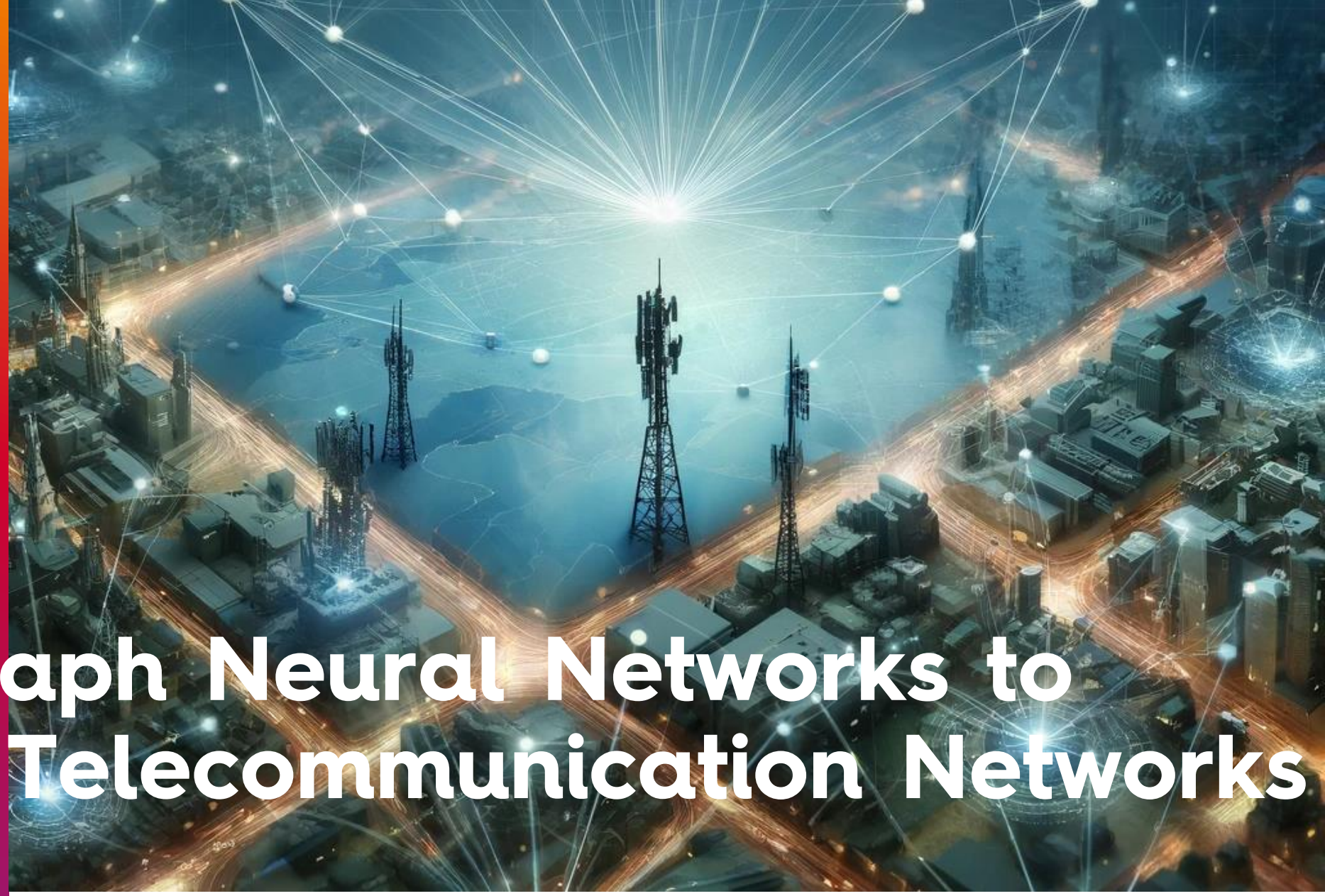
sopra steria

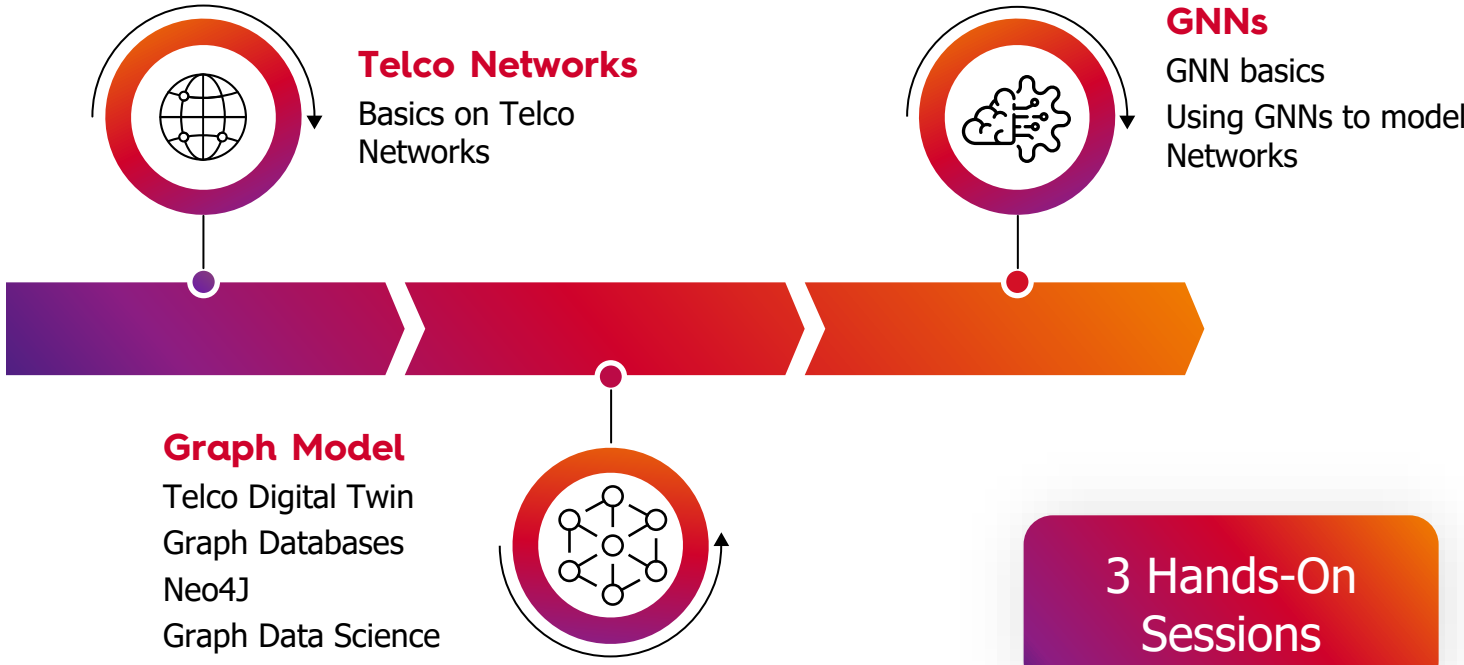
Using Graph Neural Networks to improve Telecommunication Networks

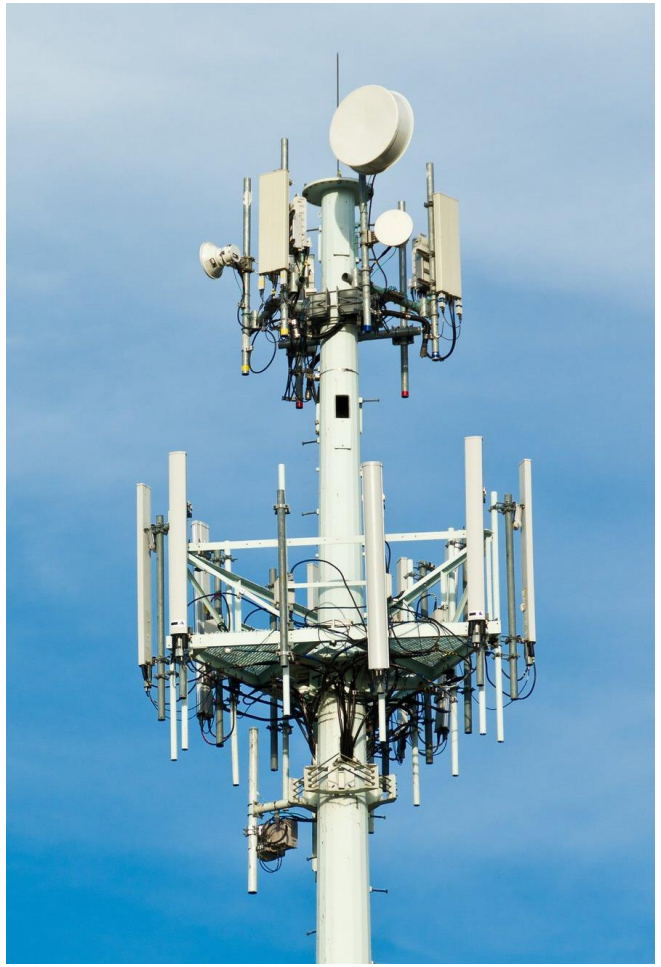


Who is Who

- Where are you from?
- Who is at ML Prague for the first time?
- What area of ML are you specialized in?
- Who has worked with Graphs before?
- Who has worked in Telco before?
- PyTorch or TensorFlow?
- Hladinka or Šnyt or Mlíko?

Todays Roadmap





sopra steria

Telecommunication & ML

ML Use Cases in Telco

- **Customer Service and Experience:** Chatbots and Virtual Assistants, Personalized Recommendations Churn Prediction, Fraud
- **Security:** Anomaly Detection, Security Automation
- **Operational Efficiency:** Inventory Management, Cost Reduction, Decision Making, Forecasting
- **Quality of Service Enhancement:** Network Performance Monitoring, Service Quality
- **Network Optimization and Management:** Predictive Maintenance, Traffic Routing and Management, Resource Allocation

What is the Challenge in telco operations?

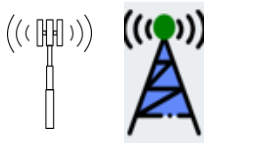
- Ensuring best possible quality of service requires constantly:
  - Building new telco sites (to fill white spots)
  - Upgrading technologies (e.g. 4G → 5G)
  - Fixing problems / Optimize routings
- Changes in one point of the network impact a different point of the network

Problem:

Telecommunications companies **can't pull over** and shut down the network for a short time, like changing a tire in a pit stop. All of the **several hundred daily changes** and adjustments must be made **during ongoing operations**.

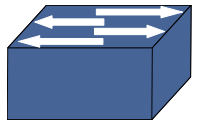
How does a mobile net(work)?

