Machine learning in packet routing process using Quagga/Zebra routing SW suite

Aleksandra Jereczek, Patrycja Kochmańska, Maciej Paczkowski

Intel Technology Poland
Gdansk, Poland
aleksandra.jereczek@intel.com, patrycja.kochmanska@intel.com, maciej.paczkowski@intel.com

Abstract

Today's networks still grow in size and scale, but also a number of their applications increases. All of these factors make requirements of effective routing mechanisms more restrictive. Many attempts were done to improve existing routing protocols. Some of them try to introduce Artificial Intelligence algorithms to the routing process.

In this paper we will show our conception of using Neural Networks together with existing link-state routing protocols' mechanisms. We will shortly introduce current state and tendencies of networks development, and past approaches to routing improvements. Basing on conclusions from this analysis we will describe our idea of how to use Artificial Neural Networks in routing process (especially with OSPF routing protocol).

Keywords

machine learning, OSPF, packet forwarding, quagga, routing

Introduction

Over the recent years we could have observed continuous and dynamic development of computer networks. We may notice that they have transformed from simple local area networks (LAN) to widely accessible public broad networks. What is less visible but not less important, networks also came into use in service providers labs, datacenters, and computational centers, where a huge number of devices by themselves form subnetwork acting as an endpoint.

Very important aspect of each network operation is a routing engine implemented in this network. Routing is a process of selecting paths for packets when they are forwarded across or between networks. It is one of the main tasks of routers to establish and maintain the best routes between particular network nodes and endpoints. As a result, they create a routing tables that are later used to determine on which interface packets with specific destination should be forwarded when crossing the network.

Routing process is quite easy when we consider networks consisted of limited number of devices and with static topology. Situation becomes more complicated when we have many devices and topology changes are frequent. When many links or network nodes often change their state, the routing protocols that are widely used nowadays, still may have some problems with fast convergence and effective path recalculations. Through the time when correct routing table is not determined, there may occur many packet losses, what evidently affects the network performance in a negative way.

There are two main classes of routing protocols implemented in networks:

- Distance-Vector routing protocols, in which routers have no information about the whole network topology and the decision about the best routes are based only on the data about costs, gained from router's nearest neighbors, and
- Link-State routing protocols, which assume that each network node creates and stores its own scheme of whole network topology and independently calculates the least-cost paths from itself to every other node.

The examples of Link-State routing protocols include Open Shortest Path First (OSPF) and Intermediate System to Intermediate System (IS-IS) protocols. The basic concept of this type of routing is that each node constructs its own map of the network that may be perceived as a graph. This graph is used to calculate routes to each other node using shortest path algorithm. In the mentioned big and dynamic environments, the time of the graph creation and paths recalculation may have significant impact on routing effectiveness, but it is also a promising field of improvements leading to better network performance.

Drawbacks of existing solutions

Our research was mainly focused on OSPF protocol, but the conclusions are adequate also to other link-state protocols. As in all the routing protocols, the goal of running OSPF is to determine the best paths between all network nodes. The current implementation is based only on Dijkstra's algorithm (also known as Shortest Path First - SPF algorithm) which operates on the data stored in Link State Data Base (LSDB) to calculate Shortest-Path Tree (SPT). The tree represents the shortest paths to each destination in given routing area within routing domain. OSPF uses hierarchical network structure and while many areas are in use, the area 0 mediates all connections between other areas (in a hub and spokes manner). LSDB is populated according to Link State Advertisement (LSA) packets, send by the nodes when a change occurs in the network. When the database contains full information about the network state, OSPF is 100% accurate. Problem

occurs when the changes are frequent and necessary recalculations are not finished before the next LSDB update.

Changes in the topological database trigger partial or full routing table recalculation with SPF algorithm. There are scenarios when the entire SPT doesn't need to be recomputed because most of previously calculated tree remains unchanged. It is intuitively known, that recomputing only a part of the tree rather than the entire tree results in faster convergence and saves CPU resources. The approach in such situations is to use *incremental SPF* which allows the system to recompute only the affected part of the tree. [1] However, even with this improvement computational cost is not predictable as it depends on the kind of LSAs received after the network state change. If the change triggers Type-1 or Type-2 LSA, the full SPT recalculation is performed.

