Non-Invasive User Profiling through Label Inference in FL-based Network Traffic Classification

Huiwen Zhang and Feng Ye
Department of Electrical and Computer Engineering
University of Wisconsin-Madison, Madison, WI, USA
Emails: hzhang2279@wisc.edu, feng.ye@wisc.edu

***Abstract*—Artificial intelligence (AI) supported network traffic classification (NTC) has been developed lately for network measurement and quality-of-service (QoS) purposes. More recently, federated learning (FL) approach has been promoted for distributed NTC development due to its nature of unshared dataset for better privacy and confidentiality in raw networking data collection and sharing. However, network measurement still require invasive probes and constant traffic monitoring. In this paper, we propose a non-invasive network traffic estimation and user profiling mechanism by leveraging label inference of FL-based NTC. In specific, the proposed scheme only monitors weight differences in FL model updates from a targeting user and recovers its network application (APP) labels as well as a rough estimate on the traffic pattern. Assuming a slotted FL update mechanism, the proposed scheme further maps inferred labels from multiple slots to different profiling classes that depend on, e.g., QoS and APP categorization. Without loss of generality, user profiles are determined based on normalized productivity, entertainment, and casual usage scores derived from an existing commercial router and its backend server. A slot extension mechanism is further developed for more accurate profiling beyond raw traffic measurement. Evaluations conducted on seven popular APPs across three user profiles demonstrate that our approach can achieve accurate networking user profiling without invasive physical probes nor constant traffic monitoring.**

***Index Terms*—User profiling, federated learning, label inference, network traffic classification**

I. Introduction

The plethora of networking devices and data-hungry applications (interchangeable with APPs in the rest of the paper) have led to a huge increase in network traffic. Due to the diversity of user demands, it is imperative to provide accurate network traffic measurement on the user level to personalize the optimal network quality of service (QoS). Traditional QoS mechanisms, such as port or payload filtering are ineffective as more APPs adopt dynamic port assignment as well as end-to-end encryption. To address these challenges, artificial intelligence (AI) based network traffic classification (NTC) solutions have been developed [1], offering high accuracy and adaptability across a wide variety of network traffic types. Albeit providing high accuracy, centralized NTC classifiers require users to share raw packet traces with a central server, raising privacy concerns and causing substantial bandwidth costs [2]. Federated Learning (FL) has emerged as a paradigm to decentralize model training, enabling multiple clients to collaboratively learn a shared classifier without exchanging raw data. In the context of NTC, FL-based approaches allow network domains to jointly train a global traffic classifier while preserving data locality and user privacy [3–5]. However, FL is not immune to privacy leakage due. Attacks such as membership inference [6], data reconstruction [7], and label inference [8, 9] exploit shared model updated parameters to recover private information, including clients' raw datasets.

Rather than exploiting user privacy, this work leverages inference on FL-based NTC for benign purposes. In specific, we propose a non-invasive network traffic estimation and user QoS profiling scheme such that APP-level QoS can be estimated and personalized for each en user, e.g., on a Wi-Fi router. The existing commercial solutions rely on adaptive QoS to personalize traffic management, such as ASUS Work Home [10] and Huawei Adaptive QoS Configuration [11]. These systems require users to manually select or define service profiles (e.g., "Gaming," "Streaming," "Websurfing") and often depend on third-party deep-packet inspection services to analyze traffic characteristics. Such approaches not only burden users with profile configuration, but also raise privacy and scalability concerns. Our proposed non-invasive profiling approach estimates network traffic by inferring users' application usage tendencies directly from the parameters exchanged during FL-NTC updates (defined as slots), without accessing nor constant monitoring raw traffic data. Focusing on the weight differences between FL-NTC updates, the proposed inference approach can recover APP appearances in each update interval for a targeting user client. These appearances are then optimized through a slot-extension mechanism, which adjusts the difference between the raw traffic pattern and the real APP usage. After that, behavior weight vectors developed based on existing commercial solutions are assigned to each APP. The final results provide a user profile towards multiple directions, including productivity, entertainment, and casual. Simulations are conducted on seven popular APPs across three distinct user profiles. The evaluation results demonstrate that the proposed non-invasive network user profiling scheme can successfully estimate the APP traffic pattern and provide accurate QoS profiles for each user. To summarize, our contributions in this paper are threefold:

- A non-invasive network traffic estimation and user QoS profiling scheme is proposed by leveraging label inference on FL-based NTC. The proposed method does not require physical probes nor constant traffic monitoring, as

opposed to the traditional QoS management mechanisms.
- A slot extension and behavior weight assignment mechanism is developed to optimize the estimation of APP usage against raw traffic patterns for better user QoS profiling.
- Extensive simulations are conducted using seven popular APPs across three distinct user profiles according to a commercial adaptive QoS solution. The results demonstrate that the user QoS profiles inferred from the proposed approach can match the ground truth well.

The remainder of this paper is organized as follows. Section II reviews related work in FL-NTC and inference attacks. Section III presents our system model and preliminaries. Section IV describes the label inference and profiling algorithms. Section V details our experimental setup and results. Finally, Section VI concludes the paper and outlines future work.

II. Related Work

A. FL-based NTC

NTC aims to assign packets or flows to application or service level categories, enabling Internet service providers and network operators to enforce policies, optimize network QoS, and detect anomalies. However, early approaches that rely on port- or payload-based rules are less effective due to the use of dynamic ports and encryption. AI-based NTC has been promoted as a viable solution, achieving high accuracy and adaptability across diverse traffic types [2, 12]. Recently, FL-based NTC has attracted attention as a way to train global traffic classifiers collaboratively without exchanging raw packet traces, thereby preserving user privacy and reducing central bandwidth requirements. For example, FLIC was one of the first to adapt FedAvg specifically for Internet traffic classification, showing that a federated protocol can match the accuracy of centralized deep models even under non-i.i.d. traffic distributions across clients, while classifying previously unseen applications on the fly [3]. Subsequent work has diversified both the application scenarios and the FL techniques used in NTC. For instance, the authors of [4] proposed FedETC, which adapts federated learning for encrypted traffic classification by training directly on TLS flows without decryption, achieving accuracy on par with centralized models while preserving end-to-end privacy. Meanwhile, cross-model FL-based NTC lets each client use its own local architecture and then fuses updates in a modality-aware way at the server, boosting classification robustness under highly non-IID traffic distributions [5].