Antennas

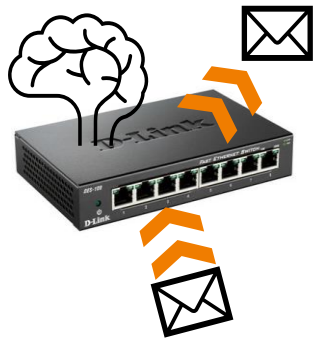


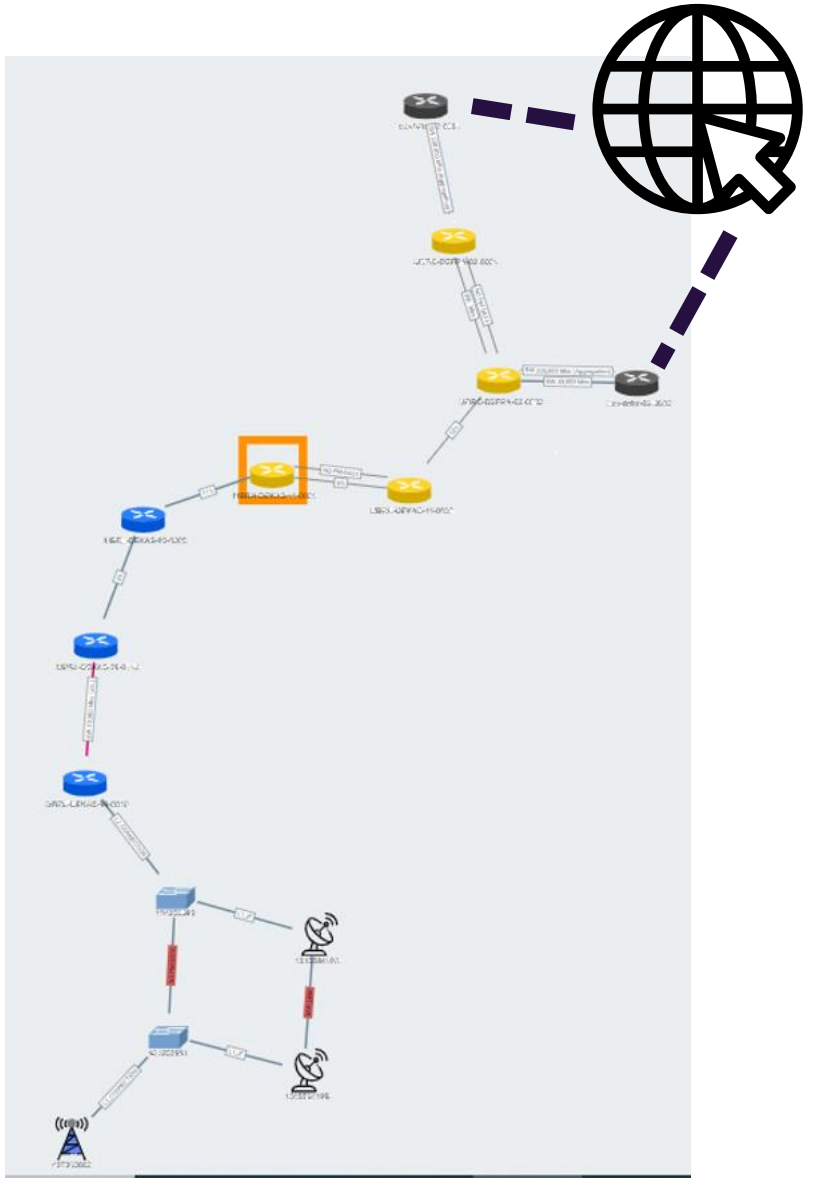
Radio link

Switch



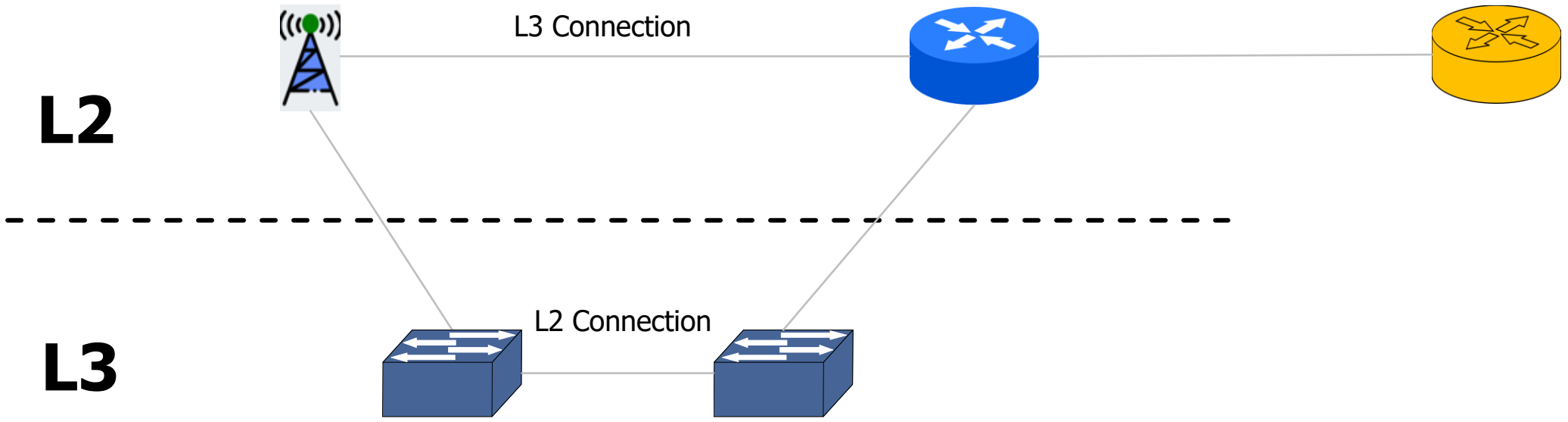
Router





sopra steria

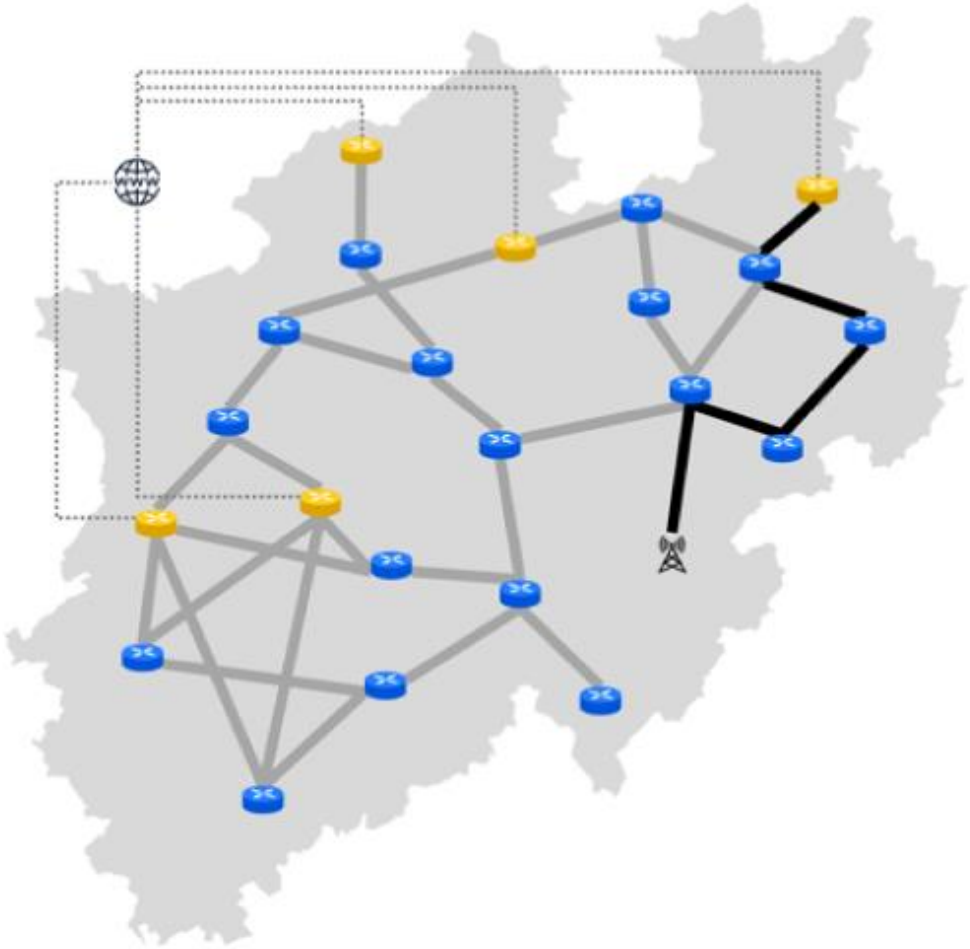
How does a mobile net(work)?



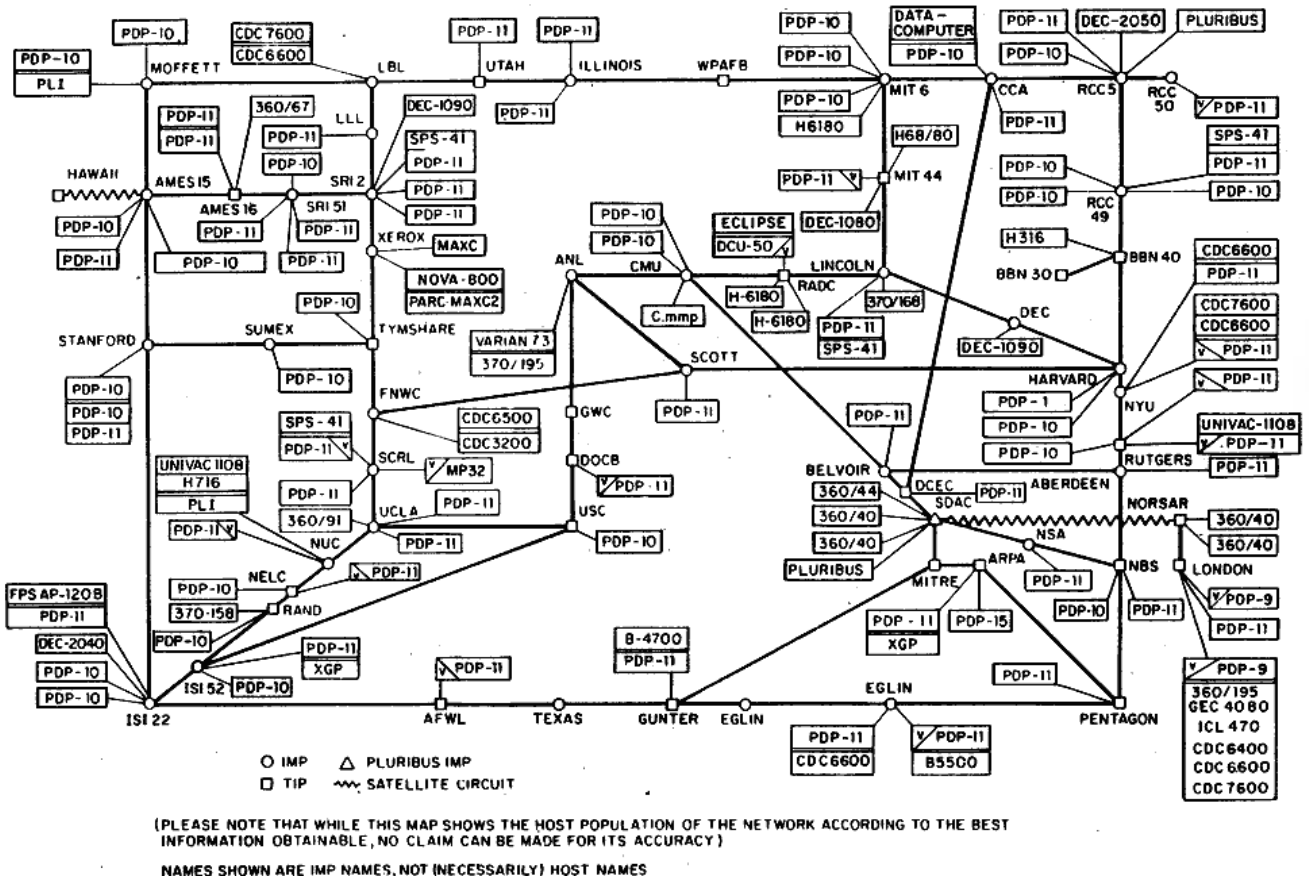
How does the data packet know its way?

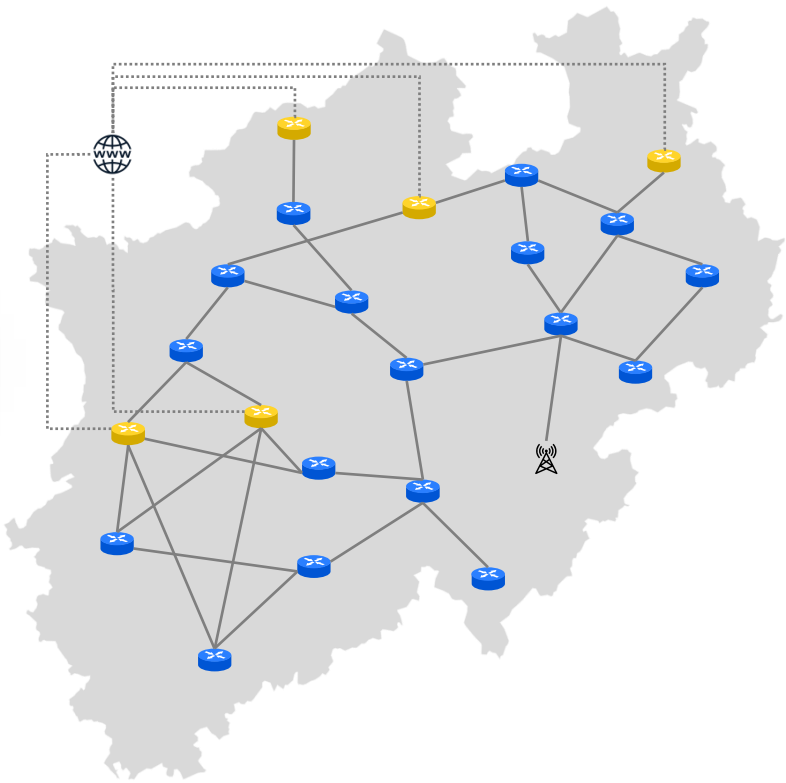
Simplified:

- Traffic must be transported from "blue" routers (ACCESS) to "yellow" routers (AGGR), which transport it further to backbone routers.
- Network experts decide which AGGR router is assigned to an ACCESS router.



