Totoro: A Scalable Federated Learning Engine for the Edge

Cheng-Wei Ching
University of California, Santa Cruz
Santa Cruz, CA, USA
cching1@ucsc.edu

Xin Chen
Georgia Institute of Technology
Atlanta, GA, USA
xchen384@gatech.edu

Taehwan Kim
Virginia Tech
Blacksburg, VA, USA
tk020@vt.edu

Bo Ji
Virginia Tech
Blacksburg, VA, USA
boji@vt.edu

Qingyang Wang
Louisiana State University
Baton Rouge, LA, USA
qwang26@lsu.edu

Dilma Da Silva
Texas A&M University
College Station, TX, USA
dilma@cse.tamu.edu

Liting Hu*
University of California, Santa Cruz and
Virginia Tech
Santa Cruz, CA, USA
liting@ucsc.edu

Abstract

Federated Learning (FL) is an emerging distributed machine learning (ML) technique that enables in-situ model training and inference on decentralized edge devices. We propose Totoro, a novel scalable FL engine, that enables massive FL applications to run simultaneously on edge networks. The key insight is to explore a distributed hash table (DHT)-based peer-to-peer (P2P) model to re-architect the centralized FL system design into a fully decentralized one. In contrast to previous studies where many FL applications shared one centralized parameter server, Totoro assigns a dedicated parameter server to each individual application. Any edge node can act as any application's coordinator, aggregator, client selector, worker (participant device), or any combination of the above, thereby radically improving scalability and adaptivity. Totoro introduces three innovations to realize its design: a locality-aware P2P multi-ring structure, a publish/subscribe-based forest abstraction, and a bandit-based exploitation-exploration path planning model. Real-world experiments on 500 Amazon EC2 servers show that Totoro scales gracefully with the number of FL applications and N edge nodes, speeds up the total training time by $1.2\times - 14.0\times$, achieves $O(logN)$ hops for model dissemination and gradient aggregation with millions of nodes, and efficiently adapts to the practical edge networks and churns.

*Corresponding author: Liting Hu, Computer Science and Engineering, University of California, Santa Cruz.

EuroSys '24, April 22–25, 2024, Athens, Greece

ACM ISBN 979-8-4007-0437-6/24/04.
https://doi.org/10.1145/3627703.3629575

CCS Concepts: • **Computer systems organization** → **Distributed architectures**; • **Computing methodologies** → **Machine learning**.

Keywords: Distributed and parallel systems for machine learning, federated learning, edge computing.

ACM Reference Format:
Cheng-Wei Ching, Xin Chen, Taehwan Kim, Bo Ji, Qingyang Wang, Dilma Da Silva, and Liting Hu. 2024. Totoro: A Scalable Federated Learning Engine for the Edge. In *European Conference on Computer Systems (EuroSys '24), April 22–25, 2024, Athens, Greece.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3627703.3629575

1 Introduction

With the emergence of 5G networks and the proliferation of connected devices, federated learning (FL) enables data processing and machine learning to take place at the edge of the network. This means that data can be processed locally on edge devices, without the need for centralized servers or data centers, reducing the amount of data that needs to be transmitted and improving privacy. FL has been used in many application domains, including predicting human activities [37, 38], learning sentiment [84], language processing [47, 61, 100], and enterprise infrastructures [66].

The problem. We consider a typical edge computing architecture where millions of edge devices (e.g., smart wearables, self-driving car sensors) are interconnected with the remote cloud via the edge layer. The edge layer has hundreds of thousands of server-grade machines, gateways, and routers, referred to as "edge nodes", maintained by different edge providers across geo-distributed sites. The raw data is collected and stored at these edge nodes, and a machine learning model is trained from the distributed data without sending the raw data from the nodes to a central place.

There has been a rise of FL tools and frameworks, such as Tensorflow Federated [15], LEAF [28], PaddleFL [8] and

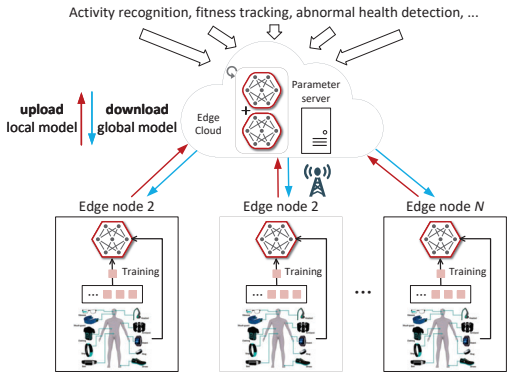


Figure 1. Edge federated learning use cases.

PySyft [11]. However, we face significant challenges in building an efficient FL engine on practical edge networks due to the edge dynamics and the high scalability requirements of emerging FL applications.

Figure 1 depicts a use case scenario that highlights these challenges. In future Smart Health systems, wearable devices such as smart watches, fitness trackers, and blood pressure meters are equipped with wide-area network access. They continuously collect physical and behavioral data such as heart rate, blood pressure, and human activity from numerous users. On the back end, many FL applications concurrently run on these devices or edge nodes, performing in-situ training using the collected sensor data. Examples of these FL applications include activity recognition application, which predicts human activities to prevent injuries and falls [39]; fitness tracking application, which analyzes calories burned during exercises [98]; and abnormal health detection application, which detects abnormal health conditions like depression, stroke, and asthma and provides timely intervention [105].

The first challenge is *scalability*: *how to scale gracefully to support a vast number of diverse FL applications that simultaneously run on edge networks?* According to Gartner Research, the edge applications market is expected to represent a 33-billion-dollar opportunity in 2025 [2]. As emerging edge applications and edge devices grow in quantity and complexity, the number of FL applications submitted to the edge will likely become *huge*. As shown in Figure 1, different FL applications may require training of various FL models for different user profiles (e.g., age, gender, weight), medical conditions, and environmental factors (e.g., indoor, outdoor, high altitude) simultaneously based on the same raw data. This results in the generation of a vast number of FL tasks. However, state-of-the-art FL systems [33, 34, 55, 82, 106] mostly employ a centralized architecture, where one or multiple central parameter servers handle all applications' activities such as model dissemination, round setup, participant selection, round management, and aggregation update. While this centralized architecture scales well in datacenters, it poses scalability bottlenecks in edge systems that have millions of nodes and numerous concurrently running FL applications that require diverse FL policies.

The second challenge is *adaptivity*: *how to achieve good training and testing performance for massive FL applications on practical edge networks?* To scale with massive FL applications, what cloud administrators usually do is to partition all nodes into many sets and assign a parameter server per each set of nodes in a static manner (e.g., one parameter server per rack). This assignment may work well for datacenters that have a global view of the system (e.g., the availability of resources in each machine). However, this approach may not quickly adapt to the edge platforms, characterized by millions of resource-constrained nodes, unreliable network links and random access protocols (e.g. in wireless networks), client mobility (e.g., in mobile ad-hoc networks), and workload surges in arbitrary geographical (and dynamic) edge locations.

Our solution: Totoro. We propose Totoro, a novel scalable federated learning (FL) engine, that enables massive FL applications to run simultaneously on edge networks. Instead of modifying today's FL systems, we explore a different angle in the design space: we propose a fully decentralized architecture. In contrast to previous studies where many FL applications shared one or multiple centralized parameter servers, Totoro assigns a dedicated parameter server to each individual application. By doing so, it avoids overloading any single edge node. Any edge node can act as any application's coordinator, aggregator, client selector, worker (participant device), or any combination of the above, thereby radically improving scalability and adaptivity.

The key insight is to explore a distributed hash table (DHT)-based peer-to-peer (P2P) model to re-architect FL systems. The P2P model is predominantly used in file-sharing applications (e.g., BitTorrent [1], Storj [13], Freenet [3]), peer-assisted CDNs [108], and blockchain [40]. In the P2P model, each node is equal to the others, having the same rights and duties. There is no central server. All nodes work collaboratively to accomplish a task or deliver a service. For example, in BitTorrent [1], a user can download a file from many other users that already have the file. At the same time, the user also uploads the file for many others who ask for it. Similar to BitTorrent, where all peers collaboratively undertake the task of file sharing, we enable all edge nodes to collaboratively undertake the task of FL training and testing for a massive number of applications simultaneously.

Totoro introduces three innovations to realize its fully decentralized design: *a locality-aware P2P multi-ring structure*, *a publish/subscribe-based forest abstraction*, and *a bandit-based path planning model*.

First, all edge nodes are self-organized into a scalable DHT-based P2P overlay. Totoro's *locality-aware P2P multi-ring*

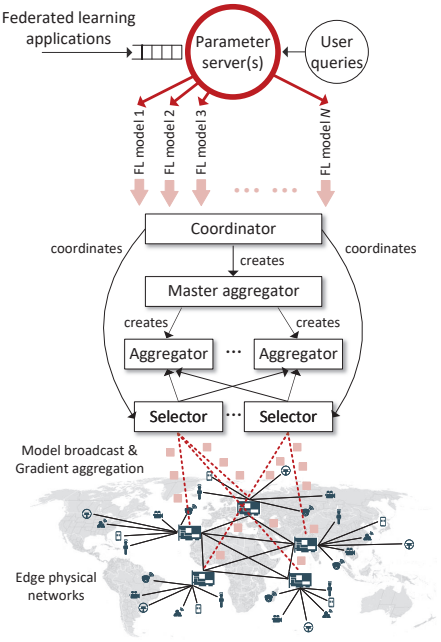


Figure 2. The data pipeline in a typical FL framework.

structure ensures edge administrative isolation, enabling efficient local processing of FL tasks and preventing private control flows from entering other edge sites.

Second, built on top of the locality-aware P2P multi-ring structure, Totoro introduces *a publish/subscribe-based forest abstraction* for model broadcast and gradient aggregation. It decouples the prevailing centralized architecture of FL systems into a "fully" decentralized architecture, where each FL application can be assigned an independent dataflow tree that operates with maximum independence, thus radically improving the scalability of FL systems.

Third, since Totoro runs in a highly dynamic distributed environment in which link disconnections and stragglers are inevitable, model broadcast and gradient aggregation may experience significant delays. Totoro introduces *a bandit-based exploitation-exploration path planning model* that can dynamically replan the data transfer paths to adapt to the edge dynamics.

Summary of results. We implement Totoro on top of the open-source Pastry DHT [9] and Keras [5] software stacks. We evaluate Totoro across various FL tasks with real-world datasets on 500 Amazon EC2 servers. Compared to state-of-the-art FL systems [7, 53], Totoro scales more gracefully with the number of concurrently running FL applications and edge nodes, and dramatically improves scalability and load balance by distributing many masters fairly across large-scale edge networks without overburdening any single node. When performing concurrently running FL applications' tasks, Totoro speeds up the total training time by 1.2×−14.0×, achieves $O(logN)$ hops for model dissemination and gradient aggregation with millions of nodes (N), and efficiently adapts to the unreliable edge networks and churn.

