A Novel Recurrent Neural Network based Resource Request Scheme Minimizing RACH Preamble Collisions in 3GPP LTE-A Networks

K R Manoj kumar
Computer science and engineering
IIT Dharwad
Dharwad, India
manojkoneni311@gmail.com

Siba Narayan swain
Computer science and engineering
IIT dharwad
Dharwad, India
sibaswain@iitdh.ac.in

Abstract

When the number of User Equipments(UEs) requesting resources from the eNBs are higher than the number of preambles(64) used by the RACH procedure for resource allocation in LTE, collisions occur among UE requests. After collisions the base stations(eNBs) cannot send a response to the collided RACH preambles so, the UEs wait for a limited amount of time and restart the RACH procedure, without any guarantee that the new request will have a unique preamble at eNb without any collision. The use of limited preambles thus cause a bottleneck on the performance of LTE. In this paper, we use some machine learning techniques to design a procedure that reduced the amount of collisions among RACH preambles and thereby increase the performance of overall system. We used RNN to predict the resource allocation among UEs. Replacing RNN with RACH doesn't give a satisfactory collision reduction whereas, when RNN is used along with existing RACH procedure the collision rate came down to less than 5 collisions from 30 collisions in a RACH operating time interval.

Keywords— LTE-A Networks, Cellular Networks, RACH procedure, Preambles, Machine Learning, Recurrent Neural Networks

I. Introduction

The Random Access Channel (RACH) procedure in Long Term Evolution (LTE) Networks serves as a synchronization mechanism between the Base Station (eNB) and the User Equipments (UEs), where the UEs compete for frequency resources by randomly choosing a preamble and getting assigned frequency resources for sending data by the eNB, provided that they do not collide with each other in the preamble-based contention phase.The UEs use the newly assigned data channel resources to communicate their connection request.
The current system was designed to handle about 128 attempts per second [1], however forecasts show that traffic could reach upwards of 370 attempts per second in the near future [2], mainly due to the expected increase of machine type traffic in the cellular access. In case of synchronized alarms, simultaneous accesses to the channel may yield up to a tenfold increase in collisions with respect to normal traffic, almost guaranteeing that all UEs will collide when accessing the data channel [3], [4].Whenever a collision occurs i.e., two or more UEs send the same RACH preambles to eNB. In this case, both the UEs will receive the same resource allocation at RAR step(Random Access Response step). And as a result, both UE would send L2/L3 message through the same resource allocation(i.e., with the same time/frequency location) to NW(Network) at L2/L3 message transmission step. when both UE transmit the exact same information on the exact same time/frequency location, there are two possibilities. One possibility is that these two signals act as interference to each other and NW decodes neither of them. In this case, none of the UEs would have any response (HARQ ACK-Hybrid Automatic Repeat Request ACK) from eNB and they all think that RACH process has failed and go back to Random Access Preamble generation step. The other possibility would be that eNB could successfully decode the message from only one UE and fail to decode it from the other UE. In this case, the UE with the successful L2/L3 decoding on NW side will get the HARQ ACK from Network.This HARQ ACK process for step L2/L3 message is called ”contention resolution” process [5].

Due to the significant increase in machine traffic, RACH overload presents a serious bottleneck on the performance of the network. State of the art 3GPP(Third Generation Partnership Project) accepted solutions to this problem include Access call barring methods, MTC specific backoff, Resource separation, Dynamic RACH allocation methods etc [6]. Access class barring (ACB) is a well-known tool to control RA(Random Access) congestion by reducing the access arrival rate. ACB operates on two factors: a set of barring access classes (ACs) in which devices are classified, and a barring time duration (T_b). Depending on the RA congestion level, the eNB broadcasts an access probability p, and barring time duration T_b as a part of SIB2(System Information Block Type 2). The intended UEs generate their own access probability q accordingly to the AC to which they belong. If q $\leq$ p, the UE gets permission to access the network; otherwise, it is barred for an ACB window T_b [7]. The other algorithm is MTC specific backoff mechanism to control RA in cellular networks. The basic idea behind backoff scheme is that it discourages the UEs to seek the access opportunity for a time duration, called Backoff Interval (BI), if their first attempt failed due to collision or channel fading. If a device fails second time to get access, it will be subjected to a larger BI than

