LightFEC: Network Adaptive FEC with a Lightweight Deep-Learning Approach

Conference Paper · October 2021

DOI: 10.1145/3474085.3475528

4 authors:

Han Hu
Peking University
6 PUBLICATIONS **16** CITATIONS
SEE PROFILE

Sheng Cheng
Peking University
6 PUBLICATIONS **9** CITATIONS
SEE PROFILE

Xinggong Zhang
Peking University
69 PUBLICATIONS **834** CITATIONS
SEE PROFILE

Zongming Guo
Yantai Nanshan University
211 PUBLICATIONS **3,523** CITATIONS
SEE PROFILE

LightFEC: Network Adaptive FEC with a Lightweight Deep-Learning Approach

Han Hu, Sheng Cheng, Xinggong Zhang, Zongming Guo*
Wangxuan Institute of Computer Technology, Peking University, Beijing, China
{huhan951753,chaser_wind,zhangxg,guozongming}@pku.edu.cn,

ABSTRACT

Nowadays, the interest of real-time video streaming reaches a peak. To deal with the problem of packet loss and optimize users' Quality of Experience (QoE), Forward error correction (FEC) has been studied and applied extensively. The performance of FEC depends on whether the future loss pattern is precisely predicted, while the previous works have not provided a robust packet loss prediction method. In this work, we propose LightFEC to make accurate and fast prediction of packet loss pattern. By applying long short-term memory (LSTM) networks, clustering algorithms and model compression methods, LightFEC is able to accurately predict packet loss in various network conditions without consuming too much time. According to the results of well-designed experiments, we find out that LightFEC outperforms other schemes on prediction accuracy, which improves the packet recovery ratio while keeping the redundancy ratio at a low level.

CCS CONCEPTS

• **Information systems** → **Multimedia streaming**; • **Networks** → **Network performance analysis**; • **Computing methodologies** → **Machine learning approaches**.

KEYWORDS

Packet Loss Prediction, Deep Learning, Network-adaptive Streaming, Long Short-term Memory (LSTM) Network

ACM Reference Format:
Han Hu, Sheng Cheng, Xinggong Zhang, Zongming Guo. 2021. LightFEC: Network Adaptive FEC with a Lightweight Deep-Learning Approach. In *Proceedings of the 29th ACM Int'l Conference on Multimedia (MM '21), Oct. 20–24, 2021, Virtual Event, China.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3474085.3475528

1 INTRODUCTION

In recent years, with real-time video streaming getting more and more prevalent, the problem of packet loss becomes more serious and intolerable as well. Network video streaming is expected to account for 82% of Internet traffic by 2022, and a growing share of network video will take the form of live streaming video [8]. However, packet loss is proved to be a critical problem. Packet loss occurs when one or more packets of data traveling across a computer network fail to reach their destination. When network conditions are terrible, packet loss may happen frequently, causing crackling noise and jitters, thus degrading users' Quality of Experience (QoE) [5, 9, 20]. Some complex protocols such as Transmission Control Protocol (TCP) have their own error correction mechanisms to avoid packet loss and provide error-checked streaming, but they may not satisfy the demand of low latency. For instance, Automatic Repeat-reQuest (ARQ) is a traditional error correction method, but its recovery delay is longer than a round trip time (RTT). Therefore, the methods like ARQ are not appropriate choices for Real-Time Communication (RTC), which has strict requirements on delay.

Forward error correction (FEC) is a widely acknowledged technique for controlling errors in data transmission over unreliable or noisy communication channels [21, 25, 28], but the existing FEC methods still have shortcomings. By encoding in a redundant way, the sender generates extra packets according to the source. When packet loss happens, the redundant packets allow the receiver to detect the errors and correct them without retransmission. With the redundant packets generated according to the source data, packet loss can be avoided to some extent. Nevertheless, the performance of FEC usually depends on the numbers of lost and redundant packets. If the redundant packets are far less than the lost ones, the receiver will be unable to correct them [23, 29, 31]. And if there are too many redundant packets, it wastes the available bandwidth while improving the redundancy ratio. As a result, an accurate forecast of packet loss can optimize the performance of FEC.

To improve the recovery ratio while reducing the redundancy ratio, adaptive FEC methods have been suggested to dynamically generate redundant packets according to the packet loss feedback information, however, they are still not so satisfactory. Figure 1 is the diagram of a typical adaptive FEC algorithm. It changes the coding strategy judging by the loss feedback from the RTC receiver.

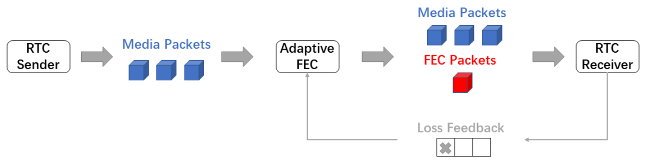
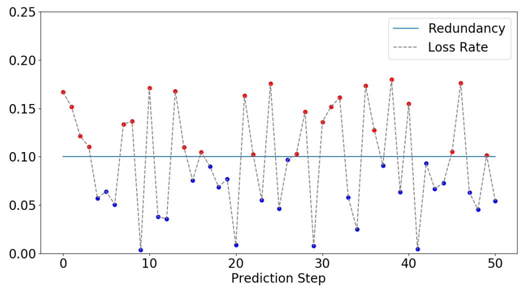
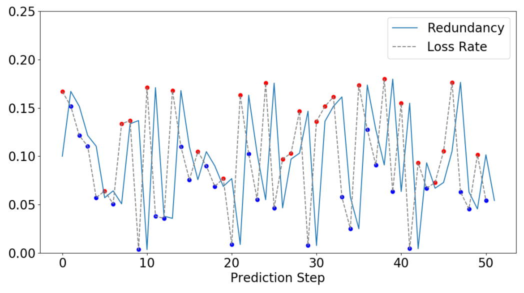


Figure 1: The diagram of a scheme using adaptive FEC. It generates redundant packets according to the source media packets and loss feedback for making correction.

*Corresponding author.
We thank the contributions of Zhimin Xu and Yue Wang, Beijing Bytedance Technology Co., Ltd.. This work was sponsored by National Key R&D Project of China (2019YFB1802701,2019YFF0302903).