Contributions. This paper contributes the following:

- *Problem:* We study the software architecture of existing FL systems and discuss the limitations of applying FL to practical edge networks.
- *Key idea:* Totoro explores the DHT-based P2P model to propose a novel, fully decentralized "many masters/many workers" architecture that radically improves scalability and adaptivity. To the best of our knowledge, we are the *first* to propose a fully decentralized FL engine for edge networks.
- *Totoro design & implementation:* Totoro introduces three innovations to realize its design: a locality-aware P2P multi-ring structure, a publish/subscribe-based forest abstraction, and a bandit-based path planning model.
- *Results:* Evaluation shows Totoro's significant scalability and adaptivity gains over the state-of-the-art.

2 Background and Challenges

We start with a quick primer of the software architecture used in state-of-the-art federated learning (FL) frameworks (§2.1). Next, we highlight the challenges that we face when applying FL to real-world edge networks (§2.2).

2.1 Federated learning system architecture

State-of-the-art FL frameworks (e.g., Meta's PAPAYA [50], LinkedIn's FLINT [93], Google's federated learning framework [26], IBM Federated Learning framework [66], and Apple's federated task processing system [73]) commonly adopt a centralized or hierarchical "single master/many workers" architecture. In this architecture, the master is typically deployed on a parameter server, acting as a coordinating service provider without data. The workers, on the other hand, connect to numerous edge devices and serve as both the data owners and beneficiaries of federated learning.

Figure 2 illustrates the data pipeline between these components in a typical FL framework.

The high-level design involves two parts: the parameter server that runs the "master", and the end-user devices that run the "workers". The parameter server comprises three main components: Coordinator, Selector, and Aggregator. While the number of Selectors and Aggregators can scale elastically based on the workload demand, there is only one Coordinator. We summarize the functions of these components as follows.

Coordinator. The Coordinator performs three main functions:

- *Task Assignment.* Whenever a new FL application is submitted to the system, the Coordinator assigns the application to a single Aggregator based on the workload among Aggregators and the estimated workload of the application such as application concurrency and model size.

- *Client Assignment.* For each available client, the Coordinator constructs a list of eligible applications. The Coordinator assigns each client to an application.
- *Task Migration.* The Coordinator moves applications between Aggregators when it detects failed or overloaded Aggregators.

Aggregator. The Aggregators are *persistent and stateful* to avoid a substantial cold start overhead for a new application. Each Aggregator is responsible for one single FL application and carries out three main responsibilities.

- *Gradient Aggregation.* Once a client completes training, it uploads the trained serialized gradient update to the server. This update is then pushed into an in-memory queue on the Aggregator, which aggregates client gradient updates to produce new versions of the server model of an application.
- *Client Guidance.* The Aggregator guides clients into running the client protocol, such as downloading, uploading, and training configurations, by responding clients requests from the Selector.
- *Client Tracking.* The Aggregator is responsible for tracking (i) if clients are satisfied with their assigned applications, and (ii) if clients are still eligible for their assigned applications. The tracking information will be used by the Coordinator to run client assignments.

Selector. The Selector is the only component that directly communicates with clients and plays two roles:

- *Task Advertisement.* The clients check in with the Selector and report their eligibility for available applications. The Selector summarizes client availability for the Coordinator.
- *Request Forwarding.* When a client is assigned an eligible application. The Selector forwards the client to the Aggregator responsible for that application. The Selector also routes clients' requests to the corresponding Aggregator, such as model broadcasting, and reporting client status and gradient updates.

2.2 Challenges

2.2.1 Challenge #1: Scaling gracefully with the number of diverse FL applications and edge nodes.

Key considerations in addressing this challenge include:

1. Distributed task management. As shown in Figure 2, existing production FL systems predominantly rely on one single instance of the Coordinator to instruct Aggregators and Selectors to handle all FL applications' training and testing activities. While this hierarchical architecture scales well in datacenter platforms, it may encounter scalability issues in edge systems characterized by millions of edge nodes and numerous concurrently running FL applications. What makes it more challenging is that the edge network comprises several edge providers, each administering a disjoint set of edge nodes, and thus, there is a critical lack of a global view of the workloads and resources. The lack of a global view makes it difficult for existing production FL systems to spread FL tasks evenly across edge nodes. This makes it challenging to scale gracefully with new edge nodes and new coming FL applications, leading to prolonged total training time.

2. Application-specific customization. With the emergence of new use-case scenarios, FL applications are becoming increasingly diverse. This calls for flexible designs of participant selection algorithms [33, 55, 71], compression techniques [18, 20, 24, 35], and communication protocols (e.g., synchronous [69], semi-synchronous [86], or asynchronous [99]) to cater to these varying needs. Unfortunately, existing FL systems often share the same parameter server among many applications and restrict FL policies to a fixed one, limiting the system's ability to accommodate diverse FL policies. To support application-specific customization, we have to rely on multiple different FL frameworks, at the expense of efficiency, maintainability, and simplicity.

2.2.2 Challenge #2: Adapt to practical edge network conditions such as varying bandwidths, unreliable links, high churn, and workload surges.

The second challenge arises from the first one. To scale with massive FL applications, what the cloud administrators usually do is to partition all nodes into many sets and assign a parameter server per each set of nodes in a static manner (e.g., one parameter server per rack). While this assignment approach may work well in datacenters, it lacks the agility to adapt quickly to edge platforms that have millions of resource-constrained nodes.

This is because the edge environment imposes unique difficulties: (1) Edge network link delays are unpredictable and vary stochastically due to unreliable links and random access protocols (e.g., in wireless networks), client mobility (e.g., in mobile ad-hoc networks), and randomness of demand (e.g., workload surges) in arbitrary (and dynamic) geographical edge locations, and (2) Edge nodes fail or lag unexpectedly (e.g., due to signal attenuation, interference, and wireless channel contention). However, unlike datacenter servers, edge nodes have limited computing resources (few-core processors, little memory, and little permanent storage [21]) and they have no backpressure. As such, there is little room to adapt to the edge dynamics or handle stragglers by over-provisioning resources or replicating links like previous studies. Efforts should be made to dynamically replan the data transfer paths to adapt to the edge dynamics.

3 Related Work

This section discusses existing FL system designs and their shortcomings for deployment in edge platforms (Table 1).

Centralized FL systems. The centralized server-client architecture is widely used in state-of-the-art FL systems such

Federated learning systems	Architecture	Central parameter server	Computing environment	Master/worker communication structure	Type of learned model	Type of adversarial model	Scale to massive diverse applications	Adapt to edge networks
FedAT [34], TiFL [33], Oort [55]	Centralized	✓	Datacenter	Hub-and-spoke	Global consensus	Non-Byzantine	✗	✗
Client-edge-cloud [64], Edge-DemLearn [72]	Hierarchical	✓	Edge	Hub-and-spoke	Global consensus	Non-Byzantine	✗	✗
BDSGD [89], Pu et al. [74]	Distributed	✓	Datacenter	Hub-and-spoke	Global consensus	Non-Byzantine	✗	✗
Sol [54], Gaia [49], WeightGrad [19]	Distributed	✓	Cloud	Hub-and-spoke	Global consensus	Non-Byzantine	✗	✗
D^2 [88], BrainTorrent [81], Lalitha et al. [58]	Decentralized	✗	Datacenter	P2P (one-hop neighbor)	Global consensus	Non-Byzantine	✗	✗
ByRDiE [101], BRIDGE [41], Gupta et al. [45]	Decentralized	✗	Datacenter	P2P (one-hop neighbor)	Global consensus	Byzantine	✗	✗
Bellet et al. [23], Vanhaesebrouck et al. [91]	Decentralized	✗	Datacenter	P2P (one-hop neighbor)	Personalized	Non-Byzantine	✗	✗
Totoro	Decentralized	✗	Edge	Dynamic-structured tree	Global consensus	Non-Byzantine	✓	✓

Table 1. Overview of state-of-the-art federated learning system designs compared to Totoro.

as Oort [55], FedAT [34], TiFL [33], and FEDRECON [82]. In these systems, a powerful centralized parameter server and sufficient node-to-server bandwidth are provided to maintain the communications between the parameter server and clients. These systems focus on optimizing participant selection strategies [33, 34, 55], improving communication efficiency [46, 62, 69, 78], taking data heterogeneity [60, 77], or ensuring privacy [43, 44, 70]. For example, many algorithms have been proposed to reduce the communication overhead concentrated on the central server by reducing communication rounds with local updates allowance [63, 94, 102], employing compression techniques to reduce the transmitted bits [22, 79, 92], and sampling a subset of clients [33, 34, 55]. While these systems work well in resource-rich datacenters, the centralized control plane with a static assignment of parameter servers may not adapt well to resource-constrained edge settings with millions of nodes, numerous FL applications, and unreliable network links.

Hierarchical FL systems. In hierarchical FL systems, an intermediate layer (e.g., edge servers) is inserted between the central server and client devices to perform partial aggregations on distributed local models before the global aggregation. The intermediate layer helps mitigate the non-independent and identically distributed (non-IID) effects of client data [72] and reduce the communication burden on the central server [17, 64, 65]. For example, Abad et al. [17] employ small cell-base stations to orchestrate federated learning among mobile users and periodically exchange model updates with the macro-base station for global model learning. Although this structure enhances scalability with additional aggregators, the aggregators can become points of failure, causing disconnection between the central server and client devices and interrupting the training process.

Peer-to-peer FL (P2PFL) systems. P2PFL is a distributed learning protocol without a central parameter server. The key idea is to leverage P2P communication between individual clients for exchanging model updates. Model aggregation and updates are performed locally on client nodes with the acquired gradients from their one-hop neighbors. Previous P2PFL research considered design from different angles: adversarial models (non-Byzantine vs Byzantine settings) and learned models (global consensus vs personalized). Non-Byzantine algorithms [58, 81, 88] focus on improving convergence speed. Conversely, Byzantine algorithms [41, 45, 101] aim to ensure that the trained model remains close to the model learned without adversaries. Global consensus learning [41, 45, 58, 81, 88, 101] aims to generate a single global model at the end of training, while personalized learning [23, 91] allows each peer to train a unique model. However, these efforts mostly focus on novel algorithm development and assume idealized datacenter environments, neglecting real-world edge dynamics such as varying bandwidths, unreliable links, and high churn. Totoro backs up the P2PFL work with cross-layer system support to implement these protocols in practical edge networks. Universal ring [32] provides services to form separate overlays but it can not ensure geographic diversity and administration isolation. SplitStream [31] proposes to split the content into k stripes and to multicast each stripe using a separate tree in order to load balance forward traffic over P2P nodes. However, when a gradient update from a worker is split into several stripes and sent via separate trees, the intermediate nodes cannot run intermediate gradient aggregation, and the whole aggregation overhead would overwhelm the master.

Geo-distributed ML systems. Federated data analytics has been a hot topic of interest in geo-distributed storage [90] and data processing systems [54, 103] that attempt to reduce latency (e.g., Sol [54]), save bandwidth (e.g., WeightGrad [19], Gaia [49]) or optimize resource usage [51, 95]. These systems, however, mostly inherit MapReduce's "single master/many workers" architecture that relies on a monolithic scheduler for scheduling all ML tasks and thus may not scale to the edge dynamics. For example, Sol is built on top of the Spark stream processing engine [12] which has only one master. These projects complement Totoro by offering resource optimization and incentive mechanisms, whereas Totoro provides them with scalable system support.

4 Totoro design

Totoro proposes a novel scalable FL engine for practical edge networks. To achieve this, Totoro incorporates several key techniques and components, which we describe in detail.