the previous one [6]. The next algorithm is Dynamic resource allocation, Dynamic allocation of RACH is a straightforward solution for the RA congestion problem. Under this scheme, the eNB can increase the RACH resources based on the RA congestion level [6]. Among these Access barring methods have a high success rate in collision reduction followed by dynamic resource allocation and then MTC specific backoff [6].Different from all the previous approaches on reducing collisions in RACH, our approach tries to first determine the resource requests in a given interval of time, and use that information to directly send RARs to the concerned users thereby avoiding the need to use preambles for getting the channel information to send the resource request. Our algorithm is a combination of a prediction algorithm, which tries to exploit the temporal structure in the resource requests of the past data and tries to predict the future requests, and RACH procedure for the resource requests that were not predicted by the prediction algorithm. This combination of two algorithms, reduces the dependence of resource requests on limited preambles and there by decreasing the collision rate.
The rest of the paper is organized as follows. Section 2 describes the LTE RACH procedure. Section 3 describes the Machine learning algorithm that we implemented for resource request predictions. section 4 and 5 describe the dataset and the results obtained from using the Machine learning algorithm along with RACH procedure for resource allocation in LTE Networks.

II. Prerequisites:

In this section we explain the required prerequisites to understand LTE-RACH procedure and Recurrent neural networks.

A. LTE RACH:

RACH stands for Random Access Channel. This is the first message from UE to eNB when you power it on.The main purpose of RACH can be described as follows.

1) Achieve UP link synchronization between UE and eNB
2) Obtain the resource for Message 3 (e.g, RRC Connection Request)

When a UE transmit a PRACH Preamble, it transmits with a specific pattern and this specific pattern is called a ”Signature”. In each LTE cell, total 64 preamble signatures are available and UE select randomly one of these signatures.The random access procedure in LTE is performed in the Physical Random Access Channel (PRACH), a dedicated physical channel with an overall bandwidth of 1.08 MHz and duration between 1 and 4 LTE sub frames.

B. RACH Procedure

The RACH procedure consists of four phases: [8]

1) Random Access Preamble. The UEs that intend to transmit data randomly choose one among N_{prb} available preambles and send it to the eNB.
2) Random Access Response (RAR). The eNB processes the received signal, consisting in the superposition of all the transmitted preambles, and detects which preambles were chosen. For each detected preamble, it then sends a RAR message to assign the uplink resources to the corresponding UE.
3) L2/L3 message. The UEs use the newly assigned data channel resources to communicate their connection request, and a unique identifier.
4) Contention Resolution Message. The eNB responds to the UEs using the identifiers they communicated in their L2/L3 message, granting the requested resources.

UEs can access the channel either in a contention-free mode, where the eNB forces the UEs to use a particular signature for the preamble generation, thus avoiding collisions, or in a contention-based mode. While contention-free access is reserved for handover and other special delay-sensitive cases, contention-based access represents the default method for UEs to access the channel, and is thus taken as the focus of this paper. In this procedure, which is illustrated in Figure 1, if multiple UEs pick the same preamble, a collision happens, and may be resolved at different stages of the procedure. If the colliding preambles are received with high enough Signal-to- Noise Ratio (SNR), and are sufficiently spaced apart in time, the eNB can detect both of them and recognize the collision, avoiding to send the RAR message to the involved UEs. Otherwise, the eNB will not detect the collision but a single preamble, and all colliding UEs will receive a RAR message and try to access the same resource simultaneously, colliding in the L2/L3 message phase. The eNB will therefore be unable to decode the received message, and will send no contention resolution message; the collided UEs can then try again in a new RACH phase. Collisions go undetected especially in urban environments, where smaller cells are employed and the distance between different UEs is too small to allow a differentiation of multiple copies of the same preamble through the delay with which they arrive at the eNB [4].

