Performance Evaluations of Computer Networks with Graph Neural Networks

Fabien Geyer

AI in Networking Summer School 2022

Thursday 24th February, 2022

Airbus Central R&T
Munich
fabien.geyer(at)airbus.com

Chair of Network Architectures and Services
Technical University of Munich (TUM)
fabien.geyer(at)tum.de

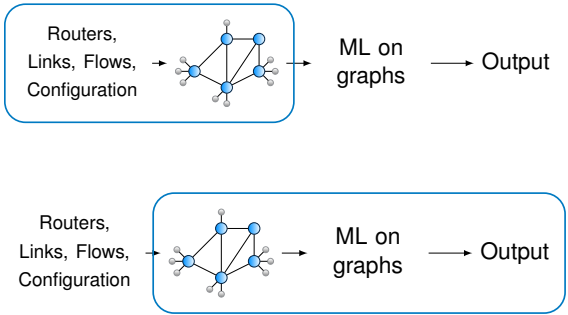
Outline

Graphs for computer networks

Graph Neural Networks

Application of GNNs for performance evaluation

Conclusions



Motivation

Machine Learning and Networking

Trends in networking and ML research

- Softwarization of networks → *SDN, OpenFlow, P4, ...*
- Hardware for flexible and customized line-rate processing → *DPDK, PISA switches, SmartNICs, RDMA, ...*
- Hardware and tools for machine learning

→ **Research towards autonomous / self-driving networks**

Various promises of ML for networks

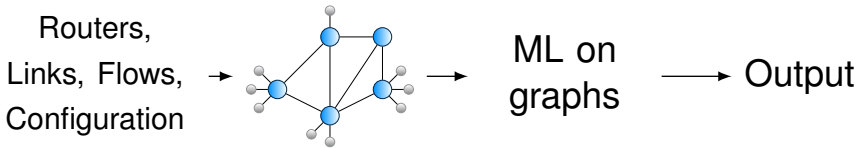
- Minimize human intervention in networks
- Use data-driven approach on monitoring data to optimize networks

→ Self-* networks: self-repairing, self-configuring, self-stabilizing, self-adjusting, ...

Motivation

Abstracting Computer Networks for ML

Machine learning workflow

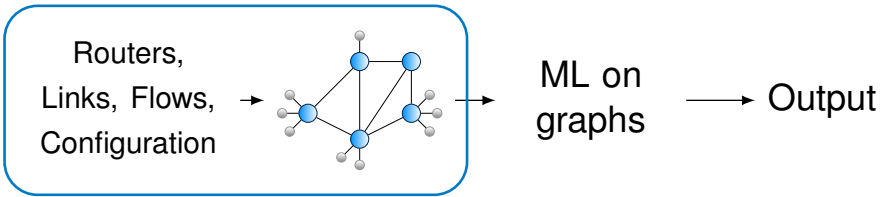


- Need abstraction which has a dynamic size and which can scale (independent of network size)

→ **Graphs as abstraction for representing networks**

→ **Use of machine learning frameworks which can learn on graphs**

Graphs for computer networks



Graphs for computer networks

Routing [Geyer and Carle, 2018]

Goal: Given a destination, routers need to know the next hop, i.e. which output interface to use

Transformation from topology to graph

- Nodes: routers and their interfaces
- Edges correspond to physical links

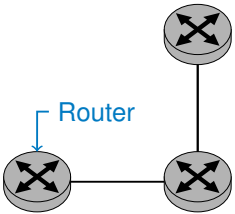


Figure 1: Network topology

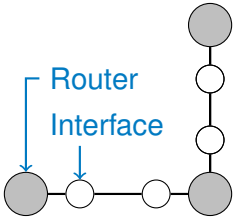


Figure 2: Graph encoding of topology

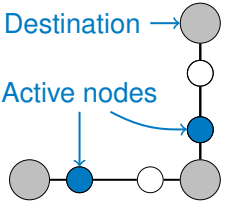


Figure 3: Output for routing (classification task)

Graphs for computer networks

Performance evaluation of flows [Geyer, 2018]

Goal: Given a network, predict the performance of flows *(eg. latency, bandwidth)* (regression task)

Transformation from topology to graph

- Nodes: routers, their interfaces, and flows
- Edges correspond to physical links between routers
- Edges also represent the path of flows

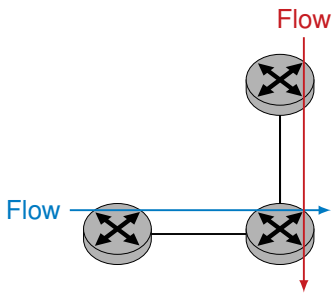


Figure 4: Network topology

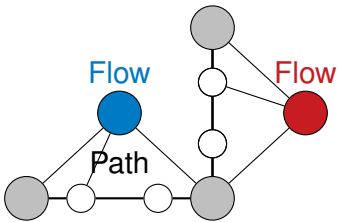


Figure 5: Graph encoding of topology

Graphs for computer networks

MPLS Networks [Geyer and Schmid, 2019]

