Notes on revision made to manuscript Paper #TMC-2023-02-0157

The authors would like to thank the editor and reviewers for their constructive comments and suggestions that have helped improve the quality of this manuscript. The manuscript has undergone a thorough revision according to the editor and reviewers' comments. Please see below our responses. For the reviewers' convenience, we have highlighted significant changes in the revised manuscript in blue.

Response to the reviewers

Reviewer 1

Reviewer Comment 1.1 — My main concern with RingSFL is the threat of model inversion attacks. Section 3.6.1 was a good addition to discuss the threat of gradient-based reconstruction attacks. However, if an adversary has some model access, either white-box or black-box, then there is a threat that the feature maps shared during forward propagation could be used to reconstruct the original data [1]. This would especially be a threat when a client's propagation length is 1, meaning an adversary may only need to learn a linear mapping to reconstruct the original data.

I am curious about the authors thoughts on these kinds of attacks, and would encourage them to at least add a subsection to discuss this topic. Under what conditions would RingSFL be vulnerable to such attacks? If the authors feel RingSFL can defend against such attacks under certain circumstances, then it would be nice to have a plot similar to Figure 5 or even experiments such as those shown in Figure 16 showing the success or failure of model inversion attacks.

Reply: Thank you so much for your constructive comments. We have added a special subsection to discuss the privacy issues arising from D2D communication between clients. As discussed in [1], there are three main settings for model inversion attacks (inversion of the model based on feature maps to reconstruct the input data): **white-box setting**, **black-box setting**, and **query-free setting**. We discuss each of these three settings and explore under which settings RingSFL can defend against model inversion attacks, and under which settings it is vulnerable.

White-box Setting: In this setting, there are two primary prerequisites for a successful attack: 1) the attacker needs to know the model structure and parameters of the targeted client; 2) the attacker needs to obtain feature maps of local dataset samples from the targeted client. Once these two prerequisites are met, the attacker will first randomly generate dummy data samples and input them into a model with the same parameters as the targeted client to obtain feature maps of these dummy data samples. Subsequently, the attacker optimizes the generated dummy data samples by minimizing the distance between the feature maps of the dummy data samples and those of real samples (obtained from the targeted client), eventually recovering the real data samples.

In white-box attacks, the attacker has complete knowledge of the target model and can use this information to optimize their attack strategy. However, it also requires a more thorough understanding of the target model and its parameters, making it potentially more difficult to execute. For honest clients participating in training or attackers without eavesdropping capabilities in RingSFL, there is no privacy leakage issue in this setting because they cannot access the local model parameters of other clients. However, RingSFL still has limitations for attackers with strong eavesdropping capabilities and may not effectively resist white-box attacks. Attackers can conduct white-box attack by eavesdropping on the communication link between the target client and the server to obtain the local model parameters of the client, and by eavesdropping on the D2D communication data between the client and its neighboring clients to obtain the feature maps of real samples in the client's training dataset.

Black-box Setting: In this setting, a successful attack requires that the attacker can input locally generated dummy data samples into the local model of the attacked client for inference and obtain the feature maps of the inference output. Once this prerequisite is met, the attacker randomly generates dummy data samples and inputs them into the local model of the attacked client for inference, obtaining the feature maps of the dummy data sample outputs. Subsequently, the attacker employs these dummy data samples and their corresponding feature maps as the dataset to train a model that can replicate the behavior of the attacked client's local model. Ultimately, through the use of this well-trained model, the model inversion attack can be executed employing the same methodology as in the white box setting.

However, during the training process of RingSFL, only the local datasets of participating clients are accessed for training, and there is no transmission or receipt of any external datasets. Consequently, the prerequisites for a successful attack cannot be met, leading to RingSFL's ability to effectively resist black-box attacks.

Query-free Setting: In this setting, the prerequisite for a successful attack is that the attacker must possess knowledge of the local dataset distribution of the target client they intend to attack. This setting assumes that the attacker is unable to directly query the target model or access its internal parameters. Instead, the attacker relies on generating dummy data samples that match the local data distribution of the target client's dataset. The attacker then trains a model using these generated samples to mimic the behavior of the target client's local model. Finally, based on this trained model, the attacker performs model inversion attack using the same method as in the white-box setting.