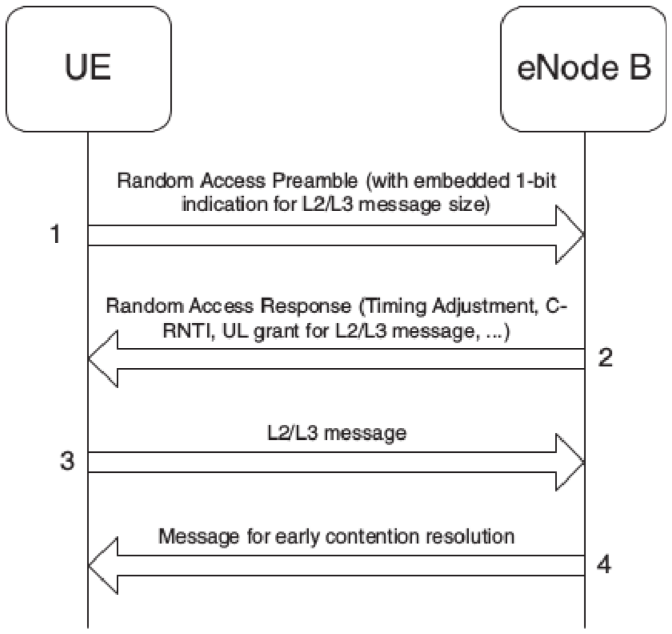

UE
eNode B
Random Access Preamble (with embedded 1-bit indication for L2/L3 message size)
1
Random Access Response (Timing Adjustment, C-RNTI, UL grant for L2/L3 message, ...)
2
L2/L3 message
3
Message for early contention resolution
4

Fig. 1. contention based RACH procedure

C. Preamble Collisions

The number of preambles that an UE can select is limited to 64. When an UE is operating in contention based mode, then there is a non-zero probability that two different UEs can select the same preamble signature. If the number of UEs requesting resources under a eNB increase the probability of RACH preamble collision increases. **It is this collision in preamble signatures that we aim to reduce thereby allowing more number of UEs resource requests to be serviced by the eNB by predicting the resource requests. The change in collision rate before and after using this algorithm is used as a measure to evaluate the performance of the algorithm.**

D. RNN:

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed graph along a sequence [9]. This directed graph like form in the network allows it to exhibit dynamic temporal behavior for a time sequence. Unlike feedforward neural networks, RNNs can use their internal state (memory) to process sequences of inputs. This makes them applicable to tasks such as unsegmented, connected handwriting recognition or speech recognition. Recurrent Neural Network comes into the picture when any model needs context to be able to provide the output based on the input. This is especially true when there is a dependency of data on time or time-series data. RNN's try to learn these temporal dependencies through feedback loops.

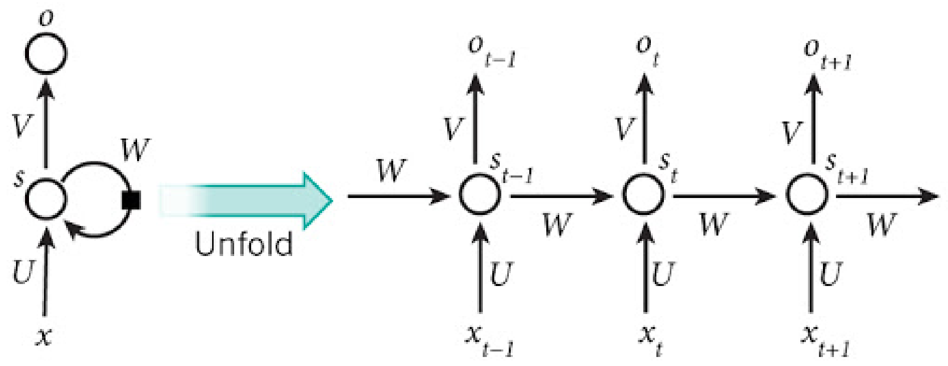
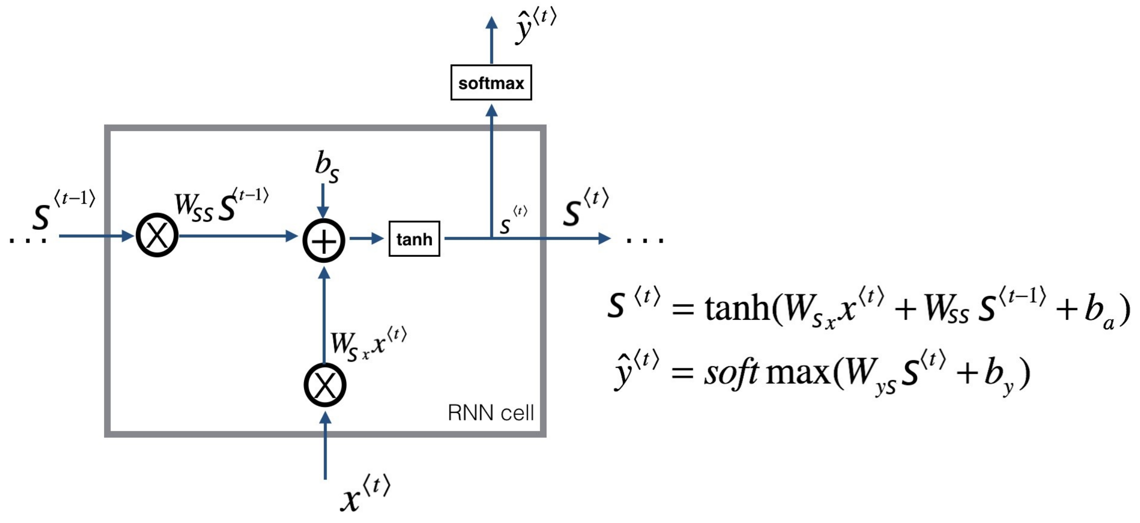


