A Federated Learning Approach to Routing in Challenged SDN-Enabled Edge Networks

Alessio Sacco† Flavio Esposito⋆ Guido Marchetto†
†Politecnico di Torino, Italy ⋆Saint Louis University, USA

***Abstract*—The edge computing paradigm allows computationally intensive tasks to be offloaded from small devices to nearby (more) powerful servers, via an edge network. The intersection between such edge computing paradigm and Machine Learning (ML), in general, and deep learning in particular, has brought to light several advantages for network operators: from automating management tasks, to gain additional insights on their networks. Most of the existing approaches that use ML to drive routing and traffic control decisions are valuable but rarely focus on challenged networks, that are characterized by continually varying network conditions and the high volume of traffic generated by edge devices. In particular, recently proposed distributed ML-based architectures require either a long synchronization phase or a training phase that is unsustainable for challenged networks.**

In this paper, we fill this knowledge gap with Blaster, a federated architecture for routing packets within a distributed edge network, to improve the application's performance and allow scalability of data-intensive applications. We also propose a novel path selection model that uses Long Short Term Memory (LSTM) to predict the optimal route. Finally, we present some initial results obtained by testing our approach via simulations and with a prototype deployed over the GENI testbed. By leveraging a Federated Learning (FL) model, our approach shows that we can optimize the communication between SDN controllers, preserving bandwidth for the data traffic.

***Index Terms*—federated learning, machine learning, edge computing, routing.**

I. Introduction

The amount of mobile and IoT devices that has become available in the past few years generates a massive amount of data, creating several challenges and opportunities for data and network orchestration. The vast majority of these devices do not have or cannot meet the computational requirements to process the data they collect. To close this gap, in recent years, several solutions have outsourced the responsibility to perform (some or all) computations to an edge cloud [1]–[3].

Software-Defined Networking (SDN) and network virtualization have attracted interests from both academia and industry due to the powerful programmability and flexible management on networks, especially relevant in edge computing. In recent years, innovation has driven network management in two ways. On the one hand, to scale up network management capabilities, to address the limited processing ability, and to provide resiliency to the single controller model, several studies have proposed to implement the control plane with logically and physically distributed controllers [4]–[8]. In these circumstances, each controller manages a subset of switches and synchronizes with other controllers to maintain a consistent network view.

978-1-7281-5684-2/20/$31.00 ©2020 IEEE

On the other hand, Artificial Intelligence (AI) has shown strengths in almost every industry and undoubtedly is helping the field of computer networking as well. Network management has the potential to benefit from AI tremendously, as some recent work has shown [9]. Collected data underlie Machine Learning (ML) models, to enable the detection, classification, and prediction of future network events. Such information is then beneficial for load balancing, routing, resource scheduling, and other tasks. Existing ML-based network management solutions are arguably inadequate for challenged network scenarios, such as an IoT emergency network deployed to support first responders [10], [11]. Adapting to the challenged conditions would require too many frequent re-training, potentially canceling out the benefit of a learning approach; this is because the continually changing network conditions would result in considerable traffic growth not captured by the learning model [12].

In this paper, we design Blaster, an architecture whose aim is to cope with this knowledge gap. Our design goal is to merge network softwarization and Federated Learning (FL) to optimize routing (and other mechanisms) decisions in a challenged network environment. Our system is inspired by other FL systems, in which many agents cooperatively train a machine learning system (via SDN controller applications in our context) while keeping the training data decentralized. FL has revealed its potential in several applications, from the privacy-preserving and security emphasis [13], [14] to the scalability of a model than needs frequent retraining [15]. However, network management has not taken advantage of FL yet, although distributed approaches for SDN and edge computing exist [16], [17].

In our system implementation, we choose the best route according to the output of a Long Short Term Memory (LSTM) model, a type of recurrent neural network that we picked as a (tunable) regression algorithm. LSTM is typically used as a solution for deep learning-based time series prediction problems. Our LSTM receives as input the graph of the network and a traffic matrix. The output is the future load on the input links, and the SDN controller (application) can use this knowledge to adapt routing if a peak load is predicted (in challenged scenarios). The routing is hence affected by the state of the network, as opposed to traditional routing protocol where paths are set once, and future updates do not

encompass the current network load. By combining FL with deep learning-based routing, we create an intelligent traffic control system that can handle a large number of information coming from the switches.