B. Inference Attack on ML and FL

Machine learning and AI models have been shown vulnerable to a variety of inference attacks in centralized environments. For example, the authors of [13] introduced the model inversion attack, demonstrating how an adversary can reconstruct sensitive input features by exploiting confidence scores returned by a target model. The authors of [14] proposed property inference attacks, where an attacker infers aggregate properties of the training data that the model designer did not intend to disclose. The authors of [6] proposed membership inference in which an adversary trains shadow models to mimic the target and then learns to distinguish whether a given example was used in training based solely on the model's confidence scores. This approach demonstrated high accuracy even in black-box settings. While FL retains the raw dataset by only exchanging parameters for model updates, it does not eliminate the risk from inference attacks. For example, FL can still be vulnerable to inference attacks such as data inference, including membership inference, data reconstruction, and label inference attacks. In particular, the authors of [15] extended the membership inference attack in FL. The authors of [7] introduced deep leakage from gradients, showing that by optimizing dummy inputs to match a single observed gradient, one can reconstruct the original training data. More recently, the authors of [9] exposed user-level label leakage by analyzing gradient updates over multiple rounds. This attack can recover the distribution of labels held by individual clients. These findings underscore that federated learning, while preserving raw-data locality, may still be exploited to reveal useful information. In this paper, instead of attacking FL-based NTC, we leverage such inference attacks as a benign approach to achieve non-invasive network traffic monitoring for user profiling, which can be further used for QoS management.

C. QoS and Network User Profiling

Traditional QoS management relies heavily on static rules, such as port-based and manual configuration of priority levels on routers or user devices [16]. While such approaches can allocate bandwidth according to coarse service classes (e.g., VoIP, video, bulk transfer), they often fail to adapt to dynamic user behavior or encrypted traffic. By integrating real-time and more accurate APP-level NTC, QoS can be made more responsive and personalized for end users. For example, machine-learning classifiers operating on flow metadata or lightweight packet features enable fine-grained identification of applications and services, allowing routers to prioritize latency-sensitive flows (e.g., video conferencing) over bulk transfers in real time [17]. However, NTC alone is not a perfect solution. Accurate classification often requires either probe-based deep-packet inspection or large caches of flow statistics, imposing significant storage and compute overhead on edge devices [18]. Moreover, unless it is a probe installed on the user device, encrypted traffic and evolving application protocols can degrade classifier accuracy on a router. Offloading NTC to a backend server can mitigate resource constraints but introduces privacy concerns, as raw or semi-processed user traffic must be transmitted and stored externally.

To address these issues, modern home routers such as those from ASUS adopt a hybrid user profiling model. ASUS's Adaptive QoS embeds NTC capabilities directly in the firmware, yet relies on a third-party cloud service to perform deep-packet inspection and behavioral analysis [10]. In the web interface, users select from predefined profiles from

"Gaming", "Streaming", "Websurfing", "Study at home" or "Work at home", or create custom service rules by assigning priority levels to applications and devices. Only after a profile is chosen does the router apply real-time flow monitoring and dynamic bandwidth allocation according to the selected settings. It is a complicated and sub-optimal process for a user to define their own policies. Since the service also requires continuous data collection by a cloud service, it raises further privacy concerns. The proposed non-invasive user profiling scheme in this paper is to address these challenges by providing network traffic estimation and user QoS profiling without manual configuration, third-party data collection, nor a probe on the user device. autonomous

III. System Model and Preliminaries

A. Studied System Model and FL-based NTC Implementation

The studied system model in this work is depicted in Fig. 1, assuming N clients participate in the FL process. In this framework, each client performs a communication and training cycle at a fixed time slot, e.g., every ten seconds. Specifically, at the beginning of each update interval, each client uploads its local model parameters to the server and simultaneously downloads the latest global model parameters. Upon receiving the global model, the client uses the newly collected data from the current update interval to perform local training, e.g., for L epochs. After completing the local updates, the client uploads its updated model parameters to the server again to participate in the next round of global aggregation. The FL process adopts a simple averaging strategy rather than the standard weighted averaging used in FedAvg. The global model parameters are updated at the server after each round as follows,

$$P_G = \frac{1}{N} \sum_{k=1}^{N} P_{L_k},$$

where P_{L_k} denotes the parameters of the local model from client k.

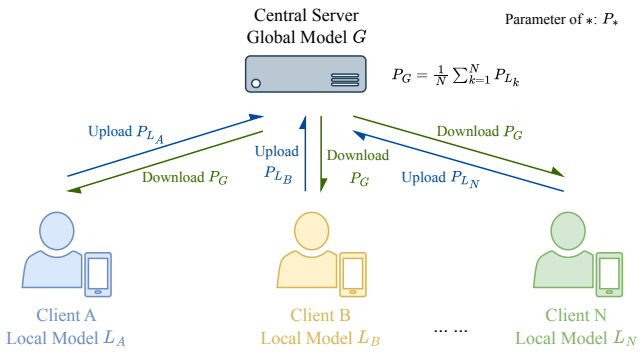


Fig. 1: Overview of the studied FL-based NTC system model.

The chosen NTC architecture consists of three linear layers: the first layer maps 500-dimensional input features to a 256-dimensional hidden representation, followed by batch normalization, LeakyReLU activation (slope 0.01), and dropout with rate 0.5. The second hidden layer reduces the dimensionality to 32, also followed by batch normalization, LeakyReLU activation, and dropout. The final output layer maps to the corresponding traffic classes. For better demonstration, the chosen NTC is kept lightweight, while still being capable of handling the chosen network applications in this work. Advanced NTC that handles more complicated network applications can be implemented into the framework straightforwardly. Moreover, we argue that the lightweight NTC with moderate performance can better demonstrate the efficacy of the proposed non-invasive network measurement and user profiling in practical scenarios where an NTC is not well-trained or during an FL update process.

