

Comments on "Tableless Routing"*

David Meyer

dmm@1-4-5.net

Last update: October 19, 2018

1 Introduction

The document provides a review of the "Tableless Routing" deck that was discussed on 09/19/2018.

The approach taken in this document will be to review each slide and provide comments where useful. These comments will be organized into three categories:

- Feedback the technical aspects of each slide. I will include the slide where doing so would provide additional information
- Feedback on where we might find colleagues (industry, academia, or other)
- Comparison to competitive approaches
- Marketing strategies

I will analyze the deck slide by slide, providing comments where appropriate. Here page number is equivalent to slide number. I will also include a section on Nits (editorial and non-technical comments) where appropriate.

1.1 Introductory Comments on Graph Neural Networks for the Routing

While I found the Graph Neural Networks and their use in routing [1, 2] interesting, I also I found a few aspects of proposed protocol troublesome. First, nodes (routers) "cold-start" from some fixed-point state of the GNN. That seems relatively innocuous. However, nodes then update their routing state using the routing state of their neighbors. This seems very distance vector-like, and so I would expect similar problems (e.g., counting to infinity, split horizon, etc).

*This report is a deliverable for Project Number HE2017070001.

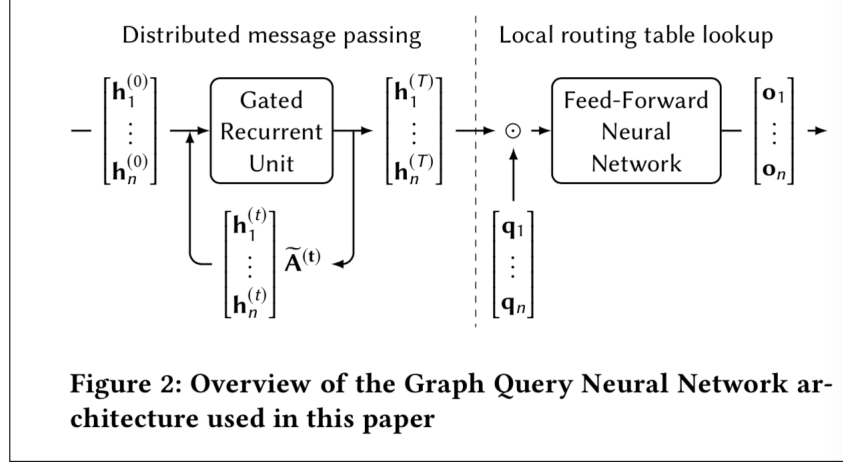


Figure 1: Figure 2 from [2]

Second, the authors use dropout [3] to regularize the training of the GNN. The idea here is that dropping out random links¹ during each training epoch (or whatever) effectively models packet loss, link failure, and router failure (or any other Byzantine failure, really). I'm skeptical that this approach can actually handle all of these failure modes.

Third, the approach is to simulate a routing protocol on the graph (network) to generate the $[\mathbf{h}_1^{(T)}, \dots, \mathbf{h}_n^{(T)}]^T \odot [\mathbf{q}_1, \dots, \mathbf{q}_n]^T \rightarrow$ output port mapping that is subsequently used for supervised training of the Feed-Forward Neural Network (FFNN) in Figure 1. So you need a routing protocol to find the output ports in the first place.

The three aspects of the routing solution mentioned above make the entire approach less compelling. The question I would expect would relate to why this approach is better than just using OSPF [4] (or whatever), since I already have to run such a protocol in simulation to build the dataset for supervised training.

Finally, I would expect that "traditional" routing protocol engineers (IS-IS, BGP, etc) will argue against such an approach. The obvious arguments are that (i). the approach above is far more complex (and perhaps less computationally efficient) than existing shortest-path routing algorithms and (ii). given the probabilistic nature of machine learning algorithms, existing protocols can be shown to be deterministically correct, while the machine learning approach can provide only probabilistic bounds.

¹The graph is represented as an adjacency matrix.

