Boryło[1] · Piotr Jaglarz[1] · Fabien Geyer[2] · Albert Cabellos[3] · Piotr Chołda[1]

Received: 13 March 2023 / Revised: 21 June 2023 / Accepted: 22 June 2023 /
Published online: 15 July 2023

Abstract
We propose a graph neural network (GNN)-based method to predict the distribution of penalties induced by outages in communication networks, where connections are protected by resources shared between working and backup paths. The GNN-based algorithm is trained only with random graphs generated on the basis of the Barabási–Albert model. However, the results obtained show that we can accurately model the penalties in a wide range of existing topologies. We show that GNNs eliminate the need to simulate complex outage scenarios for the network topologies under study—in practice, the entire time of path placement evaluation based on the prediction is no longer than 4 ms on modern hardware. In this way, we gain up to 12 000 times in speed improvement compared to calculations based on simulations.

Keywords Graph neural networks (GNNs) · Message-passing neural networks (MPNN) · Resilience management · Risk engineering · Shared protection · Value-at-risk (*VaR*)

1 Introduction

Applications of machine learning (ML) in the area of communication networks[1] (IP/MPLS-based, optical transport, and the like) focus mainly on enabling data-driven self-management. The aim is to enable a network infrastructure to react intelligently in the presence of random and adversary events. Here, we elaborate on the reliability/resilience aspects. We follow the path of ML-supported management and contribute with the concept of *resilience-aware network design*, where randomness relates to failures and consecutive recovery events. Here, we elaborate on

[1] To avoid equivocation, we will denote 'communication network' as the 'topology'. The word 'network' will be limited to 'neural network' only.

Extended author information available on the last page of the article

provisioning an efficient method to predict the quality of resilience in presence od recovery settings. We deal with the latter only in the most complex cases involving sharing of backup resources, that is, the ones for which traditional reliability modeling is extremely problematic.

In communication networks, it is generally assumed that the resilience to outages is based on the so-called *protection*, where each connection uses a precalculated pair of working and backup paths. The former is used before the outage and the latter serves afterward to bypass faulty components (routers, cross-connects, links, etc.). Resilience provisioning can be based on dedicated protection, where backup resources are designated to be used in the case of faults on working paths, and there is no danger of their shortage. However, this method is extremely costly in terms of infrastructure usage. Furthermore, the prediction of its behavior is easy to model (we can assume the independence between connections). In contrast, a practical method, although complex in modeling, is based on the so-called *shared backup path protection* (SBPP) [1]. The notion of 'sharing' consists in having the same backup resource pool for various connections. The usage of this pool makes the connections dependent and hinders exact modeling. It is worth noting that recently shared protection is also used as a design option in the context of computing systems [2].

Sharing of backup resources can incur penalties imposed on a network operator due to outages. They happen when a backup path cannot be established for a connection in the presence of resource shortage. Typically, a penalty is a monetary value proportional to the outage time experienced by a user due to a given service level agreement (SLA). From a business point of view, this is a random expense that should be included in *risk management process* during the so-called risk assessment [3].

In this paper, a RiskNet model is provided, where penalties are treated as the main performance metric to assess the quality of resilience and reflect its financial aspect. Regarding the taxonomy of approaches aimed at assessing reliability-related parameters given in [4], we apply a *data-driven intelligent-based prognosis method*. The classical approach to the evaluation of shared protection resiliency (such as the one presented in [5–7]) first assumes the quantification of reliability and, second, applies a direct prediction model. Both are not attractive to use since we would like to quantify the business aspect of resilience (so we use risk-related metrics). In addition, a prognosis with the direct analytical model is based on strong assumptions that are not always valid. These doubtful assumptions involve in many cases: first, limited distributions to provide effective model (e.g., memoryless distributions of time to failure, etc.), and second—topologies (e.g., ring or some regular ones). Although these unrealistic assumptions are taken, the modeling is still very complex. We would like to overcome these limitations by providing a general ML-based model. Additionally, the modeling used by us takes into account probabilistic estimates (similar to the traditionally used availability, mean downtime, etc.), but also considers the amplitude of the outage impact in order to use an indicator inspired by *risk engineering*. Since SBPP has no known analytical results for those indicators (e.g., expected total downtime per year), a solution is to use time-consuming simulations. Due to the rare nature of failures, the simulation must be very long, as according to the best authors' knowledge, there is no rare event simulation technique addressing

SBPP. Therefore, the main contribution of our paper is the method of risk estimation modeled as a regression problem defined on a bipartite metagraph supported by a graph neural network (GNN). The GNN maps the reliability parameters of the network's components (e.g. links) to the parameters of a distribution family. This distribution is a universal tool for determining various performance characteristics. By 'universal', we mean that a single instance of the trained GNN can be effectively used with various network topologies, no matter what kind of topologies were used during the training process. In this way, a high level of generalization is obtained, increasing the applicability potential of the proposed approach. We have also limited the need for long simulations only to the training phase conducted only once before deployment. This way, after the model is trained even with a given type of topology, we can use it as an off-the-shelf solution for topologies of different kinds. Moreover, the provided inference is extremely fast. The GNN outputs the entire distribution, and we are not restricted to any particular risk measure—this is left to the decision of the risk analyst. In the paper, we use *value-at-risk* (*VaR*), a popular risk measure, to illustrate the usefulness of the approach. The background on GNNs that is necessary to understand the presented concepts is presented in [8].

The next section presents the literature review to show the background and emphasize the originality of our approach. Section 3 theoretically elaborates on our GNN-based approach to the prediction of penalty levels. Next, we devote Sect. 4 for illustration of the whole experimental setup and for the presentation and discussion of the results. It proves that our approach meets the practical requirements (speed of work, very high accuracy of prediction). We conclude the paper and present the ideas for the extension of the given concept in Sect. 5.

2 Rationale and Related Work

The rationale for this work comes from the field of communication and computer networks, where data transmission must be protected against the potential impact of component failures. An operator that cannot provide reliable transmission to its customers must pay fines according to particular service level agreements. The fee incurred in relation to the impact of a failure is calculated according to the so-called *compensation policy*. Usually, it is based on the total downtime averaged over a period such as a year. Other possible compensation policies are based on the total number of failures experienced or the sum of squares of downtimes, etc. [9]. This relationship between physical time and monetary units connects traditional reliability analysis with business-oriented risk analysis.

The research presented lies at the intersection of reliability analysis, risk management, and machine learning. The provision of network resilience has been well studied [10]. However, less emphasis has been placed on approaches aligned with the business perspective. Here, we assume risk-based quantification, where not only the *probability* of outages, but also their *impact* is directly taken into account. In this way, we are able to characterize resilience in a more informative way than using classical resilience measures (e.g., reliability or availability functions). In the communications sector, risk has been associated with quantification of deviations from

the desired operational quality levels [11]. From a mathematical point of view, the penalty is expressed on the basis of risk theory dealing with extreme events. Generally, the most important aspect here is to estimate the whole distribution of the total penalty. Then, it is possible to quantify the penalty level with, for instance, the 95th percentile, known as value-at-risk ($VaR_{5\%}$), a measure with a well-justified tradition for communication networks [12]. However, a more robust measure is frequently used, called 'conditional *VaR*' ($CVaR_{5\%}$). In this case, the average of all penalties beyond the assumed percentile is calculated. An important advantage of *CVaR* is its property of *subadditivity*. It allows treating the penalties for individual connections as independent. Then, it is reasonable to sum the penalties for individual connections, since this sum is the upper bound for the total penalty in the network. Therefore, we can use the pessimistic approximation [13]. However, the common requirement for risk measures is the penalty distribution from which *VaR*-related measures can be derived. In the networking content, this distribution depends on the reliability parameters of network components (i.e. communication network nodes, such as routers or links).