However, during the training process of RingSFL, participating clients do not transfer their local datasets nor the distribution information of their local datasets. As a result, the necessary prerequisites for a successful attack cannot be met, allowing RingSFL to effectively resist Query-Free attacks.

Reviewer Comment 1.2 — Section 3.5 is a bit confusing to read when comparing with Figure 4. The text states that an overlapping layer is defined as a layer where multiple clients contribute gradients. However, the figure shows an overlapping layer only being present at one client. Do you say a client is contributing gradients to the layer if that client is the one that calculated the loss and started backpropagation? Or am I misunderstanding?

Reply: Thank you so much for your constructive comments. We have redesigned Fig. 4 and explained in detail our definition of overlapping layer.

Difinition of Overlapping Layer: *If a neural layer lies within the propagation range of multiple clients, we call that layer an overlapping layer.* As illustrated in Figure. 1, since layers 3-4 of the model for client 1 lie within the propagation range of both client 0 and client 1 (with both red and blue arrows passing through), these two layers are overlapping layers; whereas there is no layer on client 0 that lies within the propagation range of both clients (no layer has both red and blue arrows passing through), so there is no overlapping layer on client 0.

Reviewer Comment 1.3 — In Equation 7, since $\mathcal{U}$ is a set, it would make more sense to use $|\mathcal{U}|$ to denote the cardinality of the set, the norm notation is a bit confusing.

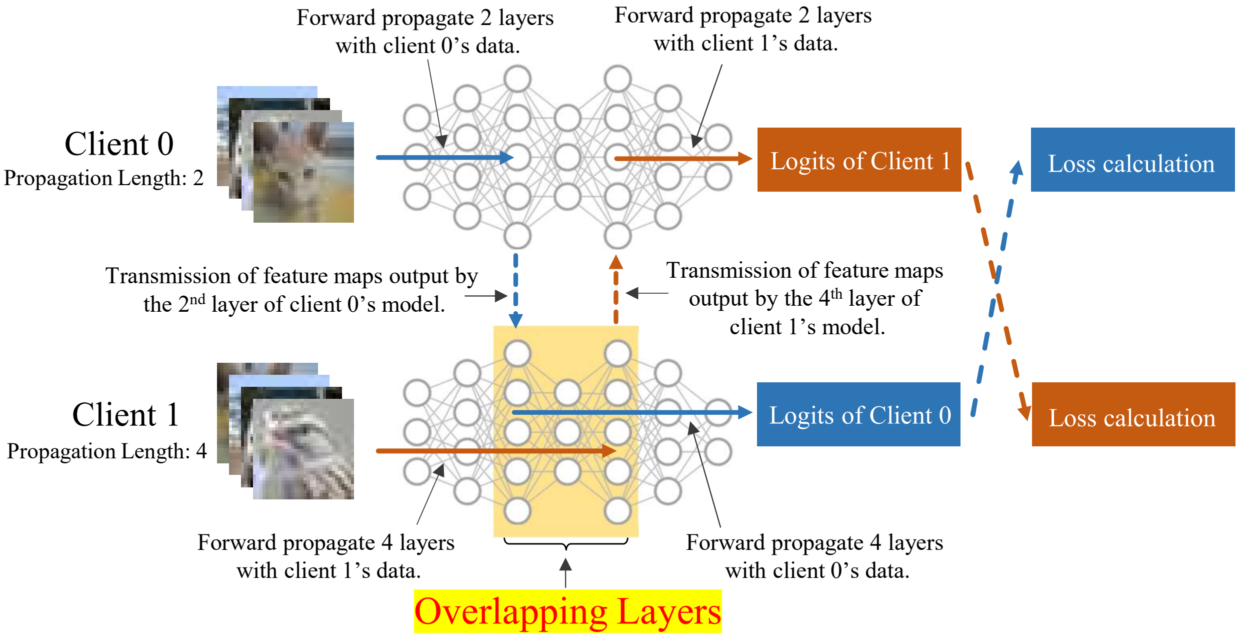

Forward propagate 2 layers with client 0's data.
Forward propagate 2 layers with client 1's data.
Client 0
Propagation Length: 2
Logits of Client 1
Loss calculation
Transmission of feature maps output by the 2nd layer of client 0's model.
Transmission of feature maps output by the 4th layer of client 1's model.
Client 1
Propagation Length: 4
Logits of Client 0
Loss calculation
Forward propagate 4 layers with client 1's data.
Forward propagate 4 layers with client 0's data.
Overlapping Layers

Figure 1: RingSFL training over 2 clients; $L_0 : L_1 = 2 : 4$.