Fig. 2. RNN unfolded

The above image 2 is an example of rnn with feedback loop unfolded. The feedback loops can be of any length depending on how many previous inputs the output depends on, and all the feedback loops obviously share the same weights. Taking a closer look at one of the unfolded rnn cells:



$\hat{y}^{\langle t\rangle}$
softmax
b_S
$S^{\langle t-1\rangle}$
$W_{SS}S^{\langle t-1\rangle}$
tanh
$S^{\langle t\rangle}$
$S^{\langle t\rangle}$
$W_{Sx}x^{\langle t\rangle}$
RNN cell
$x^{\langle t\rangle}$
$S^{\langle t\rangle} = \tanh(W_{Sx}x^{\langle t\rangle} + W_{SS}S^{\langle t-1\rangle} + b_a)$
$\hat{y}^{\langle t\rangle} = soft\max(W_{yS}S^{\langle t\rangle} + b_y)$

Fig. 3. RNN cell

Here in Figure 3, $S^{<t>}$ is called hidden state, $x^{<t>}$ is input at time-step t, $y^{<t>}$ is the predicted output at time t. For each computation of $S^{<t>}$ there is a term $S^{<t-1>}$, which includes some information about past inputs. This is combined with current input $x^{<t>}$ and bias to get the hidden state for next time-step.

III. Neural Network based Resource predictor:

We used Recurrent neural networks as a prediction model for predicting the resource requests. RNN predicts the user along with duration, time stamp of resource request. The input to the RNN has 380X1 dimensions and the output also has the same dimensions, the input layer is connected to a hidden layer which has 128 neurons. As mentioned in the RNN section, the hidden layer has a feedback loop, the number of times this loop forwards the input is a hyper-parameter and it is chosen to be 10 times. This translates to, giving RNN 10 consecutive inputs and try to predict the eleventh resource request. The input has user id part, duration if resource request and timestamp at which the resource is requested. Only one hidden layer is used. Similarly, output also has these features. The architecture of the RNN used look like:

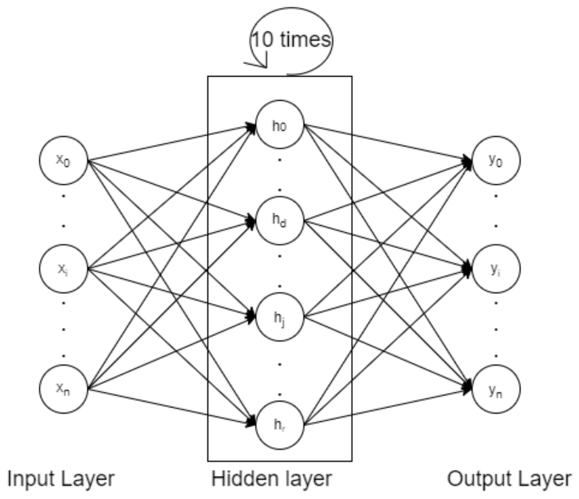


Fig. 4. RNN architecture

Network Parameter	value
Input neurons	380
Output neurons	380
Hidden layers	1
Hidden size	128
Hidden layer Activation function	tanh
Output layer Activation function	softmax
Training samples	72000
Validation samples	9000
Batch size	20
Training Epochs	100
Training Loss	0.0551
Training accuracy	79.21%
Validation Loss	0.0503
Validation Accuracy	88.03%

