Neural Networks in the Real World: Overcoming the Challenges of Deployment

Dr.K.Muthukannan[1], Professor
Department of IT, Sri Shanmugha College of Engineering and Technology, Pullipalayam, Morur (Post), Sankari (Tk), Salem, Tamil Nadu, India
muthukannan@shanmugha.edu.in
Mr.E.Sivarajan [2], Assistant Professor
Department of Computer Science and Engineering, Sri Shanmugha College of Engineering and Technology, Pullipalayam, Morur (Post), Sankari (Tk), Salem, Tamil Nadu, India
sivarajan.e@shanmugha.edu.in
K.P.Vijayalakshmi[3], Assistant Professor
Department of Computer Science and Engineering, Sri Shanmugha College of Engineering and Technology, Pullipalayam, Morur (Post), Sankari (Tk), Salem, Tamil Nadu, India
vijayalakshmi@shanmugha.edu.in
MS.Pavithra P M[4], Assistant Professor
Department of AI&DS, Sri Shanmugha College of Engineering and Technology, Pullipalayam, Morur (Post), Sankari (Tk), Salem, Tamil Nadu, India
pavithra.pm@shanmugha.edu.in

Abstract- The quest for self-driving networks poses growing pressure to manage network events at a nano-second scale. In this article, we make a case for leveraging programmable forwarding planes to achieve self-driving networks and respond to their dynamism in real time by network intelligence and without performing traffic steering/mirroring to centralized management solutions (intelligent or not). We briefly cover throughout the article preliminary ideas in the network neural networks field and discuss the technical challenges of running machine learning techniques entirely in the forwarding plane. We also highlight potential use cases of having an autonomous intelligent network capable of self-adapting to dynamic network behavior changes with minimal to no human intervention, including smart network telemetry, smart traffic engineering, real-time flow classification, and network tomography. We close with a roadmap of research opportunities enabled by distributed in network intelligence in programmable forwarding planes.

Keywords – performing traffic steering/mirroring, human intervention, including smart network telemetry, smart traffic engineering, real-time flow classification, and network tomography.

INTRODUCTION

The future is self-driving networks. As revolutionary as it may seem, self-driving networks will learn and adapt themselves without human intervention optimizing high-level service objectives in a context-aware, data-driven manner[1]. We envisage various potential use cases for self-driving networks: smart network telemetry, smart traffic engineering, distributed congestion control, real-time flow classification, and network tomography. These use cases may substantially benefit from neural networks running distributedly in the data plane that can thus obtain wider visibility of net-

work flows. There is a long road ahead to such networks, however[2]. Reaching them will involve closing the control and management loop, including measuring, analyzing, and controlling the network autonomously. Recent advances in machine learning (ML) have paved the way for self-driving networks, as ML models can identify patterns (analyze) and learn how to react to changes (control) in the network. Despite all the progress made in the area[3], most ML models are implemented and run in the control plane. The consequences are higher latency in the decision making process (up to tens of milliseconds, considering a control plane running at an edge with appropriate resources, e.g[4]., bandwidth and processing power) and limited applicability to just a few packets (typically the first packets of a given flow). Realizing self-driving networks' potential will require the (partial or full) deployment of ML algorithms on forwarding devices (e.g., programmable routers and SmartNICs)[5]. This, in turn, would bring the benefit to process every single packet and react to network conditions on the order of nanoseconds, with minimum control plane intervention. Recent advances in data plane programmability have enabled offloading simple ML applications to the data plane (e.g., binary neural networks). However, an intelligent and autonomous forwarding plane is not yet around the corner: forwarding devices still decide based on pre-computed lookup tables or specific states or conditions (like congestion), making them dependent on control plane heuristics[6]. This article sheds light on how networking researchers and practitioners conceive artificial neural networks (ANNs) in programmable data planes (PDPs). An ANN is a supervised ML method widely applied to find nonlinear correlations among network features. Each neural network might have different requirements (e.g., inter-layer communications) and be trained for a particular purpose (e.g., routing, monitoring, or scheduling)[7]. By running neural networks entirely on PDPs (i.e., an in-network ANN), we can benefit from high-performance forwarding devices to learn behaviors while packets are forwarded, take per-packet decisions in real time, and reduce the need for constant lookup table updates. As one can observe, the in-network ANN brings a whole new spectrum of potentialities for innovation in the networking core[8].

