Unveiling the potential of Graph Neural Networks

Prof. Albert Cabellos (acabello@ac.upc.edu)

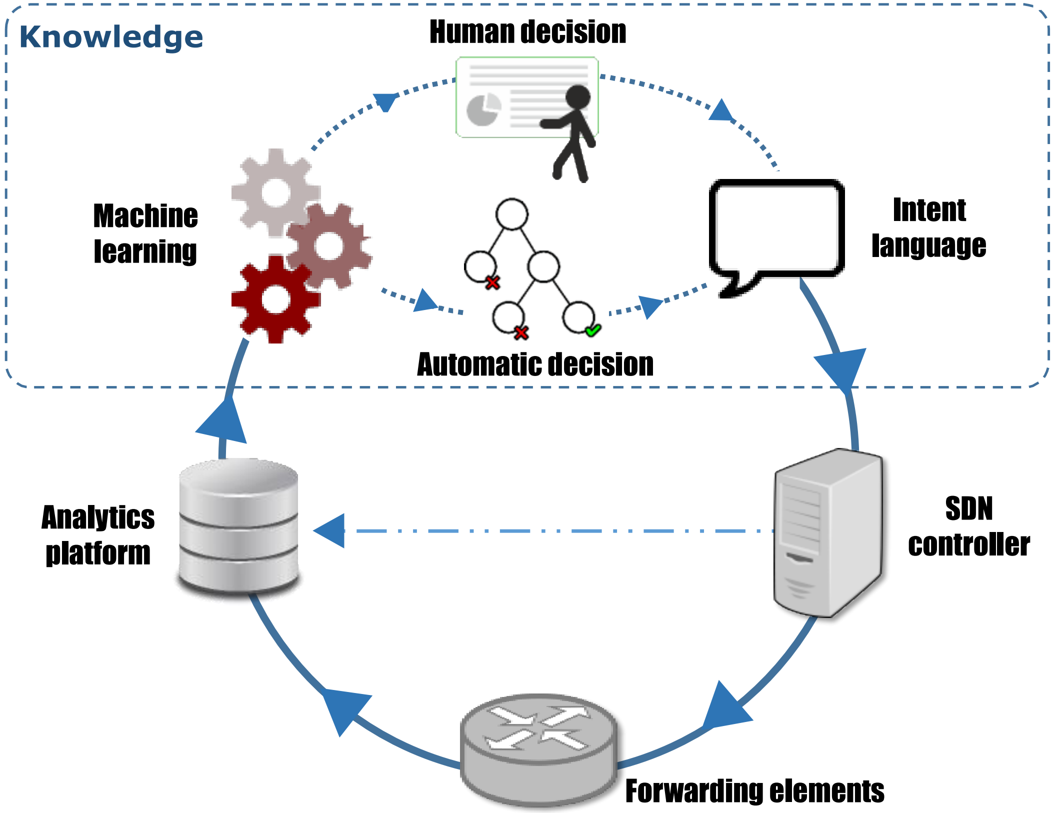
bnn.upc.edu

Universitat Politècnica de Catalunya

FINE, December 2019

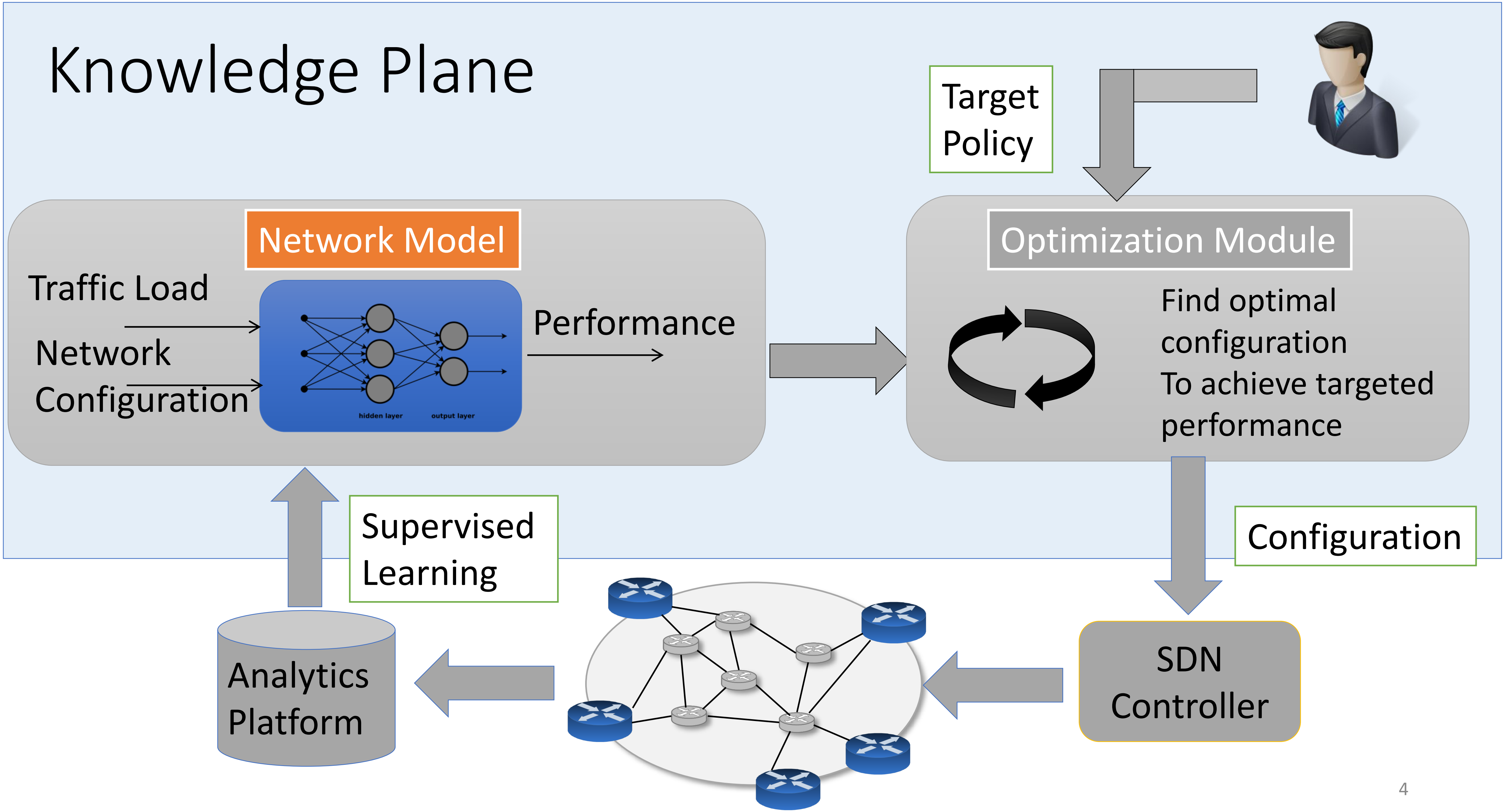
How we can apply ML techniques to Computer Networks?

Knowledge-Defined Networking



- **Goal**
  - Manage and control the network with ML techniques
  - E.g., - Chose optimal paths for a set of flows
- **Why?**
  - Network optimization
  - Self-driving networks
  - Reduced operational costs

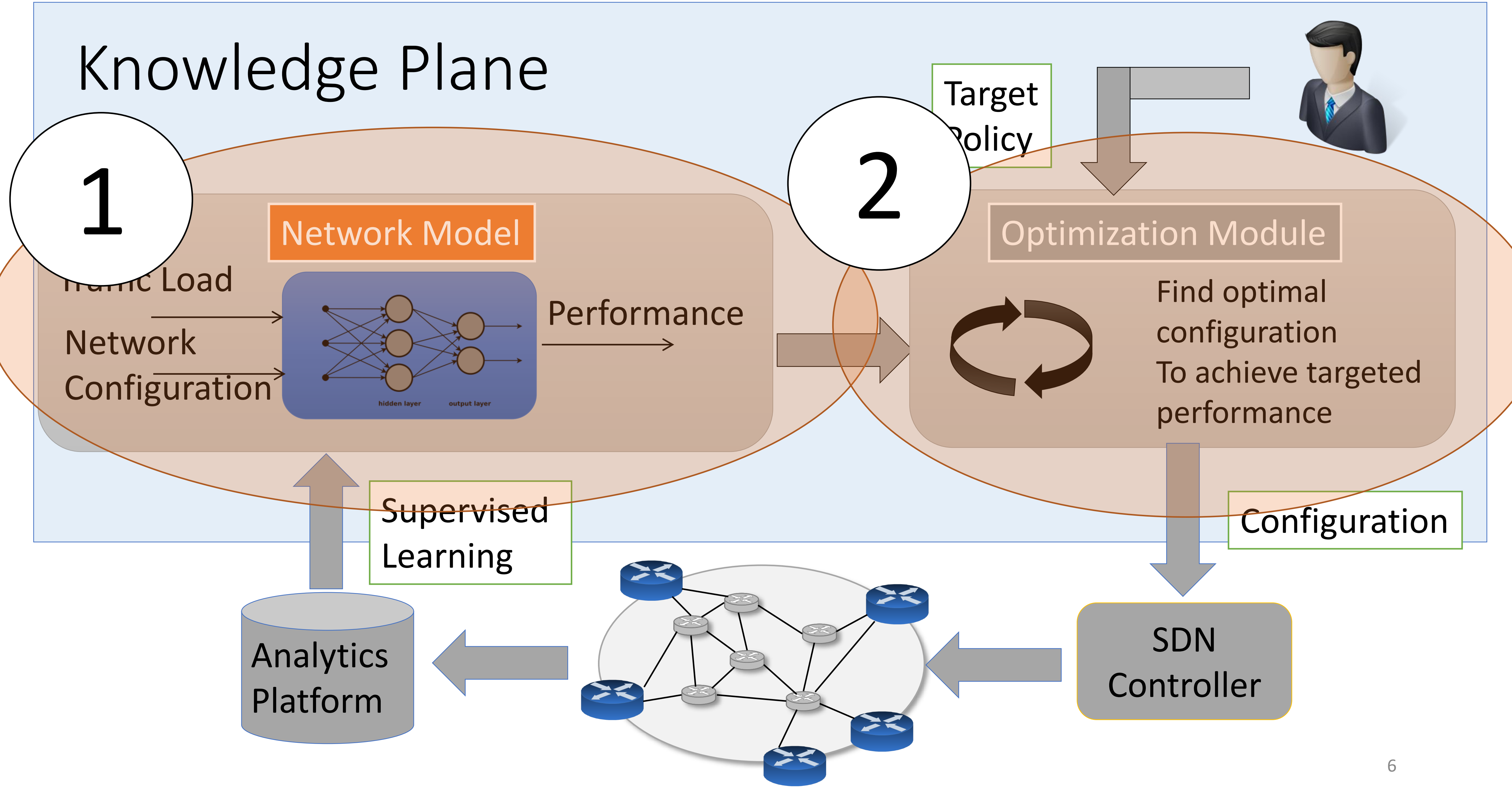
Mestres, Albert, Alberto Rodriguez-Natal, Josep Carner, Pere Barlet-Ros, Eduard Alarcón, Marc Solé, Victor Muntés-Mulero et al. "Knowledge-defined networking." *ACM SIGCOMM Computer Communication Review* 47, no. 3 (2017): 2-10.


Knowledge Plane
Target Policy
Network Model
Traffic Load
Network Configuration
hidden layer
output layer
Performance
Optimization Module
Find optimal configuration To achieve targeted performance
Supervised Learning
Configuration
Analytics Platform
SDN Controller

In this talk:

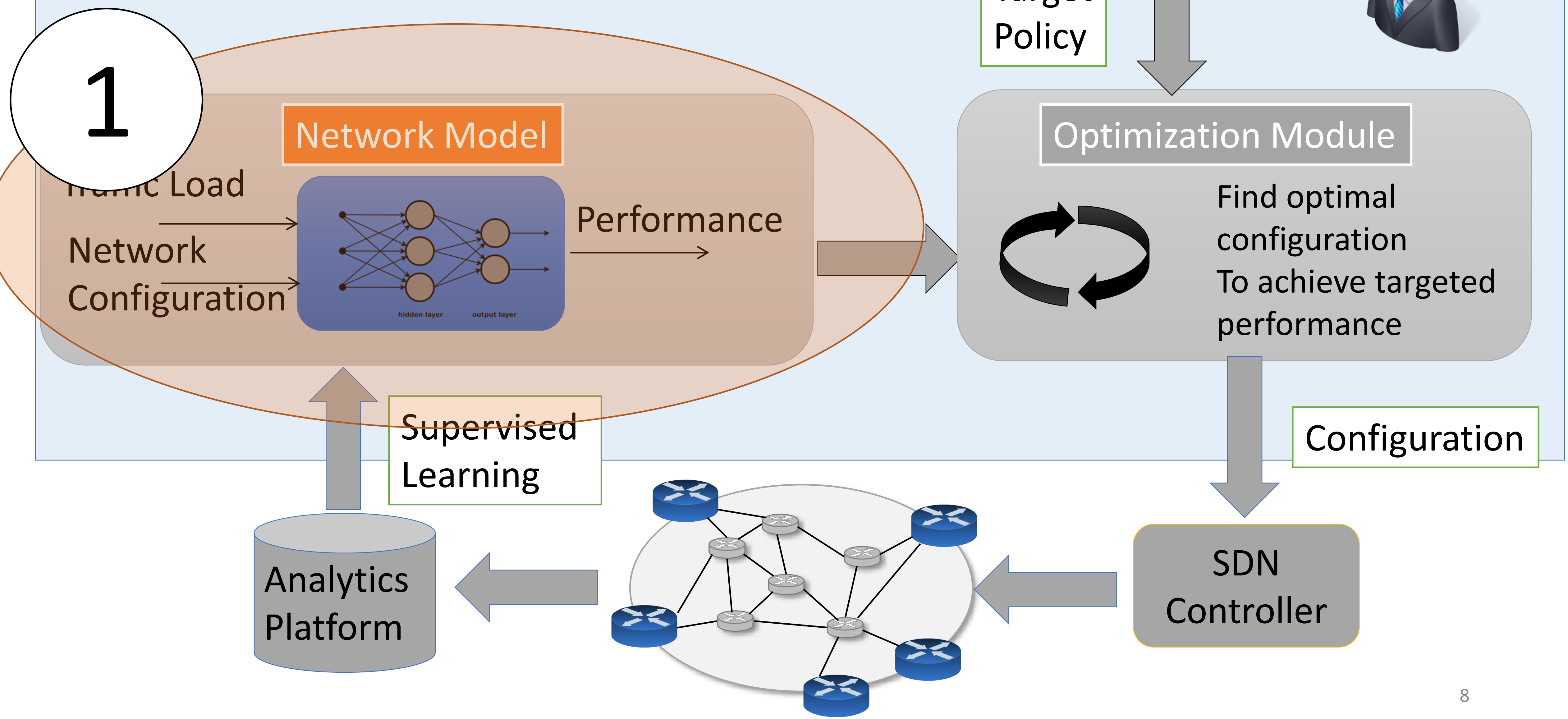
1.- How we can build a Network Model using ML?

2.- How we can optimize the network using ML?

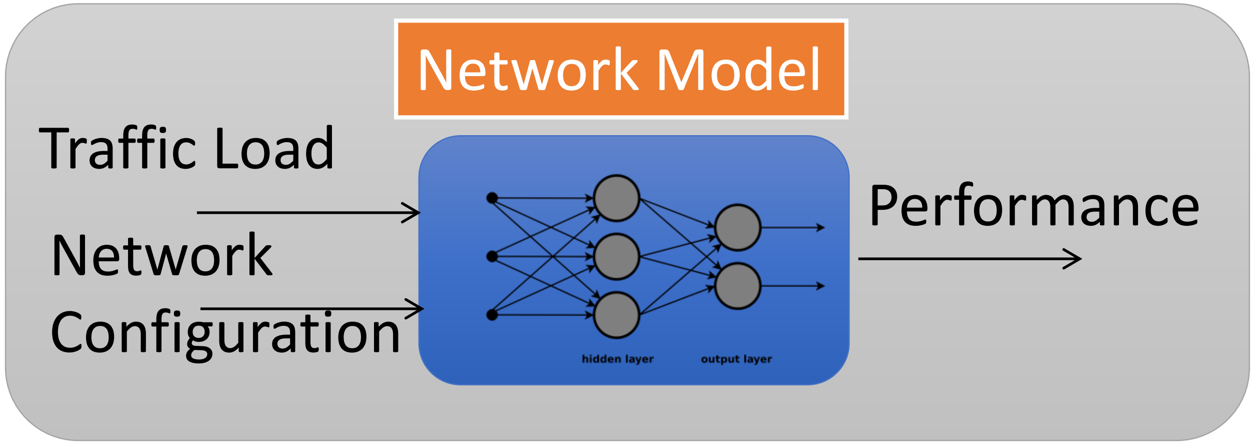

Knowledge Plane
1
Network Model
Traffic Load
Network Configuration
hidden layer
output layer
Performance
2
Target Policy
Optimization Module
Find optimal configuration
To achieve targeted performance
Supervised Learning
Configuration
Analytics Platform
SDN Controller

1.- How we can build the Network Model using ML?

Knowledge Plane

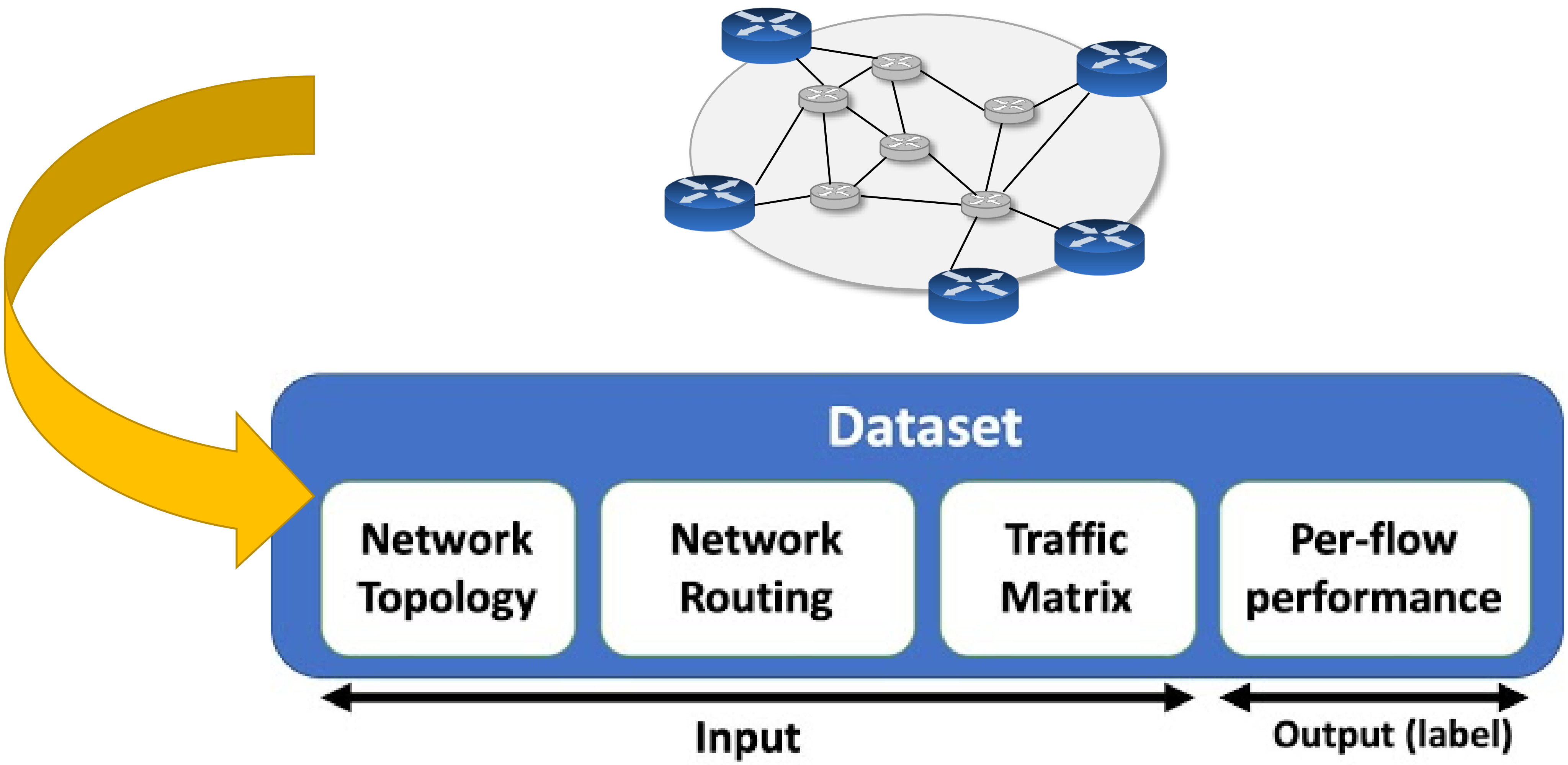


Network Model: A Digital Network Twin

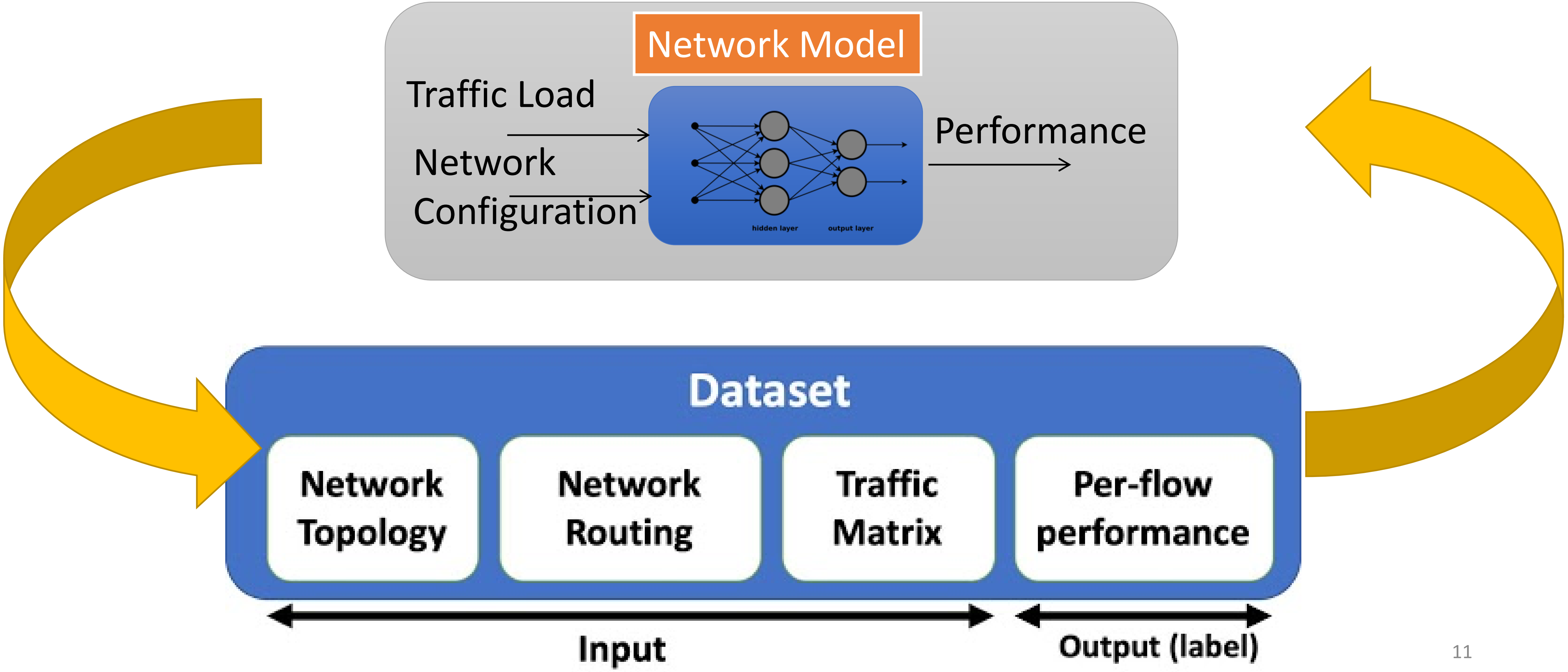


- Three main approaches
  - Computational Models: Simulation
  - Analytical Models: Queueing Theory
  - **Neural Networks - Our approach**