Each transit link (between two routers with no end users attached to it) that fails is connected to at least two routers (or more in case of multiaccess technologies like Ethernet). This results in two (or more) routers always forced to run the expensive full SPF (that scales with N^2, where N is the number of network nodes), with negative impact on the routing/forwarding efficiency of the overall network.

As explained above, the approach to decrease the computational cost of SPF can still be not fully efficient from a network perspective. In case of full routing table recalculation, SPF requires some time to generate the new Routing Information Base (RIB). This is a base that is directly connected with routing table according to which forwarding decisions are made. Due to the mentioned delay, the Forwarding Information Base (FIB), which is finally used to choose output interface to direct a packet to its destination, is not updated at the same time LSAs come into the database, and this creates network outages.

The scheme of OSPF operations is shown in Figure 1. Here we have a block diagram of Unix system-based IP routing software suite. As an example of such suite we can consider FRRouting or Quagga, that manages various IPv4

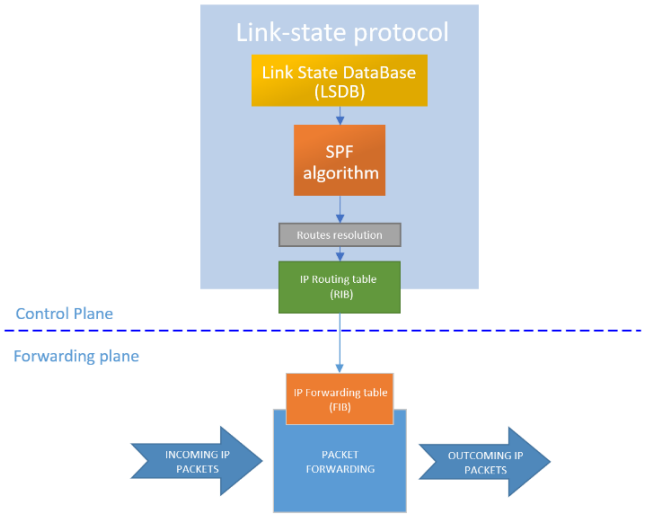

Figure 1. Unix system-based IP routing software suite

and IPv6 routing protocols. Individual blocks on the diagram are divided into two areas: control plane and forwarding plane. These are also two parts of the router architecture. Control plane participates in routing operations while forwarding plane is responsible for sending packets to a proper network interface.

The control plane operations are served in a user space of the operating system (OS) and forwarding operations are executed in a kernel space. It is rather difficult to tamper inside the kernel space, but the user space gives a big field to modify the behavior of all the routing process as long as we can put the output of control plane operations to the kernel space in an unchanged format, and this is the field we decided to use in our invention, what is described below.

Current networks tendencies

As far as 10 years ago, M. Goyal et al. had conducted a survey and proved that existing for over twenty years OSPF routing protocol was still widely deployed. [2] Nowadays, this conclusion remains valid. OSPF's original design focused on scalability and robustness against failures. At the very beginning network recovery time having reached few tens of seconds was considered sufficiently fast, so the protocol was not comprehensively optimized in this field. To achieve high scalability and protection against failures (which were, taking into account capabilities of networks of that times, more important factors) routing domains were divided into multiple areas and processing cost was limited for better CPUs utilization.

These requirements have changed when we started to use more powerful and computationally efficient devices and currently such long inoperable network state would cause unacceptable traffic loss level. Modern networks are designed to serve a large number of packets and network traffic has completely different requirements. We have new, more restrictive definition of fast delivery and packets often have to be delivered in a certain order, so losing some of them may cause dramatic quality increase.

