ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

UNVEILING THE WIRELESS NETWORK LIMITATIONS IN FEDERATED LEARNING

M.Sc. THESIS

Mümtaz Cem ERİŞ

Department of Computer Engineering

Computer Engineering Programme

May 2022

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL

UNVEILING THE WIRELESS NETWORK LIMITATIONS IN FEDERATED LEARNING

M.Sc. THESIS

Mümtaz Cem ERİŞ
(504191531)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Prof. Dr. Sema Fatma OKTUĞ
Thesis Co-Advisor: Assoc. Prof. Dr. Burak KANTARCI

May 2022

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ

KABLOSUZ İNTERNET AĞLARINDAKİ KISITLARIN FEDERE ÖĞRENMEYE OLAN ETKİLERİNİN ORTAYA ÇIKARILMASI

YÜKSEK LİSANS TEZİ

Mümtaz Cem ERİŞ
(504191531)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Sema Fatma OKTUĞ
Eş Danışmanı: Assoc. Prof. Dr. Burak KANTARCI

MAYIS 2022

Mümtaz Cem ERİŞ, a M.Sc. student of ITU Graduate School with student ID 504191531, successfully defended the thesis entitled “UNVEILING THE WIRELESS NETWORK LIMITATIONS IN FEDERATED LEARNING”, which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Prof. Dr. Sema Fatma OKTUĞ** Istanbul Technical University	....
Co-advisor :	**Assoc. Prof. Dr. Burak KANTARCI** University of Ottawa	....
Jury Members :	**Prof. Dr. Berk CANBERK** Istanbul Technical University	....
	Asst. Prof. Dr. Gökhan SEÇİNTİ Istanbul Technical University	....
	Prof. Dr. Nelson FONSECA University of Campinas	....

Date of Submission : 29 March 2022
Date of Defense : 27 May 2022

To my mother,
For her immense support.

FOREWORD

Firstly, I would like to express my gratitude to my supervisors Prof. Sema Fatma OKTUĞ and Assoc. Prof. Burak KANTARCI for all of their support and guidance during my M.Sc. thesis.

Finally, I would like to thank Compute Canada (www.computecanada.ca) for supporting my studies.

May 2022

Mümtaz Cem ERİŞ

TABLE OF CONTENTS

Page

FOREWORD .... **ix**
TABLE OF CONTENTS .... **xi**
ABBREVIATIONS .... **xiii**
SYMBOLS .... **xv**
LIST OF TABLES .... **xvii**
LIST OF FIGURES .... **xix**
SUMMARY .... **xxi**
ÖZET .... **xxv**
1. INTRODUCTION **1**
  1.1 What is Federated Learning? 1
  1.2 The Importance of Federated Learning 1
  1.3 Federated Learning in a Network Sense 2
  1.4 Contribution 3
  1.5 Organization of the Thesis 4
2. RELATED WORK **5**
  2.1 Cellular Networks 5
  2.2 Wireless Networks 8
  2.3 Unreliable Networks 10
  2.4 Ad-hoc Networks 11
  2.5 Smart Cities 11
  2.6 Wide Area Networks 12
  2.7 AR/VR and Vehicular Networks 13
3. FEDERATED LEARNING NETWORK SIMULATION MODEL **15**
  3.1 Federated Learning Scenario 15
  3.2 Network Topology 15
    3.2.1 Background Traffic Generator Node 15
    3.2.2 Communication Rounds 16
    3.2.3 Transport Layer Protocol 16
  3.3 Test Environment: NS3-AI 17
  3.4 Simulation Model 17
    3.4.1 Sending Fragmented Gradient Update UDP Packets 17
    3.4.2 Initial ns3-ai Integration 18
    3.4.3 Dumbbell Topology with 5 Nodes 19
    3.4.4 Dumbbell Topology with 8 Nodes 22
    3.4.5 Dumbbell Topology with 9 Nodes with Background Traffic 22
    3.4.6 Dumbbell Topology with 104 Nodes 23
    3.4.7 Simulation with Constant Traffic and non-IID Setting 24
    3.4.8 Setting Seed Value for the Simulation 25

3.4.9 Simulation with Poisson Distribution Based Background Traffic 28
3.5 Final Simulation Model 30
4. SIMULATION RESULTS 33
4.1 Reproducibility on Average Cross Entropy Error and Accuracy 33
4.2 Maximum Error Percentage 34
4.3 Average Success Percentage 34
4.4 Average Accuracy 37
4.5 Confidence Intervals 37
4.6 Network Metrics 38
5. ADAPTIVE PARTICIPATION FEDERATED LEARNING 41
5.1 Adjustment of Both Communication Round Limit and The Participation Rate41
5.2 Setting Maximum Number of Clients in Communication Round 42
6. CONCLUSION 45
REFERENCES 47
APPENDICES 53
APPENDIX A.1 55
APPENDIX B.1 57
CURRICULUM VITAE 61

ABBREVIATIONS

QoS **:** Quality of Service
NS3 **:** Network Simulator 3
UDP **:** User Datagram Protocol
FL **:** Federated Learning
ML **:** Machine Learning
QoE **:** Quality of Experience
AR **:** Augmented Reality
VR **:** Virtual Reality
SGD **:** Stochastic Gradient Descent
CNN **:** Convolutional Neural Networks
IID **:** Independent and Identically Distribution
FedAvg **:** Federated Averaging
MNIST **:** Modified National Institute of Standards and Technology database
LSTM **:** Long Short-Term Memory

SYMBOLS

B : Local batch size
C : Selected clients
E : Gradient descent steps
T : Time
N : Number of clients

LIST OF TABLES

Page

Table 3.1 : Hyperparameters of the ns3-ai FL Simulation.................................... **32**
Table 4.1 : Set of technical expressions for this Section....................................... **35**
Table 4.2 : Network statistics are gathered concerning the interarrival time of the background traffic. **39**

LIST OF FIGURES

Page

Figure 2.1 : Samples from CIFAR-10 are shown as colored images. Gray images are taken from Fashion-MNIST dataset [1]. **8**
Figure 2.2 : Emulated network topology can be observed in [2]. The dumbbell and multi path topologies where there are routers as bottlenecks show similarities to the simulation model that is described in Section 3. .... **10**
Figure 2.3 : Handwritten digits that are taken from MNIST dataset. Proposed algorithm of [3] classification results can be observed. **13**
Figure 3.1 : ns3-ai Federated Learning Simulation Network Topology **16**
Figure 3.2 : ns3-ai Framework. **18**
Figure 3.3 : Initial topology. **18**
Figure 3.4 : Fragment generation and scheduling. **19**
Figure 3.5 : Gradient update transmission in the first communication round. **20**
Figure 3.6 : Dumbbell topology with 5 nodes. **20**
Figure 3.7 : Python and NS3 processes side by side. **21**
Figure 3.8 : Python and NS3 processes first and second communication round... **21**
Figure 3.9 : Dumbbell topology with 8 nodes in which client with ID 3 is transmitting. **22**
Figure 3.10 : Client transmission plot in discrete time intervals. **23**
Figure 3.11 : Received packets by server is plotted. **24**
Figure 3.12 : Accuracy values of both clients and server are given with respect to virtual time. **25**
Figure 3.13 : The packet byte count over time plot of the background traffic generator. **26**
Figure 3.14 : Runs with and without background traffic side by side. **26**
Figure 3.15 : Runs with and without background traffic side by side including 100 clients and also packet drop histogram can be observed. **27**
Figure 3.16 : Non-IID data distribution in which a client only gets 600 samples which represents only two digits from MNIST dataset. **27**
Figure 3.17 : With and without the background traffic experiment with Non-IID data distribution. **27**
Figure 3.18 : Different seed values and accuracy results under constant background traffic. **28**
Figure 3.19 : Both average maximum and cross entropy error under fixed background traffic for the seed 436 .... **29**
Figure 3.20 : Average success percentages of 10 runs for each arrival rate. **30**
Figure 3.21 : Average accuracy of 10 runs for each arrival rate. **31**
Figure 4.1 : Accuracy of first and second run concerning simulation time that is supplied with same seed value. **33**

Figure 4.2 : Average cross entropy error of first and second run concerning communication rounds that is supplied with same seed value. **34**
Figure 4.3 : Maximum error percentages of various predefined traffic intensities that is set according to the comments in Section 3.4.9. .... **36**
Figure 4.4 : Various traffic intensities and their average success percentages concerning communication rounds. **36**
Figure 4.5 : Different traffic intensities that change test accuracy values drastically. **37**
Figure 4.6 : Minimized version of Figure 4.5 which presents 60 communication rounds instead of 300. .... **38**
Figure 4.7 : Confidence intervals and test accuracy values for different traffic intensities at rounds 25, 50, 100 and 200. .... **39**
Figure 5.1 : Success percentages of different participation rate values. **42**
Figure 5.2 : Client numbers in communication round and the success percentages. **43**
Figure 5.3 : Client numbers in communication round and successfully sent client numbers. **43**
Figure 5.4 : Client numbers in communication round and the accuracy values. **44**
Figure A.1 : Plots for 0.2 interarrival time values. **55**
Figure A.2 : Plots for 0.25 interarrival time values. **55**
Figure A.3 : Plots for 0.3 interarrival time values. **55**
Figure A.4 : Plots for 0.4 interarrival time values. **56**
Figure A.5 : Plots for 0.5 interarrival time values. **56**
Figure A.6 : Plots for 0.7 interarrival time values. **56**
Figure A.7 : Plots for 0.8 interarrival time values. **56**
Figure B.1 : Statistics of 0.2 interarrival time with different seeds........................ **57**
Figure B.2 : Statistics of 0.25 interarrival time with different seeds...................... **57**
Figure B.3 : Statistics of 0.3 interarrival time with different seeds........................ **58**
Figure B.4 : Statistics of 0.4 interarrival time with different seeds........................ **58**
Figure B.5 : Statistics of 0.5 interarrival time with different seeds........................ **58**
Figure B.6 : Statistics of 0.7 interarrival time with different seeds........................ **59**
Figure B.7 : Statistics of 0.8 interarrival time with different seeds........................ **59**

UNVEILING THE WIRELESS NETWORK LIMITATIONS IN FEDERATED LEARNING

SUMMARY

Huge increase in edge devices over the world with powerful processors inspired many researchers to apply decentralized machine learning techniques so that these edge devices can contribute to train deep neural networks. Among those decentralized machine learning schemes, federated learning has gained tremendous sympathy as it grants privacy to the edge devices as well as diminishing communication costs. This is because federated learning does not need to access raw data nor store it, instead, clients would learn from their raw data locally and produce gradient updates. These gradient updates would be aggregated at the server. The raw data is kept at clients untouched, to a degree that only the trained gradient updates are shared with the parameter server. As a matter of fact, the privacy and security issues are mostly scaled down and the ML models instead of raw data would save communication overhead. Considering these issues, federated learning has emerged from distributed and decentralized learning yet it revolutionizes the training as it aggregates the locally trained ML models by edge devices.

A typical federated learning scheme which is investigated in the thesis, includes many number of clients who calculate the gradient of the loss function by applying stochastic gradient descent method and it also consists of an aggregator that collects these gradients in each communication round. In each round, only randomly selected number of clients participate in federated learning with their calculated gradient. The gradient descent is estimated according to the local batch size which is the fraction of client's local raw data. Collected gradients by the server are averaged in the server and the averaged gradient is disseminated to the clients back. It is expected to see the convergence after many communication rounds, as many clients are anticipated to contribute and therefore train the model in the server about the data.

Yet, the issues related to the network limitations for the federated learning process are not covered in the literature. In such federated learning applications and simulations, the network is assumed to be stable and the limitations that come with unstable network are overlooked. These simulations are mostly written on Python and the essential network settings are implicitly asserted. Quality of Service (QoS) parameters such as packet drop ratio and delay are not considered, however they stand as key factors for federated learning convergence since they can slow down or even prevent the convergence process. In fact, there are federated learning applications proposed in the literature which are real-time such as cache-based popular content prediction applications. Meaning that these applications are sensitive to packet drops and delays

that are caused by the network. Therefore, delay and packet drops in the network must be thoroughly examined in order to make such federated learning applications feasible.

To this end, an advanced federated learning simulation is introduced and results are shared in this study. The simulation includes not only clients and server which are producing gradient updates, but also a full network backbone which allows the observation of the QoS parameters in the federated learning process. To be able to achieve this, a network which consists of clients and the server of the federated learning is simulated using reputable NS3 (Network Simulator 3). The network is designed as dumbbell topology which includes 100 clients on the left hand side of the dumbbell and server on the right hand side. This makes the left router to be the bottleneck, thus the background traffic in the network causes packet drops there. Additional node to generate background traffic is placed in the same side with clients so that the packet drops are observed and the intensity of the packet drops can be arranged by a hyperparameter which is called the interarrival time of the packets that are generated via background traffic. Poisson distribution based background traffic is produced in the manner of the interarrival time between packets at the traffic generator node.

By applying ns3-ai framework which enables NS3 and python processes to communicate, the network and the federated learning process are run simultaneously so that the observations on QoS can be made. Since millions of devices are expected to be involved in a federated learning application in which the speed of converge is essential and not all of the clients updates may increase the convergence, UDP (User Datagram Protocol) is utilized as transport layer protocol. These gradient updates are fragmented to UDP packets and are sent from clients to servers and servers to clients. Thus, whenever a UDP packet that carries client update is dropped, the whole client update must be discarded. As a result, discarded clients reduce the performance of the federated learning and cause significant drawbacks to the application.

Initially, the experiment is validated by running countless simulations with different seed values. Validation is carried out by testing the reproducibility of the same experiments by comparing cross entropy error, accuracy of both server and clients and also packet drop rates. For various interarrival time values ranging from 250 milliseconds to 900 milliseconds many simulation scenarios are designed. The replication method is used to evaluate the results. This means that each scenario with different seeds are run 10 times and the results are presented with %95 confidence interval. Among those scenarios, three of them are picked and are tagged as heavy, medium and light traffic intensity which correspond to 250, 400, 900 milliseconds interarrival time, respectively.

The results are presented by giving maximum error rates, average success rates and per round test accuracies. The most erroneous batch that is detected in aggregated gradient at server is presented by maximum error percentage after each communication round. It shows the worst performing model and it is meaningful to demonstrate the unfavorable consequences of the background traffic to the performance. With heavy intensity traffic, maximum error percentage goes up to %80 after round 90, whereas maximum error percentage is between %10 and %20 with light traffic. This shows the federated learning application's early vulnerability to the background traffic. With the assumption of the network being completely stable, then average success percentage of client update delivery becomes %100. However, it is not realistic and average success percentage reduces and fluctuates according to the traffic intensity. As the traffic gets intense, less client updates are received by the parameter server for a successful aggregation. Finally, the test accuracy of various intensity traffic configurations are presented. Packet drops because of the bottleneck queue capacity overflow causes tremendous decrease for the test accuracy which is crucial for any federated learning application. For at least 200 communication rounds, the decline in the accuracy is evidently visible when the traffic is intense. More specifically, %90 accuracy is reached over 120 rounds for high intensity traffic, while it is reached around 60 rounds for light traffic. The intensity of the background traffic becomes highly crucial consideration for potential time-critical federated learning applications. Confidence intervals on test accuracy are presented according to the traffic intensity. The convergence is achieved no matter what the traffic intensity is. Wide intervals can be seen in earlier rounds and it gets slightly wider if the intensity is higher. In addition to these, according to the traffic intensity or interarrival time, the amount of traffic data, the number of packets that are produced by the background traffic generator node, the data delivery rate and monitored interarrival time are presented as well.

In the light of these results, an adaptive federated learning is proposed in order to cope with heavy intensity traffic. By using network metrics such as upload rate, transmission and queueing delay, the maximum number of clients that can be fit in a communication round is calculated and set as participation rate. This allows server to receive more client updates and increasing the performance of the federated learning under heavy background traffic.

KABLOSUZ İNTERNET AĞLARINDAKİ KISITLARIN FEDERE ÖĞRENMEYE OLAN ETKİLERİNİN ORTAYA ÇIKARILMASI

ÖZET

Güçlü ve kapasitesi yüksek mobil cihazların artışı birçok araştırmacıyı merkezi olmayan makine öğrenmesi tekniklerinin bu cihazların üzerinde kullanılmasına ilham kaynağı olmuştur. Bu teknikler ile bu cihazlar kullanılarak derin öğrenme ağlarının eğitilmesinin önü açılmıştır. Merkezi olmayan makine öğrenmesi tekniklerinden biri olan federe öğrenme, oldukça ilgi görmüş olup cihazlardaki gizliliğin korunmasına katkı sağlar ve aynı zamanda bilgisayar ağlarındaki maliyetleri de düşürür. Bunun sebebi ise federe öğrenmenin işlenmemiş verinin olduğu gibi gönderilmesinden ziyade cihazlardan sadece gradyan güncellemelerinin alınmasıdır. Böylelikle bu işlenmemiş verilerin bir yerde tutulmasına da gerek kalmaz. Bu gradyan güncellemeleri cihazdaki veriler kullanılarak oluşturulur ve sunucuya gönderilir, sonrasında ise sunucuda bu gradyanların ortalaması alınarak yeni bir model oluşturulur. İşlenmemiş veriye asla dokunulmaz, bu veri cihazda kalır ve sadece gradyan güncellemesi sunucuyla paylaşılır. Sonuç olarak birçok cihazın bulunduğu merkezi olmayan makine öğrenmesi düşünüldüğünde, federe öğrenme güvenlik ve gizlilik konuları konusunda oldukça kolaylık sağlar. Bahsedilenler göz önünde bulundurulduğunda, federe öğrenmenin dağınık ve merkezi olmayan merkezi öğrenme sistemlerinde devrim yarattığını söylemek mümkündür.

Bu tezde, stokastik gradyan düşüşüyle kayıp fonksiyonu hesaplayan birçok cihazın bulunduğu ve her iletişim turunda bir sunucunun bu gradyanların ortalamasını aldığı tipik bir federe öğrenme sistemi incelendi. Her turda, belli bir sayıda rastgele seçilen cihazlar bu federe öğrenmeye dahil edilir ve gradyan güncellemelerini oluştururlar. Cihazdaki verinin bir kısmının kullanılarak hesaplandığı bu gradyanlar sunucu tarafından toplanılarak ortalamaları alınır, yeni bir model oluşturulur ve bu yeni model tekrar cihazlara gönderilir. Belli bir tur sayısından sonra ise doğrulukta bir yakınsama oluşması beklenilir. Bunun sebebi ise birçok cihazın katkıda bulunduğu bu öğrenme sistemi, birçok turdan sonra sunucuya yeteri kadar gradyan güncellemesi gönderecektir.

Fakat, federe öğrenmenin bilgisayar ağlarında karşılaşacağı zorluklar ve kısıtlamalar göz ardı edilmektedir. Federe öğrenme uygulamaları ve simülasyonlarında bilgisayar ağları tamamen stabil varsayılmakta ve olası kısıtlamalar incelenmemektedir. Simülasyonlar genellikle Python'da yazılmakta ve bilgisayar ağları için şart olan parametreler dolaylı yollardan varsayımlarla uygulanmaktadır veya hiç uygulanmamaktadır. Paket düşme oranları ve gecikme gibi hizmet kalitesi (Quality of Service, QoS) parametreleri simülasyonlarda incelenmemektedir, fakat bu parametrelerin

federe öğrenmede anahtar faktörleri olarak karşımıza çıkar. Bunun sebebi ise bu parametreler yakınsamayı oldukça yavaşlatabilir veya tamamen durdurabilir. Federe öğrenme için önerilen uygulamalar gerçek zamanlı olduğundan dolayı olası gecikme ve paket düşüşleri bu uygulamaların performanslarını oldukça etkileyecektir. Bu sebepten ötürü bu iki parametre detaylı bir şekilde incelenmeli ve bu bilgiler ışığında federe öğrenmenin fizibilite raporu çıkarılmalıdır.

Anlatılanlar doğrultusunda, federe öğrenme için ileri bir simülasyon önerilmiştir ve simülasyondan alınan sonuçlar bu çalışmada paylaşılmıştır. Simülasyon gradyan güncellemesi üreten cihazlar ve sunucunun dışında tam anlamıyla bir bilgisayar ağı üzerine kurulmuştur. Böylelikle yukarıda bahsedilen hizmet kalitesi parametreleri federe öğrenmede incelenebilir hale gelmiştir. Bunu başarabilmek için, çokça sayıda cihazı ve sunucuyu içinde bulunduran bilgisayar ağı, tanınırlığı ve güvenirliği yüksek olan NS3 (Network Simulator 3) kullanılarak simüle edilmiştir. Bu bilgisayar ağı dambıl topolojisi modeli kullanılarak tanımlanmıştır. Dambılın sol tarafına bağlı 100 cihaz, sağ tarafına bağlı sunucu konumlandırılmıştır. Bu dambılın sol tarafında bir darboğaz yaratmaktadır, bu şekilde bilgisayar ağındaki oluşacak trafikte paketler bu darboğazda yakalanmaktadır. Bu cihazlara ek olarak, sol tarafa arka planda oluşacak trafiği yaratan bir cihaz daha eklenmiştir. Bunun sebebi ise bu trafiğin paketler arası zaman parametresi şeklinde bu cihazda tanımlanmasıyla trafiğin istenilen sıklıkta oluşturulmasını sağlamaktır. Bu şekilde paket düşüşleri incelenecek ve trafiğin yoğunluğu istenilen ayarda yapılabilecektir. Arka planda oluşacak trafiğin paketlerinin geliş sıklıkları Poisson dağılması kullanılarak üretilmiştir.

ns3-ai uygulaması kullanılmasıyla bahsedilen NS3 ve federe öğrenmenin çalıştırıldığı Python uygulaması birbirleriyle haberleşebilmekte ve hizmet kalitesi parametreleri incelenebilmektedir. Milyonlarca cihazın federe öğrenme uygulamasında katkıda bulunacağı göz önünde bulundurulursa, yakınsama hızının önemi kaçınılmaz olur. Bazı cihazların az veriye sahip olmasından dolayı federe öğrenmeye katkısı düşük olacağından taşıma katmanında UDP uygulanmıştır. Bu gradyan güncellemeleri UDP paketlerine böldürülerek bilgisayar ağlarında bir yerden bir yere taşınmaktadır. Ne zaman gradyan güncellemesi bilgisi taşıyan bir UDP paketi düşerse tüm gradyan güncellemesi federe öğrenmeden çıkarılmalıdır. Sonuç olarak, bu zorundalıktan ötürü çıkarılan güncellemeler federe öğrenmenin performansını düşürür ve bu uygulamada çok ciddi sorunlara yol açar.

İlk olarak, farklı seed değerleriyle birçok simülasyon çalıştırılarak deneyin doğruluğu onaylandı. Bunun gerekli olmasının sebebi, bu deneyde veri dağılımı ve arka planda oluşturulacak trafiğin seed değerleriyle rastgele sağlanmış olmasından gelir. Bu doğrulama işlemi aynı deneylerin tekrar edilebilirliğinin sınanmasıyla gerçekleştirilmiştir ve burada cross entropy hatası, sunucu ve cihazlarardaki doğruluk ve paket düşme zamanlarına bakılarak yapılmıştır. 250 milisaniyeden 900 milisaniyeye olacak şekilde farklı paketler arası zaman değerleriyle farklı deney senaryoları oluşturulmuştur. Sonrasında, sonuçların elde edilmesi için tekrarlama (replication) metodolojisi kullanılmıştır. %95 güven aralığı kullanılarak her bir senaryo 10 kez çalıştırılmış ve sonuçların ortalaması alınmıştır. Bu senaryolardan yoğun, orta ve hafif trafik olarak etiketlenen ve sırasıyla 250, 400, 900 paket arası zaman değerine tekabül eden deneyler seçilmiştir.