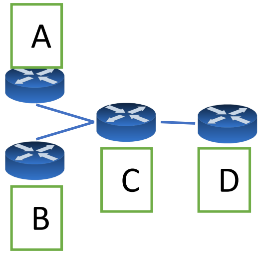
Let´s use Neural Nets to build this Dataset Generation



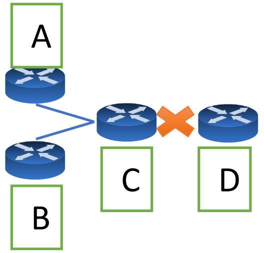
Let´s use Neural Nets to build this: Training of Neural Network



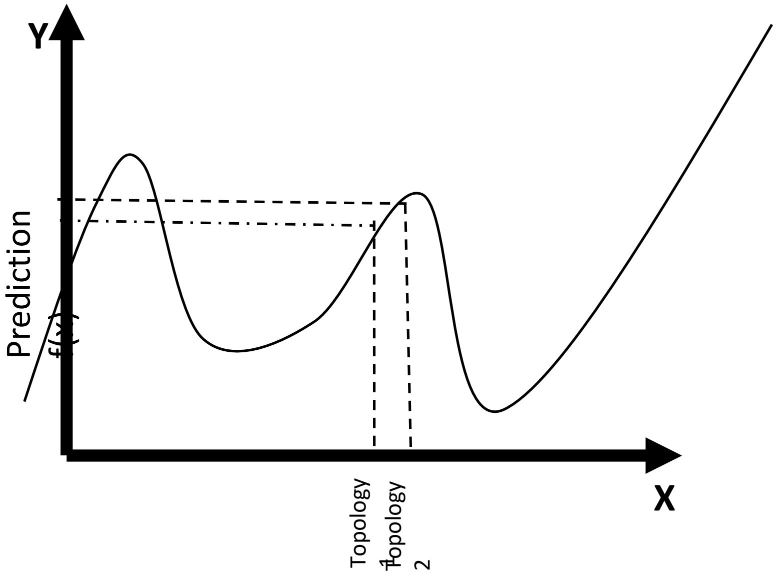
How we can represent the network topology?



	A	B	C	D
A	1	0	1	0
B	0	1	1	0
C	1	1	1	1
D	0	0	1	1



	A	B	C	D
A	1	0	1	0
B	0	1	1	0
C	1	1	1	0
D	0	0	0	1



- **Networks configuration is a graph:**
  - Routing
  - Topology
  - Etc.
- Use graphs as inputs of neural nets.
- Academics have repeatedly failed to achieve this

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 ACM SOSR 2019*

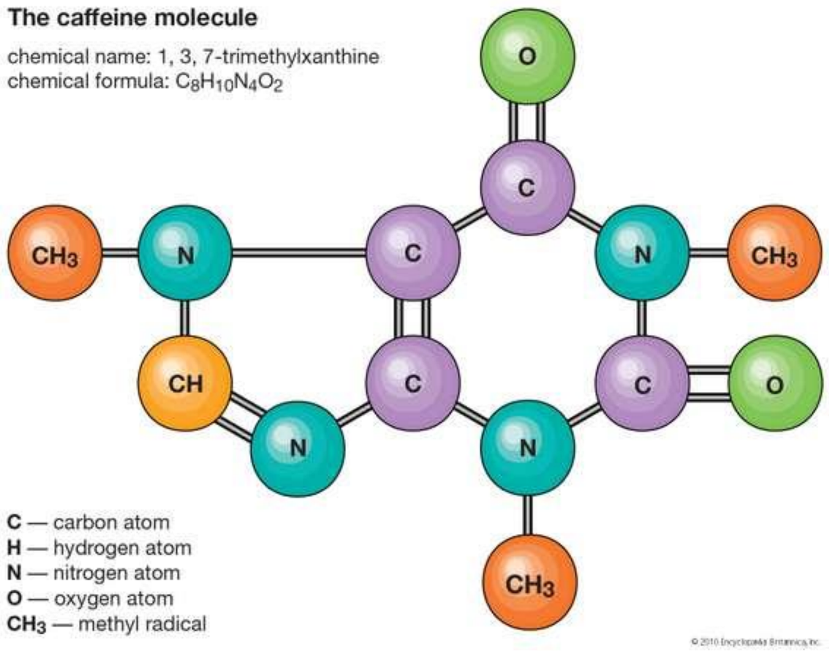
https://arxiv.org/pdf/1901.08113.pdf

ML applied to Networking

- So far we have **<u>failed</u>** to learn Computer Networks (e.g)
  - Valadarsky, A., Schapira, M., Shahaf, D., & Tamar, A. (2017, November). Learning to route. In *Proceedings of the 16th ACM Workshop on Hot Topics in Networks* (pp. 185-191). ACM.
  - Chen, X., Guo, J., Zhu, Z., Proietti, R., Castro, A., & Yoo, S. J. B. (2018, March). Deep-RMSA: A Deep-Reinforcement-Learning Routing, Modulation and Spectrum Assignment Agent for Elastic Optical Networks. In *2018 Optical Fiber Communications Conference and Exposition (OFC)* (pp. 1-3). IEEE.
  - Poor performance, in some cases worse than simple well-known heuristics
  - Ad-hoc solutions tailored to specific problems, in some cases transforming the problem to prevent learning graphs

The main reason for this is that standard Neural Networks **<u>are not suited</u>** to learn information structured as a graph

What about other fields?



- Other research areas faced similar problems
  - E.g., molecule representation in chemistry
- Graph Neural Networks (GNN)
  - Neural Network architecture to learn graph representations more suitable for learning

What are
Graph Neural Networks?

Graph Neural Networks (GNN)

- GNN have been recently proposed by DeepMind *et al.* to learn and model information structured as a **graph**

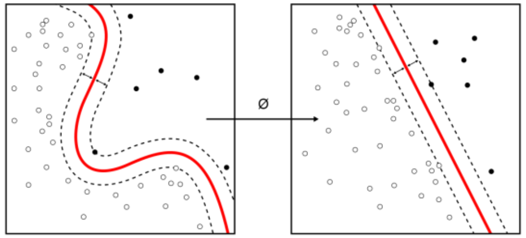
Component	Entities	Relations	Rel. inductive bias	Invariance
Fully connected	Units	All-to-all	Weak	-
Convolutional	Grid elements	Local	Locality	Spatial translation
Recurrent	Timesteps	Sequential	Sequentiality	Time translation
Graph network	Nodes	Edges	Arbitrary	Node, edge permutations

- Each application has developed their own NN architectures
  - Fully Connected = Units → General application (non-linear regression)
  - CNN = Grid elements → Images
  - RNN = Sequences → Text processing, Time-Series
  - GNN = Nodes + Edges → Networks

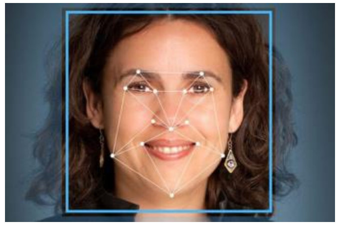
Battaglia, Peter W., et al. "Relational inductive biases, deep learning, and graph networks." *arXiv preprint arXiv:1806.01261*(2018).

Overview of the most common NN architectures

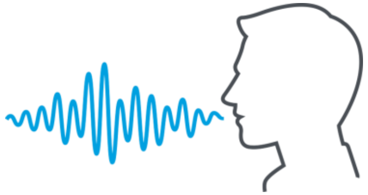
Type of NN	Information Structure
Fully Connected NN	Arbitrary
Convolutional NN	Spatial
Recurrent NN	Sequential
Graph NN	Relational



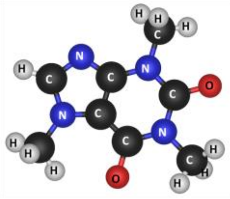
Classification, Unsupervised Learning



Images and video

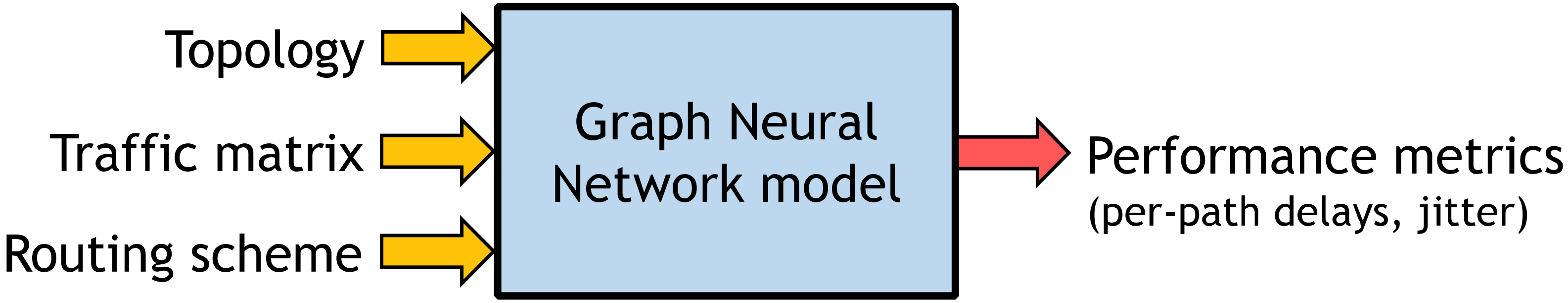


Text and voice



Graphs (molecules, maps, networks)

RouteNet: The first GNN for Computer Networks

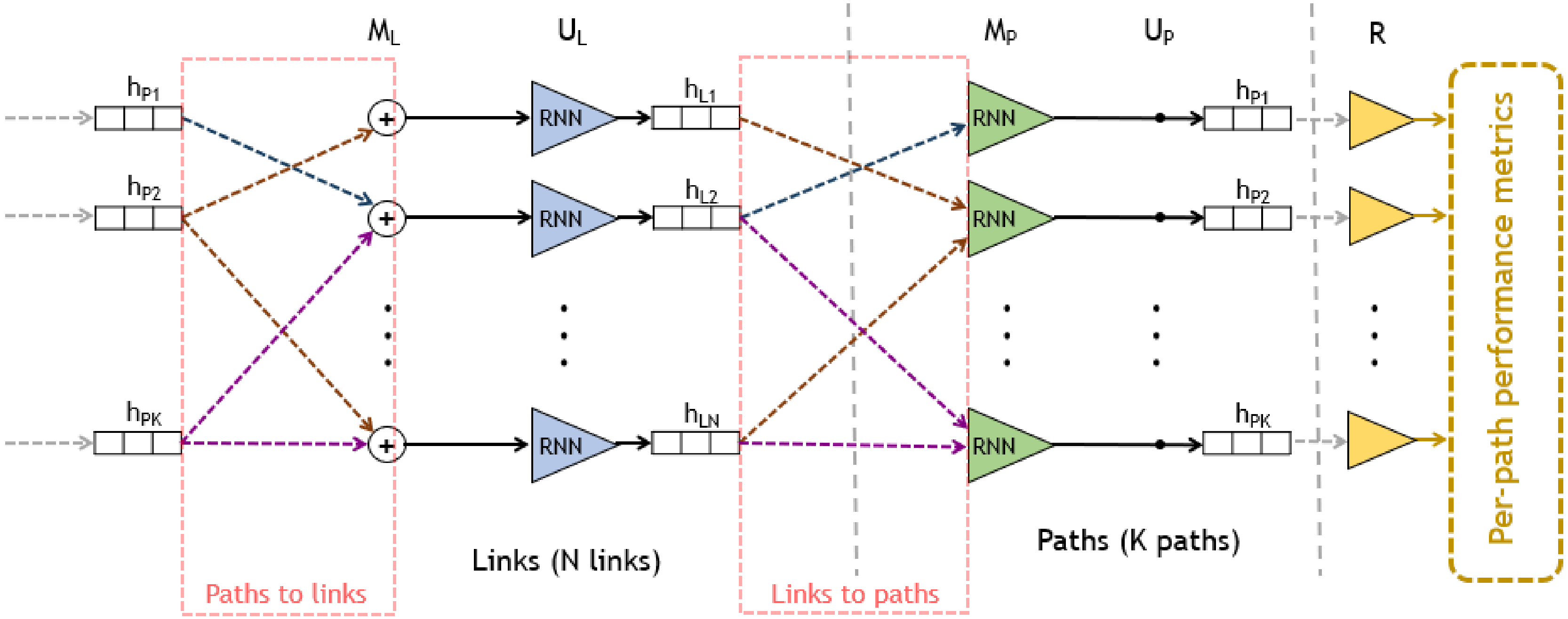


- RouteNet is the first Graph Neural Network for Computer Networks
- It learns the relationship between topology, traffic, routing and the resulting performance of the network
- Generalizes to **unseen** topologies, routings and traffics

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 ACM SOSR 2019*
https://arxiv.org/pdf/1901.08113.pdf

How does RouteNet work?

RouteNet Architecture



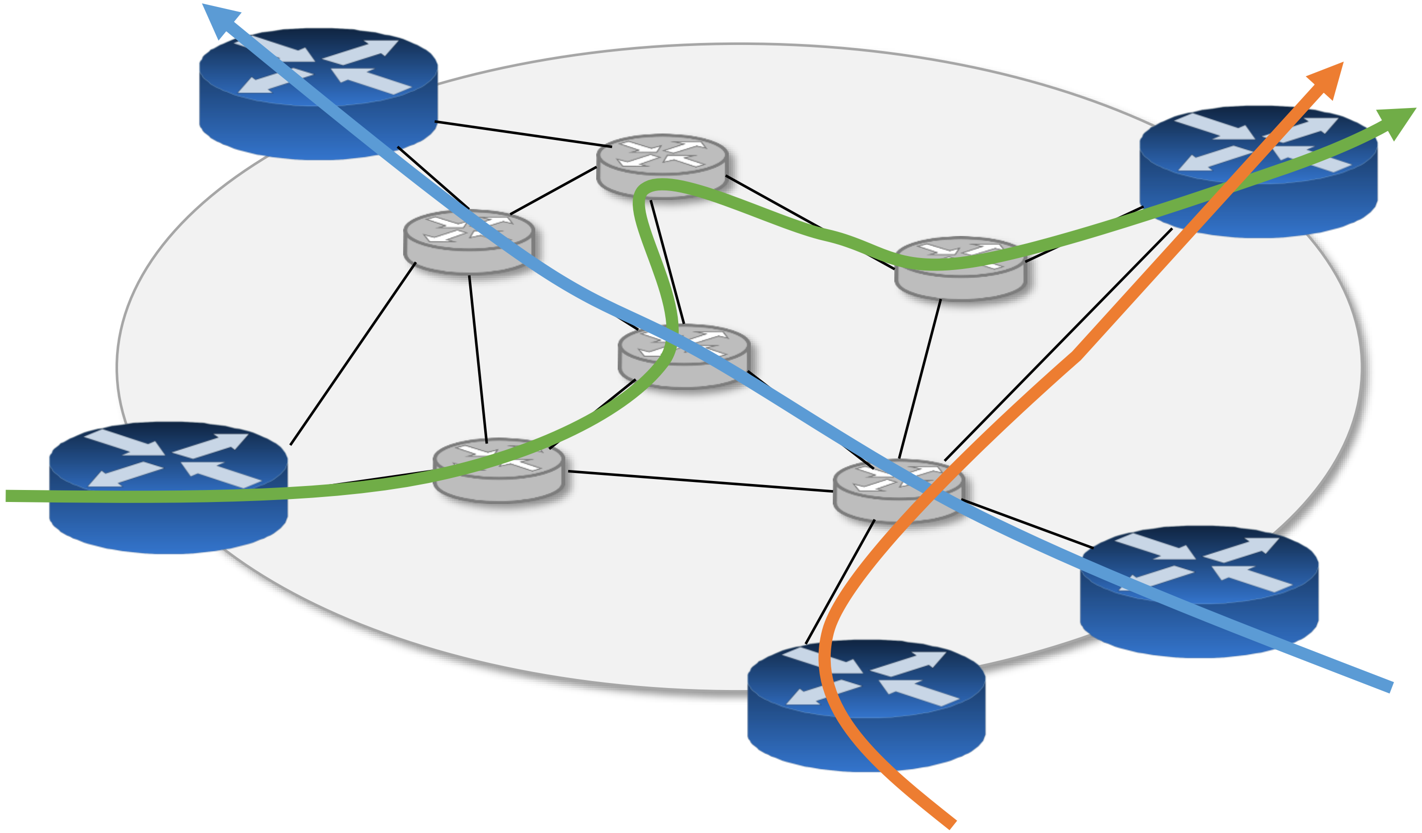
Input: x_p, x_l, $\mathcal{R}$
Output: h_p^T, h_l^T, $\hat{y}_p$

```
foreach p ∈ R do
    h_p^0 ← [x_p, 0 ..., 0]
end
foreach l ∈ N do
    h_l^0 ← [x_l, 0 ..., 0]
end
for t = 1 to T do
    foreach p ∈ R do
        foreach l ∈ p do
            h_p^t ← RNN_t(h_p^t, h_l^t)
            m̃_{p,l}^{t+1} ← h_p^t
        end
        h_p^{t+1} ← h_p^t
    end
    foreach l ∈ N do
        m_l^{t+1} ← Σ_{p:l∈p} m̃_{p,l}^{t+1}
        h_l^{t+1} ← U_t(h_l^t, m_l^{t+1})
    end
end
ŷ_p ← F_p(h_p)
```