1.2 Comments on Routing Table Size as a Problem to Solve

A few "historical" comments here. First, are *routing table size* and *IP lookup time* still compelling problems? Current day route processors (RPs) have plenty of DRAM, unlike the late 1990s/early 2000s, where routing table size and growth were critical concerns. So it is not clear that inter-domain routing table size is currently a problem. In addition, during our discussion of this deck (on 09/19/2018) it was stated that this scheme was designed for intra-domain routing, where table size really hasn't been a problem ².

In addition, it is not clear how "table-oriented" functionality would be achieved. Such functionality includes various *Virtual Private Network*, or VPN technologies ³.

Another overview comment relates to *IP lookup time*. It is not clear that this is a current problem either. The example I gave was MPLS (former Tag Switching). In this case one of the original objectives was to reduce IP lookup time from something like $O(n \log n)$ for trie-based lookup (depending on the exact shape of the trie) to $O(1)$ for a 32-bit label lookup. It was learned, however, that since the IP lookup path was already so highly optimized that fixed length lookup really didn't speed up the overall lookup time.

Finally, I'm wondering if we can use the tableless approach in *inter-domain* routing to infer bogus or hijacked prefixes. If we could do this accurately it might provide a lightweight alternative (or augmentation) for the RPKI [5]. More work would be required here, and the IETF would be a reasonable place to do this work.

This remainder document is organized as follows: Each of the following sections refers to a slide on which I have comments. Finally, Section 8 provides a few conclusions.

2 Slide 2

2.1 Technical Feedback

Slide 2 outlines the motivation for tableless routing:

- Delay: Lower lookup latency
- Chip Area Cost: Lower (on-chip) memory requirements
- Lookup bandwidth: This one isn't clear but it seems that if you don't have on-chip tables you don't need to access them.

²On the other hand, IGP adjacency state has in the past been as scaling problem.

³For example, how would Virtual Routing and Forwarding (VRF) or multicast be supported?

As mentioned above, it is not clear that lowering delay or table space is compelling. However, repurposing on-chip area that would have been used for on-chip tables *could be* motivating. More research on this point is required.

More specifically, items 2 (Area cost) and 3 (Bandwidth) overlap significantly. From my perspective Area cost is clear. The Bandwidth item (3) needs further refinement. It seems that the idea is that if you don't have tables that you need to lock or synchronize then various pipelines, both on-chip and between chips, will be simpler. While slides 3-4 address this point to some extent, the difference is not crisp. Hence we need to clarify the difference between items 2 and 3.

2.2 Possible Collaborators

Large mobile operators would be have the best information here. This is largely due to their use of IPv6 and the continuing deaggregation makes them idea candidates for feedback on tableless routing.

2.3 Marketing Strategies

As mentioned above, more research in both the technology space and into the economics of the cost/benefit tradeoffs is required before any marketing would be appropriate.

2.4 Nits

N/A

3 Slide 5

Slide 5 introduces the Graph Neural Network (GNN) architecture [1, 2] for tableless routing. The basic idea is to use the Gated Recurrent Unit (GRU) [6] (a Recurrent Neural Network (RNN) variant) to learn a feature vector $[\mathbf{h}_1^{(T)}, \dots, \mathbf{h}_n^{(T)}]^T$. This is the terminal internal state of the GRU at convergence. That is, the network is trained with the iterative Almeida-Pineda algorithm [7] which works by running the propagation of the hidden representation to convergence, and then computing gradients based upon the converged solution. This is depicted in Figures 1 and 2. Each iteration of the Almeida-Pineda algorithm is called a "communication iteration" on Slide 5. Note that as mentioned above, a key idea here is that training with *dropout* [3] actually "ensembles" all of the possible networks and hence handles all possible link (and other) failures ⁴.

⁴See Slide 7; note that this is point is neither described nor proved.