Sonuçlar her bir turda maksimum hata oranları, ortalama başarılı paket gönderme oranları ve test verisi üzerinden doğrulama oranları olarak elde edildi. Sunucuda oluşturulan model test verisi üzerinde sınandı ve sınanırken üzerinden geçtiği veri parçalarından en yüksek hataya sahip olan parçadaki hata oranı maksimum hata oranı olarak gösterildi. Bu aslında modelin en kötü performansını göstermekte ve ağsal trafiğin uygulamadaki oluşacak istenmeyen sorunları göstermede oldukça etkili olduğu görüldü. Yoğun trafik olduğu durumda maksimum hata oranı tur sayısı 90 olduğunda bile 0.8'e kadar çıkıyor. Trafğin hafif olduğu durumda ise aynı tur sayısında 0.1 ve 0.2 civarlarında seyrediyor. Eğer bilgisayar ağları tamamen stabil kabul edilirse, ortalama başarılı paket gönderme oranı sürekli %100 çıkacaktır. Fakat bu hiç de gerçekçi değildir ve ortalama başarı oranı trafik oranına göre düşmekte ve değişmektedir. Trafiğin yoğunlaşmasıyla modelini başarılı gönderebilen cihaz sayısında düşme gerçekleşmektedir. Buna ek olarak test verisi üzerinde doğruluk oranları farklı trafik yoğunluklarına göre gösterilmiştir ve güven aralıkları paylaşılmıştır. Topolojideki darboğaz yaratan kuyruktaki paket düşme oranlarının doğruluk oranlarında oldukça büyük düşüşler yarattığı görülmüştür ve bunun uygulama performansına olan yüksek etkisi bilinmektedir. 200 iletişim turunda, trafiğin yoğun olduğu durumda doğruluk oranlarındaki düşüşler kolaylıkla görünmektedir. Örnek verilecek olunursa, trafiğin yoğun olduğu durumda 0.9 doğruluk oranına 120 turda erişilmiş, trafiğin hafif olduğu durumda ise bu orana 60 turda erişilmiştir. Bu da trafik yoğunluk oranının gerçek zamanlı federe öğrenme uygulamalarını ciddi bir şekilde etkilediğini göstermektedir. Bu sonuçların yanında güven aralıkları da verilmiştir. Yakınsama belli bir tur sayısından sonra tüm trafik durumlarında erişilmiştir. Güven aralıkları ilk turlarda oldukça geniş ve eğer trafik yoğunsa bunların daha da geniş oldukları fark edilmiştir. Bunlara ek olarak, trafik yoğunluğu veya paket arası zaman değerleri, trafikle oluşturulan veri, trafik tarafından oluşturulan paket sayısı, veri oranı ve gözlemlenen paket arası zaman değerleri de gösterilmiştir.

Bu sonuçların ışığında, yoğun trafikle baş edebilecek adaptif federe öğrenme yöntemi önerilmiştir. Yükleme oranı, gecikme gibi ağ metrikleri kullanılarak, bir iletişim turunda başarılı gönderim yapabilecek maksimum cihaz sayısı hesaplanmış ve katılım oranı buna göre belirlenmiştir. Bu sunucunun yoğun trafik olduğu zamanda daha çok cihaz gönderimini almasını sağlamakta ve federe öğrenme uygulamasının performansını artırmaktadır.

1. INTRODUCTION

1.1 What is Federated Learning?

The term federated learning (FL) was initially announced by Google researchers in [4]. It is asserted that the clients form a federation in a way to develop the global ML model. The key advantage of federated learning is that model training can be achieved with the absence of the raw data. Thus, the privacy is preserved and security risks are reduced. On top of that; the unbalanced, non-IID and massively distributed nature of data in clients is pointed out along with the communication limitations in mobile devices. Uplink connection is argued to be essential in the case of mobile devices, since wireless communication is mostly limited to few Mb/s in general. Nevertheless, the computational cost is reduced due to the fact that the modern mobile devices have faster processors. Considering these factors, "Federated Averaging" algorithm (known in the literature as FedAvg or FAvg) is proposed which includes many number of clients who calculate the gradient of the loss function by applying stochastic gradient descent method and it also consists of an aggregator that collects these gradients in each communication round. In each round, only randomly selected number of clients (C) will participate in FL with their calculated gradient. The gradient descent will be estimated according to the local batch size which is the fraction of client’s local raw data (B). Collected gradients by the server then averaged in the server and the averaged gradient would be disseminated to the clients back. It is expected to see the convergence after many communication rounds.

1.2 The Importance of Federated Learning

Improvements in machine learning techniques have uncovered smart solutions in various fields such as learning from data, natural language processing (NLP), autonomous vehicular networks, computer vision, AR/VR etc. with the emerging big data and powerful computer processors [5]. If learning from data is to be considered, traditional centralized ML techniques have started to evolve to decentralized and distributed approaches, and now leveraged to the federated learning (FL) [6] that addresses limited wireless channel metrics, privacy and security issues. It is asserted by the authors in [4] such that the primary contribution of federated learning is not being in need of accessing raw data or storing it, instead, clients would learn from their raw data locally and these models would be aggregated at the server. Federated learning allowed the process of learning from millions of devices possible. Although some traditional centralized ML techniques can be utilized in the networks where limitations are minimal [7], [8] [9], due to the fact that an additional privacy layer must be implemented, makes federated learning highly favorable [10].

However, if these clients were to sent their data to a centralized server, their local data would be exposed and also it would be highly costly to collect the data. Additionally, it would be necessity to use incredibly powerful processor to process this massive data. For example, to be able to run Inception V4 neural network model, Google's Cloud TPU [11] which has 4 TB high bandwidth memory is asserted to be capable of assigning 44.3 GB of memory to it [12]. Since the process is nearly impossible, the importance of federated learning becomes obvious for use-case such as this.

1.3 Federated Learning in a Network Sense

FL does not need to collect data from clients and it automatically decreases the communication overhead when network is considered [5]. Even though, a lot of studies focused on decreasing overall communication and privacy in federated learning [13], [4], [1], [3], [14], [15], [2], [16], [17], [18], [19], [20], [21], [22], [23], [24], [25], [26] the network dynamics such as packet drops, delay and the background traffic were left untouched. Examinations were not sufficient in network sense and conclusions on the effects of network dynamics could not be derived since no network simulator such as NS3 was used for the purpose. Additionally, it is asserted in [27] that network topology effects on runtime, delay, and duration [28] for SGD iteration are not examined specifically.

As mentioned above, federated learning should be developed in a network sense in order to test it in an environment that consists of background traffic and network layer settings that correspond to wireless network. All things considered, the intend of this study is to first simulate federated learning in a network environment, then to inspect the results of the background traffic on the performance. Using both NS3 [29] and ns3-ai [30], Federated Averaging (FedAvg) [4] is simulated under the background traffic. The trend to simulate federated learning is to use Python with simple assumptions on the network side which are packet loss, jitter, latency, bandwidth, Quality of Experience (QoE) and transport layer protocol performance [5]. However, these parameters are quite crucial to observe the performance of the application and to collect QoS parameters and monitor wireless channel quality as it is implicated in respectively [31] and [6]. Eventually, for federated learning applications that will be performed in 5G, QoS constraints must be addressed. It will only be possible if these applications are simulated carefully with considering network limitations. For example, the latency must be minimized for Ultra-Reliable Low Latency Communications (URLLC) and it is given as an open issue in [32].

1.4 Contribution

Contributions are listed below:

- By including UDP as transport layer protocol, the number of clients, non-IID data distribution, learning rate, the delay of communication channel, upload download rate of clients and federated learning gradient update packet size etc. a network simulation is created adopting NS3 in order to show QoS parameters under the background traffic. Key observations of federated learning are derived by extracting packet drop rate, maximum and cross entropy error, and test accuracy. These observations reveal that reaching to 0.9 accuracy is prevented by at least 60 communication rounds when the traffic is intense and packet interarrival time is 250 ms.

- Adaptive federated learning model is proposed in which the number of clients in a communication round is maximized, and significant improvement is made with the testing accuracy of the federated learning when the traffic is heavy.

1.5 Organization of the Thesis

This thesis is organized as follows: Section 2 outlines the earlier studies. In Section 3, the simulation model is revealed and its simulation parameters are explained in detail. Results, comments and analysis are presented in Section 4. The adaptive federated learning model and its results are shared in Section 5. Conclusion and future work are given in Section 6.

2. RELATED WORK

Because the validation of federated learning algorithms is generally done by using image classification data sets, Section 2 is ordered with various types of networks that have been used in the discussions, emulations, and simulations of the reviewed studies. The types of networks that are reviewed are:

- Cellular Networks
- Wireless Networks
- Unreliable Networks
- Smart Cities
- AR/VR
- Vehicular Networks

The type of the network is essential for federated learning performance, therefore each type is examined individually and in detail. In fact, the network traffic of distinct network types is implemented in the simulation model and it is explained in detail in Sections 3 and 4.

2.1 Cellular Networks

In the first paper that federated learning system is proposed [4] and mentioned in Section 1.1, a typical cellular network with many number of client devices and a server is considered. In the evaluation of this federated learning scenario, the hyperparameters such as local batch size (B), selected clients (C), and number of gradient descent steps (E) are tested and compared. For this case, MNIST data set (consists of 60.000 examples) [33] that consists of handwritten digits is used. The neural network that is going to be constructed in FedAvg is a multilayer perceptron with 2-hidden layers with 200 units that uses ReLu as activation function. For IID setting, the data is shuffled and distributed to the 600 clients equally, wheras for non-IID setting, the digits are sorted and distributed in a way that each client would have only two digits. In the second experiment, The Complete Works of William Shakespeare data set [34] is used and LSTM model is trained to predict the next character. Optimum B, C and E parameters are presented, and the consequences of these parameters on communication rounds are discussed. In the third experiment, a CNN with two 5x5 convolutional layers, ReLu activation is trained on benchmark CIFAR-10 data that has 60.000 examples. FedAvg is compared with basic stochastic gradient descent (SGD) to show FL's effectiveness in terms of the number of communication rounds. FedAvg is able to achieve %85 accuracy with 2000 communication rounds, while SGD reach the same accuracy in 99000 rounds. Finally, the real world large scale data set that contains 10 million public posts is trained on over 500000 clients in non-IID setting. With C = 0.00002, E = 1 and B = 8, trained LSTM by FedAvg outperforms the baseline method by reaching %10.5 accuracy in just 35 communication rounds.

In [1], an improvement over the algorithm above is proposed via including network limitations in the calculations. The authors offer a new protocol named Federated Client Selection (FedCS) that optimizes client selection instead of selecting them randomly as in the case of FedAvg. FL use cases with Amazon Alexa, Google Home and autonomous vehicles are presented by stating the preserved privacy and maximized efficiency in communication overhead. It is included that no FL study has included realistic network configuration. They criticize FedAvg for not regarding processing power of end devices and wireless channel conditions. Therefore, they propose FedCS that can get as many client updates as possible per communication round, optimizes bandwidth usage by scheduling resource blocks (RB), and determines the time period which is assigned for each round (T) value. It does it so by asking each client its wireless channel state, its computational power and the data size that device has. Experiments contain 1000 clients and a base station considering MEC environment with radius 2km where clients are uniformly distributed. Mean and maximum throughput values are 1.4 Mbit/s and 8.6 Mbit/s, and 10 RBs (1.8 MHz bandwidth) are assigned to clients. Apart from these settings, the network is considered to be stable as mentioned in Section 1. Two data sets are evaluated namely benchmark CIFAR-10 and Fashion MNIST data sets that both have 60000 data samples and 10 object types. IID and non-IID settings are quite parallel to the [4] together with B = 50, E = 5 and C value, which is set to 0.1 corresponding to 100 clients. CNN with 3x3 convolution layers and ReLu activation is used which in return generate 18.3 MB and 14.4 MB data respectively for data sets and approximately 3 to 5 million parameters. Simulation duration is set to 400 minutes and the hyperparameter T is observed with different values. In both data sets, FedCS dominated FedAvg in terms of both accuracy and the convergence time period. For non-IID setting in both data sets, FedAvg does not reach to the required accuracy in limited time, whereas FedCS reaches. In addition to that, FedCS's overall performance beats FedAvg in terms of accuracy in the time period with the data being non-IID. Finally, it is claimed that the smartly determination of optimum number of clients for each round would enhance the performance of FL algorithms against non-IID data. Even though a few network variables are utilized in the experiments, both the transport layer and the background traffic is not considered.

By using federated learning on heterogeneous cellular networks, latency, privacy, bandwidth related issues are eliminated in large-scale deployment that contains many edge devices and multiple base stations [16]. Indeed, to be able to reduce communication latency in such large network, the authors apply gradient sparsification and design an algorithm that optimal resource scheme is used for gradient updates. It is claimed that in order to cope with tens of millions of parameters that are generated by deep neural networks, a powerful federated edge learning (FEEL) protocol is needed because these parameters would be transmitted over the heterogeneous cellular networks. Thus, A hierarchical FL algorithm (HFL) which applies intra-cluster (small base stations consist of edge devices) gradient aggregation method with inter-cluster (small base stations that are connected to global parameter server) model averaging strategy is proposed. The workload is divided between small base stations (SBS) and in each SBS, gradient updates are sent to the parameter server at each intra-cluster iteration. It is argued that transmitting tens of millions of parameters of deep learning over the network is a difficult task considering the wireless channel metrics, therefore sending a portion of the gradient update would be both sufficient and helpful for bandwidth. As a matter of fact, the sparsified gradient update would travel over the network. In the simulation, 7 hexagonal clusters with radius 500m is considered as heterogeneous cellular network. CIFAR-10 data set is trained using ResNet18 architecture. It is observed that the HFL (in which there are 7 SBSs and 4 parameter servers) is more robust than the traditional FL (in which there are 28 parameter servers). QoS parameters are not shared in this study.

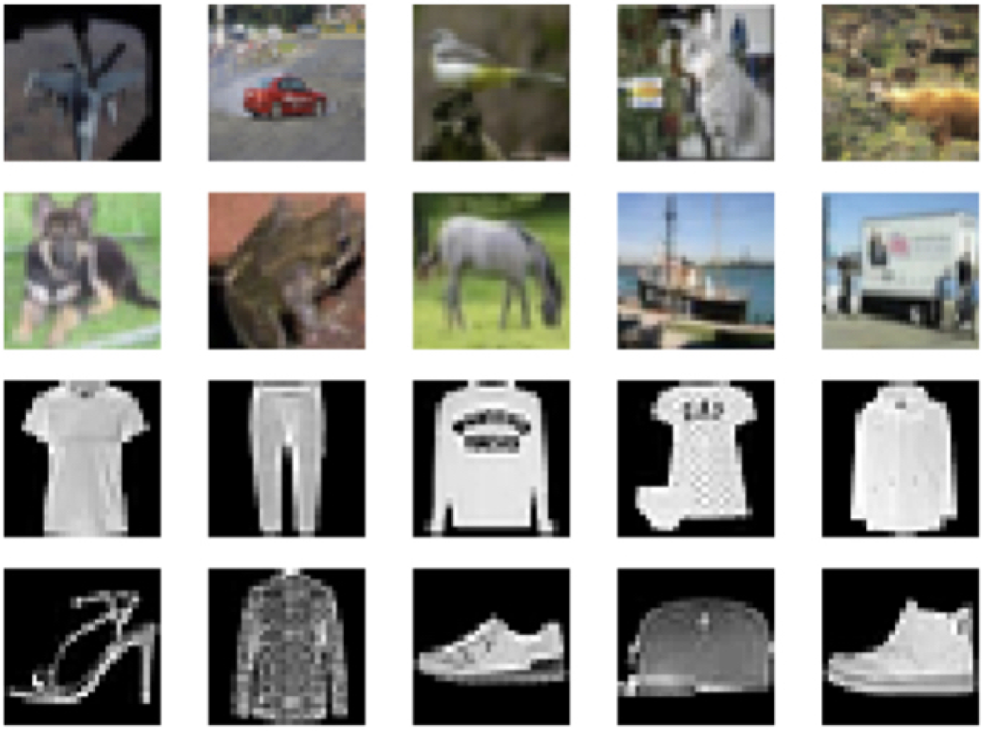


Figure 2.1 : Samples from CIFAR-10 are shown as colored images. Gray images are taken from Fashion-MNIST dataset [1].

2.2 Wireless Networks

Real wireless network systems considering their density of devices are studied in [15]. It is mentioned that VGG16 [35] data set would result in above 100 million parameters and transmission task of such gigantic data over the network depends on the network protocol. The common problems that increase delay in the communication such as path loss and multi-path fading arise in such huge data transfer in wireless networks. To cope with such difficulties, a protocol that is able to allow transmissions with the devices yielding high data rate is proposed. Because of the fact that there is a tradeoff in terms of communication time and convergence between high data rate with sparse coverage and low data rate with dense coverage. The authors firstly prove that high data rate with sparse coverage enhances the overall convergence and then evaluate it with the simulation. The simulation is carried out with PyTorch where there are six nodes in 200m x 200m topology. Fashion-MNIST data set is used with IID setting and CNN with two convolutional layers is trained. The produced model data size by clients correspond to 87.36 kB in average. With simulations having different path loss value (environment-dependent factor), it is concluded that in the cases where path loss is large, higher transmission rate accompanying sparse network topology results in superior results.

In [14], distributed gradient descent is formalized under theorems and assumptions that can be derived from using the aspects of deterministic gradient descent (DGD). Later, the resource constraints are added to the formulation so that the time, power and the other costs can be evaluated during the gradient descent process. For this, there are two resource constraints that are defined for both global aggregation and clients respectively. These constraints are assumed to be known beforehand, yet these constraints are usually unknown and they constantly change in a real world setting. The formalization and calculations allow authors to estimate the convergence upper bound so that the proposed control algorithm would eventually keep resource constraints under control. By sensing these constraints, they control both the number of local updates in between global aggregations and the total number of iterations adaptively. Firstly, they evaluate the results using practical network setting which consists of two computers and three Raspberry Pi in one network. Central server is not included, as one of the computers would act as central server. The virtual simulation had 5 to 500 end devices. The time consumption is selected as the resource constraint in the simulation. There are four cases defined on the distribution of data: IID, non-IID, each node has the entire dataset and half IID, half non-IID. MNIST handwritten digit data set is used, however since DGD is utilized in the formulation and DGD may not find a solution in huge data sets, only small portion of the data is used. SVM and CNN is trained and this control algorithm is compared with FedAvg. In different data sets and with different data distributions, the effectiveness of this method is proven over FedAvg.

Improvements over [14], are presented in [13]. In addition to [14], different types of resource constraints such as bandwidth and power consumption are included to the study. On top of that, the theoretical analysis is enhanced using the properties of FL.

Three baselines that are defined: Traditional centralized gradient descent where all the nodes have the complete data, FedAvg (fixed number of local updates in between global aggregations) and synchronous gradient descent algorithm where the number of local updates in between global aggregations is exactly equal to 1. In addition to their prior study, they also include CIFAR-10 data set to their same simulations. Interestingly, they present that when the local update number is fixed, the results become poorer in each case.

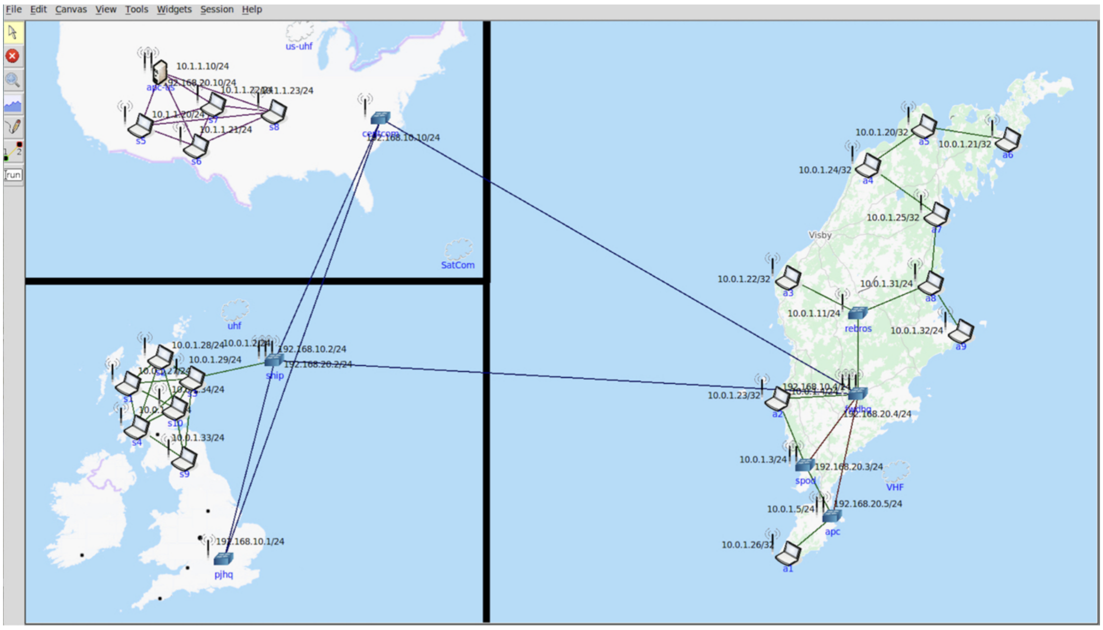


Figure 2.2 : Emulated network topology can be observed in [2]. The dumbbell and multi path topologies where there are routers as bottlenecks show similarities to the simulation model that is described in Section 3.

2.3 Unreliable Networks

The study in [36] shows a distributed FL application which uses UDP instead of TCP as transportation layer protocol. Gradient update parameters are formed into UDP packets and these packets are transmitted in the network. Yet, an additional adjacency matrix is required to define the reliability of the network edges is included to monitor packet drops. Packet drops are given to the simulation as local parameters, since the proposed solution is decentralized and the packet drop rate is not presented. To simulate packet drops, an adjacency matrix is not required in our simulation, and the communication round delivery rate is presented in Section 4.

2.4 Ad-hoc Networks

In an another study of the same authors parallel to the aforementioned references [14] [13], they demonstrate their adaptive FL algorithm wide area network emulation [2]. As indeed, they emulate fixed and mobile nodes in the dynamic and heterogeneous network system that has limited resources. Layer 1 and 2 is emulated using EMANE (The Extendable Mobile Ad-hoc Network Emulator), upper layers are constructed using CORE and ML tasks are disseminated using Delay Tolerant Networking (DTN) which guarantees all ML tasks are received by clients. On each node there is a local controller that responds requests and FL stack including TensorFlow is installed. ML task with its data type and architecture is broadcasted in the network and the nodes would reply with its availability. If it is both available and have useful data about the task it would then proceed to send a confirmation and starts to transmit its local updates. Employing their control algorithm as described above, they train a global FL model for MNIST data set and share the accuracy results and loss values with respect to time. As it is stated by the authors, this experiment would pose the effects of mobility and available bandwidth in a visual way, yet it is implicitly asserted and there is no visuals that is provided. In future work, they are planning to apply this experiment to the next generation wireless networks.

2.5 Smart Cities

In a extensive survey that considers FL in smart cities [37], the practicality of FL in smart city use cases is debated. The common problems and the advantages of FL in a smart city are presented in various themes that involve: Privacy, data collection, quality and integrity, energy consumption, user motivation. It is declared that there are approximately 50 billion devices which are connected to the Internet and they are candidates to become a non dedicated sensors. Having equipped with powerful sensors and computing power, by 2021, there are over 3.8 billions of smartphones operating worldwide [38]. On top of mobile crowd sensing, data islands are continuing to arise in companies such as banks due to the privacy, FL can be enabler for such companies to be involved in learning from the data while protecting the privacy thanks to FL. In fact, in the case of mobile crowd sensing in FL, there will not be any investment for data collection. Powerful processors on these mobile devices unlock the possibility of such case, yet security measures need to be taken because of the fact that data integrity can be disrupted via malicious users, and attackers. Since ML training would consume both battery and bandwidth, energy consumption is one of the most essential drawbacks in FL. This is included as one of the finest future directions, as it is not studied much. Yet, it is studied other research domains such as in disaster network recovery situation [39]. In the survey, the aforementioned algorithms such as FedAvg and FedCS are explained in detail and the significance of distinguishing IID and non-IID data is emphasised. FL is investigated in three categories: Vertical, Horizontal, Transfer. All of the data sets that mentioned above are parallel to the Vertical FL since data sets overlaps, however users does not overlap. In the case of merging the bank and the retailer data (assuming there will be same customers in both companies), the resulting data set would be assigned as Horizontal FL, since the customers would overlap more than data sets. In the case of training autonomous vehicle car about stop signs, since there is not much data and there are already trained models present, the Transfer FL would be the way to go.

Increasing user motivation to enhancing participation rate in the FL is an important issue in smart environments. It is studied in [40] and a reputation-aware client selection scheme is proposed. The authors in [41] introduced a reputation-aware global model aggregation scheme. They are integrated with a reverse auction-based client selection and rewarding mechanism in [42].

2.6 Wide Area Networks

In an attempt to show the feasibility of distributed deep learning, the authors try to show if the learning task involving 200 workers can be succeeded in wide area networks [43]. The authors argue that the fact that FL schemes apply gradient aggregations asynchronously, this would escalate communication time and iterations. Therefore they propose an open-source algorithm AdaComp with a novel gradient compression and gradient selection techniques. The simulation parameters are as follows: The data is assumed to be IID, the bandwidth is fixed as 100Mb/10Mb for downlink/uplink referencing typical ADSL setup, local batch size B = 10, the number of clients are

200 and one parameter server, the traffic is generated using TCP, MNIST data set is used, CNN with two convolutional layers and two full-connected layers (211,690 parameters) is trained. The algorithm is compared with previous asynchronous stochastic gradient descent algorithms in terms of the number of iterations and the ingress traffic that that is caused by the ML task. Proposed solution converges much quicker as it needs less iterations. It also shows greater performance comparing to other algorithms so that the gain in the ingress traffic is 191 times better due to the compressed gradients.