We evaluate Blaster comparing its performance with other centralized and machine learning bases solutions in simulations and on the GENI testbed [18], an open infrastructure for testing distributed systems at scale. Our results confirm that our approach can reduce the number of messages exchanged among the SDN controllers, speeds up the training of the ML model, and hence increases data delivery performance.

The rest of the paper is organized as follows: we start describing related solutions utilizing ML and FL (Section II), we then clarify the model used by our Blaster system and the key concepts underlying our approach (Section III). Section IV shows the design of the proposed architecture and the communication between our SDN agents, while in Section V we present the evaluation results. Finally Section VI concludes our paper.

II. Related Work

The core concept behind Federated Learning (FL) approaches is to train a centralized model on decentralized data that never leaves the local environment that generated it. Rather than transferring "the data to the computation", FL aims to transfer "the computation to the data". This concept was proposed by Google recently [19]–[21], with the main idea that FL can prevent data leakage. However, recent improvements have been focusing on overcoming the analytical challenges [22], [23], or to further improve the security aspects [24], [25].

The idea behind FL is also very similar to Distributed Machine Learning. However, the latter covers several aspects, including distributed operation of computing tasks, distributed storage of training data, distributed dissemination of model results, to name a few. Among them, Parameter Server [26] is a typical element in distributed machine learning. It stores data on distributed working nodes, allocates data, and computing resources through a central scheduling node, to train the model more efficiently. Differently from all these solutions, we focus on the data privacy protection of the data owner during the model training, especially in environments in which routing and other network management operations are inconvenient or impractical to share. Moreover, in the parameter server, the central node always takes control, while in our architecture, a working node represents the data owner, with full data autonomy and freedom to join a federated learning ecosystem. Hence, the proposed model faces a more complex learning environment.

Federated learning particularly fits the edge computing paradigm, as it provides protocols for coordination and security of the learning process. In [27], authors considered the generic class of machine learning models that are trained using gradient-descent approaches. The authors analyze the convergence bound of distributed gradient descent from a theoretical point of view, based on which they propose a control algorithm that determines the best trade-off between local update and global parameter aggregation. We share with this study the design goal, but we consider a different model, based on Neural Networks and for a different Machine Learning problem and formulation.

III. Modeling Federated Learning for Routing

In this section, we describe our federated learning model. We consider N edge computing aggregates. These aggregate $\{F_1, ..., F_N\}$ represent either providers or data owners. Each aggregate is bound to a domain controlled by a single SDN controller; each controller attempts to train a machine learning model by operating on a different data shard $\{D_1, ..., D_N\}$.

In centralized machine learning, the inference system merges all data and uses $D = D_1 \cup ... \cup D_N$ to train a given model M_{sum}. In our considered federated learning system, the learning process consists of the aggregate data owners that collaboratively train a model M_{fed}. Each aggregate process (agent) F_i does not need to expose its data shard D_i to any other training processes.

In addition, a design goal of the system imposes that the accuracy of M_{fed}, denoted as Acc_{fed}, is very close to the performance of M_{sum}, Acc_{sum}. Formally, if we let δ be a non-negative real number, we say the federated learning algorithm has δ-accuracy loss if and only if: $|Acc_{fed} - Acc_{sum}| < \delta$.

Let the matrix M_i denote the data held by each data owner i, where each row of the matrix represents a sample, and each column represents a feature. We then denote the features space as X and the sample IDs space as Y. Following the Vertical Federated Learning schema [28], we split data such that data sets among agents share the same sample IDs space but differ in features. Formally, $X_i \neq X_j, Y_i = Y_j, \ \forall M_i, M_j, i \neq j$.

This is opposed to the simpler Horizontal FL [29], where agents share the features utilized, but differ for the sample IDs.

A. Federated LSTM for learning how to route

The machine learning model chosen for this study is a Long-Short Term Memory (LSTM) Recurrent Neural Network. LSTM is a class of neural networks in which connections between neural network nodes form a directed cycle. These connections allow the nodes to memorize information about what has been computed so far, giving the ability to make use of sequential information and to exhibit a dynamic temporal behavior. Our model is thus able to learn the correlation between changes in the packet distribution and routing decisions over time.