Here, we do not follow the path in which reliability-related parameters are modeled directly. For example, in [5–7, 14] this approach is used for the prediction of availability or reliability functions in communication networks, while in [15, 16] the same paradigm applies to risk-related measures in various engineering systems. We appreciate these avenues; however, we seek a more universal model that is easier to change with the new data and generalizes well without using problematic assumptions taken when direct modeling is applied. In the modeling of up- and down-times, the typical approach is to assume that failures arise due to a homogeneous Poisson process [17]: the times between failures are exponentially distributed. It is statistically valid for many cases in communication networks [18], but has appeared to be limited since other distributions for times between consecutive failures have also been reported (e.g., Weibull distribution [19]). The modeling of outage times is even more controversial. Although the simplest approach also uses exponential times, these times in real networks appear to be log-normal [20] or Pareto-like [21]. In our work, we assume Student's *t*-distribution as more general (for more details, see Subsec. 3.2). We deal with a realistic case of SBPP, where up- and down-times do not follow simplistic assumptions of exponentiality. Due to the limitations of direct modeling approaches, we decided to base our analysis on graph neural networks (GNNs) [22–24]. The additional advantage of this approach is that it allows us to build a solution independent of a network topology and any particular routing scheme. By 'routing' we mean here any function mapping every SLA (connection) to a pair of working/backup paths, i.e. sets of components (links). Previously, machine learning (ML) algorithms have also been used in the context of risk management. A general framework for risk assessment using ML was proposed in [25] and validated in a drive-off scenario involving an Oil & Gas drilling rig. The authors provided a comprehensive analysis and indicated several limitations of deep neural networks (DNNs) in terms of risk assessment. Furthermore, DNNs were also used successfully for risk management in the customs [26] or financial market [27]. GNNs have already been used for computer networks, for example, by Geyer et al. [28]. The fundamental difference when comparing our work with this paper is that

we consider routing paths as input to the GNN-based algorithm, while [28] aims to find routing paths according to the specified policies. Additionally, Geyer et al. do not address the resilience aspect in the sense considered in our work (i.e., limitations in bandwidth of backup resources in SBPP). The other papers of the same research group (e.g., [29]) also differ significantly from our work: while they focus on congestion analysis related to traditionally assessed delays, we put emphasis on a topology-agnostic solution for the resilience of the data plane based on the business-oriented approach. The additional difference lies in the network representation. The other papers consider a network of queues, rather than the bipartite graph of services and resources used in this paper. The authors of [30] also focus on network performance by using GNN to extract features. They are used to calculate the routing that maximizes bandwidth utilization. Resilience is also indirectly addressed, as the proposed solution is capable of dealing with router and link failures. Despite some similarities, our approach to resilience is more business-oriented (due to adoption of the risk approach) and not bounded by any particular method of generating working and backup paths.

Recently, GNNs have also been used to control the process of network recovery, for example, in [31]. However, here we focus on prediction of the related parameters.

3 Methods

RiskNet combines probabilistic machine learning modeling with graph neural networks. In particular, we model the cost distribution for the k-th SLA as some distribution D, parametrized by the output of GNN. For simplicity, we apply a mean-field approximation and assume that penalties are independent. We are allowed to do that without losing the quality of the prediction, since we use $CVaR$ to quantify the risk and this measure is sub-additive.

We want to be explicit about the two parts of deploying the RiskNet. The first one is the training phase when the model is initialized with random parameters, and its prediction gets improved during the training by observing results form the simulator.

Once the training is complete, we are in the inference phase, the model is used for making predictions for new networks and configurations newe seen in the training. This phase is extremely fast as it only involves simple mathematical functions and linear algebra functions—the parameters are restored from training checkpoint.

To simplify, we can say that we use a type of supervised learning since we are able to provide the real values (ground truth) of the penalties and confront them with the predicted values to train the model. Real values are provided with simulations for specific configurations. On the other hand, simulations take a lot of time, that is why we can afford for them only in the training phase. During the operation of the entire system, a very fast GNN model predicts the values to be used for a communication network topology not provided during the training phase. Note that the input to the model is exactly of the same type.

The training phase is visualized in Fig. 1. First, we select random network topologies (training samples). They fed the GNN and a discrete-event simulator. Second, the message-passing is performed in GNN to obtain the convergence (1) and provide

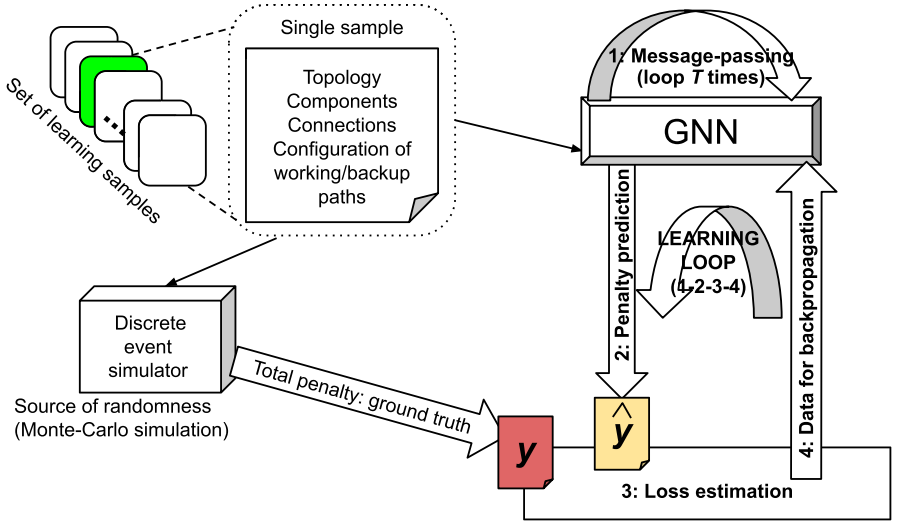

Fig. 1 Operation of the RiskNet prediction module during the training phase. Each simulation result is considered a sample from the unknown penalty distribution

a prediction of the total penalties $\hat{y}$ (2). The latter values are confronted with y, that is, the ground truth (real penalty levels) provided by the simulator. On this basis, the learning error (loss) is calculated (3) and the classical backpropagation algorithm is used to update the internal weights of the neural networks that form GNN (4). Then, the next sample (i.e., network topology) is provided. The learning samples are nominal mixed with faulty configuration. However, we do not distinguish between nominal and faulty configuration. Nominal configuration results in 0 penalty and do not contribute to the total penalty over one year of operation.

The output values $\hat{y}$ can be far from the desired label values y, and the loss is significant especially at the beginning of the training process. With the progress of the training (including iterations of message-passing), the loss decays.

We must emphasize that the consequences of network failures cannot be modeled as a typical supervised learning problem, where we have a label to predict. Due to the random nature of failures, the same input may produce different results in the simulation, since the output penalties are dependent on random samples (failures and recovery times). However, we are interested in the entire distribution of penalties conditioned on the network configuration. Knowing such distributions, we can compute any possible risk measure simply by using the analytical formula for these particular distributions. This has a clear advantage over alternatives, such as quantile regression, where the model would produce only a few quantiles of the penalty distribution. In this way, we can easily switch from optimization based on pure *VaR* to a one based on more meaningful and robust *CVaR*. From a mathematical point of view, the penalty is expressed on the basis of risk theory dealing with extreme events. First, we estimate the entire distribution of the total penalty. Then we find the 95^{th} percentile. If we were to deal with pure value-at-risk ($VaR_{5\%}$), we would only be interested in this percentile value. However, we use a more robust measure, that is,

conditional *VaR* ($CVaR_{5\%}$). Therefore, we calculate the average of all penalties above the percentile value.