Reply: Thank you so much for your constructive comments. We have modified equation 7 by using $|\mathcal{U}|$ to denote the cardinality of the set $\mathcal{U}$. Equation 7 is now rewritten as

$$\mathcal{W}^t_{i,(j)} = \mathcal{W}^t_{i,(j)} - \eta|\mathcal{U}_{i,(j)}| \sum_{k\in\mathcal{U}_{i,(j)}} a_k \boldsymbol{g}^t_{k,(j)}. \tag{1}$$

Reviewer Comment 1.4 — The claim that the eavesdropper does not know the propagation lengths seems strong. If the eavesdropper can access gradients, would it not be able to intercept messages from the server and learn the propagation length for each client?

Reply: Thank you so much for your constructive comments. Our claim that the eavesdropper is unaware of the propagation length is motivated by two main considerations

- An eavesdropper to eavesdrop on all links in the system usually requires the eavesdropper to have strong capabilities (e.g., an eavesdropper with a receiver that receives a large range of spectrum to enable it to eavesdrop on all client channels, or with multiple eavesdroppers to form an eavesdropping network to eavesdrop on all clients).
- When configuring the propagation length, the amount of data to be transmitted to the client is so small (only one number) that we can encrypt it with a very sophisticated encryption algorithm to ensure its security while guaranteeing a tolerable computational overhead for encryption and decryption. Whereas, if we encrypt the model parameters directly for transmission, it may lead to excessive computational overhead for encryption and decryption on the client side (e.g., resnet18 has tens of millions of parameters and vgg16 has hundreds of millions of parameters).

Table 1: Top-1 Accuracy (%) of Each Model under Different Algorithms.

	ResNet18 (IID / Non-IID)	**VGG16 (IID / Non-IID)**	**AlexNet (IID / Non-IID)**	**LeNet-5 (IID / Non-IID)**
RingSFLv1	82.35 ± 0.36 / $\underline{48.30 \pm 0.57}$	$\underline{79.30 \pm 0.20}$ / $\underline{40.35 \pm 0.99}$	$\underline{98.83 \pm 0.11}$ / 89.58 ± 0.55	98.82 ± 0.19 / 94.34 ± 0.56
RingSFLv2	**84.57 ± 0.17** / **56.80 ± 0.78**	**84.33 ± 0.10** / **41.26 ± 1.29**	**99.13 ± 0.07** / $\underline{94.31 \pm 0.88}$	**99.10 ± 0.04** / $\underline{95.75 \pm 0.73}$
SplitFed	75.92 ± 0.51 / 30.16 ± 4.49	72.86 ± 0.62 / 28.17 ± 2.15	98.76 ± 0.09 / 84.00 ± 4.39	98.74 ± 0.24 / 93.64 ± 0.70
vanilla FL	78.93 ± 0.27 / 48.02 ± 1.28	77.02 ± 0.34 / 39.52 ± 0.81	98.81 ± 0.07 / 91.60 ± 1.14	$\underline{98.84 \pm 0.08}$ / 94.77 ± 0.29
vanilla SL	$\underline{83.41 \pm 0.44}$ / 26.96 ± 3.58	78.50 ± 0.69 / 35.33 ± 1.29	98.69 ± 0.10 / **98.84 ± 0.08**	98.80 ± 0.14 / **98.86 ± 0.09**

Reviewer Comment 1.5 — Table 3: it would be nice for the reader if the best accuracy for each model and data distribution was in bold.

Reply: Thank you so much for your constructive comments. As shown in Table 1, we modified Table 3 by redrawing it as a three-line table with the best accuracy marked in bold, and the secondary marked in underline.