Network Documentation



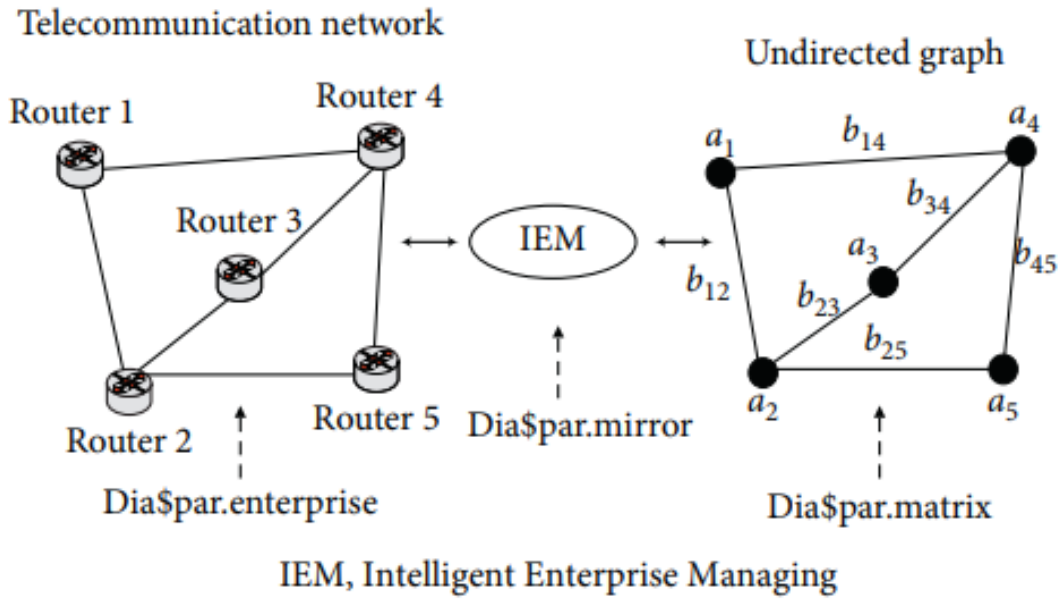


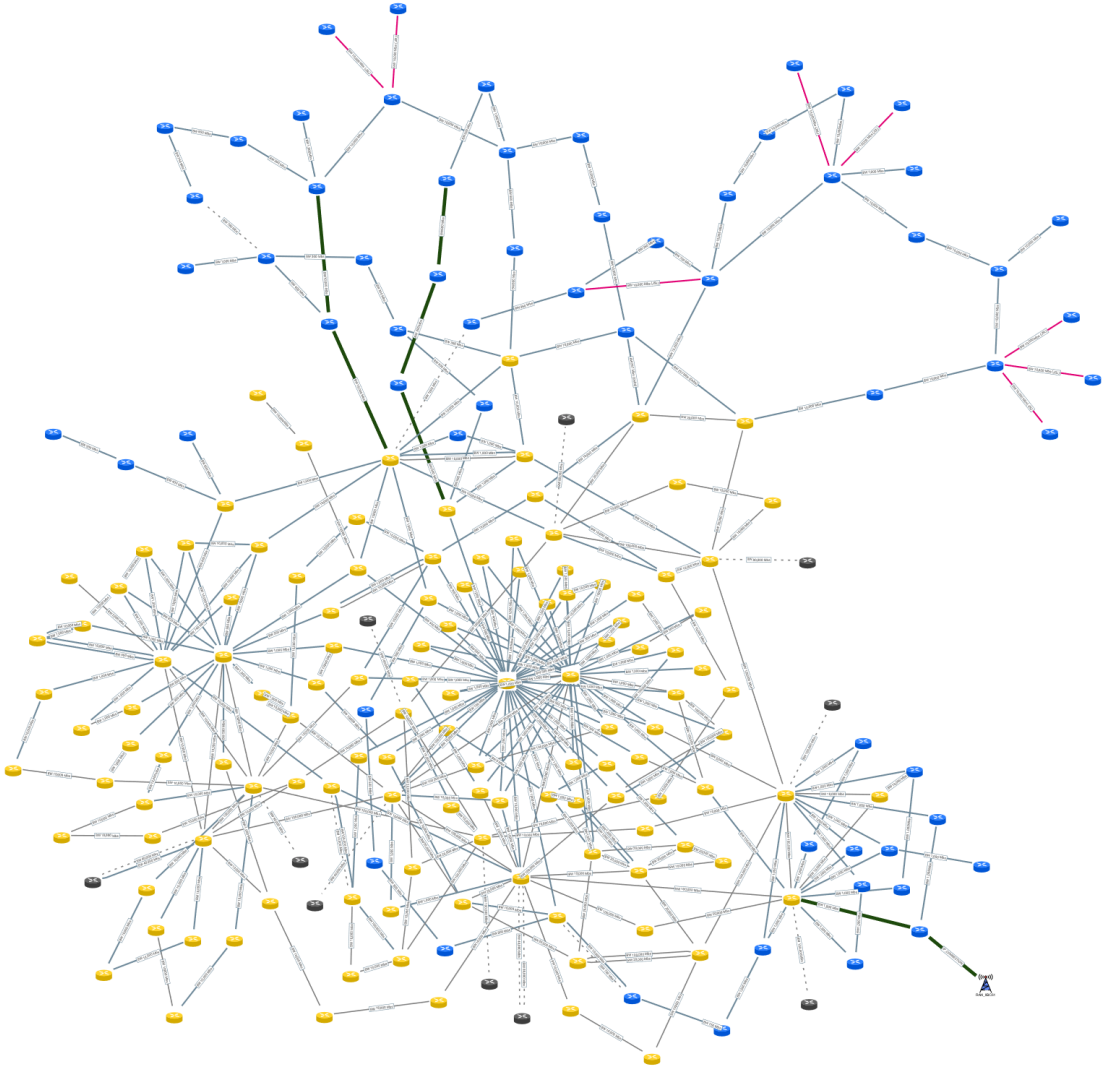
Huge gap between plans and reality

sopra steria

Digital Twin of a Telco Network

We can only optimize, what we can model

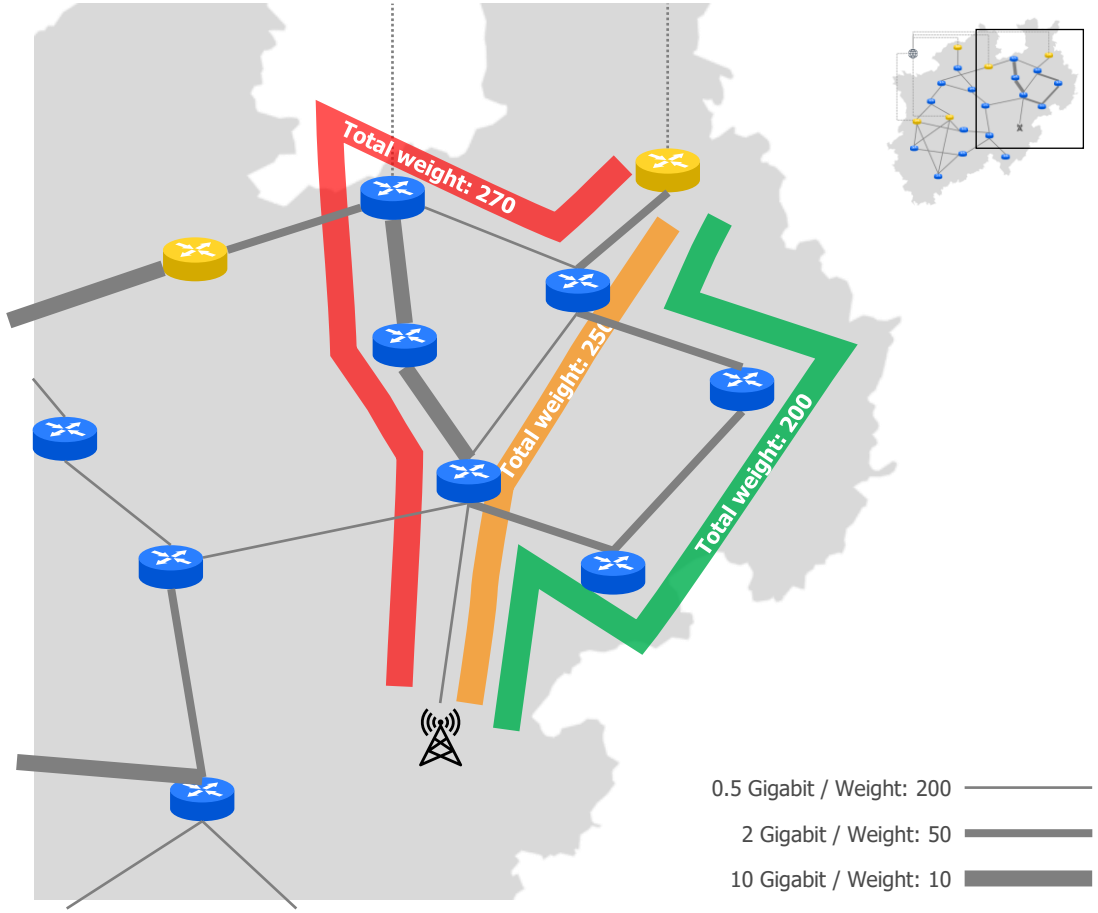




Source (left): Seilov, Sh & Kuzbayev, A & Seilov, A. & Shyngisov, D. & Goikhman, V. & Levakov, A. & Sokolov, Nikolai & Zhursinbek, Y.. (2021). The Concept of Building a Network of Digital Twins to Increase the Efficiency of Complex Telecommunication Systems. Complexity. 2021. 1-9. 10.1155/2021/9480235.

Traffic routing is based on graph algorithms

Example of traffic taking weighted shortest path



Why care about routing paths?

End-to-end traffic intelligence

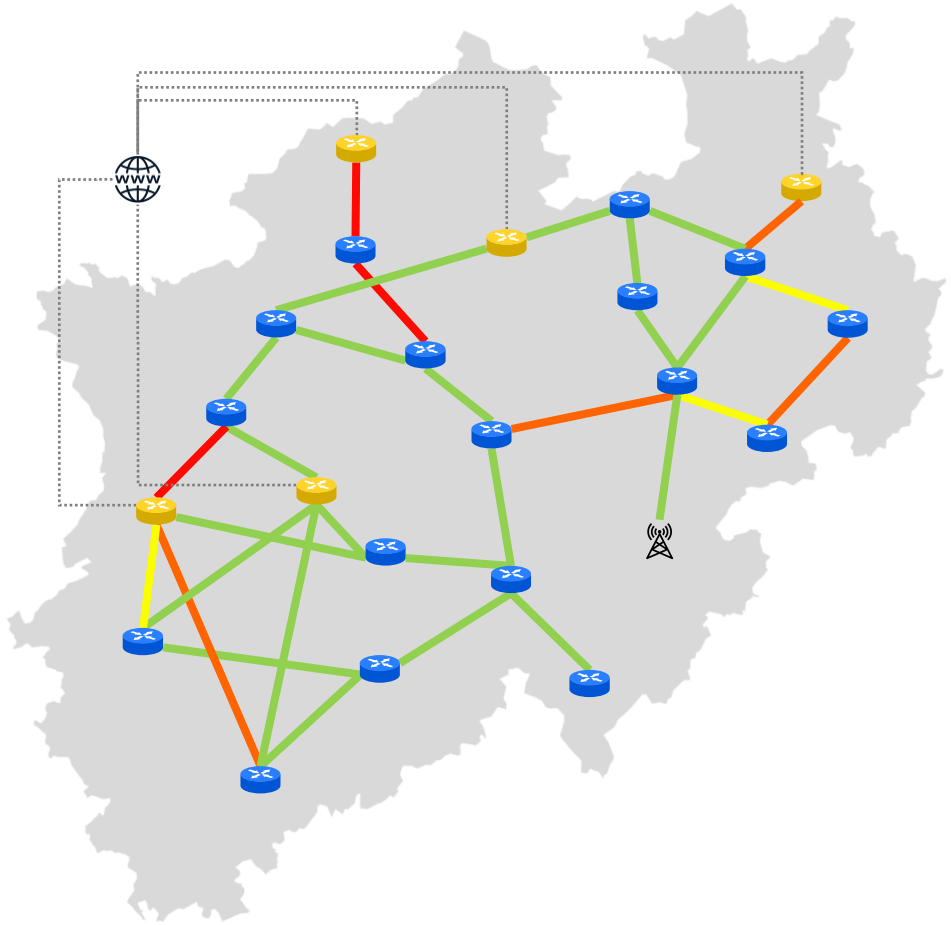
Identification of all possible **problem sources**

Starting point for **network simulation**

sopra steria

Traffic optimization in practice

5 possible paths for each end point…



Simulated annealing solves optimization challenge

800bn
possibilities

Task: assign yellow backbone router to blue router so that **utilization is minimized**
Possible combinations**:** $5^{17} = 800\text{bn}$

~25y
to calculate all possibilities

Brute force calculation of all combinations would take ~25 years at 1ms per configuration

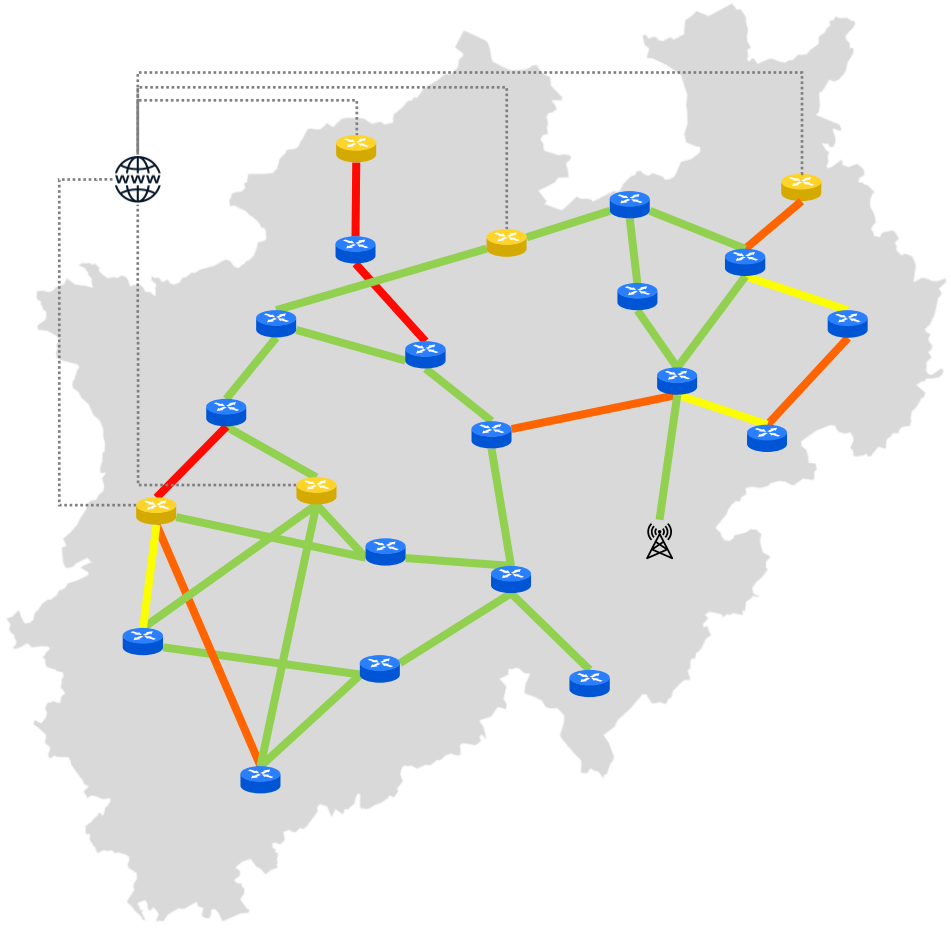
~60s
to find good optimum

Simulated annealing finds (an approximated) optimal solution within 60 s after a few thousand tested combinations