Tableless Routing—GNN framework

- GNN (Graph Neural Network) framework

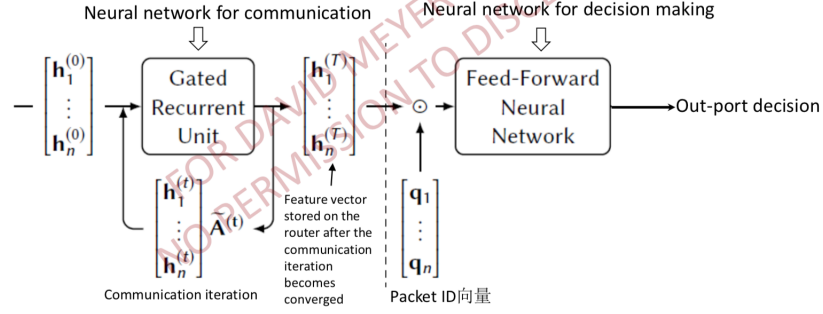


Figure 2: Slide 5

This feature vector is then combined with a packet ID vector $[\mathbf{q}_1, \dots, \mathbf{q}_n]^T$ by *element-wise* multiplication (indicated by the \odot symbol) to form the input to a Feed Forward Neural Network (FFNN) which maps its input to output port. That is

$$[\mathbf{h}_1^{(T)}, \dots, \mathbf{h}_n^{(T)}]^T \odot [\mathbf{q}_1, \dots, \mathbf{q}_n]^T \rightarrow \text{output port}$$

3.1 Technical Feedback

As mentioned above, there are a few technical issues here. See Section 1.1 and 1.2. Compelling answers to the questions there are needed.

There is also an apparent bug in the feedback loop depicted in Figure 2 of [2]. In particular, assuming

$$\begin{bmatrix} \mathbf{h}_1^{(t)} \\ \vdots \\ \mathbf{h}_n^{(t)} \end{bmatrix} \tilde{\mathbf{A}}^{(t)}$$

is an inner-product ⁵, then the 'shapes' are wrong. That is, $[\mathbf{h}_1^{(t)}, \dots, \mathbf{h}_n^{(t)}]^T$ is of dimension $n \times 1$ while $\tilde{\mathbf{A}}^{(t)}$ is of dimension $n \times n$. For the output to be of dimension $n \times 1$ (input to the GRU), we would need

$$\tilde{\mathbf{A}}^{(t)} \begin{bmatrix} \mathbf{h}_1^{(t)} \\ \vdots \\ \mathbf{h}_n^{(t)} \end{bmatrix}$$

That is, $\langle n \times n, n \times 1 \rangle \rightarrow n \times 1$. Slide 5 (Figure 2) retains this apparent error.

3.2 Possible Collaborators

As I mentioned I find this approach very interesting and the possibility for other uses of this technology seem to be many. For example, I mentioned the possibility of a lightweight alternative to the RPKI based on this technology might be possible. However, this idea would have to be fleshed out and tested.

3.3 Marketing Strategies

Because of my comments in Section 3.2 it would seem premature to engage in marketing at this time.

3.4 Nits

N/A

4 Slide 6

Slide 6 introduces an algorithm for dynamically updating a nodes feature vector as well as some important properties of the GNN.

4.1 Technical Feedback

Sub-bullet 2 appears to be incorrect. Instead of

$$X_i = \sum f_i(X_j | j \in \text{Neighbors}(X_i))$$

it seems it should be

$$X_i = \sum_{j \in \text{Neighbors}(X_i)} f(X_j)$$

⁵The inner-product of \mathbf{u} and \mathbf{v} is typically denoted with angle brackets: $\langle \mathbf{u}, \mathbf{v} \rangle$.