Reviewer Comment 1.6 — Figures 7, 9, and 10: there appears to be a plotting error, some lines from the box extend out to the left.

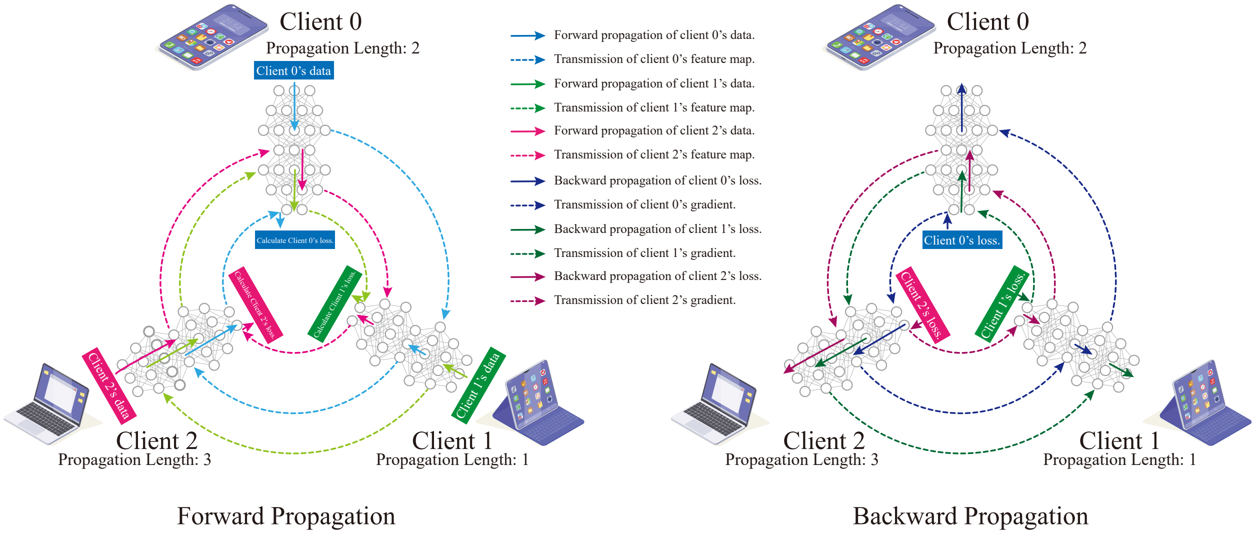
Reply: Thank you so much for your constructive comments. We are sorry for these plotting errors, it may be caused by the incompatibility of the svg images we uploaded with your pdf reader, we have replaced these images with pdf format and hope it will solve your problem.

Reviewer 2

Reviewer Comment 2.1 — Figure describing the overall algorithm: First of all, a detailed figure showing forward/backward propagation process of RingSL is necessary. Without this, a reader has to understand the detailed algorithm with only equations, which is very complicated. Authors may consider the case with 3 users and provide details on the forward/backward propagation process. This figure may also help the readers to better understand why there are overlapping updates for each layer.

Reply: Thank you so much for your constructive comments.

Reviewer Comment 2.2 — Fig. 4: Related to the comment above, if I understood the algorithm correctly the example of Fig. 4 has significant errors. In Fig. 4, although the number of (input & output & hidden layers) are 3, the number of fully connected layer is 2. Hence, the case with $L_0 = 1$ and $L_1 = 2$ doesn't make sense because the workload should be defined as the number of fully connected layers, which determines the amount of workload. In Fig. 4, the data of client 0 skips the first fully connected layer and directly becomes the input of the second fully connected layer, which doesn't make sense if the input layer size and the hidden layer size is different. Similarly, the data of client 1 skips the second fully connected layer. Moreover, the concept of "overlapping layer" should also be the "overlapping fully connected layer" (in case of MLP, and "overlapping convolutional layer" in case of CNNs) instead of "overlapping hidden layer". The authors should make all these points clearer and better describe the overall algorithm with more efforts.