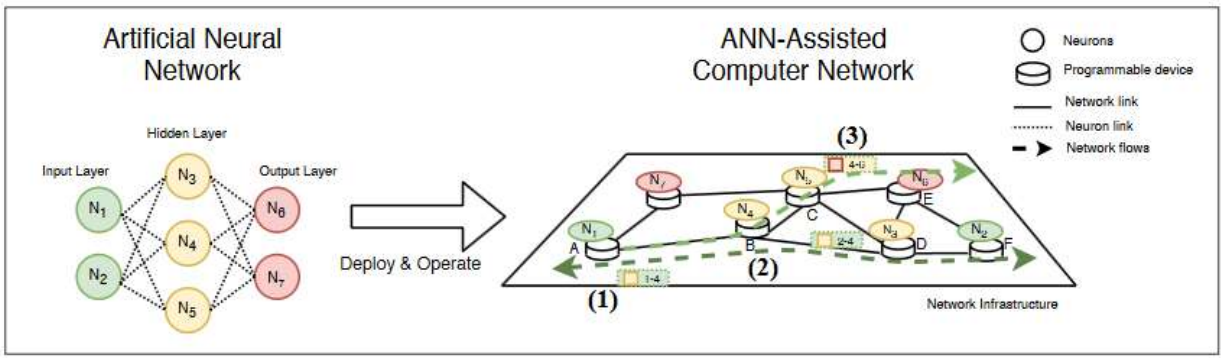

Figure 1- Example of a distributed artificial neural network being provisioned into a programmable network[7]

RELATED WORK

Programmable network devices (e.g., SmartNICs and NetFPGAs) implement variations of the protocol-independent switch architecture (PISA) to enable network developers to redefine their packet parsing and processing semantics[9]. PISA enables the definition of structural blocks (e.g., parsers, deparsers, and match-action tables) and their respective execution order. Once these structural blocks are defined, developers can build complex and intelligent network applications

(e.g., ANNs) using a data plane programming language[10]. However, network computing tasks (i.e., in network computation) must be carefully designed to deal with current limitations of programmable architectures (e.g., limited instruction set). Next, we review recent efforts toward achieving intelligent PDPs from two perspectives: the offloading to the data plane of general computations and entire ML approaches. Sapio et al. took the first steps toward offloading computations (e.g., MapReduce) to the data plane. The authors introduced DAIET, a system capable of aggregating data on routing devices. DAIET reduces network congestion while improving overall application performance[11]. Their P4 proto- type provides a data reduction ratio of around 85 percent and a similar decrease in computing time. In a follow-up work, the authors provided evidence of the feasibility of speeding up deep neural network (DNN) training by minimizing communication overhead at single-rack scale. Li et al. presented iSwitch, an in-switch acceleration solution for offloading the distributed training of reinforcement learning. iSwitch moves the gradient aggregation from server nodes into the network switches, reducing the training overhead considerably[12]. Wu et al. introduced Dejavu, a system that efficiently performs network function chaining using programmable data planes. Dejavu merges multiple functions into a single monolithic application, optimizing the packet forwarding task within a single pipeline[13]. The key idea behind merging multiple applications is to minimize recirculations required to implement a given chain, at the expense of reducing throughput super-linearly and increasing packet processing latency. Regarding the offload of ML applications to the data plane, Xiong and Zilberman introduced IIsy, a software- and hardware-based prototype for in-network packet classification. They explored packet classification using in-network supervised and unsupervised ML algorithms (e.g., decision trees, K-means, SVM, and Näive Bayes) implemented in P4[14]. In the same direction, N2Net and BaNaNa represent seminal efforts toward offloading ANN processing. N2Net proposes simplified models of ANNs, such as the binary neural network (BNN), designed for embedded applications running on devices with constrained resources and hence requiring simple arithmetic operations. N2Net implements the forwarding task of a BNN network and assumes that BNN activations (i.e., the information computed by neurons) are encoded in packet headers. The first set of layers of an ANN is performed, for example, on a general-purpose CPU. The remaining layers go through a quantization process that transforms the original ANN model into a format runnable on programmable network devices[15]. At runtime, first, the inference made by the ANN is processed on a given CPU, and then the result is encapsulated into packets and sent to a Smart-NIC. The interfaces then process the intermediate result, and outputs are encapsulated back into packets.