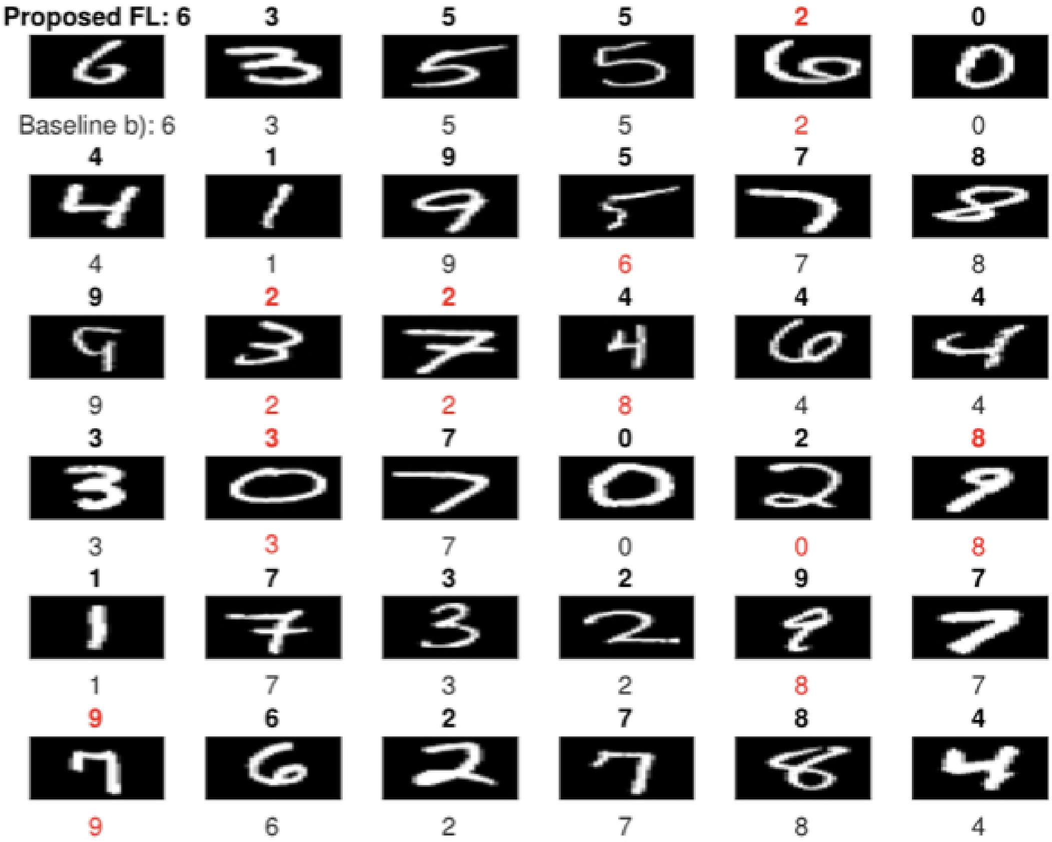


Figure 2.3 : Handwritten digits that are taken from MNIST dataset. Proposed algorithm of [3] classification results can be observed.

2.7 AR/VR and Vehicular Networks

It is asserted in [3] that the study is first to consider learning and wireless network metrics collectively. Unreliable connections, the limitations in the wireless networks and the communication costs of end devices are discussed. Ultra-reliable low latency communication (URLLC) in vehicular networks is also considered. Simulations are carried out in Matlab Machine Learning Toolbox and a circular network with 500 meters is used. The results present that the algorithm performs better than the baseline, yet the data set distribution is not mentioned.

In [6], the authors focus on FL on 5G and its applications in the sense of wireless networks which resumes to grow its complexity and its heterogeneity. The evolution of learning algorithms starting from distributed learning in wireless sensor networks to FL are displayed. An AR FL scenario that predicts the content popularity is implemented by applying AutoEncoders that consists of 1 hidden layer of 128 neurons. Publicly available data set MovieLens 1M is used to compare FL with the baseline which is rather an impossible case: Getting the raw data from users. FL starts to dominate baseline whenever the number of users pass 100. Various use cases in 5G and challenges are explained in detail, and state-of-the-art methodologies and algorithms are referenced. The performance may not be maintained if the network traffic is heavy. The network traffic consequences is presented in Section 4.

A broad study [12] which focuses heavily on many key ML concepts and neural networks, aims to provide all enablers for both ML for communication (MLC) and communication for ML (CML) in wireless channel. Essential benefits of edge ML are shown in wireless channels as well as the stragglers which are the end devices with poor bandwidth. Stragglers can diminish the performance of the learning process as it may update the global model with its outdated models or slow down the process if they are not taken care of. It is asserted that it is specifically difficult to simulate FL application over time with non-IID data distribution, resource constraints and wireless channel limitations. It is certainly challenging, as various simulations with different configurations must be carried out. Differing applications in the future direction are presented: AR/VR applications with 360 HD videos, multi player AR/VR video games, self-driving cars etc. All these applications are generating vast amount of data which is not suitable for current wireless networks. Google's automated vehicle produces 1 GB/sec real-time sensed data by its LiDAR sensors and the High-Definition camera [38]. For in the case of AR-based demonstration in museums [6] and rich content sharing in vehicular networks, the popular content would be predicted, prefetched and cached in order to reduce the latency and communication overhead [44]. Nevertheless, FL is named to be an enabler technology for such challenges.

3. FEDERATED LEARNING NETWORK SIMULATION MODEL

3.1 Federated Learning Scenario

The scenario which is mentioned in Section 2.7 is utilized in this study since these kinds of applications are in real-time and delay sensitive. Therefore the network condition plays a huge role for the success of an application as this, and it relates to the applications that are to be utilized in 5G as well. It is because of the fact that ultra-fast access and high reliability is expected for Ultra-Reliable Low-Latency Communications (URLLC), thus examinations on the delay must be delivered, and arrangements must be carried out in order to address the QoS requirements for FL applications in 5G [5]. With that being said, since there is no data available for this kind of scenario, MNIST data set and some of the configurations such as number of clients, batch size, participation rate are referenced from [4]. 60.000 digits in MNIST data set were distributed to the clients as each client gets only 600 samples representing only two digits in non-IID manner as mentioned in Section 2.

3.2 Network Topology

The simulation model is based on the time-sensitive AR application that is mentioned, mobile clients and aggregator that are connected with two routers. The topology of the network in Fig. 3.1 is designed as dumbbell network topology, the central node is stationed on the right side, and clients would be placed on left side of the dumbbell.

3.2.1 Background Traffic Generator Node

In the topology, except for all of the clients that will participate in improving the caching of the popular content in the app, there is an additional node below: Background Traffic Generator Node. This node causes background traffic using Poisson distribution process and the consequences to the federated learning can be observed in following sections. The bottleneck on left side of dumbbell allows the experiments to evaluate packet drops in federated learning setup where communication limitations arise.

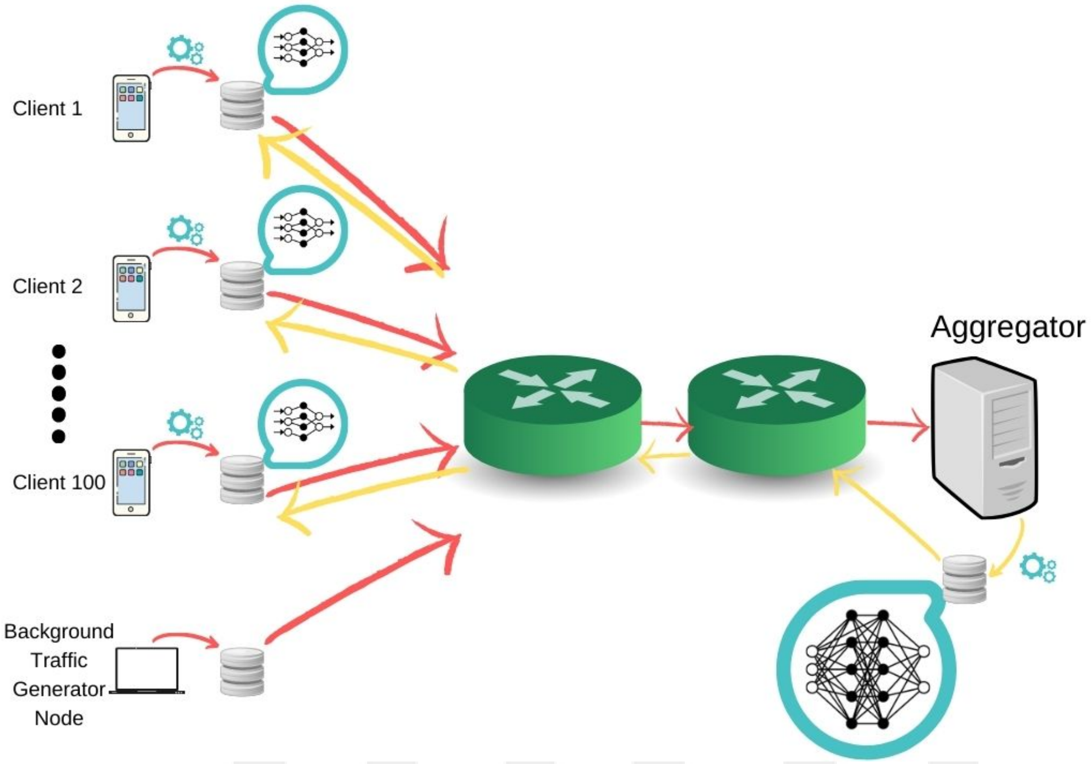


Figure 3.1 : ns3-ai Federated Learning Simulation Network Topology

3.2.2 Communication Rounds

At the first step clients run stochastic gradient descent locally using only their data and they produce gradient updates. Selected clients transmit their gradient updates at the next step to the aggregator server. These gradient updates are aggregated and the aggregated model is transmitted back to the clients. This is called a communication round which starts with first step and finishes when agreggated model is transmitted back to the clients.

3.2.3 Transport Layer Protocol

These clients will transmit their gradient updates as mentioned. UDP is picked as transport layer protocol since the application is time sensitive and there are 100 clients that need to upload and download models as soon as possible. Therefore, these gradient updates will be fragmented as UDP packets and transmitted over the network.

To be able to simulate this environment where routers, UDP transport layer protocol, clients, server and the background traffic are involved, a framework called ns3-ai [30] is used in order to combine all of the actors. Although ns3-gym [45] is a similar tool to ns3-ai, it does not give the flexibility to simulate federated learning.

3.3 Test Environment: NS3-AI

A framework called ns3-ai [30] in Fig. 3.2 unlocks the ability to exchange data between Python process in which the machine learning utilities present and NS3 in which the network is configured and observed. It enables this exchange of information via shared memory so that these Python and NS3 processes communicate through the kernel buffer. This framework has basically two use cases:

1. Sending network generated data for training
2. Testing the AI model in NS3

In this study, the second use case is used. The AI model is the federated learning model which is implemented using Python, and it is simply being tested in NS3. This model is referenced from [46] and adapted into the ns3-ai environment.

Data flow for the use case is given respectively:

1. NS3 generates the data -> writes to the memory
2. Python code reads the data -> returns AI output

These processes wait for each other. Any data structure can be used in the shared memory and only simple wrapping for the variables is needed by ns3-ai.

3.4 Simulation Model

3.4.1 Sending Fragmented Gradient Update UDP Packets

Before joining machine learning and network and also using ns3-ai, some feasibility work must be done. One of them was to be able to send fragmented gradient update packets through the network. Initially, quite simple network topology is used as shown on the Figure 3.3.

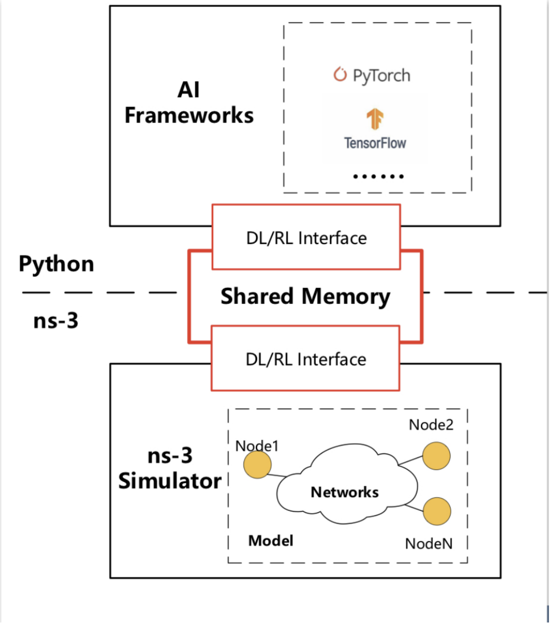
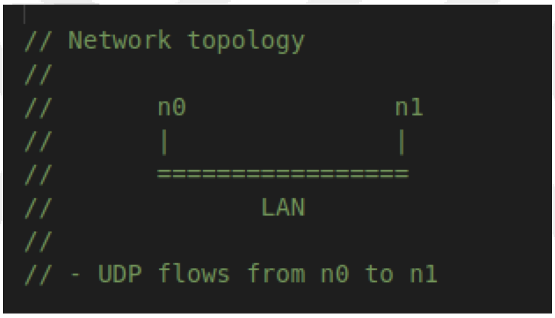


Figure 3.2 : ns3-ai Framework.



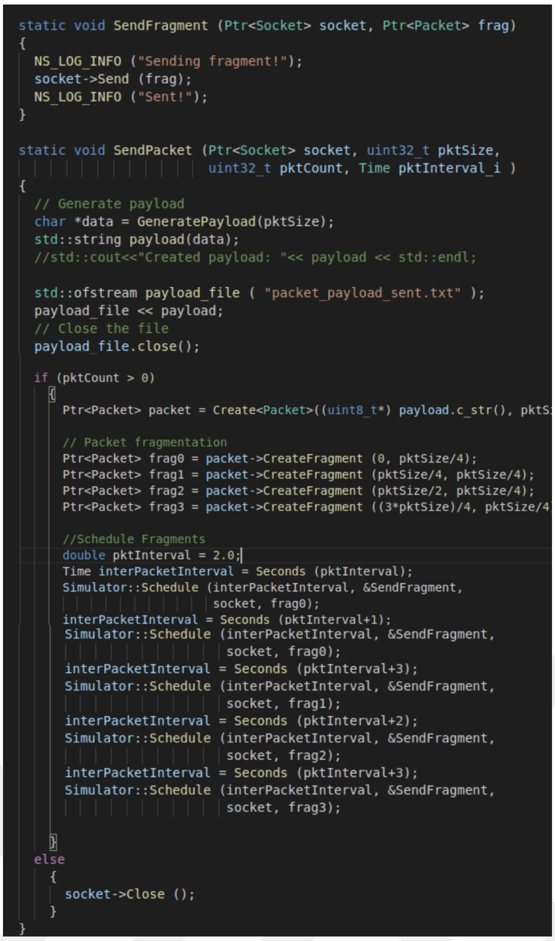
```
// Network topology
//
//       n0              n1
//       |               |
//       =================
//              LAN
//
// - UDP flows from n0 to n1
```

Figure 3.3 : Initial topology.

Simple packet generation and division of the packet into 4 fragments process is handled as given in Figure 3.4.

3.4.2 Initial ns3-ai Integration

After these fragments has been sent, the received fragments are collected. Sent and received fragment payloads are written to a ".txt" file and they are compared. This is how fragmentation in NS3 is validated. After validation, ns3-ai is integrated with a simple purpose which is getting the client ID value from the Python process in the first communication round so that the specific node would transmit its gradient update via fragments in NS3. It can be observed by the two terminals in Figure 3.4 and Figure 3.5. The terminal above in Figure 3.5 corresponds to NS3 process and it can be observed that some fragments are sent by the client and received by the server in the initial



```
static void SendFragment (Ptr<Socket> socket, Ptr<Packet> frag)
{
  NS_LOG_INFO ("Sending fragment!");
  socket->Send (frag);
  NS_LOG_INFO ("Sent!");
}

static void SendPacket (Ptr<Socket> socket, uint32_t pktSize,
                        uint32_t pktCount, Time pktInterval_i )
{
  // Generate payload
  char *data = GeneratePayload(pktSize);
  std::string payload(data);
  //std::cout<<"Created payload: "<< payload << std::endl;

  std::ofstream payload_file ( "packet_payload_sent.txt" );
  payload_file << payload;
  // Close the file
  payload_file.close();

  if (pktCount > 0)
    {
      Ptr<Packet> packet = Create<Packet>((uint8_t*) payload.c_str(), pktS

      // Packet fragmentation
      Ptr<Packet> frag0 = packet->CreateFragment (0, pktSize/4);
      Ptr<Packet> frag1 = packet->CreateFragment (pktSize/4, pktSize/4);
      Ptr<Packet> frag2 = packet->CreateFragment (pktSize/2, pktSize/4);
      Ptr<Packet> frag3 = packet->CreateFragment ((3*pktSize)/4, pktSize/4

      //Schedule Fragments
      double pktInterval = 2.0;
      Time interPacketInterval = Seconds (pktInterval);
      Simulator::Schedule (interPacketInterval, &SendFragment,
                           socket, frag0);
      interPacketInterval = Seconds (pktInterval+1);
       Simulator::Schedule (interPacketInterval, &SendFragment,
                            socket, frag0);
       interPacketInterval = Seconds (pktInterval+3);
       Simulator::Schedule (interPacketInterval, &SendFragment,
                            socket, frag1);
       interPacketInterval = Seconds (pktInterval+2);
       Simulator::Schedule (interPacketInterval, &SendFragment,
                            socket, frag2);
       interPacketInterval = Seconds (pktInterval+3);
       Simulator::Schedule (interPacketInterval, &SendFragment,
                            socket, frag3);

    }
  else
    {
     socket->Close ();
    }
}
```

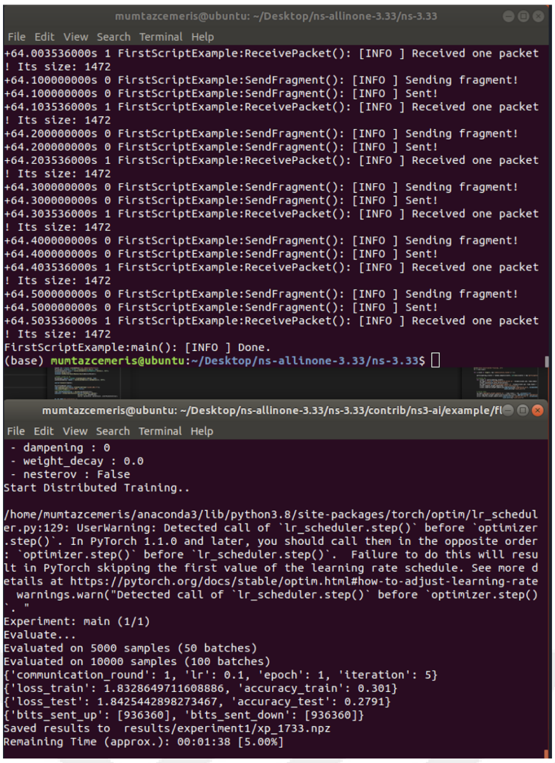
Figure 3.4 : Fragment generation and scheduling.

topology. This transmission command was given by the Python process which can be observed in the second terminal in Figure 3.5. Both processes simply communicated via ns3-ai.

3.4.3 Dumbbell Topology with 5 Nodes

Initial topology is converted to a simpler dumbbell topology which includes only 5 nodes: 2 clients, 2 routers and 1 server. In Figure 3.6, first client is transmitting fragments to server over the network in NS3.

The two clients above are defined in Python with ID values 0 and 1 and NS3 with ID values 2 and 3. This is due to the dumbbell topology ID numbering policy in NS3. In Figure 3.7, both processes can be observed side by side. On the left, MNIST IID data distribution on clients and the CNN model attributes can be observed. On the right, it can be observed that the nodes are created, the IP values are assigned and sockets are created. Below, Client 1 is being selected in Python side and this client ID and its


mumtazcemeris@ubuntu: ~/Desktop/ns-allinone-3.33/ns-3.33
File Edit View Search Terminal Help

```
+64.003536000s 1 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet
! Its size: 1472
+64.100000000s 0 FirstScriptExample:SendFragment(): [INFO ] Sending fragment!
+64.100000000s 0 FirstScriptExample:SendFragment(): [INFO ] Sent!
+64.103536000s 1 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet
! Its size: 1472
+64.200000000s 0 FirstScriptExample:SendFragment(): [INFO ] Sending fragment!
+64.200000000s 0 FirstScriptExample:SendFragment(): [INFO ] Sent!
+64.203536000s 1 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet
! Its size: 1472
+64.300000000s 0 FirstScriptExample:SendFragment(): [INFO ] Sending fragment!
+64.300000000s 0 FirstScriptExample:SendFragment(): [INFO ] Sent!
+64.303536000s 1 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet
! Its size: 1472
+64.400000000s 0 FirstScriptExample:SendFragment(): [INFO ] Sending fragment!
+64.400000000s 0 FirstScriptExample:SendFragment(): [INFO ] Sent!
+64.403536000s 1 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet
! Its size: 1472
+64.500000000s 0 FirstScriptExample:SendFragment(): [INFO ] Sending fragment!
+64.500000000s 0 FirstScriptExample:SendFragment(): [INFO ] Sent!
+64.503536000s 1 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet
! Its size: 1472
FirstScriptExample:main(): [INFO ] Done.
(base) mumtazcemeris@ubuntu:~/Desktop/ns-allinone-3.33/ns-3.33$ 
```

mumtazcemeris@ubuntu: ~/Desktop/ns-allinone-3.33/ns-3.33/contrib/ns3-ai/example/fl
File Edit View Search Terminal Help

```
 - dampening : 0
 - weight_decay : 0.0
 - nesterov : False
Start Distributed Training..

/home/mumtazcemeris/anaconda3/lib/python3.8/site-packages/torch/optim/lr_schedul
er.py:129: UserWarning: Detected call of `lr_scheduler.step()` before `optimizer
.step()`. In PyTorch 1.1.0 and later, you should call them in the opposite order
: `optimizer.step()` before `lr_scheduler.step()`.  Failure to do this will resu
lt in PyTorch skipping the first value of the learning rate schedule. See more d
etails at https://pytorch.org/docs/stable/optim.html#how-to-adjust-learning-rate
  warnings.warn("Detected call of `lr_scheduler.step()` before `optimizer.step()
`. "
Experiment: main (1/1)
Evaluate...
Evaluated on 5000 samples (50 batches)
Evaluated on 10000 samples (100 batches)
{'communication_round': 1, 'lr': 0.1, 'epoch': 1, 'iteration': 5}
{'loss_train': 1.8328649711608886, 'accuracy_train': 0.301}
{'loss_test': 1.8425442898273467, 'accuracy_test': 0.2791}
{'bits_sent_up': [936360], 'bits_sent_down': [936360]}
Saved results to  results/experiment1/xp_1733.npz
Remaining Time (approx.): 00:01:38 [5.00%]
```

Figure 3.5 : Gradient update transmission in the first communication round.