The key idea of our model is the following: *each controller in the network maintains a version of the model reflecting the entire network topology. By exploiting the Federated Learning paradigm, each controller merely has to send the weights of the (LSTM) neural network, while all other collected values are kept local, that is, within the SDN controller application. This data distribution model ensures the privacy of the data used to train the machine learning model and keeps the overhead minor (Fig. 1).*

The LSTM model has as input the historical values for the traffic on every link in the network. Given a history of N traffic

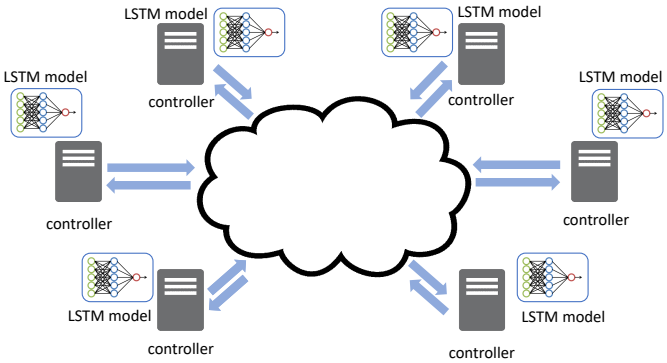


Fig. 1. Overview of the edge network and the LSTM models spread across the SDN controllers responsible for the subnet. All the values collected by the controllers are kept local. This data distribution model ensures that data privacy is preserved, while keeping the overhead minor.

values and K links in the topology, we follow the approach presented in [28], where the matrix is first inverted to apply the mechanisms of horizontal FL. For this reason, the model receives as input a matrix with dimension $k \times N$ instead of the initial $N \times K$ shape. The output is an array of K values, stating the predicted traffic for the link i in the future. The controller can then instruct the switch to avoid suboptimal links, for example, those predicted to have higher congestion.

The initial paths are decided according to the conventional learning switch procedure, very common for SDN deployments. Each switch hence acts as a traditional layer2 switch, that learns from the received packets and associates the source host to the port where the packet is received. After this initial bootstrap phase, where some packets are flooded, the controller has a complete mapping host-port that can be used for further updates. When an excessive traffic is forecast on one path, an alternative route is selected, choosing the path with the lowest predicted traffic.

The LSTM model we use in the system is determined by the number of layers, the number of neurons, $i.e.$, processing units per layer, and the interconnections between the layers. These parameters need to be tuned in a preliminary phase, where parameter exploration can reveal the best configuration. We found empirically (see Section V) that with a configuration of 4 layers and 128 neurons, we obtain the best performance in terms of accuracy and the lowest loss. Adding additional hidden layers does not improve the performance of the deep learning system. In the following sections, we show how to utilize our LSTM model for the (federated) computation of the future traffic value per link.

IV. System Design

The algorithm runs at the edge of the network, where multiple controllers communicate to reach a consensus, as described in Fig. 1. The edge of the network is divided into partitions. Each partition contains different switches to which hosts and servers are connected. These switches are managed by a controller (in our implementation, we used Floodlight). The controller keeps track of the topology and the metrics coming from the switches, saving these values on a graph database. All partitions communicate and create a view of the entire network, while each management agent (controller) manages the sub-network. The SDN controller exposes as REST APIs the file descriptor (pickle file) of the LSTM, that can be used to improve the overall model, by iterating over the previous model and virtually includes the other data. This LSTM exchange is thus done periodically, to guarantee the freshness of information, but not to overload the controller. Aside from the LSTM data, the controllers send messages when the topology under control changes. The graph can evolve, for example, after a failure, and to address this situation, we manage the hash of the graph to identify the latest version. The controller detects a new change in the network employing hash computation: for every message about a status update, a graph's hash is calculated (with SHA256) and compared to the one in memory. If they match, the message is discarded, as it refers to the same network topology. In case the hash values are different, the controller computes the new topology, by leveraging the links and ports notification of switches. This information is then sent to all the other controllers. Once the new graph is obtained, the controller must set potentially new next hops for all the managed switches, and the information is sent to them. Usually, topology's updates are rare; hence messages sent between controllers often carry LSTM's weights.