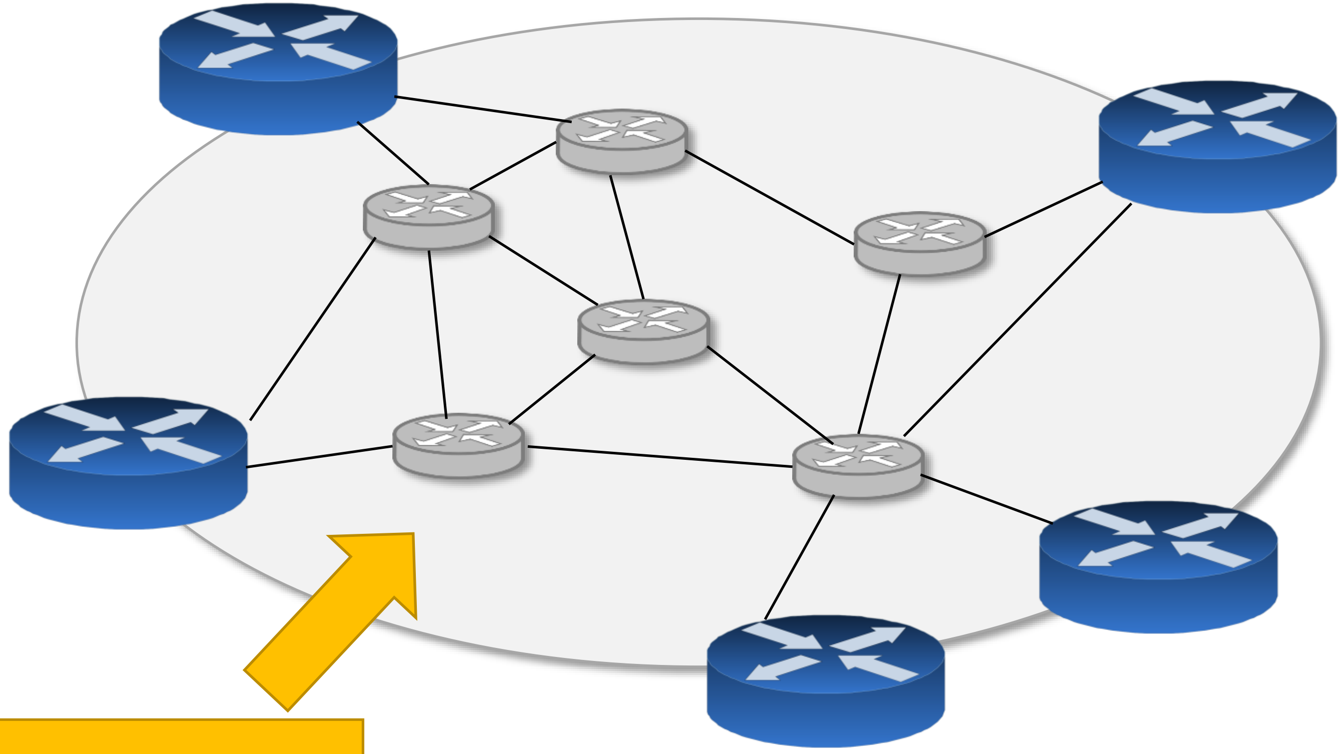
- RoueNet models the relationship between links and paths
  - State of a links depends on the paths that traverse that link
  - State of a paths depends on the links of that path
- This is a circular dependency

$$h_{l_i} = f(h_{p_1}, \ldots, h_{p_j}), \quad l_i \in p_k, k = 1, \ldots, j$$
$$h_{p_k} = g(h_{l_{k(0)}}, \ldots, h_{l_{k(|p_k|)}})$$