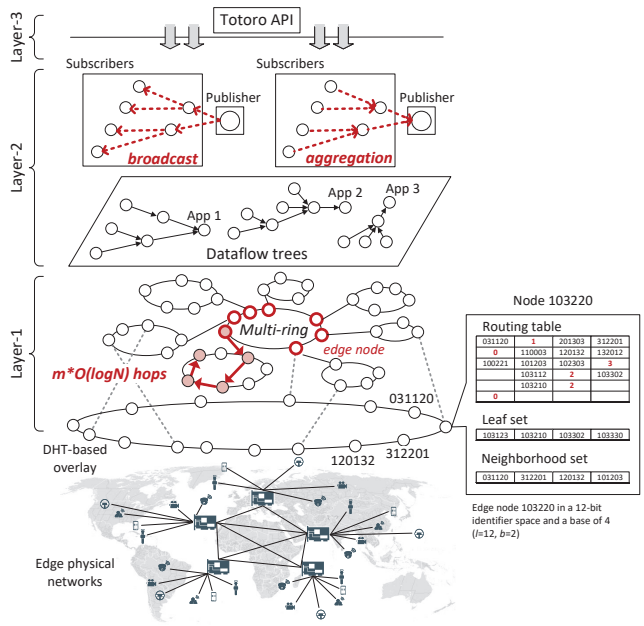


Figure 3. The Totoro system overview.

4.1 Overview

Totoro's system goals are:

- *Scalability.* Totoro can scale to process a vast number of FL applications' tasks simultaneously on millions of edge nodes without introducing any centralized bottleneck.
- *Adaptivity.* Totoro can quickly adapt to the practical edge networks characterized by varying bandwidths, unreliable links, high churn (nodes join and leave), and workload surges in arbitrary geographical edge locations.
- *Good FL performance.* When handling a vast number of FL applications, Totoro can speed up the training process for each of them.

As shown in Figure 3, Totoro has three layers: a locality-aware P2P multi-ring structure, a publish/subscribe-based forest abstraction, and a high-level API.

Layer 1: locality-aware P2P multi-ring structure. All distributed edge nodes are self-organized into a DHT-based P2P overlay. Each node has a unique 128-bit NodeId in a very large circular NodeId space. In an edge network with N nodes, the DHT-based P2P overlay guarantees that, no matter where the source node is, any FL data (e.g., model or gradient) can be routed to any destination node within $O(logN)$ hops. Compared to existing DHTs studies [80, 85, 107], our innovations are: (1) Totoro divides the original single P2P ring structure into many smaller, more manageable locality-aware P2P multi-ring structures that enable locality-aware FL processing; and (2) Totoro designs a new boundary-aware two-level routing table that ensures administrative isolation for privacy concerns.

Layer 2: publish/subscribe-based forest abstraction. Built upon Layer 1's locality-ware P2P multi-ring structure, Totoro introduces a new publish/subscribe-based forest abstraction that manages a vast number of FL applications in a scalable manner. Each FL application is assigned a dynamically-structured dataflow tree that operates with maximum independence, responsible for disseminating the model from the master to the workers and aggregating the gradients from the workers to the master. These trees together form a "forest". Our innovations are (1) a fully decentralized architecture—unlike other federated learning systems, our system does not have a static assignment of parameter servers. Instead, any edge node can be automatically promoted as a parameter server (master) when workload surges, which significantly improves load balancing and scalability. (2) DHT-based routing—the time complexity of model propagation and gradient aggregation is limited to $O(logN)$ hops.

Layer 3: high-level API. We provide a high-level API to abstract away the complexities of P2P overlay construction, dynamic-structured dataflow tree construction, model dissemination, and gradient aggregation. Totoro supports application-specific customization, allowing application owners to set their own FL policies.

4.2 Layer 1: locality-aware P2P multi-ring structure

Many times, geographic diversity or location matters for training FL applications. For example, a road traffic detection application may require nodes with varying weather condition information in different geographic locations. Training a model on a medical disease prevalent in a certain region may require information from a specific location.

Therefore, we organize distributed edge nodes into a locality-aware P2P multi-ring structure to enable locality-aware FL processing.

First, we organize distributed edge nodes into a DHT-based P2P ring overlay, which is similar to the BitTorrent nodes that use the Kademila DHT [68] for "trackerless" torrents. Each edge node is assigned a unique 128-*bit* NodeId in a very large circular NodeId space (e.g., $0 \sim 2^{128}$). NodeIds are used to identify edge nodes and route FL data (e.g., model or gradient) over large-scale edge networks.

To do that, each node needs to maintain three data structures: a routing table, a leaf set, and a neighborhood set.

- *Routing table* is used for routing FL data. It consists of node characteristics organized in rows by the length of the common prefix. The routing works based on prefix-based matching. Every node knows m other nodes in the ring and the distance of the nodes it knows increases exponentially. The routing jumps closer and closer to the destination, like a greedy algorithm, within $\lceil \log_{2^b} N - 1 \rceil$ hops, where $2^b - 1$ is the routing table's entry size.
- *Leaf set* is used for rebuilding the routing tables upon failures.

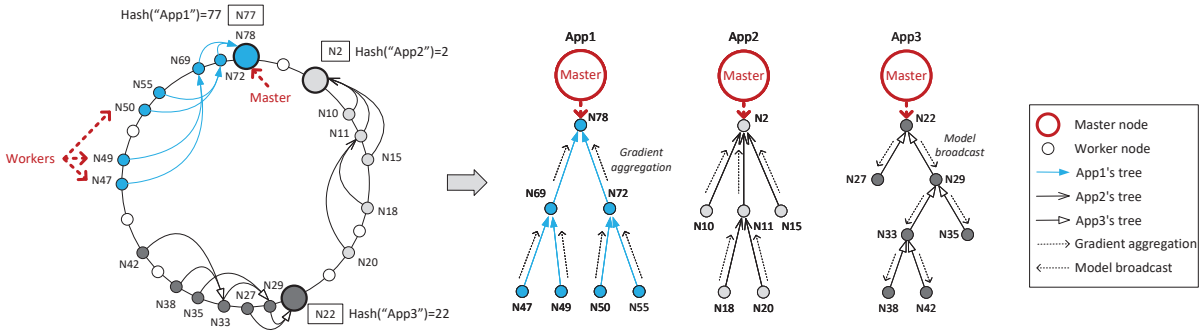


Figure 4. The workflow of Totoro's publish/subscribe-based "forest" abstraction.

- *Neighborhood set* contains a fixed number of nodes that are "physically" closest to that node for maintaining the locality properties.

Second, we divide the original large P2P ring into m smaller, more manageable, locality-aware rings, which we call "multi-rings", using Ratnasamy and Shenker's distributed binning algorithm [75] (m is a configurable parameter). Each ring is an "edge zone" and is characterized by a maximum desired network round-trip time (RTT), called *diameter*.

Third, we design a new routing table to enable administrative isolation. The challenge is, *how to achieve path convergence to enable administrative isolation, i.e., data paths from different nodes in an edge site should converge at a node in that edge site?* Existing DHTs [68, 76, 80, 85, 107] do not guarantee path convergence as those systems try to optimize the search path to reduce response latency. To route a packet to an arbitrary destination key, the packet will be routed to the destination node in another site as long as it has a longer NodeId prefix matching the key. To address this challenge, we make the following changes to existing routing tables: (1) each NodeId now has $(m+n)$-bit, where the m-bit prefix presents the zone Id and the n-bit suffix represents the NodeId within a zone. Let P denote the prefix of NodeId, i.e., $P_1 ... P_n$. Let S denote the suffix of NodeId, i.e., $S_1 ... S_n$. Then the NodeId equals $D = P * 2^n + S$; correspondingly (2) each node's routing table will have two levels: the level 1 routing table and the level 2 routing table. The i_{th} entry in the level 1 routing table with m entries at peer x equals to $(P_x + 2^{i-1}) \bmod 2^m * 2^n$. The i_{th} entry in the level 2 routing table with n entries at peer y equals to $(S_y + 2^{i-1}) \bmod 2^n$.

To achieve administration isolation, the administrator of an edge site can leverage the level 1 routing table to control the data flow among different edge zones. For example, when an FL application should be running only within an edge site, any packet generated by that FL application should only travel within this edge site. Administrators can check the destination of packets. If the packet's destination shares a different prefix with the administrator's zone Id, the administrator can block the packet before routing it outside the edge zone.

4.3 Layer 2: publish/subscribe-based forest abstraction

Built upon Layer 1, Totoro introduces a new publish/subscribe-based "forest" abstraction for managing a vast number of FL applications in a scalable manner. Specifically, our goal is to achieve a balanced distribution of masters for hundreds of thousands of FL applications, ensuring that they are not concentrated on a few overloaded nodes.

The key innovation is leveraging DHTs to decompose the FL system architecture from *1:n* to *m:n*, where each FL application can be assigned a dynamic-structured dataflow tree that operates with maximum independence, thereby radically improving load balance and scalability. A DHT is a hash table that partitions the key space and distributes the parts across a set of nodes, providing a lookup service similar to a hash table. The trick with DHTs is that the node that gets to store a particular key is found by hashing that key, so in effect, the hash-table buckets are now independent nodes in a network.

We utilize the fully decentralized nature of DHT to process FL applications' tasks at extreme scale: unlike other FL systems, Totoro does not have a static assignment of a parameter server. Instead, the parameter server is broken down into many components, such as the coordinator, client selector, and aggregator. Any edge node can act as any FL application's coordinator, aggregator, client selector, worker (participating edge device), or any combination of the above, thereby radically improving load balance and scalability.

Figure 4 lays out the construction of the publish/subscribe-based "forest". It has the following steps.

Built on top of Layer 1, all edge nodes are structured into a DHT-based P2P overlay. Here we use one ring as an example. The DHT-based P2P overlay guarantees that: *given a message and a key, no matter where the source node is, the message can be reliably routed to the node whose NodeId is numerically closest to that key, within* $\lceil \log_{2^b} N - 1 \rceil$ *hops, where* $2^b - 1$ *is the routing table's entry size.*

Join(IP, port, site) Edge node joins the DHT-based P2P overlay network.
CreateTree(app_id) Application owner creates a dynamic-structured dataflow tree and configures the parameters (e.g., fanout).
Subscribe(app_id) Edge node sends a `JOIN` message to subscribe to a dynamically-structured dataflow tree with the topic equal to app_id. Application owner can specify her client selection function in the API.
Broadcast(app_id, object) Application's master disseminates model to workers along its dynamically-structured dataflow tree. Application owner can specify her compression function in the API.
onBroadcast(app_id, object) Callback. Invoked when the worker receives any model or updates from the application's master.
Aggregate(app_id, object) Application's master aggregates updates from the workers to the root. Application owner can specify her aggregation function in the API.
onAggregate(app_id, object) Callback. Invoked when any internal node receives updates from a child node.
onTimer(app_id) Callback. Invoked periodically. Application's master uses it to get the information about the progress of training (e.g., round_num, accuracy, straggler_id) and inference.

Table 2. Totoro API.