(a) Fix-RS



(b) Adaptive-FEC

Figure 2: The packet loss rate predictions of the two traditional algorithms. The red dots means that the source packets cannot be completely recovered at that step, while the blue dots are the opposite.

Unfortunately, most of the FEC algorithms are not powerful enough, mainly because of these two shortcomings:

- **Static models.** It is normal to think that the incoming packets will emerge the same or at least a similar pattern as the historical ones, but it is not always the case [2, 26]. As a matter of fact, the network conditions change with time and the state of network may fluctuate frequently, which deeply affects the packet loss patterns [1, 10, 11]. Therefore, it is unwise to make predictions for packet loss just with a static model, which could result in intolerable mistakes.
- **Simple algorithms.** Even if when the network is relatively stable, the relation between future and historical packet loss pattern is not as simple as just sharing the same loss rate. The traditional prediction methods like regression analysis cannot derive the real correlation, and they often ignore the differences of various network conditions [13, 27].

Thanks to the development of deep learning, we are now possessing quantities of useful tools to solve the problem, while there are some necessary reformations. Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture [14, 16, 19]. With the feedback connections, LSTM is able to deal with data sequences while other standard neural networks can only process single data points. LSTM networks are well-suited to classifying and making predictions based on time series data, which makes it an appropriate model to predict the packet loss patterns [15, 32]. In the meanwhile, the clustering algorithms can help us distinguish diverse network conditions. A weakness of LSTM as well as the other neural networks is the high complexity [3, 4, 6]. It consumes a lot of time to train an LSTM network, especially when the dataset is very large. Even though a well-trained LSTM network is provided, the computation complexity of predicting is too high for RTC applications to spare the time during real-time streaming. Therefore, lightweight techniques should be applied to the LSTM networks [17, 22, 33].

In this paper, we propose **LightFEC**, a network adaptive FEC system, to more accurately predict packet loss patterns and improve the performance of FEC. Based on LSTM, a powerful deep neural network aiming at processing time-related data sequences, together with clustering methods and lightweight techniques, **LightFEC** can perfectly solve the challenges in real-time streaming applications. Firstly, it makes good use of clustering algorithms so as to fit with different network conditions. In this way, both the prediction accuracy and generalization ability can be improved. Secondly, deep learning methods have been applied to accurately derive the relation between historical and future packet loss patterns. And finally, it utilizes lightweight methods to ensure that the whole model can be implanted on the premise of normal real-time prediction of network.

We have carried out extensive experiments under controlled testbed and different network conditions with large datasets collected on-line. After comparing our proposed scheme with baseline algorithms and general LSTM networks, we found that our approach outperforms the other methods in prediction accuracy. With the lightweight technique applied, the efficiency of on-line predicting is greatly improved.

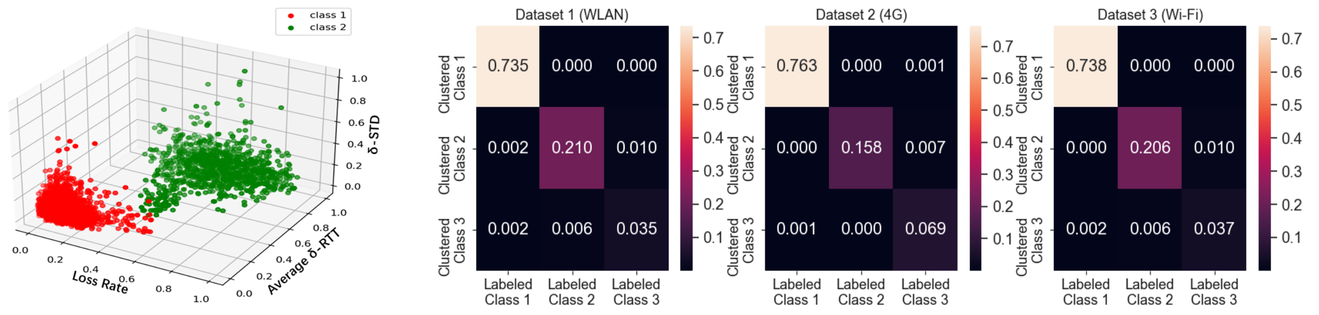
The rest of paper is organized as follows. Section 2 introduces the motivations and observations of our work. Section 3 specifically presents the structure designs as well as the processes of training and predicting of our proposed approach. In Section 4, we reveal our experiments and analyze the results to evaluate the different methods' performances, and eventually we come to the conclusion in Section 5.

2 MOTIVATIONS AND OBSERVATIONS

2.1 Motivations

There are lots of works about FEC [21, 25, 28], focusing on precisely predicting packet loss pattern and generating redundant packets according to the results, but the performance are not entirely satisfactory. On the whole, there are three problems of these works about FEC that are not yet well solved, which have great influence on the performance of FEC methods.

Problem 1: using a static model to deal with different network conditions. A representative category of FEC methods is traditional, which do not make any prediction about the changes of packet loss rate. For example, **Fix-RS** always gives a fixed number as the redundancy ratio, assuming that the network conditions will maintain on the same level all the time. Obviously, it has limitations to some extent, and may not work well in all circumstances. Figure 2(a) shows that in real-time network streaming, this kind of algorithms can easily make mistakes, thus being unable to recover all the source packets. **Fix-RS** assumes that the network conditions



(a) Clustering Result

(b) Clustering Similarity

Figure 3: The result of DBSCAN on the training dataset, together with the similarity of clustering results over different datasets.

always keep stable and do not fluctuate frequently, so it applies a static model without classifying different network conditions. That is to say, the network fluctuations which could significantly influence packet loss are neglected. Nevertheless, this premise is not always confidential, and it can result in intolerable errors.

Problem 2: the lack of powerful algorithms for figuring out the routine of packet loss patterns. Another category of FEC methods applies relatively simple prediction algorithms, trying to adapt to the fluctuations of network. **Adaptive-FEC** is a classical method, which thinks that the future packet loss pattern is completely determined by the history, and just takes the historical loss rate as prediction result. **Adaptive-FEC** wants to distinguish network conditions, but it does not use parameters to quantify the difference. On the contrast, it assumes that the future loss pattern is deeply dependent on or even almost identical to the historical loss pattern. Figure 2(b) reveals that **Adaptive-FEC** also frequently make mistakes.