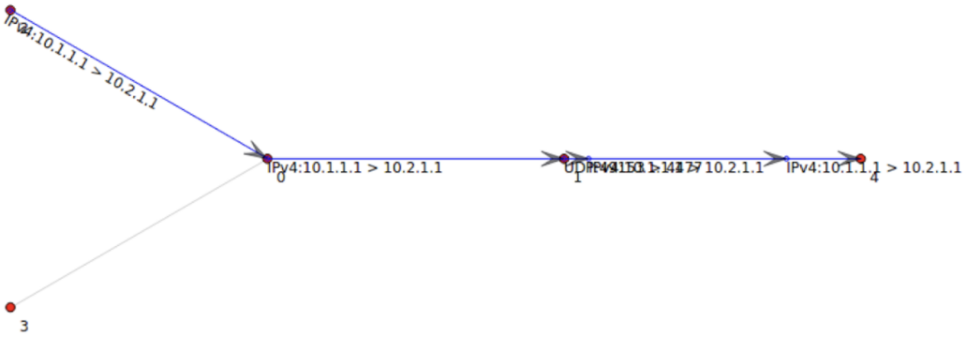
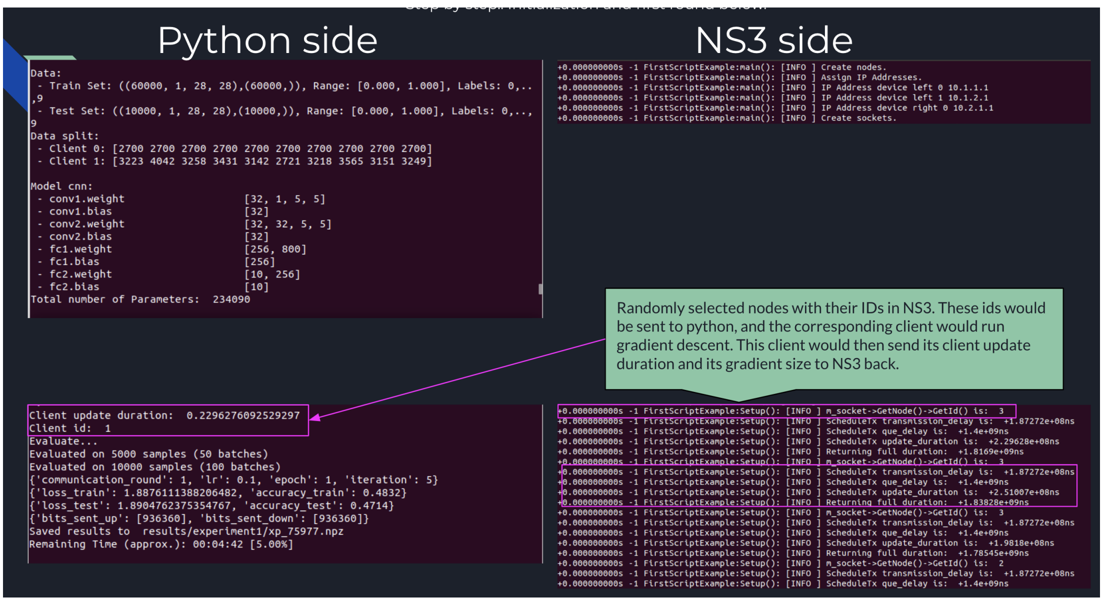


Figure 3.6 : Dumbbell topology with 5 nodes.

update duration are given to the NS3 process via ns3-ai. In NS3, the corresponding client with ID 3 is being selected and its update duration, transmission and queueing delay values are used in scheduling packet transmission.

Not only update duration and client ID values are sent to NS3 by Python process, but also the client gradient update size is sent in order to create network packets precisely. In Figure 3.8, first and second communication rounds and the calculated number of bits in the gradient updates are observed on the left hand side. On the right hand side, it can


Python side
NS3 side
Randomly selected nodes with their IDs in NS3. These ids would be sent to python, and the corresponding client would run gradient descent. This client would then send its client update duration and its gradient size to NS3 back.

```
Data:
 - Train Set: ((60000, 1, 28, 28),(60000,)), Range: [0.000, 1.000], Labels: 0,..
,9
 - Test Set: ((10000, 1, 28, 28),(10000,)), Range: [0.000, 1.000], Labels: 0,..,
9
Data split:
 - Client 0: [2700 2700 2700 2700 2700 2700 2700 2700 2700 2700]
 - Client 1: [3223 4042 3258 3431 3142 2721 3218 3565 3151 3249]

Model cnn:
 - conv1.weight                   [32, 1, 5, 5]
 - conv1.bias                     [32]
 - conv2.weight                   [32, 32, 5, 5]
 - conv2.bias                     [32]
 - fc1.weight                     [256, 800]
 - fc1.bias                       [256]
 - fc2.weight                     [10, 256]
 - fc2.bias                       [10]
Total number of Parameters:  234090
```

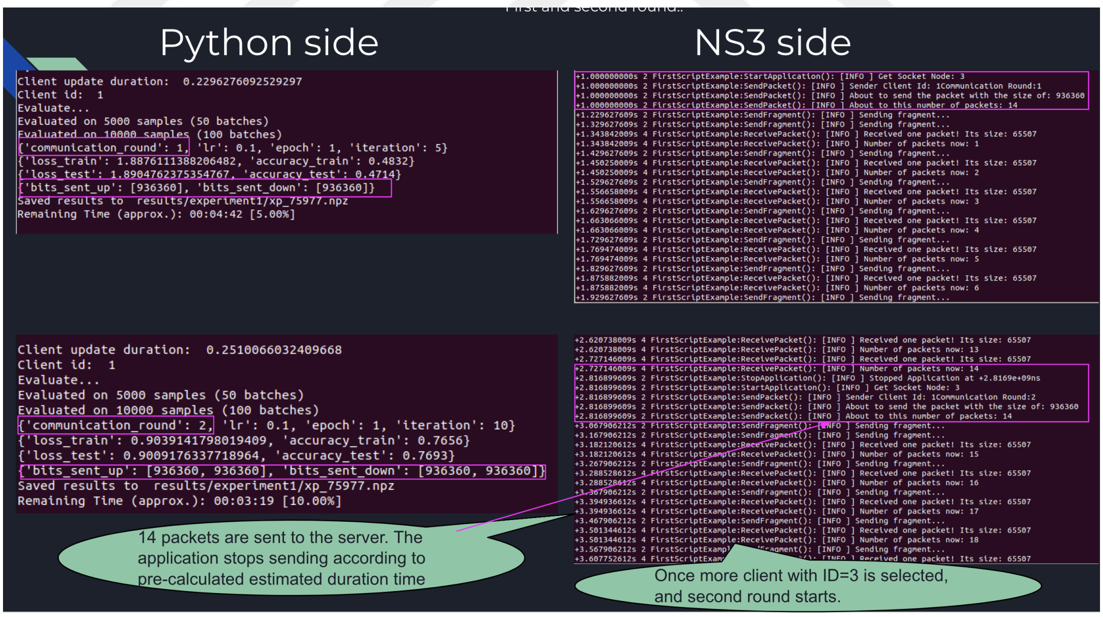
```
+0.000000000s -1 FirstScriptExample:main(): [INFO ] Create nodes.
+0.000000000s -1 FirstScriptExample:main(): [INFO ] Assign IP Addresses.
+0.000000000s -1 FirstScriptExample:main(): [INFO ] IP Address device left 0 10.1.1.1
+0.000000000s -1 FirstScriptExample:main(): [INFO ] IP Address device left 1 10.1.2.1
+0.000000000s -1 FirstScriptExample:main(): [INFO ] IP Address device right 0 10.2.1.1
+0.000000000s -1 FirstScriptExample:main(): [INFO ] Create sockets.
```

```
Client update duration:  0.2296276092529297
Client id:  1
Evaluate...
Evaluated on 5000 samples (50 batches)
Evaluated on 10000 samples (100 batches)
{'communication_round': 1, 'lr': 0.1, 'epoch': 1, 'iteration': 5}
{'loss_train': 1.8876111388206482, 'accuracy_train': 0.4832}
{'loss_test': 1.8904762375354767, 'accuracy_test': 0.4714}
{'bits_sent_up': [936360], 'bits_sent_down': [936360]}
Saved results to  results/experiment1/xp_75977.npz
Remaining Time (approx.): 00:04:42 [5.00%]
```

```
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] m_socket->GetNode()->GetId() is:  3
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] ScheduleTx transmission_delay is:  +1.87272e+08ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] ScheduleTx que_delay is:  +1.4e+09ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] ScheduleTx update_duration is:  +2.29628e+08ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] Returning full duration:  +1.8169e+09ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] m_socket->GetNode()->GetId() is:  3
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] ScheduleTx transmission_delay is:  +1.87272e+08ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] ScheduleTx que_delay is:  +1.4e+09ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] ScheduleTx update_duration is:  +2.51007e+08ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] Returning full duration:  +1.83828e+09ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] m_socket->GetNode()->GetId() is:  3
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] ScheduleTx transmission_delay is:  +1.87272e+08ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] ScheduleTx que_delay is:  +1.4e+09ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] ScheduleTx update_duration is:  +1.9818e+08ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] Returning full duration:  +1.78545e+09ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] m_socket->GetNode()->GetId() is:  2
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] ScheduleTx transmission_delay is:  +1.87272e+08ns
+0.000000000s -1 FirstScriptExample:Setup(): [INFO ] ScheduleTx que_delay is:  +1.4e+09ns
```

Figure 3.7 : Python and NS3 processes side by side.

be observed that the gradient update is fragmented and sent over the network. 936360 number of bits is divided into 14 UDP fragments with a payload 65507 number of bits. This experiment is finished in 8.3839 virtual seconds including four communication rounds and two clients.


Python side
NS3 side
14 packets are sent to the server. The application stops sending according to pre-calculated estimated duration time
Once more client with ID=3 is selected, and second round starts.

```
Client update duration:  0.2296276092529297
Client id:  1
Evaluate...
Evaluated on 5000 samples (50 batches)
Evaluated on 10000 samples (100 batches)
{'communication_round': 1, 'lr': 0.1, 'epoch': 1, 'iteration': 5}
{'loss_train': 1.8876111388206482, 'accuracy_train': 0.4832}
{'loss_test': 1.8904762375354767, 'accuracy_test': 0.4714}
{'bits_sent_up': [936360], 'bits_sent_down': [936360]}
Saved results to  results/experiment1/xp_75977.npz
Remaining Time (approx.): 00:04:42 [5.00%]
```

```
+1.000000000s 2 FirstScriptExample:StartApplication(): [INFO ] Get Socket Node: 3
+1.000000000s 2 FirstScriptExample:SendPacket(): [INFO ] Sender Client Id: 1Communication Round:1
+1.000000000s 2 FirstScriptExample:SendPacket(): [INFO ] About to send the packet with the size of: 936360
+1.000000000s 2 FirstScriptExample:SendPacket(): [INFO ] About to this number of packets: 14
+1.229627609s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
+1.329627609s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
+1.343842009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet! Its size: 65507
+1.343842009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Number of packets now: 1
+1.429627609s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
+1.450250009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet! Its size: 65507
+1.450250009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Number of packets now: 2
+1.529627609s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
+1.556658009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet! Its size: 65507
+1.556658009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Number of packets now: 3
+1.629627609s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
+1.663066009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet! Its size: 65507
+1.663066009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Number of packets now: 4
+1.729627609s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
+1.769474009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet! Its size: 65507
+1.769474009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Number of packets now: 5
+1.829627609s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
+1.875882009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet! Its size: 65507
+1.875882009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Number of packets now: 6
+1.929627609s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
```

```
Client update duration:  0.2510066032409668
Client id:  1
Evaluate...
Evaluated on 5000 samples (50 batches)
Evaluated on 10000 samples (100 batches)
{'communication_round': 2, 'lr': 0.1, 'epoch': 1, 'iteration': 10}
{'loss_train': 0.9039141798019409, 'accuracy_train': 0.7656}
{'loss_test': 0.9009176337718964, 'accuracy_test': 0.7693}
{'bits_sent_up': [936360, 936360], 'bits_sent_down': [936360, 936360]}
Saved results to  results/experiment1/xp_75977.npz
Remaining Time (approx.): 00:03:19 [10.00%]
```

```
+2.620738009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet! Its size: 65507
+2.620738009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Number of packets now: 13
+2.727146009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet! Its size: 65507
+2.727146009s 4 FirstScriptExample:ReceivePacket(): [INFO ] Number of packets now: 14
+2.816899609s 2 FirstScriptExample:StopApplication(): [INFO ] Stopped Application at +2.8169e+09ns
+2.816899609s 2 FirstScriptExample:StartApplication(): [INFO ] Get Socket Node: 3
+2.816899609s 2 FirstScriptExample:SendPacket(): [INFO ] Sender Client Id: 1Communication Round:2
+2.816899609s 2 FirstScriptExample:SendPacket(): [INFO ] About to send the packet with the size of: 936360
+2.816899609s 2 FirstScriptExample:SendPacket(): [INFO ] About to this number of packets: 14
+3.067906212s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
+3.167906212s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
+3.182120612s 4 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet! Its size: 65507
+3.182120612s 4 FirstScriptExample:ReceivePacket(): [INFO ] Number of packets now: 15
+3.267906212s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
+3.288528612s 4 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet! Its size: 65507
+3.288528612s 4 FirstScriptExample:ReceivePacket(): [INFO ] Number of packets now: 16
+3.367906212s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
+3.394936612s 4 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet! Its size: 65507
+3.394936612s 4 FirstScriptExample:ReceivePacket(): [INFO ] Number of packets now: 17
+3.467906212s 2 FirstScriptExample:SendFragment(): [INFO ] Sending fragment...
+3.501344612s 4 FirstScriptExample:ReceivePacket(): [INFO ] Received one packet! Its size: 65507
+3.501344612s 4 FirstScriptExample:ReceivePacket(): [INFO ] Number of packets now: 18
+3.567906212s 2 FirstScriptExampl    Fragment(): [INFO ] Sending fragment...
+3.607752612s 4 FirstScriptExam               [INFO ] Received one packet! Its size: 65507
```

Figure 3.8 : Python and NS3 processes first and second communication round.

3.4.4 Dumbbell Topology with 8 Nodes

This time the number of clients are increased to 5 in order to accuracy in Figure 3.9.

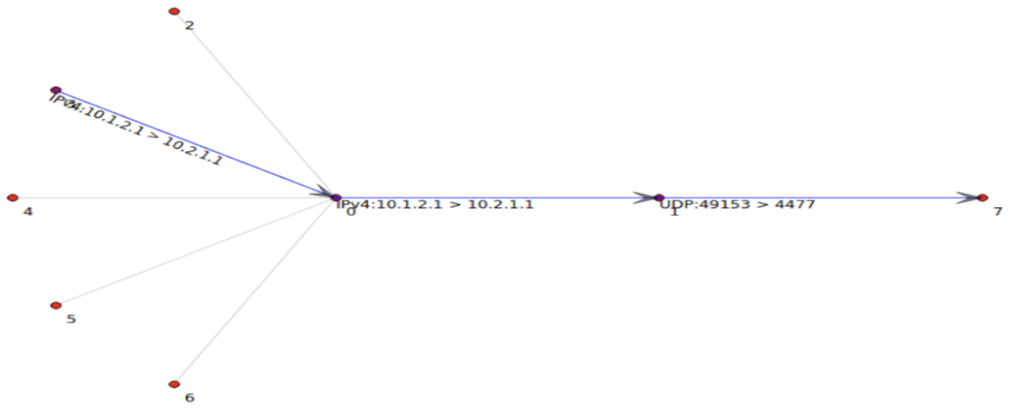


Figure 3.9 : Dumbbell topology with 8 nodes in which client with ID 3 is transmitting.

Using trace metrics in NS3, client with ID 2 Tx (transmitter) and server Rx (receiver) values are plotted using packet byte count on y axis and time on x axis. Client is being selected in particular rounds and it transmits only in specific time intervals in Figure 3.10.

In server side however, please notice that it receives most of the time, since communication rounds continue one after another without any breaks and there is no packet drop in this experiment.

This experiment is simulated and the results are plotted in Figure 3.12. The communication rounds, time intervals and which clients are involved can be seen. In addition to that, client update calculations and server aggregations can be observed as well. Since there is no background traffic and not many clients are involved, the convergence is reached nearly at the 3rd round. It is important to note that IID data distribution is applied in this experiment, and that is why the increase in the accuracy is linear.

3.4.5 Dumbbell Topology with 9 Nodes with Background Traffic

An additional node is added to the topology in Figure 3.9 on the left hand side which is acting as a background traffic generator node. This node is not included in federated

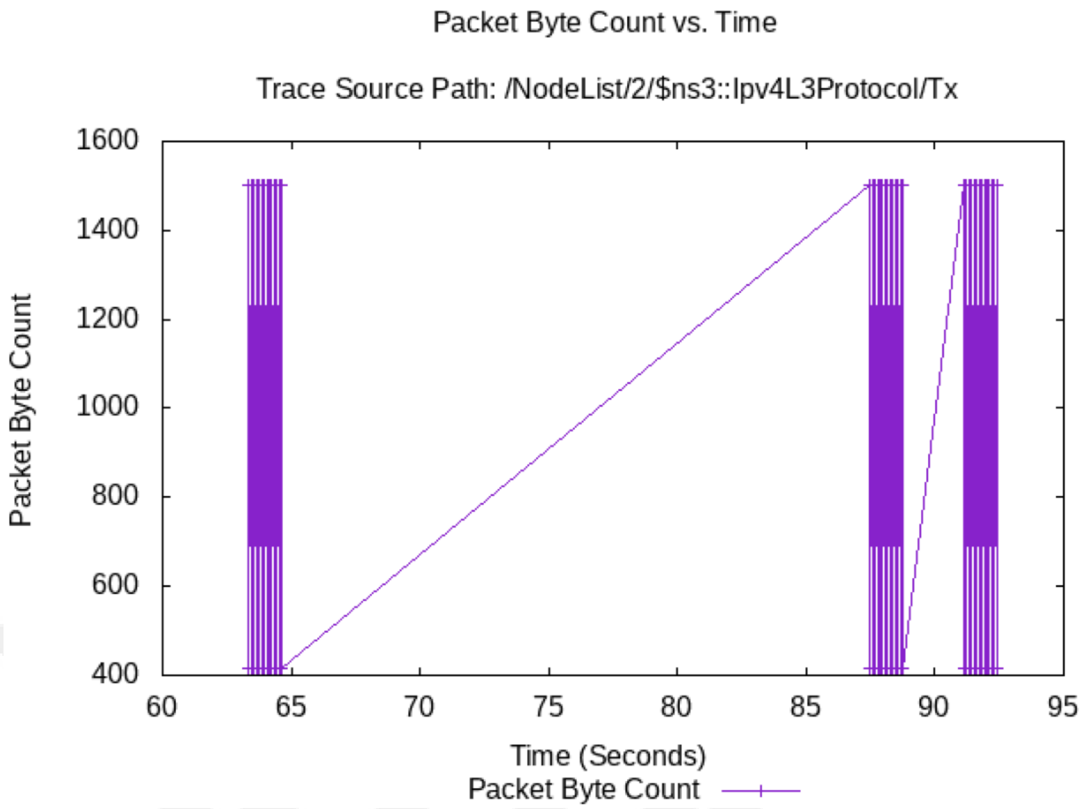


Figure 3.10 : Client transmission plot in discrete time intervals.

learning process, instead it only sends packets to the server through the network in order to fill the queue in the bottleneck. The node transmits only in specific time periods between 1-11 and 51-61 seconds and then it halts as can be seen in Figure 3.13.

It is expected to see its consequences in the federated learning accuracy and indeed the Figure 3.14. Instead of more stable increase like in Figure on the right, drops on the accuracy are observed on the left in later rounds as well. This is because of the fact that some of the clients were unable to send their unique gradient update models and it results in accuracy drops and it delays the convergence. In the end, with traffic 0.65 accuracy and without traffic almost 0.9 accuracy is achieved.

3.4.6 Dumbbell Topology with 104 Nodes

Client number is increased to 100 nodes and the topology is equal to the topology in Figure 3.1 from this point onward. With and without background traffic is inspected with respect to packet drop rate and the frequency of the background traffic is increased. The results can be observed in Figure 3.15. The arrows are pointing out

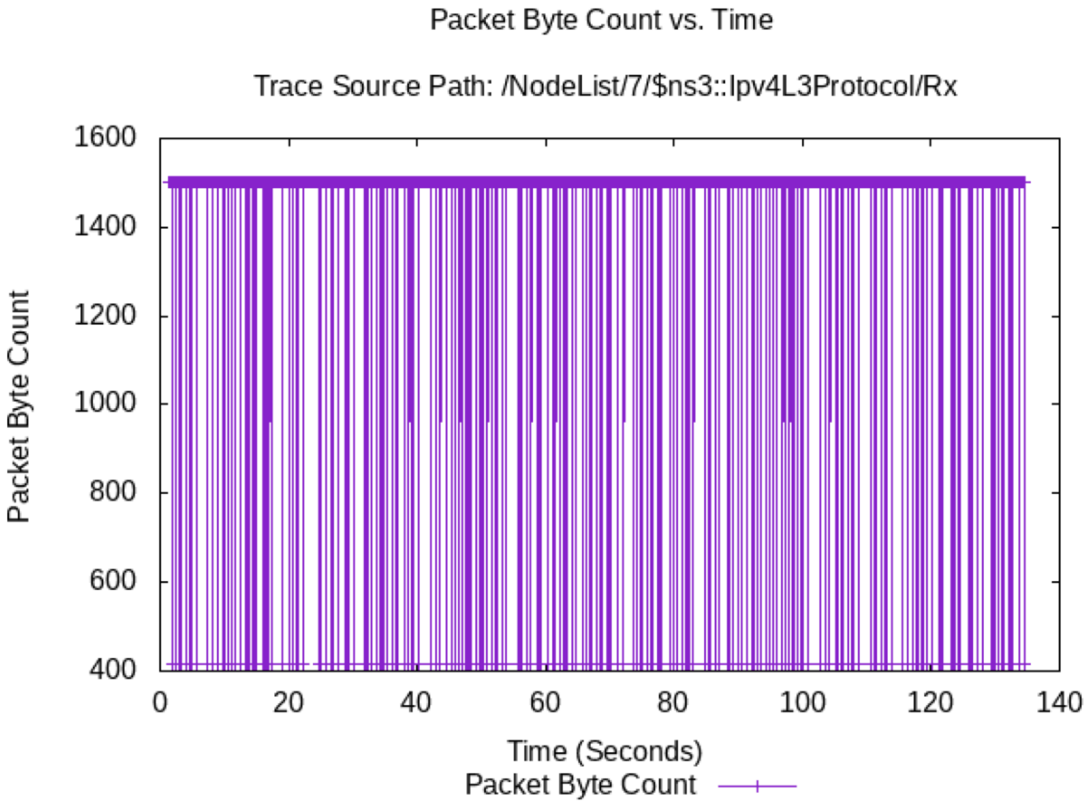


Figure 3.11 : Received packets by server is plotted.

the drawbacks that is caused by the background traffic and it shows that the immediate effects of the background traffic as well. Packets that are dropped or sent successfully can be observed in the histogram on the right. Packet drops are in parallel with accuracy drops and therefore it is worth to dive deeper into the relationship between them. The experiment with this configuration takes almost an hour to be finished on Macbook Pro with 2,3 GHz 8-Core Intel Core i9 processor and 16 GB 2667 MHz DDR4 memory.

3.4.7 Simulation with Constant Traffic and non-IID Setting

After having observed the consequences of the background traffic with IID data distribution, the traffic model is reworked and it is made as constant. 200 background traffic packets are scheduled with 100 milliseconds interval at every 100 seconds. The MNIST data is distributed as non-IID for the first time and the results on the accuracy change drastically. The non-IID data distribution on 100 clients which is explained in Section 2.1 can now be observed in the terminal output in Figure 3.16. After this point, non-IID data distribution is applied in the experiments.

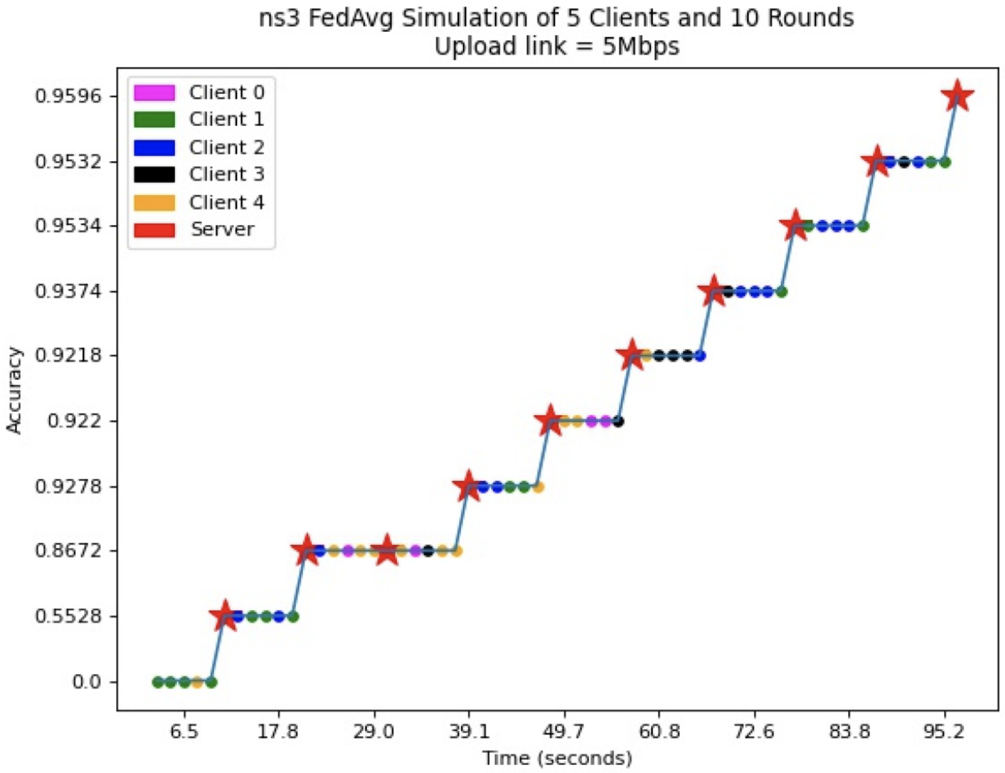


Figure 3.12 : Accuracy values of both clients and server are given with respect to virtual time.