The first step is to construct application-based logical "trees" of nodes and ensure that these trees are well balanced over the large-scale edge topologies (Figure 4 left).

a. When any new FL application is launched, we calculate the application's AppId, which equals the cryptographic hash of the application's textual name, the creator's public key, and a random salt, *AppId=hash("FL application")*. The hash is computed using the collision resistant SHA-1 hash function, ensuring a uniform distribution of AppIds.
b. Then the edge node processing the application's data routes a `JOIN` message using AppId as the key. Since all nodes belonging to the same application use the same key, their `JOIN` messages eventually arrive at a rendezvous node, with NodeId numerically close to AppId. The rendezvous node is set as the root of this application's tree.
c. The unions of all `JOIN` messages' paths are registered to construct the tree, in which the internal node, as the forwarder, maintains a children table for the group containing an entry (IP address and AppId) for each child. All of the trees together form a "forest" abstraction.
d. For each application's tree, we designate the root node as the master, the internal nodes as the coordinator, aggregator, and client selector components, and the leaf nodes as the workers (participating edge devices).

Rationale: (1) Since different applications have different AppIds, the paths and the rendezvous nodes of their spanning trees will also differ, resulting in an even distribution of trees across all edge nodes. (2) Because all nodes are equal, each node can serve as a leaf, internal, or root node for different applications, thus removing the scalability bottleneck without overburdening any single node.

The second step is to implement a topic-based publish/subscribe messaging protocol within the dataflow tree. Each FL application has a "topic". The master is the "publisher" and the workers are the "subscribers".

a. *Model broadcast.* The master disseminates the FL model to the workers along the tree. Then each worker independently trains a local model and computes the model updates (e.g., gradients and weights) on the local data.
b. *Gradient aggregation.* Once the workers have completed the computation, the master aggregates model updates from the workers, in which each level of the tree progressively aggregates the updates from tree leaves to the root. To meet the diverse needs of different applications, owners can specify different aggregation functions in their trees. For instance, FedAvg [69] works well in most situations, while FedProx [60] demonstrates superior performance in highly heterogeneous settings for more stable and accurate convergence.

Rationale: (1) Due to the loosely coupled interaction between the publisher and subscribers, Totoro can support simultaneously a large number of dataflow trees with a wide range of tree sizes, and a high rate of membership turnover. (2) The use of DHTs enables the efficient construction of aggregation trees and multicast services, as their converging properties guarantee model broadcast or gradient aggregation to be fulfilled within only $O(logN)$ hops, which places an upper bound on worst-case latency for data transfers.

4.4 Layer 3: high-level API

We abstract key components of Totoro to provide an easy-to-use API (see Table 2). Totoro supports application-specific customization, allowing application owners to set their own FL policies. The Pastry DHT and Scribe multicast infrastructure are written in Java, with end-user functionality encapsulated in a Python API. This is done so that users do not have to deal with two libraries in separate languages, and thus Totoro can be easily integrated with popular FL frameworks such as Keras [5], PySyft [11], and TensorFlow Federated [15].

Application-level customization. To prevent potential leakage of model weights to other nodes, application owners can specify various privacy techniques, such as differential privacy [43], secure aggregation [50], and homomorphic encryption [104]. Nodes that subscribe to an FL application adhere to the privacy technique specified by that application during the FL process. For example, if an application owner launches an FL application and specifies the use of differential privacy with Gaussian noise to secure weights, the edge nodes as the master, coordinator, aggregator, and client selector will operate in accordance with the privacy technique. Similarly, the leaf nodes, serving as workers, will apply Gaussian noise to local training. Morever, application owners can use more fine-grained and dedicated third-party libraries, such as Istio [4], for fine-grained enforcement of

traffic policies, complementing Totoro's packet-wise traffic control.

Multi-rings. Application owners can specify whether their applications span multiple zones. For example, a road traffic detection application may require nodes with different weather condition information in different geographical locations. If an application needs to ingest data sources across multiple zones, it will traverse multiple zones (at most m) to build the dataflow tree, resulting in $m * O(logN)$ routing hops.

4.5 Failure recovery

In the case of node failures, we use a parallel recovery approach to repairing the dynamic-structured dataflow trees.

Each node in the tree periodically sends a keep-alive message to its children nodes. If a child node cannot receive keep-alive messages from its parent node, it suspects that the parent has failed. In such a scenario, the child node routes a `JOIN` message to AppId, which triggers the overlay network to route the message to a new parent and creates an alternative route to repair the dataflow tree.

The tree repair process scales well: failure detection is done by sending messages to children nodes only. Failure recovery is also local. Only a small number of nodes ($O(log_{2^b}N)$) is involved, in which 2^b is the fanout of the tree.

5 Bandit-based Path Planning Model

Totoro introduces a new bandit-based path planning model that can replan the data transfer paths in dynamically-structured dataflow trees to adapt to unreliable edge networks.

In most real-world edge networks, link delays are unpredictable and vary stochastically. When a link becomes slow, it can disrupt communication with its child nodes and parent nodes, thereby affecting the entire path from the leaves to the root passing through this link in dataflow trees.

The challenge is that, in many cases, we don't know the quality of the network links in advance, such as the probability of successfully transmitting packets in wireless sensor networks. This information is often obtained by actually sending packets and observing the outcomes. Therefore, when we design the paths for model broadcast and gradient aggregation, we face a dilemma between exploring new or unknown links and exploiting well-known links. If we only rely on the known paths, we may miss out on finding a better path with faster communication or higher success rates. However, if we explore too many new paths, it may result in more packet losses and higher communication delays.

This is where Multi-Armed Bandit (MAB) algorithms [27, 36, 57, 87] come into play. Imagine a person is in a casino, facing multiple slot machines, each with different payout rates. She wants to maximize her winnings, but doesn't know the payout rates of the machines in advance. She can start by trying different machines and recording the results. Over time, she learns which machines offer higher payouts and focuses her efforts on those machines.

Therefore, the path planning problem can be formulated as a combinatorial MAB optimization problem: we explore different paths to learn about their rewards (i.e., link quality), and exploit the paths that offer the highest rewards. By doing so, we can gradually improve our knowledge of the edge networks to find the optimal path.

5.1 Problem formulation

The edge network can be modeled as a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of links. Given a single source-destination pair $(s, d) \in V^2$, let $\mathcal{P} \subseteq \{0, 1\}^{|E|}$ denote the set of all possible loop-free paths from node s to node d in G, where each path $p \in \mathcal{P}$ is a $|E|$-dimensional binary vector; for any $i \in E$, $p_i = 1$ if and only if i belongs to p. In practical edge networks, each link $i \in E$ can be unreliable. At any given time t, we can use a binary variable $X_i(t)$ to represent whether a transmission on link i is successful. $X_i(t)$ is a sequence of independent random events, where each event is either a success or a failure with some unknown probability θ_i. If we repeatedly attempt to send a packet on link i until it succeeds, the time it takes (called the delay) follows a geometric distribution with mean $1/\theta_i$. Among the set of paths $\mathcal{P}$, there must exist a path p^* that yields the minimal packet delay. Formally, $p^* \in \arg\min_{p \in \mathcal{P}} D_\theta(p)$, where $D_\theta(p)$ is the total packet delay of path p.

Optimization objective. Our goal is to efficiently route K packets (such as gradients in many rounds) from a worker node to a master node, minimizing the overall time taken. This can be measured in terms of *regret*, defined as the cumulative difference of expected delay between the path chosen by a policy and the unknown optimal path. Therefore, the regret $R^\pi(K)$ of policy π up to the K-th packet is the expected difference in delays between policy π and the optimal policy that selects the best path p^* for transmission:

$$R^\pi(K) = \mathbb{E}\left[\sum_{k=1}^{K} D^\pi(k)\right] - KD_\theta(p^*),$$

where $D^\pi(k)$ denotes the end-to-end delay of the k-th packet under policy π, the expectation $\mathbb{E}[\cdot]$ is taken with respect to the random transmission outcomes and possible randomization in the policy π, and $D_\theta(p^*)$ is the packet delay of the best path p^*. Our optimization objective is to find a policy π among all policies Π such that

$$\min_{\pi \in \Pi} R^\pi(K). \tag{1}$$

5.2 Our algorithm

We propose a distributed hop-by-hop path planning algorithm based on the semi-feedback bandit model [59, 87, 97].

The pseudo-code of the distributed hop-by-hop routing algorithm is given in Algorithm 1. When node v receives a

Algorithm 1 Distributed hop-by-hop algorithm for node v

1: **for** time slot $\tau \geq 1$ **do**
2: Select link $(v, v') \in E$, where
3: $v' \in \arg\min_{w \in V:(v,w)\in E} C_\tau(v, w)$, where $C_\tau(v, w) = \omega_\tau(v, w) + J_\tau(w)$
4: Update $\omega_\tau(v, v')$ of link (v, v') and $J_\tau(v')$ of node v'.

packet from a previous node at time slot τ, node v leverages a cost function $C_\tau(v, v')$ to evaluate each next possible node $v' \in V$ that has an incoming link $(v, v') \in E$ from node v. The cost function $C_\tau(v, v')$ contains two terms: the *empirical transmission cost with exploration adjustment* $\omega_\tau(v, v')$ and the *long-term routing cost* $J_t(v')$.

We first introduce the term $\omega_\tau(v, v')$. For any time slot τ, let $n(\tau)$ denote the packet number that is about to be sent or already in the network. For any link, let $\hat{\theta}_\tau(v, v')$ denote the empirical success rate of link (v, v') up to time slot τ, that is $\hat{\theta}_\tau(v, v') = s_\tau(v, v')/t'_\tau(v, v')$, where $s_\tau(v, v')$ is the number of packets routed through link (v, v') before the $n(\tau)$-th packet is sent and $t'_\tau(v, v')$ denotes the total number of transmission attempts on link (v, v') up to time slot τ.

Therefore, the empirical transmission cost of link (v, v') with exploration adjustment up to time slot τ is

$$\omega_\tau(v, v') = \min\Big\{\frac{1}{u} : u \in [\hat{\theta}_\tau(v, v'), 1], \\ t'_\tau(v, v') \cdot KL\big(\hat{\theta}_\tau(v, v'), u\big) \leq \log(\tau)\Big\},$$

where $KL(\hat{\theta}_\tau(v, v'), u)$ is the Kullback–Leibler (KL) divergence number [52] between two Bernoulli distributions with respective means $\hat{\theta}_\tau(v, v')$ and u, i.e.,

$$KL(\hat{\theta}_\tau(v, v'), u) = \hat{\theta}_\tau(v, v') \cdot \log\Big(\frac{\hat{\theta}_\tau(v, v')}{u}\Big) \\ + \big(1 - \hat{\theta}_\tau(v, v')\big) \cdot \log\Big(\frac{1 - \hat{\theta}_\tau(v, v')}{1 - u}\Big).$$

We next introduce the term $J_\tau(w)$. Let $\mathcal{P}_w$ denote the set of loop-free paths from node w to the destination of a packet. Then, $J_\tau(w)$ is the minimum total empirical transmission cost with exploration adjustment $\omega_\tau(i)$ of a path p in $\mathcal{P}_w$:

$$J_\tau(w) = \min_{p \in \mathcal{P}_w} \sum_{i \in p} \omega_\tau(i),$$

where $i \in p$ denotes link i in path p.

