Eliminating Communication Bottlenecks in Cross-Device Federated Learning with In-Network Processing at the Edge

Shouxi Luo[1] Pingzhi Fan[1] Huanlai Xing[1] Long Luo[2] Hongfang Yu[2]
[1]*Southwest Jiaotong University* [2]*University of Electronic Science and Technology of China*

Abstract*—Nowadays, cross-device federated learning (FL) is the key to achieving personalization services for mobile users and has been widely employed by companies like Google, Microsoft, and Alibaba in production. With the explosive increase of participants, the central FL server, which acts as the manager and aggregator of cross-device model training, would get overloaded, becoming the system bottlenecks. Inspired by the emerging wave of edge computing, an interesting question is: *could edge clouds help cross-device FL systems overcome the bottleneck?

This article provides a cautiously optimistic answer by proposing INP, an FL-specific In-Network Processing framework, along with the novel Model Download Protocol of MDP and Model Upload Protocol of MUP. With MDP and MUP, edge cloud nodes along the paths in INP can easily eliminate duplicated model downloads and pre-aggregate associated gradient uploads for the central FL server, thus alleviating its bottleneck effect, and further accelerating the entire training progress significantly.

***Index Terms*—Federated learning, in-network processing, edge computing, transport protocol**

I. Introduction

By sharing the models rather than the raw privacy-sensitive data, cross-device Federated Learning (FL), an emerging distributed machine learning approach that enables end devices like mobile phones to train models cooperatively, has been widely deployed in production for personalization services like *on-device item ranking*, *next-word prediction*, *content suggestions for on-device keyboards*, and *real-time e-commerce recommendations* [1]–[5]. Generically, FL models are trained iteratively: in each round, a set of dynamically selected end devices (EDs) first download the current model from the central FL server (FLS) to launch the on-device training, then upload their local gradients back to the FLS, which would aggregate received gradients to obtain the new model [1]. Obviously, with the increase in the number of EDs, the central FLS would become the bottleneck of the entire FL system. Optimizing the performance of FL systems, especially removing the bottleneck effects of FLS, becomes the key to support very large-scale federated learning tasks [3].

This work. To enhance the performance of FLS, in this article, we analyze the benefits of deploying in-network processing nodes at edge clouds to provide cache and aggregation services for large-scale cross-device federated learning, and propose the FL-specific **I**n-**N**etwork **P**rocessing framework of INP. We find that using edge nodes for in-network model caching and gradient aggregation brings a lot of benefits to FL systems. Nevertheless, a pair of new domain-specific transport protocols breaking the conventional wisdom of end-to-end transport semantics are required, as existing protocols like the raw TCP and UDP could not provide the needed features.

To fill the gap, we further propose the Model Download Protocol (MDP) and Model Upload Protocol (MUP) along with INP. Consider that multiple EDs residing in the same region are performing the same FL task. By configuring edge cloud nodes to cache recently downloaded model chunks for possible subsequent duplicated requests from other EDs, and pre-aggregate the gradient uploads from different EDs in a short time interval, INP together with MDP and MUP is able to reduce both the traffic load of FLS and the time needed by model download and gradient upload, for an FL task involving m EDs, from the magnitudes of $O(m)$ to $O(1)$ at most.

Novelty. Distinguished from the alternative idea of using edge servers as local parameter servers for cross-device FL [3], [6], both the cache of model and the aggregation of gradients in INP can be implemented as specific types of in-network processing services [7] upon existing Network Function Virtualization (NFV) platforms [8], providing best-effort service for cross-device FL traffic at the edge. As Section II-B will show, such a design is easy-to-manage, generic, future-proof, and achieves very fine-grained resource usage. Although a lot of recent papers have put forward the vision of enhancing the performance of mobile/IoT applications by deploying NFV-enabled services in edge clouds, they mainly focus on the abstracted coarse-grained resource allocation problem, without considering the detail of how functions could be implemented and supported by the underlying network [9], [10]. Indeed, as this article will show, by using the domain-specific transport protocols of MDP and MUP, INP would achieve very flexible resource allocation at the granularity of per-packet. For in-network model cache, the well-known data-centric network architecture of NDN (Named Data Networking) meets the requirements naturally. However, NDN has not been supported by today's Internet yet because of its clean-slate, incompatible design [11]. Instead, INP only relies on widely-deployed techniques thus is readily-deployable.