Nowadays, networks not only consist of simple endpoints and devices that connect them together, but also there are more and more data centers connected to the widely accessible Internet as well as used by private entities. The data center by itself may be considered as an intricate network. It is a complex pool of resources forming a cluster and using the network to host applications and store data. It can be constituted of hundreds of servers, thousands of physical interfaces, and even more virtual machines (VM) and interfaces. Partially or fully virtual environment increases the risk of failures because one crash of one physical server may cause a break-down of some bigger part of the network.

Many different experiments were done in the field of providing enough network bandwidth and creating effective data center architecture. Different Data Center Network (DCN) architectures address the issues of network scalability and oversubscriptions. As opposed to

the server part of many data centers, where low-cost commodity equipment is widely introduced, the network portion still uses enterprise-class networking equipment. [3] Both of these parts may be a source of network failures: relatively low efficient servers with virtualized parts of the network and physical networking equipment not designed to serve a large-scale transmission. Both may not be able to constantly serve many connections. Consequently, such environment demands extremely efficient routing mechanisms to serve great number of clients and queries, even when some physical or virtual links' failures occur.

Previous OSPF developments

As the routing mechanism is one of the key operational elements of networks, engineers still work on its improvements. As explained above, the issue of fast network recovery is more and more important in modern network models. Starting our works, we investigated many attempts of adapting OSFP protocol to the new requirements.

Many efforts were made to improve the convergence time of OSPF operations. Some of them, proposed even by the OSPF inventor, were focused on the organization of routing domains. Smaller routing areas limit the number of considered nodes while path calculation. The main drawback of this solution is that in case of huge routing domain the number of such areas would be high and demarcating the areas would be a complex and difficult task. The *hub and spokes* approach implemented in OSPF would cause an overload of *area 0* and in consequence an increase of connection reliability between all other areas.

Further propositions were focused on different aspects of OSPF operation. Some of them took under consideration e.g. the time of failure detection. [4, 5] The default failure detection in OSPF is provided by *Hello protocol*, introduced together with OSPF. All routers in OSPF domain regularly send *hello* packets to confirm they are in operational state. If the packet from particular router was not received, the neighbor would assume that corresponding link is down. This starts the paths recalculations and spreading the information across the network. To improve failure detection time researchers proposed to decrease the interval in which *hello* packets are sent and change the value of the timers that routers maintain to determine another router's state. By default, this time is equal to 10 seconds which is an extremely long period of time from the network perspective. Some of today's hardware solutions, e.g. in optical networks can detect a failure in a few milliseconds. However, decreasing *hello* interval to milliseconds may cause router's CPU overload what is detrimental in virtual environments willingly used in modern datacenters. Moreover, in a huge network topology transmission of a large number of *hello* packets would disturb regular packets forwarding. On the other hand, *hello* packets inform about the status of a router in order to establish and maintain neighbor relationship, thus they are not intended directly for informing all the nodes about the changes of the routers' links states. Hence other propositions introduced also some priority and differentiated treatment for LSA packets. As an example, Cisco IOS provides different types of LSA delays.

Other researchers made some efforts to optimize LSDB exchange process. [6, 7] The proposition was to speed up the process by minimizing the payload of Database Description packets or by sending hashed LSDB content instead of its full descriptive version. Both of these solutions demanded additional operations to be done by the router.

There were also some innovative approaches to routing table calculation. The examples include:

- replacing standard Dijkstra's algorithm by dynamic SPT algorithm,
- prioritizing the order in which calculated next-hops are installed,
- avoiding frequent routing table recalculation. [8]

The last of these solutions, from the perspective of modern data centers which are the main field of our interests, may be even undesirable in such a dynamic environment. The first two represent an outline slightly similar to our concept.

To sum up the above considerations, quoted examples show that there were many different attempts in various aspects of OSPF routing protocol mechanisms. Its complexity generates a lot of potentially suboptimal behaviors, on the other hand these are the fields of great improvement possibilities and they allow to adjust the protocol mechanisms implementation to the specific application.