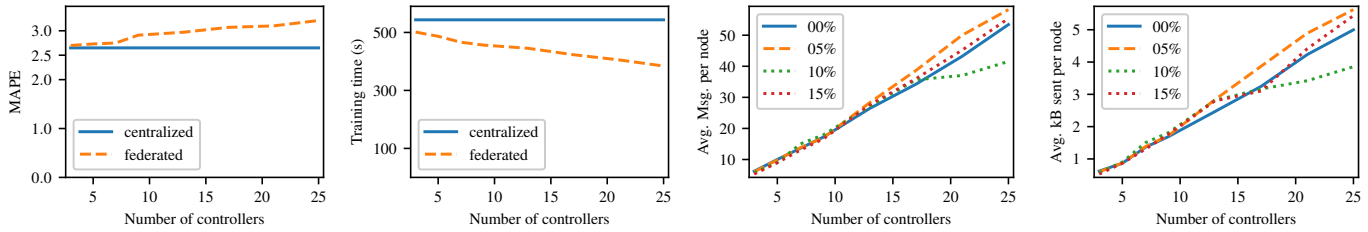
The SDN controllers select the best route for the managed switches, according to the output of the LSTM model. Such a model has as input the graph of the network, in conjunction with the traffic of the links. When a peak load is predicted, the controller verifies the link is under its control. If not, the controller neglects this data and continues its work. If the link is related to a switch controlled, a new route is obtained, avoiding such a link. This can be easily obtained by applying Dijkstra, where the link assumes a high weight. Since all the controllers have a global view of the network, the path selection process can be easily performed by each of them.

V. Evaluation Results

We validate the presented algorithm in real case scenarios, using the GENI [18] virtual network testbed, which provides physical machines and physical links for testing purposes. In the following, we deploy obtained different instances of the algorithm on the GENI testbed, and we evaluate the practicality of our approach under different conditions. The testbed consists of at most 25 SDN controllers, where each of them manages five switches and 25 hosts.

We compare the performance of the two approaches: *(i) centralized,* where the controller has the entire history available, and train the LSTM on it; *(i) federated,* where the controller has a small portion of data, but receives the LSTM's weights (related to the missing data) from neighbors. To this end, we compute the Mean Absolute Percentage Error (MAPE) which is given by:

$$MAPE = \frac{1}{n}\sum_{t=1}^{n} 100 \times \left| \frac{y_t - \overline{y_t}}{y_t} \right|, \qquad (1)$$



(a) (b) (c) (d)

Fig. 2. (a-b) MAPE error and the training time, respectively compared to a centralized solution. **Take home (a-b)**: The Loss of details of our distributed solutions does not translate into a significant performance loss. (c-d) Performance varying % of node failures. Number of messages exchanged for increasing complexity topology (c), and the number of kB sent (d). **Take home (c-d)**: the overhead is almost independent from the failure rate.

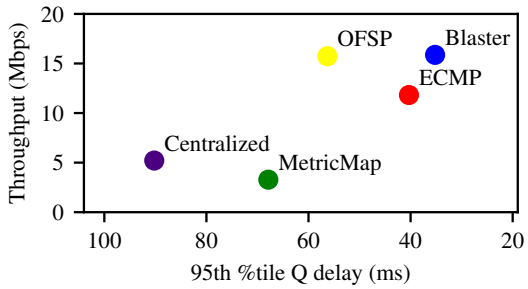


Fig. 3. Comparison for different routing strategies, highlighting the trade-off between latency and throughput. Our Blaster algorithm outperforms state-of-the-art (ML-based) approaches.

where y_t and $\overline{y_t}$ are the real and the predicted observations. A small MAPE indicates a small error, hence better performance. We ran an edge computing application [10] on the GENI testbed, and saved on file the traffic on links through the Floodlight's API. We collected a dataset made of more than 50,000 historical samples split into training (80%), validation (10%), and test (10%); the error is computed on the test only.

Fig. 2(a) shows the MAPE error for the two approaches for increasing size topologies. In the centralized version, the error is lower than in federated because the training set is more significant than in federated cases, hence the model can use more information and more knowledge. On the other hand, in our federated architecture, we use fewer values in the training phase, which leads to higher MAPE. However, the difference between the two techniques is negligible, and in the worst case, it is 0.51. Although for a more significant number of nodes, the MAPE increases, it slightly rises, and we can conclude that even for our federated use case, MAPE is almost constant for different topology's dimensions. In other words, we can state that the δ quantity previously defined (Section III) is relatively small and increases when the number of controllers rises.

Furthermore, in Fig. 2(b), we evaluate the time necessary to train the LSTM model for both centralized and federated use cases. In the centralized approach, the controller manages more processes, and the time to train the LSTM network is particularly high. With our approach, we observed a training time reduced by 30% when 20 SDN controllers are present in the network. In this case, the history of the traffic load is spread across more agents, and each of them can perform training on a small portion of data.

We then measure the number of messages needed for updating the models in case of failures in the network. The

TABLE I

MAPE VALUES COMPARISON FOR DIFFERENT CONFIGURATIONS OF LAYERS AND NEURONS OF THE LSTM MODEL. THESE QUANTITIES ARE USED FOR THE DEEP LEARNING MODEL TUNING.