This setting implies that each client gets only fraction of the data and because of that the learning process becomes quite slow. Yet, the background traffic appears to cause huge delay to the convergence as it can be seen in the Figure 3.17. In addition to that, the effect of non-IID on federated learning extends the duration of both virtual and real simulation time exceptionally in Figure 3.17 compared to Figure 3.15. As it took almost 5 hours in the same machine to finish this experiment.

The run with no background traffic converges to 0.9 accuracy in given time, whereas the run with the background traffic does not manage to converge and it has a slow start too. It may have dire effects on time-sensitive and real-time federated applications such as the scenario that is mentioned in Section 3.1.

3.4.8 Setting Seed Value for the Simulation

For reproducibility and averaging of the results, a seed must be set for the experiments. Three different seed values are set in the experiments in Figure 3.18. As pointed out in Figure 3.18, there is drastic changes between different seeds and it is because of

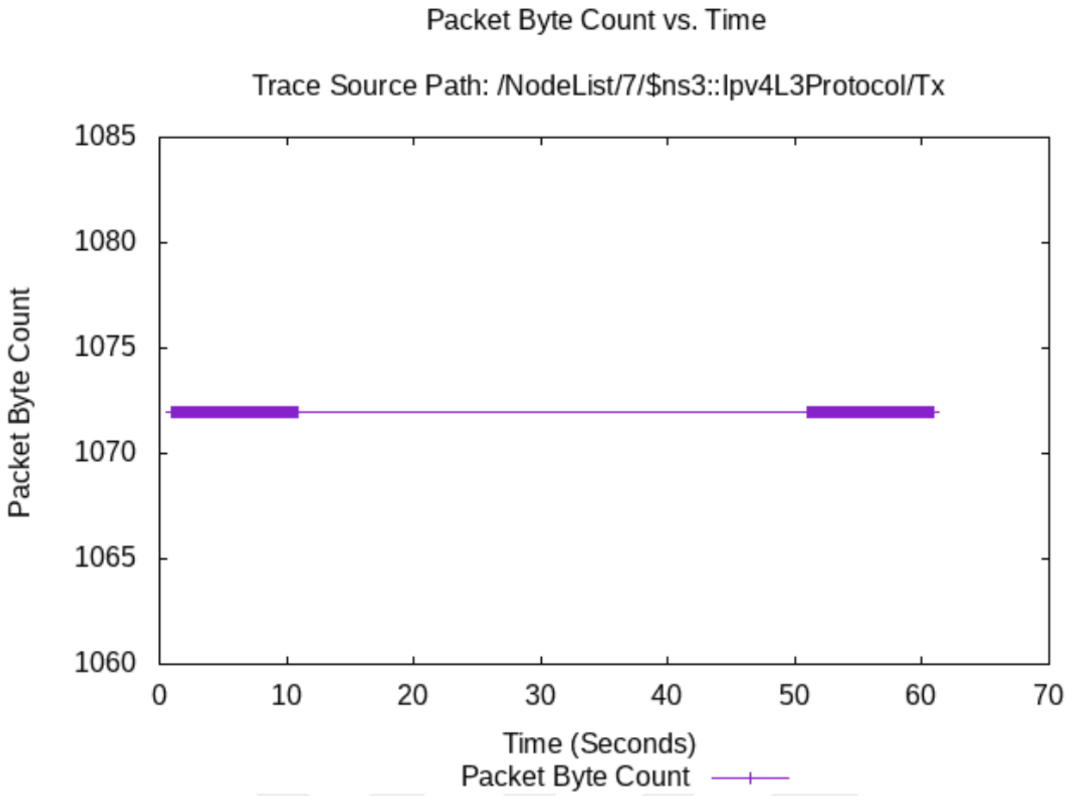


Figure 3.13 : The packet byte count over time plot of the background traffic generator.

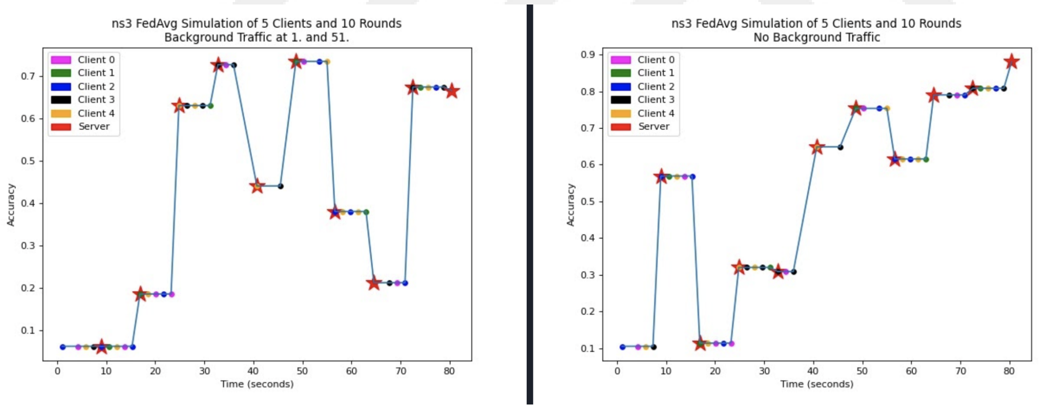


Figure 3.14 : Runs with and without background traffic side by side.

the fact that the data distribution and the randomly selected clients are connected to the seed. As seed changes, different configurations and possibilities emerge. The need for both the averaging and applying confidence intervals arise since the plots are not smooth and varies a lot.

Accuracy itself is not enough to judge the performance of the machine learning models, therefore average cross entropy error and maximum error plots are displayed in Figure 3.19. It is asserted in [5] that maximum error percentage is extracted by finding the

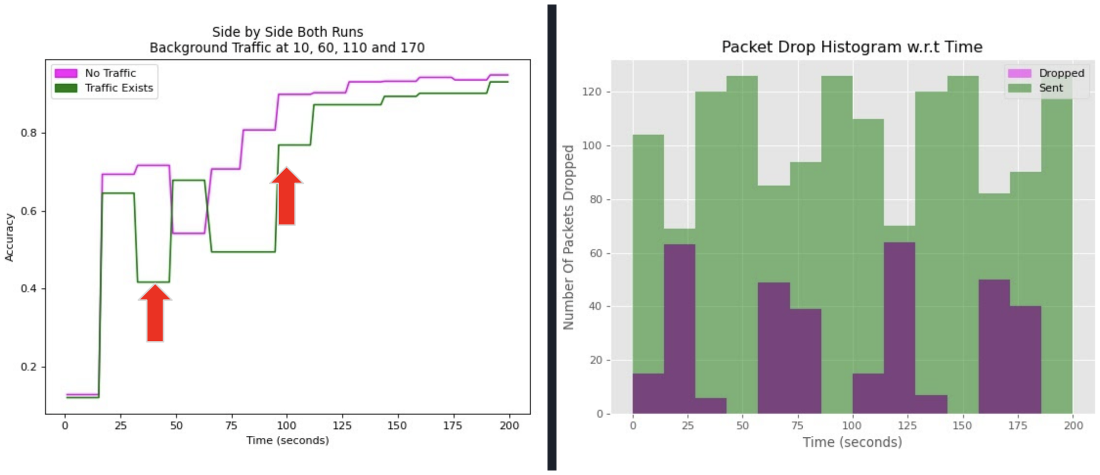
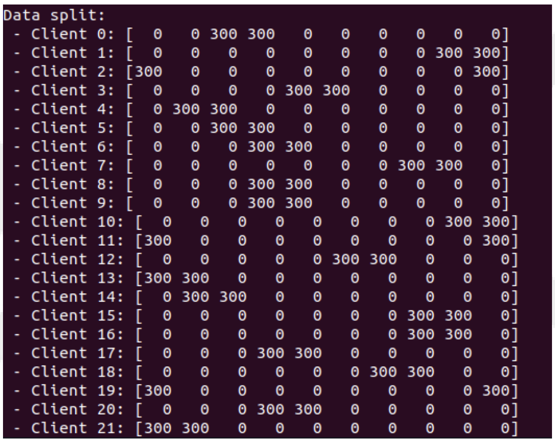


Figure 3.15 : Runs with and without background traffic side by side including 100 clients and also packet drop histogram can be observed.



```
Data split:
 - Client 0: [  0   0 300 300   0   0   0   0   0   0]
 - Client 1: [  0   0   0   0   0   0   0   0 300 300]
 - Client 2: [300   0   0   0   0   0   0   0   0 300]
 - Client 3: [  0   0   0   0 300 300   0   0   0   0]
 - Client 4: [  0 300 300   0   0   0   0   0   0   0]
 - Client 5: [  0   0 300 300   0   0   0   0   0   0]
 - Client 6: [  0   0   0 300 300   0   0   0   0   0]
 - Client 7: [  0   0   0   0   0   0   0 300 300   0]
 - Client 8: [  0   0   0 300 300   0   0   0   0   0]
 - Client 9: [  0   0   0 300 300   0   0   0   0   0]
 - Client 10: [  0   0   0   0   0   0   0   0 300 300]
 - Client 11: [300   0   0   0   0   0   0   0   0 300]
 - Client 12: [  0   0   0   0   0 300 300   0   0   0]
 - Client 13: [300 300   0   0   0   0   0   0   0   0]
 - Client 14: [  0 300 300   0   0   0   0   0   0   0]
 - Client 15: [  0   0   0   0   0   0   0 300 300   0]
 - Client 16: [  0   0   0   0   0   0   0 300 300   0]
 - Client 17: [  0   0   0 300 300   0   0   0   0   0]
 - Client 18: [  0   0   0   0   0   0 300 300   0   0]
 - Client 19: [300   0   0   0   0   0   0   0   0 300]
 - Client 20: [  0   0   0 300 300   0   0   0   0   0]
 - Client 21: [300 300   0   0   0   0   0   0   0   0]
```

Figure 3.16 : Non-IID data distribution in which a client only gets 600 samples which represents only two digits from MNIST dataset.

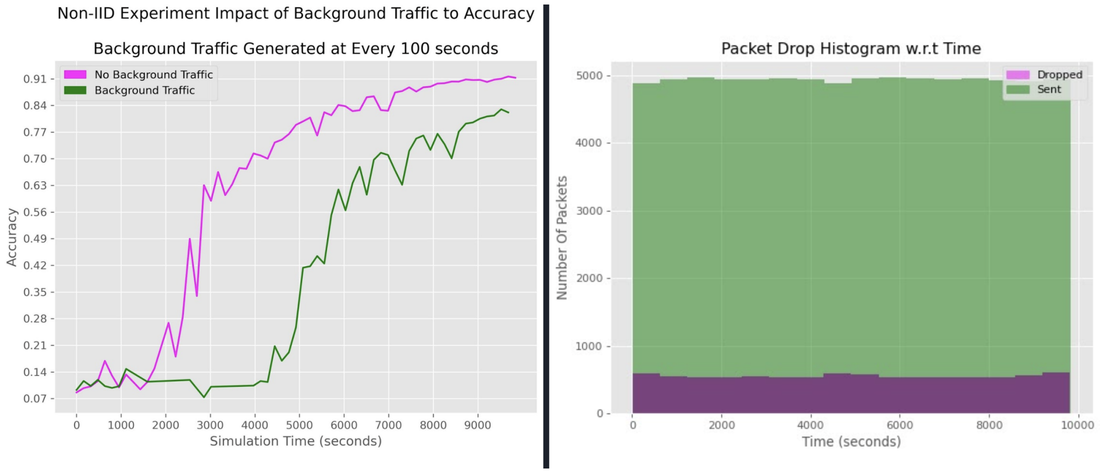


Figure 3.17 : With and without the background traffic experiment with Non-IID data distribution.

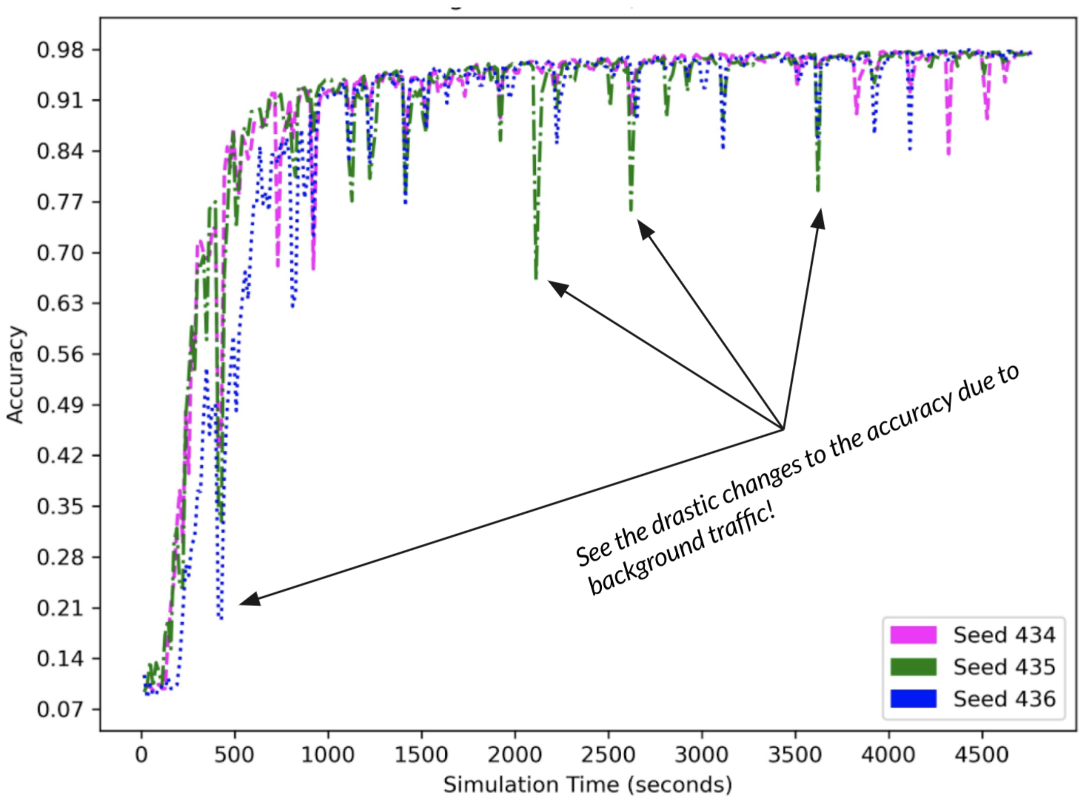


Figure 3.18 : Different seed values and accuracy results under constant background traffic.

most erroneous batch against test data in the server. The most erroneous batches are plotted against communication rounds.

3.4.9 Simulation with Poisson Distribution Based Background Traffic

Since constant traffic does not comply with the real world traffic models, Poisson distribution based background traffic is generated at the background traffic generator node. It is bound with the seed value in order to protect reproducibility, and the experiments are ran multiple times. It is done using "ExponentialRandomVariable" in NS3, which enables simulation to produce exponential distribution based on the Poisson arrival rate lambda. Given lambda value to the simulation, arrival rate changes and it is examined in order to create high, medium and light intensity background traffic.

With different interarrival time values that are ranging from 0.2 to 0.8, the success percentage and accuracy plots are given in Appendix A and network metrics such as data rate, number of packets generated etc. are presented in Appendix B. The success

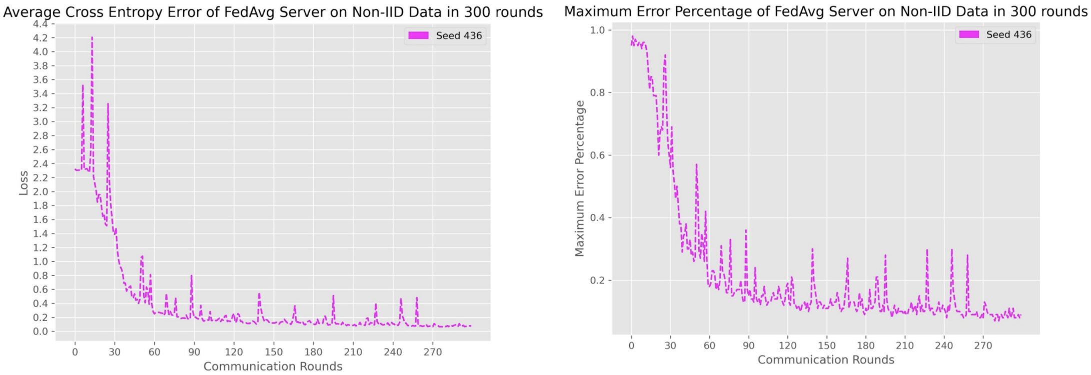


Figure 3.19 : Both average maximum and cross entropy error under fixed background traffic for the seed 436.

percentage is the delivery ratio of the clients who manage to send their gradient updates successfully. For a communication round in this experiment, 10 clients are expected to send their updates, but if only 7 of them are arrived to the server then the success percentage of the communication round becomes 0.7.

By examining these results, possible high, medium and light traffic intensities are investigated and their corresponding arrival rate values are determined. As arrival rate decrease, traffic gets intense therefore accuracy reduces and the success percentage decreases. Cumulative averaged results are given in Figure 3.20 and in Figure 3.21. Average success percentages in Figure 3.20 correlated with the accuracy results, as it can be seen in both figures. When success percentages are compared to the results in Appendix A and Appendix B, they both fluctuates, yet it stays in an interval which equals to 0.3 at maximum. In addition to that, the averaged accuracy results yield much smoother plots as in Figure 3.21 which reveals the drawbacks in performance of the federated learning much clearer especially when the interarrival rate is below 0.40.

It was the last experiment before proceeding to selection of the best representing interarrival rate values for the high, medium and low intensity background traffic. Interarrival rate term is later converted to interarrival time which make much sense. This means that the experiment is finalized and final simulation model is discussed in Section 3.5 and the results that are published in [5] are shared in Section 4.

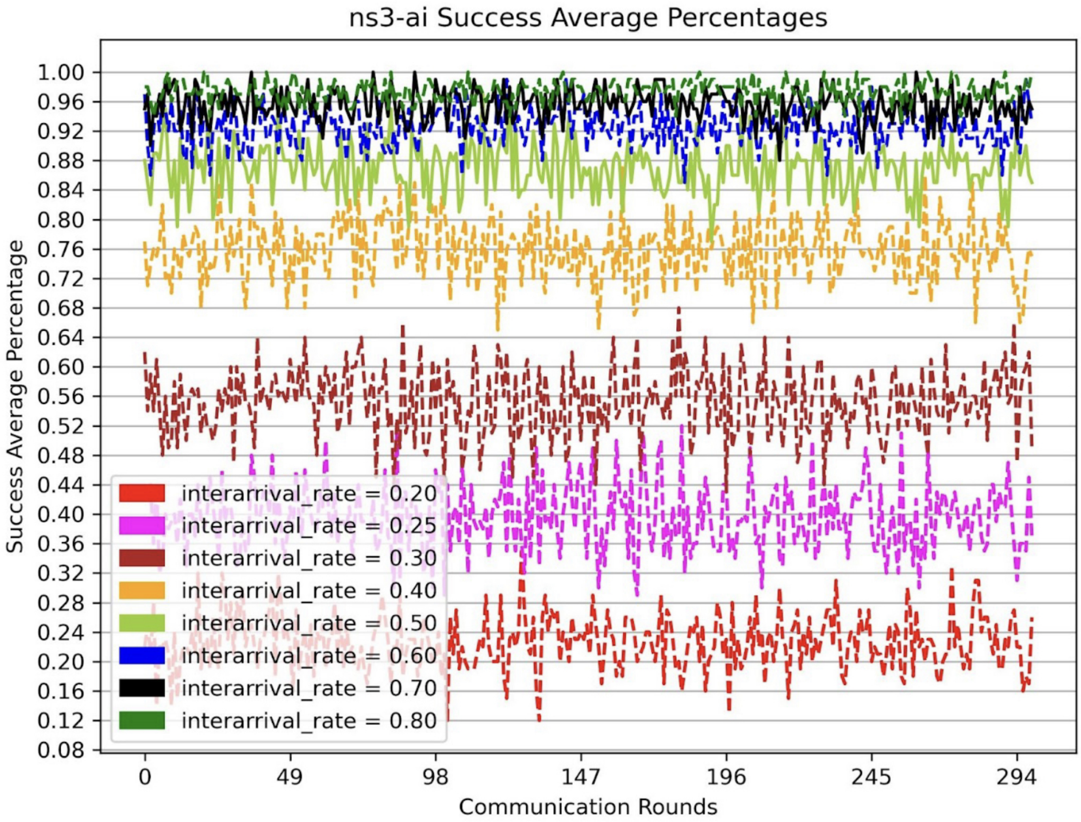


Figure 3.20 : Average success percentages of 10 runs for each arrival rate.

3.5 Final Simulation Model

The flow of the communication in the topology is modeled including QoS such as packet drop rate and success percentage so that the effects of wireless channel can be observed. Since Poisson distribution is used for the background traffic packet transmission, the confidence intervals for different packet/sec values is demonstrated using replication method in order to validate the results further in Section 4. It reveals the significance of hyperparameters in the network and the effects of fluctuations of the network. Applying different interarrival time values change the performance of the federated learning and the consequences on federated learning application are simple to observe thanks to NS3. The packet drop rate in terms of time and communication rounds in bottlenecks is plotted in success percentage format to present the effects of wireless channel metrics.

The simulations that are presented in federated learning [47] studies lack crucial network statistics. Because of no NS3 experiment is utilized before, the connection

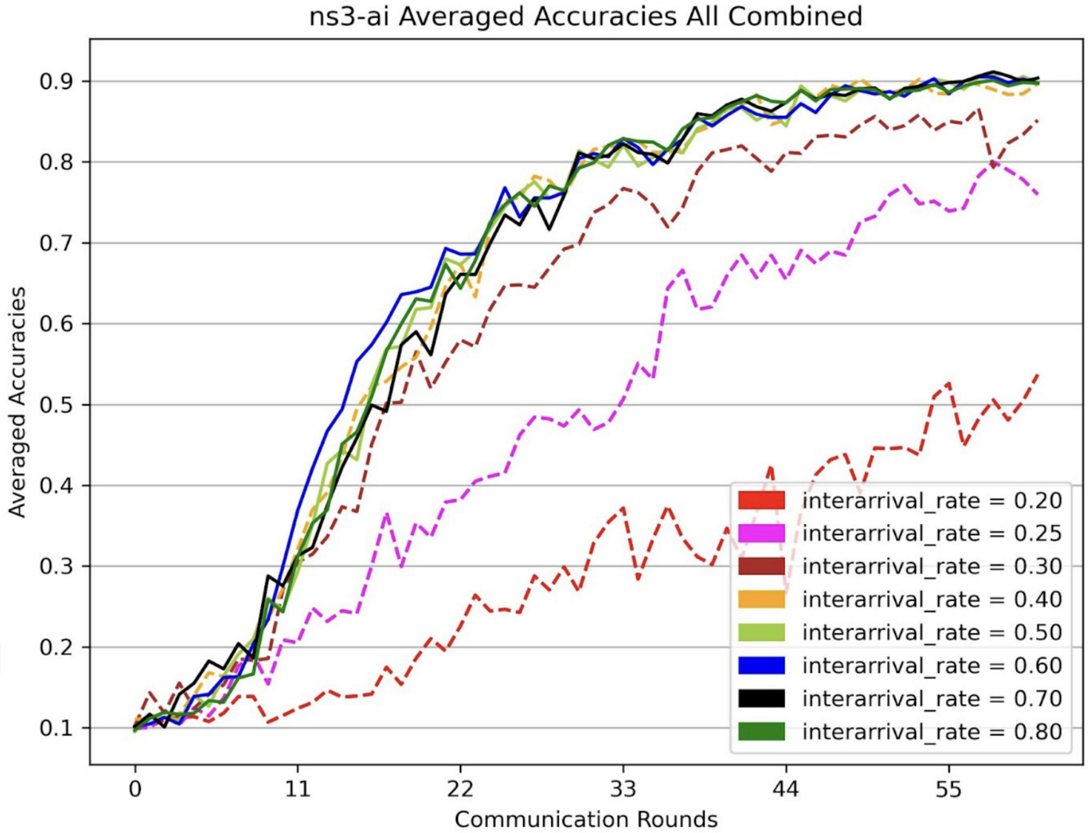


Figure 3.21 : Average accuracy of 10 runs for each arrival rate.

between packet drop rate, delay and convergence could not be shown. Feeding updates of the clients produced by Python process to NS3 has become the bridge between network metrics and the performance of the federated learning. For this, ns3-ai [30] framework is utilized. In fact, these updates of the clients are adapted to UDP packets and sent to server.