Machine Learning in computer networks

Artificial Intelligence (AI) and Machine Learning (ML) approaches become more and more common in many aspects of science, from medicine to security systems, and they are also trying to enter the networks domain. Regarding the topic of this paper, we reviewed some examinations with reference to usage of AI for the networking purposes.

One of them touched the field of shortest path calculations. It was demonstrated and supported by mathematical proofs that artificial Neural Networks (NN) may be successfully used for shortest paths calculation. [9] However, the mentioned study was not done in the context of OSPF operating and authors pointed that the issue of adaptability to varying calculated topology conditions is still open. In our tests we confirmed that the time of filling all the routing table according to NN output takes more time than using standard Dijkstra's algorithm. That's because we achieved linear increase of computation time using AI, whereas for standard algorithm this parameter has logarithmic character. This is especially significant when the number of network nodes is large.

Much newer research from 2017 year directly concerned the usage of ML for routing purposes. [10] A. Valadarsky

et al. showed that hitherto routing optimizations were oriented on specific traffic conditions. ML gives a possibility to use the information about past traffic conditions to predict future traffic shape and prepare optimal routes. This is a great solution in an environment in which traffic changes are periodic (e.g. repeatable daily) and routing issues are strictly connected with traffic engineering tasks. However, it would not be so useful in dynamic topology, where network changes are unpredictable in place and time.

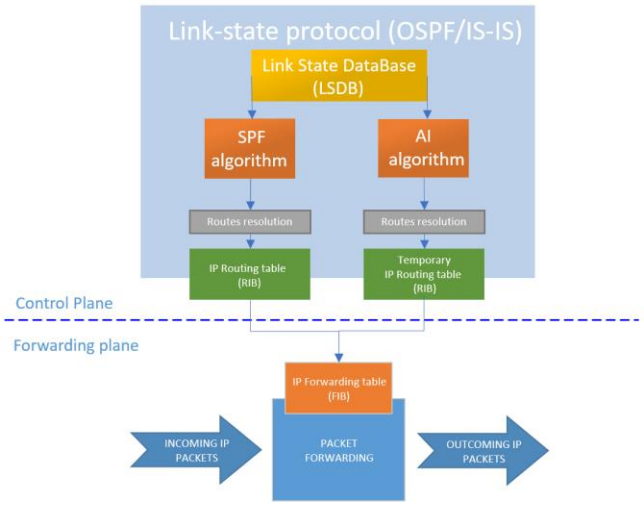

Figure 2. IP software suite with parallel NN algorithm

Proposed improvements

Our idea of improving network recovery time is to involve NN algorithms in the OSPF routing process in order to minimize the traffic loss in case of chaotic situations. This is possible thanks to the nature of link-state routing protocols in which each node (router) in the network keeps the full information about the network topology. Thanks to this, routers have enough data to feed NN for AI-based paths calculation.

Our solution is based on running AI algorithm that would compute the routes in parallel with primary OSPF mechanisms. NN would be trained to output the next hop from the local router to reach each other router in the network. The overall goal is to reduce the time needed to construct a functional FIB that is, with high probability, good enough to direct packets until the full SPF calculation has come to its end.

The additional parallel AI process may create a temporary RIB and then temporary FIB. Next, they can lead to forwarding decision which with high probability allows reaching the destination of the packets. Because of not fully predictable nature of AI algorithms, the decisions may be not 100% accurate but well-trained NN can get close to this value.

So, we are not proposing to replace the computationally intensive SPF process with NN, but to provide an alternative way to generate the useful FIB in case of network dynamic changes, as it is shown in Figure 2. The temporary FIB would be in use until the fully correct FIB is constituted by the SPF process. The AI algorithm may be trained based on previously calculated SPF routes or on a set of datasets generated in simulated environment.

Neural Network design

A Neural Network is not actually a name of some algorithm, but it is rather a set of algorithms that simulate human brain behavior. Similar to many neurons in brain, that process incoming environmental information and make some decisions based on them, NN consists of a set of artificial neurons grouped in layers. The number of such layers and the number of neurons in each layer may vary depending on the design and the complexity of a problem to solve.