Contributions. Our main contributions are four-fold:

- A thorough analysis that identifies the benefits and chal-

The work of Shouxi Luo was supported by NSFC Project 62002300 and China Postdoctoral Science Foundation Project 2019M663552. The work of Pingzhi Fan was supported by NSFC Project 62020106001 and 111 Project 111-2-14. The work of Long Luo was supported by NSFC Project 62102066.
Shouxi Luo is the corresponding author (sxluo@swjtu.edu.cn).

lenges of applying edge-cloud-based in-network processing for large-scale cross-device FL and sheds light on future directions.
- INP, an FL-specified In-Network Processing framework enabling cross-device FL systems highly scalable, and accelerating training processes significantly.
- MDP and MUP, two UDP-based domain-specific transport protocols achieving efficient model downloads and gradient uploads for FL, by using the in-network cache and aggregation services provided by edge boxes in INP.
- A simulation-based case study confirming the significant benefits of MDP, MUP, and INP.

Next, we first introduce the background and motivation of employing in-network processing for cross-device FL in Section II, then propose our INP framework together with its model download and upload protocols in Section III and IV, respectively. A case study of INP's performance follows in Section V. Finally, Section VI concludes the article.

II. Background and Motivation

FL applications today can be broadly categorized into two kinds, namely cross-device FL and cross-silo FL, respecting whether the participating clients are resource-limited devices (e.g., mobile phones, vehicles), or source-abundant clusters (e.g., private clusters own by financial or medical organizations), respectively [5]. In this article, we focus on the optimization of cross-device FL systems, in which a large number of end devices are dynamically selected to train a global model iteratively over wireless connections, with the assistance of a centralized FLS as Figure 1 shows.

In the following, we first overview the communication patterns involved in cross-device FL (§II-A), then discuss why in-network processing at the edge is more promising than the alternative idea of edge-based hierarchical FLS (§II-B), and finally review why existing in-network aggregation solutions designed for intra-datacenter distributed machine learning could not solve the problem we focus on (§II-C).

A. Communication Patterns in Cross-Device FL

A cross-device FL system generally involves one logical central parameter server (i.e., FLS) and a lot of dynamically available clients (i.e., EDs); the iterative training is conducted in rounds and each round is made of three phases namely *selection*, *configuration*, and *reporting* as Figure 1 sketches. To drive a round of training, the FLS first selects a set of EDs as the participant training clients from those checked in recently (i.e., selection); then, these selected EDs download the new model from the FLS to start the training (i.e., configuration); once completing their local training, EDs upload their local gradients to the FLS (i.e., reporting), based on which, the FLS will obtain the updated global model via aggregations [1].

Obviously, with the number of participants scaling up, the central FLS becomes the bottleneck of the entire training. To handle this problem, today's FL systems in production have to only select a few hundred out of tens of thousands of available devices for each round of training, resulting in slow convergence [1]. With the emerging and widespread employment of edge clouds, one promising optimization for cross-device FL is to employ nodes at the edge to cache the downloaded model for the elimination of duplicated fetching, and pre-aggregate multiple correlative gradient upload requests, from a group of nearby EDs within a short interval, so that both the traffic and computation loads on the FLS can be greatly reduced. Indeed, such an optimization design could be treated as a specific in-network processing service customized for cross-device FL [7], yielding two levels of advantages:

- **Making FL systems highly scalable**. With edge-based in-network processing, not only the traffic load on the FLS, for both the model download and gradient upload would be greatly reduced, but also parts of the aggregation computation originally conducted by the FLS, are offloaded and distributed to edge nodes. This makes FL systems easy to adopt a large number of devices in each round of training.
- **Accelerating the training significantly**. With the cache and aggregation services provided by edge nodes, it would take less time for EDs to download the model and upload their local gradients. As the delivery of data generally takes a non-trivial proportion or even dominates the entire time cost of each round of training, reducing the time on delivery would accelerate the training speed. Moreover, to avoid negative impacts on the user experience, EDs only train models when they are idle and would abort the training once conditions are no longer met [1]. Thus, the availability of an ED is perishable and the time slot might be short. Accordingly, reduced communication time costs, along with a scaleable FLS, would enable more EDs to contribute to each round of training, accelerating the entire training progress in turn.