MAPE	Neurons					
Layers	4	8	16	32	64	128
2	7.51	4.27	3.41	3.54	2.74	2.79
4	8.33	4.18	3.61	3.26	2.88	**2.65**
6	9.95	4.57	3.85	3.37	2.99	2.81
8	10.08	4.96	3.14	3.10	2.72	2.68

notation 05% in the graph, and the similar variants, indicates a node and link failure probability of 5% throughout the entire experiment. The first result we can see in Fig. 2(c) is that updates in the graph do not particularly affect the messages exchanged, nor the bytes (Fig. 2(d)). We can notice how increasing the number of nodes increases the messages exchanged among controllers, and consequently, the number of bytes. However, even in the presence of numerous topology updates, the messages required by the protocol do not increase. When 20% of items can fail, for a large number of nodes in the network, the messages needed are less than in case of no failures. These results confirm the scalability of the system and the protocol, and the responsiveness guaranteed by our system.

We then evaluate the routing optimization when we apply our model on a congested network. In particular, we compare our approach against the Equal-Cost-Multi-Path (ECMP), Online Flow Size Prediction (OFSP) [30], a centralized learning schema, and against MetricMap [31]. *ECMP* is a well-known and deployed algorithm, and we use it as a baseline. In *OFSP*, the Gaussian Process Regression (GPR) algorithm detects elephant flows; hence, the least congested path to route such flows is selected while the ECMP protocol is used to route mice flows. *MetricMap* uses the Very Fast Decision Tree (VFDT) online algorithm to learn and classify traffic. The routing protocol is atop MintRoute and specified for Wireless Networks, but can be generalized. Fig. 3 shows the throughput and delay of communication between two hosts of the network. While centralized approaches route the traffic on sub-optimal paths, since the time to re-train the model is time-consuming, with our Blaster algorithm, routing adjustments are more immediate and effective.

This strategy is particularly effective for IoT/Edge applications driven by SDN, when the underlying network has

challenged by unstable conditions, for example, in case of an emergency network setup by first responders in a natural or human-made disaster scenario. With our federated model, traffic is more likely to flow on under-utilized links, that lead to higher throughput and lower delay compared to all the other tested protocols.

We report in Table I the MAPE error for the parameter exploration of the LSTM model. We train the neural network on the previous dataset, for different combinations of layers and neurons. Results suggest that, under these conditions, more layers do not significantly improve performance. Noticeable improvements instead arise when we increase the number of neurons in each LSTM layer: 4-layers 128-neurons achieves the lowest error and therefore guarantees the best model.

VI. Conclusion

In this work, we presented Blaster, a federated architecture for task offloading at the edge of the challenged network. In the context of an SDN architecture, we realize a multi-agent control plane, where each controller manages a subset of switches and synchronizes with other controllers to maintain a consistent network view. The architecture aims to optimize the selection of the best path and takes into account the condition and the global traffic of the SDN network. Each controller keeps an LSTM model, used for predicting the future traffic on links based on history. When a peak load is predicted, the controller must select a new route, to avoid links with high load and take the ones under utilized. By leveraging Federated Learning concepts, all the controllers are able to have a global view of the infrastructure while exchanging very few messages among them. In such a way, entities involved do not need the send the entire information, but just a small portion, hence preserving the bandwidth for the application traffic.

Acknowledgement

This work has been partially supported by NSF under Award Numbers CNS1647084, CNS1836906, and CNS1908574.

References

[1] M. Sharifi, S. Kafaie, and O. Kashefi, "A survey and taxonomy of cyber foraging of mobile devices," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 4, pp. 1232–1243, 2011.

[2] J. Wang, J. Pan, F. Esposito, P. Calyam, Z. Yang, and P. Mohapatra, "Edge cloud offloading algorithms: Issues, methods, and perspectives," *ACM Comput. Surv.*, vol. 52, no. 1, pp. 2:1–2:23, Feb. 2019.

[3] A. Sacco, F. Esposito, P. Okorie, and G. Marchetto, "Livemicro: An edge computing system for collaborative telepathology," in *Proceedings of the 2nd USENIX Workshop on Hot Topics in Edge Computing*, ser. HotEdge '19. New York, NY, USA: ACM, 2019.

[4] T. Koponen, M. Casado *et al.*, "Onix: A distributed control platform for large-scale production networks." in *OSDI*, vol. 10, 2010, pp. 1–6.