Problem 3: the lack of generalization ability. Apart from the traditional FEC methods, some schemes use complex algorithms like deep learning methods, but there are also some shortcomings, such as over-fitting and the huge time consumption on prediction. In order to derive the correlation, it is normal to think of deep learning methods. This category of data-driven algorithms can learn from large datasets and find out the hidden relations. However, if it just relies on off-line training without preprocessing on the whole dataset, it can be easily over-fitting since the samples of different network states differ greatly in number. This problem makes the naive off-line training unable to make correct predictions under various network conditions. For instance, **DeepRS** [7], a loss-predicting system based on LSTM model, ignores the differences between various network conditions and treats all the loss patterns in the same way, which could make unbearable prediction mistakes in some clusters of samples. Besides, the structure is too complex and consumes a lot of time while predicting packet loss patterns.

In view of the deficiencies revealed by the previous works, a new tool which is network adaptive and lightweight should be raised to efficiently deal with the problem of predicting packet loss patterns.

2.2 Observations

In order to overcome the shortcomings of existed works, we put forward two assumptions, and make experiments to demonstrate them.

Assumption 1: statistics of RTT and Loss Rate are reasonable parameters to quantify network conditions. As is mentioned in Section 2.1, it is necessary to find out the metrics to quantify network conditions. RTT and Loss Rate seem to be good choices, because they are not only directly influenced by network conditions but also easy to measure or calculate. Since RTC is based on User Datagram Protocol (UDP), There is not as much reference information in the packet headers as TCP. We choose RTT, which is attached to any data packet and is able to directly present the current network conditions. In the meanwhile, we calculate packet loss rate in the historical and future block, due to that loss rate can also reveal whether the network conditions are good or not to some extent. By calculating the averages of RTT and loss rate for a packet sequence, we can estimate whether network conditions are good or bad. Besides, judging by the standard deviation of RTT, the fluctuations of network conditions can also be quantified.

To evaluate whether the three metrics can classify different network conditions, we also need an appropriate classification algorithm. Since the network conditions are not inherent properties of a packet sequence, we take the problem of distinguishing diverse network conditions as an unsupervised learning problem. Therefore, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algorithm [12] can divide the network conditions into several categories. The clustering results are shown in Figure 3(a). We use min-max normalization to transform the value of RTT into a decimal between 0 and 1, then name the new metric as δ-RTT, and we name the standard deviation of δ-RTT as δ-STD. We construct the feature space with Loss Rate, average δ-RTT and δ-STD. The points in the figure are corresponding with data samples. It can be spotted that the data samples are mainly classified into two clusters, named as Class 1 and 2. The other samples are too isolated to cluster, so they are classified into an additional Class 3. The two main clusters contain over 90% samples of training dataset, and they are clearly separated. These results prove that Loss Rate, δ-RTT and δ-STD are reasonable parameters to quantify network conditions.

Assumption 2: the classification of network conditions is robust. If our classification of network conditions is only available to the training dataset, it cannot be applied to real Internet environment.

In order to evaluate the generalization ability, we have collected data samples with different access types (WLAN, 4G, Wi-Fi). The similarity of clustering results over these datasets are shown in

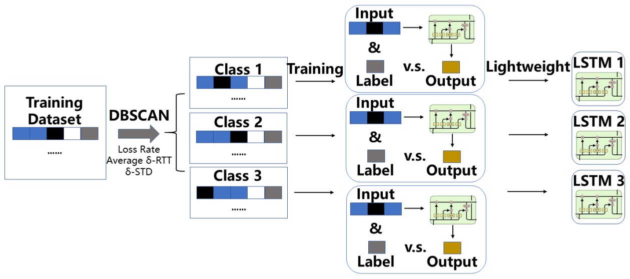


Figure 4: The process of training. All the training data go through clustering, then each cluster trains its own LSTM model.

Figure 3(b). X-axis records the results of labeling samples the cluster they belong to according to the previous trained model, and y-axis represents the results of applying DBSCAN to the datasets themselves. The value of row i, column j means the ratio of samples clustered as Class i by DBSCAN, and labeled as Class j by KNN (K-Nearest Neighbor) based on the previous trained model. For each dataset, the sum of all the nine values is 1. For Dataset 1, the ratio of samples labeled as Class 1 or 2 is over 95%, which means that most of the samples can be well classified according to network conditions. As for Dataset 2, the proportion of Labeled Class 2 drops a little, but the ratio of Labeled Class 2 increases accordingly, thus the number of samples labeled as Class 3 is still very small. The results of Dataset 3 is similar to that of Dataset 1. It can be spotted that the diagonal values are always bigger than the other ones for each dataset, meaning that the clusters mainly keep a relatively high similarity. Therefore, it is convincing that the classification of network conditions is robust whatever the dataset is.

3 SYSTEM DESIGN

We have designed **LightFEC**, a lightweight network-adaptive FEC system which has good generalization ability, to better solve the problem of predicting packet loss. We take the problem of distinguishing diverse network conditions as an unsupervised learning problem. With the help of reference information, Loss Rate, average δ-RTT and δ-STD in the historical sequences, the DBSCAN algorithm can divide the network conditions into several categories. After that, each cluster trains its own LSTM network off-line, so that the models can independently predict the samples of different network conditions. The general forms of input, output and loss function of LSTM are not suitable for packet loss pattern prediction, so we have made some improvements about them. Finally, we select weight pruning as the lightweight method to compress the LSTM network, so as to decrease the time of prediction to a level that the RTC applications can afford.

3.1 System Overview

LightFEC mainly consists of two modules: 1) DBSCAN module, for classifying data samples according to the different network conditions, and 2) Lightweight LSTM predicting module, for predicting the future loss pattern based on the historical packets.