B. Why In-Network Processing Instead of Hierarchical FLS

Compared with performing in-network processing at the edge, a very related alternative design is to deploy regional parameter servers at the edge, which together with the central FLS aggregate devices' gradients hierarchically (i.e., Hierarchical FLS) [3], [6]. We argue that in-network processing is more attractive in three aspects as Table I summarizes.

More specifically, in terms of applicability, hierarchical FLS is an FLS-specific solution. FL applications owned by different companies and organizations are generally built upon their own specific versions of FL systems. To support them all, hierarchial FLS has to deploy FLSes at the edge for each FL system separately and explicitly. In practice, these edge FLSes generally run inside virtual machines (VMs) or Linux containers with pre-configured resources; accordingly, edge cloud resources are allocated in a very coarse manner, at the granularity of VMs or containers. Moreover, in hierarchical FLS, all the deployed edge FLSes together with the central FLS form a distributed system, thus complicated gradient synchronization protocols are needed [3].

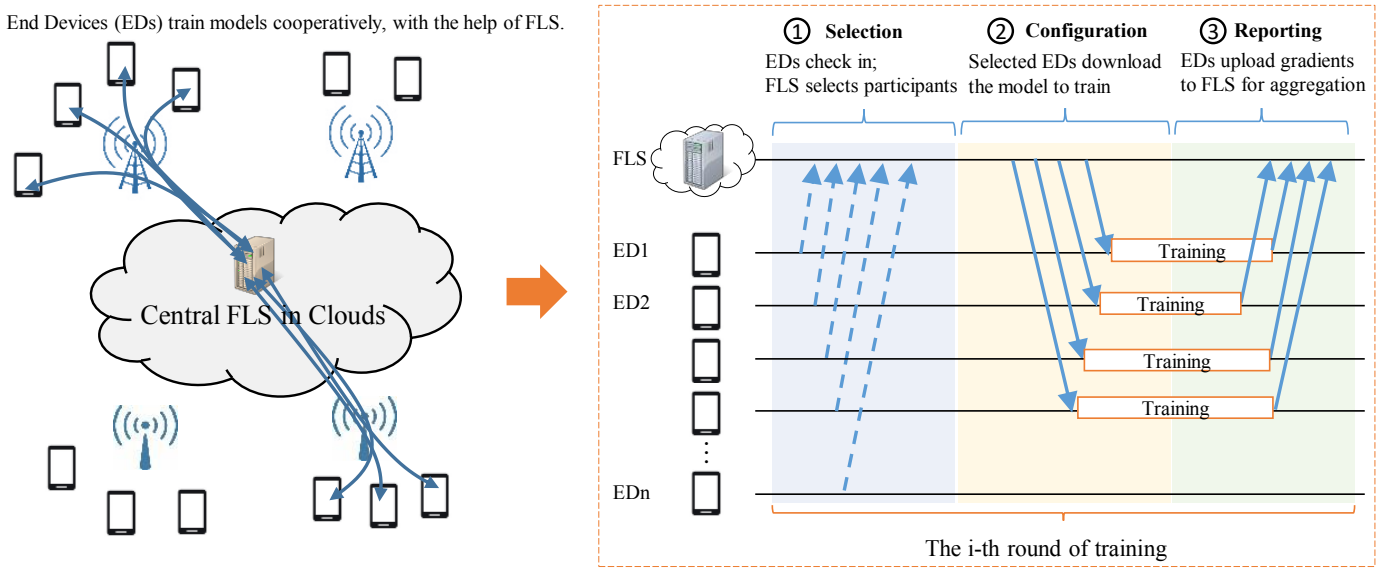


Fig. 1: The workflow and communication patterns of cross-device Federated Learning (FL) [1].

TABLE I: In-Network Processing vs. Hierarchical FLS.