A single artificial neuron is an element that takes one or more input value, combines the input with its internal state using an activation function, and produces one output value, which is then transmitted to the next level layer or is used to compose the final NN output. The layer that receives various forms of information from the outside is called input layer and the last layer, that produces the final result is the output layer. Other layers between them are the hidden layers. Between the layers there are multiple connection patterns possible. The connections have a weight parameter specifying the influence that previous neuron has on another.

There are two stages of NN operations. The main operational state in which NN produces expected output for given input is preceded by a learning stage. The learning stage is a process in which the weights of connections are established. This is done automatically by NN according to the training data it is fed with. The training dataset consists of some encoded input values for which the exact expected output value is known. Therefore, learning is the process of adjusting the weights to improve the accuracy of the result. Even after learning the error rate typically does not reach 0 and in further operational state not all of the decisions are correct, but properly selected training data can bring the correctness of the decision to the level of 100%. Once NN has been trained with enough examples it is ready to take entirely new input and it should operate with the effectiveness close to training accuracy.

In case of our testing NN design, after many probes we created a fully-connected network consisting of five layers (input layer, output layer and three hidden layers). Its scheme is shown in Figure 3. The number of neurons in input and output layers was determined by the way input/output data is encoded. Sizes of hidden layers were selected experimentally.

As an input, we give a state of all the links in the network and the destination subnet encoded as a binary vector. The first part of the vector represents operational states of particular links so the number of bits is

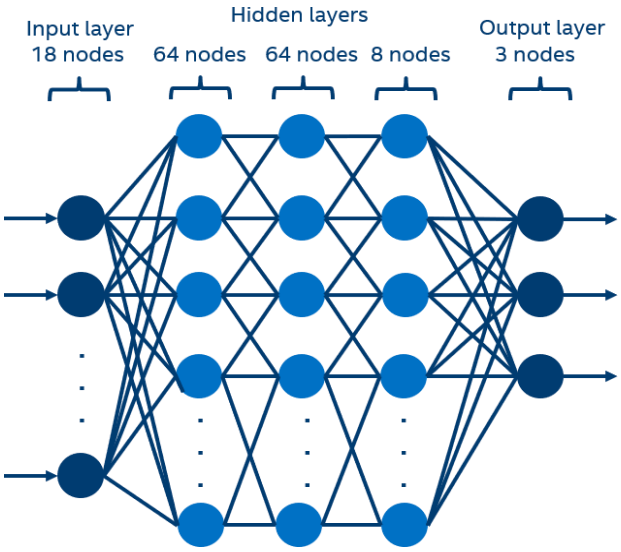

Figure 3. Designed NN scheme

determined by the number of this links. Bits with value 0 mean that link is down and with value 1 represent working links. The second part of input vector is a one-hot encoded representation of packet destination. In our simple simulation there were 9 nodes and 9 subnets in the network connected to a router with implemented NN mechanisms, so the input layer had 18 neurons – one per each link state and possible destination.

As an output we received also a binary one-hot encoded value that meant the number of router interface to which packet should be forwarded.

To use output information encoded in such a way we need to somehow pass the data to a routing table. This may be done by integrating the NN with Quagga software suite.

Quagga application

Quagga is an open source software routing suite that provides implementation of almost all of common routing protocols (including OSPF) for Unix platforms. Its architecture consists of a core daemon which intermediates to the underlying Unix kernel and presents API to Quagga clients. [10]

The OSPF client advertises the routing updates to the daemon filling FIB according to RIB. The way the RIB is created and filled with routing data is a field of modification in our project. The design assumes modification of the part of Quagga code that is responsible for path recalculation in case of receiving OSPF LSA Type 1 or Type 2 packets. If the process was indicated the parallel pre-learned AI mechanism would immediately replace outdated RIB with temporary one created with NN.