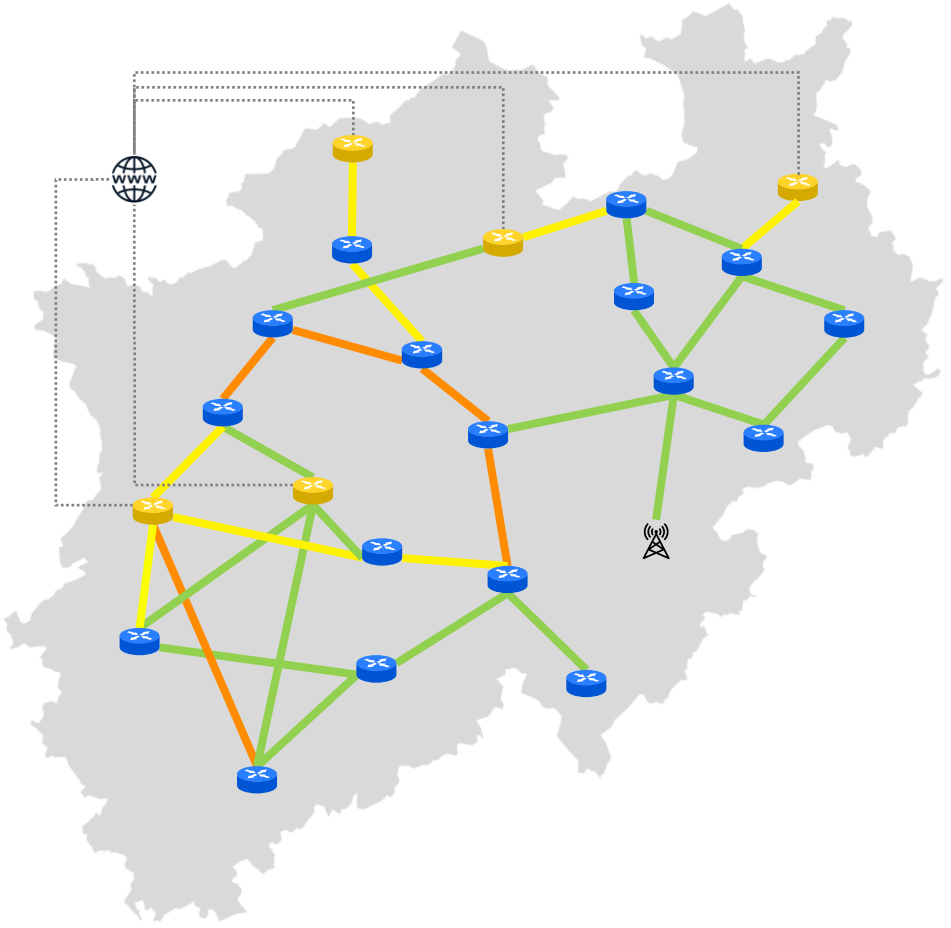
sopra steria

Optimization reduces peak utilization

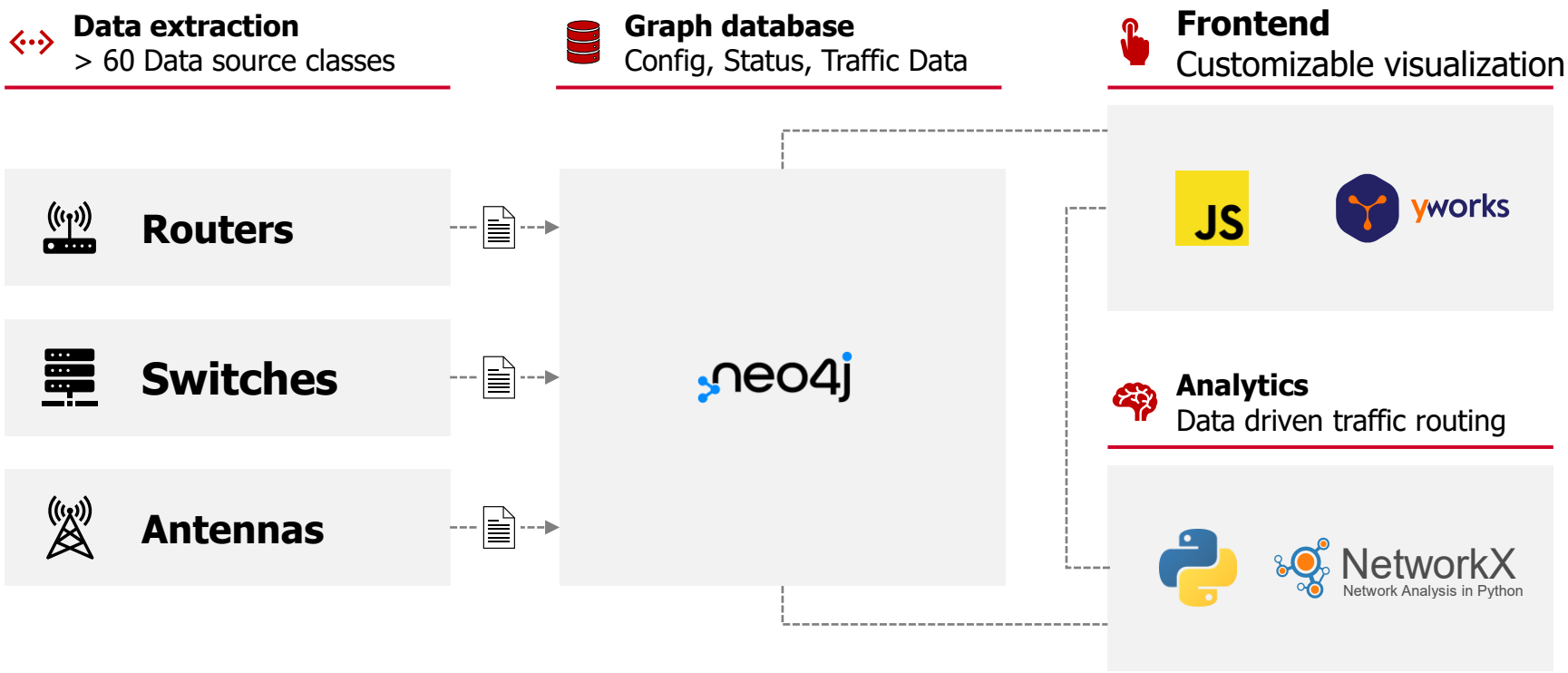
Unoptimized network with overloaded connections



Optimized network has no overloaded connections

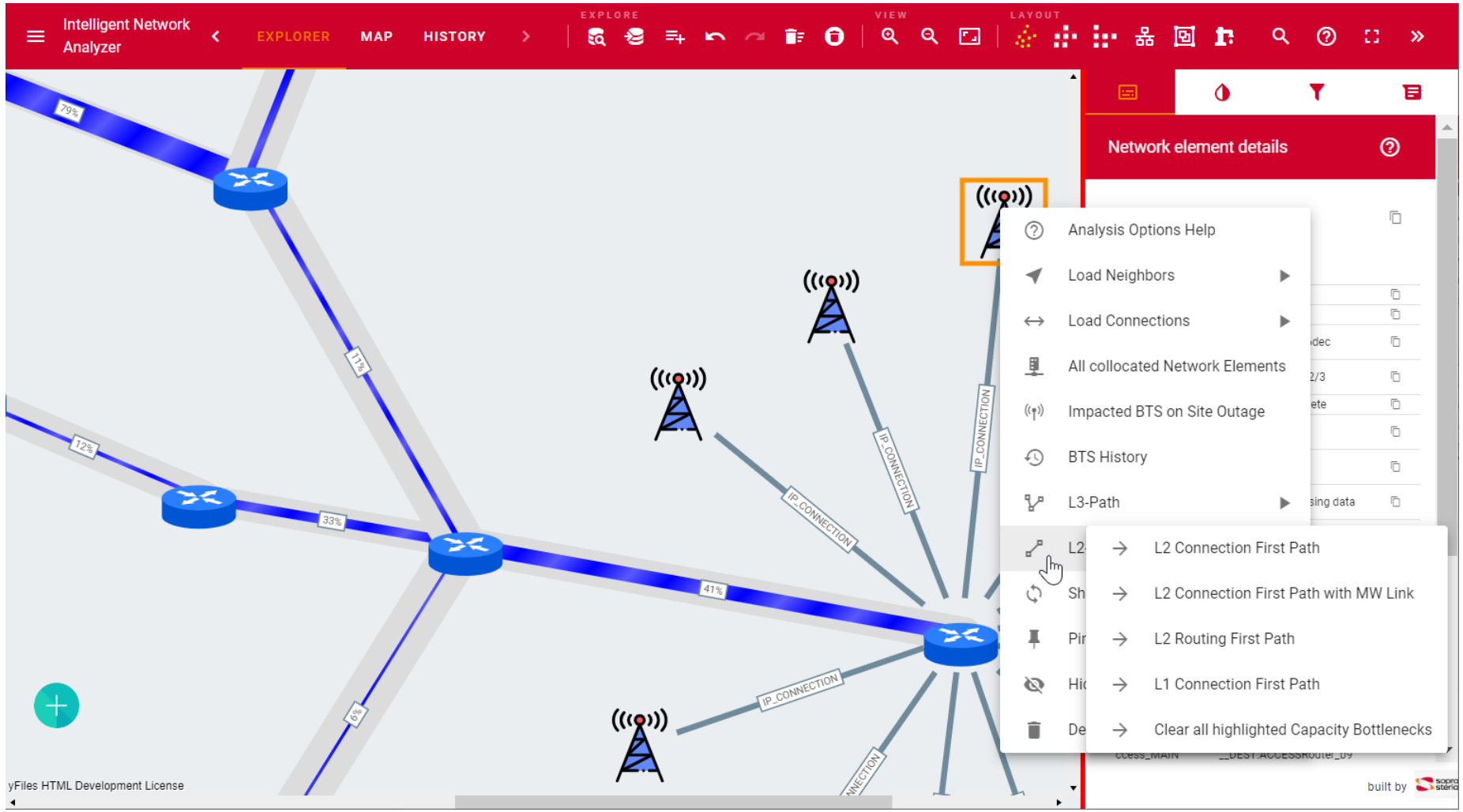


Intelligent Network Analyzer



~400k Devices | ~ 1,5m Connections

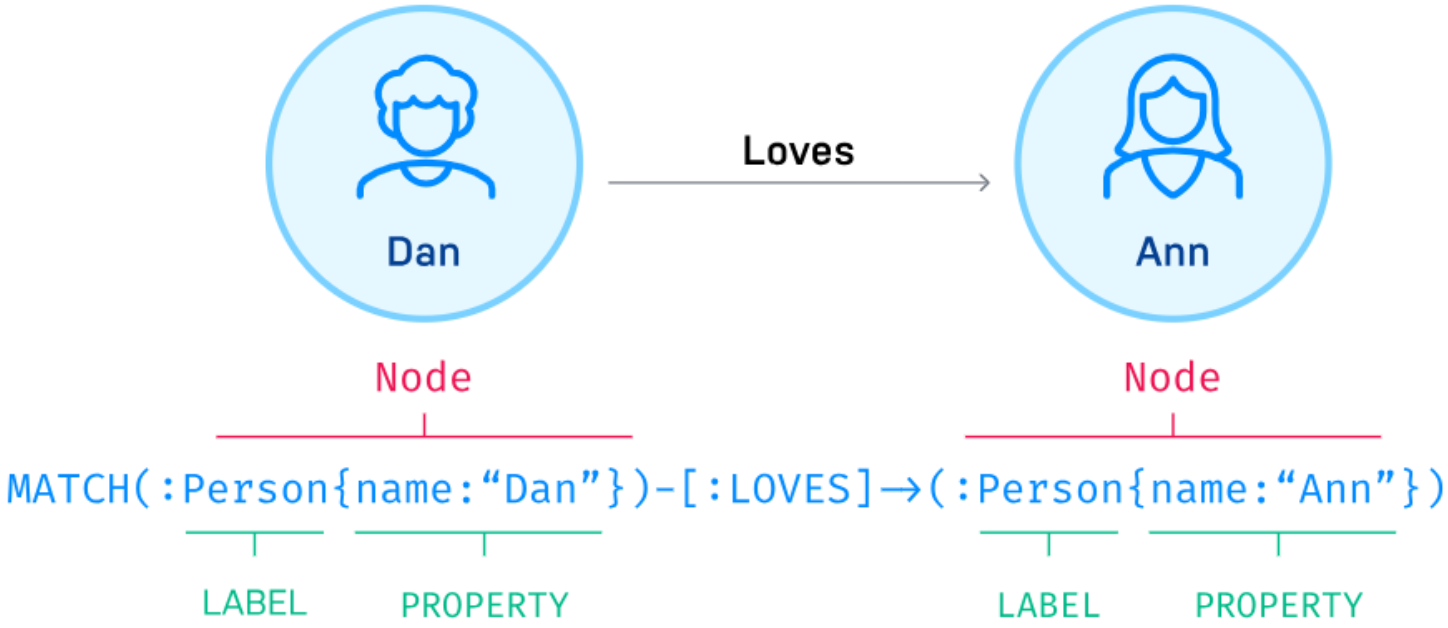
Intelligent Network Analyzer



sopra steria

Neo4J | Cypher

- **Neo4J** is a graph database management system, which uses graph structures with nodes, edges, and properties to represent and store data.
- It efficiently manages and stores relationships between data points, making it ideal for complex queries that involve deep relationships.
- **Cypher** is a declarative graph query language that allows users to efficiently and intuitively query and update graph data by focusing on what to retrieve rather than how to retrieve it.



Neo4J / Cypher

Cypher

```
MATCH (p:Product)-[:CATEGORY]→(l:ProductCategory)-[:PARENT*0..]→(:ProductCategory {name:"Dairy Products"})
RETURN p.name
```

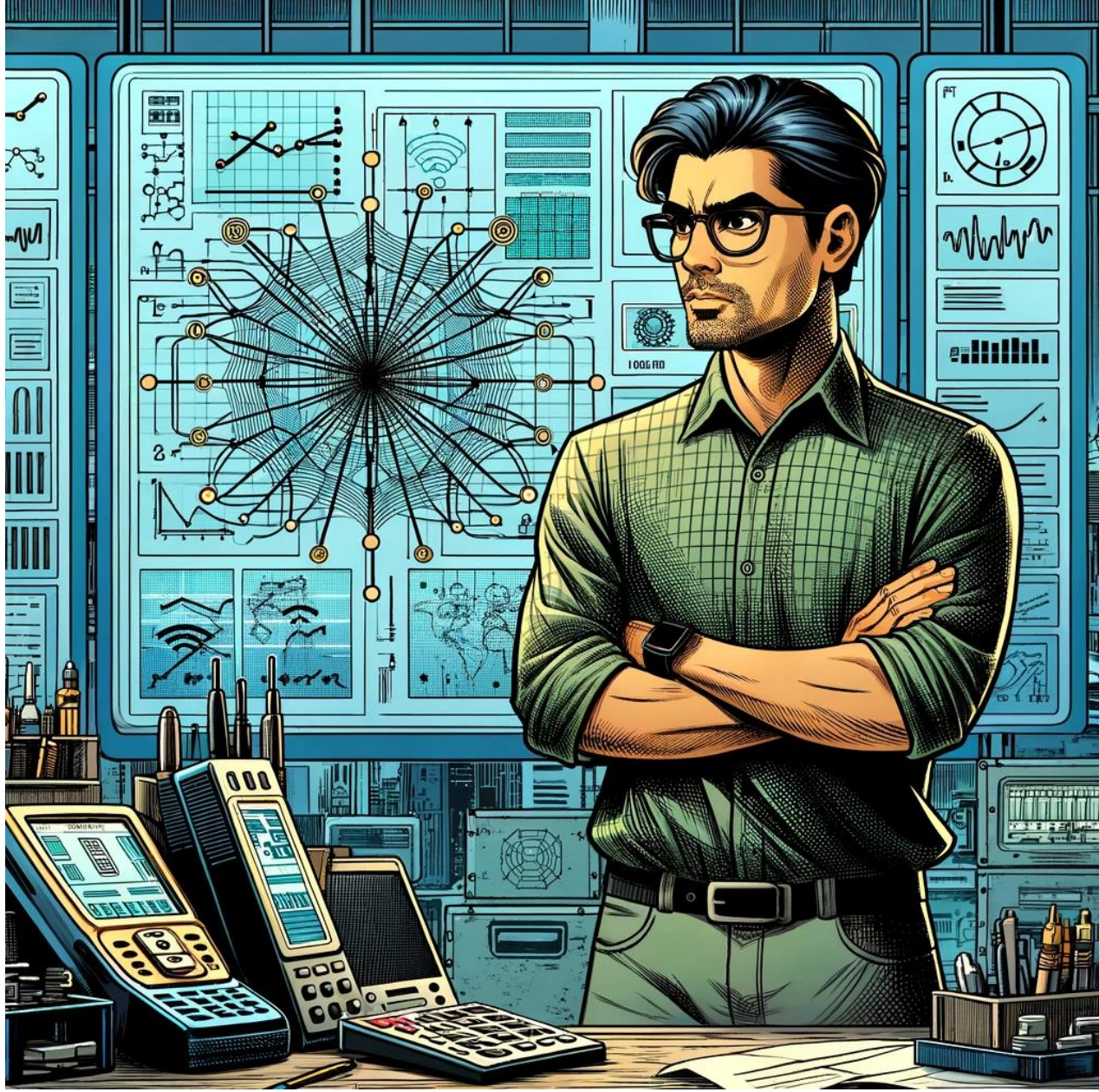
SQL

```
SELECT p.ProductName
FROM Product AS p
JOIN ProductCategory pc ON (p.CategoryID = pc.CategoryID AND pc.CategoryName = "Dairy Products")

JOIN ProductCategory pc1 ON (p.CategoryID = pc1.CategoryID)
JOIN ProductCategory pc2 ON (pc1.ParentID = pc2.CategoryID AND pc2.CategoryName = "Dairy Products")

JOIN ProductCategory pc3 ON (p.CategoryID = pc3.CategoryID)
JOIN ProductCategory pc4 ON (pc3.ParentID = pc4.CategoryID)
JOIN ProductCategory pc5 ON (pc4.ParentID = pc5.CategoryID AND pc5.CategoryName = "Dairy Products");
```