Goal: Given a network and its MPLS configuration, predict properties such as reachability (classification task)

Transformation from topology to graph

- Nodes: routers, their interfaces, and MPLS rules

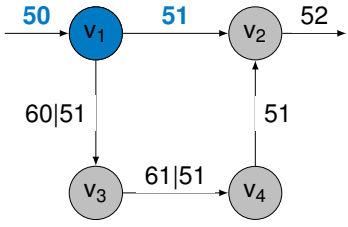


Figure 6: MPLS network, MPLS labels and rules

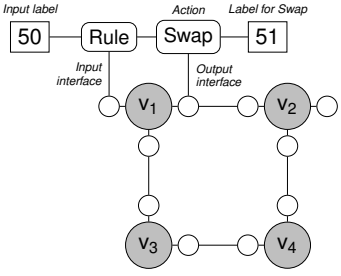
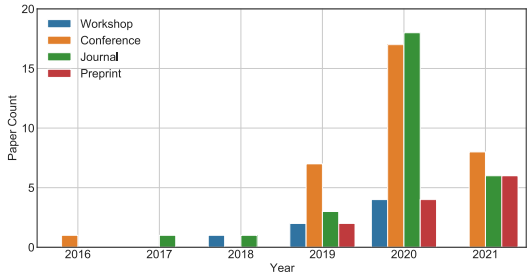


Figure 7: Graph encoding of topology and rules

Graphs for computer networks

Other examples for communication networks

- Learning **queuing theory** [Rusek and Cholda, 2018]
- Optimization in **Software Defined Networking** [Rusek et al., 2019]
- **Wireless networks**: analysis [Lee et al., 2019] of scheduling and interference [Shen et al., 2019]
- Speed-up of **formal verification** of end-to-end latencies (network calculus) [Geyer and Bondorf, 2019]
- Graph Neural Networking Challenge organized by UPC [Suárez-Varela et al., 2021]
- Recent survey showed **81 papers** ranging from 2016 to 2021 [Jiang, 2022]



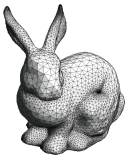
Papers count on GNNs for networking (source: [Jiang, 2022])

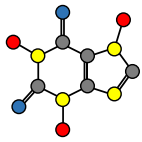
Graphs for computer networks

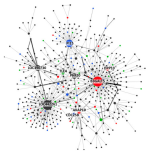
Other examples outside computer networks

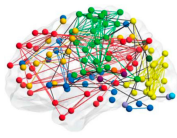
And in other fields also

- Analysis of source code and automatic bug-fixing
- Chemistry and molecule analysis *(AlphaFold)*
- Discrete optimization
- Knowledge graphs
- ...



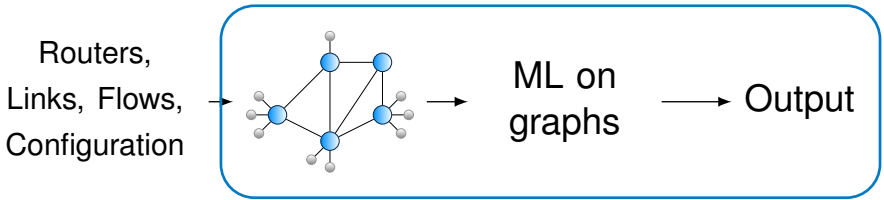






(source: [Bronstein et al., 2021])

Graph Neural Networks

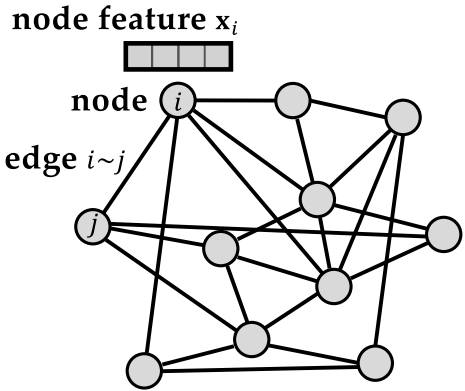


Focus of today's talk: Graph Neural Networks

(Disclaimer: some illustrations were taken from Bronstein et al.'s talk [Bronstein et al., 2021])