Solution	Applicability	Resource allocation	Manageability
Hierarchical FLS	FLS-specific, requiring case-by-case designs and deployments	Coarse-grained, per-VM or per-container	Complicated
In-Network Processing	FLS-agnostic, generic and future-proof	Fine-grained, per-packet	Easy

In contrast, as we will show through this article, in-network processing can be implemented as an optional edge service that provides best-effort in-network model caching and gradient aggregating for FL tasks. Indeed, in-network processing is an FLS-agnostic solution: in practice, the most common model aggregation strategy is to compute a weighted average of the gradients [3]; by implementing this function as a network service, in-network processing based solution is generic and able to support present and future FL algorithms. Also, there is no need to deploy and run separate software instances for different FL systems. For each FL training job, the in-network processing instance can determine the cache or aggregation operation of each packet solely, resulting in packet-level resource allocation. Moreover, as an optional network service, the management of in-network processing is simple and decoupled from those of the supported FL systems.

C. Related Solutions and Their Limitations

Indeed, INP is not the first proposal that accelerates distributed model training with in-network processing. Recently, researchers apply a similar idea named in-network aggregation for intra-DataCenter Distributed Machine Learning (DC-DML), by configuring dataplane-programmable Top-of-Rack (ToR) switches in the path as aggregators [12]–[14]. However, since the workflow, underlying network, and available intermediate processing nodes of cross-device FL are quite distinct from those of DC-DML [12]–[16], the solutions they prefer differ in many design aspects significantly.

Take ATP, the state-of-the-art in-network aggregation based solution designed for DC-DML [12], as an example. Since DC-DML systems are networked with high-throughput low-latency links and use the same set of workers for each round of training, ATP directly employs the arrival of a model chunk's new value as the acknowledgment of the associated gradient upload triggered in the previous round; on the contrary, gradient uploads in cross-device FL would not always be followed by model downloads, as the participants of each round change dynamically. Likewise, ATP employs Layer 3 multicast to deliver the new model; while cross-device FL prefers to implement Layer 7 based cache at the edge cloud since Layer 3 multicast is unavailable in the involved network. Moreover, the congestion control of ATP relies on ECN, but many devices in cross-device FL might do not enable ECN. As well, ATP only supports simple resource allocations since its aggregation logic is implemented in specific P4-programmable hardware for line-rate processing; instead, cross-device FL is faced with slow and high-latency connections, thus any advanced schemes that can be implemented in software would work. A promising design is to implement INP's cache and aggregation services upon existing NFV platforms [8].

III. INP Framework

As Figure 2 shows, a typical INP-enhanced cross-device FL system involves three types of elements, namely *FL Server (FLS)*, *End Device (ED)*, and *Edge Box (EB)*, respectively, in which the EB acts as an optional cache and aggregation box residing between the other two. To start a round of training, FLS first selects a group of EDs to pull the new model, using the cache service provided by EBs. When completing the training, EDs push their local gradients to the FLS, through

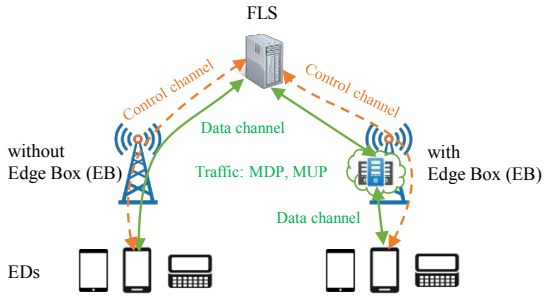


Fig. 2: The framework of INP.

the optional aggregation services provided by EBs. Once getting sufficient gradients, the FLS generates the final global aggregated model and move to the next round of training.

Inside INP, participant elements set up two types of channels, i.e., *control channel* and *data channel*, for the involved data transmissions, respecting whether their transfers can be optimized/accelerated by the EBs of INP, or not. Basically, the *control channel* is employed for the exchange of signal messages like the *join*, *leave*, and *select* of EDs, and such a feature is already supported by modern commercial FL systems [1], [2], while the *data channel* is used for the delivery of model and gradient values, with the assistance of EBs.

To get the best advantage from EBs for *data channels*, INP employs two novel transport protocols, namely MDP (Model Download Protocol) and MUP (Model Upload Protocol), to achieve efficient and reliable model downloads and gradient uploads, respectively. Both protocols are built upon UDP thus easy to deploy. Model and gradient values in INP are split into chunks, each of which, along with the MDP or MUP header, is encapsulated in a single UDP packet. Such a design not only enables INP traffic safe to go through today's wide-area networks made up of a lot of middleboxes, but also makes the proposed MDP and MUP easy to deploy at end devices like smartphones, since they can be implemented at the application layer without touching the network stack.