In the following, we describe the consecutive steps of the entire prediction concept.

3.1 Notation for Modeling

Customers carry data on the established connections on the network. We assume that the network topology is represented by the following. (a) Set $\mathcal{C} = \left\{c_i\right\}_{i=1:n_c}$ of the basic communication components c_i (links or edges) prone to failure. Only the bandwidth of the link can limit the shared protection capabilities. We assume realistic distributions of up- and down-times of links and treat them as the resilience attributes of these components. For the sake of this study, routers are treated as fully reliable. (b) Set $\mathcal{S} = \left\{\boldsymbol{s}_k\right\}_{k=1:n_s}$ of connections $\boldsymbol{s}_k$ between pairs of end-points (routers) in the network topology. The routing for each SLA is defined as a pair of sets of components $\boldsymbol{s}_k = \left(s_p^k, s_b^k\right)$, where s_p^k is the set of components on the working path and s_b^k contains components on the backup path prepared for the k-th SLA. The connections are characterized by their demand volumes. These demands should be carried out with the help of resilient connections, and this fact is reflected in service level agreements (SLAs). In essence, we identify a connection with its business-oriented description given by the SLA.

In general, such properties (features) of both components and SLAs are denoted by data vectors: $\mathbf{x}_{c_i}$ and $\mathbf{x}_{s_k}$, respectively. The vector $\mathbf{x}_{c_i} \in \mathbb{R}_+^4$ contains resilience parameters (parameters of the up- and down-time distributions) and design parameters (that is, the backup bandwidth reserved in links) of an individual component c_i. On the other hand, $\mathbf{x}_{s_k} \in \mathbb{R}_+^1$ contains SLA's parameters, in our case the demand volume for connection $\boldsymbol{s}_k$. Both features $\mathbf{x}_{s_k}$, $\mathbf{x}_{c_k}$ and the routing $\mathcal{S}$ are jointly denoted as $\boldsymbol{x} = (\mathbf{x}_{s_k}, \mathbf{x}_{c_k}, \mathcal{S})$.

3.2 Approximate Penalty Distribution

The ultimate goal in risk analysis of communication networks is the evaluation of the conditional distribution $P = \mathsf{P}(Y|\boldsymbol{x})$ and, in particular, its quantiles to obtain *VaR* or its derivatives. For simple protection schemes (e.g., dedicated protection) with the additional assumption of exponential up- and down-time distributions, P can be obtained analytically [9]. According to the best authors' knowledge, there are no analytical results for more realistic scenarios of shared protection procedures and non-Poisson downtimes. It is relatively simple to sample from P by simulation. In principle, one can estimate the risk measures with a Monte-Carlo method. However, simulations take a lot of time to obtain reliable estimates due to the rare nature of failures. The method proposed in this paper is to approximate P by a surrogate distribution $Q = \mathrm{nn}(\boldsymbol{x})$, where nn is a neural network (GNN, in our case) that maps topology properties to the family of parametric distributions. The network consists of its own parameters (weights) θ learned from simulations.

The training objective of nn used in this study is to minimize the Kullback–Leibler divergence $D_{\mathrm{KL}}(P \parallel Q)$ between the distributions. In particular, for given network parameters and simulated penalty $\mathbf{y} \in \mathbb{R}^{n_s}$ we use Monte-Carlo approximation to the true KL divergence (a single sample is unbiased estimator of the expectation in KL definition):

$$D_{\mathrm{KL}}(P \parallel Q) \approx \log \mathsf{P}_P(\mathbf{y}) - \log \mathsf{P}_Q(\mathbf{y}). \tag{1}$$

In fact, the simulated values are sampled from the unknown distribution P, thus—by the Monte-Carlo approximation (sampling from the simulator)—the loss is related to the distribution Q parametrized by GNN. The Monte-Carlo approximation is a well-known method used especially in Bayesian variation inference.

Since $\log \mathsf{P}_P(\mathbf{y})$ does not depend on θ, it does not contribute to the parameter update and we can ignore this term and use the negative log-likelihood function of the surrogate distribution as the loss function. With the additional assumption of conditional independence of the SLAs, the loss function simplifies considerably to the following:

$$\ell(\theta, \boldsymbol{x}, \mathbf{y}) = -\frac{1}{n_s} \sum_k \log \mathsf{P}_Q\big(y_k | \boldsymbol{x}, \theta\big), \tag{2}$$

Due to its generalization potential, we applied Student t-distributions as a parametric family. As used in many cases in statistical modeling (e.g., robust regression), we assume five degrees of freedom. This is a justified approach ensuring a proper heavy tail and is convenient for the training of neural networks (i.e., incurres a relatively simple likelihood function). In addition, it ensures the existence of the mean value and variance. Thus, we still have a bell-shaped distribution with an analytical PDF for efficient training of nn. Other candidate distributions involve normal and log-normal distributions. During the initial calculations, the normal distribution (suggested by the Central Limit Theorem) gave us results similar to t-distribution. In inspection of the simulations, we observed that the t-distribution matches the tail more accurately. On the other hand, the log-normal distribution has a too heavy tail, and in this case, it is sufficiently accurate only for a smaller number of failures. The bell-shaped distribution is suitable for the simulation setup, as we always observe a few failures a year in a communication topology. In the case of an extremely resilient topology with the possibility of no-failures, a zero-inflated log-normal distribution is recommended [32], as it can model both the probability of no failure and the cost distribution given that the failure has occurred.

The architecture of nn can be as simple as mutli-layer perceptron; however, since there is neither particular order of the SLAs nor the components, we additionally require model equivariance under permutations. This makes a GNN a perfect candidate for nn.

3.3 GNNs: Graph Neural Networks

The association between the set of components and the set of SLAs can be represented as *bipartite metagraph* (illustrated in Fig. 2). Both components and SLAs can

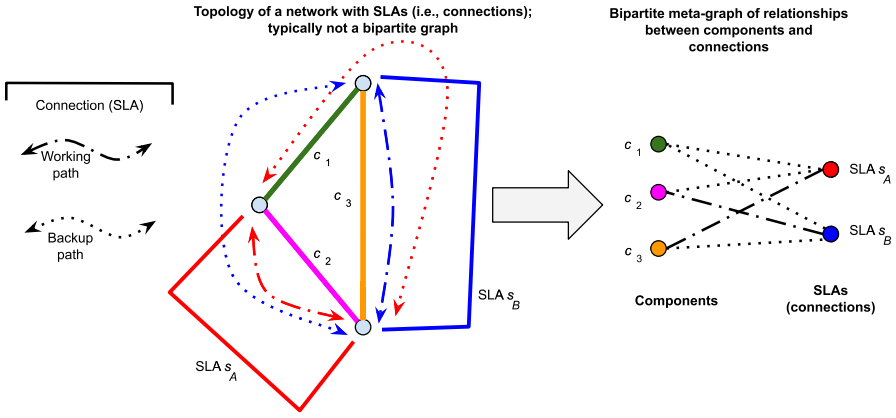

Fig. 2 Transformation of the given topology onto the associated bipartite metagraph for GNN

be represented as nodes[2] of the association graph, with edges connecting an SLA and a component if and only if the latter is used in the path for the given SLA. There are two kinds of edges in the metagraph: one representing a working path (i.e., that a component is used in SLA's working path) and the other for the backup path.