RouteNet: A Digital Twin of the Network

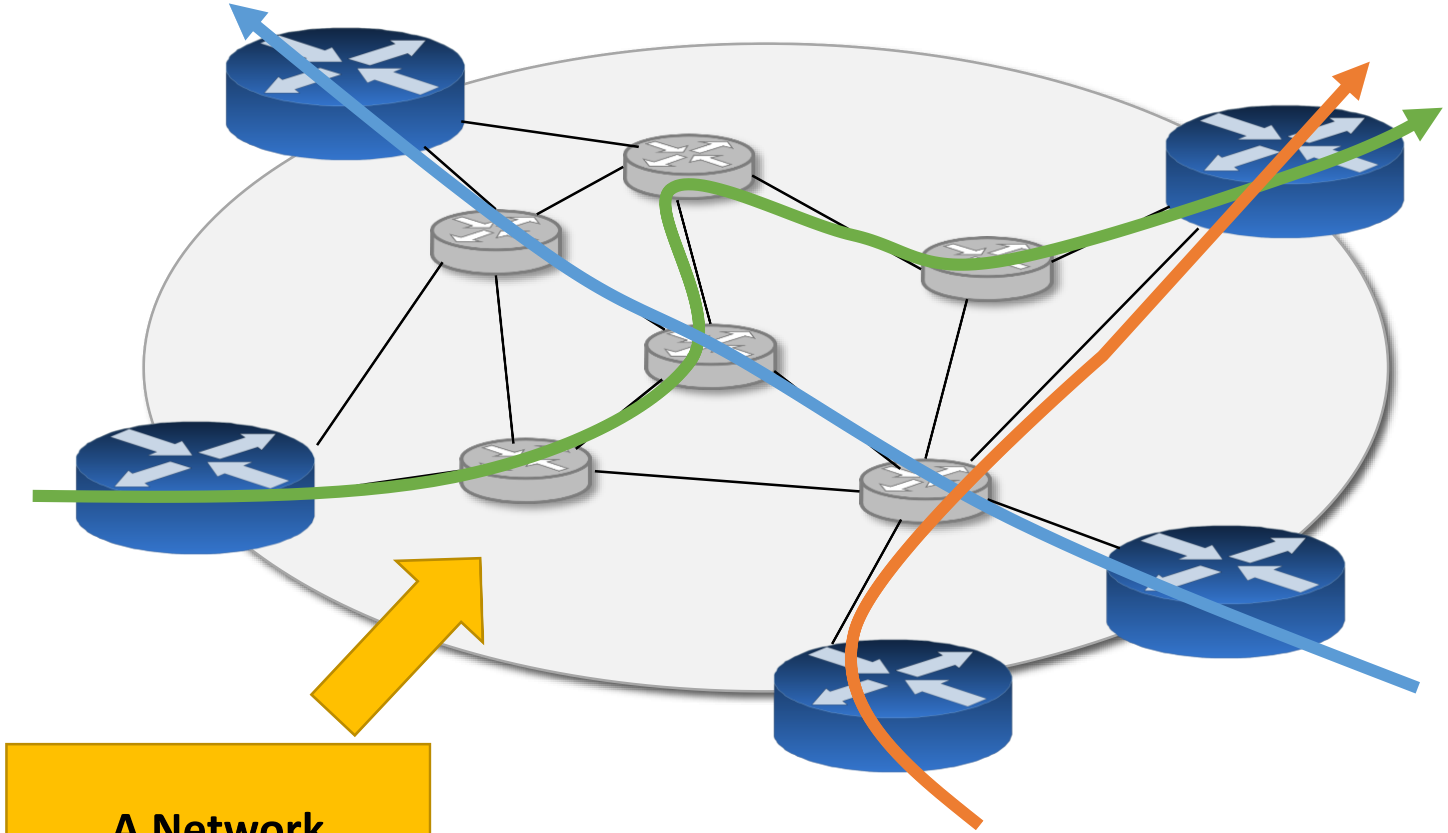


RouteNet: A Digital Twin of the Network



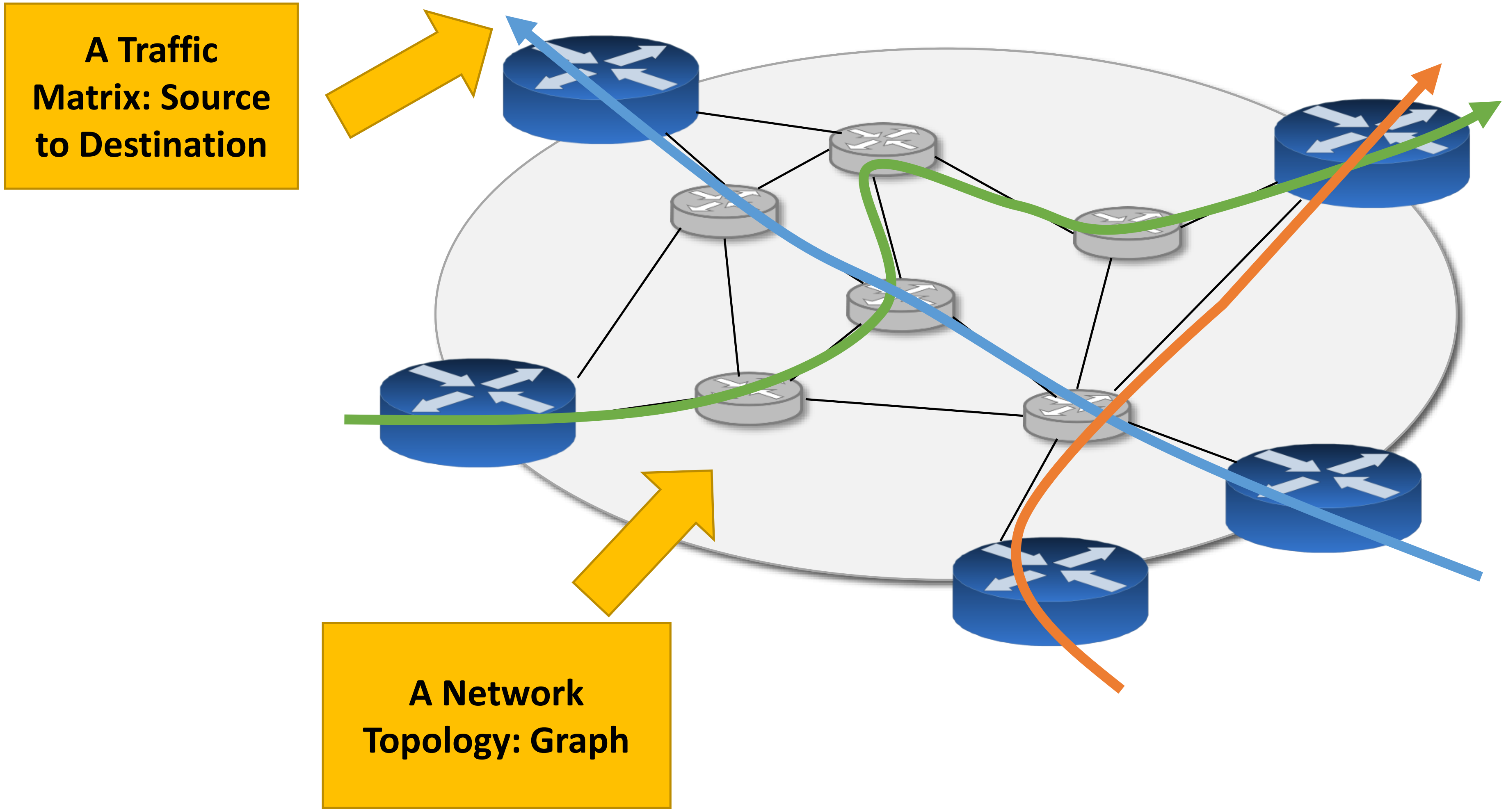
A Network Topology: Graph

RouteNet: A Digital Twin of the Network

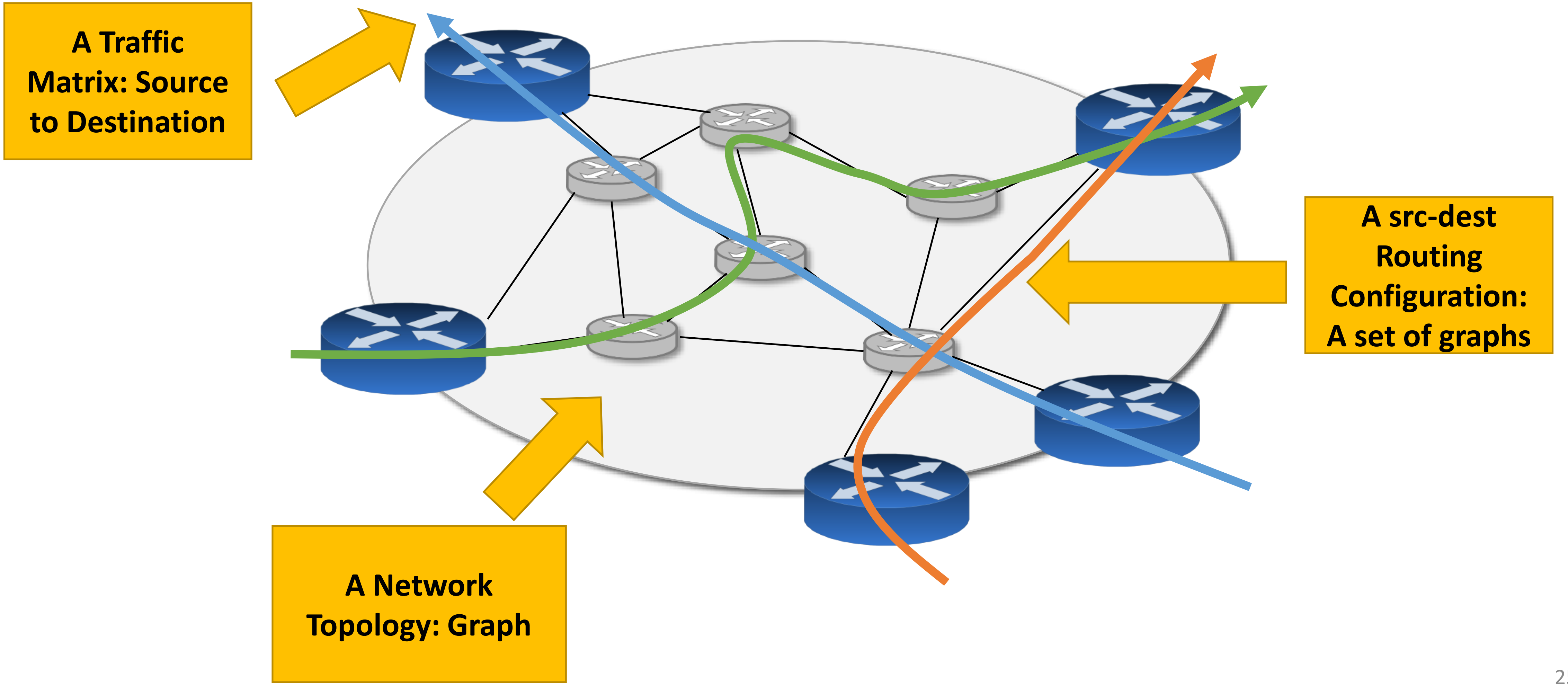


A Network Topology: Graph

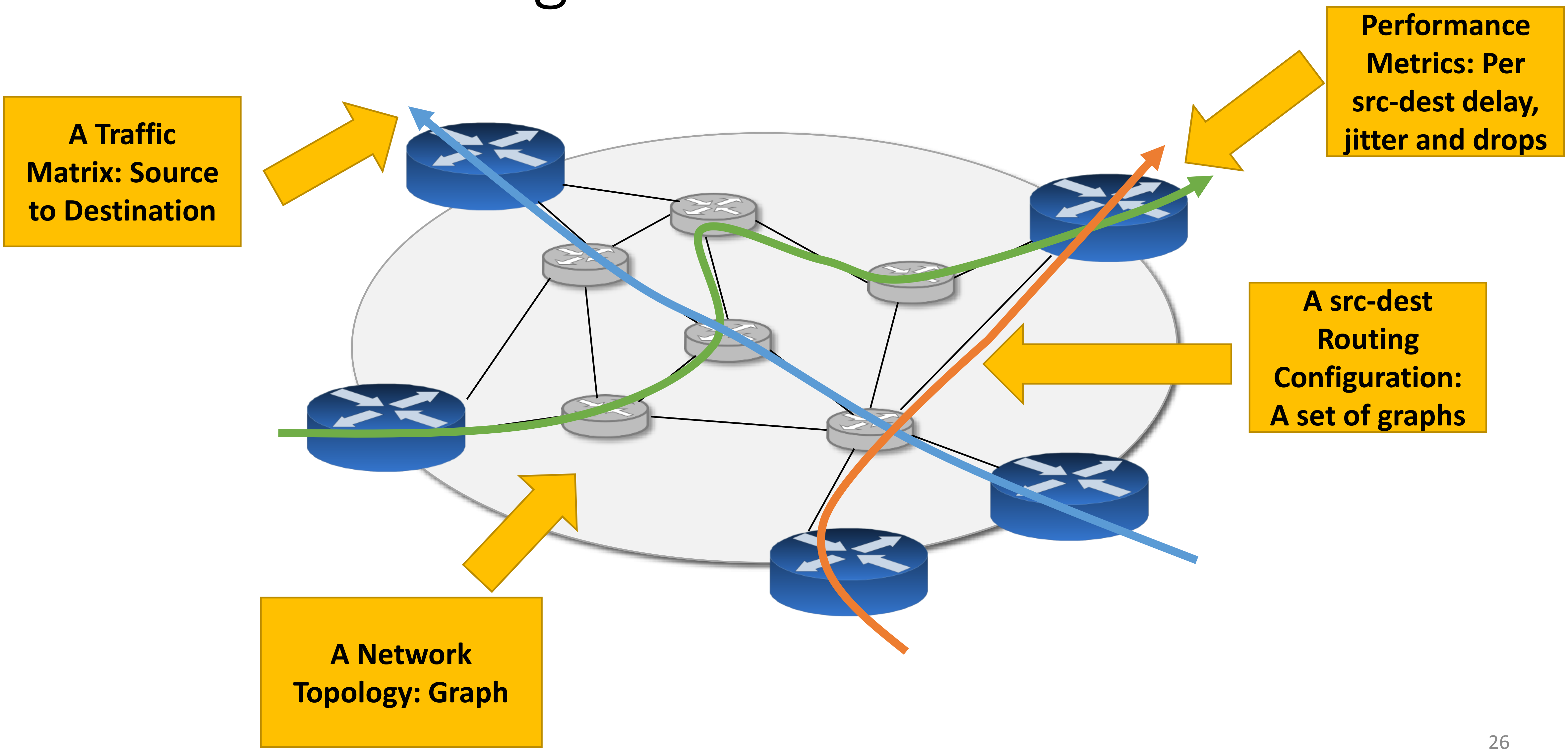
RouteNet: A Digital Twin of the Network



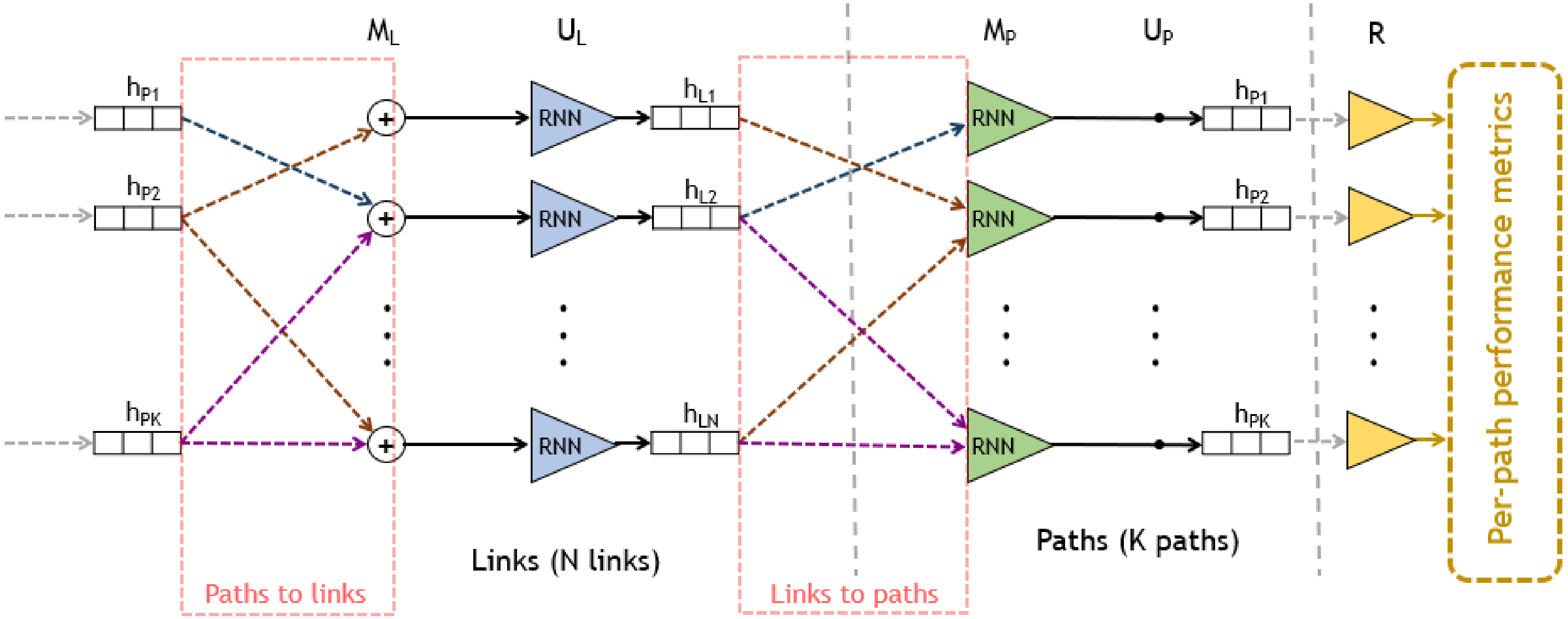
RouteNet: A Digital Twin of the Network



RouteNet: A Digital Twin of the Network



RouteNet Architecture



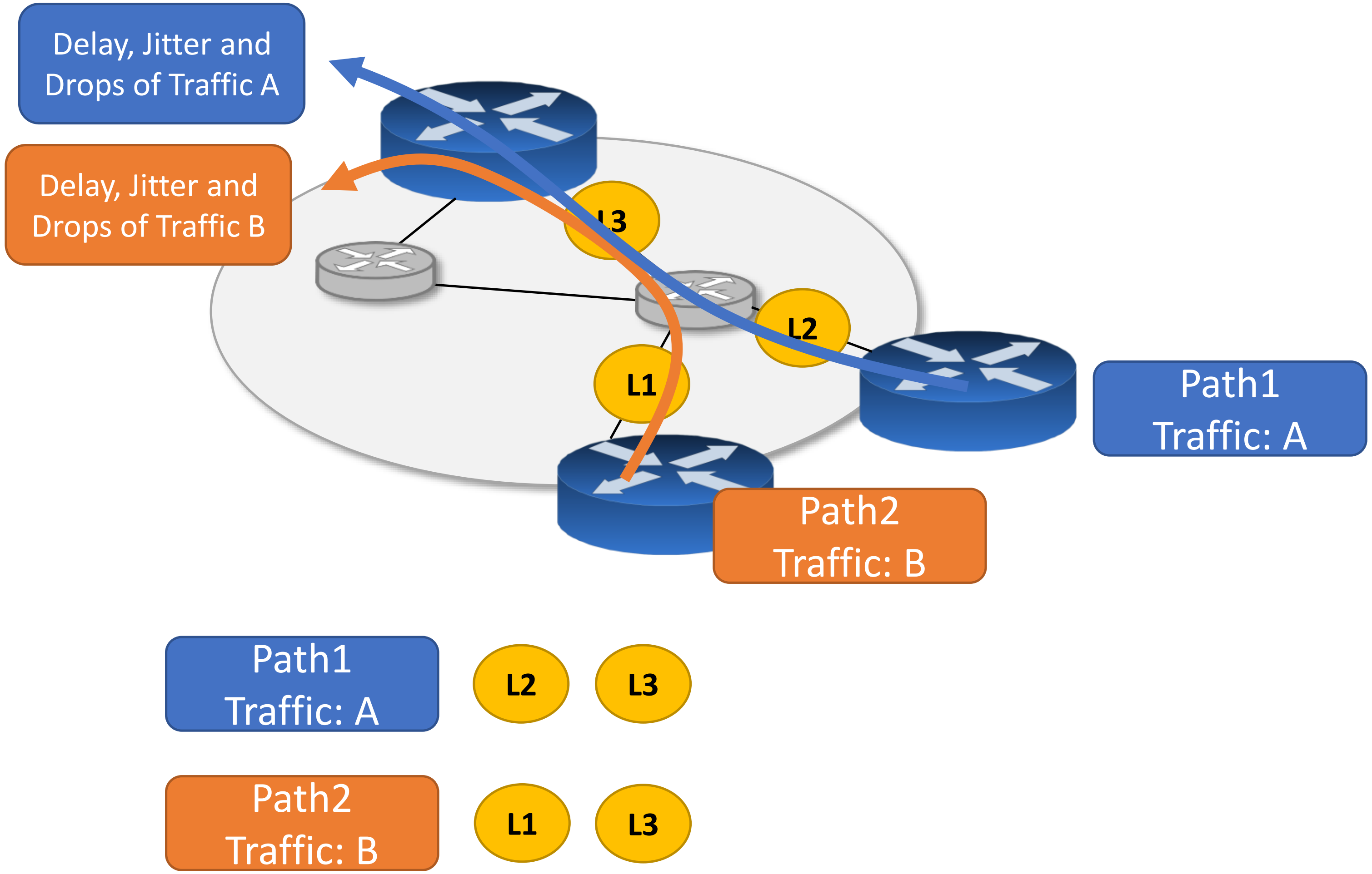
Input: $\mathbf{x}_p$, $\mathbf{x}_l$, $\mathcal{R}$
Output: $\mathbf{h}_p^T$, $\mathbf{h}_l^T$, $\hat{\mathbf{y}}_p$

foreach $p \in \mathcal{R}$ **do**
  $\mathbf{h}_p^0 \leftarrow [\mathbf{x}_p, 0 \ldots, 0]$
end
foreach $l \in \mathcal{N}$ **do**
  $\mathbf{h}_l^0 \leftarrow [\mathbf{x}_l, 0 \ldots, 0]$
end
for $t = 1$ *to* T **do**
  foreach $p \in \mathcal{R}$ **do**
    foreach $l \in p$ **do**
      $\mathbf{h}_p^t \leftarrow RNN_t(\mathbf{h}_p^t, \mathbf{h}_l^t)$
      $\tilde{\mathbf{m}}_{p,l}^{t+1} \leftarrow \mathbf{h}_p^t$
    end
    $\mathbf{h}_p^{t+1} \leftarrow \mathbf{h}_p^t$
  end
  foreach $l \in \mathcal{N}$ **do**
    $\mathbf{m}_l^{t+1} \leftarrow \sum_{p:l \in p} \tilde{\mathbf{m}}_{p,l}^{t+1}$
    $\mathbf{h}_l^{t+1} \leftarrow U_t\left(\mathbf{h}_l^t, \mathbf{m}_l^{t+1}\right)$
  end
end
$\hat{\mathbf{y}}_p \leftarrow F_p(\mathbf{h}_p)$

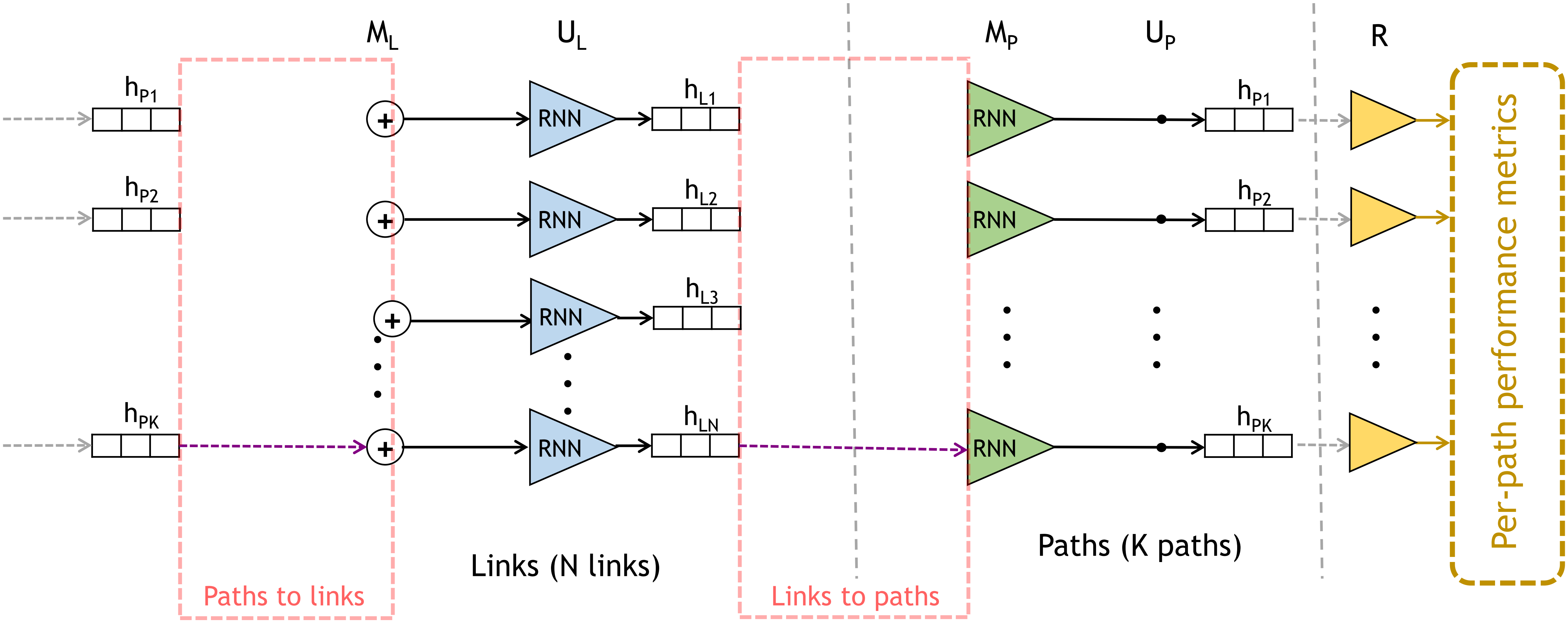
- RoueNet models the relationship between links and paths
  - State of a links depends on the paths that traverse that link
  - State of a paths depends on the links of that path
- This is a circular dependency

$$\mathbf{h}_{l_i} = f(\mathbf{h}_{p_1}, \ldots, \mathbf{h}_{p_j}), \quad l_i \in p_k, k = 1, \ldots, j$$
$$\mathbf{h}_{p_k} = g(\mathbf{h}_{l_{k(0)}}, \ldots, \mathbf{h}_{l_{k(|p_k|)}})$$

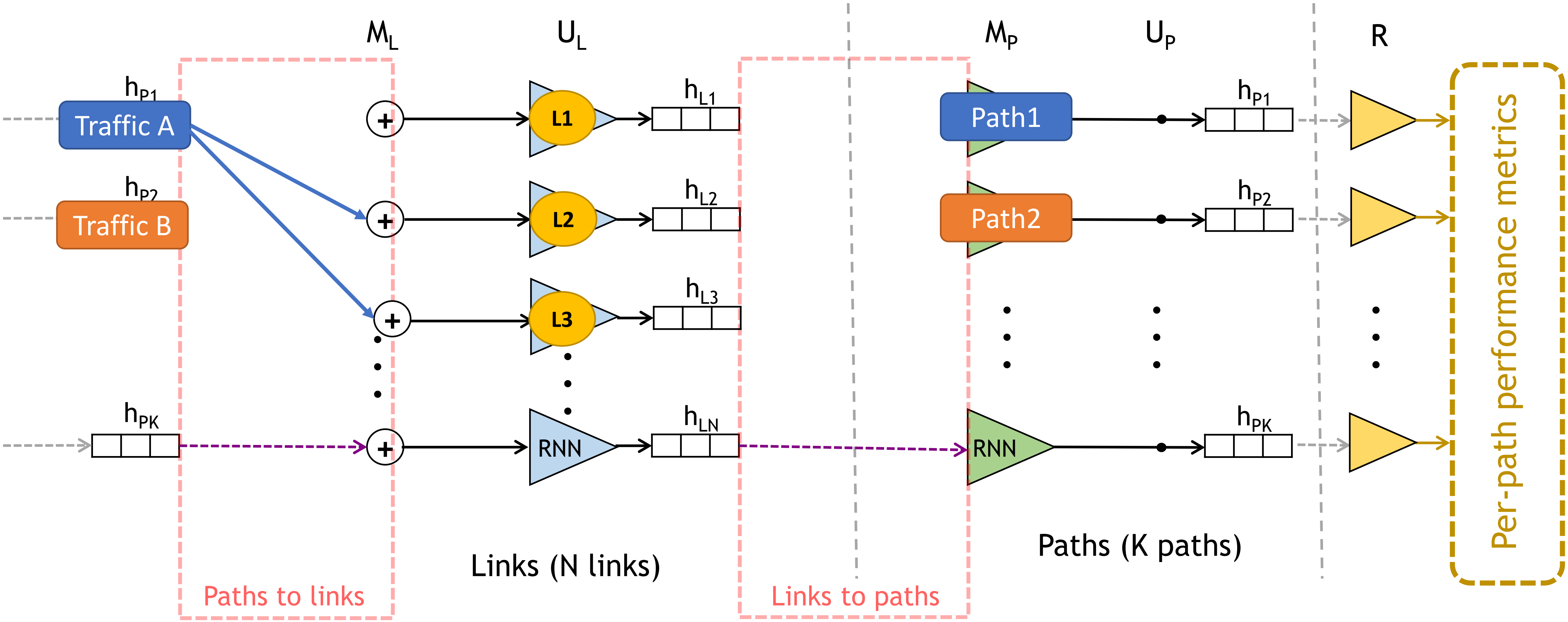
RouteNet: A working example



RouteNet Architecture

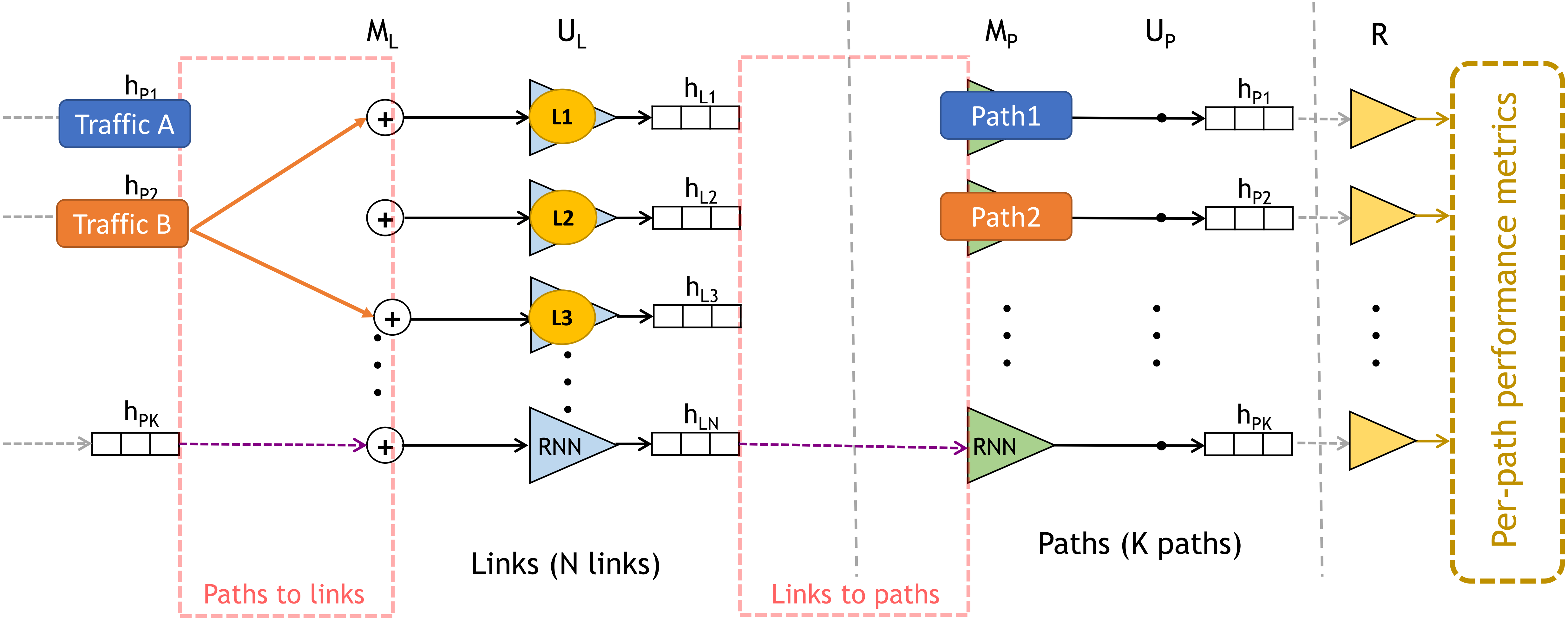


RouteNet Architecture



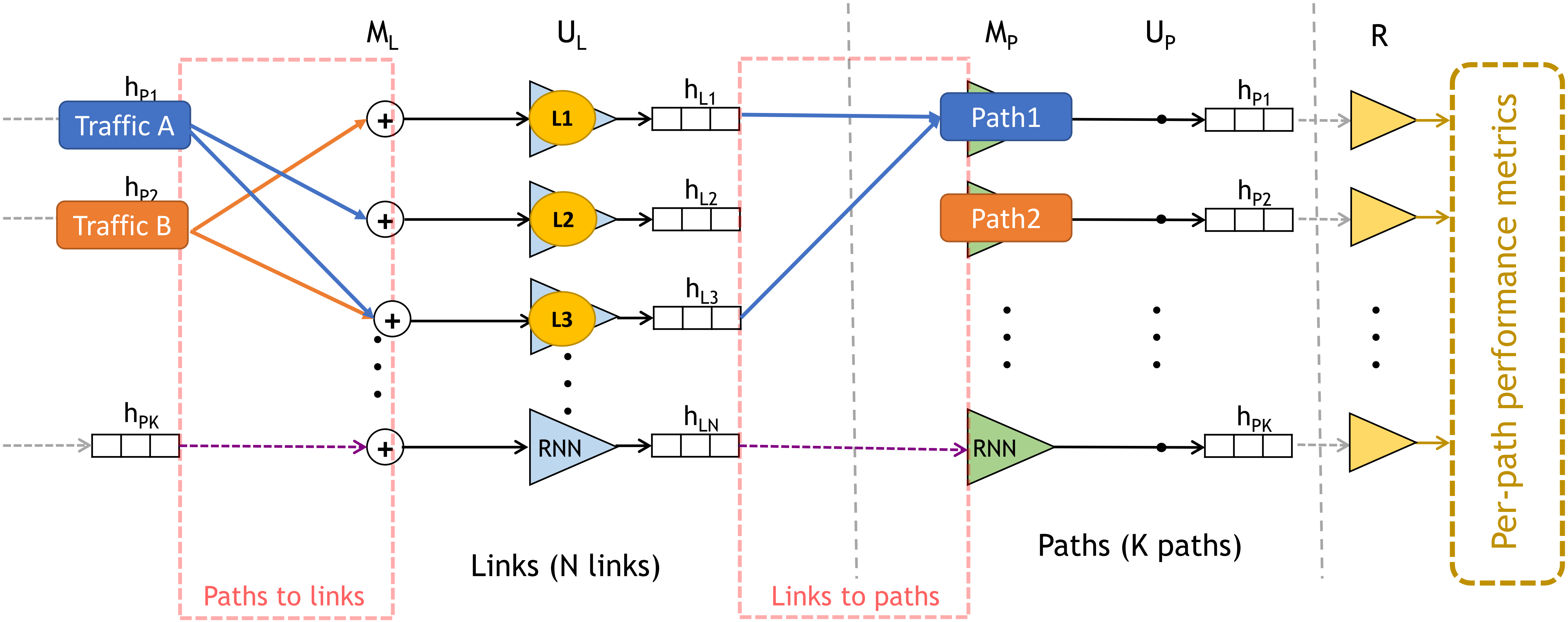
State of a links depends on the paths that traverse that link

RouteNet Architecture



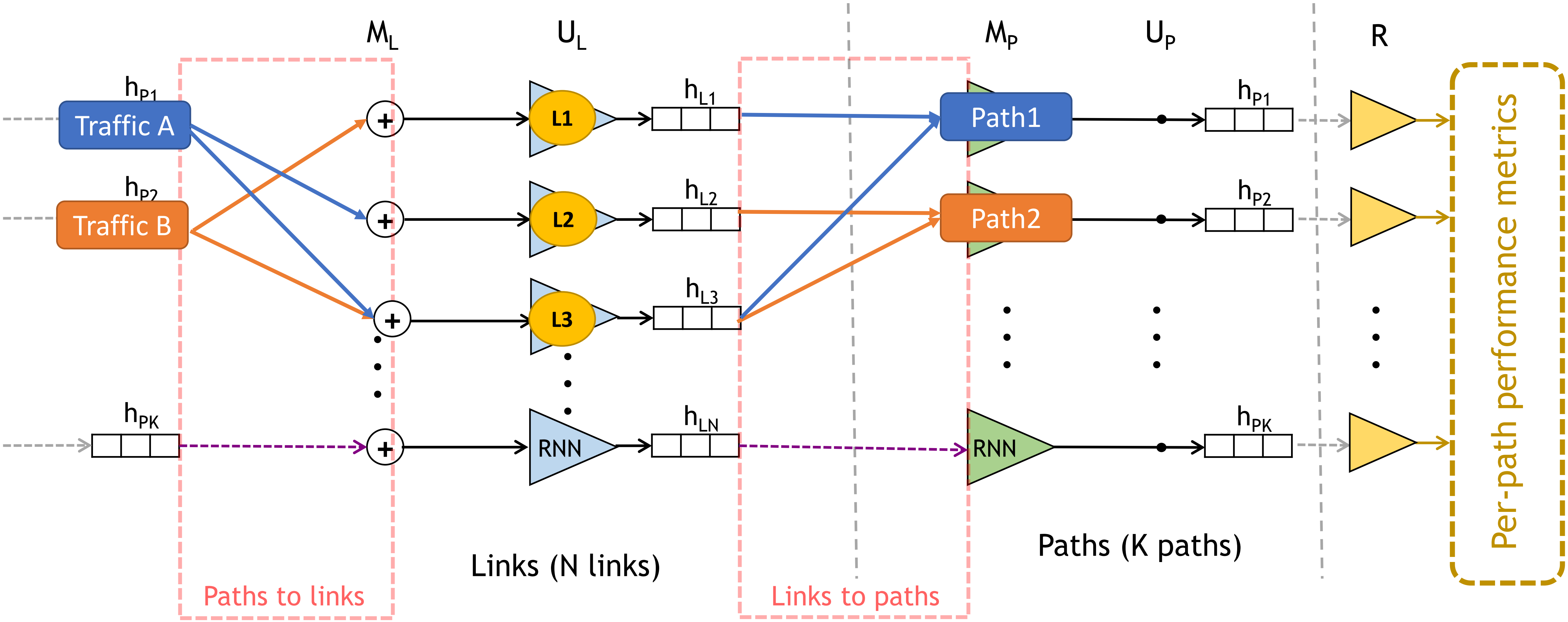
State of a links depends on the paths that traverse that link