Tableless Routing—GNN

- GNN Features:

- 1. Each node (i.e., router) stores a feature vector
- 2. Each node updates its feature vector based on the feature vectors of its neighbor nodes:

$X_i = \sum f_i(X_j | j \in \text{Neighbour}(X_i))$, f_i function is implemented by a neural network (i.e., communication neural network)

- 3. Fixed-Point theory of GNN: with finite iteration steps, the feature vector can converge to a stable value (i.e., fixed point)
- 4. The neural networks can be first trained in the controller offline and then deployed on the routers

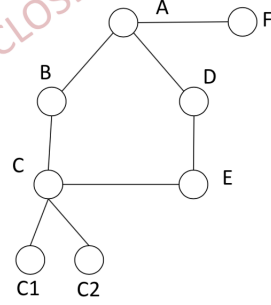


Figure 3: Slide 6

where f the function approximated by the GNN. Note that since the GNN shares parameters across time, it seems that what you have as $f_i(\cdot)$ should be $f(\cdot)$. More specifically, according to [2], we have

$$\mathbf{h}_v^{(t)} = f\left(\left\{\mathbf{h}_{\text{Nbr}(v)}^{(t-1)}\right\}\right) = \sum_{u \in \text{Nbr}(v)} f^*\left(\mathbf{h}_u^{(t-1)}\right)$$

where $f^*(\cdot)$ is a feed-forward neural network.

Sub-bullet 3 talks about "Fixed-Point theory of GNN" and makes claims about the termination properties of the algorithm, notably that it converges in a finite number of steps. However, there is no discussion of what these bounds look like or citations to the original (or other) work on this topic.

4.2 Possible Collaborators

Both network hardware designers, machine learning people and network operators would be useful collaborators here. However, as I mentioned above (Section 1.1 and 1.2), there are many issues that need to be resolved/supported.

Tableless Routing—GNN

- Advantage of GNN
 - 1. Flexible optimization objectives
 - Shortest path
 - QoS optimization (delay), Load balancing
 - 2. Highly-robust training: leading to a faster convergence
 - Use the dropout training method to simulate random link failures in the topology. Enable a faster convergence speed when link fails.
 - Use the incremental training method to deal with the case of adding new nodes into the network. Enable a fast convergence speed when network scales up.

HUAWEI TECHNOLOGIES CO., LTD.

Huawei Confidential

7


 HUAWEI

Figure 4: Slide 7

4.3 Marketing Strategies

Again, in my opinion it is too early to start marketing activities.

4.4 Nits

N/A

5 Slide 7

Slide 7 (Figure 5) discusses the advantages of using a GNN to solve the routing problem.

5.1 Technical Feedback

Bullet 1 on slide 7 discusses the flexibility of the method, mentioning that the method can do both shortest path (e.g., OSPF [4]) and/or load balancing. Again, since you have to use OSPF (RIP [8], BGP [9], or others) to find the labels (output ports) to train the FFNN, a question one might ask is why all of the complexity of the GNN approach is really needed.

Bullet 2 discusses the use of dropout to simulate link and other failures, and the incremental

Tableless Routing—GNN

- Advantage of GNN
 - 1. Flexible optimization objectives
 - Shortest path
 - QoS optimization (delay), Load balancing
 - 2. Highly-robust training: leading to a faster convergence
 - Use the dropout training method to simulate random link failures in the topology. Enable a faster convergence speed when link fails.
 - Use the incremental training method to deal with the case of adding new nodes into the network. Enable a fast convergence speed when network scales up.

HUAWEI TECHNOLOGIES CO., LTD.

Huawei Confidential

7



Figure 5: Slide 7

$$\forall v \in \{\mathcal{V}_1 \cap \mathcal{V}_2\}, \quad \mathbf{h}_{v, \mathcal{G}_1}^{(t=T)} = \mathbf{h}_{v, \mathcal{G}_2}^{(t=0)} \quad (16)$$

Figure 6: Graph Update Procedure

training method for adding/subtracting nodes. There seems to be an error in [2] regarding incremental updates. In particular, Equation 16 in [2], shown in Figure 6, appears to be incorrect (it is backwards). That is, it should say

$$\mathbf{h}_{v, \mathcal{G}_2}^{(t=0)} := \mathbf{h}_{v, \mathcal{G}_1}^{(t=T)} \quad (1)$$

Note that I used $:=$ rather than $=$ in Equation 1 because it is really an assignment of the initial state for the iterative process on \mathcal{G}_2 .