Our main idea of the RiskNet prediction system is to run a heterogeneous GNN on the presented bipartite metagraph to get the surrogate penalty distribution. We base our analysis on a version of GNNs known as *message-passing neural networks* [33].

The way the system is trained is shown in general in Fig. 1. The GNN core algorithm is presented with the help of a pseudocode given in Fig. 3. A GNN uses an architecture of the neural networks for regression independent of the topology of the communication network they deal with. In this way, GNN can provide a universal representation of the properties of any topology represented as a graph. As a result, we can select the neural network architecture in advance, despite the fact that we do not know which size will be most suitable to a particular case. It makes the particular neural architecture topology-invariant, in contrast to the widely used topology-aware approaches in ML. In fact, a notion of GNN in singular is a little bit misleading, since the whole method uses as many as seven different neural networks ($M^p_{s\to c,t}$, $M^b_{s\to c,t}$, $M^p_{c\to s,t}$, $M^b_{c\to s,t}$, U^c_t, U^s_t and F; their meaning is defined below) to find the final output.

We use a message-passing GNN built of differentiable layers; therefore, they can be trained with backpropagation algorithms. The 'layer' notion used here should not be limited to a layer inside a neural network. The idea of GNN enables us to train neural networks to be able to maintain internal relationships for any topology. These internal relationships represent a sort of knowledge about the topology and related relationships kept in the nodes of our bipartite metagraph. This knowledge is represented with vectors associated with the components and nodes related to SLA and

[2] To avoid misunderstanding, the vertices of the bipartite metagraph are denoted as 'nodes', while the vertices of the studied communication network topology are called 'routers'.

Fig. 3 Internal architecture of the RiskNet prediction module

Input: $\mathbf{x}_c$, $\mathbf{x}_s$,$\mathcal{S}$
Output: $\mathbf{h}_c^T$, $\mathbf{h}_s^T$, $\hat{\mathbf{y}}_s$ // The output will include the prediction of the penalty related to each SLA, $\hat{\mathbf{y}}_s$
foreach $c \in \mathcal{C}$ **do** $\mathbf{h}_c^0 \leftarrow [\mathbf{x}_c, 0 \ldots, 0]$;
foreach $\boldsymbol{s} \in \mathcal{S}$ **do** $\mathbf{h}_s^0 \leftarrow [\mathbf{x}_s, 0 \ldots, 0]$;
// Initialization of internal states
for $t = 0$ *to* $T - 1$ **do**
  foreach $\boldsymbol{s} \in \mathcal{S}$ // Message-passing between SLAs and components used by their working/backup paths **do**
    foreach $c \in s_p$ // Message-passing from SLAs to components in their working path **do**
      $\tilde{\mathbf{m}}_{p,s\to c}^{t+1} \leftarrow M_{p,s\to c}^t(\mathbf{h}_s^t, \mathbf{h}_c^t)$
    end
    foreach $c \in s_b$ // Message-passing from SLAs to components in their backup path **do**
      $\tilde{\mathbf{m}}_{b,s\to c}^{t+1} \leftarrow M_{b,s\to c}^t(\mathbf{h}_s^t, \mathbf{h}_c^t)$
    end
  end
  foreach $c \in \mathcal{C}$ **do**
    foreach $\boldsymbol{s} : c \in s_p$ // Message-passing from components to SLAs using them in working paths **do**
      $\tilde{\mathbf{m}}_{p,c\to s}^{t+1} \leftarrow M_{p,c\to s}^t(\mathbf{h}_s^t, \mathbf{h}_c^t)$
    end
    foreach $\boldsymbol{s} : c \in s_b$ // Message-passing from components to SLAs using them in backup paths **do**
      $\tilde{\mathbf{m}}_{b,c\to s}^{t+1} \leftarrow M_{b,c\to s}^t(\mathbf{h}_s^t, \mathbf{h}_c^t)$
    end
  end
  foreach $c \in \mathcal{C}$ **do**
    $\mathbf{m}_c^{t+1} = \sum_{\boldsymbol{s}:c\in s_p} \tilde{\mathbf{m}}_{p,s\to c}^{t+1} + \sum_{\boldsymbol{s}:c\in s_b} \tilde{\mathbf{m}}_{b,s\to c}^{t+1}$ // Message obtained by a component
    $\mathbf{h}_c^{t+1} \leftarrow U_c^t\left(\mathbf{h}_c^t, \mathbf{m}_c^{t+1}\right)$ // Update of internal state of a component
  end
  foreach $s \in \mathcal{S}$ **do**
    $\mathbf{m}_s^{t+1} = \sum_{c:c\in s_p} \tilde{\mathbf{m}}_{p,c\to s}^{t+1} + \sum_{c:c\in s_b} \tilde{\mathbf{m}}_{b,c\to s}^{t+1}$ // Message obtained by an SLA
    $\mathbf{h}_s^{t+1} \leftarrow U_s^t\left(\mathbf{h}_s^t, \mathbf{m}_s^{t+1}\right)$ // Update of internal state of an SLA
  end
end
$\hat{y}_s \leftarrow F(\mathbf{h}_s)$ // Read-out: prediction (on the basis of convergent internal state) of a penalty for not meeting an SLA

is indicated as $\boldsymbol{h}_c^t$ (for the component c) and $\boldsymbol{h}_s^t$ (for the SLA s), respectively. These vectors, known as internal (or hidden) states, are found to be one of the results of our algorithm's operation. Due to the inherent problems with interpretability of neural networks, we are not necessarily able to tell what the exact values mean. Hidden states change iteratively during the message-passing process. Therefore, we also denote a given iteration with superscripts t. The internal state of one node influences the internal states of other nodes if they are adjacent in our bipartite metagraph. Modifications are made in the form of an iterative exchange of messages dependent on internal states. These messages have nothing in common with routing messages (or anything of this kind that is exchanged in communication structures) and are only a part of a specific GNN operation. In Fig. 3, the messages are represented as $\tilde{\mathbf{m}}_{r,u\to z}^t$, where t represents the iteration; $r \in \{p, b\}$ represents the message related to the working (p) or backup (b) path, and $u, z \in \{c, s\}$ represents in which direction the message is passed ($s \to c$ denotes the SLA to the component message, while $c \to s$ denotes the message in the opposite way). At the end of each iteration, the total message obtained by a node of the bipartite metagraph (that is, component or SLA) is calculated as the sum of all the above-mentioned messages directed to this node (we represent it as $\mathbf{m}_u^{t+1}$ with $u \in \{c, s\}$ in Fig. 3). The messages exchanged between two nodes are calculated as functions of the internal states of these nodes. The function is obtained as an output of a neural network represented as $M_{r,u\to v}^t$ (the above-mentioned notation related to $\tilde{\mathbf{m}}_{r,u\to z}^t$ is again valid). These neural networks are called *message functions*. They encode the information exchanged between the related components and the SLAs. We can see that the working and backup paths have different message functions and that they can deal with two directions. Therefore, we have four message functions used ($M_{p,s\to c}^t$, $M_{b,s\to c}^t$, $M_{p,c\to s}^t$, $M_{b,c\to s}^t$). These functions can also be different in various iterations (that is, why we also use the superscript t for them). Additionally, the new internal state of a node is based on its previous internal state and the messages obtained in the current iteration. To calculate it, we use neural networks denoted as U_c^t and U_s^t. These *update functions* encode the combined incoming information into the hidden state. The forward pass begins with zero-padded components and SLA feature vectors, followed by an iterative exchange of messages and state updates. In particular, the result of embedding every edge in the metagraph gives vectors $\tilde{\mathbf{m}}_{r,u\to z}^t$.