EBs in INP are designed to act as transparent proxies between EDs and the central FLS. For FL systems involving a huge amount of EDs across multiple regions, a group of EBs could be deployed and configured to work hierarchically, such that the load of the central FLS could be further reduced.

IV. INP Protocols

Now, we present how INP protocols achieve efficient model downloads and gradient uploads, by using EBs as cache nodes and in-network aggregators, respectively. In production, the functions of both the cache and aggregation involved by EBs are easy to implement as software, thus EBs can be built upon today's NFV systems [7], [8]. To be practical, INP protocols are designed to work gracefully without EBs.

A. MDP

As Figure 3 shows, MDP is a "request-reply" based protocol: to download the model, the ED first sends a request (or *req* for short) to the FLS; on getting a *req*, the FLS replies with the

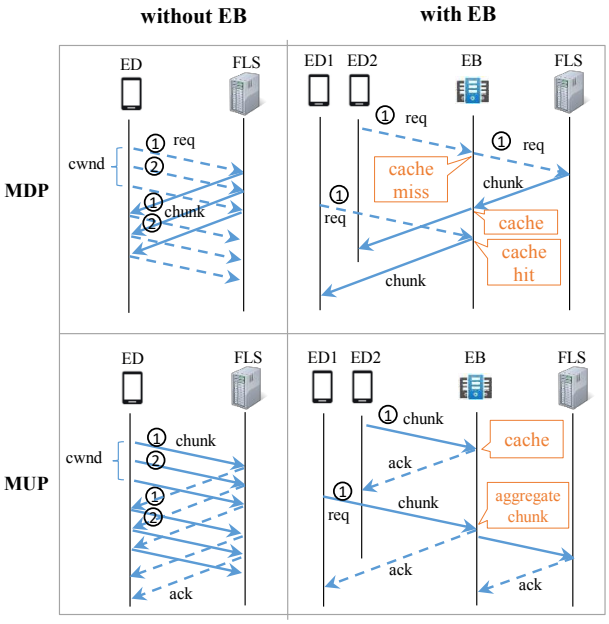


Fig. 3: The workflows of MDP and MUP.

desired chunk(s) of the model value. Like the reliable design of TCP, in case a sent request is considered as lost, the ED would resend; to make efficient use of the available bandwidth, the ED also maintains an *Additive Increase Multiplicative Decrease (AIMD)* congestion window (i.e., *cwnd*) like that of TCP [17] to control the number of in-flight requests.

Basically, compared with TCP, besides relying on UDP rather than raw IP, MDP mainly has four differences, making it outstanding for the download of chunked model values. First, as signal and management messages in INP are delivered via the separated *control channels*, MDP does not require handshakes to establish or close *data channel* connections for chunks. Second, for each download task, MDP only needs a one-way, rather than bidirectional, connection. Third, a download task completes only when all its involved values are fetched, thus MDP does not require severe in-order delivery; such a characteristic can be employed to optimize the involved congestion controls. As for the loss of packets, like the design of PUSH protocol [18], EDs would treat an MDP request or reply as lost, on receiving the replies of AHEADREPLYNUM (e.g., 3) requests sent after it. Fourth and most important, MDP is designed to leverage EBs on the way as cache nodes, so that multiple download requests of the same chunk from nearby EDs can be accelerated by their shared EB.

The workflow of EB-assisted MDP is as the upper right of Figure 3 shows. On receiving a request, the EB checks whether it holds the desired chunk; if so (i.e., cache hit), it replies directly, otherwise (i.e., cache miss), forwards the request to the FLS, and caches the corresponding reply when it goes by. Essentially, the EB works as a transparent cache proxy for MDP; thus, it is the duty of MDP to deal with the loss of packets. In practice, to achieve high performance, the cache needed by MDP is generally implemented in memories, which

are with limited volume sizes. For the management of cache, the well-known algorithm of *Time-aware Least Recently Used (TLRU)* is a promising candidate.