RouteNet Architecture



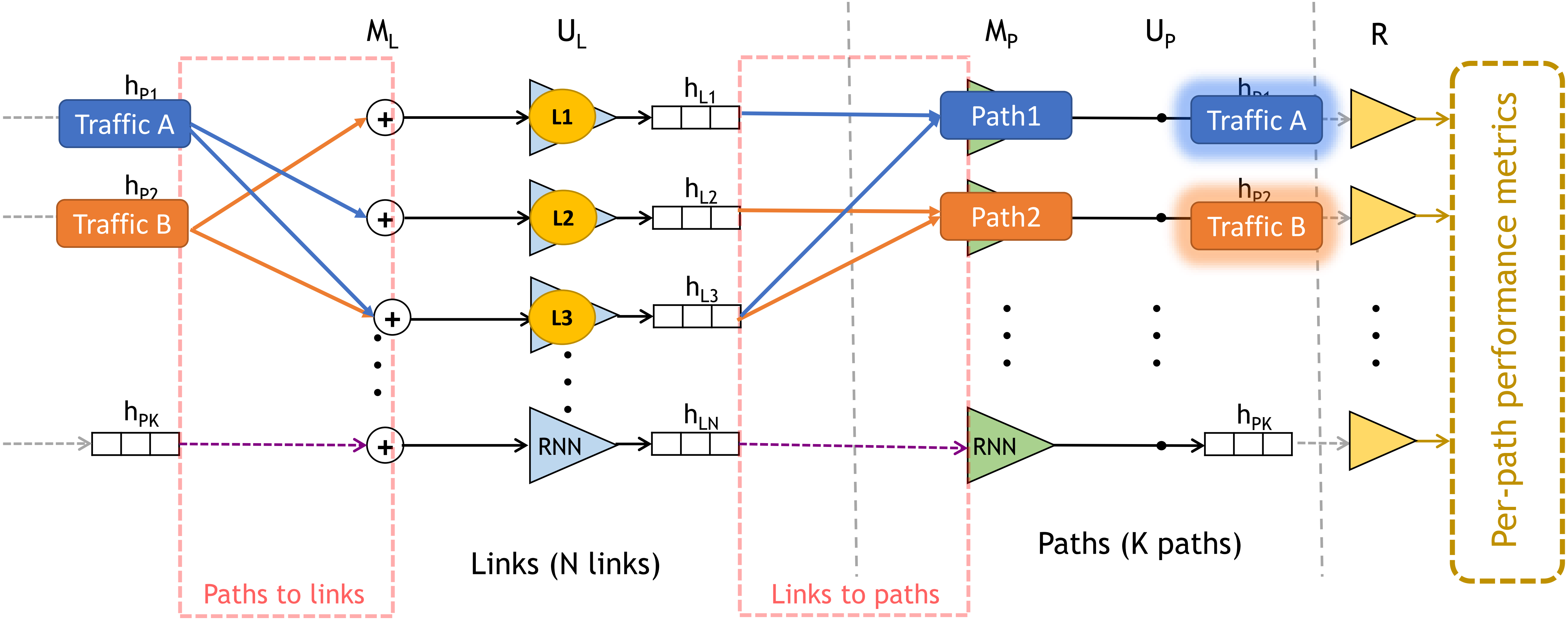
State of a paths depends on the links of that path

RouteNet Architecture

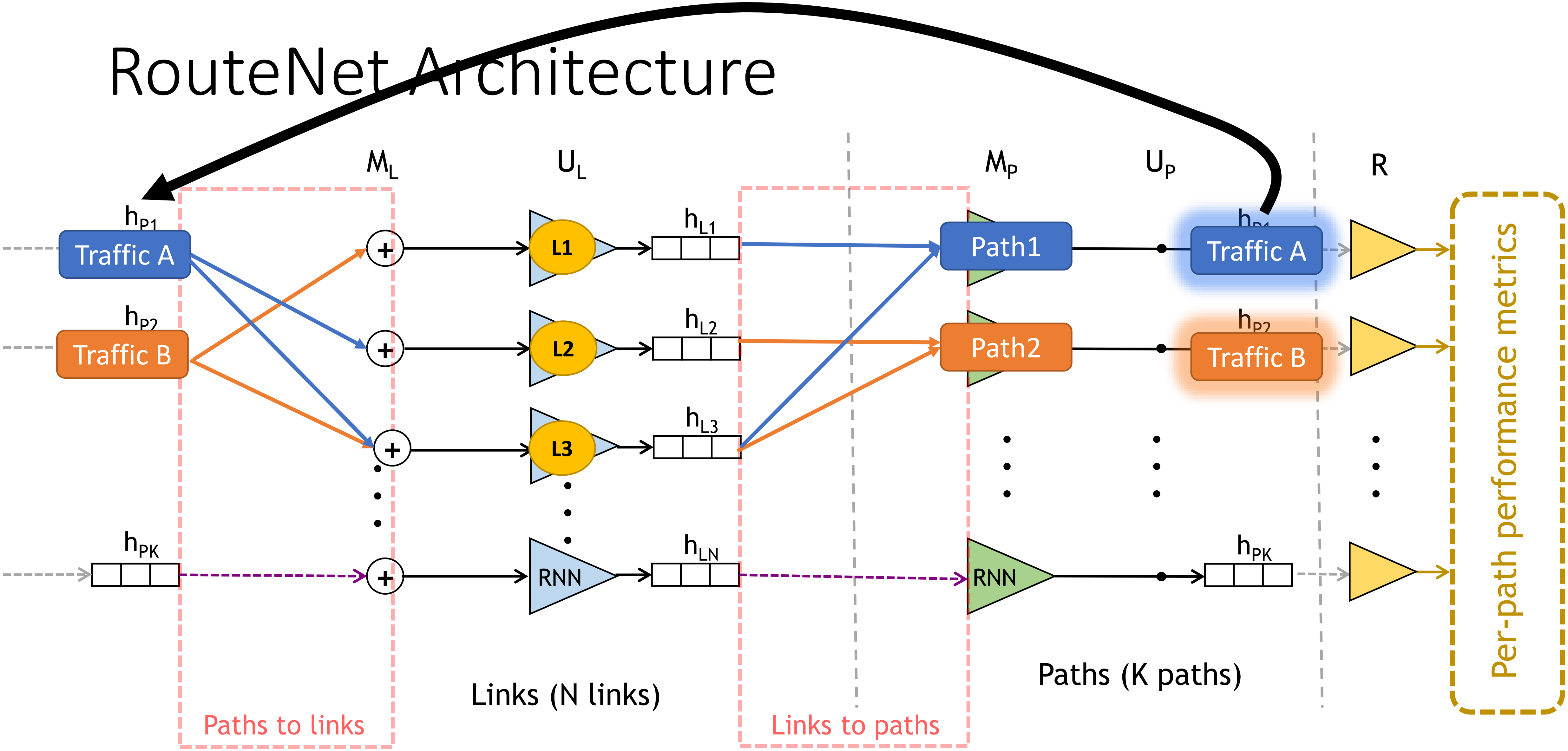


State of a paths depends on the links of that path

RouteNet Architecture

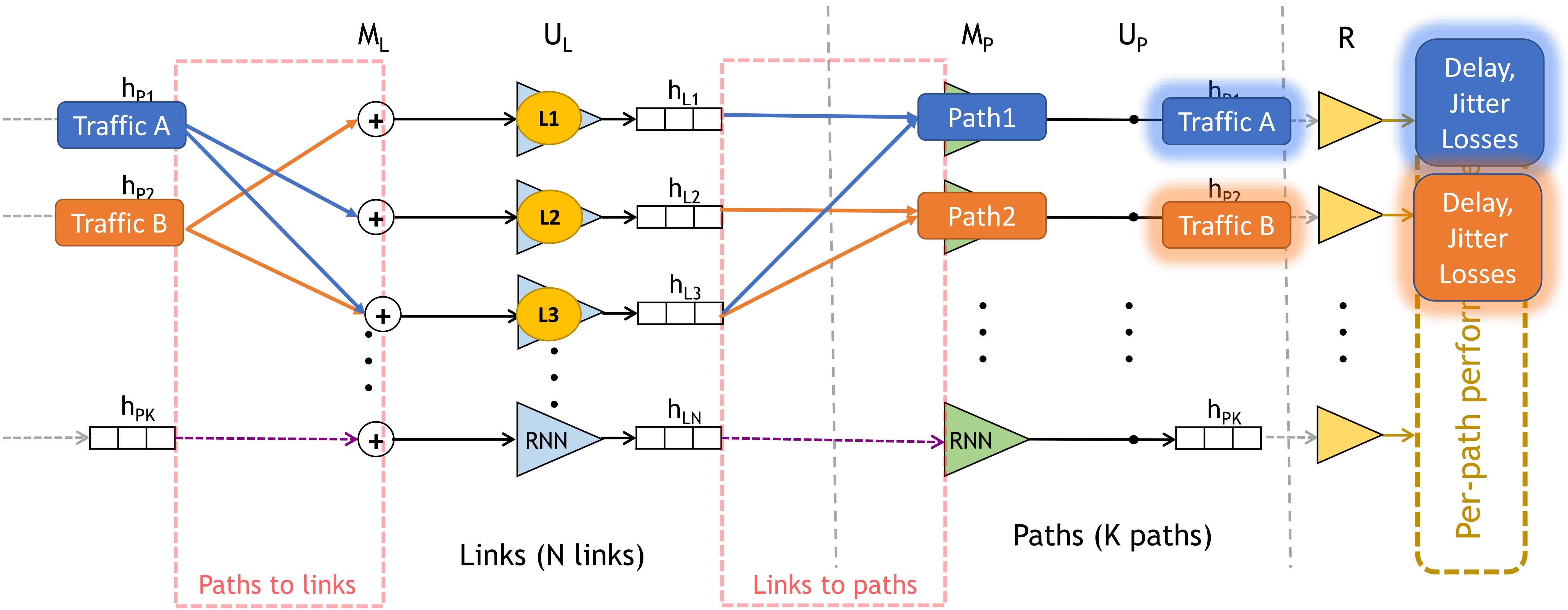


RouteNet Architecture

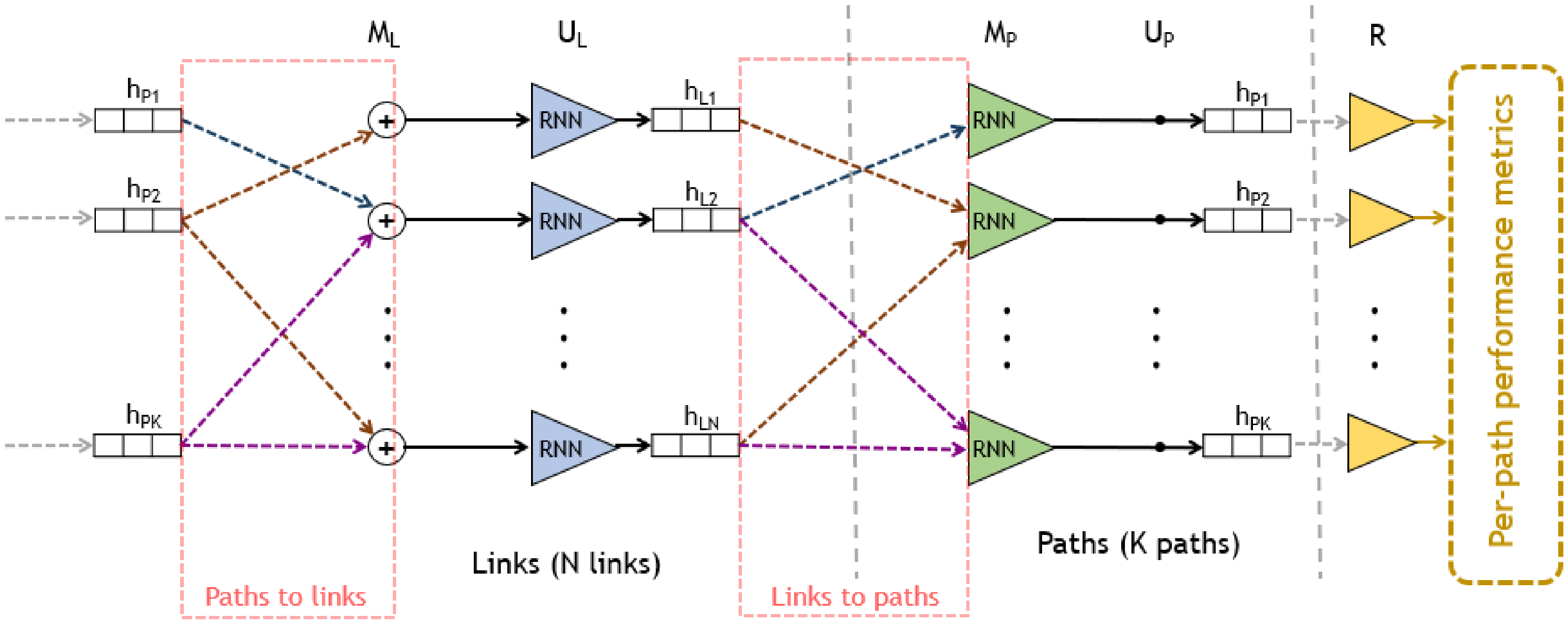


This is a circular dependency

RouteNet Architecture



RouteNet Architecture



Input: $\mathbf{x}_p$, $\mathbf{x}_l$, $\mathcal{R}$
Output: $\mathbf{h}_p^T$, $\mathbf{h}_l^T$, $\hat{\mathbf{y}}_p$

foreach $p \in \mathcal{R}$ **do**
  $\mathbf{h}_p^0 \leftarrow [\mathbf{x}_p, 0 \ldots, 0]$
end
foreach $l \in \mathcal{N}$ **do**
  $\mathbf{h}_l^0 \leftarrow [\mathbf{x}_l, 0 \ldots, 0]$
end
for $t = 1$ *to* T **do**
  foreach $p \in \mathcal{R}$ **do**
    foreach $l \in p$ **do**
      $\mathbf{h}_p^t \leftarrow RNN_t(\mathbf{h}_p^t, \mathbf{h}_l^t)$
      $\tilde{\mathbf{m}}_{p,l}^{t+1} \leftarrow \mathbf{h}_p^t$
    end
    $\mathbf{h}_p^{t+1} \leftarrow \mathbf{h}_p^t$
  end
  foreach $l \in \mathcal{N}$ **do**
    $\mathbf{m}_l^{t+1} \leftarrow \sum_{p: l \in p} \tilde{\mathbf{m}}_{p,l}^{t+1}$
    $\mathbf{h}_l^{t+1} \leftarrow U_t\left(\mathbf{h}_l^t, \mathbf{m}_l^{t+1}\right)$
  end
end
$\hat{\mathbf{y}}_p \leftarrow F_p(\mathbf{h}_p)$

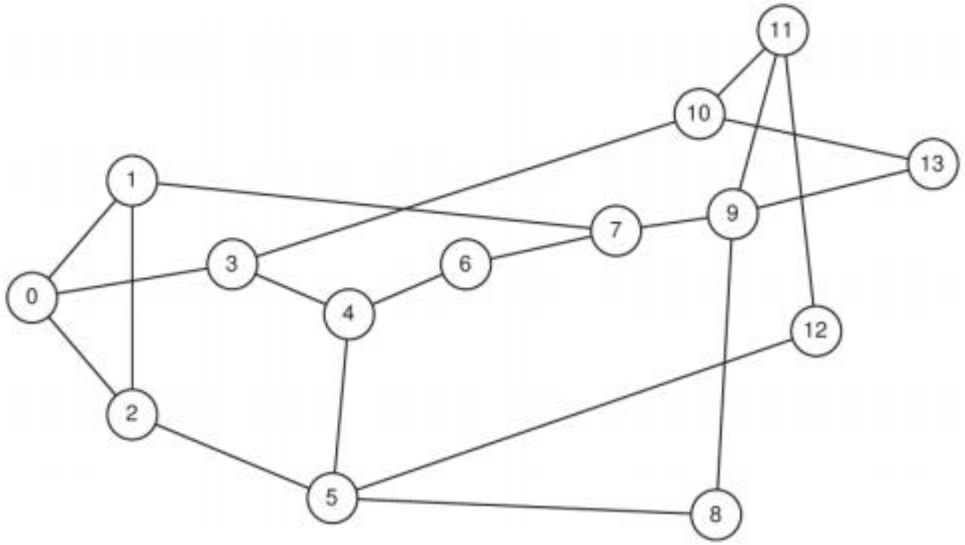
Graph Neural Networks are **not a black-box** and require a **custom architecture** for each problem we are modeling. This needs to be done by a **ML & Networking expert.**

$$\mathbf{h}_{l_i} = f(\mathbf{h}_{p_1}, \ldots, \mathbf{h}_{p_j}), \quad l_i \in p_k, k = 1, \ldots, j$$
$$\mathbf{h}_{p_k} = g(\mathbf{h}_{l_{k(0)}}, \ldots, \mathbf{h}_{l_{k(|p_k|)}})$$

How accurate is RouteNet?

RouteNet: Dataset

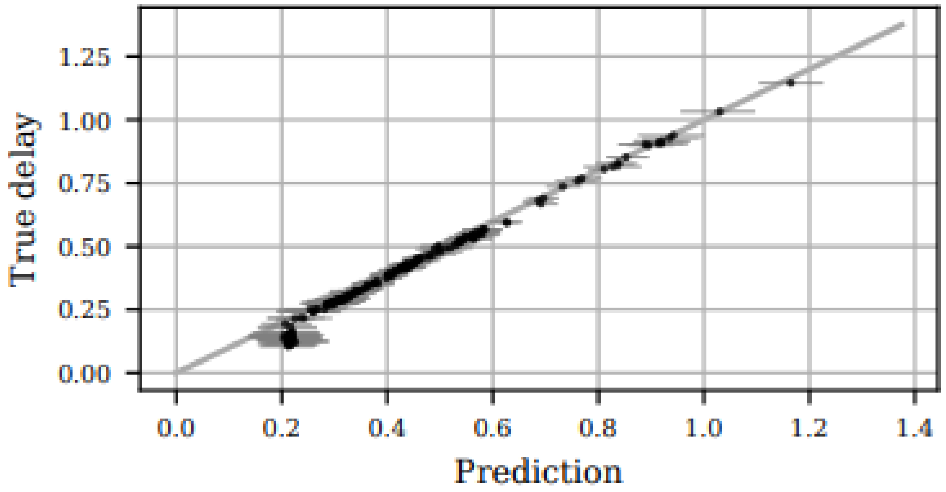
- Dataset obtained with simulation
  - Omnet++
  - Event per-packet simulator that considers queuing
- Trained with the NSFnet 14-node topology
- 260k samples of random (uniform)
  - Traffic Matrices
  - Routing Configurations
  - Resulting per-packet average delay, jitter and losses



NSFnet Topology

RouteNet: Accuracy

	NSF	
	Delay	**Jitter**
R^2	0.99	0.98
ρ	0.998	0.993

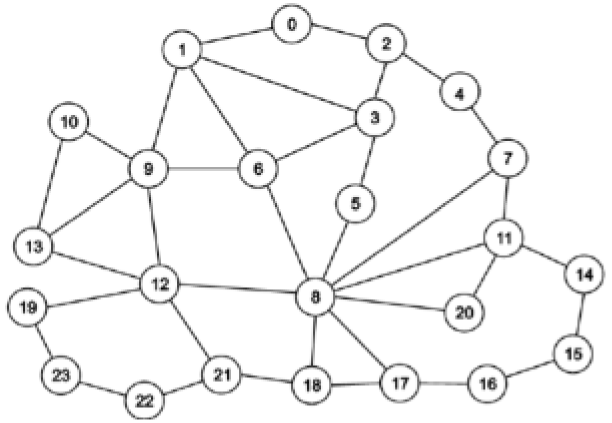


- **RouteNet achieves good accuracy (R^2~99)**
- The Readout is used as dropout to prevent overfitting
- Transfer learning is used to improve learning for jitter and drops.

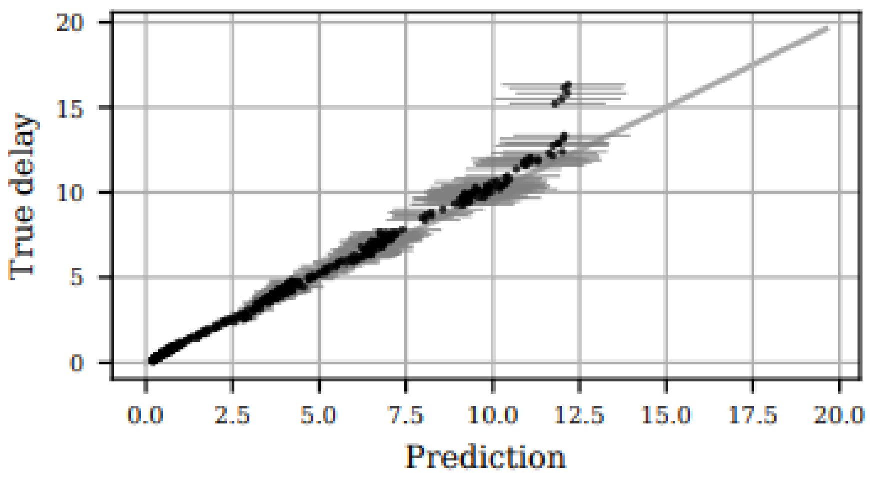
Can RouteNet generalize to unseen topologies?