The entire process typically converges after a few (T) iterations (i.e., the internal states cease to exchange). The parameter T controls the range of interactions between SLAs; for example, for $T = 1$ each SLA receives messages only from its components, and the model is basically a DeepSet [34]. The higher T allows information to be exchanged between SLAs through components. In our case, the steady state is obtained even for as few as six iterations ($T = 6$ in our case; while the original paper introducing the notion of message-passing [35] shows examples with $T = 4$). We would like to note this attractive aspect of the method, although fast convergence is the phenomenon observed experimentally and we do not have a theoretical justification for its repetition. We can only speculate about the interpretation of message passing as some sort of refining process. In this picture, a GNN is an iterative algorithm, although, we advise using the same T during both training and inference.

The predicted penalty related to an SLA is found using a small *read-out* neural network (represented as F) applied to the final hidden state of the SLA. Its output represents the parameters of the penalty distribution for this particular SLA. In terms of particular neural architectures, we use *affine functions* for message propagation (M) and *GRU units* for update (U). These are the proven units that are used in many GNN architectures. They are selected as a trade-off between simplicity, performance, and flexibility of the model. Similarly to previously proposed models, the weights of the message and update functions are reused for subsequent message-passing iterations. The read-out function proposed in this paper is a multilayer perceptron with the *SeLU* activation function. The mapping from the raw read-out output to the Student t-distribution location parameter is the identity; however, the scale must be constrained to positive numbers, so we use the softplus function.

At the end of this subsection, we give a few pieces of information on the complexity of the whole algorithm. As the basic parameter used to express the complexity, we consider the number of components N. Since it is equal to the number of links that in typical topologies (namely: not dense) is of the same order as the number of topology vertices $\mathcal{O}(N)$, we can assume that the order of SLAs, related to a maximum number of different pairs in topologies, is $\mathcal{O}\left(N^2\right)$. It is known [36] that the complexity of the message-passing neural network is at the level of $\mathcal{O}\left(V^2G^2\right)$, where V represents the number of nodes in the graph on which the GNN is run (in our case: the bipartite metagraph of components and SLA), and G represents the number of dimensions of property vectors representing the internal states in the GNN. In our case, $V = \mathcal{O}\left(N^2\right)$ (stems of the number of SLAs), and G is constant and equal to 32. The latter is related to the fact that we decided to use the vectors that contain the number of all SLAs. In this way, the overall computational complexity of our methods is quadruple $\mathcal{O}\left(N^4\right)$. On the other hand, the space complexity is simply $\mathcal{O}\left(V^2\right)$ [36] (the vectors used dominate the complexity); therefore, it is also quadruple $\mathcal{O}\left(N^4\right)$.

3.4 Simulation

All experiments reported in this paper are supported by the previously used discrete event risk simulator and verified by comparison with the theoretical results in [9] in a series of unit tests. The software is written in C++. The simulator is treated as the source of ground truth and is used for the creation of training datasets for GNNs. A network topology is the starting point for building the configuration. In the experiments, the topologies under study are either those based on the random Barabási–Albert model (with the output power distribution of node degrees) or the existing topologies retrieved from the SNDLib library (http://sndlib.zib.de). It is necessary to emphasize that the fact that we are able to effectively use artificially induced topologies is a great advantage over our method: this way, we gain virtually unlimited number of training data, while we can test the quality of the operation of the model with existing topologies (and we have a very small number of them). We need to emphasize that this is a very

interesting result: altough the structure of Barabási–Albert networks is of high homogeneity and cannot cover most of the features in the realistic topologies, the obtained deviations of prediction for existing topologies are not considerable. During training, the direct knowledge of the existing topologies does not influence our model, which is extremely important to provide generalization and to apply our method practically. According to the random model related to the Barabási–Albert concept, every new router is attached to at least two existing ones. This method always makes it possible to construct the working and backup paths for any connection between a pair of routers. Although there are other methods to construct random graphs (e.g., with the most classical one, proposed by Erdös-Rényi), the Barabási–Albert one is perceived as the most adequate for existing communication network topologies, since it generates scale-free topologies (with nodal degree power distribution) [37].

Afterwards, for a selected topology, we first generate the working and backup path for every connection (SLA) between all routers. For the creation of training datasets, we do not use a typical approach of looking for the two-shortest candidate paths. Instead, we chose router-disjoint path randomization from the set of all disjoint paths found for all connections. Namely, we set the probability of drawing a given path as decreasing as a function of a path's length (e.g., a number of components on the path). We select a parameter ξ and then draw paths with probability $p_k \sim e^{-\xi|s_k|}$. This value is normalized. Then, for small values of ξ, the distribution is flat and long paths have a high probability of being selected, while for large ξ, we have mainly the shortest paths, as p_k drops quickly with length. For the training phase, we use $\xi = 0.1$. This approach allows us to explore the configuration space and gives the GNN a highly divergent set of training samples. The volume of demand for a connection is proportional to the product of the size of the source and sink routers for the connection. The size of a router depends on its degree d and is uniformly distributed in the interval $10 \times (d \pm 1)$. In the end, the resilience parameters of different components are generated. Here, we assume Poissonian failures (i.e. exponential up-times) and Pareto-distributed down-times. Both are parametrized according to the estimates reported in [21]. The link lengths are obtained from the scaled spring layout of the network topology. The simulation output is the total penalty for each SLA in each simulated year of operation. This value is used as a target (label value y) in the GNN training process.

4 Numerical Results

To simplify, we can say that we use a type of supervised learning (heteroscedastic regression), since we are able to provide the real values (ground truth) of the penalties and confront them with the predicted values to train the model. Real values are provided by simulations for specific configurations. On the other hand, simulations take a lot of time; that is why we can afford them only in the training phase. During the operation of the whole system, a very fast GNN model predicts the values for given working/protection path settings.

4.1 Training and Hyperparameters

During the training phase, we first select different network topologies with randomly generated parameters (training samples). They fed the GNN and a discrete-event simulator, and the result is stored for offline training. Second, the message-passing is run in GNN to obtain the convergence and provide prediction of the total penalties $\hat{y}$. The predicted penalties are confronted with y, that is, the ground truth (real penalty levels) provided by the simulator. On this basis, the learning error (loss) is calculated and the classical backpropagation algorithm is used to update the internal weights of the neural networks forming GNN. Concerning inference, the output of the prediction model follows the same path as in the case of the training. The only exception to this is when one wants to use dropout to estimate the uncertainty of the prediction. Then, the output is equal to the average of multiple stochastic forward passes.

In the spirit of the modern deep learning approach, we used the raw simulator parameters as input to RiskNet and let the model learn a meaningful internal representation. The SLA feature contains only the demand volume. The vector of components has four dimensions (failure intensity, α and β parameters of the Pareto downtime distribution, and the capacity reserved for protection). The only transformation applied to the data is the z-score normalization, as it improves the training process. We do not consider routing as a feature but rather as metadata.