Note that, the EB would also work as the aggregator for gradient uploads as §IV-B will show. To know the proportion of EDs whose gradient uploads have been aggregated, the EB would count and record the number of EDs that have downloaded each model in each round as well.

B. MUP

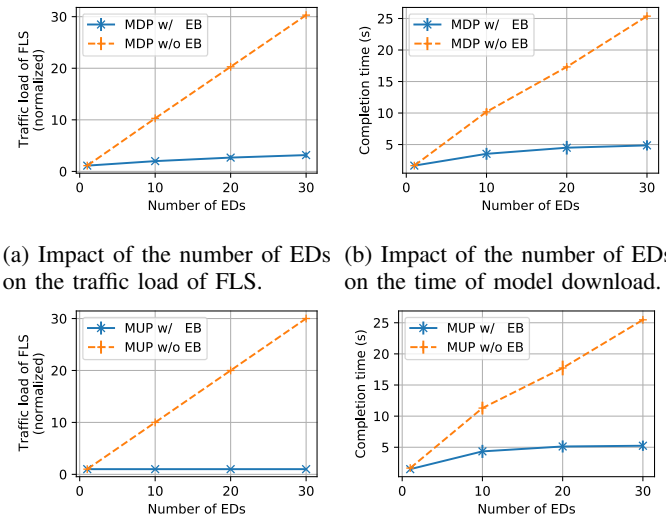
Very similar to the design of MDP, MUP is a "send-ack" based protocol designed for the upload of gradients as Figure 3 shows. When local gradients are ready, the ED should split gradients into chunks; then, each chunk, together with its index (for serialization) and weight (for weighted aggregation, with the initial value of 1) will be packed as the payload of a UDP datagram then sent to the FLS. On getting a gradient chunk, the FLS sends an acknowledgment packet (or *ack* for short) immediately. Also, by measuring the arrival of *ack*s, like the design of MDP, the ED adjusts its *cwnd*, learns the possible loss of data packets and resends.

As Figure 3 shows, when there exists an EB in the path, it would work as an aggregator. More specifically, on receiving a piece of gradient chunk, besides generating the acknowledgment, this EB would cache it, or update the cached weighted average value of the gradients, in which the value of the carried weight represents the total number of EDs this gradient chunk is computed from. Once the EB has received most or all the possible gradients that would go through this EB, or the time starting from the caching/creation of this gradient chunk exceeds a pre-defined timeout threshold, the EB would act as a specified ED and sends the aggregated gradient values, along with the updated weight value, to the final FLS. By default, the value of the weight represents the number of EDs that the aggregated gradient chunk is computed from. In case an aggregated chunk does not get acknowledged, the EB would conduct retransmissions just like an ED.

Recall that, the EB also maintains these unsent and unacknowledged aggregated chunks in the memory. When the memory runs out, like the case of MDP, the EB will proactively evict some unreported gradient chunks from the memory and send them to the FLS, in the *First-In-First-Out (FIFO)* manner. In the worst case, due to the delay or loss of acknowledgment, there might be no chunk that could be evicted. Then, the EB can just let incoming MUP packets bypass the aggregation.

V. Performance Study

Basically, the direct and significant benefit of INP is the ability to reduce the involved traffic load on FLS for both the download of models and upload of gradients with in-network cache and aggregation at the edge, thus reducing the time costs and supporting more numbers of EDs in turn. Now, let's verify the above advantages, by comparing the performance of EB-assisted INP with that of EB-disabled INP. In short, packet-level detailed simulations imply that, for an FL task involving m EDs, INP equipped with MDP and MUP is able to reduce



(a) Impact of the number of EDs on the traffic load of FLS.

(b) Impact of the number of EDs on the time of model download.

(c) Impact of the number of EDs on the traffic load of the FLS.

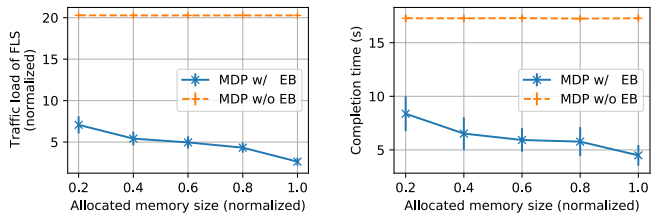
(d) Impact of the number of EDs on the time of gradient upload.