Solution details

AI algorithm will be trained separately for each node to output the next hop from the local router to reach each other router in the network. Thanks to that we will have a FIB that is, with high probability, good enough for the topological state the network is in after the topological change, until the full SPF recalculation has come to its end. NN is trained based on previously calculated SPF routes and may be retrained adaptively while network is operating. The dataset on which the AI algorithm is trained with is a per-router dataset and can be easily obtained in a simulated version of the network itself.

For the simulation purposes we consider making use of Mininet network emulation environment. It allows us to create a computer network of virtual hosts, switches, routers, controllers, and links. [11] It provides simple and inexpensive network testbed and extensible Python API for network creation. We assume the environment can run together with routing suite and provide training data for NN simulating potential issues in real network and taking the topology information from routing software.

In the simulation step one dataset for each router can indeed be easily generated, with many LSDB scenarios and all the other router-IDs as inputs and with the local-router next-hop to reach the other routers in the network as outputs.

For example, a dataset to train the AI algorithm for one router might be generated in a simulated environment considering from 1 to 3 faults in the network topology, which would correspond to a dataset size of “M choose 1 + M choose 2 + M choose 3” x N, with M number of links in the network and N number of routers, where “a choose b” means the binomial coefficient of a over b.

This way the AI algorithm would memorize the correct next hop to reach any other router in the network in the most probable failure scenarios, while relying on AI generalization capabilities for those scenarios not present in the dataset.

The AI algorithm will anticipate the SPF recalculation providing, probably, the best next hop given that LSDB state. The calculation of these next hops corresponds, from the packet forwarding perspective, to the minimum spanning tree calculation, which is computationally intensive and scale with the N^2, where N is the number of routers.

Assumed benefits

Normal OSPF operations and RIB/FIB population steps are:

- SPF calculation (each router builds a tree to reach the other routers),
- route resolutions (dependent routes are then resolved, choosing what router to reach in order to route to that destination),
- routing table build-up,
- pushing the routing table to kernel space-FIB.

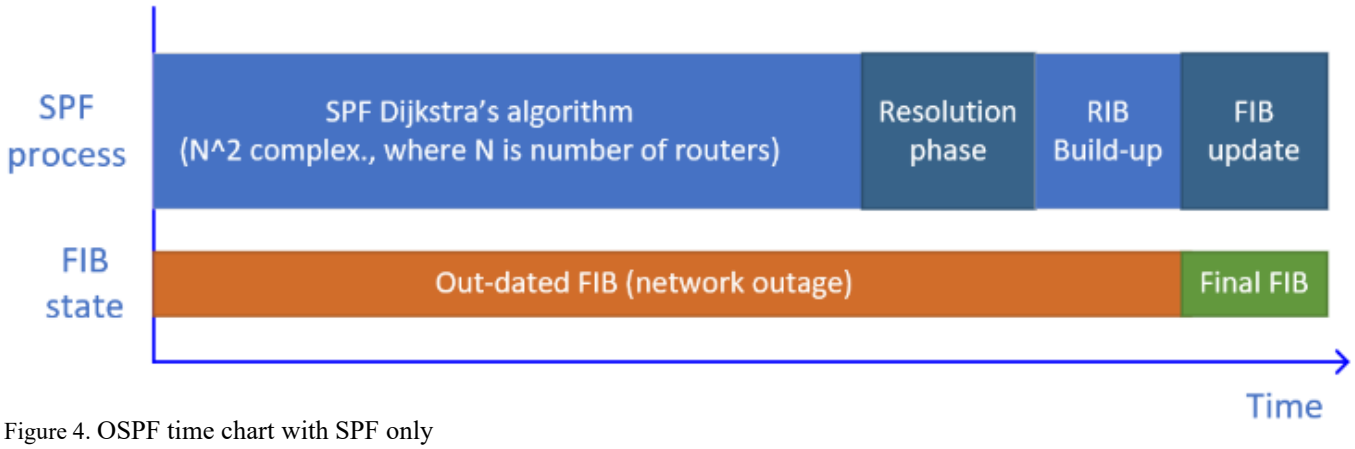

Figure 4. OSPF time chart with SPF only

These may be illustrated as shown in Figure 4. We can notice that network outage time is relatively long in this case.