Training a deep neural network typically requires hundreds of thousands of samples. Learning from simulations makes it easier to obtain samples; however, we are still limited by training and simulation time. Our training set is constructed from a simulation of 1000 random network topologies (generated according to Barabási–Albert model) with a number of routers uniformly distributed in the range [10, 40]. Each topology was simulated for up to 1000 years of operation. Since for some of the largest networks, not all simulations finished under the assumed time constraint, we ended up with around 829 000 training examples. Using the same procedure, we generated additional 20 000 test samples to spot signs of overfitting. With this training set, we tested multiple RiskNet configurations, mostly differentiated by hyperparameter values. On this basis and according to our prior knowledge of GNNs, we chose the final configuration. Both hidden vectors have 32 dimensions. The message has 64 dimensions. The kernel of the affine message function is regularized with the coefficient 0.01, and the bias is not regularized. The message-passing loop is iterated six times. Finally, the read-out function has three hidden layers of sizes (64, 64, 32) interleaved with two dropout layers with dropout rates 0.2 and 0.1, respectively. The model characterized above was trained for 54 epochs of 12 900 iterations of the Adam optimizer on a batch of 64 topologies. The learning rate was set at 0.0001 for the first 20 epochs. Then it decayed by 0.99 per epoch. The entire training took 11 hours 40 min with the number of iterations in GNN equal to $T = 6$.

The model was implemented entirely in TensorFlow. The hardware supporting calculations embraces 36 cores Intel(R) Xeon(R) Gold 5220 CPU @ 2.20GHz, while the graphics processor unit used is GPU Tesla V100 SXM2. As concerns the effective speed of the calculations, we emphasize that they are extremely fast. For example, the average calculation time for the janos-us network equals 4 ± 0.2 ms for

Table 1 GNN evaluation loss

Topology	GNN $T = 6$	Baseline	GNN $T = 1$
Train	− 1.37	–	− 1.34
Test	− 1.43	–	− 1.38
Validation	− 1.42	1.10	− 1.39
Average SNDLib	− 0.88	1.62	− 0.74
dfn-bwin	0.73	4.09	1.19
Abilene	− 1.67	0.87	− 1.62
Nobel-germany	− 1.37	0.88	− 1.32
cost266	− 1.32	1.00	− 1.25
Geant	− 1.38	1.01	− 1.33
nobel-eu	− 1.32	0.97	− 1.20
janos-us	− 1.14	1.11	− 1.04

a single evaluation of the penalty by GNN (11 000 evaluations per minute). On the other hand, a single simulation of 600 years of network operation takes 49±2 sec. with 36 cores of CPU. In this way, we obtain 12 000 times in the speed improvement of the calculation. Furthermore, the full distribution from GNN allows for an easy switch to different metrics being optimized.

4.2 Evaluation

The final model was evaluated with the third synthetic dataset, as well as with the majority of topologies retrieved from SNDLib. To show the benefits of RiskNet, we compare the results with the baseline model. Here, the baseline is a marginal distribution of all penalties, that is, the distribution of penalties in all experiments without any distinction based on $\boldsymbol{x}$. Since the training data was normalized, the baseline distribution is a standard Student t-distribution with five degrees of freedom. GNN can improve prediction using information from the features $\boldsymbol{x}$ as summarized in Table 1.

Due to the fact that GNN estimates the entire distribution, common metrics, such as mean squared error or mean percentage error, are no longer meaningful. The loss must be expressed with negative log-likelihood. Therefore, the negative or positive value is not easy to interpret. However, the smaller the value, the better, and one can observe a significant improvement over the baseline provided by our approach. Furthermore, we can see that the results of the test evaluations are close. This proves the generalizability of the model. The effect is further confirmed by the results obtained with the real topologies, where GNN provides average scores at the level of −0.88 versus the baseline result of 1.62. Note that the negative log-likelihood values are not surprising since we are using a continuous distribution, and the values of logarithms of the probability density functions are not limited from above.

The loss obtained for RiskNet is much lower compared to baseline. This indicates that the model actually learns the information from a network topology. We can make this statement more precise in the context of information theory. The difference between log-likelihoods is a measure of information the model has learned. By

changing the base of the logarithm to 2, we can express this information in bits. In particular, for the validation set, the RiskNet system reduces the description length on average by 3.6 bits per path (compared to the baseline). For reference, the entropy of the baseline distribution is 1.6 bits.

In the experiments, we considered simpler models with weaker interactions between SLAs (as measured by T). In particular, setting $T = 1$ produces a surprisingly accurate model[3] without interactions, whose test loss quickly begins to grow during training. Similar behavior was observed in other weekly interacting models—all of them suffered from overfitting. We conclude that a high value of T acts as a regularizer for the model. We explain this by the fact that networks in the simulation were highly reliable and most of the contributions to the penalty were due to a single failure only. Having said that, we emphasize that, in general, SLAs must exchange information, since—by definition — they do interact in the case of shared protection.

Despite the fact that the negative log-likelihood applied as a loss function is a theoretically justified measure used for parameter estimation, it is difficult to state how accurate the fit is by reasoning on the basis of a single value. Therefore, to provide a more intuitive measure, we propose using probability plots (pp-plots, see Fig. 4) produced according to the following procedure. For every network configuration $\boldsymbol{x}$, RiskNet predicts the whole distribution of penalties Q. The ground truth $\boldsymbol{y}$ is obtained from the simulation. Given some probability value q, we construct a Bernoulli random variable $1_{\boldsymbol{y}<y_q}$, where y_q is a q-quantile of Q. If $\boldsymbol{y}$ were sampled from Q, the probability of this Bernoulli-distributed random variable would be equal to q. Since the predicted distribution is not the exact sampling distribution P, the estimated probability $\hat{q}$ will be different. The closer it is to q, the better approximation of the distribution we obtain. For the Bernoulli distribution, the unbiased estimator of the probability is just the average value, so we use $\hat{q} = \overline{1_{\boldsymbol{y}<y_q}}$. The pp-plot is constructed as a line plot of $\hat{q}$ vs. q. The diagonal line is added as a reference. Any deviation from this line indicates a mismatch in the distribution. We can observe that the distribution produced by RiskNet is much closer to the diagonal line than the baseline. The even more important aspect is the fact that the deviation from the diagonal is small for all probabilities. This tells us that RiskNet correctly predicts multiple quantiles of the distribution. This is a highly necessary feature, as it allows us to use the same model for the estimation of risk at different levels (that is, various percentiles p of $CVaR_{(1-p)\%}$). This property is practically useful when we would like to estimate various risk levels (e.g., due to some business applications).

The only case where the RiskNet distribution significantly deviates from the empirical samples is for the *dfn-bwin* network. However, this network is different from distribution samples. The network topology is almost the full graph in contrast to small-world topologies produced by the Barabási–Albert model. The fact that the baseline is

[3] The information difference between models is 0.04 bit per SLA.