6 Implementation

We implement Totoro on top of the Pastry (v.2.1) [9] and Keras (v.2.7.0) [5] software stacks. Such implementation choice is motivated by the following considerations: (1) Pastry [80] is a widely used overlay and routing network for the implementation of a DHT similar to Chord [85]. Instead of implementing another distributed system core, we can leverage Pastry's excellent routing substrate (e.g., $O(logN)$ node lookup), self-repairing routing table, message transportation layer, and scalable application-level multicast infrastructure (Scribe [30]). These features greatly simplify the development process. (2) Keras [5] offers an easy and intuitive API for neural networks. For example, Keras supports various backend platforms such as Tensorflow [15], Microsoft Cognitive Toolkit [6], Theano [16], and PlaidML [10].

We made the following major modifications: (1) We changed the original single P2P ring structure to a new locality-aware P2P multi-ring structure. (2) We implemented a fully decentralized "many master/many workers" architecture by utilizing DHTs and adding several data structures: a list of operations for tracking routing paths, selecting masters and workers, and constructing dynamic-structured training trees. (3) We implemented a publish/subscribe messaging pattern for scalable model propagation and gradient aggregation. We introduced a serialization mechanism to convert trained models into binary arrays for low-cost communication over edge networks. (4) We implemented a bandit-based path planning model to replan or repair the dynamically-structured training tree by collecting feedback, detecting node stragglers or failures, and creating alternative routes.

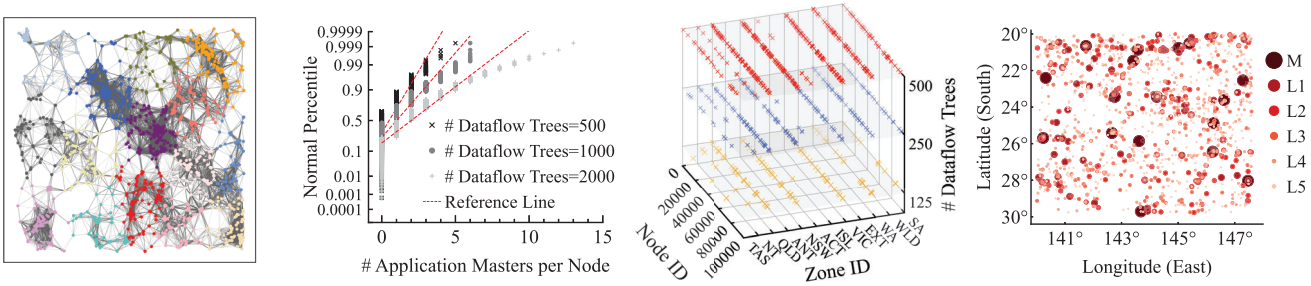
7 Evaluation

We evaluate Totoro's performance in a real-world distributed environment that includes 500 Amazon EC2 nodes and uses real-world computer vision (CV) and natural language processing (NLP) datasets at different scales. We have the following key results.

- Totoro scales the number of masters to handle a varying workload (from 125 to 2000 concurrently running FL applications) in real-world edge topologies (§7.2).
- Totoro achieves $O(logN)$ hops for model dissemination and gradient aggregation with millions of nodes (N) (§7.3).
- Totoro outperforms state-of-the-art FL systems (OpenFL [7] and FedScale [53]) by speeding up the training time 1.2×-14.0× to reach the equivalent model accuracy (§7.4).
- Totoro efficiently adapts to the unreliable edge networks and node churns (§7.5).

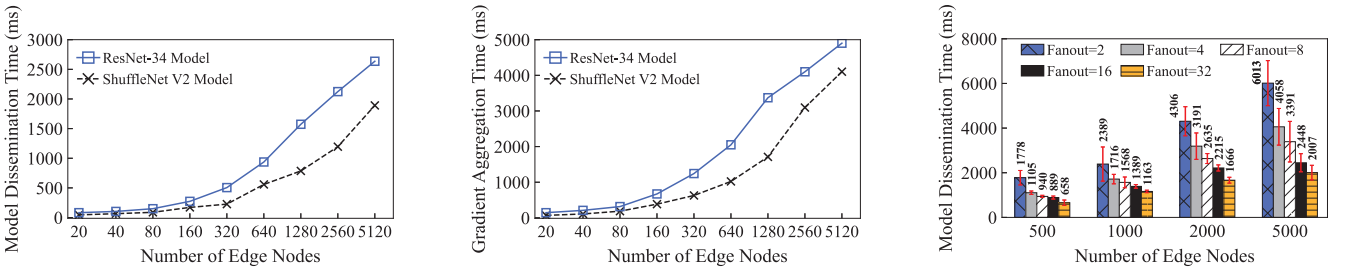
7.1 Methodology

Experimental setup. Totoro is designed to operate in large deployments with millions of edge nodes. However, such a deployment is prohibitively expensive and impractical in an academic environment. As such, we resort to emulating a real-world edge setting on 500 AWS EC2 `t2.medium` nodes, each of which has 2 vCPUs and 4GB of RAM, and 100 GB of disk: (1) we use one JVM to represent one edge node and emulate up to 100k edge nodes on the testbed; (2) we divide the 100k edge nodes into geo-distributed zones based on the real-world EUA dataset [56], which consists of 95,271 edge nodes distributed across 12 Australian states and regions.



(a) Real-world edge topologies generated from the EUA dataset [56].
(b) Normal probability plot of the number of masters mapped per node.
(c) Totoro scales with #masters to handle the varying workload.
(d) The distribution of Totoro's dataflow trees over edge topologies.

Figure 5. Totoro excels at scalability by fairly distributing many dynamic-structured dataflow trees across large-scale edge topologies.



(a) Model dissemination time.
(b) Gradient aggregation time.
(c) Model dissemination time of different fanouts.

Figure 6. Totoro scales with an exponentially increasing number of edge nodes for model dissemination and gradient aggregation.

Baselines. We use OpenFL (v.1.3) [7] and FedScale (v.0.5) [53] as the baseline. FedScale is a scalable and extensible open-source FL engine and benchmarking suite developed by Symbiotic Lab [14], which provides high-level and flexible API to implement, evaluate, and deploy FL algorithms easily in both standalone (single CPU/GPU) and distributed (multiple machines) settings. OpenFL is an open-source framework developed by Intel that runs FL in a single-machine setting.

Parameters. Totoro's Pastry DHT is configured with a leaf set of 24, max open sockets of 5000, and a transport buffer size of 6 MB. We configure different fanouts 8 (2^3), 16 (2^4), and 32 (2^5) for Totoro's dataflow trees by changing the DHT routing table base bit values to 3, 4, and 5, respectively. The minibatch size of each node is 20 in image classification and speech recognition tasks. The initial learning rate for the ShuffleNet V2 model is 0.05 and 0.1 for the ResNet-34 model.

Metrics. We focus on Totoro's scalability, adaptivity, and FL effectiveness. To evaluate scalability, we measure how numerous applications' dataflow trees are distributed over large-scale edge topologies. We also measure how Totoro scales with the number of nodes in terms of *model broadcast time* and *gradient aggregation time*. To evaluate the FL effectiveness, we measure *time-to-accuracy* performance, which is the duration of model training tasks on the testing set to achieve the target accuracy. To evaluate adaptivity, we measure *regret comparison of different algorithms*, *path selection frequencies generated by different algorithms*, and *failure recovery time*. We also measure Totoro's *runtime overhead*.

7.2 Scalability analysis

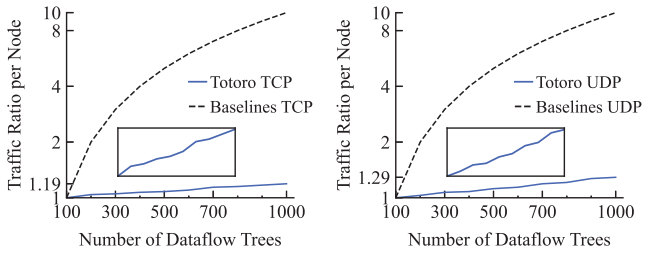
Figure 5a shows the real-world edge zones generated from the EUA dataset [56]. Australian Communications and Media Authority publishes the EUA dataset [56], which contains the geographical locations of 95,271 cellular base stations in 12 states of Australia (ACT: 931, ANT: 15, EXT: 8, ISL: 36, NSW: 24574, NT: 3137, QLD: 21576, SA: 7682, TAS: 3213, VIC: 18163, WA: 15933, WLD: 3). We estimate the maximum round-trip time based on the distances between nodes in the dataset, and then use Ratnasamy and Shenker's distributed binning algorithm [75] to divide them into different zones.

Totoro fairly distributes masters. Figure 5b shows the normal probability plot of the number of masters mapped on each node in a 1000-node edge zone under stress testing. The results show that when we create a large number of dataflow trees like 500, 99.5% of the nodes are the roots of 3 trees or less. The results illustrate a good load balance among participating devices when performing a large number of FL applications' tasks simultaneously. Figure 5c shows the distribution of Totoro's masters over different edge zones that have different workloads. We assume densely populated topologies have heavy workloads and sparsely populated topologies have light workloads. The results show that Totoro automatically scales the number of masters to handle the varying workload.

Totoro excels at load balancing. Figure 5d shows the distribution of all branches in 17 Totoro dataflow trees on 1946

Task	Dataset	Accuracy Target	Model	Number of Applications	Speedup for OpenFL [7]			Speedup for FedScale [53]		
					Fan.=8	Fan.=16	Fan.=32	Fan.=8	Fan.=16	Fan.=32
Speech Recognition	Google Speech [96]	53.0%	ResNet-34 [48]	5	3.7×	3.1×	3.5×	3.6×	3.0×	3.4×
				10	6.4×	6.2×	6.9×	6.2×	6.0×	6.7×
				20	13.0×	11.8×	14.0×	12.4×	11.3×	13.5×
Image Classification	FEMNIST [29]	75.5%	ShuffleNet V2 [67]	5	1.2×	1.4×	3.1×	1.4×	1.6×	3.4×
				10	2.4×	3.2×	5.6×	2.7×	3.5×	6.1×
				20	5.0×	5.5×	10.3×	5.6×	6.1×	11.5×

Table 3. Summary of time-to-accuracy comparison of Totoro, OpenFL, and FedScale. We set three different fanouts for Totoro's dataflow tree: 8, 16, and 32. Totoro outperforms state-of-the-art FL systems by speeding up the training time at different scales. The speedup gap increases as the number of concurrently running applications' tasks increases.



(a) TCP traffic cost. **(b)** UDP traffic cost.

Figure 7. Totoro reduces communication cost.

edge nodes over the 3 most popular topologies. Each tree has a fanout of 8 (2^3) and a random number of depths (from level 1 to level 6). Different colors represent different levels of nodes in a tree, with the darkest color representing the root node. The results show that these dataflow trees are well balanced across different topologies, not only the root nodes but also the forwarder nodes and the leaf nodes, demonstrating Totoro's attractive scalability and load balancing properties.

7.3 Model dissemination and gradient aggregation

Totoro scales with #nodes. Figure 6a and Figure 6b show Totoro's model dissemination and gradient aggregation times for an exponentially increasing number of edge nodes in a single training tree. When the number of nodes grows *exponentially* (from 20 to 5120), the dissemination time and aggregation time only increase *linearly*. This is because the dissemination time and aggregation time are limited by the dataflow tree depth ($O(logN)$) by using the DHT-based P2P overlay. Therefore, when the number of edge devices grows to the scale of millions or even billions, Totoro guarantees that only a few extra hops are needed for them to receive updated FL models or send updated gradients.