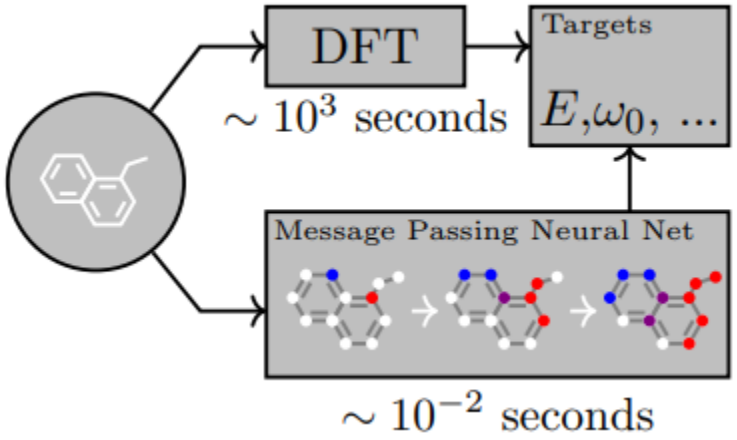
Hands-On Time

- We will use Cypher to explore and analyze network data in a Neo4J graph database
- The link to the Notebook can be found in the Github Repo:

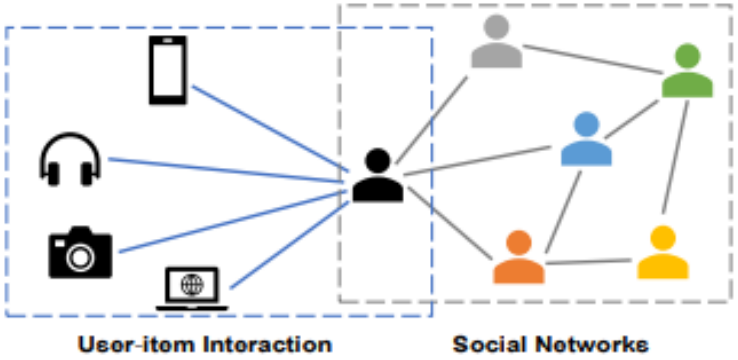
https://github.com/MalteWagner/ml_prague



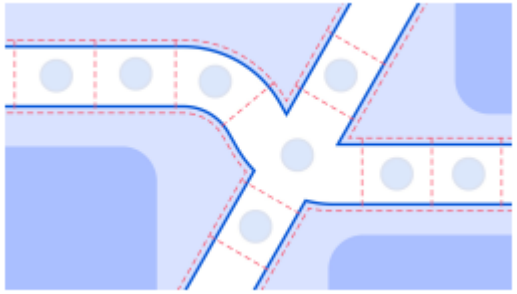
Introduction to GNNs

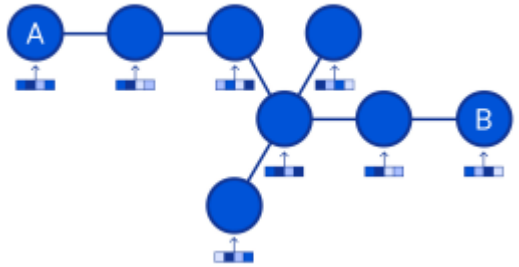


https://arxiv.org/pdf/1704.01212.pdf

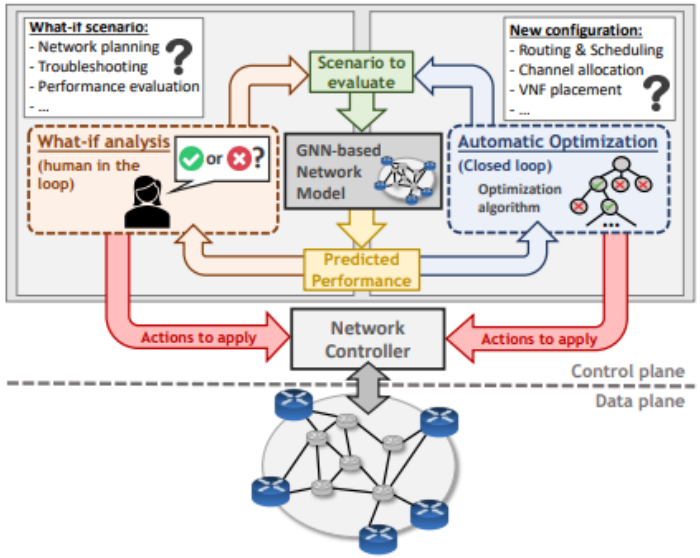


https://arxiv.org/pdf/2109.12843.pdf





https://arxiv.org/pdf/2108.11482.pdf



https://arxiv.org/pdf/2112.14792.pdf

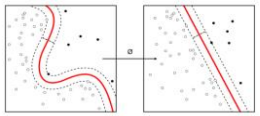
sopra steria

Introduction to GNNs

- GNNs are to relational data, what RNNs are to sequences and CNNs are to image/video data
- Suited for unfeasible simulations on large graphs – stochastic digital twin of a network
- Network modeling is difficult for RNNs and MLPs in a sense that they struggle to evaluate the impact of link failures on the network – if not seen sufficiently in training

  → GNNs provide better predictions based on structure
- MLPs struggle to generalize to unseen topologies
- RNNs can capture path sequences but can not handle circular dependencies in networks
- GNNs keep good generalization capabilities as long as the spectral properties of graphs are like those seen during training
- GNNs come in a wide variety of architectures, for an overview: Survey on Graph Neural Networks

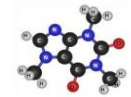
Type of NN	Information Structure
Fully Connected NN	Arbitrary
Convolutional NN	Spatial
Recurrent NN	Sequential
Graph NN	Relational



Generic classification, non-linear regression

Images and video

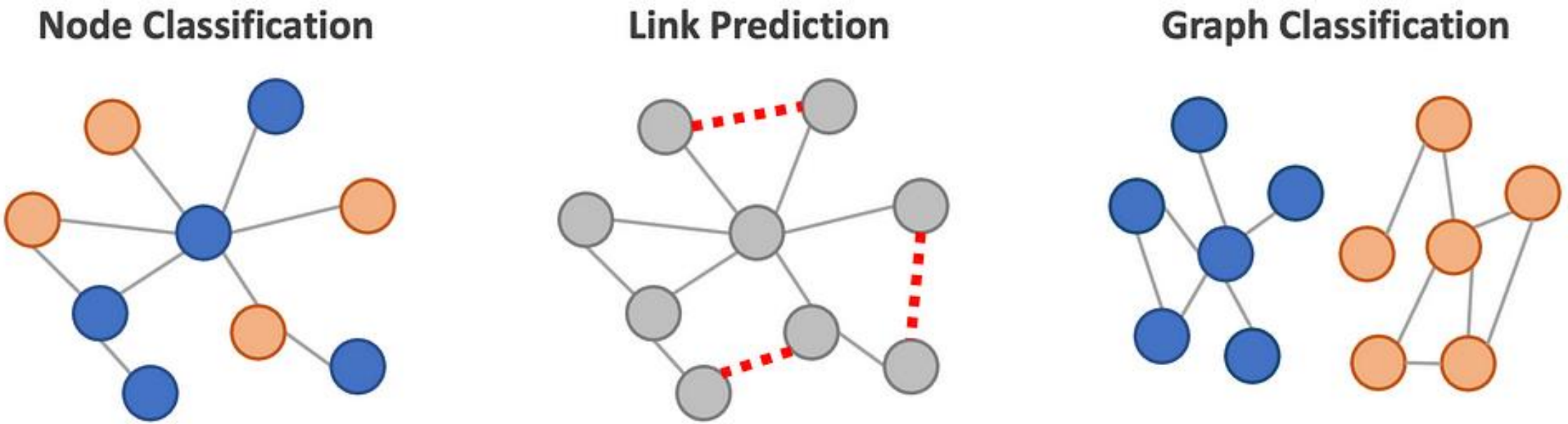
Text and voice



Graphs (molecules, maps, networks)

Source: José Suárez-Varela – GNN Challenge 2020

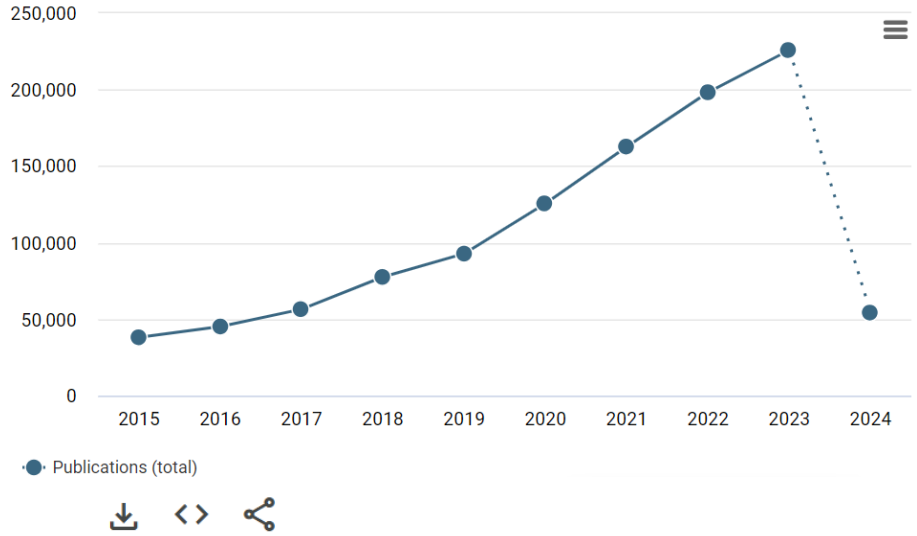
Problem framing



Source: https://towardsdatascience.com/graph-neural-networks-with-pyg-on-node-classification-link-prediction-and-anomaly-detection-14aa38fe1275

GNN Research Interest

- GNN have risen in popularity over the last 10 years, both in public and in research
- Almost 5x as many papers published on the topic than 2015
- Peak on Google Trends



sopra steria

General Problem

- Neural Networks and CNNs especially assume input to be geometrically structured. Graphs are not.

  →Structure should be considered as input of the ML model

Example: The number of neighbors – in contrast to CNNs – can vary in graphs.

e.g. Each pixel in an image has eight neighbors

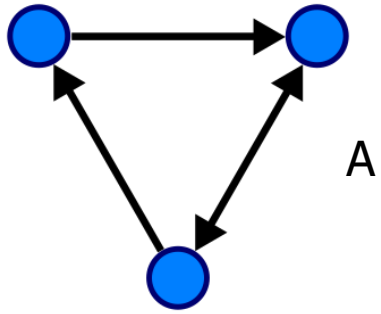
- Algorithms used on array-like structure like convolutions using kernels or pooling are not defined for graphs

Challenge: Method to learn from the structure itself, not just the data

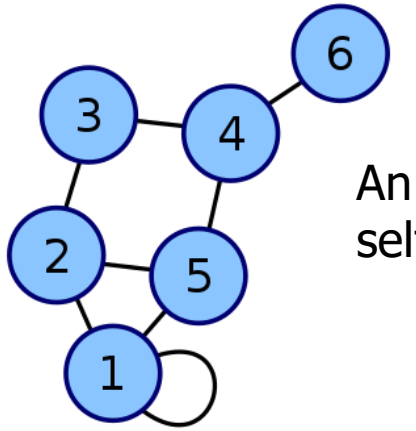
Graph Theory

- A **Graph** is a structure that relates some objects in some kind of way to each other

- **Node:** Object, e.g. Routers, Switches, BTS and other network elements
- **Edge:** Relation between Objects, e.g. R2R Connections (physical) or IP Connection (digital)
- **Node Degree:** Number of *neighbors* a node has
- **Self loops:** Nodes that are connected to themselves
- **Isolated Nodes:** Nodes that are not connected to other parts of the graph via edges
- **Directed/undirected graphs:** A graph is called directed, if the edges have directions.



A directed graph



An undirected graph with a self loop at node 1

Notation

Graph G = (V,E)	where V are the labeled Nodes {1,…,N} and E are the edge set E = V x V
Feature Matrix X $\in R^{NxD_N}$	where D_N is the number of features
Adjacency Matrix A $\in R^{NxN}$	where, a_{ij} = 0 if edge doesn't exist, 1 else. For i = j typically 1.
Degree Matrix D $\in R^{NxN}$	diagonal describes number of neighbors, 0 elsewhere
Neighborhood $\mathcal{N}$(u)	All nodes directly connected to node u

Capital letters denote general concept (Node, Edge etc.), lowercase denotes *specific* ones.