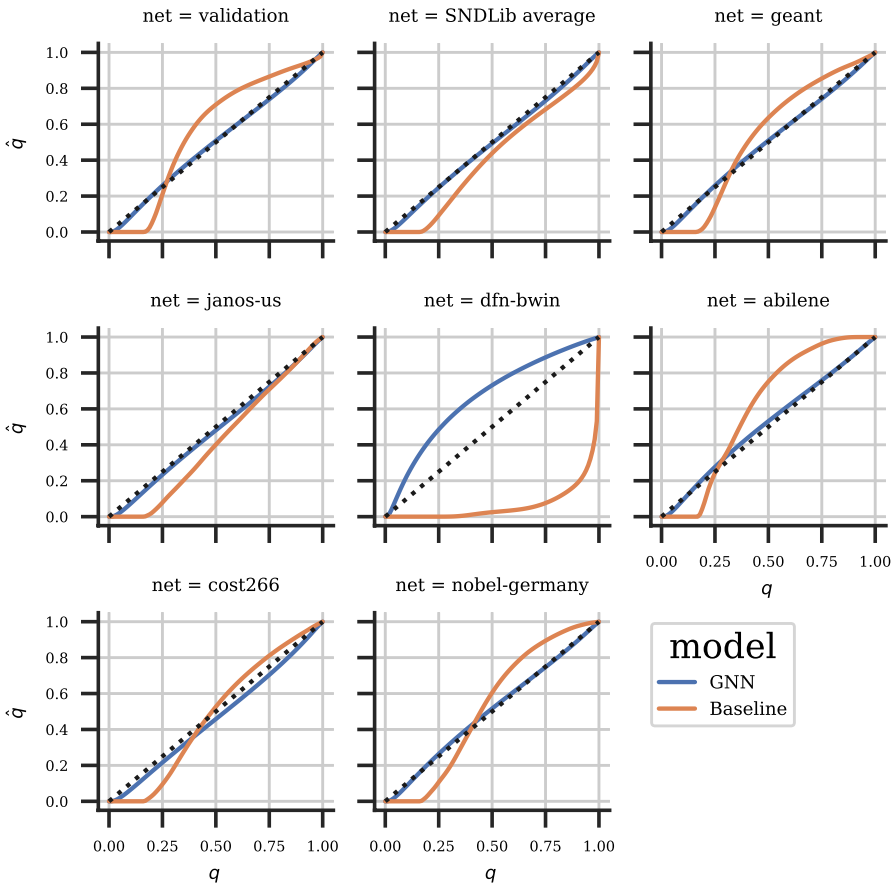

Fig. 4 Probability plots for topologies simulated for 1000 years of operation

also much less accurate for this network supports our claims even further. One must remember that RiskNet is a statistical model and, despite its generalization capabilities, there are some edge cases where the model cannot be considered as a good approximation. However, in some cases, a simpler model may be acceptable. From the viewpoint of applicability of our model, we can then state that the direct usefulness in relationship to IP long haul networks, our approach provides very good quality. We could be more skeptical about the data center or internal cloud networks, since the character of connections is more tending towards full graphs. On the other hand, here we just show some limitations when the training uses the Barabási–Albert model. If one plans to use our model in relation to more dense topologies, the model should just be trained with this type of random graphs.

5 Conclusions

In this paper, we propose a risk prediction system based on a graph neural network (GNN) and a bipartite metagraph, where penalties are treated as the main performance metric to assess the quality of resilience and reflect its financial aspect. The main idea of the proposed model is to run a heterogeneous GNN on the presented metagraph to get the surrogate penalty cost distribution due to component failures in a network. In this way, the weights of the message and update functions are reused for subsequent message-passing iterations. In this way, GNN parameterizes the Student's t-distribution to approximate the penalty cost distribution due to component failures in a network. Training is performed only on Barabási–Albert topologies that do not contain information on existing telecommunication topologies. However, since this is a very useful result, we are able to obtain a very good level of model generalization: the final model is evaluated with the majority of topologies retrieved from SNDLib. It proves that our approach meets practical requirements (speed of work and very high accuracy of prediction). It replaces time-consuming simulations, being an intuitive alternative to our proposal, by a very fast prediction method used—it can be applied by the network designer during connection optimization.

Although this work is derived and motivated by our experience in the field of communication and computer networks, it generalizes well beyond this area. Similar concepts of network or unreliable components arise in logistics and other areas of business importance. Since the penalty under consideration is based on downtimes, we expect this work to be extendable to the downtime-related quantities. A great advantage of the presented approach is related to the fact that we solve the problem without taking into account an analytical solution. Even if in some extreme cases such a solution can be found, we do not have to use it. Additionally, we can train our model once on random topologies and then reuse it in numerous different practical deployments. We also provide the whole distribution so that business people can freely apply various risk measures.

Our approach allows us to: (a) omit very complex, time-consuming, and ineffective modeling of resilience parameters for shared protection; (b) improve practically useful prediction quality of business-oriented risk parameters by using an ML-based module, providing very good results in comparison to the baseline case; (c) replace time-consuming simulations, being an intuitive alternative to our proposal, by a very fast prediction method used—it can be applied by a network designer during connection optimization.

Obviously, the proposed approach can be further extended. So far, we do not take into account the restoration (rerouting) methods. We plan to broaden the proposed model to include it. Additionally, we now assume that penalties for different connections are independent of each other. It is challenging, but tempting, to model a more realistic situation when they are dependent. This can be especially interesting when quality metrics are also taken into account. For example, switching traffic from one path may influence other paths, since it produces additional delays.

Author Contributions KR: concept and calculations; PJ, PB, PC: results analysis and description. FG, AC: consulation of the model and work on narrative. All authors reviewed the manuscript.

Funding This work was supported by the Polish Ministry of Science and Higher Education with the subvention funds of the Faculty of Computer Science, Electronics and Telecommunications of AGH University of Science and Technology (P.B., P.C.) and by the PL-Grid Infrastructure (K.R.).

Data Availability Not applicable.

Declarations

Conflicts of interest The authors declare no interests of a financial or personal nature.

Ethical Approval Not applicable.

References

1. Ghazizadeh, A., Akbari, B., Tajiki, M.M.: Joint reliability-aware and cost efficient path allocation and vnf placement using sharing scheme. J. Netw. Syst. Manag. **30**(5), 1–28 (2022)
2. Fujita, R., He, F., Oki, E.: Shared backup resource assignment for middleboxes considering server protection capabilities. Comput. Netw. **186**, 107734 (2021)
3. Stergiopoulos, G., Gritzalis, D., Kouktzoglou, V.: Using formal distributions for threat likelihood estimation in cloud-enabled it risk assessment. Comput. Netw. **134**, 23–45 (2018)
4. Kordestani, M., Saif, M., Orchard, M.E., Razavi-Far, R., Khorasani, K.: Failure prognosis and applications-a survey of recent literature. IEEE Trans. Reliab. **70**(2), 728–748 (2021)
5. Ozaki, H., Kara, A.: Reliability analysis of *M*-for-*N* shared protection systems with general repair-time distributions. IEEE Trans. Reliab. **60**(3), 647–657 (2011)
6. Yuan, S., Wang, B.: Highly available path routing in mesh networks under multiple link failures. IEEE Trans. Reliab. **60**(4), 823–832 (2011)
7. Riffat Ali, S.: Next Generation and Advanced Network Reliability Analysis. Using Markov Models and Software Reliability Engineering, Springer, Cham (2019)
8. Wu, L., Cui, P., Pei, J., Zhao, L.: Graph Neural Networks: Foundations, Frontiers, and Applications. Springer, Singapore (2022)
9. Rusek, K., Guzik, P., Chołda, P.: Effective risk assessment in resilient communication networks. J. Netw. Syst. Manag. **24**(3), 491–515 (2016)
10. Schupke, D.A.: Multilayer and multidomain resilience in optical networks. Proc. IEEE **100**(5), 1140–1148 (2012)
11. Teixeira, A., Sou, K.C., Sandberg, H., Johansson, K.H.: Secure control systems: a quantitative risk management approach. IEEE Control Syst. Mag. **35**(1), 24–45 (2015)
12. Wang, J., Chaudhury, A., Rao, H.R.: A value-at-risk approach to information security investment. Info. Sys. Res. **19**(1), 106–120 (2008)
13. Righi, M.B., Ceretta, P.S.: A comparison of expected shortfall estimation models. J. Econ. Bus. **78**, 14–47 (2015)
14. Bistouni, F., Jahanshahi, M.: Reliability analysis of Ethernet ring mesh networks. IEEE Trans. Reliab. **66**(4), 1238–1252 (2017)