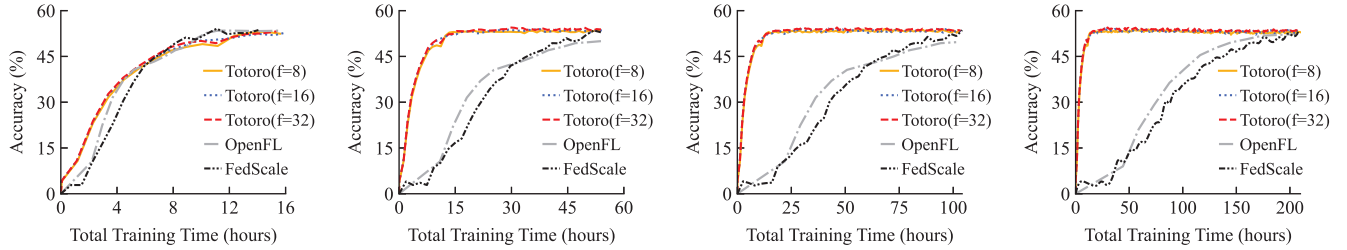
Totoro offers flexible tree fanouts. Figure 6c shows the model dissemination time of different tree fanouts. We can observe that a larger fanout leads to less model dissemination time because it has a smaller tree depth. However, a larger fanout tree is not always good: if an internal node fails or leaves, the new one needs to rebuild many connections between the children nodes and the parent node to rebuild the tree, and the new node may become an I/O bottleneck.

Totoro reduces communication cost. Figure 7 shows the comparison of the traffic per node of Totoro and the baseline FL systems. Communication is a critical bottleneck in FL systems. We can observe that the additional network traffic imposed by Totoro is small. The network traffic is increased by only 1.19× for TCP and 1.29× for UDP when the number of dataflow trees is increased by 10×. This is because when a new training tree is created, it merely routes `JOIN` messages toward the root of the tree using the overlay, adding overlay links to reconstruct the tree without establishing a new connection. Then the overhead for creating new dataflow trees can be amortized over the overhead of the overlay.

7.4 Federated learning effectiveness

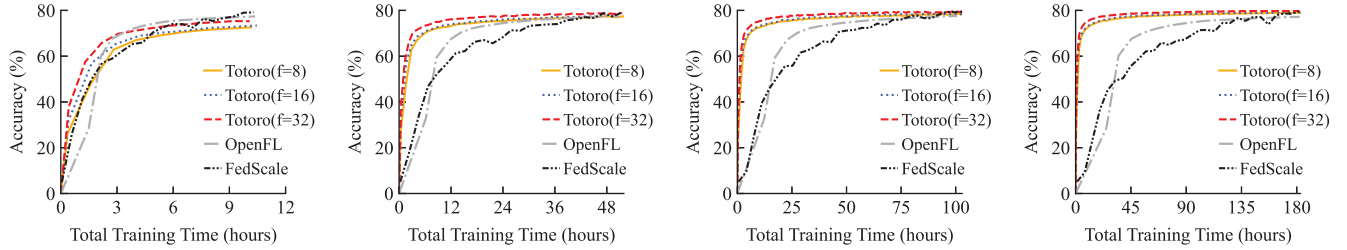
In this section, we evaluate Totoro's performance on model training when an increasing number of models (1 to 20) are simultaneously trained on large-scale edge topologies, and compare it with OpenFL [7] and FedScale [53]. Both FedScale and OpenFL rely on a server-client architecture and leverage a logically central coordinator to spawn aggregators and orchestrate many distributed clients to collaboratively train a model.

Totoro reduces the total model training time. Table 3 summarizes the results of time-to-accuracy comparison of Totoro, OpenFL and FedScale. When 5∼20 models are simultaneously trained on the same platform, we notice that Totoro speeds up the total training time 3.0×-14.0× to reach the equivalent model accuracy on the middle-scale Google Speech dataset [96]; speedup on the large-scale FEMNIST [29] dataset is 1.2×-11.5×. The speedup gap increases as the number of concurrently running applications' tasks increases. This is because both OpenFL and FedScale rely on a server-client architecture where a logically central coordinator orchestrates many distributed clients to collaboratively train a model. When there are a massive number of models that need to be trained simultaneously, the central coordinator needs to handle them one by one on a first-come, first-served basis, which causes large queuing delays. By contrast, Totoro's



(a) 1 application's model **(b)** 5 models trained simultaneously **(c)** 10 models trained simultaneously **(d)** 20 models trained simultaneously

Figure 8. Time-to-accuracy comparison of Totoro, OpenFL, and FedScale. When 5~20 applications' models are simultaneously trained, Totoro speeds up the total training time 3.0×-14.0× to reach the equivalent model accuracy on the middle-scale Google Speech dataset.



(a) 1 application's model **(b)** 5 models trained simultaneously **(c)** 10 models trained simultaneously **(d)** 20 models trained simultaneously

Figure 9. Time-to-accuracy comparison of Totoro, OpenFL, and FedScale. When 5~20 applications' models are simultaneously trained, Totoro speeds up the total training time 1.2×-11.5× to reach the equivalent model accuracy on the large-scale FEMNIST dataset.

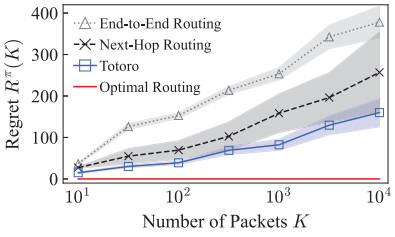


Figure 10. Regret comparison of different algorithms.

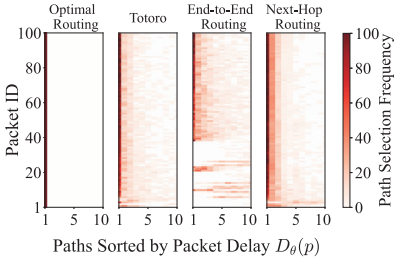


Figure 11. Path selection frequencies generated by different algorithms.

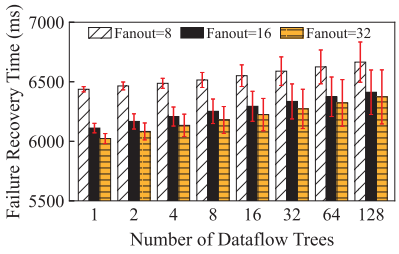


Figure 12. Totoro achieves a stable recovery time for many simultaneous failures.

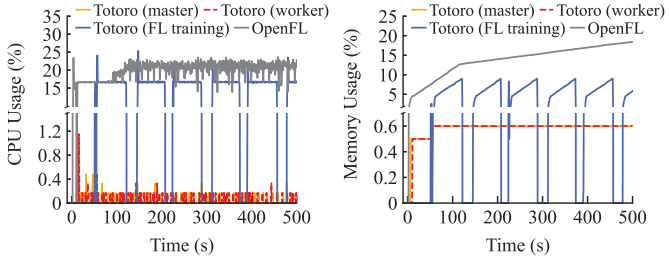
distributed masters can train many models in parallel, thus precluding them from being stuck in a central coordinator.

Totoro scales with #applications. Figure 8 and Figure 9 show the time-to-accuracy performance comparison of Totoro, OpenFL, and FedScale. The results show that Totoro scales well with a large number of applications' training tasks. Totoro takes almost the same total training time to reach model convergence for training 1 model (15.41 hours), 5 models (15.43 hours), 10 models (15.44 hours), and 20 models (15.47 hours) when fanout is equal to 32. The rationale behind the results lies in that (1) Totoro decomposes the conventional "*1:n*" architecture into an "*m:n*" architecture, which ensures that every peer can participate in the process of model training, gradients calculation, and aggregation; and (2) ideally, any peer in the system can act as a worker, a master, a forwarder, or any combination of the above. This helps avoid the scalability bottleneck caused by any single node, auto-scale as needed, balance the workload, and improve the scalability.

7.5 Adaptivity analysis

Adapt to heterogeneous nodes. To handle the heterogeneity of compute resources, we make resource-rich physical edge nodes map to more "P2P nodes", while letting resource-constrained edge nodes only map to fewer "P2P nodes". A Pastry P2P node can be seen as a "logical" node. For example, suppose there are three physical nodes with compute resources being 2, 4, 8 CPU cores, respectively. Accordingly, the physical nodes with 4 and 8 CPU cores can serve as 2 and 3 logical P2P nodes in the DHT-based P2P overlay, respectively. The physical nodes with rich compute resources are more likely to bear more computational overhead.

Adapt to unreliable networks. Figure 10 shows the regret comparison of Totoro's bandit-based path planning algorithm and its counterparts (end-to-end routing [42] and



(a) The CPU overhead. **(b)** The memory overhead.

Figure 13. Overhead comparison of Totoro and OpenFL.

next-hop routing [25]). End-to-end routing selects paths by considering the lower confidence bound (LCB) [83]. Next-hop routing selects the next hop based on empirical packet delays. We can see that Totoro achieves lower regret. This is because Totoro considers the packet delay not only of the next hop but also of the subsequent path starting from the next hop to the destination, which avoids selecting paths with a low-delay first link but with a high overall delay.

Figure 11 shows the path selection frequencies generated by different algorithms, in which each color grid shows the frequencies of selecting the x_{th} path for each packet. The X-axis represents the path in order from the best path to the worst path. The Y-axis represents the sequential order of packets. The baseline (optimal routing) always selects the best path for each packet. Next-hop routing [25] sometimes finds the best path, but it selects some mediocre paths as well. End-to-end routing [42] is the last to find the optimal path. Compared to them, Totoro finds the optimal path the fastest and balances the exploration-exploitation tradeoff.

Adapt to node churns. Figure 12 shows the failure recovery time for an exponentially increasing number of dynamic-structured dataflow trees. Each tree has 5% of nodes that fail or leave at the same time. The results show that Totoro achieves a stable recovery time for many simultaneous trees' failures. This is because each failed node can be quickly detected and recovered by its neighbors through keep-alive messages without having to talk to a central coordinator, so many simultaneous failures can be repaired in parallel.

7.6 Overhead analysis

We train a feedforward model for text classification with a single Totoro's dataflow tree of 10 nodes and compare the overhead with OpenFL.

CPU overhead. Figure 13a shows the CPU overhead comparison of Totoro and OpenFL. For a fair comparison, we divide the CPU overhead into two parts: FL-related tasks, and DHT-related tasks (including constructing P2P overlay, overlay maintenance, building dataflow trees, etc.). The results show that Totoro uses less CPU than OpenFL for FL-related tasks. For DHT-related tasks, Totoro only adds negligible CPU overhead, demonstrating Totoro's portability to resource-constrained edge nodes.

Memory overhead. Figure 13b shows the memory overhead comparison of Totoro and OpenFL. The results show that Totoro uses less memory than OpenFL. The initial increase in the memory overhead is due to the construction of the P2P overlay, routing tables, neighborhood sets, leaf sets, and dataflow trees. After the DHT starts functioning, no additional memory overhead is added in either the master (root node) or the leaf nodes.