During the training process, we carry out some preliminaries for the two modules, as is shown in Figure 4. Firstly, the whole training dataset is divided into several clusters by applying DBSCAN algorithm. Samples in the same cluster share similar network conditions, while samples of different clusters may be collected under distinct network states. Secondly, each cluster separately trains an LSTM model based on its own data samples, so as to obtain a reliable tool to predict packet loss pattern of its network conditions. Eventually we apply compression methods to the trained models, and obtain the final lightweight LSTM networks.

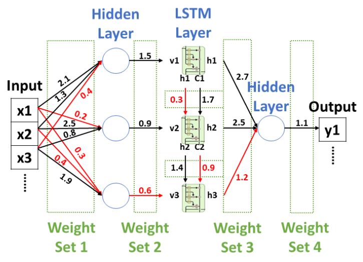


Figure 5: Compressing neural networks by weight pruning method. For each weight set, we sort the weights by their absolute values and remove them by masking to zero. The red lines represent the weights to be pruned.

3.2 Clustering Algorithm

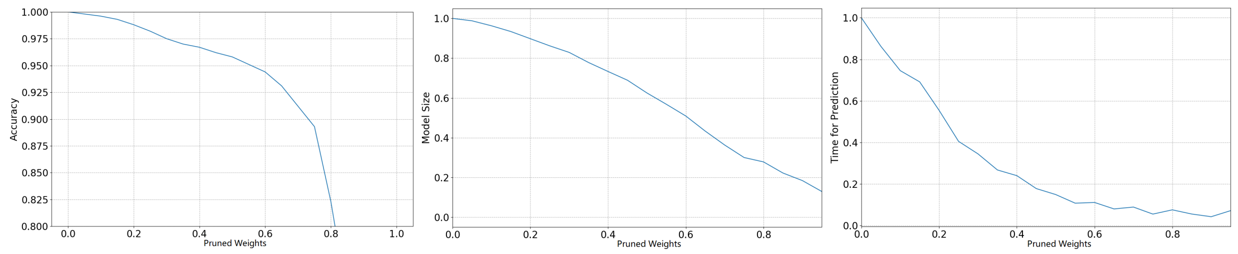
We apply clustering algorithm to independently train models for different network conditions so as to overcome the shortcoming that a model trained without distinguishing network states could make unbearable mistakes. If we train the model on the whole dataset, it is impossible to distinguish different network conditions. A clustering algorithm can not only solve this problem but also improve the generalization ability of the whole model.

As for the specific clustering algorithm, we choose DBSCAN to divide the feature space in a data-driven way. DBSCAN is a density-based clustering algorithm: given a set of points in some space, it groups together points with many nearby neighbors, and mark as outliers points that lie alone in low-density regions. Considering a set of points in some space to be clustered, DBSCAN uses a parameter ϵ, which represents the threshold of density, to classify the points as core points, density-reachable points and outliers. Unlike k-means [30], DBSCAN does not require the number of clusters as a parameter. Rather it infers the number of clusters based on the data, and it can discover clusters of arbitrary shape. For comparison, k-means usually discovers spherical clusters. In Figure 3(a), we set $\epsilon = 0.0037$ to ensure the ratio of main clusters is over 90%. Section 2.2 tells why we select Loss Rate, average δ-RTT and δ-STD to identify different network conditions. For a packet sequence, the Loss Rate and average δ-RTT can reveal the overall quality of network, while we can calculate δ-STD to measure the fluctuations of network. δ-RTT is defined as:

$$\delta\text{-}RTT = \frac{RTT - RTT_{min}}{RTT_{max} - RTT_{min}},$$

then we can calculate average δ-RTT:

$$\overline{\delta\text{-}RTT} = \frac{1}{N}\sum_{i=1}^{N} \delta\text{-}RTT_i,$$



(a) Accuracy (b) Model Size (c) Time of Prediction

Figure 6: The ratio of accuracy, model size and computation time of prediction after lightweight process. y-value represents the relative value of primary LSTM network without applying lightweight method.

and δ-STD:

$$\delta\text{-}STD = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\delta\text{-}RTT_i - \overline{\delta\text{-}RTT})^2}.$$

Therefore, we choose these three parameters to form the input vector of DBSCAN:

$$\mathbf{v}_{DBSCAN} = \{LossRate, \overline{\delta\text{-}RTT}, \delta\text{-}STD\},$$

and it outputs the cluster which the sample belongs to.

3.3 LSTM Design

In this section, we introduce the design of LSTM module, with which we can make accurate predictions for the time-related packet sequences. LSTM is an artificial RNN architecture. With the feedback connections, LSTM networks are well-suited to classifying and making predictions based on time series data, which makes it an appropriate model to predict the packet loss patterns.

In order to improve the performance of LSTM, we have changed the form of input and output vectors for this module. To train and evaluate the LSTM networks for predicting the packet loss pattern, we have collected data traces from real Internet. During an RTC streaming, for every 500 packets, we take the first 300 packets as the historical loss pattern, the second 100 packets as a gap, and the last 100 packets as the future loss pattern to be predicted. The gap is set for this reason. Real-time multimedia streaming carried by protocols like UDP is strictly constrained on delay. Consequently, source packets are sent continuously with a very short interval which is less than RTT significantly. In another word, a source packet is already sent before the sender receives the feedback of the last packet. After setting the gap, we can ensure that while applying this scheme in real world, the historical loss pattern of every sample has always been collected before predicting. Instead of simply using a boolean value to represent each packet is lost or not, we choose RTT to record the state:

$$\mathbf{v}_{LSTM} = \{r_1, r_2, ..., r_n\},$$

while

$$r_i = \begin{cases} -1, & \text{if packet loss happens,} \\ RTT, & \text{otherwise,} \end{cases}$$

for $i \in \{1, 2, ..., n\}$, and:

$$N = \begin{cases} 300, & \text{input,} \\ 100, & \text{output.} \end{cases}$$

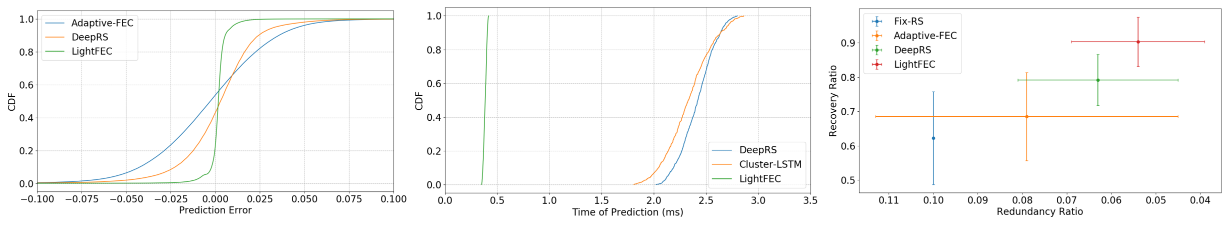
If a packet is successfully received, it has its own RTT, while a lost packet cannot confirm its RTT, so the RTT sequence can also express the loss pattern and contain more information than boolean vector. Although we have already calculated average RTT of the historical sequence for each sample as a feature of network conditions, the RTT of every packet can reveal the packet's state more precisely.

To improve the adaptability towards various network conditions, we have revised the form of loss function. General LSTM normally uses L2-norm loss function. However, we have found that for the samples in some clusters, the prediction result tend to lean to one side, so we changed the form of loss function as:

$$L = \frac{\lambda}{2n_1}\sum_{x\in S_1}\|y(x) - a(x)\|^2 + \frac{\mu}{2n_2}\sum_{x\in S_2}\|y(x) - a(x)\|^2$$

where L represents the cost, x represents the sample, y represents the actual value, and a represents the output value. $n_1 = |S_1|$, $n_2 = |S_2|$, $\lambda + \mu = 1$. S_1 includes samples of which the predicted loss rate is lower than the real loss rate, and S_2 consists of samples that the predicted loss rate is higher. λ and μ represent the weights of the two sets. By adjusting the weights, we can give different punishments to the two kinds of samples and improve the prediction accuracy.

The final result is also reformed. We do not simply use the output vector of LSTM network as the prediction result, instead, we calculate the packet loss rate based on the vector, and regard that number as the final result. For the output, a vector form is not suitable for solving the problem of packet loss. On the one hand, according to the network conditions, the amount of lost packets in an incoming block can be predicted by learning from the historical pattern, but the loss state of each packet is hard to be exactly determined because of randomness. On the other hand, actually we do not need to know the packet loss state for each packet accurately. What we are concerned about is the count of losses in this block, which decides the parameters of FEC packets. As a result, we attach a fully connected layer to the end of LSTM model, reducing the dimension of the output to 1.



(a) Prediction Accuracy (b) Time of Prediction (c) Redundancy and Recovery Ratio

Figure 7: Performance over test dataset. Error bars show 95% confidence intervals.

3.4 Lightweight Method

Because of the high complexity of LSTM network, it is necessary to reduce the computation cost of LSTM at inferring phase, which cannot fulfill the requirement for low time consumption in RTC applications. According to our LSTM design, we should predict the packet loss pattern for every 100 packets, and encode FEC packets for them. Therefore, it is necessary to put forward lightweight method to compress the models, so as to decrease the time cost of on-line predicting.

For the well-trained LSTM models, we adopt weight pruning algorithm [18, 24] as our lightweight technique, which can significantly reduce the network size and computational complexity. Pruning algorithm usually aims to finding an effective evaluation method to judge the importance of connections (i.e., weight pruning) or neurons (i.e., neuron pruning), and to cut the unimportant ones to reduce the redundancy of the model. After pruning phase, the model should be retrained to ensure that the prediction accuracy does not reduce significantly.

Figure 5 is the diagram of our weight pruning algorithm. Firstly, we sort the weights by their absolute values. Then we remove the smallest weights of a certain ratio by setting them to zero. Since our LSTM network contains of different layers, we divide the weights into several sets, so as to separately sort and prune them. The weight sets are divided by the layers they connect. For the weights between LSTM cells, they are classified into the same set as the weights between LSTM layer and the next hidden layer. We repeat the process of pruning and retraining for the original LSTM network, so that we obtain the lightweight model. In this way, we can use a smaller weight matrix to store the neuron network, and reduce the time cost by making fewer calculations during the process of prediction.

The pruning method could reduce the size and complexity of LSTM, but it also drops the accuracy, so that we should decide the ratio of pruned weights by both our need of time cost and the accuracy loss. We have iterated all the pruned ratios, and the results are demonstrated in Figure 6. With the ratio of pruned weights increasing, the prediction accuracy tends to decline, but the difference is not so obvious before the ratio reaches 70%. The model size nearly has a linear correlation with pruned weights. As for time cost of making a prediction, it drops rapidly at first, but it does not change obviously after the ratio reaches 80%. According to these phenomenons, it can be spotted that the pruning methods work well on solving our problem. Even if we remove 60% of the less important weights, the accuracy loss is less than 5%, while the time of prediction can reduce to less than 20% of the initial value. Therefore, 60% may be an appropriate ratio of weights to be pruned, and we decide this value to prune and retrain our LSTM network so as to obtain the final model.

3.5 Process of Predicting

In this section, we summarize what happens during the process of predicting, to show how **LightFEC** achieves its generalization ability.

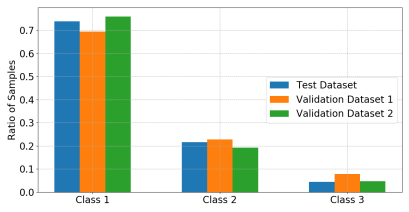
As for the the process of predicting, each new unlabeled sample checks Loss Rate, average δ-RTT and δ-STD of its input vector, then using KNN to label which cluster it belongs to, instead of doing DBSCAN again. In this way, we do not need to spend extra time collecting a new dataset and clustering. On the contrast, we apply the previous well-trained clustering results to the new samples, which is proved convincing in Section 2.2. After that, the LSTM model corresponding to this cluster makes prediction and gives the prospective future loss pattern. Finally, we generate the redundant packets according to the results of predicted packet loss pattern. The adaptation of the entirely new test samples reveals the generalization ability of the scheme.