15. Zhou, J., Liu, Y., Xiahou, T., Huang, T.: A novel FMEA-based approach to risk analysis of product design using extended Choquet integral. IEEE Trans. Reliab. **71**(3), 1264–1280 (2022)
16. Ke, C., Wu, J., Xiao, F., Huang, Z., Meng, Y.: A privacy risk assessment scheme for fog nodes in access control system. IEEE Trans. Reliab. **71**(4), 1513–1526 (2022)
17. Ahmad, W., Hasan, O., Pervez, U., Qadir, J.: Reliability modeling and analysis of communication networks. J. Netw. Comput. Appl. **78**, 191–215 (2017)
18. Gonzalez, A.J., Helvik, B.E., Hellan, J.K., Kuusela, P.: Analysis of dependencies between failures in the UNINETT IP backbone network. In: 2010 IEEE 16th Pacific Rim International Symposium on Dependable Computing, pp. 149–156 (2010)
19. Markopoulou, A., Iannaccone, G., Bhattacharyya, S., Chuah, C., Ganjali, Y., Diot, C.: Characterization of failures in an operational IP backbone network. IEEE/ACM Trans. Netw. **16**(4), 749–762 (2008)
20. Garraghan, P., Moreno, I.S., Townend, P., Xu, J.: An analysis of failure-related energy waste in a large-scale cloud environment. IEEE Trans. Emerg. Top. Comput. **2**(2), 166–180 (2014)
21. Kuusela, P., Norros, I.: On/Off process modeling of IP network failures. In: 2010 IEEE/IFIP international conference on dependable systems networks (DSN), pp. 585–594 (2010)
22. Ward, I.R., Joyner, J., Lickfold, C., Guo, Y., Bennamoun, M.: A practical tutorial on graph neural networks. ACM Comput. Surv. **10**, 54 (2022)
23. Xiao, Y., Pei, Q., Xiao, T., Yao, L., Liu, H.: MutualRec: joint friend and item recommendations with mutualistic attentional graph neural networks. J. Netw. Comput. Appl. **177**, 102954 (2021)
24. Zou, X., Li, K., Chen, C., Yang, X., Wei, W., Li, K.: DGSLN: differentiable graph structure learning neural network for robust graph representations. Inform. Sci. **626**, 94–113 (2023)
25. Paltrinieri, N., Comfort, L., Reniers, G.: Learning about risk: machine learning for risk assessment. Saf. Sci. **118**, 475–486 (2019)
26. Regmi, R.H., Timalsina, A.K.: Risk management in customs using deep neural network. In: 2018 IEEE 3rd international conference on computing, communication and security (ICCCS), pp. 133–137 (2018)
27. Chandrinos, S.K., Sakkas, G., Lagaros, N.D.: AIRMS: a risk management tool using machine learning. Expert Syst. Appl. **105**, 34–48 (2018)
28. Geyer, F.: DeepComNet: performance evaluation of network topologies using graph-based deep learning. Perform. Eval. **130**, 1–16 (2019)
29. Geyer, F., Bondorf, S.: DeepTMA: predicting effective contention models for network calculus using graph neural networks. In: IEEE INFOCOM 2019—IEEE conference on computer communications, pp. 1009–1017 (2019)
30. Sawada, K., Kotani, D., Okabe, Y.: Network routing optimization based on machine learning using graph networks robust against topology change. In: 2020 International conference on information networking (ICOIN), pp. 608–615 (2020)
31. Jiang, W., Bai, Y.: APGNN : alarm propagation graph neural network for fault detection and alarm root cause analysis. Comput. Netw. **220**, 109485 (2023)
32. McDavid, A., Gottardo, R., Simon, N., Drton, M.: Graphical models for zero-inflated single cell gene expression. Ann. Appl. Stat. **13**(2), 848–873 (2019)
33. Rusek, K., Chołda, P.: Message-passing neural networks learn Little's Law. IEEE Commun. Lett. **23**(2), 274–277 (2019)
34. Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R.R., Smola, A.J.: Deep sets. In: Advances in Neural Information Processing Systems (2017)
35. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. IEEE Trans. Neural Netw. **20**(1), 61–80 (2009)
36. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry (2017) arXiv:1704.01212
37. Newman, M.: Networks. Oxford University Press, Oxford (2018)

Krzysztof Rusek is an assistant professor at AGH and a data scientist at the Barcelona Neural Networking Center. He defended his Ph.D. Thesis on queueing theory in 2016 at AGH. Prior to that, he has worked as a system administrator and machine learning engineer in the research group focused on processing and protecting multimedia content. His main research interests are performance evaluation of telecommunications systems, machine learning, and data mining. Currently, he is working on the applications of Graph Neural Networks and probabilistic modeling for performance evaluation of communications systems and data mining in astronomy.

Piotr Boryło earned his doctorate and D.Sc. in telecommunications from AGH University of Science and Technology (Krakow, Poland) in 2016 and 2023, respectively. Since 2015, he has been a faculty member at the university's Institute of Telecommunications, where he currently serves as an Associate Professor. His research interests include, among others, software defined networks, network function virtualization, cloud computing, 5G core networks, emerging network services and network optimization. He is a co-author of numerous research papers, including works prepared in international teams from Italy, France and Canada.

Piotr Jaglarz received the M.S. degree in Information and Communication Technologies in 2017 from AGH University of Science and Technology, Krakow, Poland. He is especially interested in resource optimization in software-defined networks. He is the co-author of 2 international journal papers and a few conference papers.

Fabien Geyer is currently with Airbus Central Research & Technologies and Technical University of Munich (TUM) working on methods for network analytics, network performance, and architectures. He received the master of engineering in telecommunications from Telecom Bretagne and the Ph.D. degree in computer science from TUM. His research interests include novel methods for data-driven networking and formal methods for performance and network modeling.

Albert Cabellos is an assistant professor at Universitat Politècnica de Catalunya (UPC), where he obtained his Ph.D. in computer science engineering in 2008. He is a director of the Barcelona Neural Networking Center (BNNUPC) and a scientific director of the NaNoNetworking Center in Catalunya. He has been a visiting researcher at Cisco Systems and Agilent Technologies, and a visiting professor at the KTH, Sweden, and the MIT, USA. His research interests include the application of Machine Learning to networking and nanocommunications. His research achievements have been awarded by the Catalan Government, his university, and INTEL. He also regularly participates in standardization bodies such as the IETF.

Piotr Chołda earned his doctorate in telecommunications from AGH University of Science and Technology (Krakow, Poland) in 2006. Since then, he has been a faculty member at the university's Institute of Telecommunications, where he currently serves as an Associate Professor. His research expertise lies in the design and management of computer and communication networks.

Authors and Affiliations

Krzysztof Rusek[1] · Piotr Boryło[1] · Piotr Jaglarz[1] · Fabien Geyer[2] · Albert Cabellos[3] · Piotr Chołda[1]

✉ Piotr Chołda
piotr.cholda@agh.edu.pl

Krzysztof Rusek
krusek@agh.edu.pl

Piotr Boryło
borylo@agh.edu.pl

Piotr Jaglarz
piotrj9@gmail.com

Fabien Geyer
fgeyer@net.in.tum.de

Albert Cabellos
alberto.cabellos@upc.edu

[1] Institute of Telecommunications, AGH University of Science and Technology, Al. Mickiewicza 30, 30-059 Krakow, Poland

[2] Department of Informatics, Technical University of Munich, Boltzmannstr. 3, Garching bei München, Germany

[3] Barcelona Neural Networking Center, Universitat Politècnica de Catalunya, Barcelona, Spain