RouteNet: Generalization

	NSF		Geant2	
	Delay	Jitter	Delay	Jitter
R^2	0.99	0.98	0.97	0.86
ρ	0.998	0.993	0.991	0.942



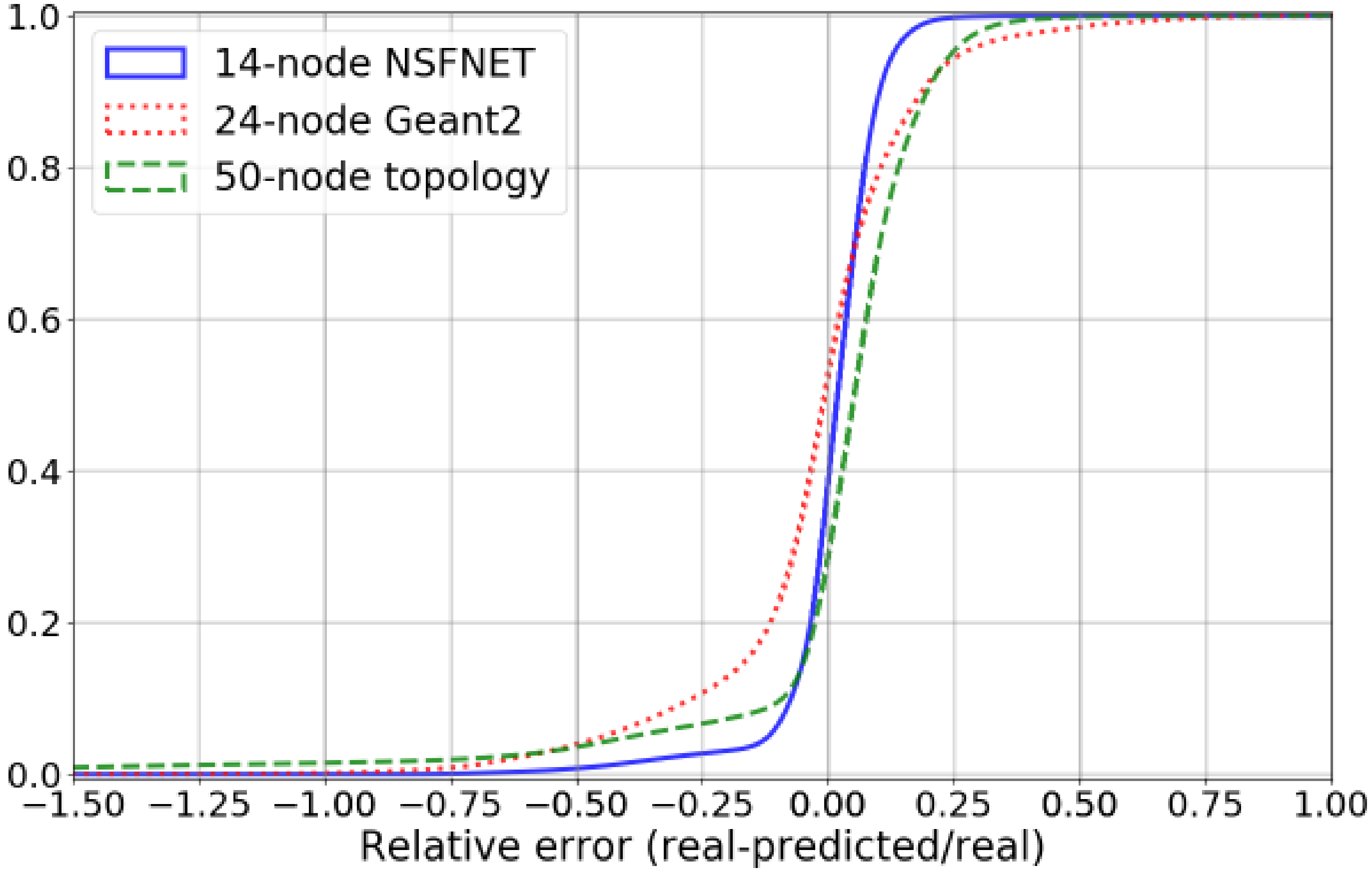
Geant Topology



- What happens when we evaluate RouteNet with an **unseen** topology?
- We tested RouteNet with the 24-node Geant topology
- RouteNet produces accurate estimates for an unseen topologies.

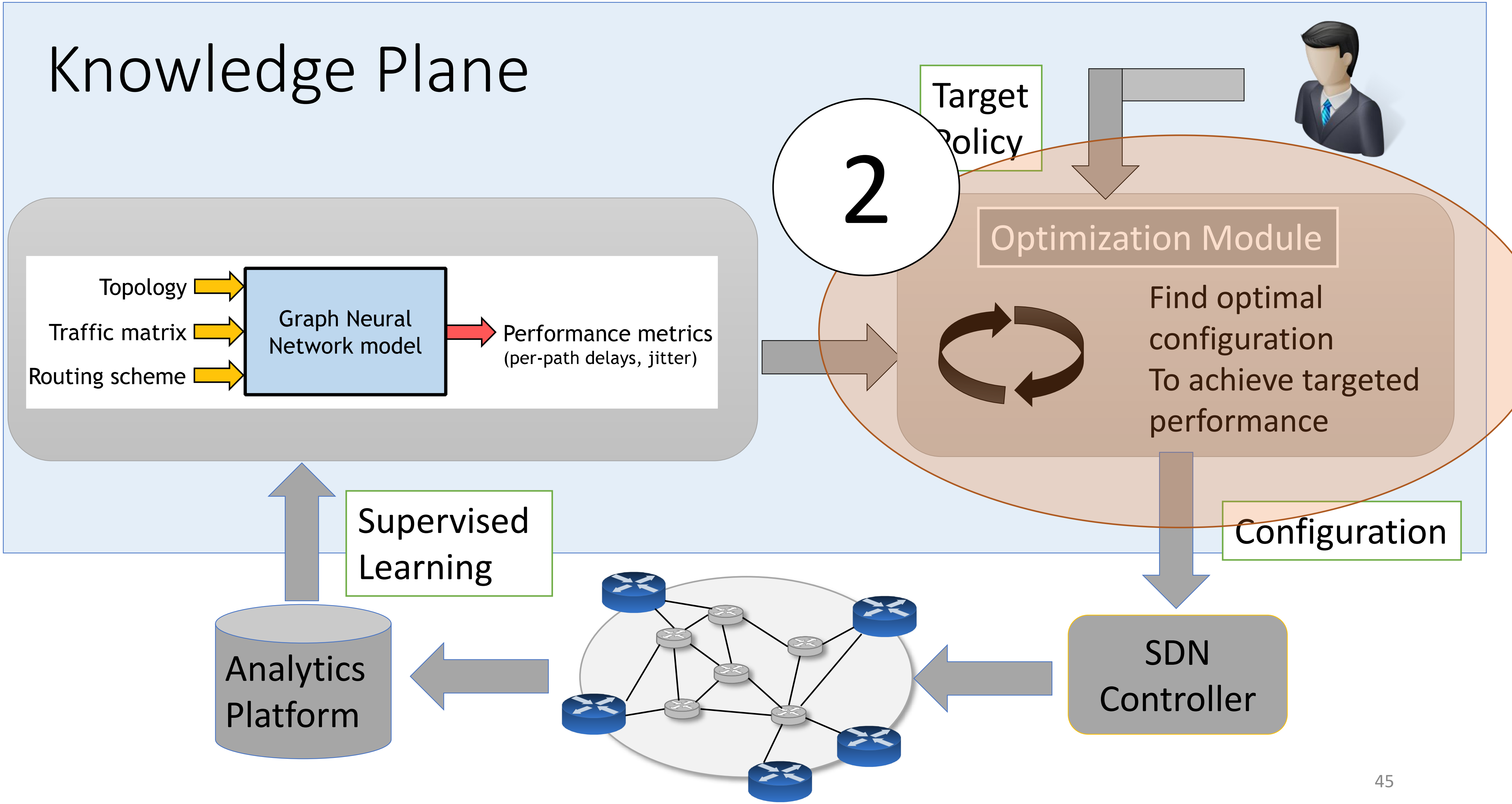
RouteNet can ***generalize*** *to unseen* topologies, routings and traffic matrices.

RouteNet: Generalization

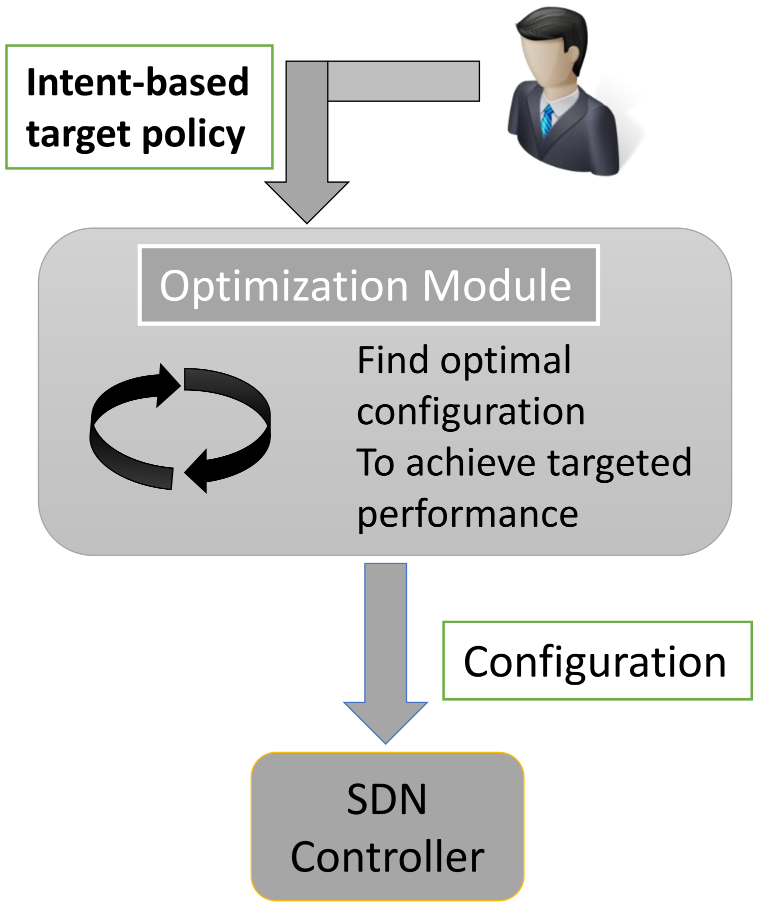


- RouteNet trained with a 24-node topology
- Evaluated in unseen 14-node and 50-node topology
- **RouteNet achieves consistent accuracy**

2.- How we can optimize the network using ML?


Knowledge Plane
Target Policy
2
Optimization Module
Find optimal configuration
To achieve targeted performance
Topology
Traffic matrix
Routing scheme
Graph Neural Network model
Performance metrics
(per-path delays, jitter)
Supervised Learning
Configuration
Analytics Platform
SDN Controller

Network Optimization



- Traditional optimization techniques:
  - Heuristics
  - Routing algorithms (e.g., Shortest path)
  - General-purpose optimization algorithms (e.g., hill climbing)
- Possibility to optimize with

Deep Reinforcement Learning (DRL)

Deep Learning + Reinforcement Learning

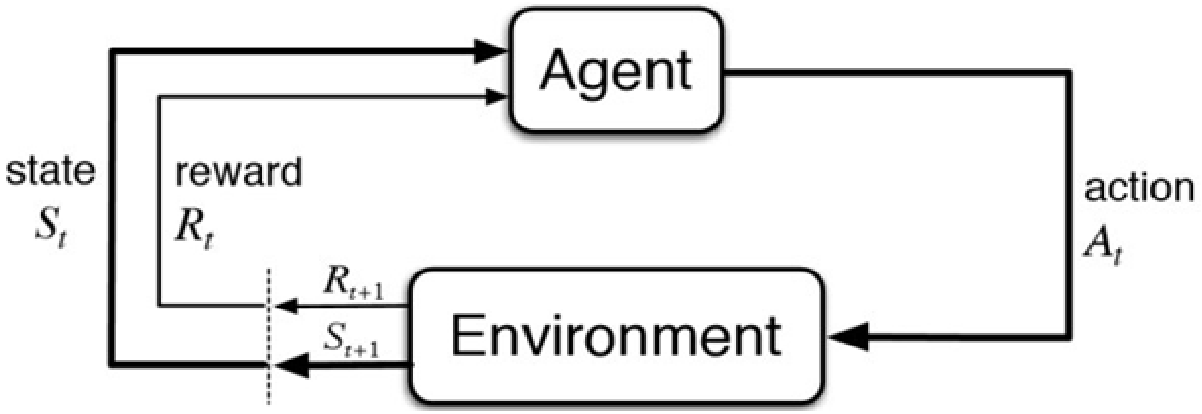
Neural networks (RouteNet)

Optimization algorithm

DRL vs Traditional Optimization

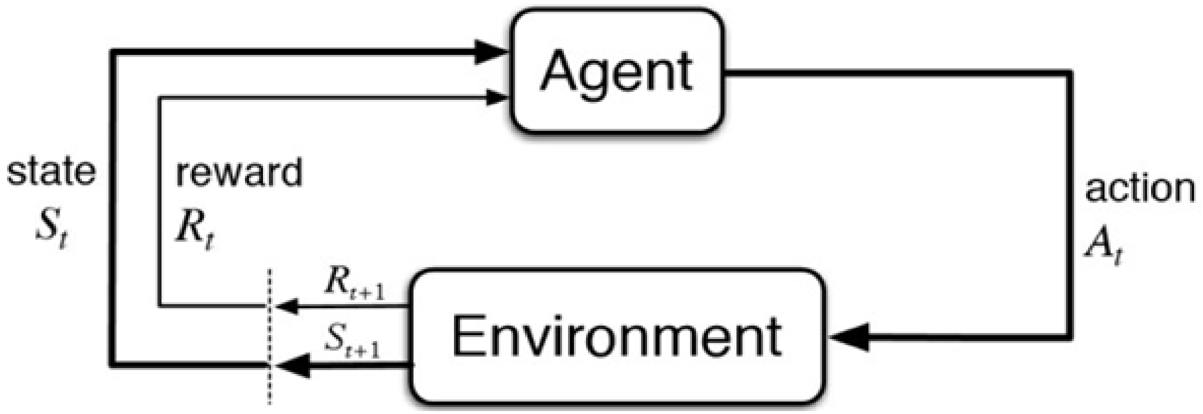
- Traditional optimization techniques have some limitations:
  - Simple heuristics offer suboptimal performance
    (e.g., shortest path, ECMP…)
  - Compute the optimal solution is too costly
    (e.g., evaluate all the possible routing configurations)
- Deep Reinforcement Learning (DRL):
  - In other fields is showing outstanding performance in decision making and automated control problems
  - Fast operation (1-step decision making)

Deep Reinforcement Learning (DRL)



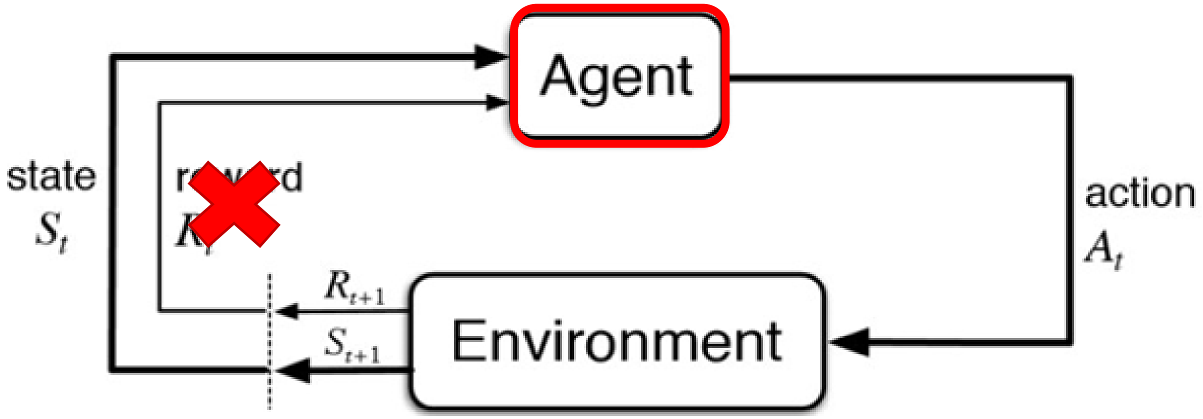
- The DRL agent is modeled by neural networks that learn the target policy
- No domain-specific knowledge (tabula rasa learning)
- The agent only knows the basic rules of the environment (possible actions)

Deep Reinforcement Learning (DRL)



- Training (offline):
  - The learning is achieved by iteratively exploring states and actions (trial and error process)
  - Reward function → Defines the optimization objective
  - Objective → Maximize the cumulative reward (long-term strategy)
  - Smart exploration strategies leveraging the Neural Network models

Deep Reinforcement Learning (DRL)

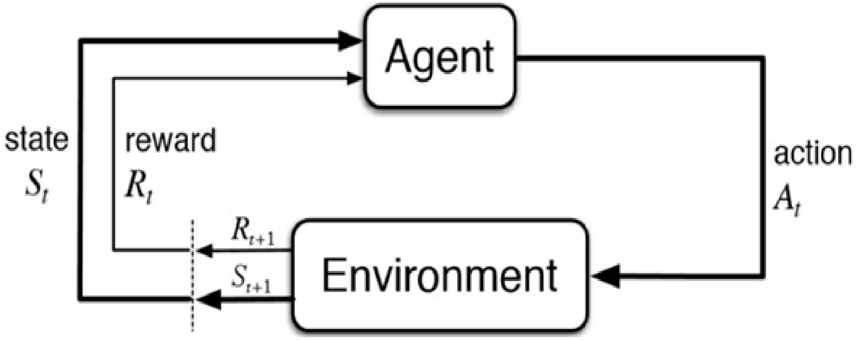


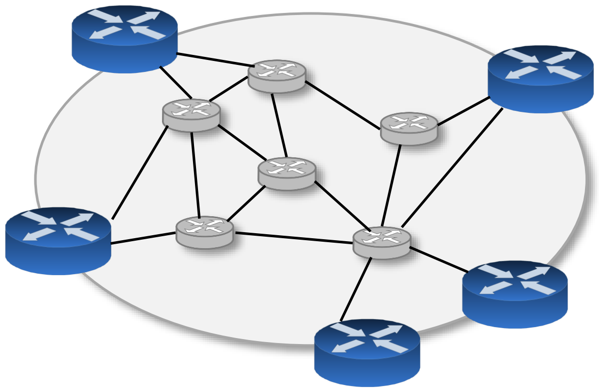
- Operation (online): 1-step decision making

Input state → Output action

The main challenge of DRL for Computer Networks: Generalization

Generalization of DRL

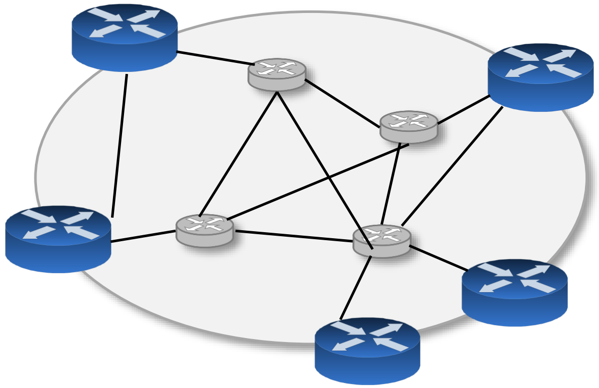




Training

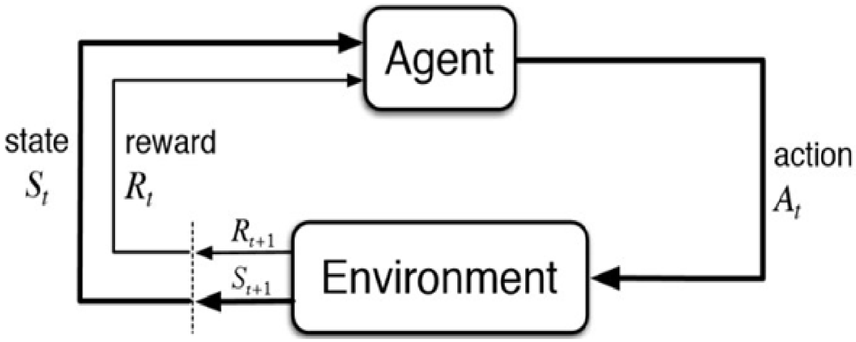
Product

Trained DRL Agent



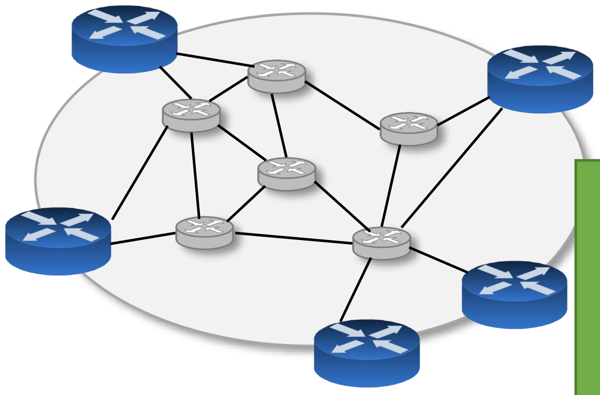
Operation

Generalization of DRL

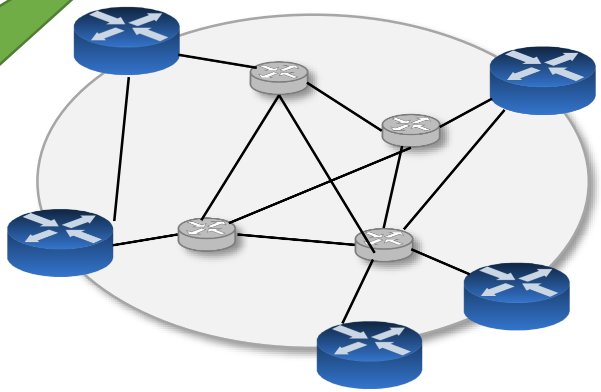


Trained DRL Agent

Product



Wait! I´ve been not been trained for this network. I don´t know what to do!