Finally, what isn't discussed here is the complexity of the update procedure, and where it would be executed.

5.2 Possible Collaborators

N/A

5.3 Marketing Strategies

N/A

5.4 Nits

N/A

6 Slide 8

Slide 8 reviews some of the key concepts of GNN routing and forwarding, focusing on "Neural network forwarding" (bullet 4). My first question was about the details of the current (recent) plan, namely, using a "two-layer BNN (only XOR&ADD required), supported by existing hardware (P4), with some precision loss". More detail on how and why this works would be useful. Slide 9 shows how this might be implemented in P4.

6.1 Technical Feedback

The most obvious question one might have here is whether the advantages of tableless routing (see Figure 5) outweigh the additional cost of on-board computational resources such as are proposed in the medium and longer term plans.

6.2 Possible Collaborators

As I've mentioned on other slides, the obvious collaborators would be service providers, machine learning academics, and other network engineers.

6.3 Marketing Strategies

N/A

6.4 Nits

Slide 9 is a busy slide that tries to fill in some of the details.

7 Slide 10

Slide 10 introduces the reliability challenges with the GNN routing approach.

7.1 Technical Feedback

The key issue here that while the approach has error rates of less than 5% so that 95% of node pairs learn the shortest path between them, the algorithm does not provide guarantees for the remaining 5% of pairs. This means, roughly, that each node has a probability $p < 0.05$ chance that a packet will be misrouted⁶. As pointed out, the error rate is due to the probabilistic nature of neural networks.

One of the innovations in this work is to borrow ideas from reachability assurance used in an IoT setting. In particular, a *Reliable Tableless Forwarding* (RTF) method with a 100% reachability guarantee is proposed.

7.1.1 Aside: *Route* vs. *Reachability*

The slide states that RTF provides a 100% reachability guarantee. Actually, routing doesn't provide reachability guarantees. For example, even if the route points to a correct interface at some time t , existence of the route *does not* guarantee reachability. That is, even if we know the correct next hop toward the destination, any number of events can prevent a packet from getting to its destination; examples include ingress filtering, in which case choosing the wrong source address⁷ can cause a packet which is sent on the correct interface towards the destination to be dropped, and many others.

Slide 11 overviews a Link Reversal Routing (LRR) algorithm that converts a Directed Acyclic Graph (DAG) into a Destination-oriented DAG (DDAG). The interesting point here is that the algorithm can convert a DAG into a DDAG in a finite number of steps; however, the complexity of the algorithm isn't outlined. Slide 12 shows an implementation of the LRR algorithm.

7.2 Possible Collaborators

N/A

7.3 Marketing Strategies

N/A

7.4 Nits

N/A

⁶This misrouting can apparently result in routing loops, blackholes, and the like.

⁷In the case of multi-homing.

8 Summary & Conclusions

The key features of Tableless Routing are summarized as follows (Slide 13):

- Minimized table storage requirements
- No computation-based forwarding
- Flexible optimization objectives, e.g.
 - Shortest-path routing
 - QoS routing of various forms
- 100% reachability guarantee for any number of link/node failures or neural network errors (if the network is still connected after link/node failures)
- Key technologies: GNN and Tableless LRR

8.0.1 Final Comments

First, regarding routing table storage, lookup latency and algorithm complexity: as I mentioned in the introduction, is the overall routing table size problem compelling these days? This is discussed in Section 1.2. Since the complexity of the solution appears to be significantly greater than say, computing shortest paths with Dijkstra’s algorithm [10], the significant additional complexity that Tableless LRR incurs needs stronger motivation.

I also mentioned a few problems with the idea of using Graph Neural Networks for routing. First, the way nodes update their routing state is reminiscent of distance-vector routing algorithms. As such one might expect classic distance-vector problems such as counting to infinity. Next, the authors use dropout [3] to regularize the training of the GNN. The idea here is that dropping out random links during each training cycle, effectively models link failure. But does this approach effectively model link failure, and how would you show that this is true? Finally, the generation of the data labelled with output port (required to train the FFNN) requires the simulation of *an existing routing protocol* on the graph (network) So you need a routing protocol to find the output ports in the first place.