B. Gradients Attack and Label Leakage

The Label Leakage from Gradients (LLG) attack, first introduced in [8, 9], exploits the gradients shared during the federated learning process to infer private label information about clients' local data. The core insight behind LLG is that the gradients of a neural network, particularly those computed during the first few training steps, are highly informative about the class labels used to generate them. This is because gradients encode how the model's predictions differ from the true labels. In our FL-based NTC setup, N edge routers (clients) each collect a few batches of packets and participate in periodic model update exchanges with a central server. In each round, clients download the current global model, perform local training on their freshly collected traffic data, and upload only the resulting parameter updates. Leveraging the concept of LLG attack, we assume the central server is honest-but-curious (i.e. it will correctly perform aggregation but may inspect or manipulate received gradients) and can therefore act as an adversary to infer sensitive information about clients' local traffic labels. Note that the central server is indeed a benign and trusted user. It only matches technically its role to an adversary of LLG attack. Moreover, although LLG was first described in terms of raw gradient vectors, we propose to operate on the weight deltas instead during local training. This approach enables high-accuracy label recovery without any deep-packet inspection.

IV. Non-Invasive Network User Profiling

A. Overview of the Network User Profiling Scheme

The overall network user profiling scheme follows a two-stage process, as illustrated in Fig. 2. In the first stage, label

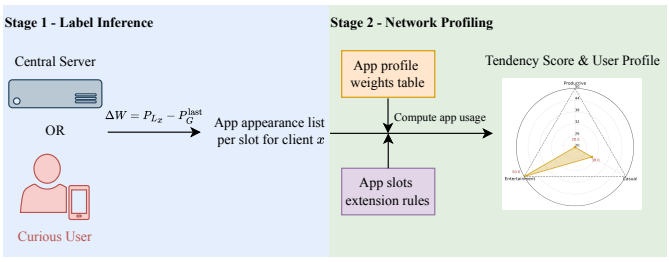


Fig. 2: Overview of the user profiling mechanism.

inference is performed to recover the label appearance list for each time slot. This step can be carried out either by the central server or by a curious client, and it requires access to both the global model parameters and the target client's local model parameters. In the second stage, the recovered label appearance list is used to conduct network profiling. App usage profiles are computed by combining the app profile weights table and the slot extension rules, which compensate for caching effects by extending app appearances across adjacent slots. Based on the aggregated app usage, normalized tendency scores across different usage types are calculated to generate the user's final profile. For better illustration, each time slot is set to 10 seconds. Each client uploads its locally updated model parameters to the server at the end of each slot.

B. Label Inference and Traffic Estimation

Compared to the existing LLG, our method does not attempt to recover the full set of labels or the precise occurrence counts within a batch. Rather, we focus solely on identifying which labels are present in a given client update and approximating their proportions. Furthermore, unlike LLG, where the federated aggregation is performed over shared gradients, our setting aggregates model parameters obtained after local training. Correspondingly, our label inference is conducted on the weight differences between the latest local client updates and the previous global parameters, rather than directly on the gradients. Specifically, for a target client x, the weight difference is defined as follows,

$$\Delta W_x = P_{L_x} - P_G^{\text{last}}, \tag{1}$$

where P_{L_x} denotes the locally trained model parameters from client x and P_G^{last} denotes the global model parameters from the previous round. We further observe that positive entries in ΔW_x are indicative of label appearances, opposite to LLG, where negative gradients reveal present labels. The actual label inference is processed in four stages.

1) For each class c, the aggregate score is computed as follows,
$$g_c = \sum_{i=1}^{d} \Delta W_{c,i}. \tag{2}$$
All classes with $g_c > 0$ are collected into the candidate set $\mathcal{H}_1$.
2) The average per-sample impact of each positive class is estimated as follows,
$$\text{impact} = \frac{1}{L}\left(\frac{1}{B}\sum_{c\in\mathcal{H}_1} g_c\right)\left(1+\tfrac{1}{C}\right), \tag{3}$$
where B is the batch size, C the number of classes, and L the number of local iterations.
3) One sample of each class in $\mathcal{H}_1$ is assigned by subtracting impact from its g_c. These samples are collected into the provisional multiset $\mathcal{P}$.
4) While $|\mathcal{P}| < B$, we repeatedly select class c^* with the largest remaining g_c, add one more sample of c^* to $\mathcal{P}$, and decrement its g_{c^*} by impact. The resulting predicted distribution is then
$$\widehat{p}_c \;=\; \frac{\#\{\, p \in \mathcal{P} : p = c\}}{B} \times 100\%. \tag{4}$$

After defining the unique measures for inference, the remainder of the scheme mostly align with LLG. The concise step-by-step summary is given in Alg. 1.

Algorithm 1 Label distribution inference.

Input: Weight difference ΔW_x, batch size B, number of classes C, local iterations L;
Output: Predicted label distribution $\{\widehat{p}_c\}_{c=1}^{C}$;
Compute aggregate scores $g_c \leftarrow \sum_i \Delta W_{c,i}$ for all c;
Form candidate set $\mathcal{H}_1 = \{c : g_c > 0\}$;
Compute per-sample impact as described in Eq. (3);
Initialize multiset $\mathcal{P} \leftarrow \emptyset$;
for each $c \in \mathcal{H}_1$ **do**
    Add one sample of c into $\mathcal{P}$ and decrement g_c by impact;
end for
while $|\mathcal{P}| < B$ **do**
    $c^* = \arg\max_c g_c$;
    Add one sample of c^* into $\mathcal{P}$ and decrement g_{c^*} by impact;
end while
Compute $\widehat{p}_c$ as described in Eq. (4);

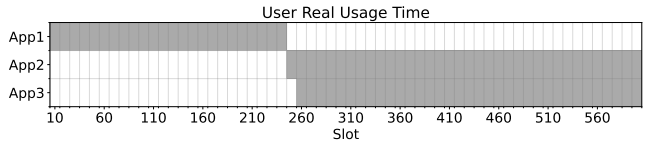
C. Non-Invasive User Profiling

Participants can be profiled once the application appearance list per slot for each client is obtained. However, to derive meaningful behavioral profiles, additional information is required, including the *APP-type weights* that quantify each application's contribution to different behavior types (e.g., productive, entertaining, casual), and the *APP slot extension rules* that account for temporal effects such as caching, which may cause certain applications to influence multiple adjacent time slots beyond their initial appearance.