E.g. u,v ∈ V or e_{ij} ∈ E

Message Passing

- Message passing (MP) is used to aggregate information from neighboring nodes
- Almost all GNNs use message passing as their way to learn from the structure
- During the message passing, each node gets an embedding, which gets updated depending on the neighboring nodes
- Message passing consists of two main steps
  - 1. AGGREGATE: Aggregate the hidden embedding of each neighboring node
  - 2. UPDATE: Updates the selected node, using a learnable weight matrix and some nonlinearity

Hidden embedding $\mathbf{h_u^{(k+1)}} = \text{UPDATE}(h_u^{(k)}, \text{AGGREGATE}(h_v^{(k)}))$

$\forall v \in N(u)$ where N(u) are all the neighbors from u and k is the k-th iteration

$h_u^{(0)} = x_u$ The first embedding of node u is set equal to the features of node u

Message Passing

- Message passing (MP) is used to aggregate information from neighboring nodes
- Almost all GNNs use message passing as their way to learn from the structure
- During the message passing, each node gets an embedding, which gets updated depending on the neighboring nodes
- Message passing consists of two main steps
  - 1. AGGREGATE: Aggregate the hidden embedding of each neighboring node
  - 2. UPDATE: Updates the selected node, using a learnable weight matrix and some nonlinearity

$$\textbf{Hidden embedding } h_u^{(k+1)} = \text{UPDATE}(h_u^{(k)}, \text{AGGREGATE}(h_v^{(k)}))$$

This is called the message

$\forall v \in N(u)$ where N(u) are all the neighbors from u and k is the k-th iteration

$h_u^{(0)} = x_u$ The first embedding of node u is set equal to the features or labels of node u

Message Passing

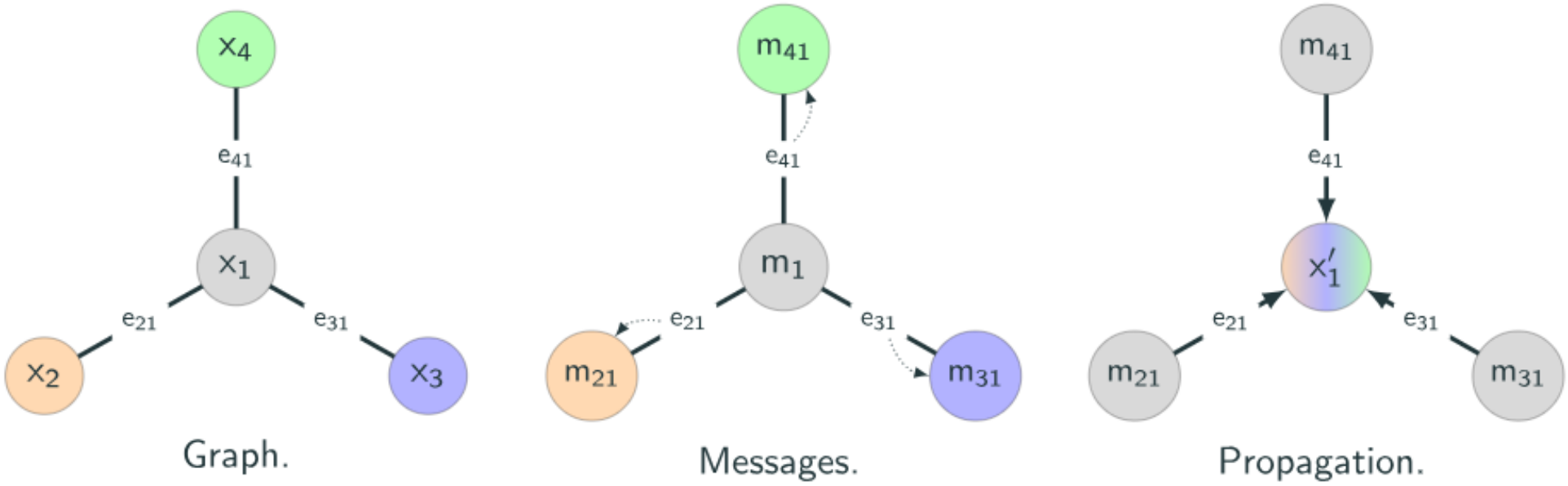
$$\textbf{Hidden embedding } h_u^{(k+1)} = \text{UPDATE}(h_u^{(k)}, m_{N(u)})$$

Remarks:

- AGGREGATE and UPDATE both need to be differentiable functions
- AGGREGATE needs to be permutation invariant (typically it‘s just the sum) to ensure, that the order in which nodes get aggregated doesn‘t matter
- UPDATE needs to introduce learning parameters as well as the non-linearity
- $h_u^{(k)}$ doesn’t need to be included into the update function

Visual of Message Passing

Message passing for node x_1



Source: Message Passing - J. Gilmer et al., "Neural message passing for quantum chemistry," 2017

From Message Passing towards GNNs

Hidden embedding $h_u^{(k+1)}$ = UPDATE($h_u^{(k)}$,$m_{N(u)}$)

Basic GNN:

Trainable parameters in W and b, nonlinearity (e.g. ReLU) in σ.

W ∈ $R^{D_N x D_N}$, b ∈ R^{D_N}

$$\mathbf{h}_u^{(k)} = \sigma\left(\mathbf{W}_{\text{self}}^{(k)}\mathbf{h}_u^{(k-1)} + \mathbf{W}_{\text{neigh}}^{(k)} \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v^{(k-1)} + \mathbf{b}^{(k)}\right)$$

$$\mathbf{m}_{\mathcal{N}(u)} = \sum_{v \in \mathcal{N}(u)} \mathbf{h}_v,$$

$$\text{UPDATE}(\mathbf{h}_u, \mathbf{m}_{\mathcal{N}(u)}) = \sigma\left(\mathbf{W}_{\text{self}}\mathbf{h}_u + \mathbf{W}_{\text{neigh}}\mathbf{m}_{\mathcal{N}(u)}\right)$$

Graph Convolutional Network GCN

- GCNs are an improvement on normal message passing
- Principle:
  - 1. Each embedding is initialized with the features of the respective node
  - 2. Message aggregation is performed
  - 3. **The aggregated message is sent through a dense layer of chosen shape**
  - 4. Repeat k-times (k = # of GCN Layers)

- Layers very similar to common basic neural networks

GCN: $$\mathbf{x}_v^{(\ell+1)} = \mathbf{W}^{(\ell+1)} \sum_{w \in \mathcal{N}(v) \cup \{v\}} \frac{1}{c_{w,v}} \cdot \mathbf{x}_w^{(\ell)}$$

GCN by Kipf et al.

What do we have now?

- Formalism to collect structure and node information of a graph
- Message passing, as a way for nodes to communicate their information with each other
- Network layers defined similar to common ML models
- Number of layers as a distance from which nodes can infer information
- Learnable parameters $W^{(k)}$ and $b^{(k)}$ for training

→ Simple Graph Neural Network

Improvements to the basic Model

AGGREGATE:

- Theoretically every permutation invariant function possible like sum, max, median, mean
- Further improvements with special aggregation functions, e.g. symmetrical normalization:

$$m_{N(u)} = \sum_{v \epsilon N(u)} \frac{h_v}{\sqrt{|N(u)||N(v)|}}$$

- Learning of AGGREGATE itself possible with an MLP
- Use sum, but weigh each node individually with a learnable parameter (attention, sometimes referred to as **G**raph **A**ttention **N**etwork)

Remark: *It is possible to build a transformer and reduce it to a GNN. This can make use of existing libraries and techniques at cost of computational complexity*

Improvements to the basic Model

UPDATE:

- In general: Updating the embeddings can be seen as a smoothing in regard to the neighboring nodes
- Main problem: For k approaching infinity, the graph loses its information of neighborhood structures (for realistic graphs, structures are mostly lost at O(log(|V|)))

→ Deep basic GNNs hurt performance, due to so called *oversmoothing*

→ Information becomes washed out

This happens, if the information coming from the surrounding nodes dominates in the UPDATE process compared to the current nodes' information

Hands-On Time

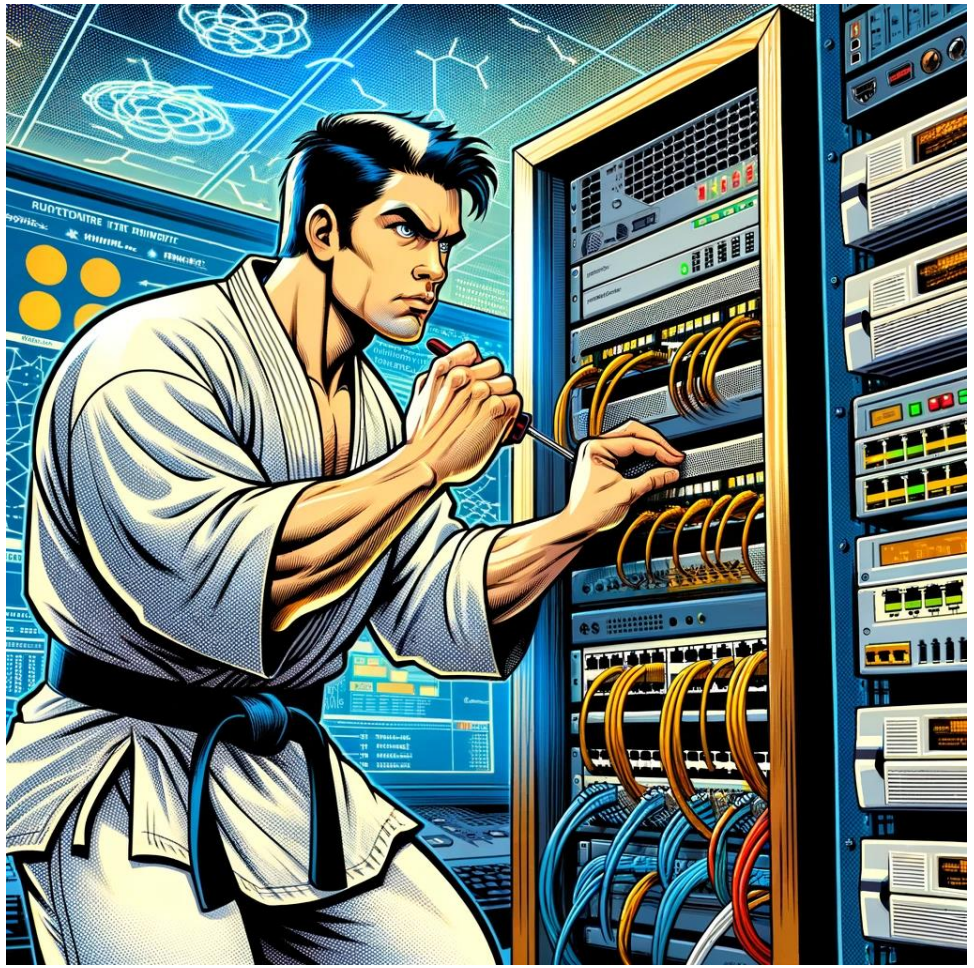
- We will train a GNN to detect communities in Zachary's karate club
- We use an architecture based GCN Layers (Kipf et al. (2017)) implemented with PyTorch Geometric (PyG)
- The link to the Notebook can be found in the Github Repo:

https://github.com/MalteWagner/ml_prague

Back to Telco Networks

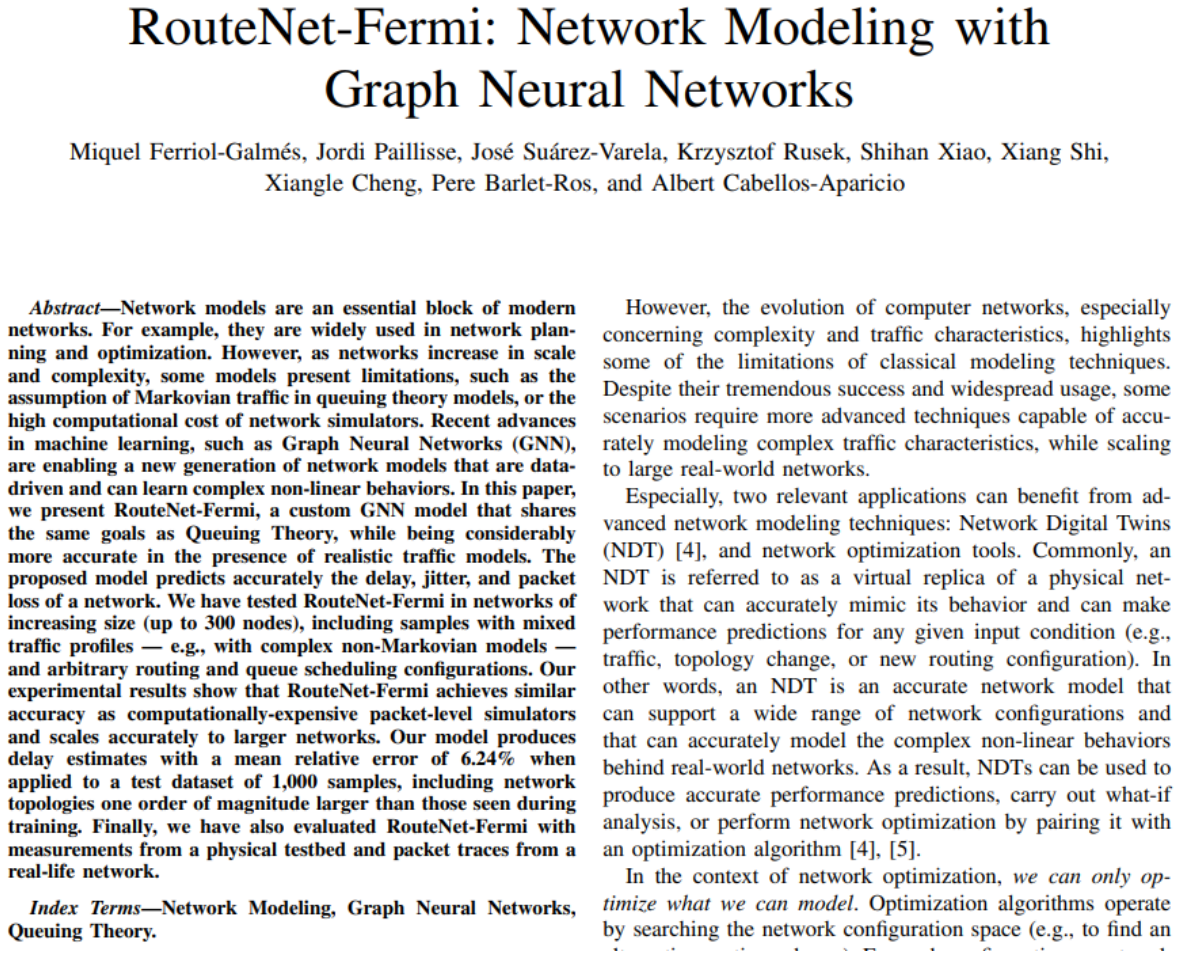
- We saw the basic Idea of GNNs with quite simple graph structure that:
  - was small in size
  - was clean
  - had no node attributes
  - had no edge attributes

- Data from telco networks is the opposite! So, we have a few more challenges to overcome:
  - Data preprocessing
  - Data quality
  - Model architecture
  - Sensitive data!