Fig. 4: INP greatly reduces the traffic load (in terms of the total volume) of FLS and the time needed by EDs for both the model download and gradient uploads, nearly from the magnitudes of $O(m)$ to $O(1)$, for a task involving m EDs.

both the traffic load of FLS and the time needed by model download and gradient upload up to m times.

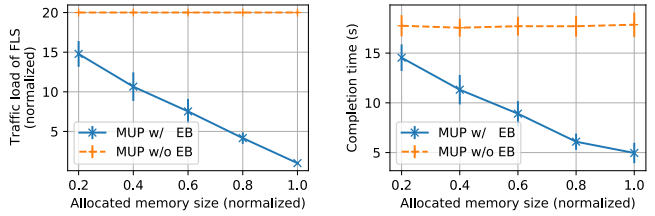
Methodology. Consider a simplified case in which multiple EDs are training a model with the size of 10MB through 100Mbps connections jointly. We assume that INP splits this model into chunks with the size of 1KB; and if enabled, the EB at the edge can perform in-network cache and aggregation for the passing model download and gradient upload requests, simultaneously. The arrival time of each download or upload request is generated according to $\min(X, 5)$, where X follows the exponential distribution of $Exp(1)$. To study the performance of INP under various numbers of participant EDs and allocated in-network processing memory sizes, we implement an ns3-alike discrete event-driven simulator in Python 3 based on [18], which precisely simulates the behavior of INP with and without the EB, respectively. By default, there are 20 EDs and the EB has sufficient memory for both the cache and aggregation. For each parameter setting, we conduct 40 trials to compute their mean values and standard deviations.

Case study. As Figure 4a shows, provided the EB has sufficient memory for cache, while the traffic load of FLS without the assistance of EB grows linearly with the increase in the number of involved EDs, the load only increases a little by using the cache service provided by EB. Basically, the slightly increased traffic amount is reasonable respecting the current design of INP: when multiple EDs are downloading the same model in a very short time before the fetched model chunk arrives, EB simply forwards all these download requests to the FLS, even if they are querying the same



(a) Impact of the cache size on the traffic load of FLS.

(b) Impact of the cache size on the time of model download.



(c) Impact of the cache size on the traffic load of FLS.

(d) Impact of the cache size on the time of gradient upload.

Fig. 5: Compared with EB-assisted model downloads (MDP), the performances of EB-assisted gradient uploads (MUP) are more sensitive to the size of allocated cache memory.

chunk. Such increased traffic can be reduced with advanced optimizations like hanging duplicated requests up, which are left as future work. Regarding the in-network aggregation of gradient uploads, as Figure 4c shows, with aggregation, the traffic load of FLS remains unchanged. Notably, it is obvious from Figures 4b and 4d that for both model download and gradient upload, the completion times under the assistance of EB grow with the number of EDs, and keep consistent after exceeding 5s. This is mainly due to the test settings: basically, a download is considered as completed when all EDs have held the model, and an upload is treated as completed when the FLS has received all ED gradients, whereas the arrival time of each download and upload request is $\min(X, 5)$ seconds and X follows $Exp(1)$. All these results indicate that EB helps MDP and MUP achieve near-optimal performance on the download of model and upload of gradients.

Impact of cache size. Note that, both the cache and aggregation needed by INP would occupy the memory of EB for chunk caching. Hence, we also want to know how the performance would change in case the EB runs out of memory. Figure 5 shows the impacts of allocated memory volumes, normalized by the model size of 10MB, on the performance of INP. Without surprise, for both the model download and gradient upload, fewer memory sizes yield worse performances, in terms of both the traffic volume of FLS and completion time. And interestingly, compared with gradient uploads, the performance of model downloads is less sensitive to the size of allocated memory. For instance, in the studied case, for model downloading, just 2MB cache memory can eliminate more than 60% of FLS's traffic, saving about 50% of the time, while the reduced traffic volume and time cost for gradient upload are less than 30% and 20%, respectively. Currently, EBs manage their memories for model cache and gradient aggregation following the simple TLRU and FIFO strategies, respectively. There is room for optimization and we leave the design of advanced algorithms as future work.