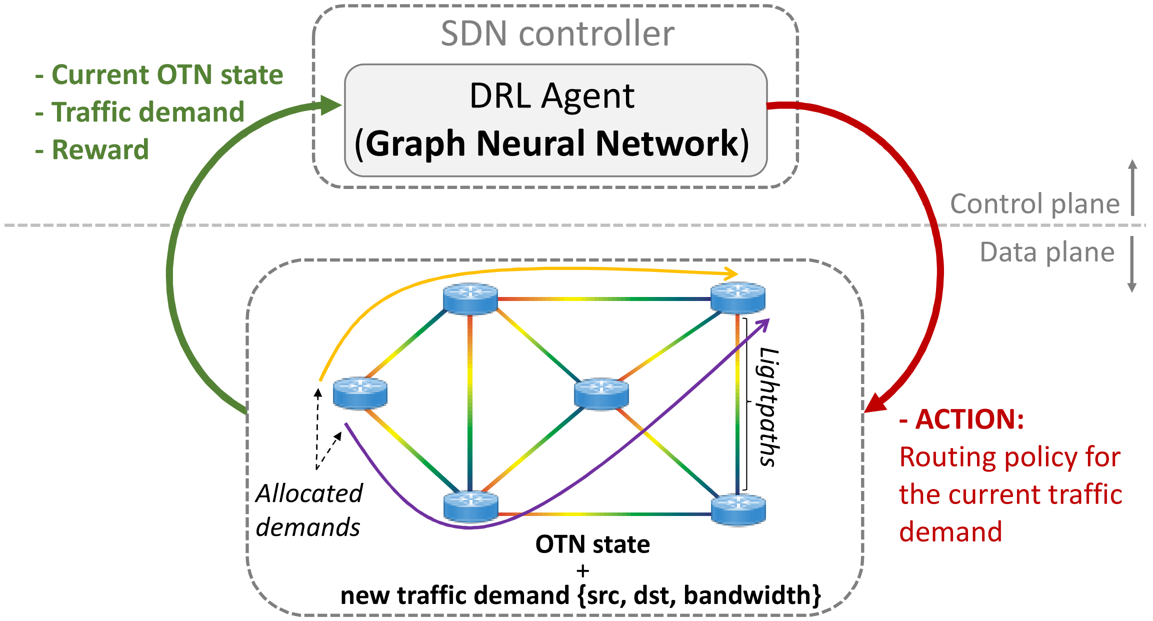


Training

Operation

How can we use DRL techniques with GNN?

DRL+GNN: A network optimization architecture



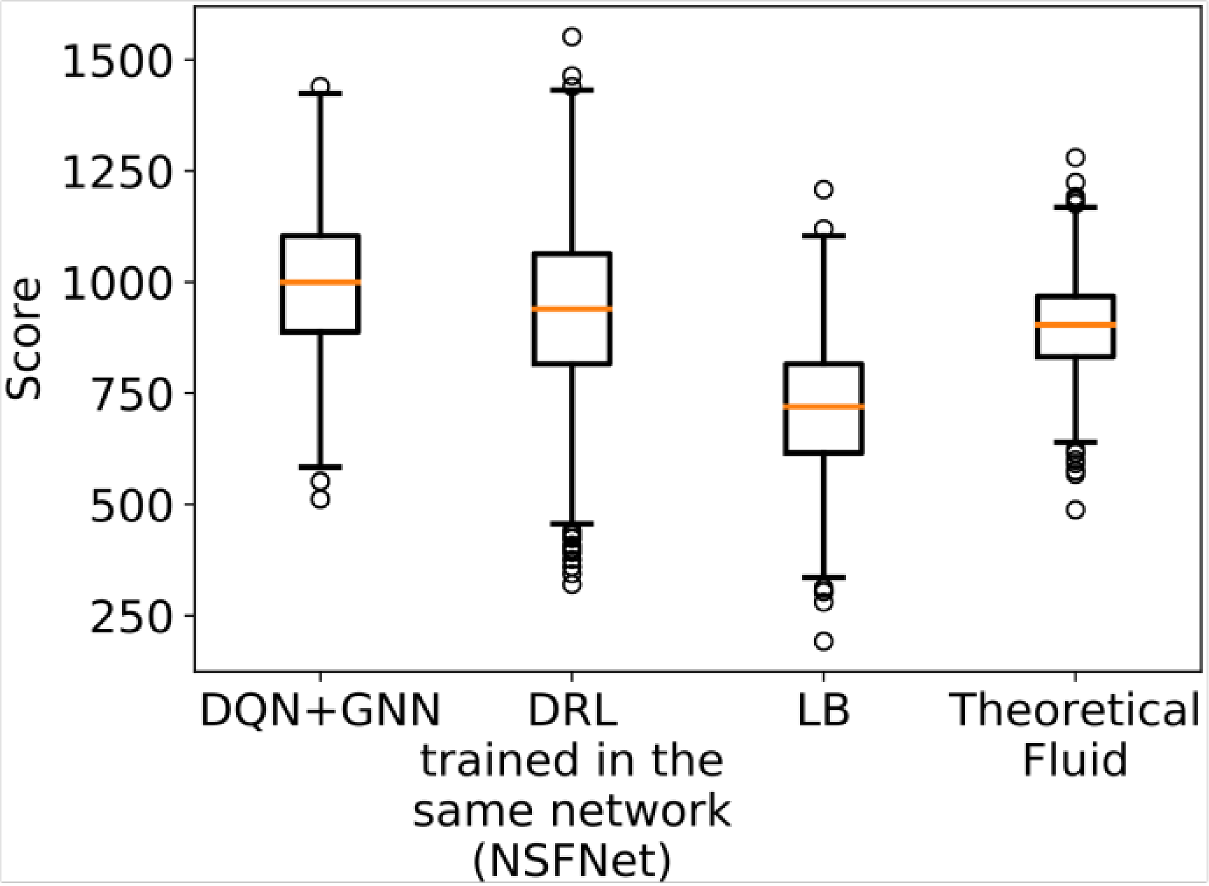
Algorithm 2 DRL Agent operation

```
s, src, dst, bw ← env.init_env()
reward ← 0
k ← 4
agt.mem ← { }
Done ← False
while not Done do
    k_q_values ← { }
    k_shortest_paths ← compute_k_paths(k, src, dst)
    for i in 0, ..., k do
        p′ ← get_path(i, k_shortest_paths)
        s′ ← env.alloc_demand(s, p', src, dst, dem)
        k_q_values[i] ← compute_q_value(s', p')
    q_value ← epsilon_greedy(k_q_values, ϵ)
    a ← get_action(q_value, k_shortest_paths, s)
    r, Done, s′, src′, dst′, bw′ ← env.step(s, a)
    agt.rmb(s, src, dst, bw, a, r, s′, src′, dst′, bw′)
    reward ← reward + r
    If training_steps % M == 0: agt.replay()
    src ← src'; dst ← dst'; bw ← bw', s ← s'
```


Almasan, P., Suárez-Varela, J., Badia-Sampera, A., Rusek, K., Barlet-Ros, P., & Cabellos-Aparicio, A. (2019). Deep Reinforcement Learning meets Graph Neural Networks: An optical network routing use case. *arXiv preprint arXiv:1910.07421*.

What is the performance of DRL+GNN wrt. the state-of-the-art?

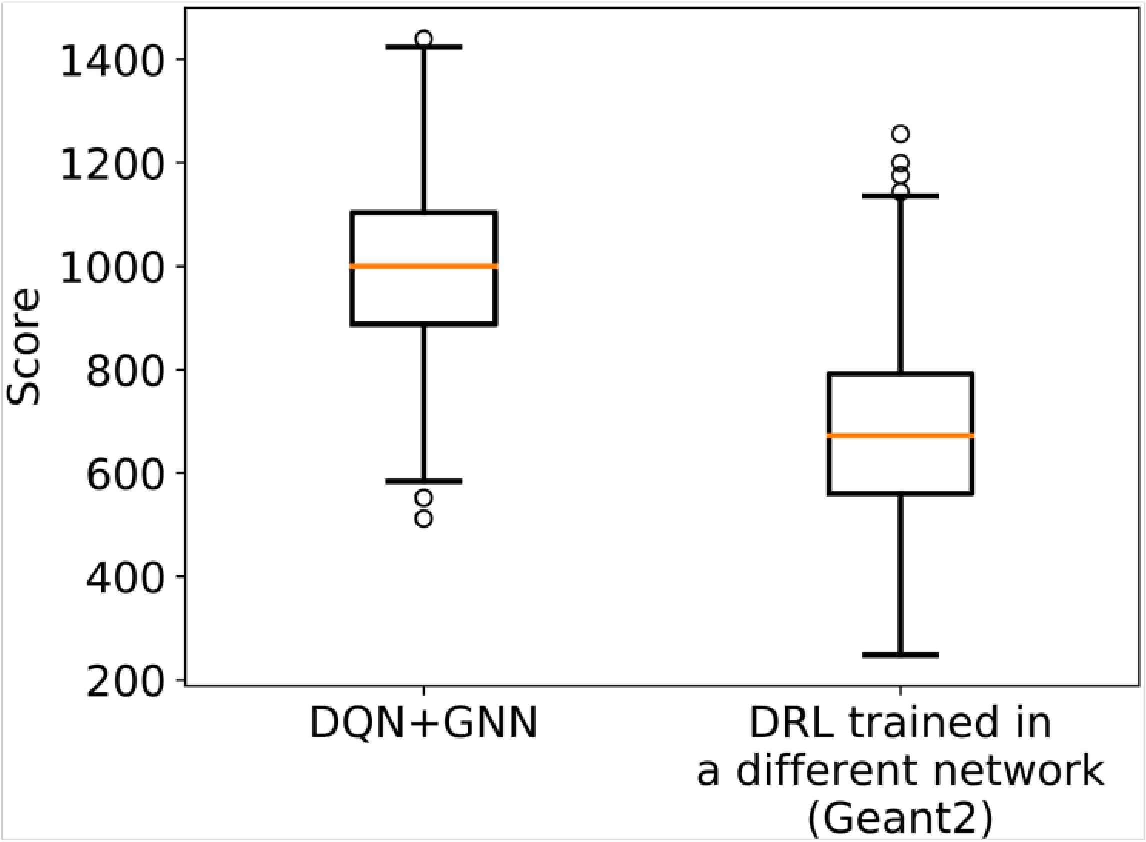
DRL+GNN vs. State-of-the-art



- We compare DRL+GNN against state-of-the-art DRL and heuristics
- DRL+GNN outperforms traditional heuristics
- **DRL+GNN achieves similar performance to the DRL state-of-the-art algorithms**

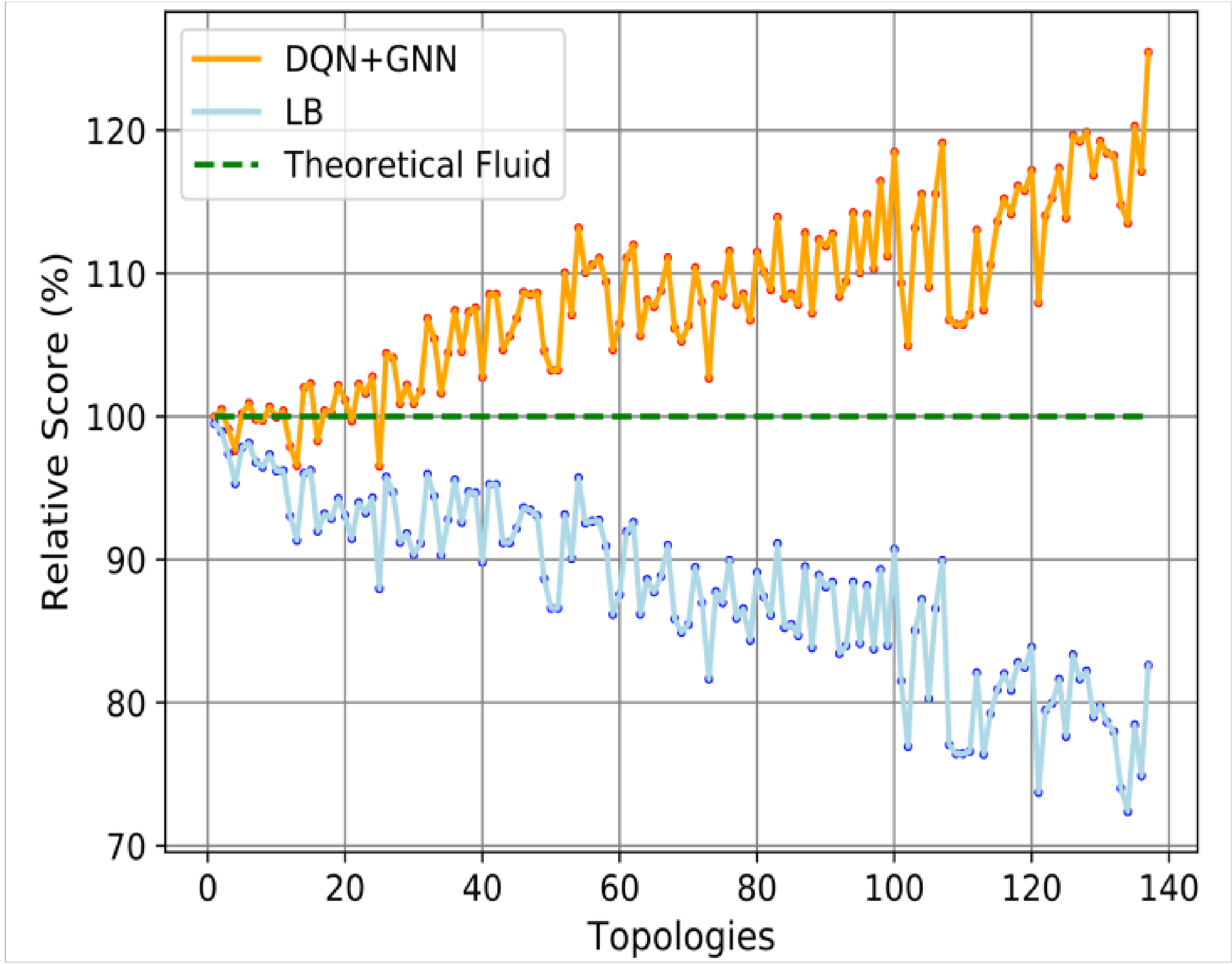
What happens when we apply DRL+GNN in unseen networks?

Generalization capabilities of DRL+GNN



- DRL+GNN is trained in one network and applied to a different unseen network
- **DRL+GNN keeps the same level of performance, even if has never seen this network during training**

Generalization capabilities of DRL+GNN



- We test DRL+GNN in 100+ unseen random network topologies
- **DRL+GNN achieves consistent performance in unseen networks**

Conclusions

Conclusions

- Graph Neural Network is an efficient tool to **accurately model computer networks**
- GNN-based models can **generalize to unseen** computer networks
- DRL+GNN looks as a promising technique to **build self-driving networks**

GNNs are as important to Computer Networks as CNNs to Computer Vision

Learn about GNN: All papers are free online

github.com/knowledgedefinednetworking/Papers/wiki

Play with GNN: Code and Datasets open-source

github.com/knowledgedefinednetworking/demo-routenet

knowledgedefinednetworking.org

Challenge yourself!

Participate in the GNN BNN 2020 Challenge

bnn.upc.edu/challenge2020

Join us - Barcelona Neural Networking

Prof. Albert Cabellos
Director

Prof. Pere Barlet
Scientific Director

Albert López
Head of Engineering

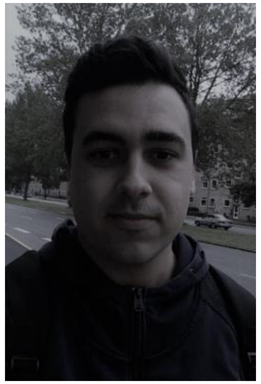
Jose Suárez
Post-Doc (Dec 2019)

Dr. Sergi Abadal
Post-Doc

Dr. Kryztof Rusek
Data Scientist

Paul Almasan
PhD Student

Maria Gkotsopoulou
MSc Student



Sergi Carol
MSc Student

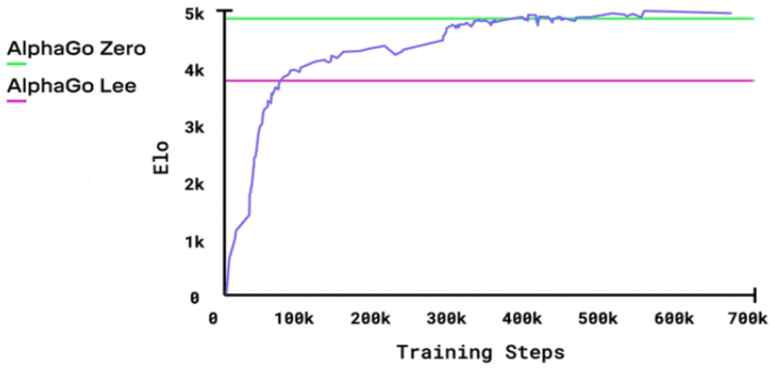
Albert Canyelles
BSc Student

Backup - AlphaZero

AlphaZero

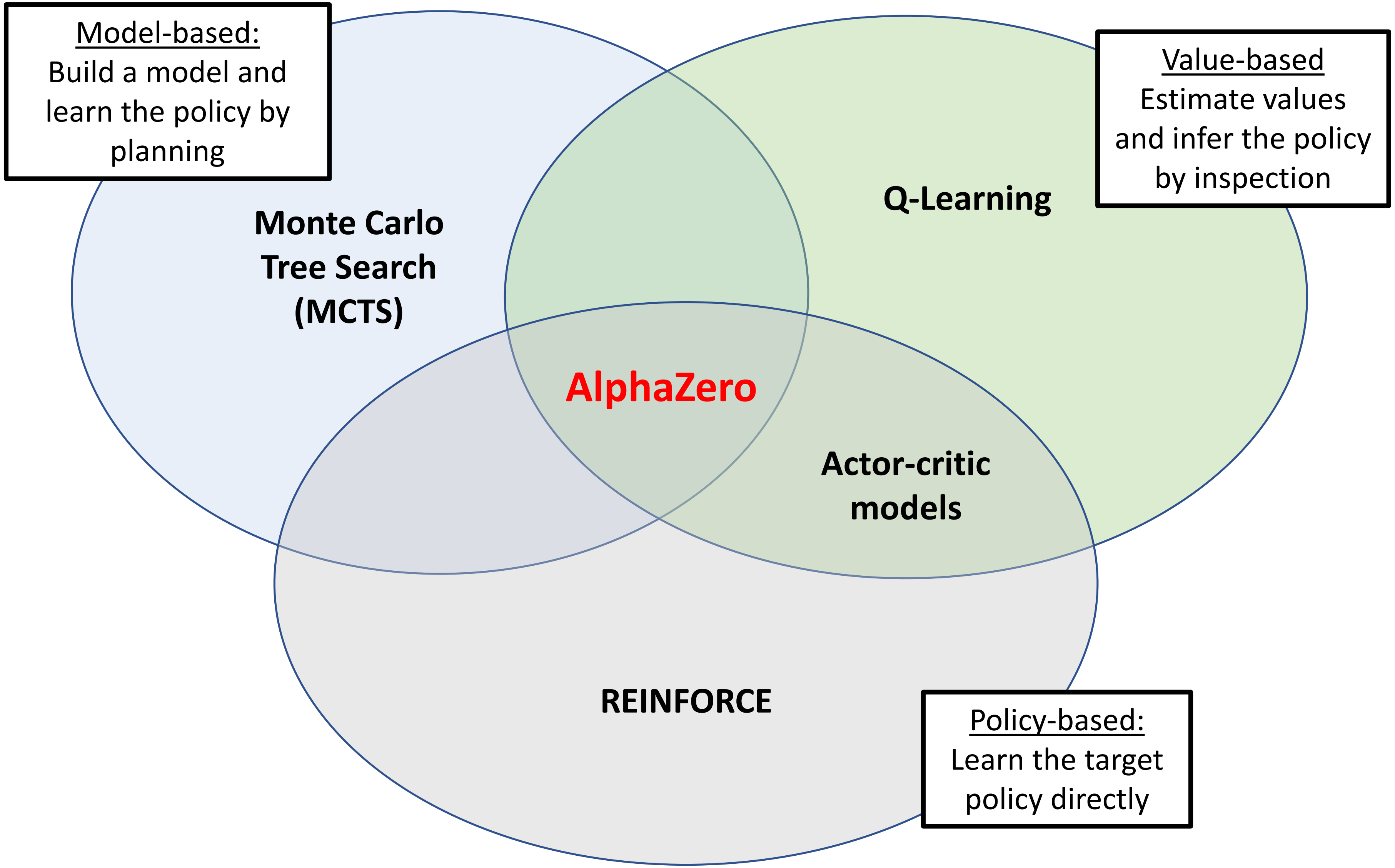
- In late 2017, Google DeepMind presented AlphaGo Zero
- Super-human performance in the Go board game
- AlphaZero is a generalization of AlphaGo Zero that **beated all the world-champion programs in the games of Chess** (after 4 hours of training)**, Shogi** (after 2 hours) **and Go**
- To learn each game, an untrained neural network (**tabula rasa**) plays millions of games against

  itself (**self-play**)