Client 0
Propagation Length: 2
Client 0's data
Calculate Client 0's loss.
Calculate Client 2's loss.
Calculate Client 1's loss.
Client 2's data
Client 1's data
Client 2
Propagation Length: 3
Client 1
Propagation Length: 1
Forward propagation of client 0's data.
Transmission of client 0's feature map.
Forward propagation of client 1's data.
Transmission of client 1's feature map.
Forward propagation of client 2's data.
Transmission of client 2's feature map.
Backward propagation of client 0's loss.
Transmission of client 0's gradient.
Backward propagation of client 1's loss.
Transmission of client 1's gradient.
Backward propagation of client 2's loss.
Transmission of client 2's gradient.
Client 0
Propagation Length: 2
Client 0's loss.
Client 2's loss.
Client 1's loss.
Client 2
Propagation Length: 3
Client 1
Propagation Length: 1

Forward Propagation

Backward Propagation

Reply: Thank you so much for your constructive comments.

Reviewer Comment 2.3 — Fig. 1: Fig. 1 is also somewhat confusing. What does the colored hidden layers mean in Fig. 1? If these represent the propagation length, it doesn't make sense since the sum of the portion of responsible layers should become 1, which is not the case in Fig. 1.

Reply: Thank you so much for your constructive comments.

Reviewer Comment 2.4 — The term split: I am also not sure if the term "split" is appropriately used in many sentences (e.g., By properly splitting the model and allocating it to heterogeneous clients), because all clients have the full model during training, not the split model; each client has different number of propagation layers, but eventually, most of the clients would update the whole model by repeating forward/backward process using data of all clients in the system.

Reply: Thank you so much for your constructive comments.

Reviewer Comment 2.5 — Privacy for sharing features among clients: Although RingSL has better privacy against the server by sending the blended models, it requires sharing the features to the adjacent clients, which can also cause privacy issues. The authors may need to discuss about this.

Reply: Thank you so much for your constructive comments.

Reviewer Comment 2.6 — Multiple mini-batch: In federated learning, multiple mini-batch updates are performed before sending the updated model to the server to handle the communication bottleneck. However, in this scheme, to perform another mini-batch update, the full communication round among clients are required for the forward/backward propagation process.

Reply: Thank you so much for your constructive comments.

Reviewer Comment 2.7 — Finally, I am not very convinced with the ring-shaped architecture itself for handling the straggler issue. For example, when a specific client becomes outage due to the battery issue as described in the introduction, it results in significant bottleneck since every forward/backward signals cannot pass that client. In contrast, in conventional FL with a single server, it is acceptable to have some stragglers in each round because one can ignore them.

Reply: Thank you so much for your constructive comments.

Reviewer 3

Reviewer Comment 3.1 — Does a ring structure leads to exposure of client identities (as clients communicate with each other directly).

Reply: Thank you so much for your constructive comments.

Reviewer Comment 3.2 — Could distributing training process based on available lead to privacy problems for machine configuration? (e.g., say that the client is aware of model architecture, but not parameters, could still infer the amount of resources that the remaining clients have based on the propagation length assigned to itself).

Reply: Thank you so much for your constructive comments.

Reviewer Comment 3.3 — The author listed a series of related works from different perspective and then say that RingSFL improves on existing techniques. Could you discuss deeper into how does the approach compare to existing techniques? I noticed that there are "Benefits and limitations" subsections, are these intended for this purpose? If they are then the authors could point out specific design decisions that corresponds to the differences.

Reply: Thank you so much for your constructive comments.

Reviewer Comment 3.4 — How does RingSFL handles failure of client? Does the model get distributed?

Reply: Thank you so much for your constructive comments.

Reviewer Comment 3.5 — Overlapping layers — Is it an explicit choice that different components of the network converges differently? I understand the learning rate could be adaptive when overlapping layers occur, but is there a more balanced approach so that the model has the same performance as the non-distributed one.

Reply: Thank you so much for your constructive comments.

Reviewer Comment 3.6 — Could a client with significant amount of resources be able to infer propagation length of all other clients.

Reply: Thank you so much for your constructive comments.

References

[1] Z. He, T. Zhang, and R. B. Lee, “Model inversion attacks against collaborative inference,” in *Proceedings of the 35th Annual Computer Security Applications Conference*, 2019, pp. 148–162.