Graph Neural Networks

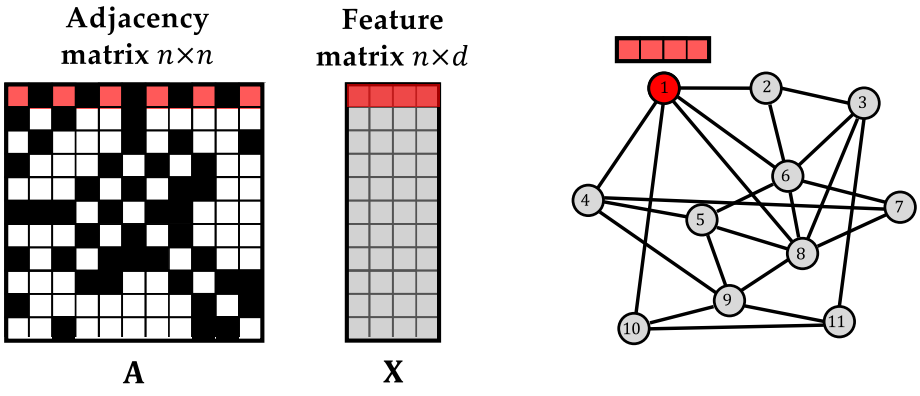
Key properties of graphs



(source: [Bronstein et al., 2021])

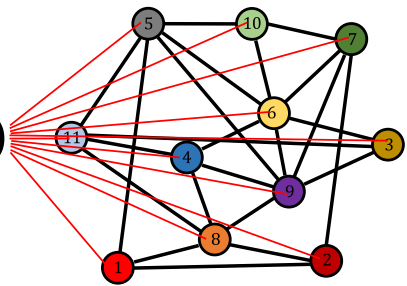
Graph Neural Networks

Key properties of graphs: arbitrary ordering



(source: [Bronstein et al., 2021])

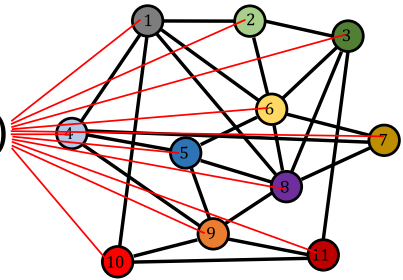
graph function $f(\mathbf{X}, \mathbf{A})$



(source: [Bronstein et al., 2021])

permutation-invariant

$$f\left(\mathbf{PX}, \mathbf{PAP}^{\top}\right) = f(\mathbf{X}, \mathbf{A})$$



(source: [Bronstein et al., 2021])

Graph Neural Networks

Introduction

Graph Neural Networks [Scarselli et al., 2009] and related neural network architectures are able to process general graphs and predict feature of nodes $\mathbf{o}_v$

Principle

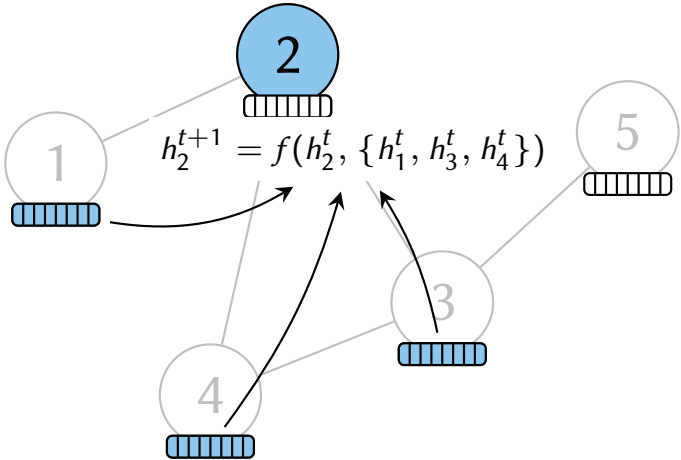
- Each node has a *hidden* vector $\mathbf{h}_v \in \mathbb{R}^k$
- ... computed according to the vector of its neighbors
- ... and are propagated through the graph
- Once the propagation is done, a prediction is made
  - Node prediction *(classification or regression)*
  - Edge prediction

Iterative algorithm

- Initialize $\mathbf{h}_v^{(0)}$ according to features of nodes ($\mathbf{X}$)
- for $t = 1, \ldots, T$ do
  - $\mathbf{a}_v^{(t)} = AGGREGATE\left(\left\{\mathbf{h}_u^{(t-1)} \mid u \in Nbr(v)\right\}\right)$
  - $\mathbf{h}_v^{(t)} = COMBINE\left(\mathbf{h}_v^{(t-1)}, \mathbf{a}_v^{(t)}\right)$
- return $READOUT\left(\mathbf{h}_v^{(T)}\right)$

Graph Neural Networks

Illustration



Graph Neural Networks

Implementation

Implementation (simplified)