TABLE I
RNN PARAMETERS

Here, n=380, r = 128(size of the hidden layer). The hidden layer is repeated 10 times, for each time step a different input is given. Intuitively for 1 forward pass through the RNN involves, providing RNN with 1st resource request then predicting 2nd resource request, providing 2nd resource request and predicting the 3rd resource request and so on till providing 10th request and predicting the 11th request. So, one row of RNN input has 10 resource requests stacked and the output also has 10 predicting resource requests, we only take the last predicted resource request as the output and store it i.e., input has [1,2,3,...., 10] resource requests output has [2,3,4,...,11] predicted resource requests and we take 11th predicted request as the output of RNN discarding others.
The non-linear activation function used is hyperbolic tangent function(tanh(x) = $\frac{e^x - e^{-x}}{e^x + e^{-x}}$). All the weights and biases are initialized with Xavier weight initialization(W,b $\sim$ Uniform($-\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}, +\frac{\sqrt{6}}{\sqrt{n_i + n_{i+1}}}$)). For the output layer, softmax function(softmax(x_i) = $\frac{e^{-x_i}}{\sum_{i=1}^{n} e^{-x_i}}$) is used on the user part of the output to convert the values into probabilities. Since the RNN output has both categorical output(user part) and continuous data output(duration and time stamp parts), we use two different loss functions on these. For the user part we use Cross-entropy loss function between target value and predicted value, for the continuous data part we use Mean square loss on target value and predicted values. To adjust the weights from error we used Adam optimizer.

A. Integrating RNN with RACH:

RNN is used only to limit the use of RACH procedure, not completely replace it. So, the main algorithm is a RNN algorithm integrated with RACH procedure. The pseudo-code for the integration looks like this:

```
Data: All resource requests till the previous time interval.
while True do
    i=1;
    while $i \leq K$ do
        prediction = RNN(data[-10:]);
        data.append(prediction);
        i++;
    end
    Resource_requests = data[-K:];
    remaining_UEs = set(All_UEs) - set(Resource_requests);
    RACH(remaining_UEs);
end
```

Algorithm 1: RACH+RNN algorithm

For each time interval, RNN algorithm predicts the next K resource requests, these requests are directly assigned data channels. Resource requests that are not satisfied/ predicted by the RNN are satisfied through the RACH procedure. If the prediction accuracy of RNN is high the dependency on RACH procedure is reduced there by reducing the collision rate occurring due to the finite number of preambles.

IV. Performance Evaluation:

A. Dataset:

The dataset we used for our experiments is a calls dataset [10] containing 27 users and their calls information including duration of the call, time at which the call started. Each row in the dataset has 3 columns i.e., ”user”, ”duration”, ”timestamp” column.

user	duration	timestamp
7981267897	0	Thu Sep 09 20:57:55 +0100 2010
7981267897	605	Fri Sep 10 20:17:00 +0100 2010
7981267897	1	Sat Sep 11 16:44:56 +0100 2010
7981267897	59	Sat Sep 11 17:00:59 +0100 2010
7981267897	1201	Mon Sep 13 14:53:26 +0100 2010

TABLE II
SAMPLE DATA

The amount of data was very less, so we synthetically generated more users from these 27 users. In the synthetic user generation, we assumed that ”duration” and ”timestamp” for each user comes from a bi-variate Gaussian distribution. From these, new mean and variance for new users are generated. Let the mean, variance and number of calls of one user are μ_1, Σ_1, n_1 and that of other user are μ_2, Σ_2, n_2 belonging to Gaussian distributions $G(\mu_1, \Sigma_1)$ and $G(\mu_2, \Sigma_2)$. The new users mean, variance and number of calls are calculated using:

$$\mu = \frac{n_1\mu_1 + n_2\mu_2}{n_1 + n_2}$$

$$\Sigma = \frac{n_1^2\Sigma_1 + n_2^2\Sigma_2 + (n_1\mu_1 + n_2\mu_2)(n_1\mu_1 + n_2\mu_2)^T}{(n_1 + n_2)^2} - \mu\mu^T$$

$$n = \frac{n_1 + n_2}{2}$$

Using this method of user generation we generated 351 new users by selecting different pair of users each time and generated a total of 101942 new calls in the dataset by sampling n values from Gaussian distribution $G(\mu, \Sigma)$ with the new mean and variance values.