There is an issue of *route vs. reachability* as discussed in Section 7.1.1. The key point here is that the existence of a route to a destination (really the next-hop towards the destination) doesn’t tell you that the destination is *reachable*, so saying that there is a “100% reachability guarantee” isn’t really accurate.

Another important observation here is that while the GNN formalism is an interesting approach to learning inference on a graph, it doesn't learn control ⁸. For example, Equation 4 of [2]:

$$\mathbf{h}_v^{(t)} = f\left(\left\{\mathbf{h}_{\text{Nbr}(v)}^{(t-1)}\right\}\right) = \sum_{u \in \text{Nbr}(v)} f^*\left(\mathbf{h}_u^{(t-1)}\right)$$

shows that the model doesn't really learn control; rather, the hidden state $\mathbf{h}_v^{(t)}$ is hard-coded to be a function of $\text{Nbr}(v)$, the neighbors of node (router) v . In Reinforcement Learning (RL) parlance, the GNN model does not learn a *policy* π such that $\pi : S \rightarrow A$, where S is the set of states and A are the possible actions in state S . As such I would expect that the main competitors for routing approaches based on GNNs would be RL algorithms designed to learn optimized control. See, for example, [11].

Finally, there is another problem with any data-driven approach that is particularly acute for routing problems: you don't see what happens to the data in the long term. More specifically, what looks good to your metrics right now might kill the product in a few years. As a result, continual retraining ⁹, perhaps along the lines of [12], will be required. How exactly to do this and what data to use for this is still open problems.

⁸While one might argue that the feedback in an RNN is a form of control, an RNN doesn't learn a *policy* function $\pi : S \rightarrow A$.

⁹Such retraining will need to be incremental and capable of dealing with non-stationary distributions.

References

- [1] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, “The graph neural network model,” *Trans. Neur. Netw.*, vol. 20, pp. 61–80, Jan. 2009.
- [2] F. Geyer and G. Carle, “Learning and generating distributed routing protocols using graph-based deep learning,” in *Proceedings of the 2018 Workshop on Big Data Analytics and Machine Learning for Data Communication Networks, Big-DAMA@SIGCOMM 2018, Budapest, Hungary, August 20, 2018*, pp. 40–45, 2018.
- [3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jan. 2014.
- [4] J. Moy, “OSPF Version 2.” RFC 2328, Apr. 1998.
- [5] R. Bush and R. Austein, “The Resource Public Key Infrastructure (RPKI) to Router Protocol.” RFC 6810, Jan. 2013.
- [6] R. Dey and F. M. Salem, “Gate-Variants of Gated Recurrent Unit (GRU) Neural Networks,” *ArXiv e-prints*, Jan. 2017.
- [7] F. J. Pineda, “Generalization of back-propagation to recurrent neural networks,” *Phys. Rev. Lett.*, vol. 59, pp. 2229–2232, Nov 1987.
- [8] G. S. Malkin, “RIP Version 2.” RFC 2453, Nov. 1998.
- [9] Y. Rekhter, S. Hares, and T. Li, “A Border Gateway Protocol 4 (BGP-4).” RFC 4271, Jan. 2006.
- [10] R. K. Ahuja, K. Mehlhorn, J. Orlin, and R. E. Tarjan, “Faster algorithms for the shortest path problem,” *J. ACM*, vol. 37, pp. 213–223, Apr. 1990.
- [11] A. Valadarsky, M. Schapira, D. Shahaf, and A. Tamar, “A Machine Learning Approach to Routing,” *ArXiv e-prints*, Aug. 2017.
- [12] R. Istrate, A. Cristiano Innocenza Malossi, C. Bekas, and D. Nikolopoulos, “Incremental Training of Deep Convolutional Neural Networks,” *ArXiv e-prints*, Mar. 2018.