VI. Conclusions

In conclusion, we design INP, an In-Network Processing framework, along with the novel Model Download Protocol (MDP) and Model Upload Protocol (MUP), to accelerate the data deliveries involved in large-scale cross-device FL systems. The key of INP is to let *Edge Boxes* (EBs) residing between the selected training *End Devices (EDs)* and the *FL Server (FLS)* act as the cache node for model downloads and aggregator for gradient uploads. A packet-level case study of the performance indicates that INP could successfully reduce both the traffic load of FLS and the needed times of model downloads and gradient uploads, for a training task involving m EDs, from the magnitudes of $O(m)$ to $O(1)$ at most.

References

[1] K. Bonawitz, H. Eichner, W. Grieskamp *et al.*, "Towards federated learning at scale: System design," *Proceedings of Machine Learning and Systems*, vol. 1, pp. 374–388, 2019.

[2] C. Niu, F. Wu, S. Tang *et al.*, "Billion-scale federated learning on mobile clients: A submodel design with tunable privacy," in *26th MobiCom*. ACM, 2020.

[3] Y. Shi, K. Yang, T. Jiang *et al.*, "Communication-efficient edge ai: Algorithms and systems," *IEEE Commun. Surveys Tuts.*, vol. 22, no. 4, pp. 2167–2191, 2020.

[4] Y. Zhan, P. Li, Z. Qu *et al.*, "A learning-based incentive mechanism for federated learning," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 6360–6368, 2020.

[5] P. Kairouz, H. B. McMahan, B. Avent *et al.*, "Advances and open problems in federated learning," *CoRR*, vol. abs/1912.04977, 2019.

[6] L. Liu, J. Zhang, S. H. Song *et al.*, "Client-edge-cloud hierarchical federated learning," in *IEEE ICC*, 2020.

[7] R. Stoenescu, V. Olteanu, M. Popovici *et al.*, "In-net: In-network processing for the masses," in *10th EuroSys*, 2015.

[8] A. Panda, S. Han, K. Jang *et al.*, "Netbricks: Taking the v out of NFV," in *12th USENIX OSDI*, Nov. 2016, pp. 203–216.

[9] Y.-Y. Shih, H.-P. Lin, A.-C. Pang *et al.*, "An nfv-based service framework for iot applications in edge computing environments," *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 4, pp. 1419–1434, 2019.

[10] P. Jin, X. Fei, Q. Zhang *et al.*, "Latency-aware vnf chain deployment with efficient resource reuse at network edge," in *IEEE INFOCOM*, 2020, pp. 267–276.

[11] L. Zhang, A. Afanasyev, J. Burke *et al.*, "Named data networking," *SIGCOMM Comput. Commun. Rev.*, vol. 44, no. 3, pp. 66–73, Jul. 2014.

[12] C. Lao, Y. Le, K. Mahajan *et al.*, "ATP: In-network aggregation for multi-tenant learning," in *18th USENIX NSDI*, Apr. 2021.

[13] A. Sapio, M. Canini, C.-Y. Ho *et al.*, "Scaling distributed machine learning with in-network aggregation," in *18th USENIX NSDI*, Apr. 2021, pp. 785–808.

[14] N. Gebara, M. Ghobadi, and P. Costa, "In-network aggregation for shared machine learning clusters," in *Proceedings of Machine Learning and Systems*, vol. 3, 2021, pp. 829–844.

[15] S. Luo, P. Fan, K. Li *et al.*, "Fast parameter synchronization for distributed learning with selective multicast," in *IEEE ICC*, 2022.

[16] S. Luo, H. Yu, K. Li *et al.*, "Efficient file dissemination in data center networks with priority-based adaptive multicast," *IEEE J. Sel. Areas Commun.*, vol. 38, no. 6, pp. 1161–1175, 2020.

[17] M. Polese, F. Chiariotti, E. Bonetto *et al.*, "A survey on recent advances in transport layer protocols," *IEEE Commun. Surveys Tuts.*, vol. 21, no. 4, pp. 3584–3608, 2019.

[18] S. Luo, T. Ma, W. Shan *et al.*, "Efficient multisource data delivery in edge cloud with rateless parallel push," *IEEE Internet of Things Journal*, vol. 7, no. 10, pp. 10 495–10 510, 2020.