Only dumbbell topology is inspected in the thesis and an additional node is used for the background traffic generation via Poisson distribution process so that the packet drop rate and its consequences can be observed.

Federated Averaging [4] algorithm is used and MNIST digit data is distributed among clients with non-IID setting. The most successful hyperparameters are referenced from [4] and which are B = 10, E = 20 and can be seen in Table 3.1. The total number of clients are picked as 100. As the transport layer protocol UDP is adopted. Peer-to-Peer (P2P) data rate and delay are selected as 5 Mbps and 1 ms which is common for wireless applications in NS3. CNN with four layer with ReLu activation is carried out in the application [48]. The interarrival time is provided to the experiment as hyperparameter as well.

Table 3.1 : Hyperparameters of the ns3-ai FL Simulation

Federated Learning Related	
Dataset	MNIST digit data
Client Number (K)	100
Participation Rate	0.1
Number of Epochs on Local Dataset (E)	20
Batch Size (B)	10
Momentum	0.2
Learning rate	0.1
Digits per client	2 (Non-IID)
Number of Rounds	300
Network Related	
Transport Layer Protocol	UDP
Data rate p2p	5 Mbps
Delay on p2p	1 ms
Background Traffic Packet Size	1024 bytes
Interarrival Times of Background Traffic	250, 300, .., 900 ms

4. SIMULATION RESULTS

The finalized and refined results that combine the network and federated learning with clear explanations and plots are given in this section. The simulation model in Section 3.5 is utilized. These results are also published and presented in [5].

4.1 Reproducibility on Average Cross Entropy Error and Accuracy

Because of the randomness that is brought naturally by non-IID setting, client selection in Federated Averaging algorithm and the background traffic generation using Poisson distribution, the validation of reproducibility of simulation is essential. Seed settings and multiple runs with different seeds progression is discussed in 3. Following that, these multiple runs are validated by comparing both average cross entropy error and accuracy which can be observed in Figure 4.1 and 4.2.

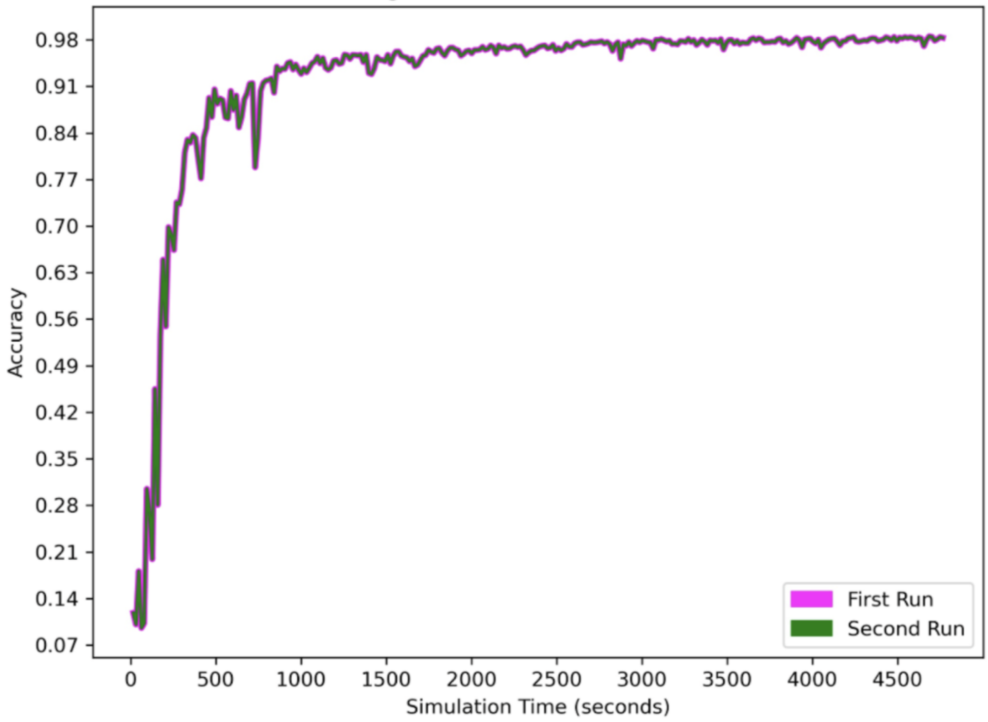


Figure 4.1 : Accuracy of first and second run concerning simulation time that is supplied with same seed value.

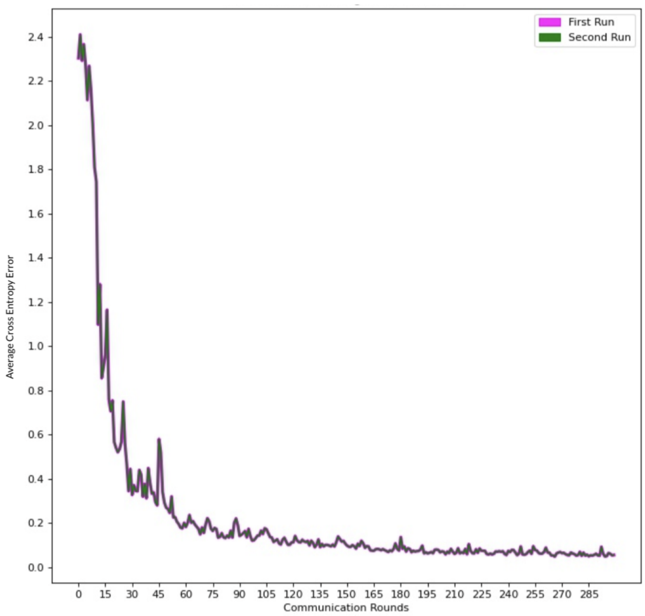


Figure 4.2 : Average cross entropy error of first and second run concerning communication rounds that is supplied with same seed value.

4.2 Maximum Error Percentage

See the Table 4.1 for set of technical expressions that are used throughout in this Section. One of them, maximum error percentage, is a great representation of the performance of federated learning in worst case. It is observed that the worst case performance is affected by the background traffic tremendously as it can be seen when the interarrival time is 250 ms around round 90 in Figure 4.3. Beyond communication round 90, the difference in error is more visible as 900 ms interarrival time has maximum error percentage between 0.1 and 0.2, whereas 250 ms interarrival time has an error rate more than 0.8 [5]. It shows the adverse effects of the accuracy of the federated learning model after so many rounds when the background traffic is heavy.

4.3 Average Success Percentage

Among 100 clients, 10 of the clients are randomly selected and are expected to send their updates as discussed in 3. If 10 of them have sent all of their UDP packets successfully, then the success percentage becomes %100. However, this is not the case when the background traffic is present and the 10 run averaged success percentages of high, medium and light intensity traffic results can be observed in Figure 4.4. As indeed, average success percentage affects the performance of the federated learning

Table 4.1 : Set of technical expressions for this Section.

Terminology	
Communication round	Clients run stochastic gradient descent locally using only their data and they produce gradient updates. Selected clients transmit their gradient updates at the next step to the aggregator server. These gradient updates are aggregated and the aggregated model is transmitted back to the clients. This is called a communication round which starts with first step and finishes when agreggated model is transmitted back to the clients.
Background traffic generator node	Except for all of the clients that will participate in improving the caching of the popular content in the app, there is an additional node: Background Traffic Generator Node. This node causes background traffic using Poisson distribution process.
Interarrival time	Since constant traffic does not comply with the real world traffic models, Poisson distribution based background traffic is generated at the background traffic generator node. Based on the Poisson arrival rate lambda, given lambda value to the simulation, arrival rate changes and it is examined in order to create high, medium and light intensity background traffic.
Maximum error percentage	Maximum error percentage is extracted by finding the most erroneous batch against test data in the server. The most erroneous batches are plotted against communication rounds.

application due to non-IID setting, since each client has only limited amount of samples which are representing only two digits. The information which are carried by UDP packets is lost if the packet is dropped and more gradient updates are discarded as the traffic gets intense. Its effect on maximum error is already been observed in Figure 4.3 and the effects on the accuracy are presented in Section 4.4.

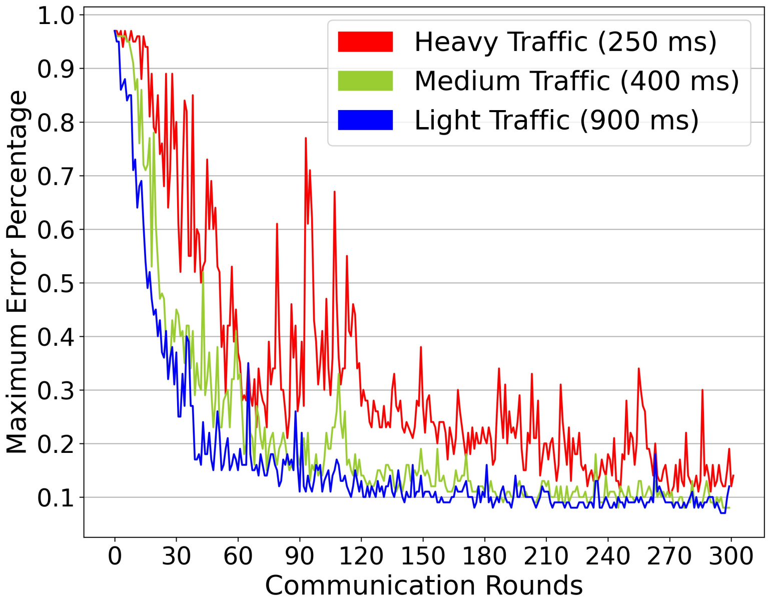


Figure 4.3 : Maximum error percentages of various predefined traffic intensities that is set according to the comments in Section 3.4.9.

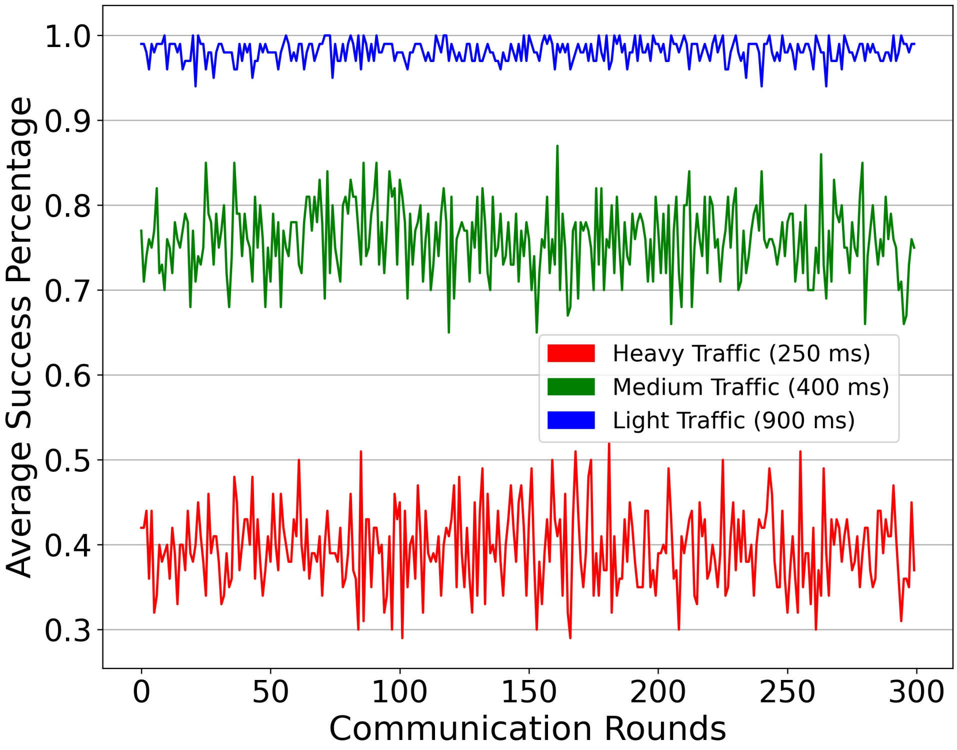


Figure 4.4 : Various traffic intensities and their average success percentages concerning communication rounds.

4.4 Average Accuracy

The packet drops reduce the accuracy in general and this can be observed in Figures 4.5 and its cut version Figure 4.6. Discarded UDP packets weaken the accuracy of the federated learning especially in the first 200 rounds when the traffic is heavy. The convergence is delayed enormously, as heavy traffic run reaches 0.9 around at 120 communication round compared to 60 communication round which is achieved by light traffic run. In fact, this is critical for the scenario which is mentioned in Section 3.1 and other delay sensitive applications [5]. Although medium and light traffic convergence closely, the maximum error gap is still huge which is presented in Figure 4.3.

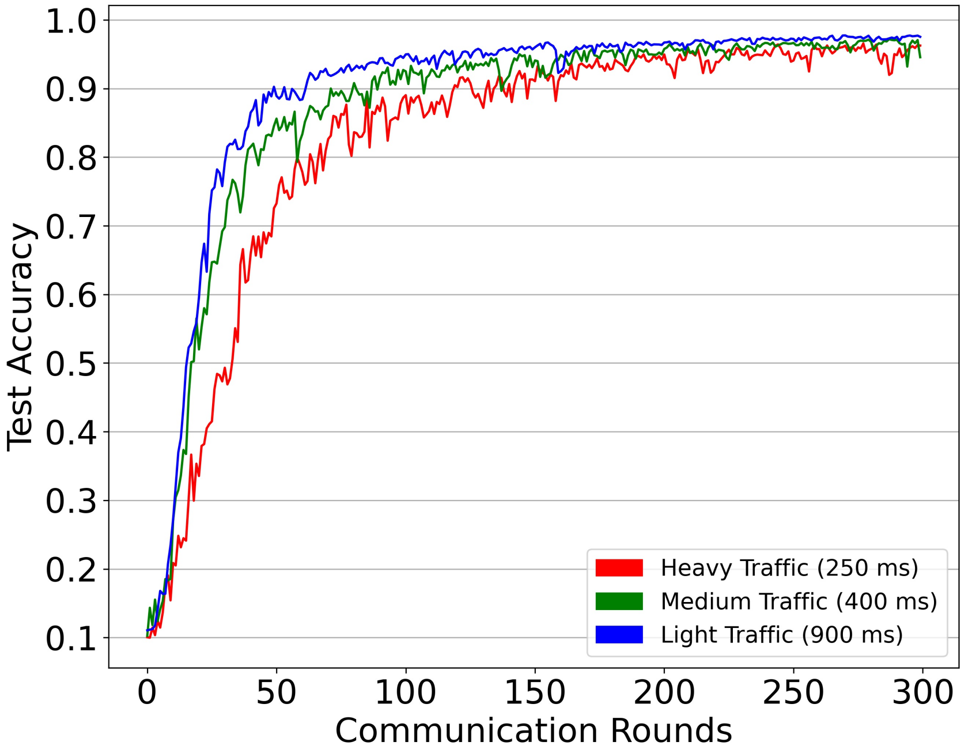


Figure 4.5 : Different traffic intensities that change test accuracy values drastically.

4.5 Confidence Intervals

Due to non-IID setting and number of communication rounds, the application is expected to converge at some point after round 200 since most of the gradients that are

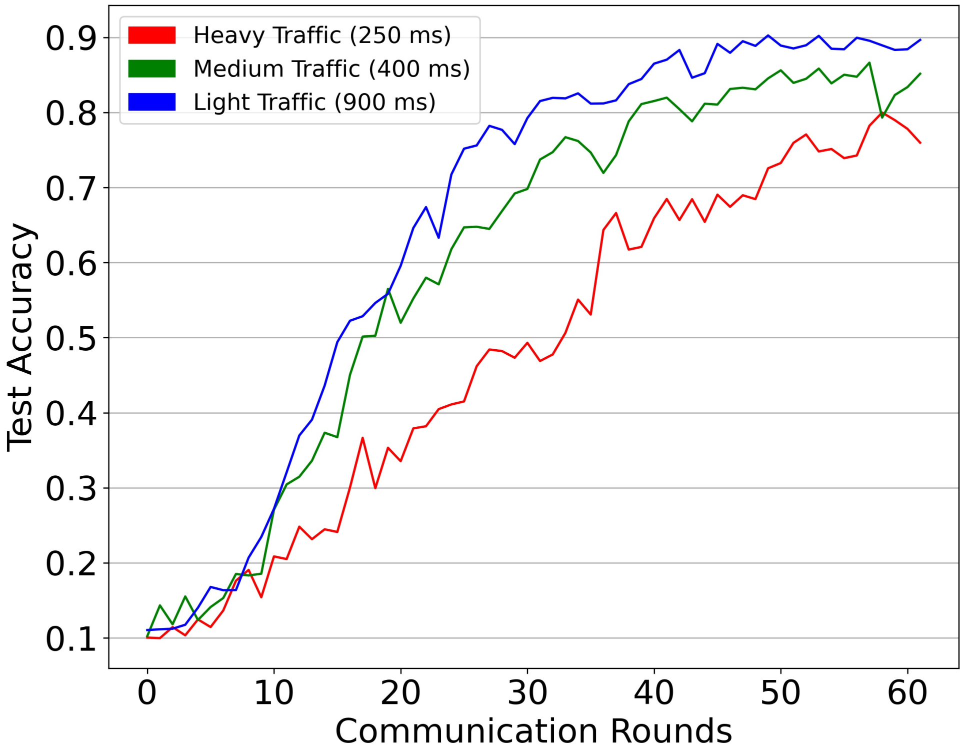


Figure 4.6 : Minimized version of Figure 4.5 which presents 60 communication rounds instead of 300.

carrying different digits information must have been sent to the server. This is validated in Figure 4.7, along with wide intervals that show the diversity of the accuracy corresponding to multiple simulation runs with different seeds. Most diversity is seen in earlier rounds which shows the effects of the randomness of non-IID setting. As the background traffic gets denser, intervals widens due to frequent packet drops.

4.6 Network Metrics

At the end, the interarrival time hyperparameter that are provided to the experiment and the resulting interarrival time of the packets that are observed by the simulation are compared and they are shown in Table 4.2. These are NS3 traces which corresponds to the data rate of the client, the total amount sent by the traffic generator.

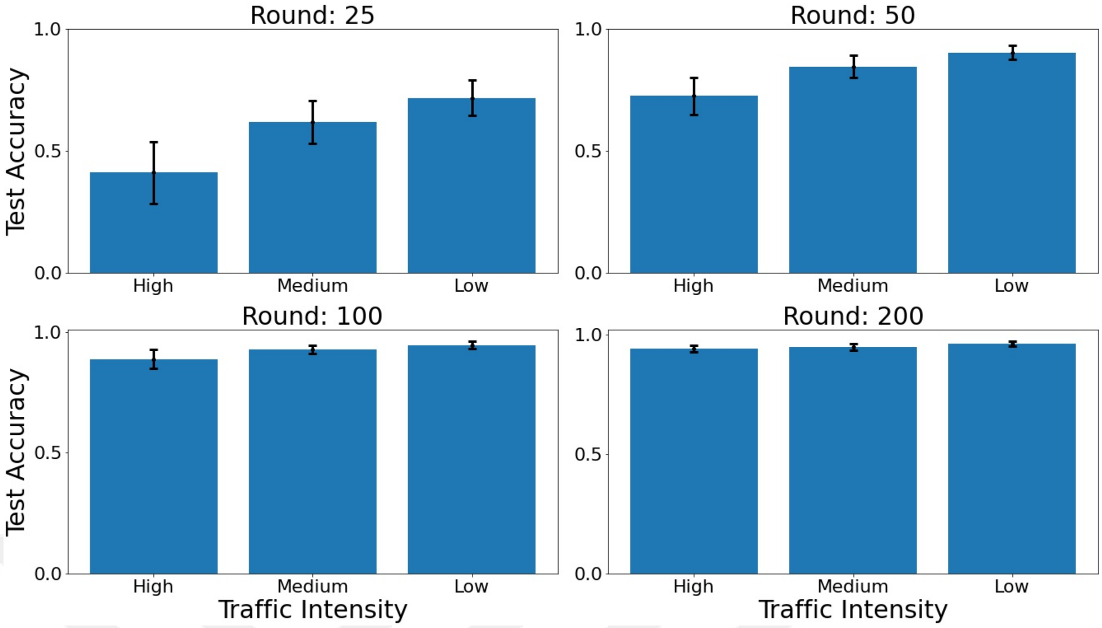


Figure 4.7 : Confidence intervals and test accuracy values for different traffic intensities at rounds 25, 50, 100 and 200.

Table 4.2 : Network statistics are gathered concerning the interarrival time of the background traffic.

Interarrival Time (Hyperparameter) [ms]	Data Sent by the Traffic Generator [kB]	Number of Packets Generated by the Traffic Generator	Data Rate [kB/s]	Interarrival Time (Observed) [ms]
250	19199.8	18832	4.0	253
300	16052.5	15745	3.4	303
400	12017.2	11787	2.5	404
500	9730.4	9544	2.0	499
600	8095.1	7940	1.7	600
700	6942.0	6809	1.5	700
800	6075.4	5959	1.3	800
900	5375.0	5272	1.1	904

5. ADAPTIVE PARTICIPATION FEDERATED LEARNING

Given the consequences of the background traffic, a simple network based approach is given in this section. The participation rate hyperparameter is considered and it is adjusted according to the state of the network. In this chapter, the congestion in the network is detected by NS3 via analyzing the success percentage in real-time. It is desired to have a responsive and robust FL algorithm [49] against packet drops. With respect to the %30 to %50 success average under heavy traffic in Section 4.3, the threshold value for the success percentage is set to %60 percent, hence whenever 6 or less client updates are received by the server in the communication round, the participation rate is adjusted. It is asserted in [50] that instead of static communication round configuration, the more adaptive solution must be fit to increase maximum success average. Therefore, it is aimed to increase the success of the federated learning algorithm under heavy background traffic by adapting it to the state of the network.

5.1 Adjustment of Both Communication Round Limit and The Participation Rate

To tackle the challenge of providing more gradient updates to the server, a couple of experiments are conducted to investigate the participation rate effects on accuracy under the background traffic. Without altering none of the hyperparamaters except for communication round limit in terms of simulation time and the participation rate, both of these values are doubled and divided by two. Interarrival time is set to 250 milliseconds and success percentages are given in Figure 5.1. Although the frequency of transmitting packets and communication round limit is changed, the results show that the success percentage is not affected overall. Therefore adjustment of both communication round limit and the participation rate at the same time is abandoned and the focus is shifted to the participation rate instead.

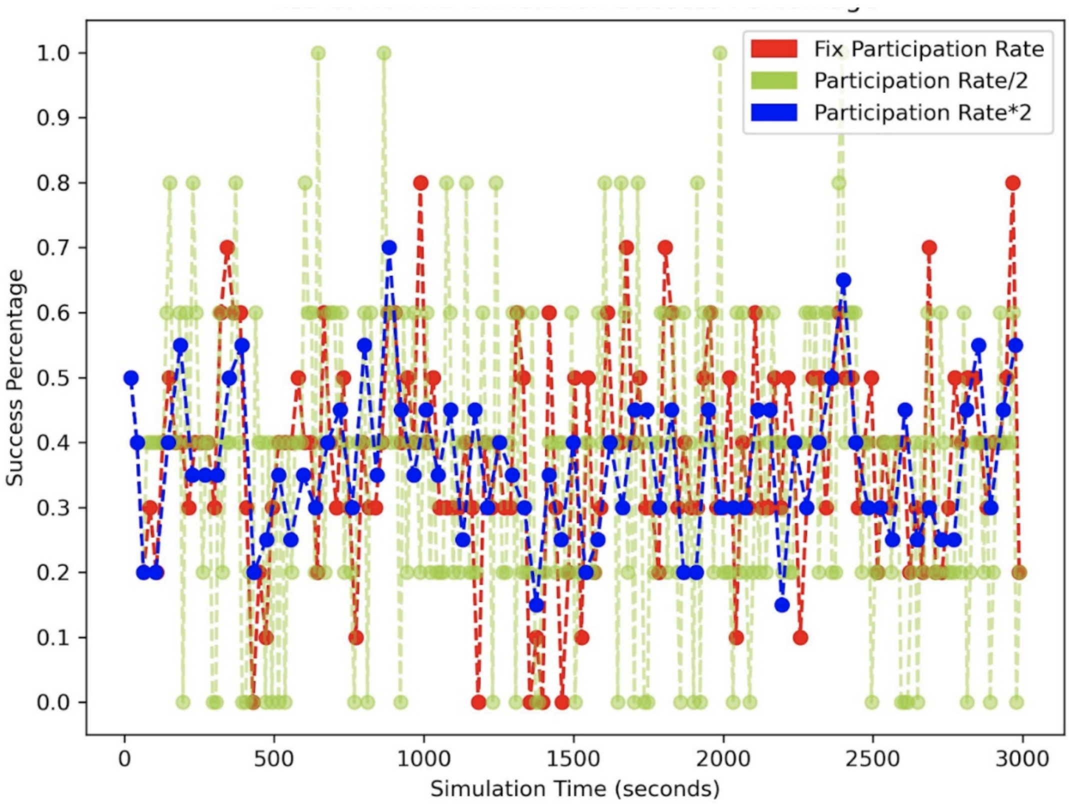


Figure 5.1 : Success percentages of different participation rate values.