Figure 5 presents the same process with parallel AI path. Temporary FIB significantly decreases the outage time and even if routing decisions based on it are not 100% accurate, they can decrease packet loss level when network state is not stable.

As mentioned above, it is worth remembering that the complexity of the generation of this FIB table with our algorithm scales with N, where N is the number our routers, as for each router we need to compute the next hop from the local router, and this is done with a constant cost that is the cost of having the AI algorithm to generate the output for each router to reach. After this, the usual resolution phase is triggered as well as the FIB generation.

Possible applications

As it was mentioned above the main targets of our solution are big datacenters with many physical and virtual devices connected in a network. The proposed OSPF recovery time optimization applies to big and dynamic networks, where full SPF recalculations often occur, and packet loss tolerance is quite low. Moreover, the AI part of offered solution may be supported by using hardware AI accelerators that, additionally, would decrease the CPU usage of datacenters servers (which may have different efficiency). We may imagine that the solution can be implemented on hardware working in a plug&play manner and thus it will accelerate datacenters network performance without significant inference in the network devices.

This project can be also adopted to be used in smart Network Interface Cards (smart NICs), that can act as a router while being a traditional hardware NIC.

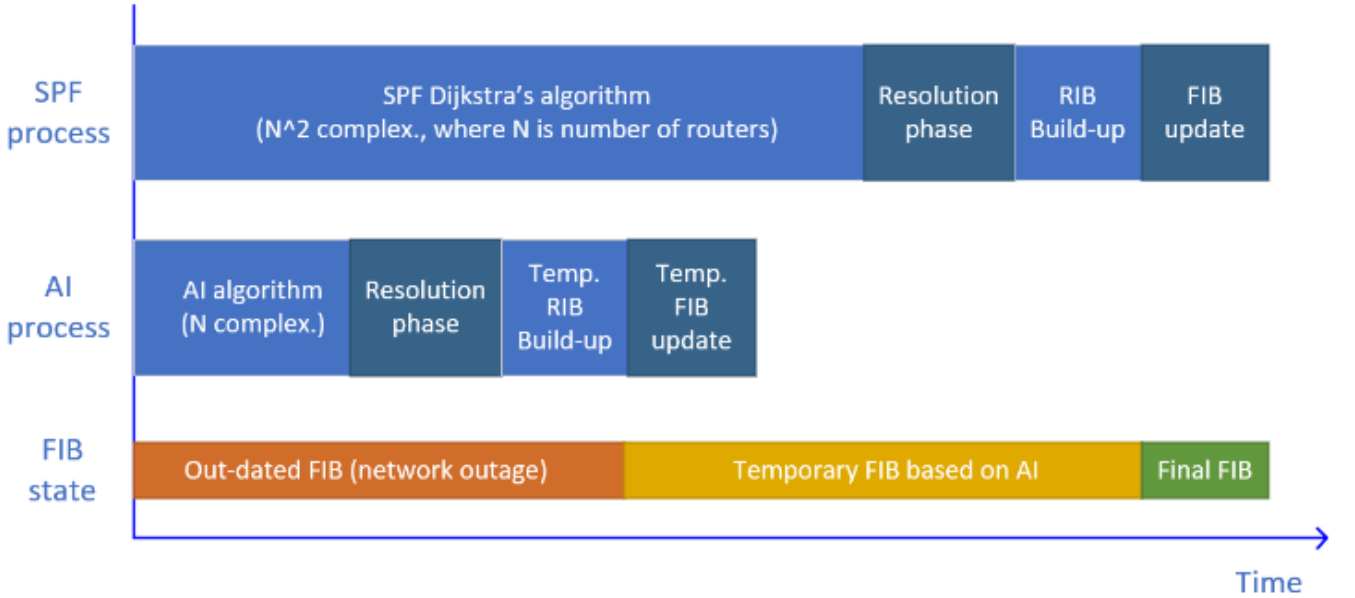

Figure 5. OSPF time chart with parallel AI algorithm

Conclusion

In this paper, we presented our idea to support OSPF routing protocol by running NN in parallel with its standard mechanisms. This may improve overall routing performance in case of unstable network state in a large and dynamic topology. We also proposed the implementation inside open source Quagga routing software suit. The results of first simulations are promising so the field of packets routing may be the next wide domain of AI applications.