Introduction to RouteNet

- For the last part of the workshop, we will work with RouteNet, a GNN architecture specifically designed to estimate end-to-end performance metrics in networks
- RoutNet was developed by researchers from the Barcelona Neural Networking Center and is used in different community Challenges. Have a look if you like: GGNet Challenge 2023
- It is based on a custom Tensorflow/Keras implementation
- It was trained on small topologies but can generalize well on larger topologies



RouteNet-Fermi: Network Modeling with Graph Neural Networks

Miquel Ferriol-Galmés, Jordi Paillisse, José Suárez-Varela, Krzysztof Rusek, Shihan Xiao, Xiang Shi, Xiangle Cheng, Pere Barlet-Ros, and Albert Cabellos-Aparicio

***Abstract*—Network models are an essential block of modern networks. For example, they are widely used in network planning and optimization. However, as networks increase in scale and complexity, some models present limitations, such as the assumption of Markovian traffic in queuing theory models, or the high computational cost of network simulators. Recent advances in machine learning, such as Graph Neural Networks (GNN), are enabling a new generation of network models that are data-driven and can learn complex non-linear behaviors. In this paper, we present RouteNet-Fermi, a custom GNN model that shares the same goals as Queuing Theory, while being considerably more accurate in the presence of realistic traffic models. The proposed model predicts accurately the delay, jitter, and packet loss of a network. We have tested RouteNet-Fermi in networks of increasing size (up to 300 nodes), including samples with mixed traffic profiles — e.g., with complex non-Markovian models — and arbitrary routing and queue scheduling configurations. Our experimental results show that RouteNet-Fermi achieves similar accuracy as computationally-expensive packet-level simulators and scales accurately to larger networks. Our model produces delay estimates with a mean relative error of 6.24% when applied to a test dataset of 1,000 samples, including network topologies one order of magnitude larger than those seen during training. Finally, we have also evaluated RouteNet-Fermi with measurements from a physical testbed and packet traces from a real-life network.**

***Index Terms*—Network Modeling, Graph Neural Networks, Queuing Theory.**

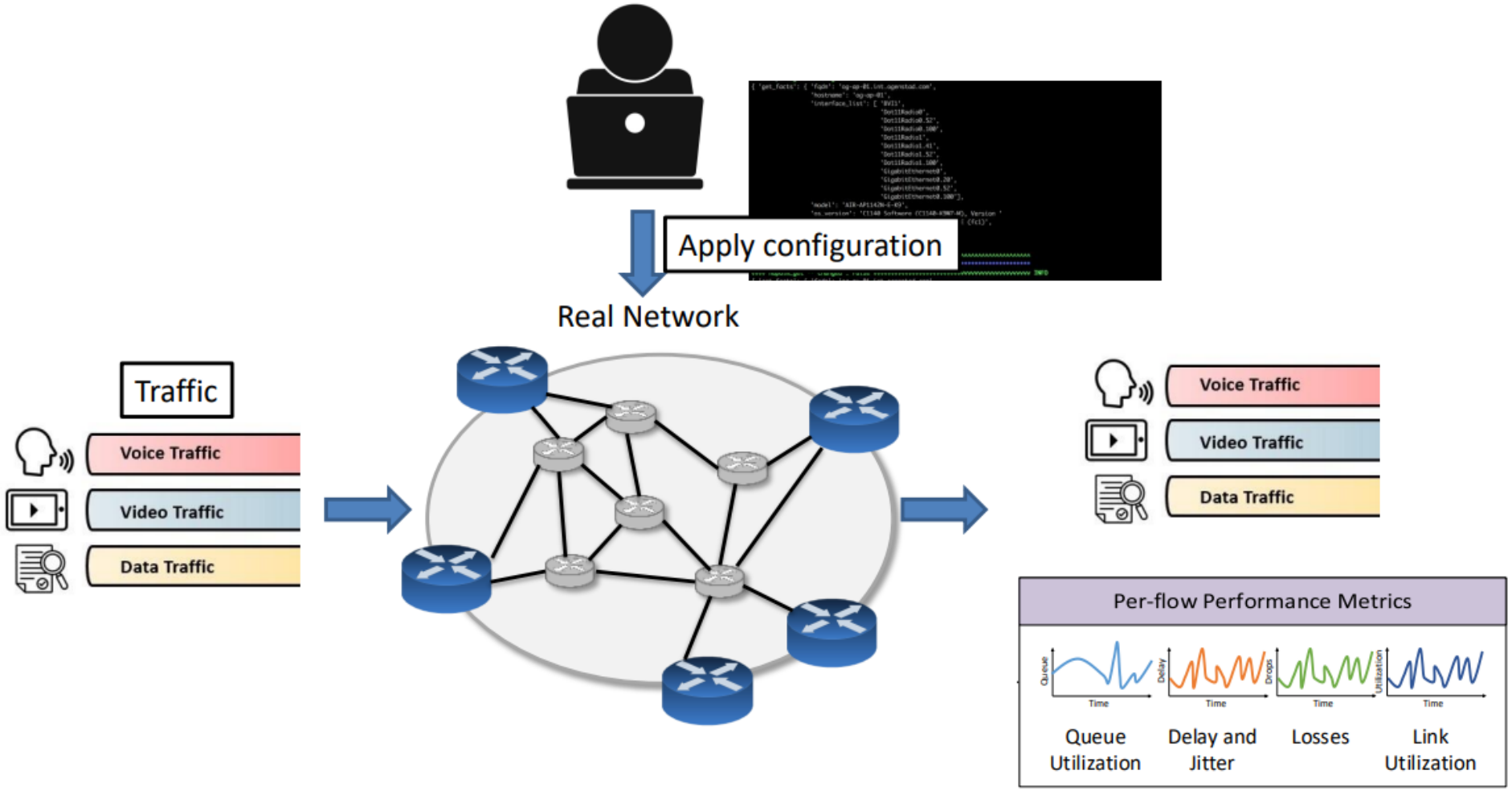
However, the evolution of computer networks, especially concerning complexity and traffic characteristics, highlights some of the limitations of classical modeling techniques. Despite their tremendous success and widespread usage, some scenarios require more advanced techniques capable of accurately modeling complex traffic characteristics, while scaling to large real-world networks.

Especially, two relevant applications can benefit from advanced network modeling techniques: Network Digital Twins (NDT) [4], and network optimization tools. Commonly, an NDT is referred to as a virtual replica of a physical network that can accurately mimic its behavior and can make performance predictions for any given input condition (e.g., traffic, topology change, or new routing configuration). In other words, an NDT is an accurate network model that can support a wide range of network configurations and that can accurately model the complex non-linear behaviors behind real-world networks. As a result, NDTs can be used to produce accurate performance predictions, carry out what-if analysis, or perform network optimization by pairing it with an optimization algorithm [4], [5].

In the context of network optimization, *we can only optimize what we can model*. Optimization algorithms operate by searching the network configuration space (e.g., to find an

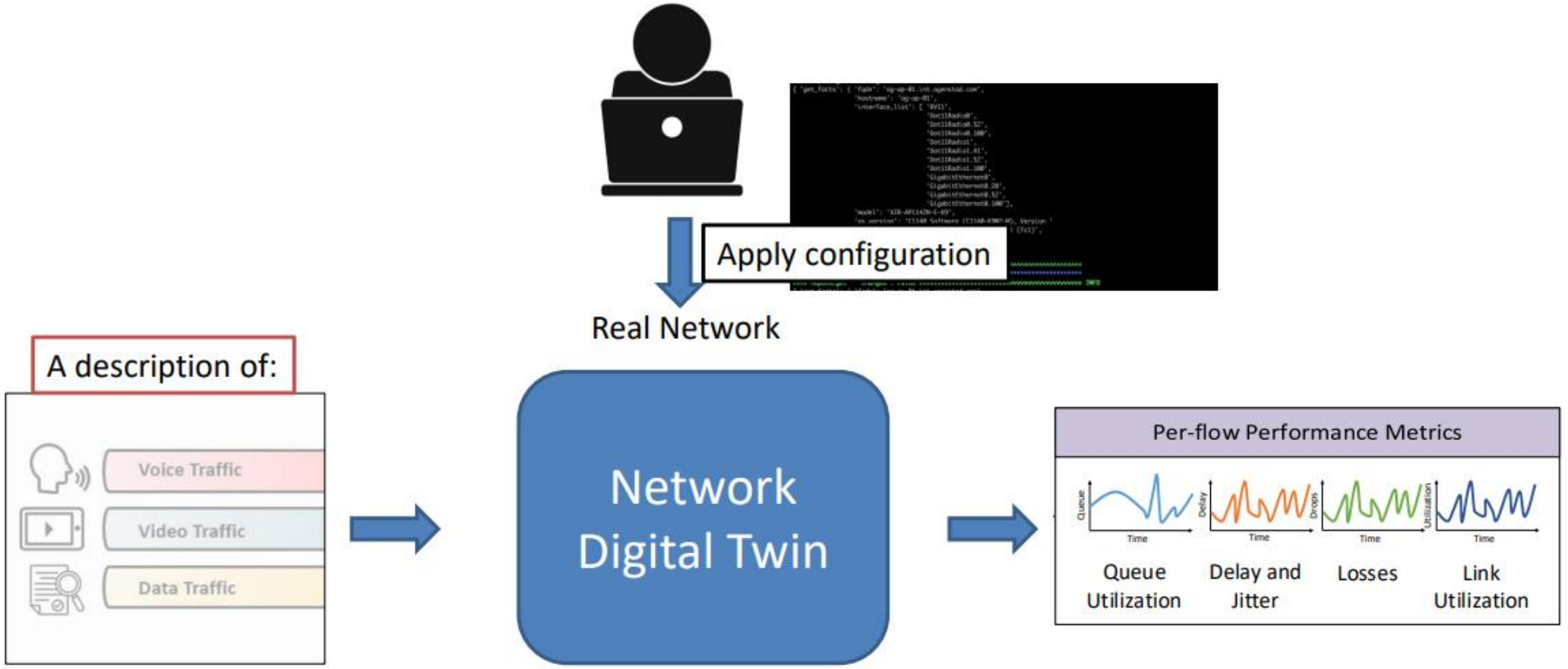
J. Suárez-Varela, et al., "The Graph Neural Networking challenge: a world-wide competition for education in AI/ML for networks," ACM SIGCOMM Computer Communication Review, vol. 51, no. 3, pp. 9–16, 2021.