DESIGN OF ARTIFICIAL NEURONS AND ACTIVATION FUNCTIONS

Conceiving intelligent apps for PDPs will likely mandate a framework for creating and instantiating ANNs, akin to TensorFlow. Existing domain-specific languages for PDPs, like P4, NPL, and Lyra, do not represent feasible options, as they will impose a significant burden on ANN developers. For example, they only offer abstractions to parse and process network packets[11]. Instead, a framework for developing intelligent apps for PDPs must offer abstractions for:

- Creating neurons or perceptrons by the definition of nonlinear activation or transfer functions
- Configuration of the neuron layers and their chaining (multilayer perceptron)
- Specification of training routines The latter must include, for example, loss functions to assess the ANN precision, recall, accuracy, and F1-measure during training, error back-propagation, feature learning, and optimizers for distributed learning.

Several activation functions can be used in in-network ANNs: sigmoid, hyperbolic tangent, exponential, or rectified linear (ReLU). The most appropriate option will depend on the type of ANN developed in the network, and whether function saturation and sensitivity are determining factors[12]. ReLU activation functions have the advantage of being more suitable for existing programmable switching architectures, as they do not require exponential calculations and are thus computationally cheaper.

Solution	Deployment location	PDP computing offload strategy	Neuron communication
DAIET [3]	Hybrid control and data plane	General computing only	Done at the control plane
SwitchML [4]	Hybrid control and data plane	General computing only	Done at the control plane
iSwitch [5]	Hybrid control and data plane	General computing only	Done at the control plane
Dejavu [6]	Hybrid control and data plane	General computing only	Performed in a single program
Ilsy [7]	Data plane	Offload entire ML application	Performed in a single program
N2Net [2]	Data plane	Offload entire ML application	Performed in a single program
BaNaNa [8]	Data plane	Offload entire ML application	Done using header encoded data

Table 1- Recent efforts toward intelligent PDPs[8]

ANN TRAINING

Neural network training could be supervised by the control plane or done entirely in the data plane. In either case, operating intelligent apps in the data plane may require revisiting traditional ML research challenges. Examples include how to deal with noisy labeled training data, over-fitting, and over-training. Over-fitting refers to the situation in which the model tries to predict trends from noisy training data[13]. In this case, in addition to traditional strategies, like model simplification and random neuron dropout, ANN developers for the data plane might need mechanisms and policies that may ultimately prevent the network from taking undesired actions overproduction flows. Over-training refers to the situation in which the ANN is trained with training data contemplating small intra-class variation, or excessive training using a small training set, thus limiting the generalization capabilities of the ANN. One mitigation strategy could be defining fail-safe policies in case of high uncertainty in the ANN decisions. The in-network ANNs must be robust against adversaries attempting to tamper with its training process or manipulating flows to induce bogus responses[14]. The policy mechanism described in the previous subsection could also be used to prevent an attacker from interfering with the ANN training and operation. For example, a set of policies could determine which flows to consider for training, and another set of policies could limit the ANN's behavior, ensuring that the ANN does not make any harmful decisions on production traffic.

SMART NETWORK TELEMETRY

INT is a near-real-time monitoring approach that provides network operators superior network visibility through low-level monitoring statistics obtained directly from the data plane. INT-enabled programmable devices are instrumented by packets to collect and encapsulate telemetry information into production traffic whenever possible[15]. This telemetry information might encompass switch-internal states (e.g., queue occupancy) and performance metrics (e.g., latency), for instance. In the process of INT, the collected information is carried into a packet along its routing path. The data is extracted and reported to a collector at some point in the network, which

processes and eventually reacts to observed events. By collecting low-level statistics, INT can quickly provide the required network visibility to identify short-lived behaviors (e.g., micro-burst) and anomalies (e.g., routing violations). The rapid identification of anomalies is paramount to the success of self-driving networks (i.e., to close the loop) and the operation of network applications with strict responsiveness requirements. INT is practicable in real life and has been discussed by the Internet Engineering Task Force (IETF) and networking companies[13]. However, current efforts do not consider the importance of given telemetry data on the decision making process. In other words, regardless of its values (i.e., whether it is an expected value or not), the telemetry data is gathered and routed to an INT collector. This increases the overhead of such monitoring mechanisms substantially as additional telemetry data is transmitted[14].