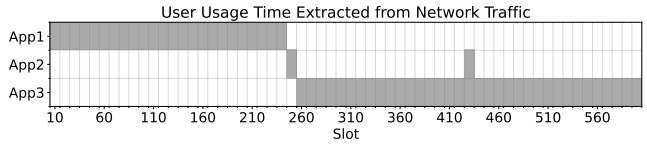
The APP-type weights define how representative an APP is in different profiles, e.g., productive, entertaining, and casual. This work assumes that each APP is associated with a set of values corresponding to these three properties. The major property, which best characterizes the APP's typical usage, is assigned the highest value, indicating that when a user engages with this application, their behavior is most aligned with this profile type. The remaining two properties are assigned lower values, where the lowest value reflects the profile type that is least represented when the application is used. For example, popular entertainment apps such as YouTube, Spotify, and Twitch are assigned entertainment as the major property. Since they have little relation to productivity, they receive the lowest score on the productive profile. APPs such as Outlook and Zoom are tagged as productive by default, earning their minimum weight in the entertaining dimension. In the casual dimension, all of these applications are given a

moderate weight, reflecting that users may engage with them in an informal or background capacity regardless of their primary purpose. Note that the weights and profiles can be adjusted based on user preference and QoS needs. For better illustration, the scoring mechanism of these three properties in this work is derived from the application categories available in official app stores [19, 20], and is further refined by referencing the classification schemes used in ASUS Adaptive QoS [10, 21]. This ensures that the assigned profile tendencies are grounded in widely accepted categorizations of application usage behaviors.

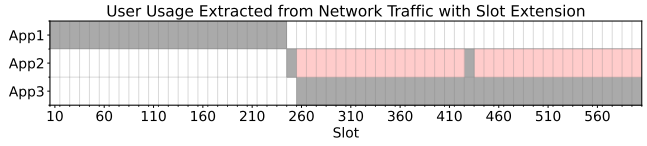
The APP slot extension is a mechanism developed to better estimate the APP usage by compensating missing traffic measurement due to caching. This mechanism extends the influence of an application's activity across multiple adjacent time slots, accounting for sustained network activity caused by background buffering. The rule of slot extension is derived from the observed traffic slot patterns of each application and its behavioral properties. To better illustrate the APP slot extension mechanism, the network traffic patterns of an example of one user with three APPs are shown in Fig. 3. In particular, Fig. 3(a) plots the ground-truth APP usage timeline; Fig. 3(b) plots the raw network traffic pattern from constant monitoring; and Fig. 3(c) plots the APP usage estimation after applying slot extension. Note that App 2 exhibits continuous real usage across many slots, yet the raw traffic measurement only marks two isolated slots as active, showing a strong evidence of background buffering behavior. By extending each detected slot of App 2 into its subsequent neighboring slots according to its observed buffering pattern, the adjusted timeline in Fig. 3(c) closely matches the true usage in Fig. 3(a).



(a) Ground-truth of user APP usage timeline.



(b) Raw network traffic pattern inferred APP usage timeline.



(c) APP usage timeline estimation with slot extension mechanism.

Fig. 3: Example of application usage, raw traffic pattern, and APP slot extension mechanism.

After defining the APP-type weights and the APP slot extension, the profiling mechanism computes the tendency toward each behavior type by aggregating the contributions of all APPs used for each client. Specifically, each APP class c has a recorded active time $\text{time}_x(c)$, determined by counting the number of time slots in which it appears (after applying slot extension) and multiplying by the slot duration (10 seconds in this work). Each application class c is associated with a vector of weights $\{w_c^{(\text{prod})}, w_c^{(\text{ent})}, w_c^{(\text{cas})}\}$, corresponding to its behavioral alignment with the productive, entertaining, and casual profiles. These weights reflect the degree to which the use of application c contributes to each behavior type. To quantify a client's behavioral tendency, we compute the following score for each profile type $t \in \{\text{prod}, \text{ent}, \text{cas}\}$:

$$\text{tendency}_x(t) = \sum_c \left(\text{time}_x(c) \cdot w_c^{(t)} \right),$$

where $\text{time}_x(c)$ is the total active time of application class c for client x, and $w_c^{(t)}$ is the corresponding weight for type t. The raw tendency scores are then normalized by the client's total active time total_time_x, defined as follows,

$$\text{total\_time}_x = 10 \times |\mathcal{S}_x|,$$

where $\mathcal{S}_x$ is the set of all unique time slots where any application was active for client x. This yields the normalized tendency as follows,

$$\text{tendency\_norm}_x(t) = \frac{\text{tendency}_x(t)}{\text{total\_time}_x}.$$

Finally, a profile vector can be computed as a percentage distribution across behavior types as follows,

$$\text{norm\_score}_x(t) = \frac{\text{tendency\_norm}_x(t)}{\sum_{t' \in \{\text{prod,ent,cas}\}} \text{tendency\_norm}_x(t')} \times 100.$$

These normalized scores represent the estimated behavioral profile of each client as a proportion of productive, entertaining, and casual usage tendencies.