- Initialize $\mathbf{h}_v^{(0)}$ according to features of nodes
- for $t = 1, \ldots, T$ do
  - *AGGREGATE* $\rightarrow \mathbf{a}_v^{(t)} = \sum_{u \in Nbr(v)} \mathbf{h}_u^{(t-1)}$
  - *COMBINE* $\rightarrow \mathbf{h}_v^{(t)} = \textit{Neural Network}\left(\mathbf{h}_v^{(t-1)}, \mathbf{a}_v^{(t)}\right)$
- *READOUT* $\rightarrow$ return $\textit{Neural Network}\left(\mathbf{h}_v^{(T)}\right)$

Training

- Using standard gradient descent techniques

Different approaches

- **Gated-Graph Neural Network** [Li et al., 2016]
- Graph Convolution Network
- Graph Attention Networks
- Graph Spatial-Temporal Networks
- ...

$\rightarrow$ **Hot area of research in the ML community**

$\rightarrow$ Various open-source libraries implementing GNNs

Graph Neural Networks

History of Graph Neural Networks according to Machine Learners

A. Sperduti

Labeling RAAM

1994

C. Goller

A. Küchler

Backprop through structure

1996

M. Gori

F. Scarselli

Graph Neural Networks

2005, 2008

Y. Li

Gated GNN

2015

(source: [Bronstein et al., 2021])

Some references (non-exhaustive list)

Introductions / Theory / Books

- [Gilmer et al., 2017]
- [Battaglia et al., 2018]
- [Wang et al., 2018]
- `https://geometricdeeplearning.com`
- `https://www.cs.mcgill.ca/~wlh/grl_book/`
- `https://distill.pub/2021/gnn-intro/`

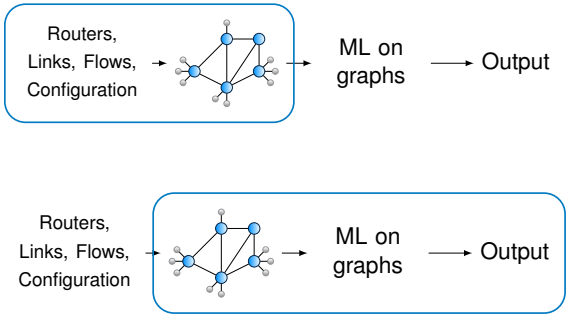
Libraries

- pytorch-geometric *(used for today's talk)*
- TensorFlow GNN
- DGL
- DeepMind's Graph Nets and jgraph
- Spektral
- Keras

Talks and keynotes on YouTube

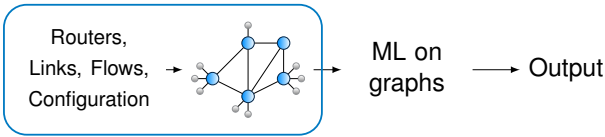
- ICLR 2021 Keynote - "Geometric Deep Learning: The Erlangen Programme of ML" - M. Bronstein
- Intro to graph neural networks - P. Veličković
- Beyond the Patterns 28 - Petar Veličković - Geometric Deep Learning

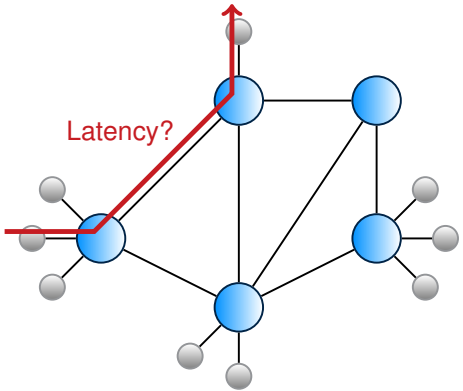
Application of GNNs for performance evaluation



Application of GNNs for performance evaluation

Introduction





Focus for the remaining slides: Network calculus

Application of GNNs for performance evaluation

Introduction to network calculus

Network calculus is a set of mathematical results which give insights into man-made systems such as concurrent programs, digital circuits and communication networks. [Le Boudec and Thiran, 2001]

Flows

- Flows represent unidirectional transfer of data in a network
- Modeled as cumulative function A, where $A(t)$ represents the amount of data sent in the interval $[0, t)$
- Token buckets are often used as envelope: $A(t) - A(s) \leq \mathit{rate} \cdot (t - s) + \mathit{burst}$

Servers

- Servers represent entities processing the data in the network: queues, schedulers, links, etc.
- They model the relation between some arrival cumulative curve A and some departure cumulative curve D.
- Rate-latency curves are often used as envelope: $D(t) - A(s) \leq \mathit{rate} \cdot [(t - s) - \mathit{latency}]^+$

Application of GNNs for performance evaluation

Introduction to network calculus

Bounds

- Based on this model and the envelopes it is possible to compute bounds
- **Latency bound**, representing an upper bound on the end-to-end delay
- **Backlog bound**, representing an upper bound on the queue size