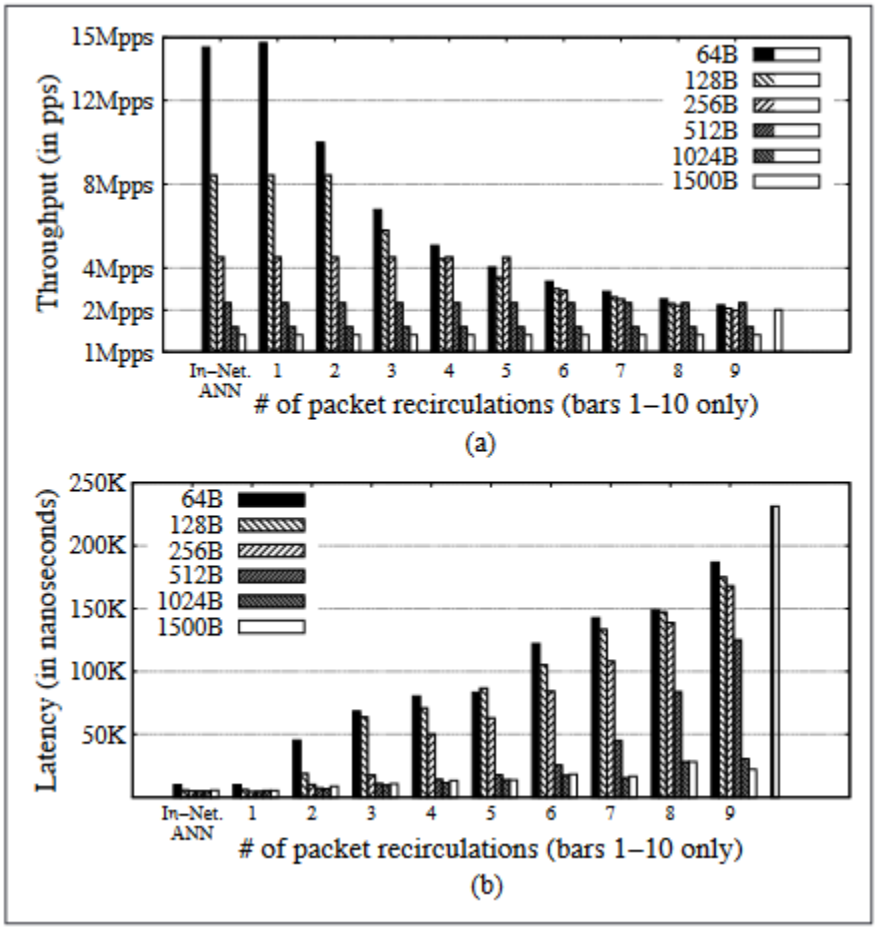

Figure 2- Analyzing the performance of an ANN implementation using packet recirculation (bars 1 to 10) in comparison to an in-network ANN in a Smart-NIC[11]

(i) Benefits- The in-network ANN could decrease the amount of telemetry data transmitted through the network. An ANN could be trained to instrument whether a single (or multiple) telemetry data is essential for collection. ANNs can be deployed in a distributed way, so the neuron's input can come from many devices. The ability to correlate multiple telemetry data and infer its behavior in real time opens up opportunities regarding the design of INT-based monitoring applications. As a proof of concept, we implemented an in-network ANN with P4, trained it offline to identify expected/unexpected INT behaviors (e.g., processing time), and operated it on a Netronome SmartNIC and on a quaternary fat-tree topology (emulated using Mininet). Our ANN

follows a three-layer structure with input neurons placed on the leaf nodes [15-20]. The data plane reports only unexpected events to an INT collector. Figure 2 illustrates the obtained throughput (Fig. 2a) and PDP latency (Fig. 2b) for an ANN implementation using a monolithic P4 application baseline (using packet recirculation) in comparison to our proposed distributed in-network ANN. Note that in the baseline application, the more the packet recirculation, the more the neurons in the ANN. Our prototype implementation reaches up to 14.52 Mp/s (e.g., for a 64 B packet), while incurring a negligible latency overhead of 9908 ns.

(ii) Challenges- When performing selective INT, one significant challenge is to keep network visibility as similar to the conventional approach, when all telemetry data is collected. In, the authors gave the first steps to instrument the collection of telemetry data via ML[14], but still limited to control plane decisions. More recently, applied randomization techniques to ease the burden of full telemetry collection. Both strategies focused on minimizing the number of telemetry items collected while keeping the network visibility updated.

CONCLUSION

The possibility of running ANNs directly in a programmable forwarding plane opens up an avenue of research directions and application possibilities for self-driving networks. However, realizing the full potential of ANNs will require devising novel frameworks for fast prototyping of neural networks, dealing with limitations of existing programmable switching architectures. It will also require proper training and operation of ANNs to ensure their effectiveness and minimize exposure to safety and security incidents. In this article, we cover use cases that highlight the potentialities of running ANNs in programmable forwarding planes and lay out challenges that must be addressed by the research community toward realizing the vision of a fully programmable, intelligent, and autonomous forwarding plane.