B. Data Preprocessing:

This section talks about preprocessing which refers to adding missing values, creating new features from existing features, removing redundant features, standardizing or normalizing data etc.
In the first step, There were some rows of data(8 rows) with year 1980 in the timestamp in the original dataset. These rows were removed. In the second step, the ”user” column is converted into one-hot encoding vector of 378 dimension(because it has 378 users i.e., 351 new+ 27 old users).In the third step,”duration” column is normalized by subtracting minimum value and dividing by the columns range(maximum value-minimum value). The ”timestamp” column had 4 months of data in string format, the entire column is converted into seconds and then normalized using the same approach done for ”duration” column. Now the data has 380 columns(378 ”users” + ”duration” + ”timestamp”). Normalization of data for neural network training can improve accuracy and reduce the time required to train the model [11].

C. Training the RNN:

The dataset is sorted by the time-stamp values and first 72000 rows are used for training next 9000 rows for validation and the last 20942 rows for testing data. For RNN training and testing batch-size of 20 is used. The weights are updated with a learning rate of 1e-3. The RNN model is trained for a total of 100 epochs and at the end of each epoch average accuracy in predicting the correct user and the error in predicting user, duration and timestamp are reported. The best model had an training accuracy of 80% and validation accuracy of about 88% and similar test accuracy($\sim$ 83%) with train error 0.0745 and validation error 0.0503.

D. LTE Network setup:

To simulate the RACH procedure, we have taken 378 UEs, 1 eNB and the number of preambles in RACH procedure is fixed at 64. The RACH procedure is simulated through a uniform distribution of resource requests in an interval (Table III).

In this RACH procedure simulation we can show the dependence of collision on the number of preambles. The figure 5 is plotted assuming each time interval has 100 resource requests at an average and the number of preambles for RACH are varied from 10 to 100. From the figure 5 we can observe a inverse relationship between number of preambles and number of collisions.

Network parameter	value
number of UEs	378
number of eNBs	1
number of preambles	64

TABLE III
LTE NETWORK SIMULATION PARAMETERS

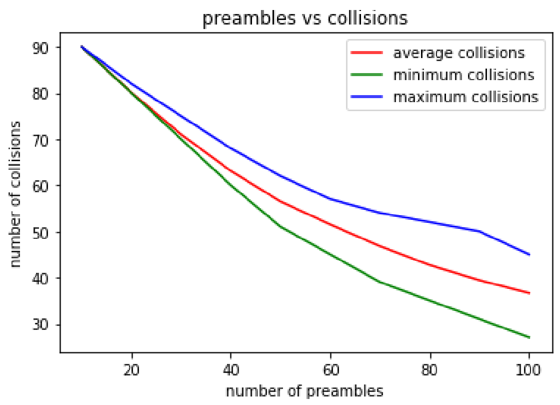


Fig. 5. Dependence of collision rate on number of preambles

E. Results:

The ”timestamp” column in the dataset is spread over 4 months, when this column is converted into seconds the values were huge. Although after normalization, there were in (0,1] range. After training, the timestamp predictions were generated with a little error, when the predictions are scaled back, the difference between the original timestamp and predicted timestamp was very large. Also, RNN can predict only a fixed number of UEs in a time interval. So, instead of running RNN, RACH procedure at a sub-frame level, We calculate the number of collisions if there are K number of requests in a time interval on an average.

To compare the performance of RACH procedure and the RNN based resource prediction, we divided entire test dataset into segments of K resource requests, which can be interpreted as finding out the collision rate if there are K resource requests on an average in each time interval. For K=50,100,120 the results look like these:

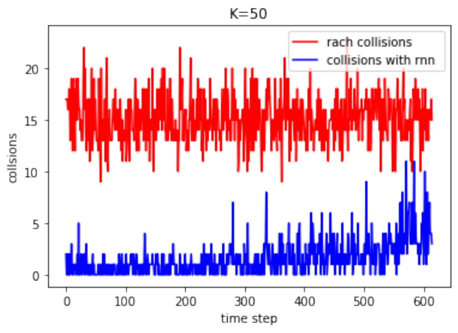


Fig. 6. RNN vs RACH collisions(for K=50 UEs)