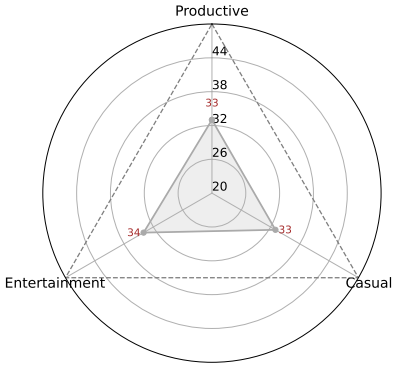


Fig. 4: An example of the user profile radar chart.

Note that the proposed user profiling scheme does not categorize a user into a specific profile. Rather, it scores a

user in all of the possible profiles. For better illustration, each user profile is depicted using a radar chart, as illustrated in Fig. 4. The weight assignments designed in Sec.V-B constrain each category score to lie between 20 and 50, the three axes form a triangle inscribed within the outer circle. The three category scores always sum to 100. A score of 50 in any one dimension indicates a purely single-profile user, whereas if any category score exceeds 40, it signals that the user leans strongly toward that usage type. Furthermore, roughly equal scores across all three dimensions imply that the user does not fit a single profile and exhibits a mixed usage pattern. Note that the actual scores can be normalized straightforwardly if needed. The overall profiling scheme is summarized in Alg. 2.

Algorithm 2 Client profiling with APP slot extensions.

Input: Labels per slot for clients, app weights, extension mechanism
Output: Client profiles

for each client x **do**
  Initialize: $\text{slots}_x(c) = 0$ for all classes, $\mathcal{S}_x = \emptyset$
  for each slot s with classes c **do**
    Increment $\text{slots}_x(c)$ for each class in s;
    Add s to $\mathcal{S}_x$;
  end for
  for each class c with extension mechanism **do**
    Extend appearances to allowed gaps;
    Update $\text{slots}_x(c)$ and $\mathcal{S}_x$ accordingly;
  end for
  $\text{time}_x(c) = 10 \times \text{slots}_x(c)$, $\text{total\_time}_x = 10 \times |\mathcal{S}_x|$;
  for each type t **do**
    $\text{tendency}_x(t) = \sum_c \text{time}_x(c) \times w_c^{(t)}$;
    $\text{tendency\_norm}_x(t) = \text{tendency}_x(t)/\text{total\_time}_x$;
  end for
  $\text{norm\_score}_x(t) = \frac{\text{tendency\_norm}_x(t)}{\sum_{t'};\text{tendency\_norm}_x(t')} \times 100$;
end for

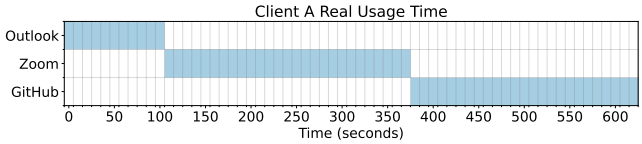
V. Evaluation Results

A. Chosen Dataset and User Profile Setup

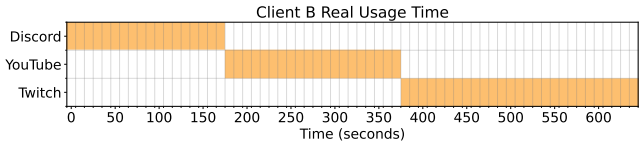
Three distinct types of APP usage behaviors are simulated across seven applications and are assigned to three clients participating in the federated learning process. The resulting network traffic generated under each simulated behavior is used as the dataset for each corresponding client. Seven APPs, i.e., YouTube, Outlook, Spotify, Zoom, Twitch, Discord, and GitHub, are included in the simulation.

- Client A uses Outlook, Zoom, and GitHub, simulating a productive-oriented profile;
- Client B uses Discord, YouTube, and Twitch, simulating an entertaining-oriented profile; and
- Client C uses YouTube, Outlook, and Spotify without a clear tendency toward a specific profile.

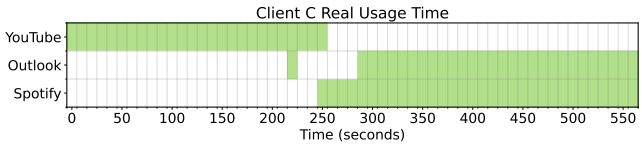
Client A's captured data spans 621 seconds, resulting in 63 slots; Client B's data spans 647 seconds, corresponding to 65 slots; and Client C's data covers 569 seconds, resulting in 57 slots. The detailed application usage timeline for each client is shown in Fig. 5. Each slot is 10 seconds in duration. During each slot, the collected network traffic is used to train the local model, and the updated model parameters are subsequently uploaded to the server at the end of the slot as part of the federated learning process. The network traffic data was collected on a smartphone using PCAPdroid [22], which is an open-source tool for capturing device-level traffic. Following a typical NTC implementation [2], the 20-byte packet header was stripped and each packet payload was truncated to 500 bytes, with the remaining data discarded.



(a) A productive-oriented profile for client A.



(b) An entertaining-oriented for client B.

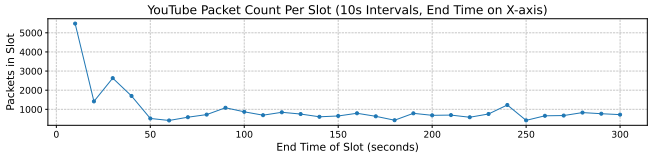


(c) No obvious profile for client C.

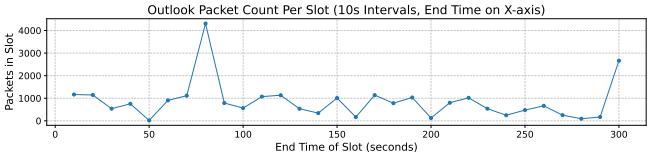
Fig. 5: Real applications usage timeline of three simulated clients with different network profiles.