4 EXPERIMENTS AND EVALUATIONS

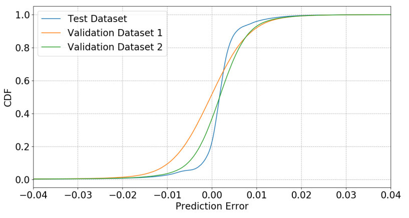
We have collected packet loss data samples from November 1st to December 30th on a cloud server. The whole dataset contains about 540 million data samples, each of which is a packet loss sequence of 500 packets, with the dividing mode described in Section 3.3. We randomly choose half of this dataset for training the data-driven models, and evaluate them in the following sections without extra training. The other half of the dataset is used for experiments in Section 4.1.

To validate the performance of **LightFEC**, we have carried out extensive experiments, including the evaluations over various datasets. Based on the results in Section 3.4, we select 60% as the ratio of pruning to build our final **LightFEC** model. After all the reform and improvements of our proposed scheme, we can evaluate its performance on the real-world validation dataset and compare it with other packet loss pattern prediction algorithms. We select the following methods for comparison:

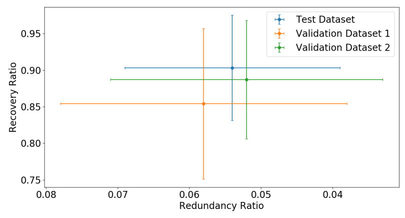
- **Fix-RS**: A naive RS methods, which applies a fixed redundancy ratio.
- **Adaptive-FEC**: The widely used packet loss prediction method, which calculates the loss rate of historical pattern and just take it as the prediction result.
- **DeepRS**: An LSTM network trained on the whole training dataset. The form of its input is binary vectors, and the



(a) Sample Distribution



(b) Prediction Accuracy



(c) Redundancy and Recovery Ratio

Figure 8: Performance over various network conditions. Error bars show 95% confidence intervals.

loss function is general L2-norm loss function without any adjustment.

- **LightFEC**: This is our proposed method. It utilizes DBSCAN to divide the training dataset, independently train an LSTM network for each cluster with RTT pattern input and adjusted loss function. After training, all the LSTM networks go though 60% weight pruning to improve the efficiency.

In performance comparison, we take the following measurement metrics into consideration:

- **Recovery Ratio**. The ratio of recovered packets to all lost packets. For instance, recovery ratio is 1 when all lost packets are recovered. Conversely, recovery ratio is 0 if all lost packets cannot be recovered.
- **Redundancy Ratio**. The ratio of redundant packets to source packets. For instance, if RS module generates k FEC packets with a block including b source packets, the redundancy ratio is $\frac{k}{b}$.

4.1 Performance over Test Dataset

The first part of our experiments is to evaluate the schemes in the same network conditions as training environment. In this section, we construct Test Dataset by choosing the other half of the data samples which are not used to train the models, as is mentioned in Section 4. In this way, we can compare the performance of different schemes over Test Dataset, mainly judging by three aspects: a) the prediction accuracy of schemes using adaptive algorithms, b) the time cost of making a prediction, and c) the redundancy and recovery ratio for each scheme.

In order to evaluate whether the network-adaptive schemes can precisely make predictions for packet loss patterns, we summarize the distributions of prediction error, and record the results in Figure 7(a). For each sample, the difference value between its real loss rate and the prediction result is defined as prediction error. **Fix-RS** always construct FEC packets with the same redundancy ratio, without making any prediction, so it is not included in this figure. It can be spotted that **Adaptive-FEC** tend to make more mistakes in predicting packet loss patterns comparing with the other schemes, because of its naive algorithm. Once the network conditions change frequently, the prediction results of **Adaptive-FEC** are no more credible. Both **DeepRS** and **LightFEC** make relatively accurate predictions owing to their LSTM network, but **LightFEC** outperforms **DeepRS** with the reformation of network and the pre-classification of clustering algorithm.

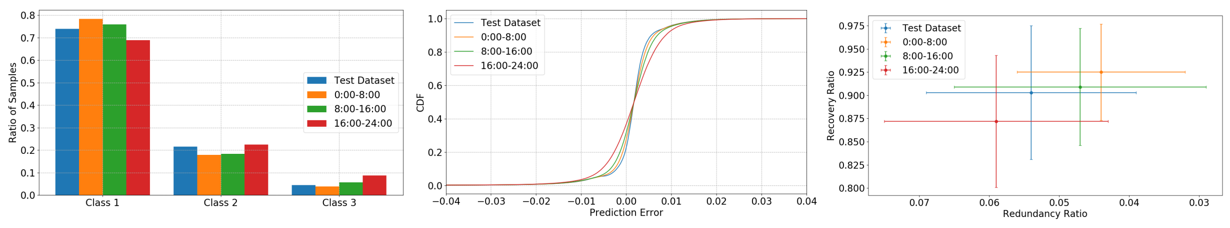
The detailed evaluation of computation complexity is shown in Figure 7(b). **Cluster-LSTM** refers to the scheme containing DBSCAN module and LSTM network. The only difference between **Cluster-LSTM** and **LightFEC** is that the former does not apply the lightweight method. By comparing the two versions, we can figure out the influence in performance of pruning method. Among the three LSTM-network-based schemes, **DeepRS** and **Cluster-LSTM** do not apply lightweight methods, thus consuming a lot of time to make predictions, mainly between 2-3ms. As is mentioned in Section 3.4, the time cost cannot always fulfill the requirements of RTC applications. However, as for **LightFEC**, the time of making a prediction is about 0.4ms, much shorter than the other schemes. The removal of less important weights saves the time, leading into a faster prediction process.

We also compared the recovery and redundancy ratio in Figure 7(c), for the four schemes mentioned in Section 4. The higher recovery ratio (y-value) and the lower redundancy ratio (x-value) mean the better performance, thus we reverse x-axis so that the top rightmost point represents the best performance. According to the results, we can see that even if pruning method leads to some loss, **LightFEC** still gets the lowest redundancy ratio and the highest recovery ratio, comparing with the other schemes. **Fix-RS** always chooses the same redundancy ratio, which cannot efficiently solve the problem of packet loss. For **Adaptive-FEC**, the redundancy ratio varies significantly, but it does not bring too much improvement in recovery ratio. Although **DeepRS** benefits from LSTM network, it still obtain worse performance compared with **LightFEC**.