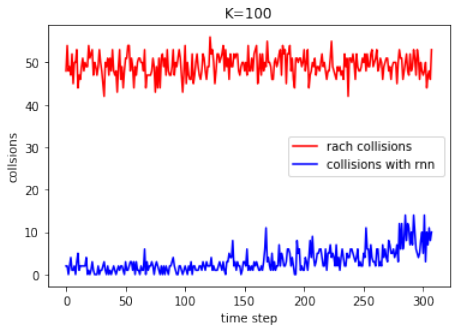


Fig. 7. RNN vs RACH collisions(for K=100 UEs)

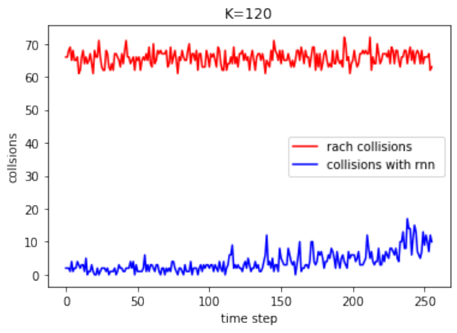


Fig. 8. RNN vs RACH collisions(for K=120 UEs)

From the Figures 6, 7 and 8 we can conclude that collision in RACH procedure are always higher that the collisions in RNN resource prediction procedure.The minimum, maximum and average collisions are tabulated in IV.

And also as the number of average UEs per time interval increased the average number of collisions of RACH increased rapidly from 15.4 to 49.3 and then to 65.7 collisions, but the collisions from RNN did not increase as high as the RACH procedures. From the last two graphs we notice increase in collisions from RNN at the end this is due to the change in user behaviour over time and these type of collisions can be reduced by retraining the RNN with recent data whenever the number of collision cross a user defined threshold.

K	RACH average collisions	RNN Average collisions	RACH min. collisions	RNN min. collisions	RACH max. collisions	RNN max. collisions
50	15.14	1.59	9	0	17	23
100	49.3	3.21	42	0	56	14
120	65.7	3.83	61	0	72	17

TABLE IV
COLLISION RATE COMPARISON

The results when RNN is used along with RACH procedure for the collisions from RNN, the results look better, Table V :

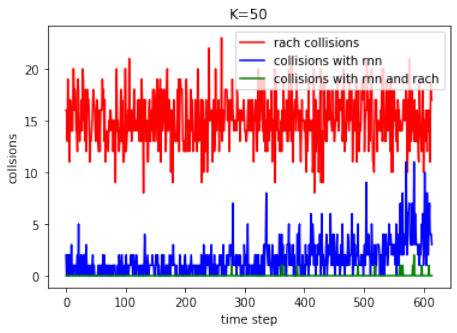


Fig. 9. collisions vs time steps(for K=50 UEs)

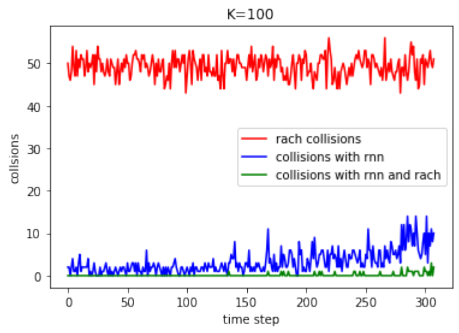


Fig. 10. collisions vs time steps(for K=100 UEs)

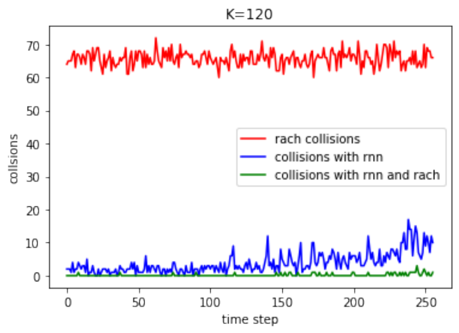


Fig. 11. collisions vs time steps(for K=120 UEs)

K	RACH + RNN average collisions	RACH + RNN min. collisions	RACH + RNN max. collisions
50	0.0278	0	1
100	0.088	0	3
120	0.14	0	4

TABLE V
COLLISION RATE

The maximum number of collisions in a time interval were very less(less than 5) when RNN resource prediction is combined with RACH procedure i.e., RACH preamble procedure is used only for the users that were incorrectly predicted from RNN resource prediction procedure.