B. Experiment Settings

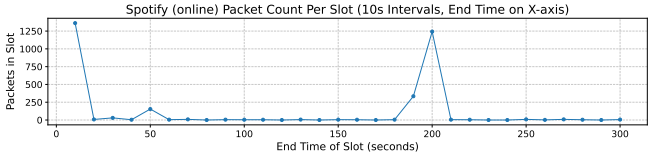
To model user behavior patterns from application usage, user intents are categorized into three types: *productive*, *entertaining*, and *casual*. Each application is evaluated for its relevance to these behavior types based on publicly available app descriptions from the application store (Apple APP store [20] and Google Play Store [19]) and the application type classifications from QoS categorization lists used in commercial routers (ASUS Adaptive QoS [10]). For each application, three behavior-type attributes are selected and assigned scores of 5, 3, and 2, corresponding to the *major*, *secondary*, and *minor* usage intents, respectively. These scores are then used to form a behavior contribution weight vector $\mathbf{w}_a = [w_p, w_e, w_c]$ such that the total contribution sums to 10. The major property of Outlook, GitHub, and Zoom is *productive*, and the remaining applications are classified as *entertaining*; no app is categorized as *casual* in this experi-



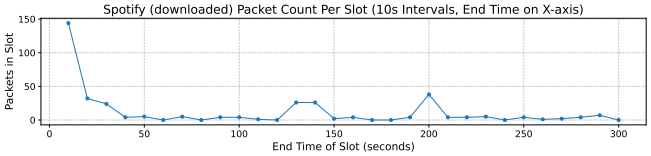
(a) YouTube slotted traffic pattern.



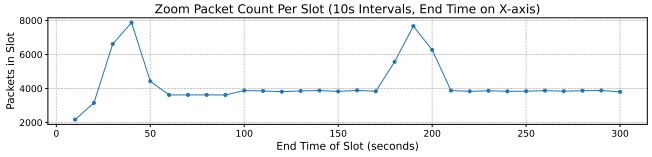
(b) Outlook slotted traffic pattern.



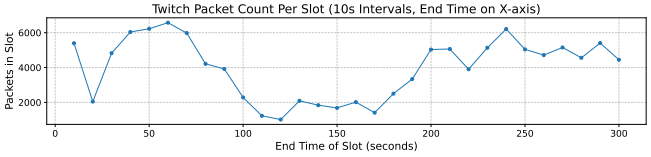
(c) Spotify online listening slotted traffic pattern.



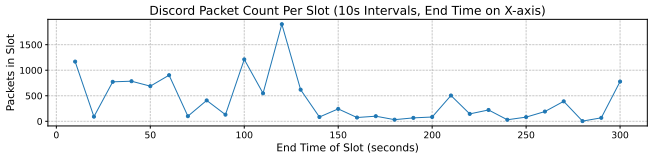
(d) Spotify downloaded listening slot traffic pattern



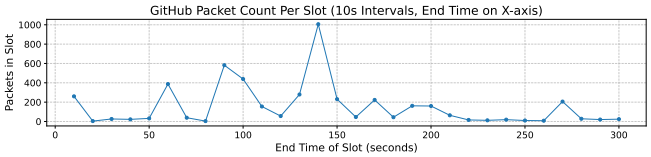
(e) Zoom slotted traffic pattern.



(f) Twitch slotted traffic pattern.



(g) Discord slotted traffic pattern.



(h) GitHub slotted traffic pattern.

Fig. 6: Slotted network traffic patterns for different APPs.

ment. The final weights for the seven applications are shown in Table I.

TABLE I: Behavior contribution weights for each application.

APP	**Casual** (w_c)	**Entertaining** (w_e)	**Productive** (w_p)
YouTube	3	5	2
Outlook	3	2	5
Spotify	3	5	2
Zoom	3	2	5
Twitch	3	5	2
Discord	3	5	2
GitHub	3	2	5

The compensations in the proposed APP slot extension mechanism are extracted from five-minute traffic monitoring of each APP separately. In particular, the traffic patterns were visualized by plotting the number of packets over time, where time was discretized into 10-second slots to align with the profiling mechanism, as shown in Fig 6. Based on these measurements, three APPs, i.e., Spotify, Discord, and GitHub, would benefit from the slot extension mechanism. The network pattern plots (Fig. 6(c) and Fig. 6(d)) demonstrate that Spotify tends to generate a burst of packets at the beginning of each track, while the remaining duration of the song exhibits minimal or no packet transmission. Even when playing downloaded music, the phone generates network traffic if it is connected to the internet. According to the 2024 Music Report by Chartmetric [23], the majority of tracks on Spotify have durations ranging from 2:51 to 3:09 minutes. Taking the average duration within this range, we estimate the playback time to be approximately 3 minutes, which corresponds to 18 slots under the 10-second slot granularity. As a result, the slot extension for Spotify is set to 17. In addition, both Discord (Fig. 6(g)) and GitHub (Fig. 6(h)) generate little network traffic during typical messaging and browsing. Most of the traffic occurs during the initial loading phase. According to [24], the average response time for users actively engaging with instant messaging APPs is approximately 30 seconds, which corresponds to 2-3 slots. Therefore, the extension length for Discord is set to 2. For GitHub, user behavior analysis shows that the average time spent on a page is approximately 54 seconds [25], leading to an extension of 5 slots. These extension mechanisms are applied during the inferred timeline construction to ensure alignment with realistic APP behavior.

C. Evaluation Results

Fig. 7, Fig. 8, and Fig. 9 demonstrate three views of our simulated clients' activities, including the APP usage timelines extracted from raw traffic data, the usage timeline inferred by

pure label inference approach, and the label inference approach with slot-extension mechanism. Comparison with actual usage (Figure 5) reveals that some apps generate network traffic The chosen lightweight FL-NTC achieves over 90% classification accuracy after convergence, though it typically requires several update rounds to adapt when users switch APPs, causing a period of precision reduction. The weight-based inference method typically neglects APPs contributing less than 10% of packet volume, an inaccuracy that is allowed to filter background traffic and transient usage. The final inferred timeline, particularly after implementing the slot-extension mechanism, corresponds more accurately with actual user behavior, successfully capturing dominant APP usage while suppressing insignificant traffic and idle-period communications.