Mathematics and methods

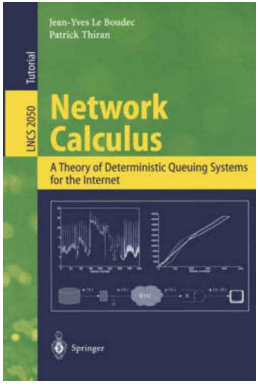
- The **(min,plus) algebra** can be defined for working with the envelopes
- Interesting properties result from the algebra: *server concatenation, pay burst / multiplexing only once*

Applications

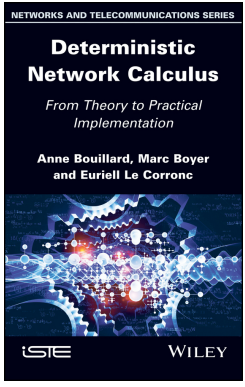
- Industrial networks where real time guarantees are important (eg. aircraft)
- IEEE Time Sensitive Networking (TSN)

Application of GNNs for performance evaluation

Introduction to network calculus – References



by Jean-Yves Le Boudec and Patrick Thiran
Download: `https://leboudec.github.io/netcal/`



by Anne Bouillard, Marc Boyer, and Euriell Le Corronc

Application of GNNs for performance evaluation

Back to ML and GNNs

Inputs

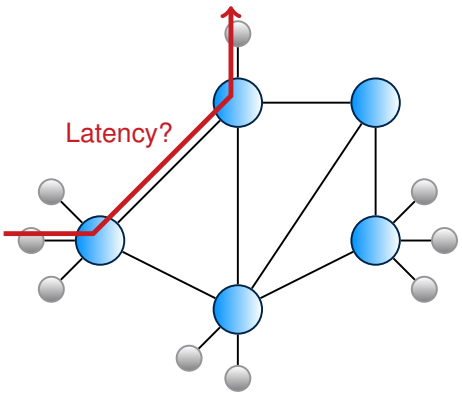
- Network topology, with the queues, links and their properties
- Flows, with their path in the network, and their traffic envelopes

→ **Modeled as graph**

Target

- Predict end-to-end delay bounds

→ **Regression task for each flow in the network**



Application of GNNs for performance evaluation

Code

`https://github.com/fabgeyer/ainet-summer-school-2022`

(code and slides located there)

Libraries used:

- pytorch
- pytorch-geometric
- networkx

Step 1: Getting and understanding the dataset

We will use the dataset from [Geyer et al., 2021], with instructions on `https://github.com/fabgeyer/dataset-rtas2021`

```
$ wget -r ftp://m1596901:m1596901@dataserv.ub.tum.de/
$ mv dataset.ub.tum.de dataset
$ tree dataset
dataset
 checksums.sha512
 dataset-evaluation-large.pbz
 dataset-evaluation.pbz
 dataset_structure.proto
 dataset-train.pbz
```

We will read the dataset using `https://github.com/fabgeyer/pbzlib-py`

```
$ pip install pbzlib
```

Step 1: Getting and understanding the dataset

```
import pbzlib
for network in pbzlib.open_pbz("dataset/dataset-train.pbz"):
  print(network)
  break
```

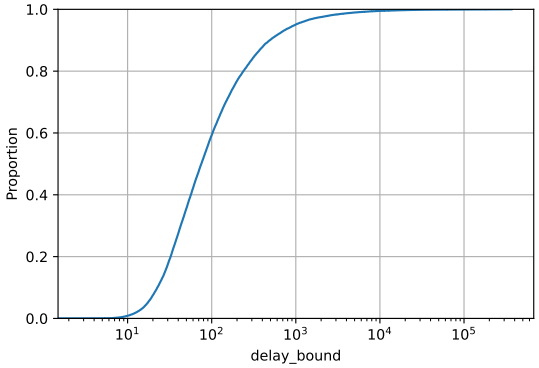
```
server {
  id: 0
  rate: 0.652298881523439
  latency: 0.5384246880613892
}
server {
  id: 1
  rate: 0.1981014874337499
  latency: 0.14038693814170045
}
...
```

```
flow {
  id: 0
  rate: 0.00031810534748519687
  burst: 0.968261581700407
  path: 5
  path: 4
  ...
  pmoo {
    delay_bound: 41.661718156913125
  }
  ...
}
flow {
  id: 1
```

Step 1: Getting and understanding the dataset

Let's understand the distribution of delay bounds:

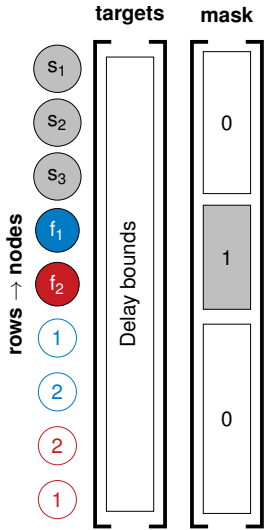
```
import seaborn
from pbzlib import open_pbz

delays = []
for network in open_pbz("dataset/dataset-
    train.pbz"):
  for flow in network.flow:
    if flow.pmoo.delay_bound > 0:
      delays.append(flow.pmoo.delay_bound)

seaborn.ecdfplot(delays)
plt.xscale("log")
```



First lesson

- The delay bounds range over multiple orders of magnitude!
- We will need to scale it for training the GNN → `sklearn.preprocessing` package

Step 2: Graph transformation



Steps for transforming a network to graph:

- Each server → node in the graph
- Each link between servers → edges between servers
- Each flow → node in the graph
- How to represent the path?
  - Edges connecting the flow with the servers it traverses
  - Order is important, so add so-called *path order* nodes

→ **Matrix representation of the graph**

→ **Matrix representation of the targets and mask**

→ **Edges represented as adjacency list**

`network2graph` and `graph2torch` functions in the code

Step 3: Build the GNN

```
import torch_geometric.nn as gnn
class GNNModel(gnn.MessagePassing):
  def __init__(self, num_features, num_classes, hidden_size):
    super(GNNModel, self).__init__()
    self.fcin = nn.Sequential(nn.Linear(num_features, hidden_size), nn.LeakyReLU())
    self.cell = gnn.GatedGraphConv(hidden_size, 1)
    self.fcout = nn.Sequential(
      nn.Linear(hidden_size, hidden_size),
      nn.LeakyReLU(),
      nn.Linear(hidden_size, num_classes))

  def forward(self, x, edge_index, nunroll):
    h = self.fcin(x)
    for _ in range(nunroll):
      h = self.cell(h, edge_index)
    h = self.fcout(h)
    return h
```

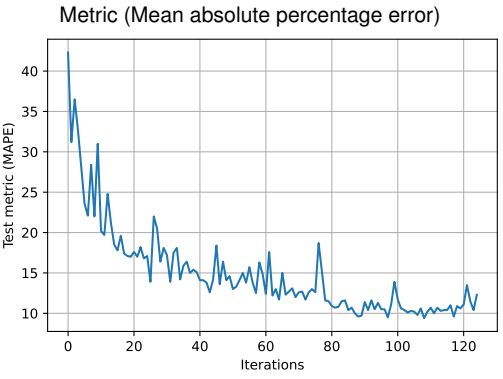
Step 4: Train the GNN

Main training loop

```
model = GNNModel(...)
optimizer = torch.optim.AdamW(model.parameters())
criterion = nn.MSELoss()

for dt in loader:
  optimizer.zero_grad()
  output = model(dt.x, dt.edge_index)
  nodes_of_interest = torch.where(dt.mask)[0]
  output = torch.index_select(output, 0, nodes_of_interest)
  target = torch.index_select(dt.y, 0, nodes_of_interest)
  loss = criterion(output, target)
  loss.backward()
  optimizer.step()
```

The loss function is only applied to a subset of nodes

Step 4: Train the GNN



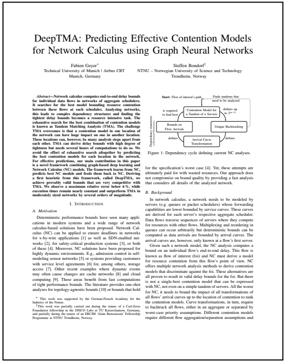
Improve the training performance

Hyper-parameter optimization is often needed to find the best training parameters (eg. learning rate)
Various tools available (eg. `https://nni.readthedocs.io` used in the code)

GNNs and network calculus

This tutorial is only a **toy example**.

Applications of GNNs for network calculus: **speeding up the analysis by avoiding exhaustive searches**



DeepTMA: Predicting Effective Contention Models for Network Calculus using Graph Neural Networks

Fabien Geyer*
Technical University of Munich | Airbus CRT
Munich, Germany

Steffen Bondorf†
NTNU – Norwegian University of Science and Technology
Trondheim, Norway

Abstract—**Network calculus computes end-to-end delay bounds for individual data flows in networks of aggregate schedulers. It searches for the best model bounding resource contention between these flows at each scheduler. Analyzing networks, this leads to complex dependency structures and finding the tightest delay bounds becomes a resource intensive task. The exhaustive search for the best combination of contention models is known as Tandem Matching Analysis (TMA). The challenge TMA overcomes is that a contention model in one location of the network can have huge impact on one in another location. These locations can, however, be many analysis steps apart from each other. TMA can derive delay bounds with high degree of tightness but needs several hours of computations to do so. We avoid the effort of exhaustive search altogether by predicting the best contention models for each location in the network. For effective predictions, our main contribution in this paper is a novel framework combining graph-based deep learning and Network Calculus (NC) models. The framework learns from NC, predicts best NC models and feeds them back to NC. Deriving a first heuristic from this framework, called DeepTMA, we achieve provably valid bounds that are very competitive with TMA. We observe a maximum relative error below 6 %, while execution times remain nearly constant and outperform TMA in moderately sized networks by several orders of magnitude.**