REFERENCES

[1]. J. Zerwas et al., “Netboa: Self-Driving Network Benchmarking,” Proc. Wksp. Network Meets AI & ML, ACM, 2023, pp. 8–14.

[2]. G. Siracusano and R. Bifulco, “In-Network Neural Networks,” CoRR, vol. abs/1801.05731, 2024; http://arxiv.org/abs/1801.05731.

[3]. A. Sapio et al., “Innetwork Computation Is a Dumb Idea Whose Time Has Come,” Proc. ACM HotNets ’17, 2023, pp. 150–56.

[4]. “Scaling Distributed Machine Learning with In-Network Aggregation,” Apr. 2023; https://www.usenix.org/conference/ nsdi21/presentation/sapio

[5]. Y. Li et al., “Accelerating Distributed Reinforcement Learning with In-Switch Computing,” Proc. Int’l. Symp. Computer Architecture, ACM, 2024, p. 279–91.

[6]. D. Wu et al., “Accelerated Service Chaining on A Single Switch ASIC,” Proc. ACM HotNets’19, 2023, pp. 141–49.

[7]. Sanjay Kumar Suman, Dhananjay Kumar and L. Bhagyalakshmi, “Non Cooperative Power Control Game with New Pricing for Wireless Ad hoc Networks”, International Review on Computers and Software, vol. 9, no. 1, pp. 18-28, 2014. ISSN: 1828-6003,

[8]. S. Porselvi, Sanjay Kumar Suman and L. Bhagyalakshmi, “Harvesting RF energy for mobile charging”, Australian Journal of Basic and Applied Science, vol. 9, no. 20, pp. 454-465, June 2015.

[9]. K. Swapna, P. Rajalakshmi and Sanjay Kumar Suman, “Security Enhancement in MANET using Game Theory”, Middle East Journal of Scientific Research, vol. 23, pp. 190-195, 2015.

[10]. VinaySrivatsan, Sanjay Kumar Suman, L. Bhagyalakshmi and S. Porselvi, "Non radiative wireless power transfer", Journal of Advances in Natural and Applied Sciences, vol. 10, no. 16, pp. 147-153, Nov. 2016.

[11]. Sujeetha Devi, Bhagyalakshmi L and Sanjay Kumar Suman, "Cluster based energy efficient joint routing algorithm for delay minimization in wireless sensor networks", International Journal of Pure and Applied Mathematics, vol. 119, no. 15, 307-313, 2018

[12]. Z. Xiong and N. Zilberman, "Do Switches Dream of Machine Learning? Toward In-Network Classification," Proc. ACM HotNets '19, 2024, pp. 25–33.

[13]. D. Sanvito, G. Siracusano, and R. Bifulco, "Can the Network Be the Ai Accelerator?," Proc. Wksp. In-Network Computing, 2023, pp. 20–25.

[14]. F. Estrada-Solano, O. M. Caicedo, and N. L. S. Da Fonseca, "Nelly: Flow Detection Using Incremental Learning at the Server Side of SDN-Based Data Centers," IEEE Trans. Industrial Informatics, vol. 16, no. 2, 2024, pp. 1362–72.

[15]. R. B. Basat et al., "Pint: Probabilistic In-Band Network Telemetry," ACM SIGCOMM 2023.

[16]. R. Hohemberger et al., "Orchestrating In-Band Data Plane Telemetry with Machine Learning," IEEE Commun. Letters, vol. 23, no. 12, Dec 2024, pp. 2247–51.

[17]. M. Hamdan et al., "Flow-Aware Elephant Flow Detection for Software-Defined Networks," IEEE Access, vol. 8, 2024, pp. 72,585–97.

[18]. K.-F. Hsu et al., "Adaptive Weighted Traffic Splitting in Programmable Data Planes," ACM SOSR, 2024, pp. 103–09.

[19]. T. Guan et al., "Recursive Binary Neural Network Training Model for Efficient Usage of on-Chip Memory," IEEE Trans. Circuits and Systems I, vol. 66, no. 7, 2023, pp. 2593–2605.

[20]. Q. Qin et al., "Line-Speed and Scalable Intrusion Detection at the Network Edge Via Federated Learning," IFIP Networking, 2024, pp. 352–60.