References

1. Mukul Goyal, Mohd Soperi, Emmanuel Baccelli, Gagan Choudhury, Aman Shaikh, Hossein Hosseini, and Kishor Trivedi. "Improving convergence speed and scalability in OSPF: A survey". *IEEE Communications Surveys & Tutorials* 14, no. 2 (2011): 443-463.
2. Lee Kok-Keong, Lim Fung, and Ong Beng-Hui. *Building Resilient IP Networks: Building Resilient IP Networks*. Cisco Press, 2012. 146
3. Kashif Bilal, Samee Ullah Khan, Joanna Kolodziej, Limin Zhang, Khizar Hayat, Sajjad Ahmad Madani, Nasro Min-Allah, Lizhe Wang, and Dan Chen. "A Comparative Study Of Data Center Network Architectures". In *ECMS*, pp. 526-532. 2012.
4. Mukul Goyal, K. K. Ramakrishnan, and Wu-chi Feng. "Achieving faster failure detection in OSPF networks". In *IEEE International Conference on Communications, 2003. ICC'03.*, vol. 1, pp. 296-300. IEEE, 2003.
5. G. Choudhury, V. Sapozhnikov, G. Ash, A. Maunder, and V. Manral. "Prioritized treatment of specific OSPF version 2 packets and congestion avoidance". *Internet Engineering Task Force, Request For Comments (Best Current Practice) RFC* 4222 (2005).
6. R. Ogier, "OSPF database exchange summary list optimization". *Internet Engineering Task Force, Request For Comments (Informational) RFC 5243* (2008).
7. Emmanuel Baccelli, T. Clausen, and Philippe Jacquet. "OSPF database exchange and reliable synchronization in mobile ad hoc networks". In *Proc. IASTED Conference on Wireless Networks and Emerging Technologies (WNET), Banff, Canada*. 2004.
8. Pierre Francois, Clarence Filsfils, John Evans, and Olivier Bonaventure. "Achieving sub-second IGP convergence in large IP networks.". *ACM SIGCOMM Computer Communication Review* 35, no. 3 (2005): 35-44.
9. Filipe Araujo, Bernardete Ribeiro, and Luis Rodrigues. "A neural network for shortest path computation". *IEEE Transactions on Neural Networks* 12, no. 5 (2001): 1067-1073.
10. Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. "A machine learning approach to routing." *arXiv preprint arXiv:1708.03074* (2017).
11. Quagga Routing Suite website, accessed July 10, 2020, https://quagga.net/
12. Mininet network emulator website, accessed July 10, 2020, http://mininet.org/

Bibliography

1. John T. Moy. *OSPF: anatomy of an Internet routing protocol*. Addison-Wesley Professional, 1998.
2. John T. Moy. “OSPF Version 2” *Internet Engineering Task Force, Request For Comments RFC 2328* (1998).
3. David R. Oran. “OSI IS-IS Intra-domain Routing Protocol”. *Internet Engineering Task Force, Request For Comments, RFC 1142* (1990).
4. Chris Woodford. “Neural Networks”, June 17, 2020, accessed July 10, 2020, https://www.explainthatstuff.com/introduction-to-neural-networks.html
5. Larry Hardesty. “Explained: Neural networks”, Massachusetts Institute of Technology News website, April 14, 2017, accessed July 10, 2020, http://news.mit.edu/2017/explained-neural-networks-deep-learning-0414