I. INTRODUCTION

A. Motivation

Deterministic performance bounds have seen many applications in modern systems and a wide range of network calculus-based solutions have been proposed. Network Calculus (NC) can be applied to ensure deadlines in networks for x-by-wire applications [1] as well as SDN-enabled networks [2], for safety-critical production systems [3], or both of these [4]. Moreover, NC solutions have been proposed for highly dynamic environments. E.g., admission control in self-modeling sensor networks [5] or systems providing customers with service level agreements [6] for, among others, storage access [7]. Other recent examples where dynamic events may often cause changes are cache networks [8] and cloud computing [9]. These areas benefit from fast computations of tight performance bounds. The literature provides one-shot analyses for topology-agnostic bounds [10] or bounds that hold

* This work was supported by the German-French Academy for the Industry of the Future.
†This work was partially carried out during the tenure of a Carl-Zeiss Foundation fellowship in the DISCO Labs at TU Kaiserslautern, Germany, and partially during the tenure of an ERCIM 'Alain Bensoussan' Fellowship Programme at NTNU Trondheim, Norway.

Start: Flow of interest's path
Finds tandems that need to be analyzed
is required to find best
Contention Model for a Tandem of n Servers
defines up to $2^{(n-1)}$
Dependency structures
Bounds on Flow Arrivals
Unique Backtracking
defines
Arrival Curve Transformation
defines

Figure 1: Dependency cycle defining current NC analyses.

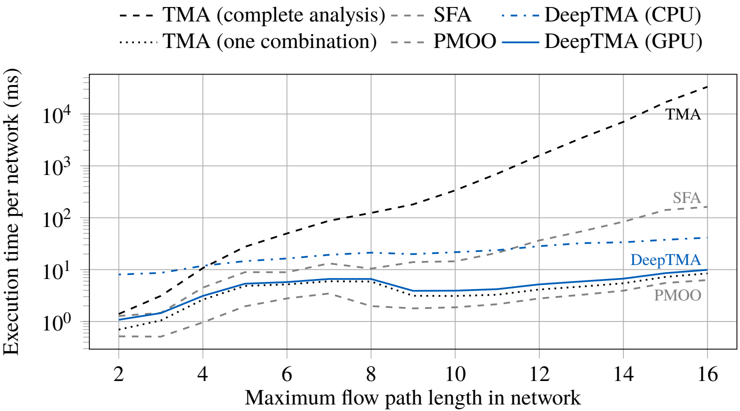
for the specification's worst case [4]. Yet, these attempts are ultimately paid for with wasted resources. Our approach does not compromise on bound quality by providing a fast analysis that considers all details of the analyzed network.

B. Background

In network calculus, a network needs to be modeled by servers (e.g. queues or packet schedulers) whose forwarding capabilities are lower bounded by service curves. These curves are derived for each server's respective aggregate scheduler. Data flows traverse sequences of servers where they compete for resources with other flows. Multiplexing and reordering in queues can occur arbitrarily but deterministic bounds can be computed as data arrivals are bounded by arrival curves. The arrival curves are, however, only known at a flow's first server.

Given such a network model, the NC analysis computes a bound on an individual flow's end-to-end delay. This flow is known as flow of interest (foi) and NC must derive a model for resource contention from this flow's point of view. NC offers multiple network analysis methods to derive contention models that discriminate against the foi. These alternatives are all proven to result in valid delay bounds for the foi. But there is not a single-best contention model that can be expressed with NC, not even on a simple tandem of servers. All the worse for NC, it needs to bound the impact of all transformations of all flows' arrival curves up to the location of contention to rank the contention models. Curve transformations, in turn, require to backtrack all flows, either in an aggregate or separated by worst-case priority assumptions. Different contention models require different flow aggregation/separation assumptions and

10.1109/INFOCOM.2019.8737496



Conclusions

Conclusions

Summary of today's talk

- Overview over **Graph Neural Networks'** theory and practice
- Examples of **graph transformations** for networking problems
- A **concrete application** of GNNs for predicting latency bounds in computer networks
- **Code** to play with GNNs and better understand some practical aspects