Introduction to RouteNet | Digital Twin



Source: https://bnn.upc.edu/challenge/gnnet2023/

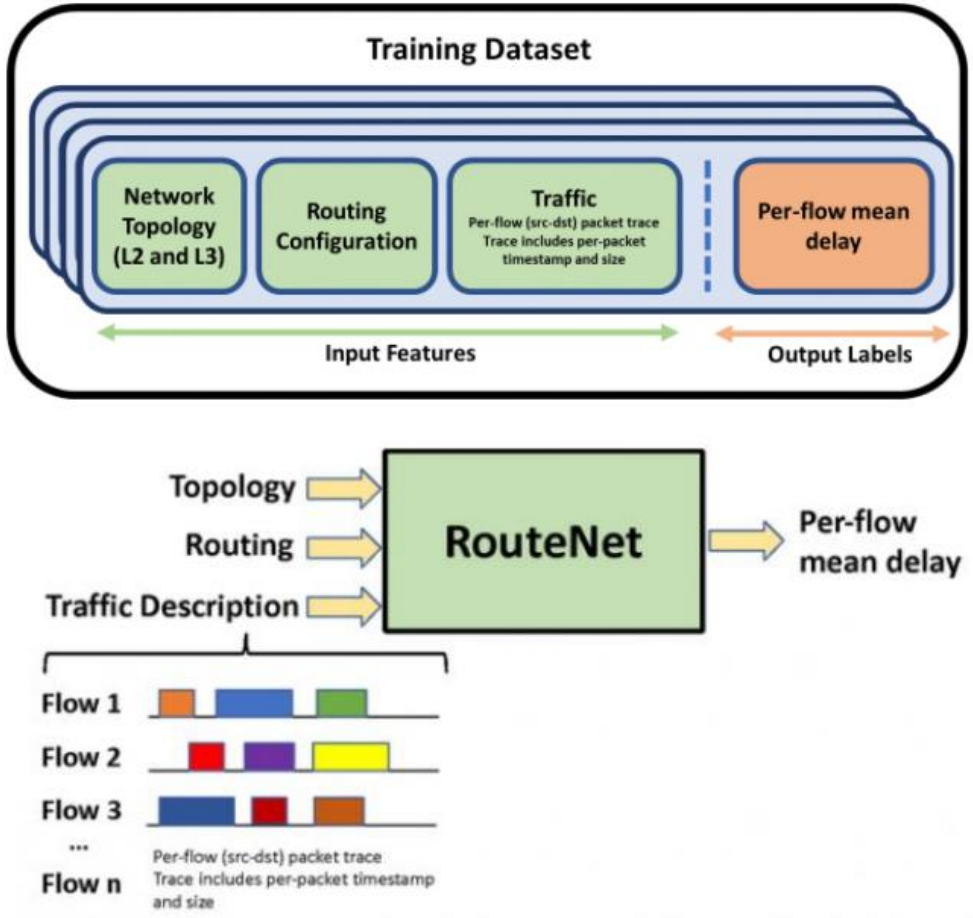
Introduction to RouteNet | Digital Twin



Source: https://bnn.upc.edu/challenge/gnnet2023/

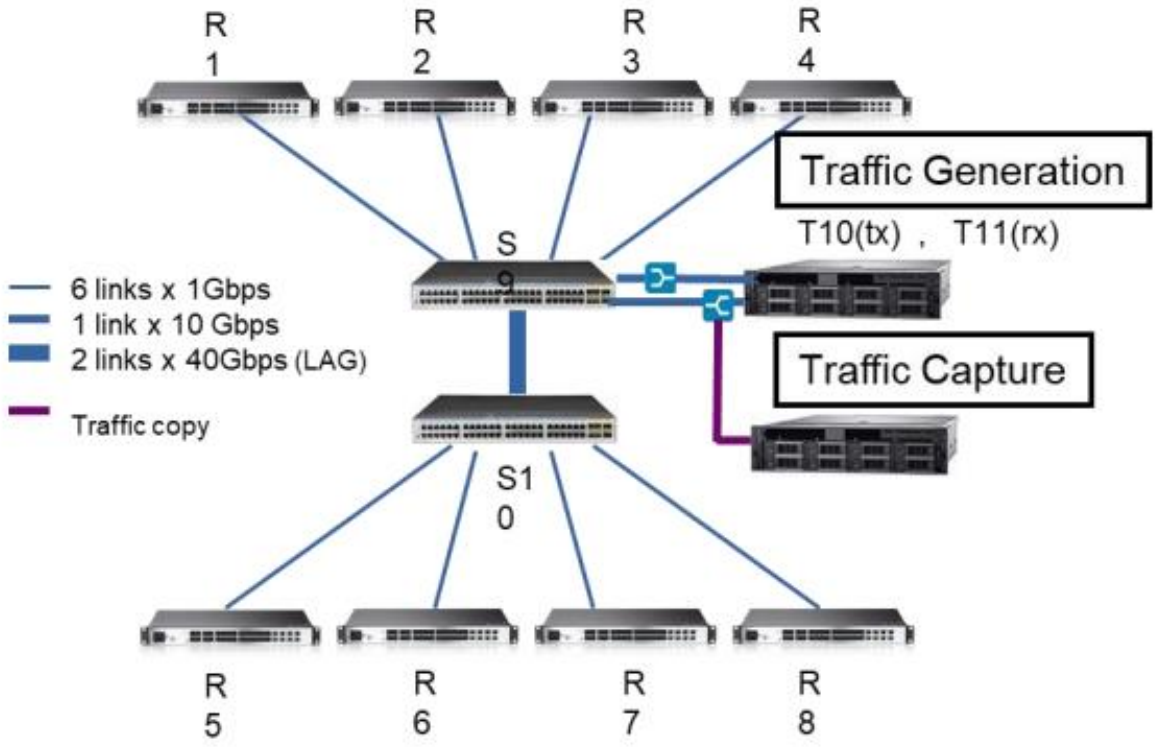
Introduction to RouteNet | Data

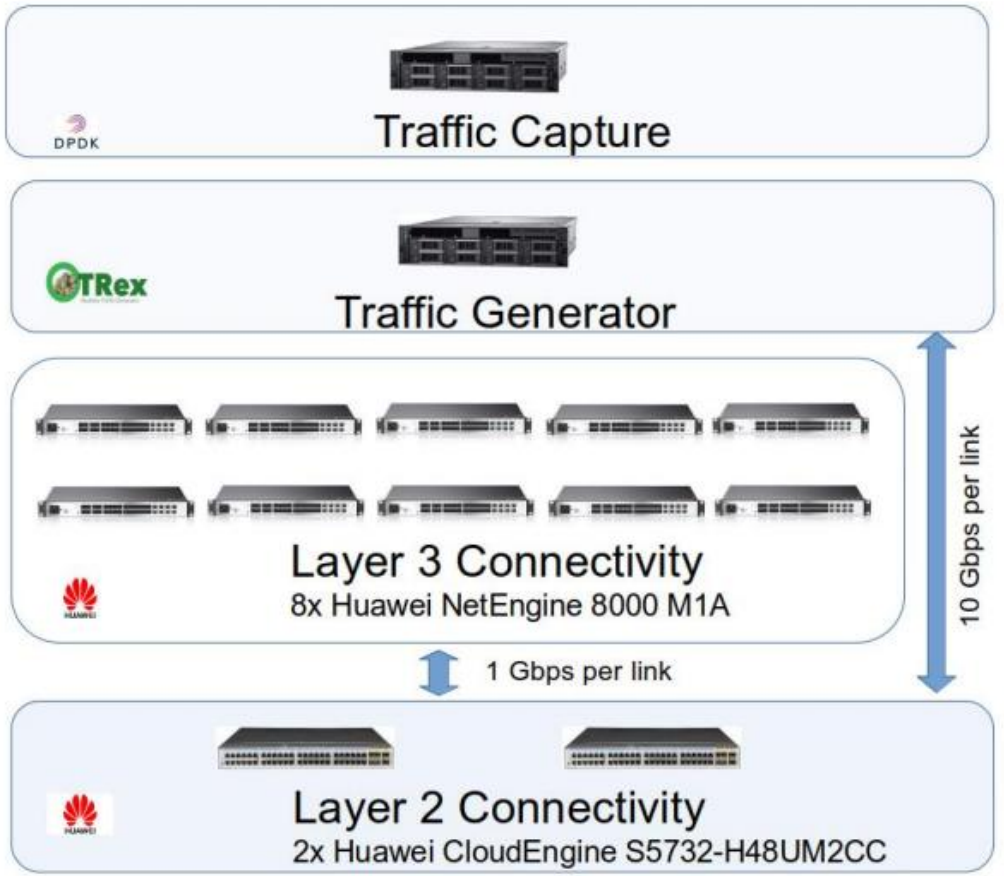
- Dataset includes ≈4.200 samples
  - Flow information: distribution parameters, number of packets sent, packet size
  - Path information: network routing and physical path
  - Topology information (router connections, link capacity)
  - Packet-level traces
  - Performance information: average flow and path delay, jitter, packet loss rates



Source: https://bnn.upc.edu/challenge/gnnet2023/

Introduction to RouteNet | Data

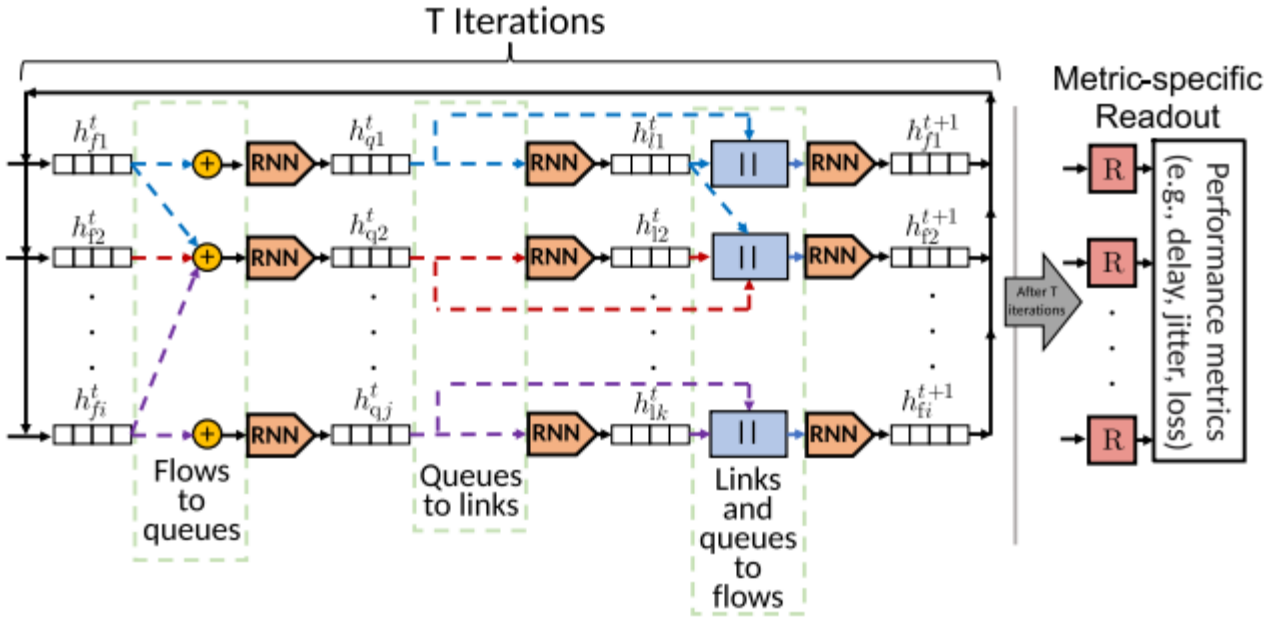




Source: https://bnn.upc.edu/challenge/gnnet2023/

Introduction to RouteNet I Model

- RouteNet-(Fermi) has a three-stage message passing algorithm that combines the states of flows, queues and links, and updates them iteratively.
- States are combined to estimate flow-level delays, jitters, and packet loss.



Input: $\mathcal{F}$, Q, $\mathcal{L}$, $\boldsymbol{x}_f$, $\boldsymbol{x}_q$, $\boldsymbol{x}_l$
Output: $\hat{y}_{f_d}$

1: **for each** $f \in \mathcal{F}$ **do** $\boldsymbol{h}_f^0 \leftarrow HS_f(\boldsymbol{x}_f)$
2: **for each** $q \in Q$ **do** $\boldsymbol{h}_q^0 \leftarrow HS_q(\boldsymbol{x}_q)$
3: **for each** $l \in \mathcal{L}$ **do** $\boldsymbol{h}_l^0 \leftarrow HS_l(\boldsymbol{x}_l)$

4: **for** t = 0 to T-1 **do** ▷ Message Passing Phase
5: **for each** $f \in \mathcal{F}$ **do** ▷ Message Passing on Flows
6: $\Theta([\cdot,\cdot]) \leftarrow FRNN(\boldsymbol{h}_f^t, [\cdot,\cdot])$ ▷ FRNN Initialization
7: **for each** $(q,l) \in f$ **do**
8: $\boldsymbol{h}_{f,l}^t \leftarrow \Theta([\boldsymbol{h}_q^t, \boldsymbol{h}_l^t])$ ▷ Flow: Aggr. and Update
9: $\widetilde{m}_{f,q}^{t+1} \leftarrow \boldsymbol{h}_{f,l}^t$ ▷ Flow: Message Generation
10: $\boldsymbol{h}_f^{t+1} \leftarrow \boldsymbol{h}_{f,l}^t$
11: **for each** $q \in Q$ **do** ▷ Message Passing on Queues
12: $M_q^{t+1} \leftarrow \sum_{f \in Q_f(q)} \widetilde{m}_{f,q}^{t+1}$ ▷ Queue: Aggregation
13: $\boldsymbol{h}_q^{t+1} \leftarrow U_q(\boldsymbol{h}_q^t, M_q^{t+1})$ ▷ Queue: Update
14: $\widetilde{m}_q^{t+1} \leftarrow \boldsymbol{h}_q^{t+1}$ ▷ Queue: Message Generation
15: **for each** $l \in \mathcal{L}$ **do** ▷ Message Passing on Links
16: $\Psi(\cdot) \leftarrow LRNN(\boldsymbol{h}_l^t, \cdot)$ ▷ LRNN Initialization
17: **for each** $q \in L_q(l)$ **do**
18: $\boldsymbol{h}_l^t \leftarrow \Psi(\widetilde{m}_q^{t+1})$ ▷ Link: Aggr. and Update
19: $\boldsymbol{h}_l^{t+1} \leftarrow \boldsymbol{h}_l^t$

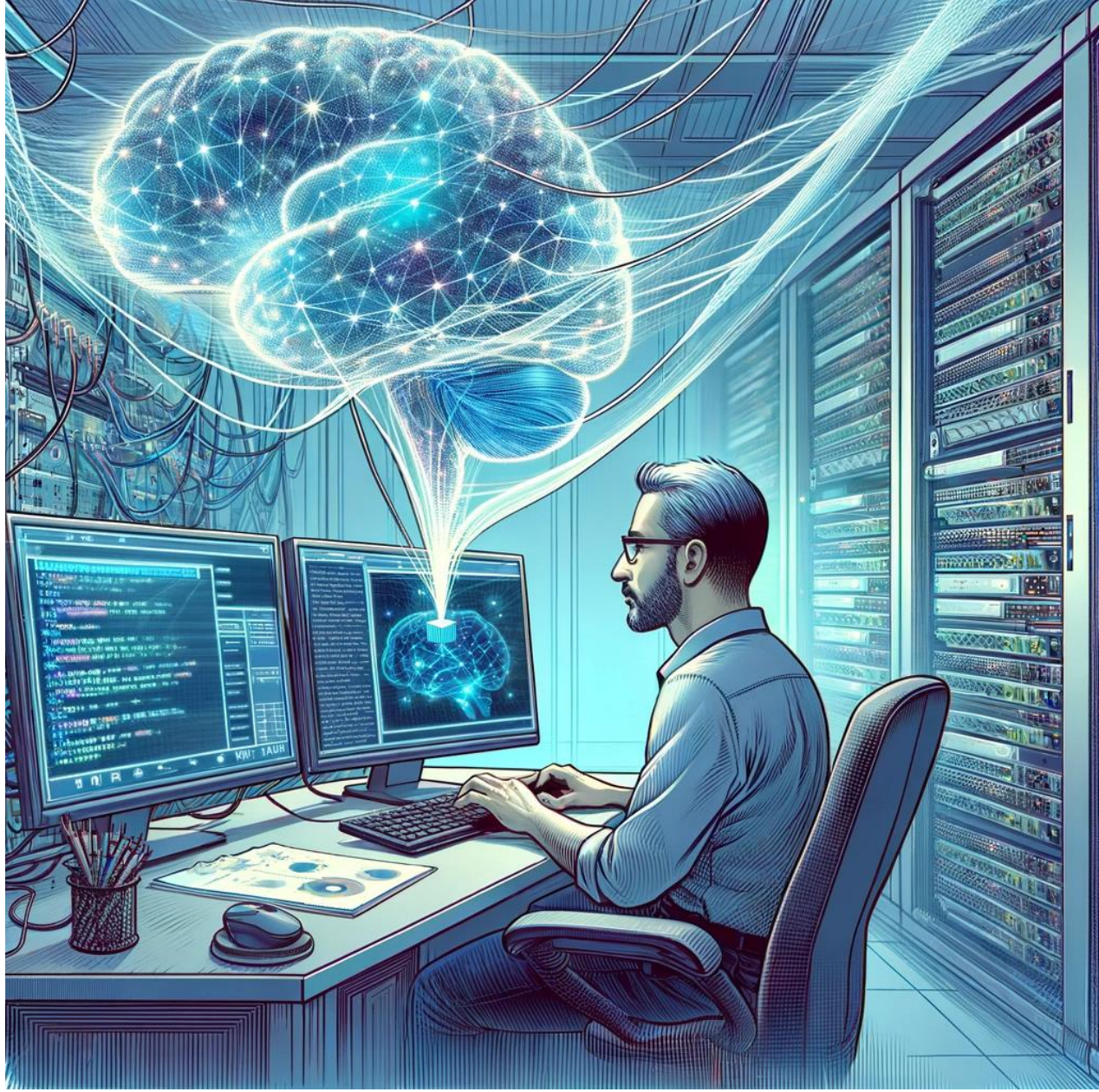
20: **for each** $f \in F$ **do** ▷ Flow: Readout
21: $\hat{y}_{f_d} = 0$ ▷ Initializing the flow delay
22: $\hat{y}_{f_j} = 0$ ▷ Initializing the flow jitter
23: **for each** $(q,l) \in f$ **do**
24: $\hat{d}_q = R_{f_d}(\boldsymbol{h}_{f,l}^T)/\boldsymbol{x}_{l_c}$ ▷ Queuing delay
25: $\hat{d}_t = \boldsymbol{x}_{f_{ps}}/\boldsymbol{x}_{l_c}$ ▷ Transmission delay
26: $\hat{d}_{link} = \hat{d}_q + \hat{d}_t$
27: $\hat{y}_{f_d} = \hat{y}_{f_d} + \hat{d}_{link}$ ▷ Sum of link delays along the flow
28: $\hat{y}_{f_j} = \hat{y}_{f_j} + R_{f_j}(\boldsymbol{h}_{f,l}^T)/\boldsymbol{x}_{l_c}$ ▷ Sum of link jitters along the flow
29: $\hat{y}_{f_l} = R_{f_l}(\boldsymbol{h}_f^T)$ ▷ Packet loss prediction

Source: https://arxiv.org/pdf/2212.12070.pdf

Hands-On Time

- We will explore Network Data used for RoutNet training and use it to predict network performance based on different configurations
- We pretrained a RoutNet Model in advance for you due to time limitations, but we will show you, how you could do it yourselves
- The link to the Notebook can be found in the Github Repo:

https://github.com/MalteWagner/ml_prague



Final thoughts

- Telco Networks are complex and natively in a graph structure
- GNNs can work well and generalize on telco network data
- We have specific challenges in creating and preparing datasets for training and designing architectures suited for the problem
- GNNs allow fast inference on optimization tasks or modelling what-if-scenarios
- Building digital twins that generalize on unseen topologies (e.g., different MNO)


sopra steria