F. RNN vs MLP:

When a Multi-layer perceptron(MLP) is used to predict the resource requests instead of a RNN. From the table VI it is evident that most of the parameters for RNN algorithm and MLP algorithm are same. The performance was very poor the train error and validation error were not changing(decreasing) and the prediction accuracy of MLP was close to zero as it is evident from Figure 12.
This is due to the MLP not being able to capture the temporal structure in a time-series data and due to the high-dimensional and sparse user part of the dataset. We tried changing the learning rate and the dimensionality of the hidden layer in MLP but none of them seemed to decrease error or improve prediction accuracy.

V. CONCLUSION:

With RACH overload becoming an increasingly serious issue in massive access LTE scenarios, it is necessary to develop techniques to better manage the contention of resources by multiple users. In this work we have shown that using RNNs along with RACH procedure can improve the resource allocation procedure by reducing the number of collisions.
Although the 83% test accuracy is not always desirable, RNN request prediction along with RACH procedure produces very good results with minimum collision rate and reducing the bottleneck created by having limited number of preambles in RACH procedure, which can further decrease the wait time in UEs due to collisions.

REFERENCES

[1] S. Sesia, I. Toufik, and M. Baker, *LTE, The UMTS Long Term Evolution: From Theory to Practice*. Wiley Publishing, 2009.
[2] G. C. Madueño, Stefanović, and P. Popovski, “Reengineering gsm/gprs towards a dedicated network for massive smart metering,” in *2014 IEEE International Conference on Smart Grid Communications (SmartGridComm)*, pp. 338–343, 2014.

Network parameter	value
Input neurons	380
Output neurons	380
Hidden layers	1
Hidden size	128
activation function	tanh
Training samples	72000
Validation samples	9000
Batch size	20
Training epochs	100
Training Loss	5.895
Training accuracy	0
Validation Loss	5.895
Validation accuracy	0

TABLE VI
MLP PARAMETERS

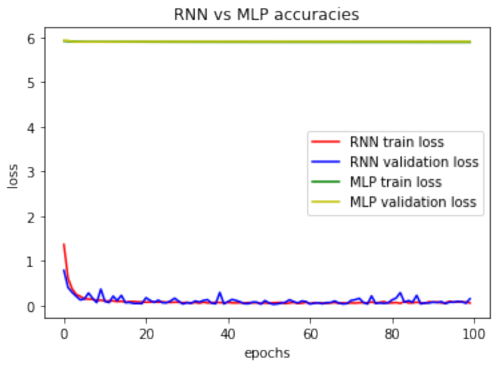


Fig. 12. MLP vs RNN training

[3] M. Cheng, G. Lin, H. Wei, and A. C. Hsu, "Overload control for machine-type-communications in lte-advanced system," *IEEE Communications Magazine*, vol. 50, no. 6, pp. 38–45, 2012.
[4] A. Laya, L. Alonso, and J. Alonso-Zarate, "Is the random access channel of lte and lte-a suitable for m2m communications? a survey of alternatives," *IEEE Communications Surveys Tutorials*, vol. 16, no. 1, pp. 4–16, 2014.
[5] "contention resolution process." www.sharetechnote.com/html/RACH_LTE.html.
[6] M. S. Ali, E. Hossain, and D. I. Kim, "Lte/lte-a random access for massive machine-type communications in smart cities," *IEEE Communications Magazine*, vol. 55, no. 1, pp. 76–83, 2017.
[7] O. Arouk, A. Ksentini, and T. Taleb, "Group paging-based energy saving for massive mtc accesses in lte and beyond networks," *IEEE Journal on Selected Areas in Communications*, vol. 34, no. 5, pp. 1086–1102, 2016.
[8] D. Magrin, C. Pielli, C. Stefanovic, and M. Zorzi, "Enabling LTE RACH collision multiplicity detection via machine learning," *CoRR*, vol. abs/1805.11482, 2018.
[9] "Rnn." https://en.wikipedia.org/wiki/Recurrent_neural_network.
[10] A. McDiarmid, J. Irvine, S. Bell, and J. Banford, "CRAWDAD dataset strath/nodobo (v. 2011-03-23)." Downloaded from https://crawdad.org/strath/nodobo/20110323, Mar. 2011.
[11] M. Shanker, M. Y. Hu, and M. S. Hung, "Effect of data standardization on neural network training," *Omega*, vol. 24, pp. 385–397, August 1996.