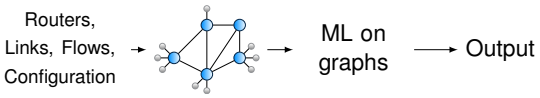
Next steps for you

- Identify if graphs are the adequate data structure for the challenge you are trying to address
- Better understand the theory behind GNNs
- Play with one of the GNN libraries and your deep learning framework of choice

Conclusions

Key messages

- Interesting time for ML applied to computer networks
- **Graphs as abstraction for computer networks for ML**
- Use of **Graph Neural Networks** to make predictions on nodes and edges



Future work – **Collaborations welcome!**

- Application to more challenging and dynamic tasks → **self-* networks**
- **Deployment in real networks** (eg. GNN-based SDN controller)
- Evaluation of most suitable neural network architecture and their scalability

Q & A

My personal page
(contact and publications)

Code and slides

[Battaglia et al., 2018] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gulcehre, C., Song, F., Ballard, A., Gilmer, J., Dahl, G., Vaswani, A., Allen, K., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y., and Pascanu, R. (2018).
Relational inductive biases, deep learning, and graph networks.
arxiv:1806.01261.

[Bronstein et al., 2021] Bronstein, M., Bruna, J., Cohen, T., and Veličković, P. (2021).
Geometric deep learning.
African Master in Machine Intelligence – Summer 2021.

[Geyer, 2018] Geyer, F. (2018).
DeepComNet: Performance Evaluation of Network Topologies using Graph-Based Deep Learning.
Performance Evaluation.

[Geyer and Bondorf, 2019] Geyer, F. and Bondorf, S. (2019).
DeepTMA: Predicting Effective Contention Models for Network Calculus using Graph Neural Networks.
In *Proc. IEEE INFOCOM.*

[Geyer and Carle, 2018] Geyer, F. and Carle, G. (2018).
Learning and Generating Distributed Routing Protocols Using Graph-Based Deep Learning.
In *Proc. SIGCOMM Workshop on Big Data Analytics and Machine Learning for Data Communication Networks (Big-DAMA)*, pages 40–45. ACM.

[Geyer et al., 2021] Geyer, F., Scheffler, A., and Bondorf, S. (2021).
Tightening Network Calculus Delay Bounds by Predicting Flow Prolongations in the FIFO Analysis.
In *Proceedings of the 27th IEEE Real-Time and Embedded Technology and Applications Symposium*, RTAS 2021.

[Geyer and Schmid, 2019] Geyer, F. and Schmid, S. (2019).
DeepMPLS: Fast Analysis of MPLS Configurations Using Deep Learning.
In *IFIP Networking 2019.*

[Gilmer et al., 2017] Gilmer, J., Schoenholz, S. S., Riley, P. F., Vinyals, O., and Dahl, G. E. (2017).
Neural message passing for quantum chemistry.
In *Proc. of NIPS*.

[Jiang, 2022] Jiang, W. (2022).
Graph-based deep learning for communication networks: A survey.
Computer Communications, 185:40–54.

[Le Boudec and Thiran, 2001] Le Boudec, J.-Y. and Thiran, P. (2001).
Network Calculus: A Theory of Deterministic Queuing Systems for the Internet.
Springer-Verlag, Berlin, Heidelberg.

[Lee et al., 2019] Lee, M., Yu, G., and Li, G. Y. (2019).
Graph embedding based wireless link scheduling with few training samples.

[Li et al., 2016] Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2016).
Gated Graph Sequence Neural Networks.
In *Proc. of ICLR*.

[Rusek and Cholda, 2018] Rusek, K. and Cholda, P. (2018).
Message-Passing Neural Networks Learn Little's Law.
IEEE Communications Letters.

[Rusek et al., 2019] Rusek, K., Suárez-Varela, J., Mestres, A., Barlet-Ros, P., and Cabellos-Aparicio, A. (2019).
Unveiling the potential of graph neural networks for network modeling and optimization in sdn.
In *Proceedings of the 2019 ACM Symposium on SDN Research*, SOSR '19, pages 140–151. ACM.

[Scarselli et al., 2009] Scarselli, F., Gori, M., Tsoi, A. C., Hagenbuchner, M., and Monfardini, G. (2009).
The Graph Neural Network Model.
IEEE Transactions on Neural Networks, 20(1):61–80.

[Shen et al., 2019] Shen, Y., Shi, Y., Zhang, J., and Letaief, K. B. (2019).
A graph neural network approach for scalable wireless power control.

[Suárez-Varela et al., 2021] Suárez-Varela, J. et al. (2021).
The graph neural networking challenge: a worldwide competition for education in ai/ml for networks.
ACM SIGCOMM Computer Communication Review, 51(3):9–16.

[Wang et al., 2018] Wang, X., Girshick, R., Gupta, A., and He, K. (2018).
Non-local neural networks.
In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.