8 Conclusion

We present Totoro, a novel scalable federated learning engine for edge networks. It presents three design innovations: a locality-aware P2P multi-ring structure, a publish/subscribe-based forest abstraction, and a bandit-based path planning model. We evaluate Totoro on 500 Amazon EC2 nodes by using real-world CV and NLP datasets at different scales. We compare Totoro with the state-of-the-art and demonstrate substantial advancements in scalability and load balancing, while significantly speeding up the total training time for many concurrently running applications, reducing the communication overhead, and efficiently adapting to unreliable edge networks and churns. Totoro and the workload data will be open-sourced for use by the community.

Acknowledgments

We would like to thank the anonymous reviewers and our shepherd, Umesh Deshpande, for their insightful suggestions and comments that improved this paper. This work is supported by the National Science Foundation (NSF-OAC-2313738, NSF-CAREER-2313737, NSF-SPX-1919181, NSF-SPX-2202859, and NSF-CNS-2322919).

References

[1] Bittorrent. http://www.bittorrent.com/.
[2] Forecast: Iot-enabled software, worldwide, 2019-2025, gartner. https://www.gartner.com/en/documents/4009207.
[3] Freenet: The free network. https://freenetproject.org/.
[4] Istio. https://istio.io/latest/.
[5] Keras. https://keras.io/.
[6] Microsoft cognitive toolkit (cntk), an open source deep-learning toolkit. https://github.com/Microsoft/CNTK/.
[7] OpenFL - an open-source framework for federated learning. https://openfl.readthedocs.io/.
[8] PaddleFL: Paddle federated learning. https://github.com/PaddlePaddle/PaddleFL.
[9] Pastry. https://www.freepastry.org/FreePastry/.
[10] Plaidml - a framework for making deep learning work everywhere. https://github.com/plaidml/plaidml/.
[11] PySyft: A library for easy federated learning. https://github.com/OpenMined/PySyft.
[12] Spark streaming. https://spark.apache.org/streaming/.
[13] Storj.io. http://storj.io/.
[14] Symbiotic lab. https://symbioticlab.org/.
[15] Tensorflow federated: Machine learning on decentralized data. https://www.tensorflow.org/federated.
[16] Theano. https://github.com/Theano/Theano.

[17] M. S. H. Abad, E. Ozfatura, D. Gunduz, and O. Ercetin. Hierarchical federated learning across heterogeneous cellular networks. In *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8866–8870, 2020.

[18] Afshin Abdi and Faramarz Fekri. Quantized compressive sampling of stochastic gradients for efficient communication in distributed deep learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 34(04):3105–3112, Apr. 2020.

[19] Syeda Nahida Akter and Muhammad Abdullah Adnan. Weightgrad: Geo-distributed data analysis using quantization for faster convergence and better accuracy. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 546–556, New York, NY, USA, 2020. Association for Computing Machinery.

[20] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[21] Mohammad S. Aslanpour, Adel N. Toosi, Claudio Cicconetti, Bahman Javadi, Peter Sbarski, Davide Taibi, Marcos Assuncao, Sukhpal Singh Gill, Raj Gaire, and Schahram Dustdar. Serverless edge computing: Vision and challenges. In *Proceedings of the 2021 Australasian Computer Science Week Multiconference*, ACSW '21, New York, NY, USA, 2021. Association for Computing Machinery.

[22] Debraj Basu, Deepesh Data, Can Karakus, and Suhas Diggavi. Qsparse-local-sgd: Distributed sgd with quantization, sparsification and local computations. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[23] Aurélien Bellet, Rachid Guerraoui, Mahsa Taziki, and Marc Tommasi. Personalized and private peer-to-peer machine learning. In *International Conference on Artificial Intelligence and Statistics*, pages 473–481. PMLR, 2018.

[24] Jeremy Bernstein, Jiawei Zhao, Kamyar Azizzadenesheli, and Anima Anandkumar. signSGD with majority vote is communication efficient and fault tolerant. In *International Conference on Learning Representations*, 2019.

[25] Abhijeet A. Bhorkar, Mohammad Naghshvar, Tara Javidi, and Bhaskar D. Rao. Adaptive opportunistic routing for wireless ad hoc networks. *IEEE/ACM Transactions on Networking*, 20(1):243–256, 2012.

[26] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. Towards federated learning at scale: System design. In A. Talwalkar, V. Smith, and M. Zaharia, editors, *Proceedings of Machine Learning and Systems*, volume 1, pages 374–388, 2019.

[27] Sébastien Bubeck, Nicolo Cesa-Bianchi, et al. Regret analysis of stochastic and nonstochastic multi-armed bandit problems. *Foundations and Trends® in Machine Learning*, 5(1):1–122, 2012.

[28] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[29] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečnỳ, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

[30] M. Castro, P. Druschel, A.-M. Kermarrec, and A.I.T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[31] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, Animesh Nandi, Antony Rowstron, and Atul Singh. Splitstream: High-bandwidth multicast in cooperative environments. In *Proceedings of the Nineteenth ACM Symposium on Operating Systems Principles*, SOSP '03, page 298–313, New York, NY, USA, 2003. Association for Computing Machinery.

[32] Miguel Castro, Peter Druschel, Anne-Marie Kermarrec, and Antony Rowstron. One ring to rule them all: Service discovery and binding in structured peer-to-peer overlay networks. In *Proceedings of the 10th Workshop on ACM SIGOPS European Workshop*, EW 10, page 140–145, New York, NY, USA, 2002. Association for Computing Machinery.

[33] Zheng Chai, Ahsan Ali, Syed Zawad, Stacey Truex, Ali Anwar, Nathalie Baracaldo, Yi Zhou, Heiko Ludwig, Feng Yan, and Yue Cheng. Tifl: A tier-based federated learning system. In *Proceedings of the 29th International Symposium on High-Performance Parallel and Distributed Computing*, HPDC '20, page 125–136, New York, NY, USA, 2020. Association for Computing Machinery.

[34] Zheng Chai, Yujing Chen, Ali Anwar, Liang Zhao, Yue Cheng, and Huzefa Rangwala. Fedat: A high-performance and communication-efficient federated learning system with asynchronous tiers. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[35] Chia-Yu Chen, Jiamin Ni, Songtao Lu, Xiaodong Cui, Pin-Yu Chen, Xiao Sun, Naigang Wang, Swagath Venkataramani, Vijayalakshmi Srinivasan, Wei Zhang, and Kailash Gopalakrishnan. Scalecom: Scalable sparsified gradient compression for communication-efficient distributed training. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS'20, Red Hook, NY, USA, 2020. Curran Associates Inc.

[36] Wei Chen, Yajun Wang, and Yang Yuan. Combinatorial multi-armed bandit: General framework and applications. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 151–159, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[37] Y. Chen, Y. Ning, M. Slawski, and H. Rangwala. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24, Los Alamitos, CA, USA, dec 2020. IEEE Computer Society.

[38] Yang Chen, Xiaoyan Sun, and Yaochu Jin. Communication-efficient federated deep learning with layerwise asynchronous model update and temporally weighted aggregation. *IEEE Transactions on Neural Networks and Learning Systems*, 31(10):4229–4238, 2020.

[39] Yiqiang Chen, Xin Qin, Jindong Wang, Chaohui Yu, and Wen Gao. Fedhealth: A federated transfer learning framework for wearable healthcare. *IEEE Intelligent Systems*, 35(4):83–93, 2020.

[40] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed E. Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. On scaling decentralized blockchains - (A position paper). In Jeremy Clark, Sarah Meiklejohn, Peter Y. A. Ryan, Dan S. Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security - FC 2016 International Workshops, BITCOIN, VOTING, and WAHC, Christ Church, Barbados, February 26, 2016, Revised Selected Papers*, volume 9604 of *Lecture Notes in Computer Science*, pages 106–125. Springer, 2016.

[41] Cheng Fang, Zhixiong Yang, and Waheed U. Bajwa. Bridge: Byzantine-resilient decentralized gradient descent. *IEEE Transactions on Signal and Information Processing over Networks*, 8:610–626, 2022.

[42] Yi Gai, Bhaskar Krishnamachari, and Rahul Jain. Combinatorial network optimization with unknown variables: Multi-armed bandits

with linear rewards and individual observations. *IEEE/ACM Transactions on Networking*, 20(5):1466–1478, 2012.

[43] Robin C Geyer, Tassilo Klein, and Moin Nabi. Differentially private federated learning: A client level perspective. In *Advances in Neural Information Processing Systems Workshop: Machine Learning on the Phone and other Consumer Devices*, 2017.

[44] Antonious Girgis, Deepesh Data, Suhas Diggavi, Peter Kairouz, and Ananda Theertha Suresh. Shuffled model of differential privacy in federated learning. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 2521–2529. PMLR, 13–15 Apr 2021.

[45] Nirupam Gupta and Nitin H Vaidya. Byzantine fault-tolerance in peer-to-peer distributed gradient-descent. *arXiv preprint arXiv:2101.12316*, 2021.

[46] Farzin Haddadpour, Mohammad Mahdi Kamani, Aryan Mokhtari, and Mehrdad Mahdavi. Federated learning with compression: Unified analysis and sharp guarantees. In Arindam Banerjee and Kenji Fukumizu, editors, *Proceedings of The 24th International Conference on Artificial Intelligence and Statistics*, volume 130 of *Proceedings of Machine Learning Research*, pages 2350–2358. PMLR, 13–15 Apr 2021.

[47] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604*, 2018.

[48] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[49] Kevin Hsieh, Aaron Harlap, Nandita Vijaykumar, Dimitris Konomis, Gregory R. Ganger, Phillip B. Gibbons, and Onur Mutlu. Gaia: Geo-distributed machine learning approaching lan speeds. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 629–647, USA, 2017. USENIX Association.

[50] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustinov, Harish Srinivas, Kaikai Wang, Anthony Shoumikhin, Jesik Min, and Mani Malek. Papaya: Practical, private, and scalable federated learning. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of Machine Learning and Systems*, volume 4, pages 814–832, 2022.

[51] Latif U. Khan, Shashi Raj Pandey, Nguyen H. Tran, Walid Saad, Zhu Han, Minh N. H. Nguyen, and Choong Seon Hong. Federated learning for edge networks: Resource optimization and incentive mechanism. *IEEE Communications Magazine*, 58(10):88–93, 2020.

[52] Solomon Kullback and Richard A Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[53] Fan Lai, Yinwei Dai, Sanjay Singapuram, Jiachen Liu, Xiangfeng Zhu, Harsha Madhyastha, and Mosharaf Chowdhury. FedScale: Benchmarking model and system performance of federated learning at scale. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 11814–11827. PMLR, 17–23 Jul 2022.

[54] Fan Lai, Jie You, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Sol: Fast distributed computation over slow networks. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation*, NSDI'20, page 273–288, USA, 2020. USENIX Association.

[55] Fan Lai, Xiangfeng Zhu, Harsha V. Madhyastha, and Mosharaf Chowdhury. Oort: Efficient federated learning via guided participant selection. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 19–35. USENIX Association, July 2021.