5.2 Setting Maximum Number of Clients in Communication Round

Utilizing the advantages of the network side, transmission delay and queuing delay is calculated by clients. This information is sent to the server and the maximum number of clients in a communication round is calculated in equation 5.1.

$$N_{the\_maximum\_number\_of\_clients} = T_{round\_limit} / (T_{transmission\_delay} + T_{queueing\_delay}) \quad \textbf{(5.1)}$$

As discussed, if the success percentage threshold is less than %60 percent then the maximum number of clients in a communication round is calculated and a new participation rate is set by the server. According to the same setting in the previous section, the experiments are run and averaged. Number 18 is computed to be the maximum and three different client numbers in the communication round is presented in Figure 5.2. In fact, it seems that the success percentages does not change, the difference is definitely visible in Figure 5.3 as it shows the number of successfully sent clients.

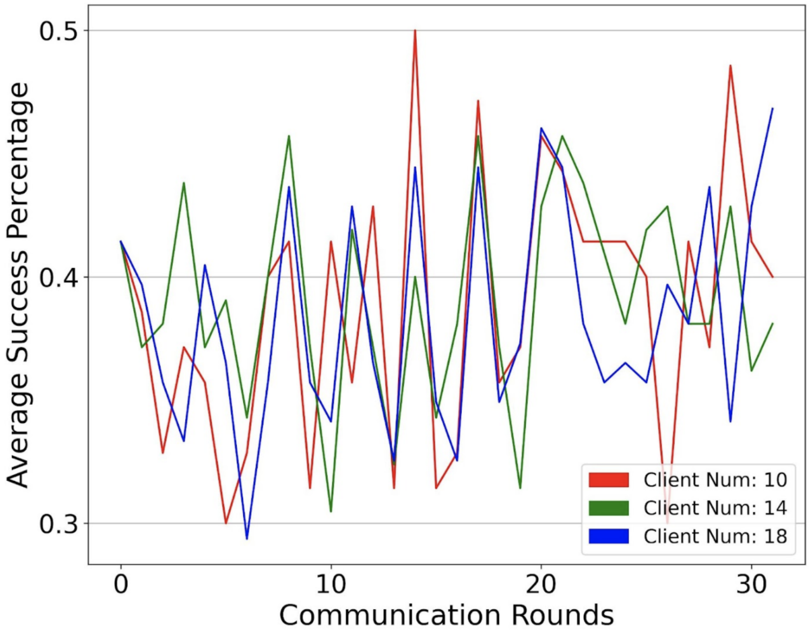


Figure 5.2 : Client numbers in communication round and the success percentages.

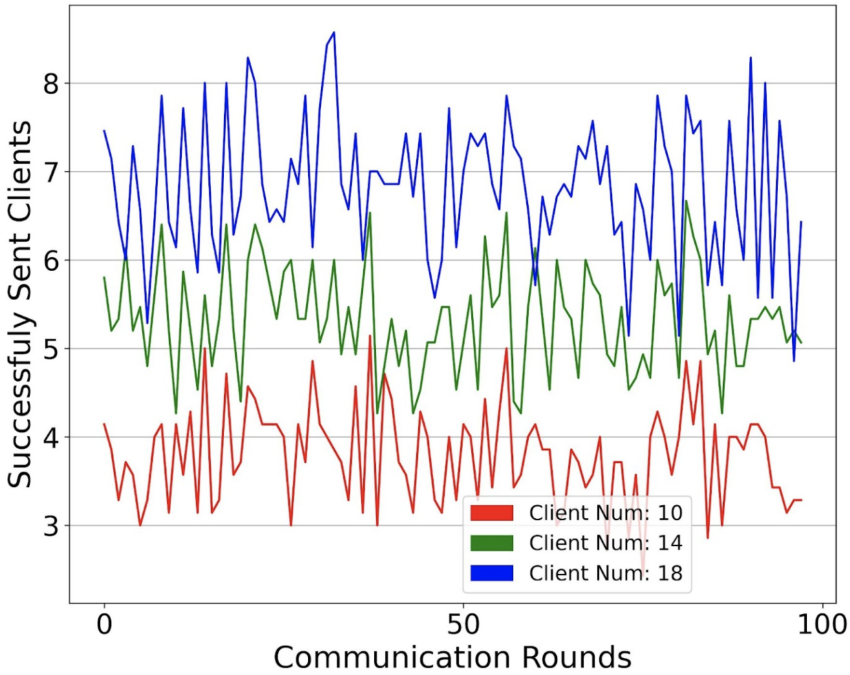


Figure 5.3 : Client numbers in communication round and successfully sent client numbers.

Impact of getting more client updates is visible in the accuracy plot and the improvement is obvious with the maximum number of clients in a communication round. It is presented in Figure 5.4, the convergence is much swifter than fix participation client number 10 compared to 14 and 18. In fact, in earlier rounds before 50, the accuracy gain is almost %50 percent and higher accuracy is achieved compared to fix participation rate in general.

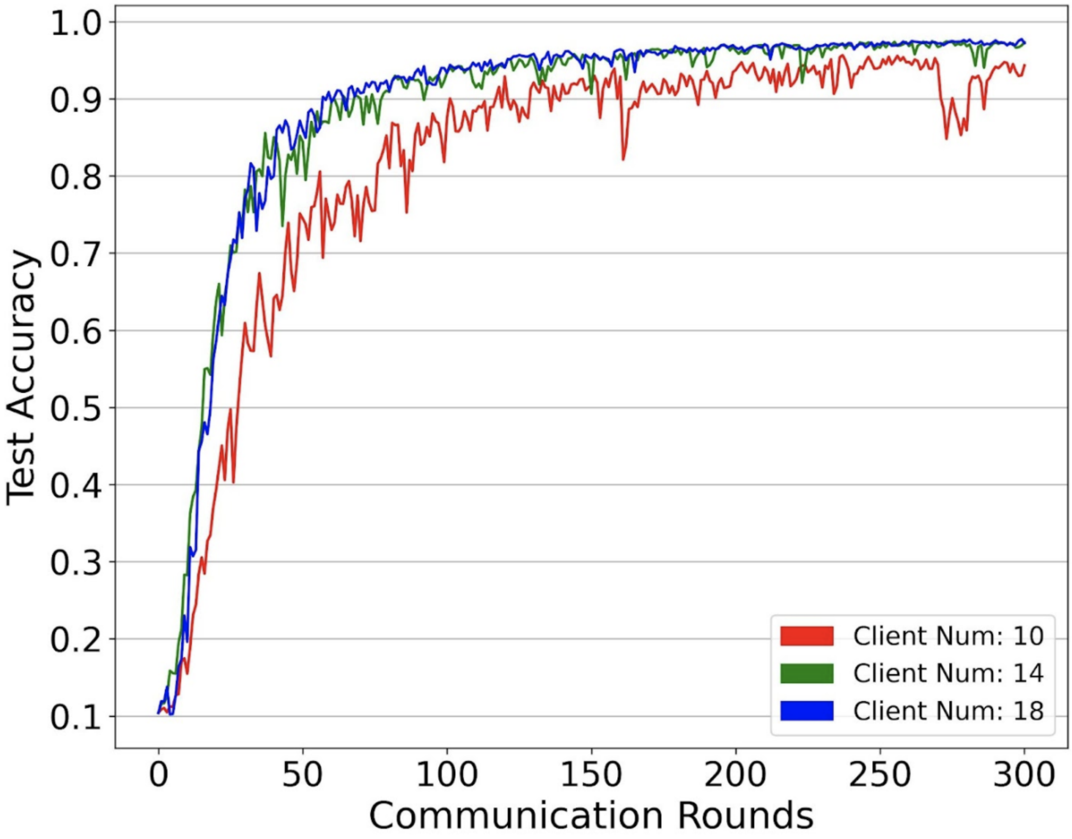


Figure 5.4 : Client numbers in communication round and the accuracy values.

6. CONCLUSION

Starting from early stages of decentralized learning, federated learning journey on the networks is presented. It is covered in detail as well as various type of networks that it is considered to be deployed. Its advantages in general, and its disadvantages with the network in particular are explained. It reduces concerns in diverse topics such as privacy, security and the communication overhead, but the network side remains uncovered.

The network side is investigated by combining NS3 and federated learning in order to test it against one of the QoS parameters which is packet drop rate. A background traffic is generated in a basic network topology and consequences on convergence of the federated learning are explored. Many simulation with different traffic intensities are executed and results are averaged utilizing replication method. As a result, average accuracy, average communication round success percentage, average cross-entropy error and maximum error are displayed with %95 percent confidence interval. It is observed that the success of the federated learning is prevented by at least 60 communication rounds under background traffic. In addition to that the heavy intensity background traffic causes the accuracy to be reduced by half in the first 30 communication rounds.

To this end, a network adaptive federated learning is proposed to tackle the issues related to QoS. Using the benefits of the network side, the maximum number of clients that can be sent in a single communication round is calculated if the communication round success percentage is reached below certain threshold. It is shown that huge accuracy gain is achieved via this approach and it is recorded as %50 percent in earlier communication rounds.

Given the results that are gathered via many simulation rounds and a network adaptive federated learning approach, a wireless channel metrics-aware adaptive federated learning which can be applicable for different types of federated learning is a great future direction. NS3 integration creates many possible developments and there are various network topologies such as vehicular networks that federated learning algorithms can be conducted and their QoS parameters can be observed.

The consideration of the computational power and various upload download rate of mobile devices in wireless channel conditions must be made and effective incentive mechanisms should be set to reward successful clients further.

Both mobility and energy consumption of the desired devices in federated learning can be included in the simulation so that the effects of them can be investigated since federated learning is not limited to mobile devices.

Certain data sets should be tested in a network simulations and especially the kind of data sets that contain videos, since QoS parameters turn into the key success of federated learning. This suits the scenarios such as AR/VR and vehicular applications.

Ultra-Reliable Low-Latency Communications (URLLC) challenges networks in terms of ultra-fast access and high reliability, thus the delay on both communication and computation must be examined thoroughly, and certain actions must be taken in order to meet the QoS for federated learning applications in 5G. Therefore there is a need for plenty of simulations to test federated learning in new generation wireless networks.

REFERENCES

[1] **Nishio, T. and Yonetani, R.** (2019). Client Selection for Federated Learning with Heterogeneous Resources in Mobile Edge, *International Conference on Communications (ICC), Shanghai, China pp*, pp.1–7.

[2] **Conway-Jones, D.**, **Tuor, T.**, **Wang, S. and Leung, K.K.** (2019). Demonstration of Federated Learning in a Resource-Constrained Networked Environment, *2019 IEEE International Conference on Smart Computing (SMARTCOMP)*, Washington, DC, USA, pp.484–486.

[3] **Chen, M.**, **Yang, Z.**, **Saad, W.**, **Yin, C.**, **Poor, H.V. and Cui, S.** (2021). A Joint Learning and Communications Framework for Federated Learning Over Wireless Networks, *IEEE Transactions on Wireless Communications*, *20*(1), 269–283.

[4] **McMahan, H.B.**, **Moore, E.**, **Ramage, D.**, **Hampson, S. and y Arcas, B.A.** (2017). Communication-efficient learning of deep networks from decentralized data, *Proc. of AISTATS*, Fort Lauderdale, FL, USA.

[5] **Eris, M.C.**, **Kantarci, B. and Oktug, S.** (2021). Unveiling the Wireless Network Limitations in Federated Learning, *2021 IEEE International Symposium on Dynamic Spectrum Access Networks (DySPAN)*, pp.262–267.

[6] **Niknam, S.**, **Dhillon, H.S. and Reed, J.H.** (2020). Federated Learning for Wireless Communications: Motivation, Opportunities, and Challenges, *in IEEE Communications Magazine*, *58*(6), 46–51.

[7] **Pu, L.**, **Yuan, X.**, **Xu, X.**, **Chen, X.**, **Zhou, P. and Xu, J.** (2021). Cost-efficient and Skew-aware Data Scheduling for Incremental Learning in 5G Networks, *IEEE Journal on Selected Areas in Communications*.

[8] **Tang, H.**, **Yu, C.**, **Lian, X.**, **Zhang, T. and Liu, J.** (2019). Doublesqueeze: Parallel stochastic gradient descent with double-pass error-compensated compression, *International Conference on Machine Learning*, PMLR, pp.6155–6165.

[9] **Sato, K.**, **Satoh, Y. and Sugimura, D.** (2020). Network-density-controlled decentralized parallel stochastic gradient descent in wireless systems, *ICC 2020-2020 IEEE International Conference on Communications (ICC)*, IEEE, pp.1–6.

[10] **McMahan, H.B.**, **Ramage, D.**, **Talwar, K. and Zhang, L.** (2017). Learning differentially private recurrent language models, *arXiv preprint arXiv:1710.06963*.

[11] Google, “Cloud TPU.”, online, Accessed: 2018-12-04 at: `https://cloud.google.com/tpu/`.

[12] **Park, J.**, **Samarakoon, S. and Bennis, M.a.** & Debbah, mérouane. (2019). Wireless Network Intelligence at the Edge, *Proceedings of the IEEE*, 107. 10.1109/JPROC.2019.2941458.

[13] **al., S.W.** (2019). Adaptive Federated Learning in Resource Constrained Edge Computing Systems, *in IEEE Journal on Selected Areas in Communications*, *37*(6), 1205–1221.

[14] **al., S.W.** (2018). When Edge Meets Learning: Adaptive Control for Resource-Constrained Distributed Machine Learning, ***I. INFOCOM****, editor, IEEE Conference on Computer Communications*, Honolulu, HI, pp.63–71.

[15] **Sato, K.**, **Satoh, Y. and Sugimura, D.** (2020). Network-Density-Controlled Decentralized Parallel Stochastic Gradient Descent in Wireless Systems, ***I. Ieee****, editor, International Conference on Communications (ICC), Dublin, Ireland pp*, 1-6, pp.1–6.

[16] **Abad, M.S.H.**, **Ozfatura, E.**, **GUndUz, D. and Ercetin, O.** (2020). Hierarchical Federated Learning ACROSS Heterogeneous Cellular Networks, ***ICASSP****, editor, International Conference on Acoustics, Speech and Signal Processing (ICASSP), Barcelona, Spain pp*, ICASSP, pp.8866–8870.

[17] **Chen, M.**, **Mathews, R.**, **Ouyang, T. and Beaufays, F.** (2019). Federated learning of out-of-vocabulary words, *arXiv preprint arXiv:1903.10635*.

[18] **Zhu, W.**, **Kairouz, P.**, **McMahan, B.**, **Sun, H. and Li, W.** (2020). Federated heavy hitters discovery with differential privacy, *International Conference on Artificial Intelligence and Statistics*, PMLR, pp.3837–3847.

[19] **Bonawitz, K.**, **Ivanov, V.**, **Kreuter, B.**, **Marcedone, A.**, **McMahan, H.B.**, **Patel, S.**, **Ramage, D.**, **Segal, A. and Seth, K.** (2016). Practical secure aggregation for federated learning on user-held data, *arXiv preprint arXiv:1611.04482*.

[20] **Reddi, S.**, **Charles, Z.**, **Zaheer, M.**, **Garrett, Z.**, **Rush, K.**, **Konečnỳ, J.**, **Kumar, S. and McMahan, H.B.** (2020). Adaptive federated optimization, *arXiv preprint arXiv:2003.00295*.

[21] **Ren, J.**, **Wang, H.**, **Hou, T.**, **Zheng, S. and Tang, C.** (2019). Federated Learning-Based Computation Offloading Optimization in Edge Computing-Supported Internet of Things, *IEEE Access*, *7*, 69194–69201.

[22] **WANG, L.**, **WANG, W. and LI, B.** (2019). CMFL: Mitigating Communication Overhead for Federated Learning, *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp.954–964.

[23] **Zeng, Q.**, **Du, Y.**, **Huang, K. and Leung, K.K.** (2020). Energy-efficient radio resource allocation for federated edge learning, *2020 IEEE International Conference on Communications Workshops (ICC Workshops)*, IEEE, pp.1–6.

[24] **Liu, Y.**, **Yu, J.J.Q.**, **Kang, J.**, **Niyato, D. and Zhang, S.** (2020). Privacy-Preserving Traffic Flow Prediction: A Federated Learning Approach, *IEEE Internet of Things Journal*, *7*(8), 7751–7763.

[25] **Li, T.**, **Sahu, A.K.**, **Zaheer, M.**, **Sanjabi, M.**, **Talwalkar, A. and Smith, V.** (2020). Federated optimization in heterogeneous networks, *Proceedings of Machine Learning and Systems*, *2*, 429–450.

[26] **Avdiukhin, D. and Kasiviswanathan, S.** (2021). Federated Learning under Arbitrary Communication Patterns, ***M. Meila and T. Zhang**, editors, Proceedings of the 38th International Conference on Machine Learning*, volume139 of *Proceedings of Machine Learning Research*, PMLR, pp.425–435, `https://proceedings.mlr.press/v139/avdiukhin21a.html`.

[27] **Kairouz, P.**, **McMahan, H.B.**, **Avent, B.**, **Bellet, A.**, **Bennis, M.**, **Bhagoji, A.N.**, **Bonawitz, K.**, **Charles, Z.**, **Cormode, G.**, **Cummings, R.** *et al.* (2021). Advances and open problems in federated learning, *Foundations and Trends® in Machine Learning*, *14*(1–2), 1–210.

[28] **Xu, C.**, **Qu, Y.**, **Xiang, Y. and Gao, L.** (2021). Asynchronous federated learning on heterogeneous devices: A survey, *arXiv preprint arXiv:2109.04269.*

[29] **G.F.Riley and T.R.Henderson**, (2010). The ns-3 network simulator, modeling and Tools for Network Simulation.

[30] **Yin, H.**, **Liu, P.**, **Liu, K.**, **Cao, L.**, **Zhang, L.**, **Gao, Y. and Hei, X.**, (2020). Ns3-ai: Fostering Artificial Intelligence Algorithms for Networking Research, Proceedings of the 2020 Workshop on ns-3 (WNS3 2020), Association for Computing Machinery, New York, NY, USA, pp.57–64.

[31] **Wang, X.**, **Han, Y.**, **Wang, C.**, **Zhao, Q.**, **Chen, X. and Chen, M.** (2019). In-edge ai: Intelligentizing mobile edge computing, caching and communication by federated learning, *IEEE Network*, *33*(5), 156–165.

[32] **Khan, L.U.** *et al.* (2021). *Federated learning for internet of things: Recent advances, taxonomy, and open challenges*, IEEE Communications Surveys & Tutorials, ieee communications surveys & tutorials edition.

[33] **LeCun, Y.**, **Bottou, L.**, **Bengio, Y. and Haffner, P.** (1998). Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, 86(11).

[34] William Shakespeare. The Complete Works of William Shakespeare, Publically available at: `https://www.gutenberg.org/ebooks/100`.

[35] **Simonyan, K. and Zisserman, A.** (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition, *arXiv 1409.1556*.

[36] **Ye, H.**, **Liang, L. and Li, G.**, (2021). Decentralized Federated Learning with Unreliable Communications. arXiv, preprint, `2108.02397`.

[37] **Jiang, J.C.**, **Kantarci, B.**, **Oktug, S. and Soyata, T.** (2020). Federated Learning in Smart City Sensing: Challenges and Opportunities, *Sensors (Basel, Switzerland)*, *20*.

[38] **O'Dea., S.**, (2019), Number of Smart Phone Users Worldwide|Statista.[Online], Available at: `https://www.statista.com/statistics/330695/number-of-smartphone-users-worldwide`.

[39] **Mezghani, F. and Mitton, N.** (2018). Opportunistic Disaster Recovery, *Internet Technology Letters*, *1*, e29.

[40] **Wang, Y. and Kantarci, B.** (2020). A Novel Reputation-aware Client Selection Scheme for Federated Learning within Mobile Environments, *IEEE 25th Intl. Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD*, *25*, 1–6.

[41] **Wang, Y. and Kantarci, B.** (2021). Reputation-enabled Federated Learning Model Aggregation in Mobile Platforms, *ICC 2021 - IEEE International Conference on Communications*, pp.1–6.

[42] **Wang, Y.**, **Kantarci, B. and Mardini, W.** (2021). Aggregation of Incentivized Learning Models in Mobile Federated Learning Environments, *IEEE Networking Letters*, *10*, 1109.

[43] **Hardy, C.**, **Merrer, E. and Sericola, B.** (2017). Distributed deep learning on edge-devices: Feasibility via adaptive compression, pp.1–8.

[44] **Berri, S.**, **Zhang, J.**, **Bensaou, B. and Labiod, H.** (2019). Joint Data-Prefetching and Broadcast-Scheduling for Hybrid Vehicular Networks, *ICC 2019 - 2019 IEEE International Conference on Communications (ICC)*, pp.1–6.

[45] **Gawłowicz, P. and Zubow, A.** (2019). Ns-3 meets OpenAI Gym: The Playground for Machine Learning in Networking Research, *Proc. of the 22nd Intl ACM Conference on Modeling*, Analysis and Simulation of Wireless and Mobile Systems (MSWIM), pp.113–120.

[46] **Sattler, F.**, **Wiedemann, S.**, **Müller, K.R. and Samek, W.** (2019). Robust and communication-efficient federated learning from non-iid data, *IEEE transactions on neural networks and learning systems*, *31*(9), 3400–3413.

[47] **Mowla, N.I.**, **Tran, N.H.**, **Doh, I. and Chae, K.** (2020). Federated Learning-Based Cognitive Detection of Jamming Attack in Flying Ad-Hoc Network, *in IEEE Access*, *8*, 4338–4350.

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

[49] **Lim, W.Y.B.**, **Luong, N.C.**, **Hoang, D.T.**, **Jiao, Y.**, **Liang, Y.C.**, **Yang, Q.**, **Niyato, D. and Miao, C.** (2020). Federated Learning in Mobile Edge Networks: A Comprehensive Survey, *IEEE Communications Surveys Tutorials*, *22*(3), 2031–2063.

[50] **Bonawitz, K.**, **Eichner, H.**, **Grieskamp, W.**, **Huba, D.**, **Ingerman, A.**, **Ivanov, V.**, **Kiddon, C.**, **Konečnỳ, J.**, **Mazzocchi, S.**, **McMahan, B.** *et al.* (2019). Towards federated learning at scale: System design, *Proceedings of Machine Learning and Systems*, *1*, 374–388.

YÖK

APPENDICES

APPENDIX A.1 : Success percentage and accuracy plots with different interarrival time values.
APPENDIX B.1 : Statistics of various interarrival time values with different seeds

APPENDIX A.1

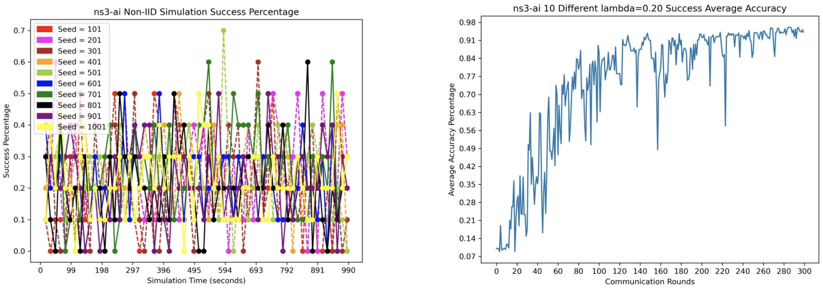


Figure A.1 : Plots for 0.2 interarrival time values.