[5] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, vol. 3, 2010.

[6] Z. Guo, W. Feng, S. Liu, W. Jiang, Y. Xu, and Z.-L. Zhang, "Retroflow: Maintaining control resiliency and flow programmability for software-defined wans," *arXiv preprint arXiv:1905.03945*, 2019.

[7] A. S.-W. Tam, K. Xi, and H. J. Chao, "Use of devolved controllers in data center networks," in *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2011, pp. 596–601.

[8] S. H. Yeganeh *et al.*, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot topics in software defined networks*. ACM, 2012, pp. 19–24.

[9] R. Boutaba, M. A. Salahuddin, N. Limam *et al.*, "A comprehensive survey on machine learning for networking: evolution, applications and research opportunities," *Journal of Internet Services and Applications*, vol. 9, no. 1, p. 16, Jun 2018.

[10] A. V. Ventrella, F. Esposito, and L. A. Grieco, "Load profiling and migration for effective cyber foraging in disaster scenarios with formica," in *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*. IEEE, 2018, pp. 80–87.

[11] D. Chemodanov, F. Esposito *et al.*, "AGRA: ai-augmented geographic routing approach for iot-based incident-supporting applications," *Future Generation Comp. Syst.*, vol. 92, pp. 1051–1065, 2019.

[12] Z. M. Fadlullah, F. Tang *et al.*, "State-of-the-art deep learning: Evolving machine intelligence toward tomorrow's intelligent network traffic control systems," *IEEE Communications Surveys & Tutorials*, vol. 19, no. 4, pp. 2432–2455, 2017.

[13] Q. Yang, Y. Liu, T. Chen, and Y. Tong, "Federated machine learning: Concept and applications," *ACM Transactions on Intelligent Systems and Technology (TIST)*, vol. 10, no. 2, p. 12, 2019.

[14] P. Kairouz, H. B. McMahan, B. Avent *et al.*, "Advances and open problems in federated learning," *arXiv preprint arXiv:1912.04977*, 2019.

[15] P. Moritz, R. Nishihara *et al.*, "Ray: A distributed framework for emerging ai applications," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 561–577.

[16] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed sdn controller," *ACM SIGCOMM computer communication review*, vol. 43, no. 4, pp. 7–12, 2013.

[17] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," in *2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–4.

[18] Geni. [Online]. Available: https://www.geni.net/

[19] J. Konečnỳ, H. B. McMahan, D. Ramage, and P. Richtárik, "Federated optimization: Distributed machine learning for on-device intelligence," *arXiv preprint arXiv:1610.02527*, 2016.

[20] J. Konečnỳ, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," *arXiv preprint arXiv:1610.05492*, 2016.

[21] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[22] V. Smith, C.-K. Chiang, M. Sanjabi, and A. S. Talwalkar, "Federated multi-task learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4424–4434.

[23] Y. Zhao, M. Li, L. Lai, N. Suda, D. Civin, and V. Chandra, "Federated learning with non-iid data," *arXiv preprint arXiv:1806.00582*, 2018.

[24] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans, "Secure linear regression on vertically partitioned datasets." *IACR Cryptology ePrint Archive*, vol. 2016, p. 892, 2016.

[25] K. Bonawitz, V. Ivanov, B. Kreuter *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2017, pp. 1175–1191.

[26] Q. Ho, J. Cipar, H. Cui *et al.*, "More effective distributed ml via a stale synchronous parallel parameter server," in *Advances in neural information processing systems*, 2013, pp. 1223–1231.

[27] S. Wang *et al.*, "When edge meets learning: Adaptive control for resource-constrained distributed machine learning," in *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*. IEEE, 2018, pp. 63–71.

[28] W. Du, Y. S. Han, and S. Chen, "Privacy-preserving multivariate statistical analysis: Linear regression and classification," in *Proceedings of the 2004 SIAM international conference on data mining*. SIAM, 2004, pp. 222–233.

[29] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 2015, pp. 1310–1321.

[30] P. Poupart, Z. Chen, P. Jaini *et al.*, "Online flow size prediction for improved network routing," in *2016 IEEE 24th International Conference on Network Protocols (ICNP)*, Nov 2016, pp. 1–6.

[31] Y. Wang, M. Martonosi, and L.-S. Peh, "Predicting link quality using supervised learning in wireless sensor networks," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 11, no. 3, pp. 71–83, Jul. 2007.