[56] Phu Lai, Qiang He, Mohamed Abdelrazek, Feifei Chen, John Hosking, John Grundy, and Yun Yang. Optimal edge user allocation in edge computing with variable sized vector bin packing. In Claus Pahl, Maja Vukovic, Jianwei Yin, and Qi Yu, editors, *Service-Oriented Computing*, pages 230–245, Cham, 2018. Springer International Publishing.

[57] T.L Lai and Herbert Robbins. Asymptotically efficient adaptive allocation rules. *Advances in Applied Mathematics*, 6(1):4–22, mar 1985.

[58] Anusha Lalitha, Shubhanshu Shekhar, Tara Javidi, and Farinaz Koushanfar. Fully decentralized federated learning. In *Advances in Neural Information Processing Systems Workshop on Bayesian Deep Learning*, volume 2, 2018.

[59] Tor Lattimore and Csaba Szepesvári. *Bandit algorithms*. Cambridge University Press, 2020.

[60] Tian Li, Anit Kumar Sahu, Manzil Zaheer, Maziar Sanjabi, Ameet Talwalkar, and Virginia Smith. Federated optimization in heterogeneous networks. *Proceedings of Machine learning and systems*, 2:429–450, 2020.

[61] Tian Li, Maziar Sanjabi, Ahmad Beirami, and Virginia Smith. Fair resource allocation in federated learning. In *International Conference on Learning Representations*, 2020.

[62] Zhize Li, Dmitry Kovalev, Xun Qian, and Peter Richtárik. Acceleration for compressed gradient descent in distributed and federated optimization. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[63] Tao Lin, Sebastian U. Stich, Kumar Kshitij Patel, and Martin Jaggi. Don't use large mini-batches, use local sgd. In *International Conference on Learning Representations*, 2020.

[64] Lumin Liu, Jun Zhang, S.H. Song, and Khaled B. Letaief. Client-edge-cloud hierarchical federated learning. In *ICC 2020 - 2020 IEEE International Conference on Communications (ICC)*, pages 1–6, 2020.

[65] Lumin Liu, Jun Zhang, Shenghui Song, and Khaled B. Letaief. Hierarchical federated learning with quantization: Convergence analysis and system design. *IEEE Transactions on Wireless Communications*, 22(1):2–18, 2023.

[66] Heiko Ludwig, Nathalie Baracaldo, Gegi Thomas, Yi Zhou, Ali Anwar, Shashank Rajamoni, Yuya Ong, Jayaram Radhakrishnan, Ashish Verma, Mathieu Sinn, et al. IBM federated learning: an enterprise framework white paper v0. 1. *arXiv preprint arXiv:2007.10987*, 2020.

[67] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *European Conference on Computer Vision – ECCV 2018*, pages 122–138, Cham, 2018. Springer International Publishing.

[68] Petar Maymounkov and David Mazières. Kademlia: A peer-to-peer information system based on the xor metric. In *Revised Papers from the First International Workshop on Peer-to-Peer Systems*, IPTPS '01, page 53–65, Berlin, Heidelberg, 2002. Springer-Verlag.

[69] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 20–22 Apr 2017.

[70] Vaikkunth Mugunthan, Antigoni Polychroniadou, David Byrd, and Tucker Hybinette Balch. Smpai: Secure multi-party computation for federated learning. In *Advances in Neural Information Processing Systems Workshop on Robust AI in Financial Services*, 2019.

[71] Takayuki Nishio and Ryo Yonetani. Client selection for federated learning with heterogeneous resources in mobile edge. In *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pages 1–7, 2019.

[72] Shashi Raj Pandey, Minh N. H. Nguyen, Tri Nguyen Dang, Nguyen H. Tran, Kyi Thar, Zhu Han, and Choong Seon Hong. Edge-assisted democratized learning toward federated analytics. *IEEE Internet of Things Journal*, 9(1):572–588, 2022.

[73] Matthias Paulik, Matt Seigel, Henry Mason, Dominic Telaar, Joris Kluivers, Rogier van Dalen, Chi Wai Lau, Luke Carlson, Filip Granqvist, Chris Vandevelde, et al. Federated evaluation and tuning for on-device personalization: System design & applications. *arXiv preprint arXiv:2102.08503*, 2021.

[74] Shi Pu, Alex Olshevsky, and Ioannis Ch. Paschalidis. Asymptotic network independence in distributed stochastic optimization for machine learning: Examining distributed and centralized stochastic gradient descent. *IEEE Signal Processing Magazine*, 37(3):114–122, 2020.

[75] S. Ratnasamy, M. Handley, R. Karp, and S. Shenker. Topologically-aware overlay construction and server selection. In *Proceedings.Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 3, pages 1190–1199 vol.3, 2002.

[76] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, page 161–172, New York, NY, USA, 2001. Association for Computing Machinery.

[77] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *International Conference on Learning Representations*, 2021.

[78] Amirhossein Reisizadeh, Aryan Mokhtari, Hamed Hassani, Ali Jadbabaie, and Ramtin Pedarsani. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In Silvia Chiappa and Roberto Calandra, editors, *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics*, volume 108 of *Proceedings of Machine Learning Research*, pages 2021–2031. PMLR, 26–28 Aug 2020.

[79] Daniel Rothchild, Ashwinee Panda, Enayat Ullah, Nikita Ivkin, Ion Stoica, Vladimir Braverman, Joseph Gonzalez, and Raman Arora. FetchSGD: Communication-efficient federated learning with sketching. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8253–8265. PMLR, 13–18 Jul 2020.

[80] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proceedings of the IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg*, Middleware '01, page 329–350, Berlin, Heidelberg, 2001. Springer-Verlag.

[81] Abhijit Guha Roy, Shayan Siddiqui, Sebastian Pölsterl, Nassir Navab, and Christian Wachinger. Braintorrent: A peer-to-peer environment for decentralized federated learning. *arXiv preprint arXiv:1905.06731*, 2019.

[82] Karan Singhal, Hakim Sidahmed, Zachary Garrett, Shanshan Wu, John Rush, and Sushant Prakash. Federated reconstruction: Partially local federated learning. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan, editors, *Advances in Neural Information Processing Systems*, volume 34, pages 11220–11232. Curran Associates, Inc., 2021.

[83] Aleksandrs Slivkins et al. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

[84] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet S Talwalkar. Federated multi-task learning. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[85] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, page 149–160, New York, NY, USA, 2001. Association for Computing Machinery.

[86] Dimitris Stripelis, Paul M. Thompson, and José Luis Ambite. Semi-synchronous federated learning for energy-efficient training and accelerated convergence in cross-silo settings. *ACM Transactions on Intelligent Systems and Technology*, 13(5), jun 2022.

[87] Mohammad Sadegh Talebi, Zhenhua Zou, Richard Combes, Alexandre Proutiere, and Mikael Johansson. Stochastic online shortest path routing: The value of feedback. *IEEE Transactions on Automatic Control*, 63(4):915–930, 2018.

[88] Hanlin Tang, Xiangru Lian, Ming Yan, Ce Zhang, and Ji Liu. d^2: Decentralized training over decentralized data. In Jennifer Dy and Andreas Krause, editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 4848–4856. PMLR, 10–15 Jul 2018.

[89] Michael Teng and Frank Wood. Bayesian distributed stochastic gradient descent. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018.

[90] Muhammed Uluyol, Anthony Huang, Ayush Goel, Mosharaf Chowdhury, and Harsha V. Madhyastha. Near-optimal latency versus cost tradeoffs in geo-distributed storage. In *Proceedings of the 17th Usenix Conference on Networked Systems Design and Implementation*, NSDI'20, page 157–180, USA, 2020. USENIX Association.

[91] Paul Vanhaesebrouck, Aurélien Bellet, and Marc Tommasi. Decentralized Collaborative Learning of Personalized Models over Networks. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 509–517. PMLR, 20–22 Apr 2017.

[92] Thijs Vogels, Sai Praneeth Karimireddy, and Martin Jaggi. Powersgd: Practical low-rank gradient compression for distributed optimization. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

[93] Ewen Wang, Boyi Chen, Mosharaf Chowdhury, Ajay Kannan, and Franco Liang. Flint: A platform for federated learning integration. *Proceedings of Machine Learning and Systems*, 2023.

[94] Jianyu Wang and Gauri Joshi. Adaptive communication strategies to achieve the best error-runtime trade-off in local-update sgd. *Proceedings of Machine Learning and Systems*, 1:212–229, 2019.

[95] Shiqiang Wang, Tiffany Tuor, Theodoros Salonidis, Kin K. Leung, Christian Makaya, Ting He, and Kevin Chan. Adaptive federated learning in resource constrained edge computing systems. *IEEE Journal on Selected Areas in Communications*, 37(6):1205–1221, 2019.

[96] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[97] Zheng Wen, Branislav Kveton, Michal Valko, and Sharan Vaswani. Online influence maximization under independent cascade model with semi-bandit feedback. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[98] Qiong Wu, Xu Chen, Zhi Zhou, and Junshan Zhang. Fedhome: Cloud-edge based personalized federated learning for in-home health monitoring. *IEEE Transactions on Mobile Computing*, 21(8):2818–2832, 2022.

[99] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. In *Advances in Neural Information Processing Systems Workshop on Optimization for Machine Learning*, 2020.

[100] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903*, 2018.

[101] Zhixiong Yang and Waheed U. Bajwa. Byrdie: Byzantine-resilient distributed coordinate descent for decentralized learning. *IEEE Transactions on Signal and Information Processing over Networks*, 5(4):611–627, 2019.

[102] Hao Yu, Sen Yang, and Shenghuo Zhu. Parallel restarted sgd with faster convergence and less communication: Demystifying why model averaging works for deep learning. In *Proceedings of the Thirty-Third AAAI Conference on Artificial Intelligence and Thirty-First Innovative Applications of Artificial Intelligence Conference and Ninth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'19/IAAI'19/EAAI'19. AAAI Press, 2019.

[103] Ben Zhang, Xin Jin, Sylvia Ratnasamy, John Wawrzynek, and Edward A. Lee. Awstream: Adaptive wide-area streaming analytics. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, SIGCOMM '18, page 236–252, New York, NY, USA, 2018. Association for Computing Machinery.

[104] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. BatchCrypt: Efficient homomorphic encryption for Cross-Silo federated learning. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 493–506. USENIX Association, July 2020.

[105] Daniel Yue Zhang, Ziyi Kou, and Dong Wang. Fedsens: A federated learning approach for smart health sensing with class imbalance in resource constrained edge computing. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.

[106] Yu Zhang, Morning Duan, Duo Liu, Li Li, Ao Ren, Xianzhang Chen, Yujuan Tan, and Chengliang Wang. CSAFL: A clustered semi-asynchronous federated learning framework. In *2021 International Joint Conference on Neural Networks (IJCNN)*, pages 1–10, 2021.

[107] B. Y. Zhao, Ling Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22(1):41–53, sep 2006.

[108] Mingchen Zhao, Paarijaat Aditya, Ang Chen, Yin Lin, Andreas Haeberlen, Peter Druschel, Bruce Maggs, Bill Wishon, and Miroslav Ponec. Peer-assisted content distribution in akamai netsession. In *Proceedings of the 2013 Conference on Internet Measurement Conference*, IMC '13, page 31–42, New York, NY, USA, 2013. Association for Computing Machinery.