Source: https://deepmind.com/blog/

AlphaZero


Model-based:
Build a model and learn the policy by planning
Value-based
Estimate values and infer the policy by inspection
Monte Carlo Tree Search (MCTS)
Q-Learning
AlphaZero
Actor-critic models
REINFORCE
Policy-based:
Learn the target policy directly

AlphaZero

- Main features of AlphaZero:

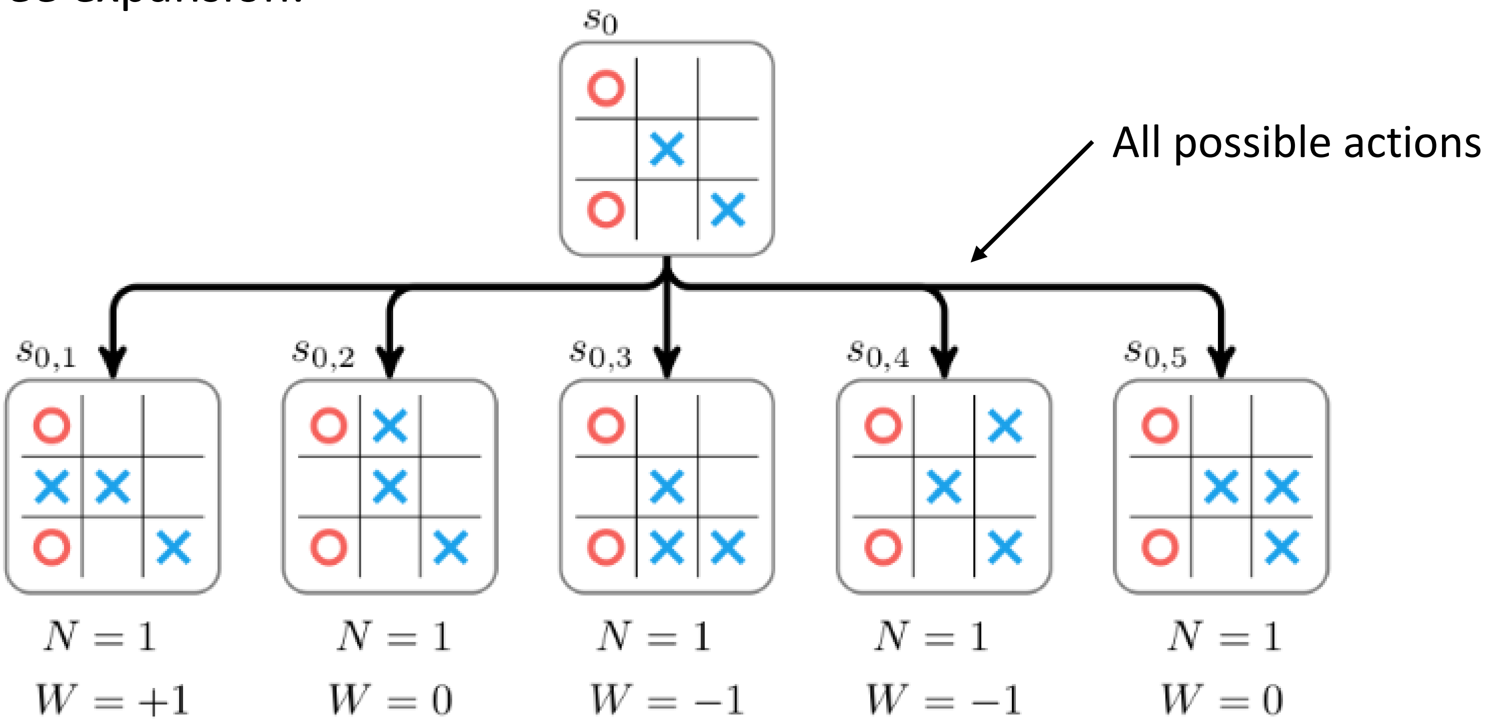
  - Monte Carlo Tree Search (MCTS) → Smart exploration

  - Combination of Deep Neural Network models:
    - Actor NN → Predicts the target policy
    - Critic NN → Predicts the value of states (potential reward at the end of the episode)

  - Self-play from tabula rasa → No human bias

Toy example: Tic-Tac-Toe game

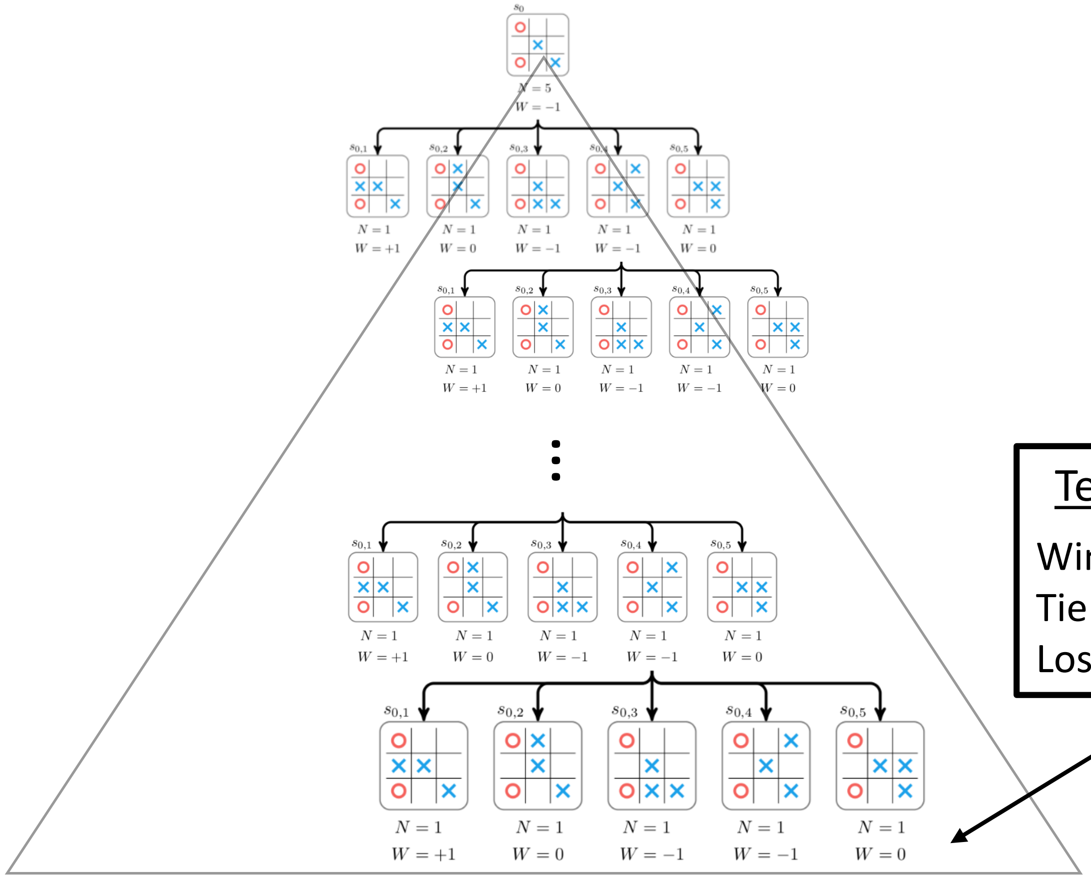
- Monte Carlo Tree Search (MCTS)
  - Tree expansion:



N: Node visit count
W: State value (potential reward)

Toy example: Tic-Tac-Toe game

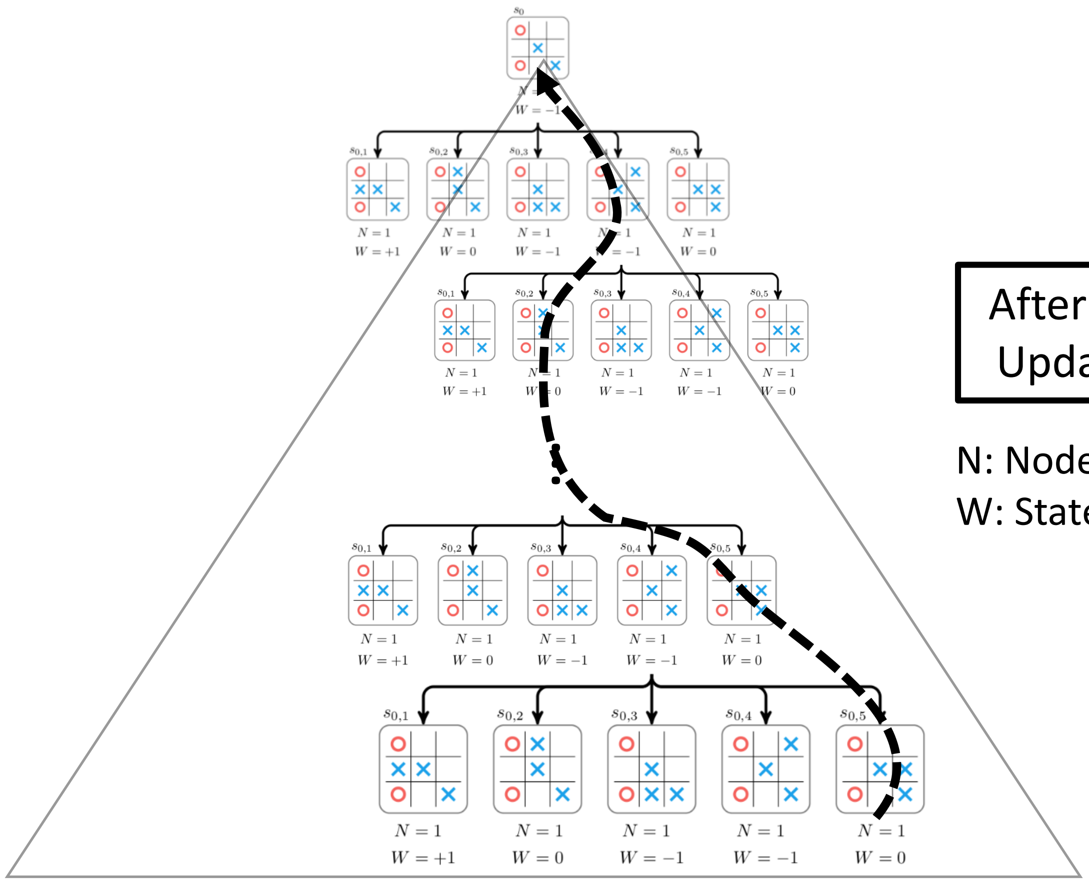
- Monte Carlo Tree Search (MCTS)
  - Exploration:



Terminal state

Win → W + 1
Tie → W + 0
Loss → W - 1

- Monte Carlo Tree Search (MCTS)
    - Backpropagation:

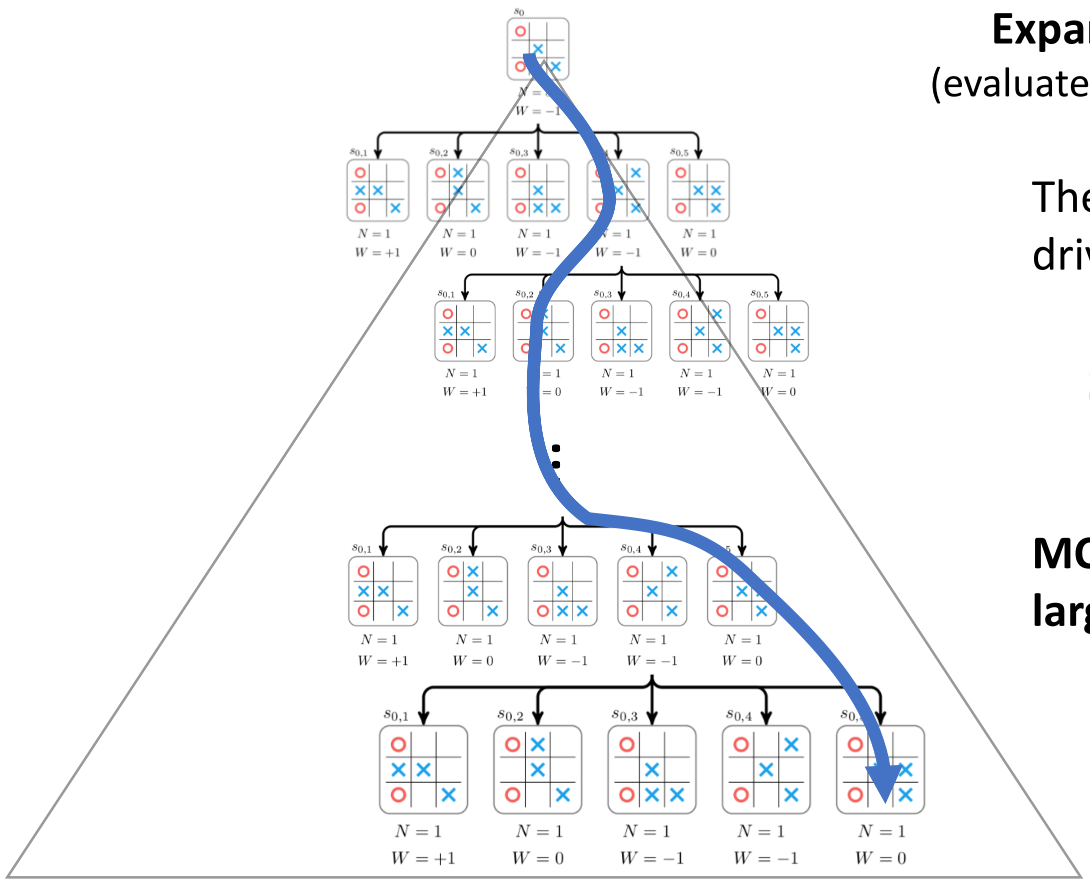


After N simulations:
Update ‘W’ and ‘N’

N: Node visit count

W: State value (potential reward)

- Monte Carlo Tree Search (MCTS):
  - Smart exploration:



Expand all the actions is not feasible!
(evaluate all combinations of states and actions)

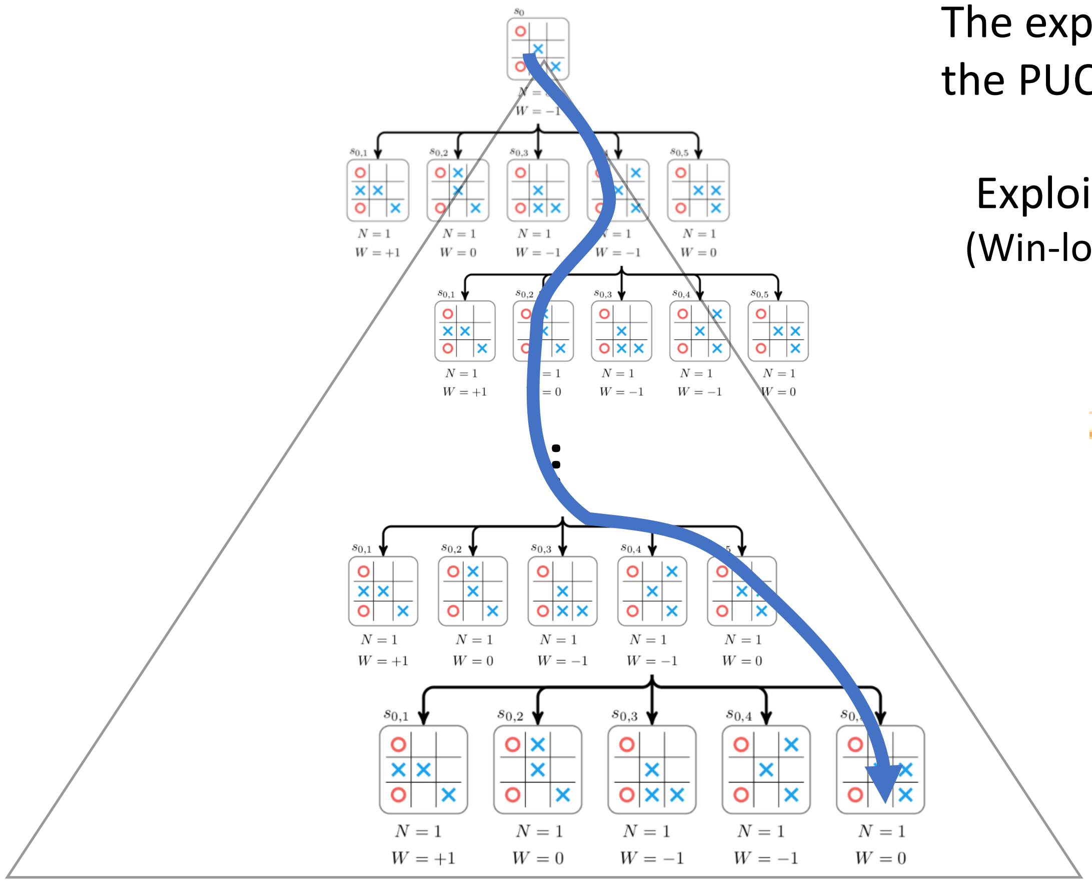
The exploration in AlphaZero is driven by the PUCT formula:

$$U_i = \frac{W_i}{N_i} + cP_i \sqrt{\frac{\ln N_p}{1 + N_i}}$$

MCTS explores the action with larger U_i score

PUCT (Polynomial Upper Confidence Bound applied to Trees)

- Monte Carlo Tree Search (MCTS):
  - Smart exploration:



The exploration in AlphaZero is driven by the PUCT formula:

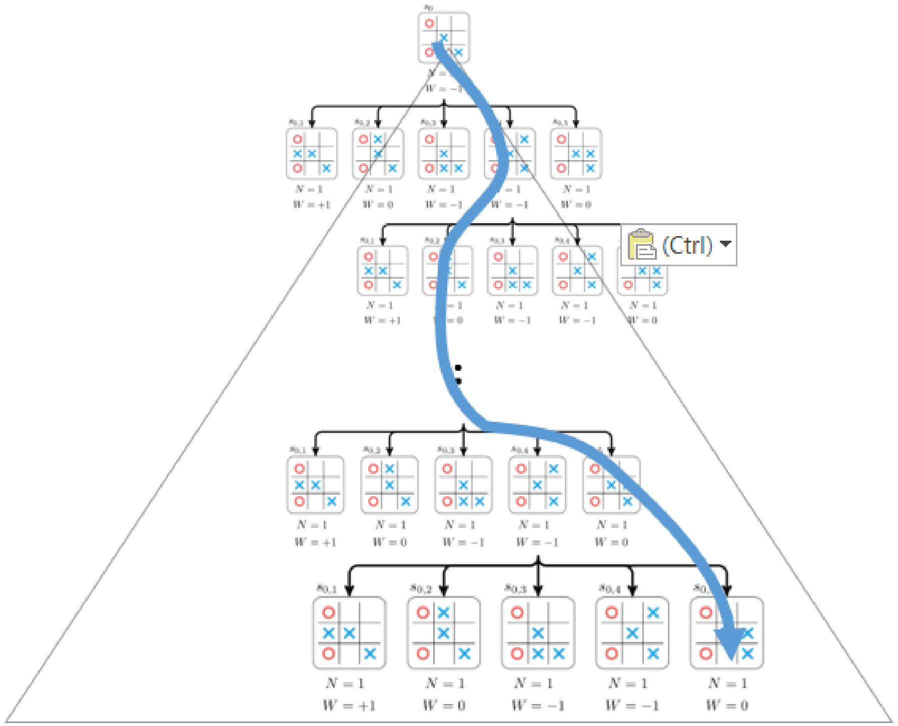
Exploitation (Win-loss rate) vs Exploration (Visit counts rate)

$$U_i = \boxed{\frac{W_i}{N_i}} + cP_i \boxed{\sqrt{\frac{\ln N_p}{1 + N_i}}}$$

trade-off between Exploration and exploitation

W: Total value of the node
N_i: Visits node
N_P: Visits parent node

- Monte Carlo Tree Search (MCTS):
  - Smart exploration with actor-critic NN models:



The actor and critic NN are used to direct the exploration

$$U_i = \boxed{\frac{W_i}{N_i}} + c\boxed{P_i}\sqrt{\frac{\ln N_p}{1+N_i}}$$

Critic NN

$\widehat{W}(s) = v$

Actor NN

$P_i = \pi(a_i | s_t)$