4.2 Performance over Various Network Conditions

To validate the performance over different network conditions, we have collected packet loss data from December 31st to January 15th on another cloud server, with access type set as 4G and Wi-Fi, named as Validation Dataset 1 and 2. The datasets contain about 150 million data samples, each of which is a packet loss sequence of 500 packets, with the dividing mode described in Section 3.3. By changing the server and type of access link, the network conditions may vary greatly, which makes the performance more convincing. We use the same trained models as in Section 4.1 without any training over the new datasets.

In order to figure out whether the results of clustering are still convincing, we calculate the ratios of samples classified as Class 1, 2 and 3, by straightly using the well-learned model over training dataset to label them, instead of applying DBSCAN algorithm to the validation datasets. The distributions are summarized in Figure 8(a).



(a) Sample Distribution (b) Prediction Accuracy (c) Redundancy and Recovery Ratio

Figure 9: Performance over time-varying datasets. Error bars show 95% confidence intervals.

Table 1: Performance over Various Network Conditions ([Average Recovery Ratio, Average Redundancy Ratio])

	Fix-RS	Adaptive-FEC	DeepRS	LightFEC
Test	[0.623, 0.100]	[0.685, 0.079]	[0.792, 0.063]	[0.903, 0.054]
Validation 1	[0.598, 0.100]	[0.624, 0.088]	[0.743, 0.075]	[0.854, 0.058]
Validation 2	[0.619, 0.100]	[0.633, 0.085]	[0.787, 0.068]	[0.887, 0.052]
All Data	[0.616, 0.100]	[0.657, 0.083]	[0.779, 0.067]	[0.887, 0.055]

As is mentioned in Section 3.2, Class 3 mostly consists of outliers. For each dataset, most of the samples are classified in Class 1 and 2, meaning that the network conditions can be well identified.

The LSTM module is also evaluated over these validation datasets. As for the prediction accuracy of packet loss patterns, Figure 8(b) reveals whether **LightFEC** can precisely predict the packet loss patterns for these datasets. Although the access type has an significant influence on the network conditions, it can be spotted that **LightFEC** maintains its prediction accuracy.

As for the overall performance, Figure 8(c) shows the results of evaluating our proposed **LightFEC** according to redundancy and recovery ratio. Since Validation Dataset 1 is collected over 4G network, there are probably more network fluctuations, leading to relatively higher loss rate together with higher redundancy ratio. But the degradation is still tolerable, and the recovery ratio still keeps at a high level. For Validation Dataset 2, the performance is similar to that of Test Dataset. The results prove that **LightFEC** has the potential to be applied into various network conditions without changing the primary trained model.

The evaluation results of the four schemes over the different datasets are listed in Table 1. Whichever the scheme is, the overall performance drops over the validation datasets, since the network conditions have gaps. However, the degradation of **LightFEC** is still acceptable, and its recovery ratio is always higher comparing with the other schemes.

4.3 Performance over Time-varying Datasets

Besides the properties of network links, the time of data transmission also makes influence on packet loss patterns, since network conditions can vary over time. In order to evaluate whether our proposed scheme is able to overcome this kind of fluctuations, we divide Test Dataset into three parts according to the time of day, and validate **LightFEC** over them. We still use the same trained models as in Section 4.1 without extra training over the new datasets.

Owing to the need of evaluating the performance of clustering module, the sample distributions of these datasets are displayed in Figure 9(a). As is shown in Figure 3(a), Class 1 contains of data samples with relatively lower Loss Rate and RTT, which means that the network conditions may be smooth without too much error. The samples collected from 0:00 to 16:00 do not need to face too much network congestion, so the ratio of Class 1 is a little higher. On the contrast, The samples collected from 16:00 to 24:00 are more likely to suffer from network congestion, resulting in higher ratio of Class 2. However, Class 3 still consists of the fewest samples, proving that these datasets are well classified.

Table 2: Performance over Time-varying Datasets ([Average Recovery Ratio, Average Redundancy Ratio])

	Fix-RS	Adaptive-FEC	DeepRS	LightFEC
0:00-8:00	[0.709, 0.100]	[0.722, 0.035]	[0.875, 0.058]	[0.925, 0.044]
8:00-16:00	[0.651, 0.100]	[0.694, 0.067]	[0.834, 0.062]	[0.909, 0.047]
16:00-24:00	[0.570, 0.100]	[0.621, 0.093]	[0.802, 0.079]	[0.872, 0.059]
All Data (Test)	[0.623, 0.100]	[0.685, 0.079]	[0.792, 0.063]	[0.903, 0.054]

As for the prediction accuracy of LSTM module, Figure 9(b) shows the results of prediction error. It can be spotted that the prediction accuracy does not reduce evidently, which means that the LSTM module still works well in making predictions over these time-varying datasets.

The performance over these datasets are also validated. According to redundancy and recovery ratio in Figure 9(c), the results are even better with the datasets of 0:00-16:00. For the samples of 16:00-24:00, they may suffer from the worst network conditions, which leads to the highest redundancy ratio, but the reduce in recovery ratio is still acceptable. The results inform that **LightFEC** has the potential to deal with network fluctuations overtime.

We also evaluate the overall performance of the four schemes over the time-varying datasets, and list the results in Table 2. Comparing with the other schemes, **LightFEC** always gets the lowest redundancy ratio and the highest recovery ratio, owing to its well-trained models and generalization ability.

5 CONCLUSION

In this paper, we propose LightFEC, a packet loss pattern prediction scheme based on LSTM networks, to accurately predict packet loss pattern in real-time streaming applications. LightFEC is network adaptive by using DBSCAN to distinguish network conditions according to statistics of historical RTT and packet loss rate, and lightweight by applying pruning methods. Unlike traditional packet loss prediction algorithms, LightFEC can deal with various network conditions and analysis the relation between historical and future loss patterns. According to the results of well-designed experiments, we find out that LightFEC outperforms other schemes on prediction accuracy, and improves the packet recovery ratio while keeping the redundancy ratio at a low level.