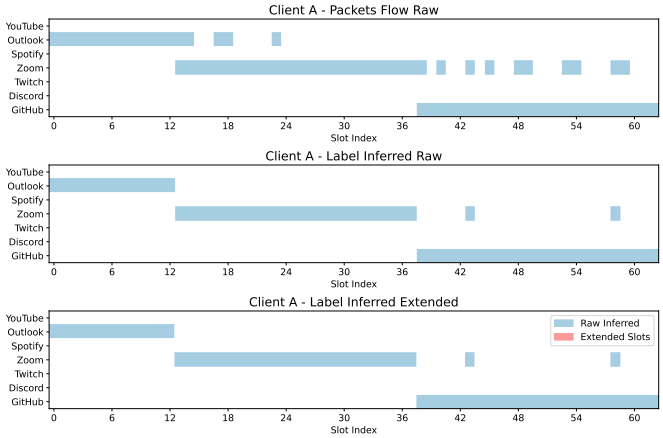


Fig. 7: APP timelines of client A from different approaches.

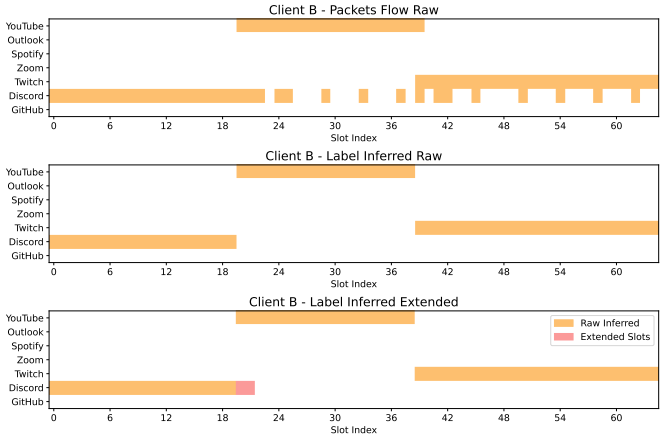


Fig. 8: APP timelines of client B from different approaches.

Fig. 10, Fig. 11, and Fig. 12 present radar charts for each of the three clients, generated from (1) ground truth, (2) raw traffic monitoring, and (3) the proposed method. Since each APP's behavior contribution weight allocates five points to major intents and two points to minor intents across ten time slots, the maximum attainable score for any profile dimension is fifty and the minimum is twenty. As the results show, both

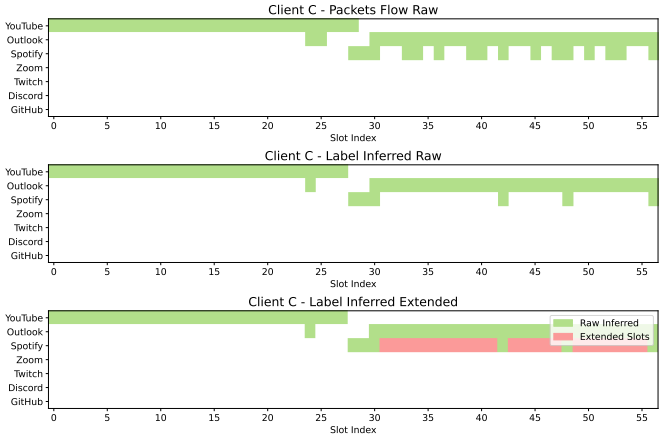
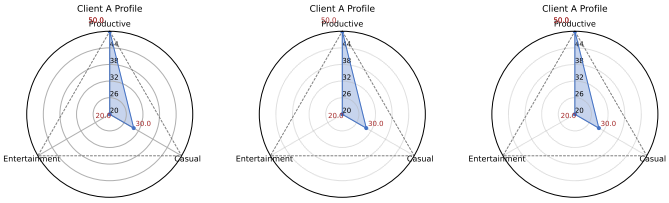


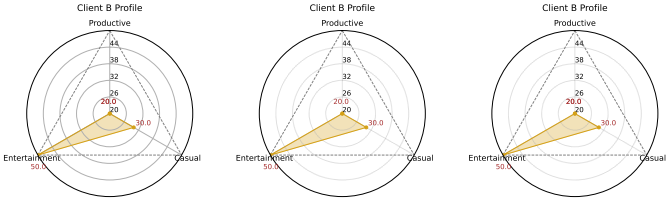
Fig. 9: APP timelines of client C from different approaches.

clients A and B exhibit a single dominant profile with the three respective radar charts being nearly identical. Meanwhile, the inferred flows for client C after slot-extension (productive 29.5, entertainment 40.5, and casual 30) align more closely with the ground truth (productive 30, entertainment 40, and casual 30), demonstrating the precision of our slot-extension mechanism.



(a) Ground truth. (b) Raw traffic. (c) Proposed method.

Fig. 10: Profiling results of user A from different approaches.

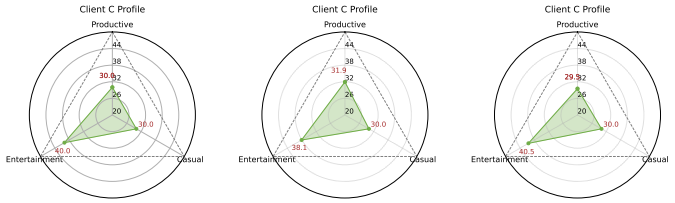


(a) Ground truth. (b) Raw traffic. (c) Proposed method.

Fig. 11: Profiling results of user B from different approaches.

D. More Discussions

The proposed framework is a coarse profiling based on the user's network behavior. While the NTC accuracy in the framework may be low, it nonetheless proves effective in profiling. Importantly, our profiling process does not require access to raw packet payloads or detailed flow-level statistics; instead, it relies on the updates transmitted during federated



(a) Ground truth. (b) Raw traffic. (c) Proposed method.