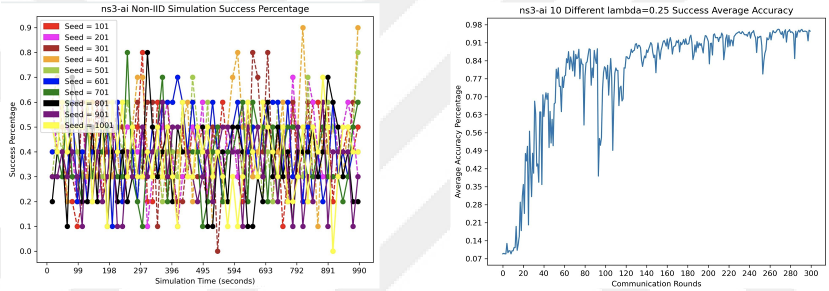


Figure A.2 : Plots for 0.25 interarrival time values.

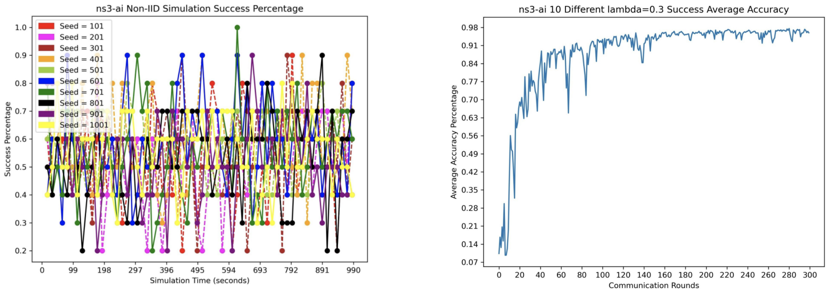


Figure A.3 : Plots for 0.3 interarrival time values.

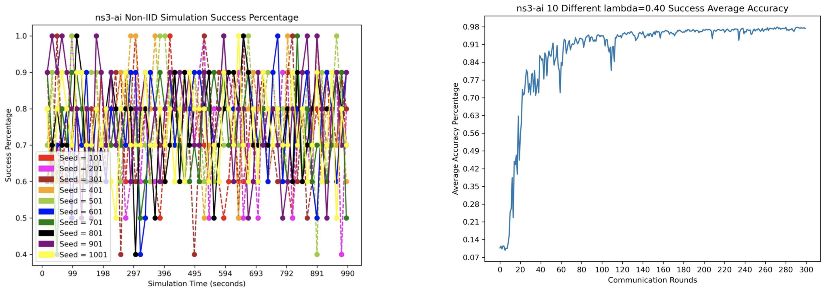


Figure A.4 : Plots for 0.4 interarrival time values.

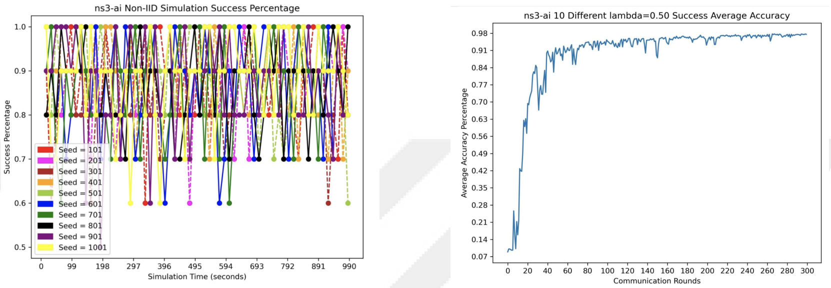


Figure A.5 : Plots for 0.5 interarrival time values.

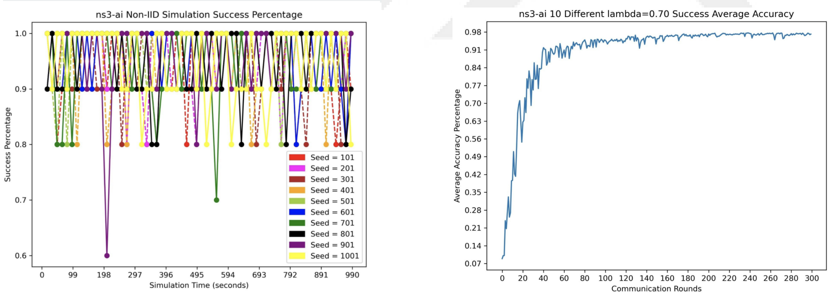


Figure A.6 : Plots for 0.7 interarrival time values.

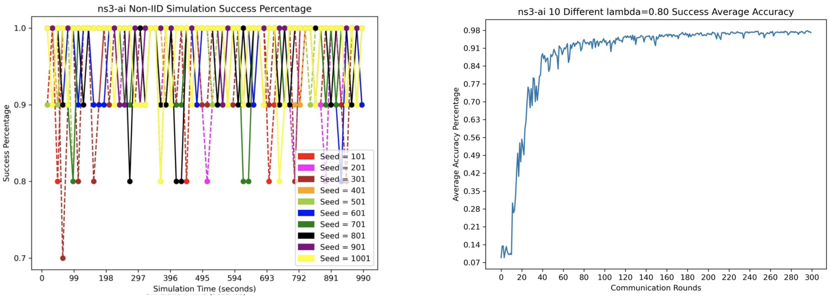
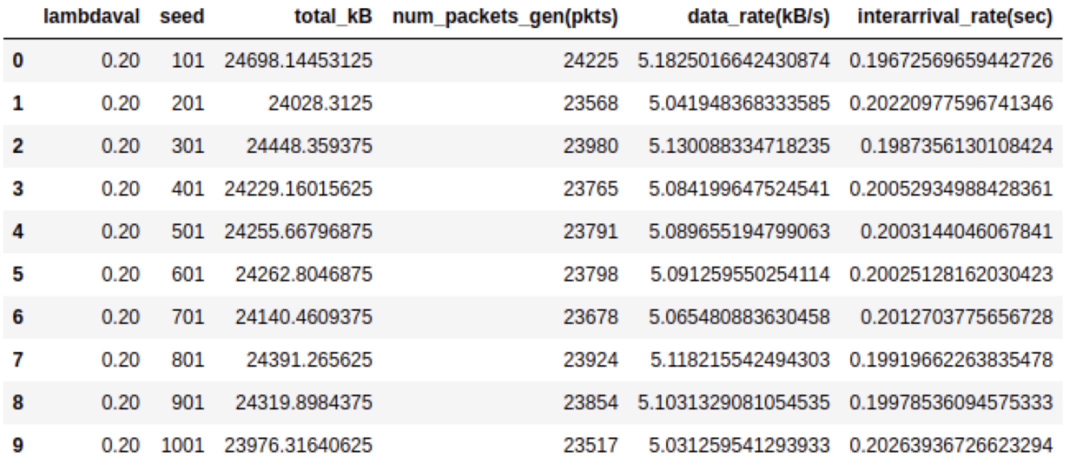


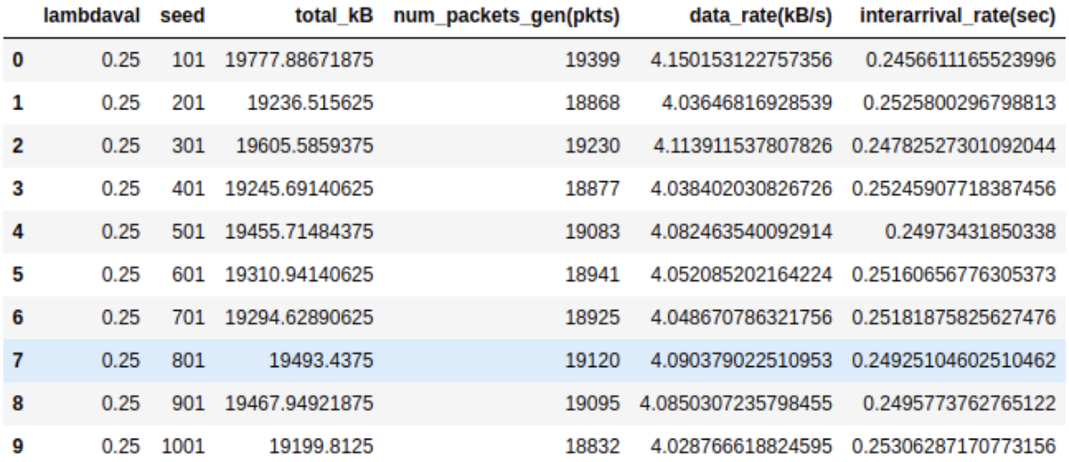
Figure A.7 : Plots for 0.8 interarrival time values.

APPENDIX B.1



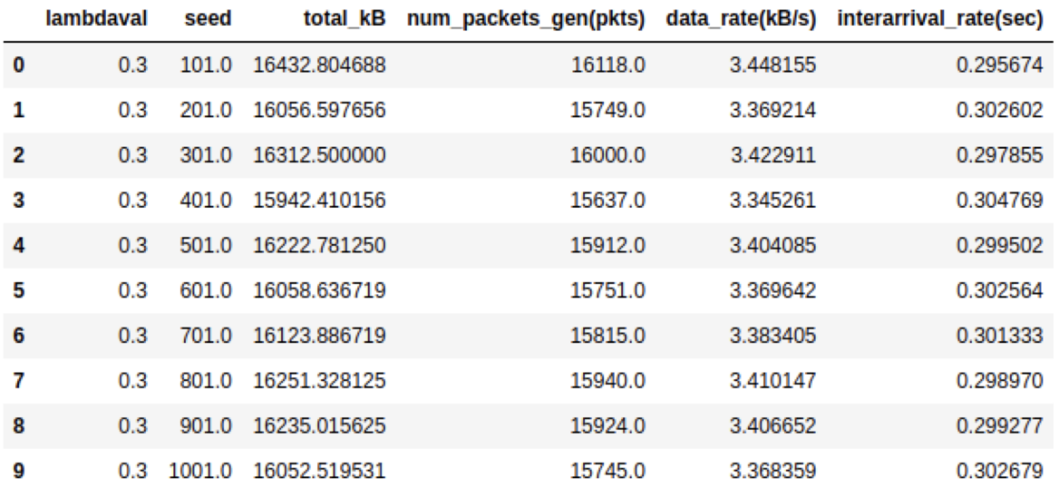
	lambdaval	seed	total_kB	num_packets_gen(pkts)	data_rate(kB/s)	interarrival_rate(sec)
0	0.20	101	24698.14453125	24225	5.1825016642430874	0.19672569659442726
1	0.20	201	24028.3125	23568	5.041948368333585	0.20220977596741346
2	0.20	301	24448.359375	23980	5.130088334718235	0.1987356130108424
3	0.20	401	24229.16015625	23765	5.084199647524541	0.20052934988428361
4	0.20	501	24255.66796875	23791	5.089655194799063	0.2003144046067841
5	0.20	601	24262.8046875	23798	5.091259550254114	0.20025128162030423
6	0.20	701	24140.4609375	23678	5.065480883630458	0.2012703775656728
7	0.20	801	24391.265625	23924	5.118215542494303	0.19919662263835478
8	0.20	901	24319.8984375	23854	5.1031329081054535	0.19978536094575333
9	0.20	1001	23976.31640625	23517	5.031259541293933	0.20263936726623294

Figure B.1 : Statistics of 0.2 interarrival time with different seeds.



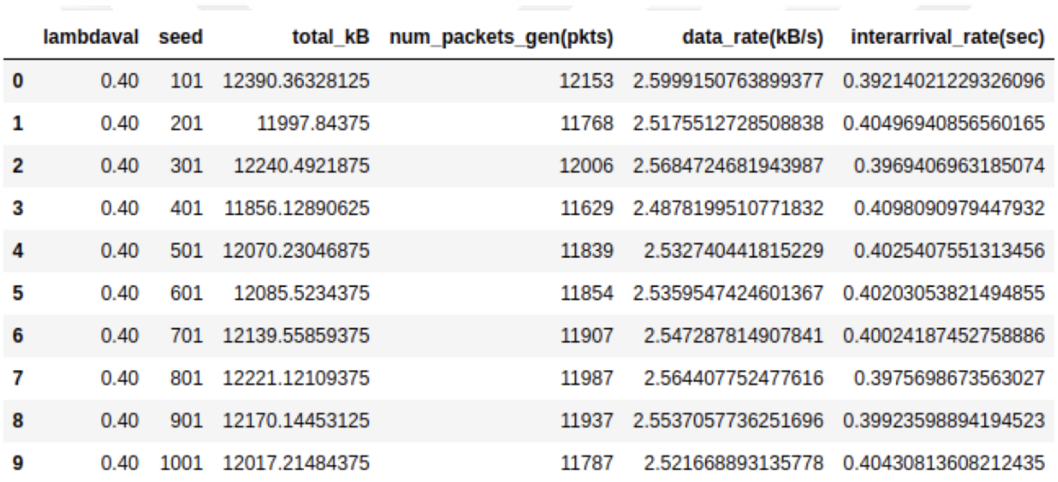
	lambdaval	seed	total_kB	num_packets_gen(pkts)	data_rate(kB/s)	interarrival_rate(sec)
0	0.25	101	19777.88671875	19399	4.150153122757356	0.2456611165523996
1	0.25	201	19236.515625	18868	4.03646816928539	0.2525800296798813
2	0.25	301	19605.5859375	19230	4.113911537807826	0.24782527301092044
3	0.25	401	19245.69140625	18877	4.038402030826726	0.25245907718387456
4	0.25	501	19455.71484375	19083	4.082463540092914	0.24973431850338
5	0.25	601	19310.94140625	18941	4.052085202164224	0.25160656776305373
6	0.25	701	19294.62890625	18925	4.048670786321756	0.25181875825627476
7	0.25	801	19493.4375	19120	4.090379022510953	0.24925104602510462
8	0.25	901	19467.94921875	19095	4.0850307235798455	0.2495773762765122
9	0.25	1001	19199.8125	18832	4.028766618824595	0.25306287170773156

Figure B.2 : Statistics of 0.25 interarrival time with different seeds.



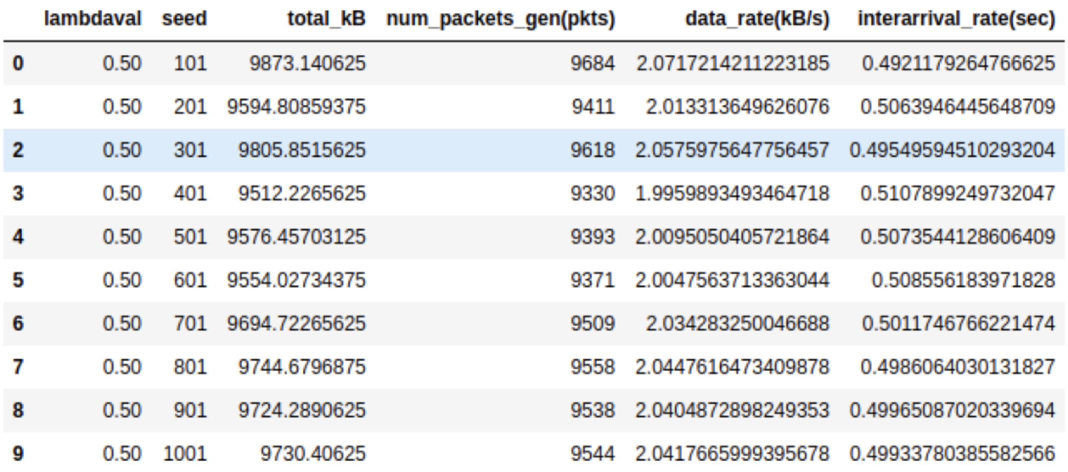
	lambdaval	seed	total_kB	num_packets_gen(pkts)	data_rate(kB/s)	interarrival_rate(sec)
0	0.3	101.0	16432.804688	16118.0	3.448155	0.295674
1	0.3	201.0	16056.597656	15749.0	3.369214	0.302602
2	0.3	301.0	16312.500000	16000.0	3.422911	0.297855
3	0.3	401.0	15942.410156	15637.0	3.345261	0.304769
4	0.3	501.0	16222.781250	15912.0	3.404085	0.299502
5	0.3	601.0	16058.636719	15751.0	3.369642	0.302564
6	0.3	701.0	16123.886719	15815.0	3.383405	0.301333
7	0.3	801.0	16251.328125	15940.0	3.410147	0.298970
8	0.3	901.0	16235.015625	15924.0	3.406652	0.299277
9	0.3	1001.0	16052.519531	15745.0	3.368359	0.302679

Figure B.3 : Statistics of 0.3 interarrival time with different seeds.



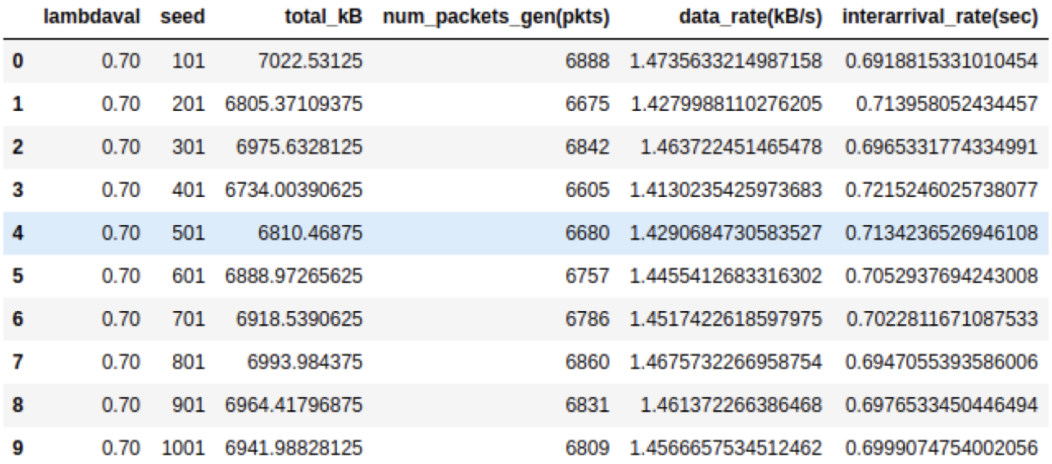
	lambdaval	seed	total_kB	num_packets_gen(pkts)	data_rate(kB/s)	interarrival_rate(sec)
0	0.40	101	12390.36328125	12153	2.5999150763899377	0.39214021229326096
1	0.40	201	11997.84375	11768	2.5175512728508838	0.40496940856560165
2	0.40	301	12240.4921875	12006	2.5684724681943987	0.3969406963185074
3	0.40	401	11856.12890625	11629	2.4878199510771832	0.4098090979447932
4	0.40	501	12070.23046875	11839	2.532740441815229	0.4025407551313456
5	0.40	601	12085.5234375	11854	2.5359547424601367	0.40203053821494855
6	0.40	701	12139.55859375	11907	2.547287814907841	0.40024187452758886
7	0.40	801	12221.12109375	11987	2.564407752477616	0.3975698673563027
8	0.40	901	12170.14453125	11937	2.5537057736251696	0.39923598894194523
9	0.40	1001	12017.21484375	11787	2.521668893135778	0.40430813608212435

Figure B.4 : Statistics of 0.4 interarrival time with different seeds.



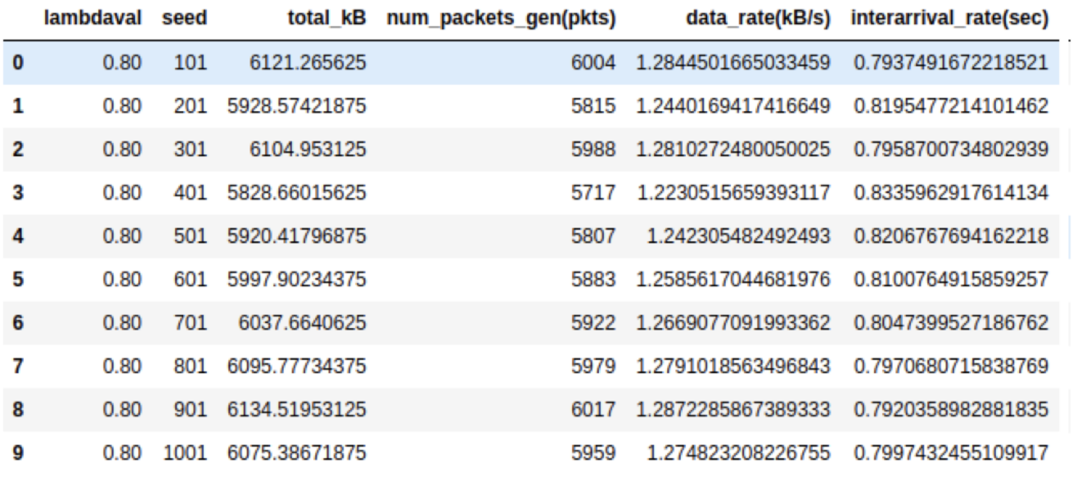
	lambdaval	seed	total_kB	num_packets_gen(pkts)	data_rate(kB/s)	interarrival_rate(sec)
0	0.50	101	9873.140625	9684	2.0717214211223185	0.492117926476625
1	0.50	201	9594.80859375	9411	2.013313649626076	0.5063946445648709
2	0.50	301	9805.8515625	9618	2.0575975647756457	0.49549594510293204
3	0.50	401	9512.2265625	9330	1.9959893493464718	0.5107899249732047
4	0.50	501	9576.45703125	9393	2.0095050405721864	0.5073544128606409
5	0.50	601	9554.02734375	9371	2.0047563713363044	0.508556183971828
6	0.50	701	9694.72265625	9509	2.034283250046688	0.5011746766221474
7	0.50	801	9744.6796875	9558	2.0447616473409878	0.4986064030131827
8	0.50	901	9724.2890625	9538	2.0404872898249353	0.49965087020339694
9	0.50	1001	9730.40625	9544	2.0417665999395678	0.49933780385582566

Figure B.5 : Statistics of 0.5 interarrival time with different seeds.



	lambdaval	seed	total_kB	num_packets_gen(pkts)	data_rate(kB/s)	interarrival_rate(sec)
0	0.70	101	7022.53125	6888	1.4735633214987158	0.6918815331010454
1	0.70	201	6805.37109375	6675	1.4279988110276205	0.713958052434457
2	0.70	301	6975.6328125	6842	1.463722451465478	0.6965331774334991
3	0.70	401	6734.00390625	6605	1.4130235425973683	0.7215246025738077
4	0.70	501	6810.46875	6680	1.4290684730583527	0.7134236526946108
5	0.70	601	6888.97265625	6757	1.4455412683316302	0.7052937694243008
6	0.70	701	6918.5390625	6786	1.4517422618597975	0.7022811671087533
7	0.70	801	6993.984375	6860	1.4675732266958754	0.6947055393586006
8	0.70	901	6964.41796875	6831	1.461372266386468	0.6976533450446494
9	0.70	1001	6941.98828125	6809	1.4566657534512462	0.6999074754002056

Figure B.6 : Statistics of 0.7 interarrival time with different seeds.



	lambdaval	seed	total_kB	num_packets_gen(pkts)	data_rate(kB/s)	interarrival_rate(sec)
0	0.80	101	6121.265625	6004	1.2844501665033459	0.7937491672218521
1	0.80	201	5928.57421875	5815	1.2440169417416649	0.8195477214101462
2	0.80	301	6104.953125	5988	1.2810272480050025	0.7958700734802939
3	0.80	401	5828.66015625	5717	1.2230515659393117	0.8335962917614134
4	0.80	501	5920.41796875	5807	1.242305482492493	0.8206767694162218
5	0.80	601	5997.90234375	5883	1.2585617044681976	0.8100764915859257
6	0.80	701	6037.6640625	5922	1.2669077091993362	0.8047399527186762
7	0.80	801	6095.77734375	5979	1.2791018563496843	0.7970680715838769
8	0.80	901	6134.51953125	6017	1.2872285867389333	0.7920358982881835
9	0.80	1001	6075.38671875	5959	1.274823208226755	0.7997432455109917

Figure B.7 : Statistics of 0.8 interarrival time with different seeds.

CURRICULUM VITAE

Name Surname : Mümtaz Cem Eriş

[redacted]

EDUCATION

- **B.Sc.** : 2018, Istanbul Technical University, Computer and Informatics Faculty, Computer Engineering Department

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2021-Present Software Developer at Maxitech
- 2017-2021 Software Developer at Softtech
- 2019-Present National Athlete (Top 6) at Turkish Dance Sport Federation (TDSF)

PUBLICATIONS:

- **Eriş M., Kantarci B., Oktug S.** 2021. Unveiling the Wireless Network Limitations in Federated Learning. *IEEE International Symposium on Dynamic Spectrum Access Networks (IEEE DySPAN 2021)*, December 13–15, 2021.

M. C. ERİŞ

UNVEILING THE WIRELESS NETWORK LIMITATIONS IN FEDERATED LEARNING

2021