REFERENCES

[1] Daron Acemoglu, Vasco M Carvalho, Asuman Ozdaglar, and Alireza Tahbaz-Salehi. 2012. The network origins of aggregate fluctuations. *Econometrica* 80, 5 (2012), 1977–2016.

[2] Ender Ayanoglu, Richard D Gitlin, Thomas F La Porta, Sanjoy Paul, and Krishan K Sabnani. 1997. Adaptive forward error correction system. US Patent 5,600,663.

[3] Peter L Bartlett. 1998. The sample complexity of pattern classification with neural networks: the size of the weights is more important than the size of the network. *IEEE transactions on Information Theory* 44, 2 (1998), 525–536.

[4] Monica Bianchini and Franco Scarselli. 2014. On the complexity of neural network classifiers: A comparison between shallow and deep architectures. *IEEE transactions on neural networks and learning systems* 25, 8 (2014), 1553–1565.

[5] Jill M Boyce and Robert D Gaglianello. 1998. Packet loss effects on MPEG video sent over the public Internet. In *Proceedings of the sixth ACM international conference on Multimedia*. ACM, 181–190.

[6] Colin Campbell. 1997. Constructive learning techniques for designing neural network systems. (1997).

[7] Sheng Cheng, Han Hu, Xinggong Zhang, and Zongming Guo. 2020. DeepRS: Deep-learning Based Network-Adaptive FEC for Real-Time Video Communications. *arXiv preprint arXiv:2001.07852* (2020).

[8] Cisco. 2019. “Cisco Visual Networking Index: Forecast and Trends, 2017–2022 White Paper”. https://www.cisco.com/c/en/us/\solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html.

[9] Alan D Clark, Ph D Fellow Iee, et al. 2001. Modeling the effects of burst packet loss and recency on subjective voice quality. (2001).

[10] Paulin Coulibaly, Francois Anctil, Ramon Aravena, and Bernard Bobée. 2001. Artificial neural network modeling of water table depth fluctuations. *Water resources research* 37, 4 (2001), 885–896.

[11] M Argollo De Menezes and A-L Barabási. 2004. Fluctuations in network dynamics. *Physical review letters* 92, 2 (2004), 028701.

[12] M. Ester. 1996. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. *Proc.int.conf.knowledge Discovery & Data Mining* (1996).

[13] Venkat Raju Gandikota, Bheemajun Reddy Tamma, and C Siva Ram Murthy. 2008. Adaptive FEC-based packet loss resilience scheme for supporting voice communication over ad hoc wireless networks. *IEEE Transactions on Mobile Computing* 7, 10 (2008), 1184–1199.

[14] Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. 1999. Learning to forget: Continual prediction with LSTM. (1999).

[15] Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural networks* 18, 5-6 (2005), 602–610.

[16] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. 2016. LSTM: A search space odyssey. *IEEE transactions on neural networks and learning systems* 28, 10 (2016), 2222–2232.

[17] Song Han, Huizi Mao, and William J Dally. 2015. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149* (2015).

[18] Yang He, Ping Liu, Ziwei Wang, Zhilan Hu, and Yi Yang. 2019. Filter pruning via geometric median for deep convolutional neural networks acceleration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 4340–4349.

[19] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation* 9, 8 (1997), 1735–1780.

[20] Wenyu Jiang and Henning Schulzrinne. 2000. Modeling of packet loss and delay and their effect on real-time multimedia service quality. In *Proc. NOSSDAV*.

[21] Michael Luby, Lorenzo Vicisano, Jim Gemmell, Luigi Rizzo, M Handley, and Jon Crowcroft. 2002. *The use of forward error correction (FEC) in reliable multicast*. Technical Report. RFC 3453, December.

[22] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. 2017. Thinet: A filter level pruning method for deep neural network compression. In *Proceedings of the IEEE international conference on computer vision*. 5058–5066.

[23] Anthony J McAuley. 1990. Reliable broadband communication using a burst erasure correcting code. In *ACM SIGCOMM Computer Communication Review*, Vol. 20. ACM, 297–306.

[24] Pavlo Molchanov, Arun Mallya, Stephen Tyree, Iuri Frosio, and Jan Kautz. 2019. Importance Estimation for Neural Network Pruning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 11264–11272.

[25] Abdelhamid Nafaa, Tarik Taleb, and Liam Murphy. 2008. Forward error correction strategies for media streaming over wireless networks. *IEEE Communications Magazine* 46, 1 (2008), 72–79.

[26] Chinmay Padhye, Kenneth J Christensen, and Wilfrido Moreno. 2000. A new adaptive FEC loss control algorithm for voice over IP applications. In *Conference Proceedings of the 2000 IEEE International Performance, Computing, and Communications Conference (Cat. No. 00CH37086)*. IEEE, 307–313.

[27] Kihong Park and Wei Wang. 1998. AFEC: An adaptive forward error correction protocol for end-to-end transport of real-time traffic. In *Proceedings 7th International Conference on Computer Communications and Networks (Cat. No. 98EX226)*. IEEE, 196–205.

[28] Rohit Puri and Kannan Ramchandran. 1999. Multiple description source coding using forward error correction codes. In *Conference Record of the Thirty-Third Asilomar Conference on Signals, Systems, and Computers (Cat. No. CH37020)*, Vol. 1. IEEE, 342–346.

[29] Madhu Sudan. 1997. Decoding of Reed Solomon codes beyond the error-correction bound. *Journal of complexity* 13, 1 (1997), 180–193.

[30] Kiri Wagstaff, Claire Cardie, Seth Rogers, Stefan Schrödl, et al. 2001. Constrained k-means clustering with background knowledge. In *Icml*, Vol. 1. 577–584.

[31] Stephen B Wicker and Vijay K Bhargava. 1999. *Reed-Solomon codes and their applications*. John Wiley and Sons.

[32] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. 2015. Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*. 802–810.

[33] M. Zhu and S. Gupta. 2017. To prune, or not to prune: exploring the efficacy of pruning for model compression. (2017).