Fig. 12: Profiling results of user C from different approaches.

training. By operating solely on these model-update metadata, the framework limits exposure to sensitive user data, mitigating the privacy risks associated with traditional adaptive QoS systems that depend on deep packet inspection and flow monitoring. Therefore, the proposed approach preserves user anonymity and confidentiality. However, only a limited set of profile categories and applications were tested, which constrains its generalizability.

VI. Conclusion and Future works

In this work, we proposed a non-invasive networking user profiling scheme by exploiting the updated model parameters uploaded by each client in an FL-based NTC framework. Without accessing raw packet payloads, our method extracts coarse network traffic patterns and assigns behavioral profiles, such as productivity-oriented, entertainment-oriented, and casual usage patterns directly from the parameter updates contributed during federated training. Evaluation results demonstrated the efficacy of the proposed method in network user profiling, even with a lightweight and moderately-accurate AI-NTC. In future work, we will refine the granularity of behavior categories. Moreover, we aim to validate and scale our solution in real-world network environments, to assess its operational performance, scalability, and impact on personalized QoS management.

Acknowledgment

This project is partially supported by the U.S. National Science Foundation under grant CNS-2344341.

References

[1] F. Pacheco, E. Exposito, M. Gineste, C. Baudoin, and J. Aguilar, "Towards the deployment of machine learning solutions in network traffic classification: A systematic survey," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 2, pp. 1988–2014, 2019.

[2] J. Zhang, F. Li, and F. Ye, "Sustaining the high performance of ai-based network traffic classification models," *IEEE/ACM Transactions on Networking*, vol. 31, no. 2, pp. 816–827, 2022.

[3] H. Mun and Y. Lee, "Internet traffic classification with federated learning," *Electronics*, vol. 10, no. 1, 2021. [Online]. Available: https://www.mdpi.com/2079-9292/10/1/27

[4] Z. Jin, K. Duan, C. Chen, M. He, S. Jiang, and H. Xue, "Fedetc: Encrypted traffic classification based on federated learning," *Heliyon*, vol. 10, no. 16, p. e35962, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405844024119937

[5] K. Ibrar, F. Palmieri, P. Fusco, and M. Ficco, "Cross-model federated learning-based network traffic classification," in *Advanced Information Networking and Applications*, L. Barolli, Ed. Cham: Springer Nature Switzerland, 2025, pp. 250–259.

[6] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," 2017. [Online]. Available: https://arxiv.org/abs/1610.05820

[7] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," 2019. [Online]. Available: https://arxiv.org/abs/1906.08935

[8] A. Wainakh, T. Müßig, T. Grube, and M. Mühlhäuser, "Label leakage from gradients in distributed machine learning," in *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, 2021, pp. 1–4.

[9] A. Wainakh, F. Ventola, T. Müßig, J. Keim, C. Garcia Cordero, E. Zimmer, T. Grube, K. Kersting, and M. Mühlhäuser, "User-level label leakage from gradients in federated learning," *Proceedings on Privacy Enhancing Technologies*, vol. 2022, pp. 227–244, 04 2022.

[10] ASUS Computer Inc., "Stay at home easily – work from home with smooth wifi," https://www.asus.com/event/stay-at-home/Networking/fi/, 2025.

[11] Huawei Technologies Co., Ltd., "Adaptive qos configuration case and usage description," https://info.support.huawei.com/hedex/api/pages/adaptive-qos-config-case-and-usage-description, 2023, accessed: 2025-05-05.

[12] A. Azab, M. Khasawneh, S. Alrabaee, K.-K. R. Choo, and M. Sarsour, "Network traffic classification: Techniques, datasets, and challenges," *Digital Communications and Networks*, vol. 10, no. 3, pp. 676–692, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352864822001845

[13] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1322–1333. [Online]. Available: https://doi.org/10.1145/2810103.2813677

[14] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, "Property inference attacks on fully connected neural networks using permutation invariant representations," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 619–633. [Online]. Available: https://doi.org/10.1145/3243734.3243834

[15] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE Symposium on Security and Privacy (SP)*, 2019, pp. 739–753.

[16] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "Rfc2475: An architecture for differentiated service," USA, 1998.

[17] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *The IEEE Conference on Local Computer Networks 30th Anniversary (LCN'05)l*, 2005, pp. 250–257.

[18] S. Degli Esposti, V. Pavone, and E. Santiago-Gómez, "Aligning security and privacy: The case of deep packet inspection," in *Surveillance, Privacy and Security*. Routledge, 2017, pp. 71–90.

[19] Google LLC, "Google play," https://play.google.com/store, 2025.

[20] Apple Inc., "App store," https://www.apple.com/app-store/, 2025.

[21] dave14305, "Flexqos - flexible qos enhancement script for adaptive qos on asuswrt-merlin," https://github.com/dave14305/FlexQoS, 2025, accessed: 2025-04-28.

[22] E. Fusillo, "Pcapdroid: No-root network monitor, firewall and pcap dumper for android," https://github.com/emanuele-f/PCAPdroid, 2020, accessed: 2025-04-29.

[23] Chartmetric, "Chartmetric: A year in music 2024," 2024, accessed: 2025-05-03. [Online]. Available: https://reports.chartmetric.com/2024/chartmetric-year-in-music-2024?ref=hmc.chartmetric.com

[24] D. Avrahami, S. R. Fussell, and S. E. Hudson, "Im waiting: timing and responsiveness in semi-synchronous communication," in *Proceedings of the 2008 ACM Conference on Computer Supported Cooperative Work*, ser. CSCW '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 285–294. [Online]. Available: https://doi.org/10.1145/1460563.1460610

[25] N. Tambe and A. Jain. (2024) Top website statistics for 2024. Forbes Advisor. Accessed: 2025-05-03. [Online]. Available: https://www.forbes.com/advisor/in/business/software/website-statistics