Impact of Network Performance on Cloud Speech Recognition

Mehdi Assefi, Mike Wittie, and Allan Knight†
Department of Computer Science, Montana State University
†Citrix Systems, Inc.
{mehdi.assefi, mwittie}@cs.montana.edu, {allan.knight}@citrix.com

***Abstract*—Interactive real-time communication between people and machine enables innovations in transportation, health care, etc. Using voice or gesture commands improves usability and broad public appeal of such systems. In this paper we experimentally evaluate Google speech recognition and Apple Siri – two of the most popular cloud-based speech recognition systems. Our goal is to evaluate the performance of these systems under different network conditions in terms of command recognition accuracy and round trip delay – two metrics that affect interactive application usability. Our results show that speech recognition systems are affected by loss and jitter, commonly present in cellular and WiFi networks. Finally, we propose and evaluate a network coding transport solution to improve the quality of voice transmission to cloud-based speech recognition systems. Experiments show that our approach improves the accuracy and delay of cloud speech recognizers under different loss and jitter values.**

***Index Terms*—Cloud speech recognition, Quality of Experience, Network coding, Streaming media, Real-time systems.**

I. Introduction

The assessment of the performance of cloud speech recognition applications under different network conditions has received much less attention than development of new systems. Although Apple Siri and Google Speech Recognition (GSR) are widely used to recognize spoken commands given to mobile devices, a solution robust to poor network performance is still missing. The result is user frustration in existence of network traffic problems.

The quality of user experience with applications that rely on cloud-based speech recognition depends upon the accuracy and delay of speech transcription. Therefore, voice transmission over a network and its conversion into commands are crucial building blocks of real-time interactive applications. In such applications, clients send voice, which need to be accurately converted to machine commands at the server. The performance of the speech recognition system should be consistent across different network conditions and robust to variation in network performance in terms of jitter, packet loss, and varying available bandwidth. To date, there has not been a systematic study of how speech recognition systems perform under different network conditions.

In this paper, we study cloud speech recognition systems under different network conditions. Specifically, we analyze popular interactive streaming systems – GSR, and Apple Siri under different network conditions. We then focus on behavior of standard UDP and TCP streaming methods to find solutions to improve streaming voice transmission to Dragon, on which Siri is based[3]. We evaluate each system under a range of jitter, and packet loss values, and consider the accuracy and delay of the recognized speech.

Results of our study show that packet loss and jitter have a considerable effect on the accuracy and round trip delay of cloud speech recognition. Using fountain codes with standard UDP we demonstrate a real-time interactive streaming system that improves Dragon's accuracy and delay under various traffic conditions [14]. Our results show that network coding over UDP improves the delay of speech recognition by much as 30% under packet loss. We also show that our solution improves the delay under jitter values. We argue that, future systems can offer better quality of service compared to current cloud speech recognition systems and offer better quality of experience in a wide range of interactive applications.

The reminder of this paper is organized as follows. In section II we describe our experimental testbeds. We evaluate our testbeds through extensive experiments in section III. In section IV we review the state of the art on Fountain codes in streaming systems and describe our solution using Fountain codes over UDP connection and evaluate the solution through the same experiments as experimental testbeds. In section V we review some related works and finally in section VI we conclude the paper.

II. Experimental testbeds

We create an evaluation testbed to study the performance of several speech recognition systems. Clients transmit voice data through a netem Linux box which introduces delay, jitter, and packet loss in the network. We set bandwidth to 2Mbps typical on 3G connections [12]. The server receives voice packets, translates the voice into text, and sends the text back to the client. Client receives the text and calculates the accuracy and delay of the received text. Levenshtein distance is used to calculate the transcription accuracy, or the match percentage of the original string and the string that is created by the speech recognizer [18]. The client runs Wireshark Version 1.12.4 to timestamp the traffic of voice transmission and reply packets from server [6]. Another application on the client time stamps the voice playback. All experiments were performed on a Windows 7 for GSR, TCP and UDP testbeds, and on iOS 7.0 for Siri. The netem box runs Fedora Linux operating system. After each round of streaming, the text editor compares the

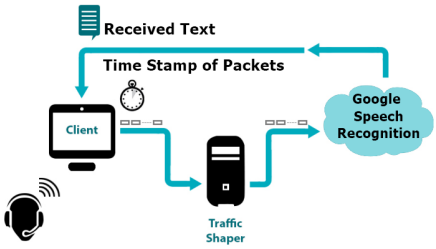


Figure 1: Experimental testbed for GSR.

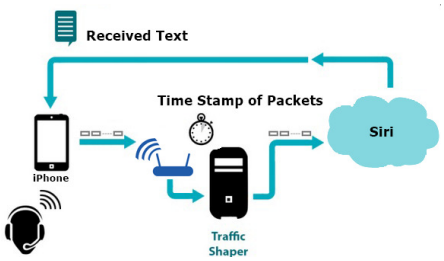


Figure 2: Experimental testbed for Siri.

translated string to the original we use and calculates a match percentage.

A. Experimental testbed for the GSR

We choose to use the GSR service that is embedded in Google Chrome. There is another alternative for using GSR. Google offers a web service for application developers. This service has limitations on length of voice and also number of queries a program can run during a certain time but the GSR service that is embedded in Google Chrome does not have these limitations. Figure 1 shows the architecture of our testbed. Client receives the voice and transmits voice packets to the Google server. This transmission goes through the netem box that manipulates network traffic performance. Google starts to translate voice packets as soon as it receives the first voice packet and sends recognized text back to the client. The client records the timestamp of each reply packet and also start and end time of the voice transmission to calculate transcription delay. The client also compares the returned text to the original one to calculate the accuracy of transmission. Levenshtein distance is used to calculate the match percentage of the original string and the string that is created by the GSR. Levenshtein distance is the algorithm we used for finding the match percentage of strings. Levenshtein distance between two words or strings is defined as the minimum number of insertions, deletions, or substitutions needed to convert a word or string to another one [18].

B. Experimental testbed for Apple Siri

We used the similar testbed configurations for Siri. The client is an iPhone connected to Internet through a WiFi router to a netem box. We used Wireshark to timestamp the transmission of voice packets and reception of replies from Siri server. Figure 2 shows this setup.

C. Experimental testbed for Nuance Dragon

We consider key characteristics of Siri and GSR to design a basic testbed that shows the same behavior. Siri and GSR both use TCP transport protocol [7], [11]. To replicate speech recognition algorithms we used Nuance technology which uses the same algorithms to convert the voice to the text as Siri [1], [3]. Nuance technology is available as Dragon Naturally Speaking software [2].

Our testbed consists of a client that is connected to a speech recognition server through netem. Client streams the voice over a TCP connection that goes through the netem box. Server starts to convert voice to the text as soon as it receives the first voice packet. The server sends the resulting text back to the client. We record accuracy of the returning text and the round trip time of the process to evaluate the performance of the system. We repeat the experience 30 times for each traffic setup. Figure 3 shows the architecture of the TCP streaming testbed.

The program timestamps when the voice playback starts and finishes. We call these timestamps fct and lct (stands for first client transfer and last client transfer), respectively. Network traffic conditions are controlled by netem. A program on the server is responsible to keep the timestamp of packets and store the first and last timestamp in a file. We call these timestamps fsr and lsr(stand for first server packet received and last server packet received), respectively. In order to timestamp the transcription delay, we developed a text editor to collect the Dragon's output and timestamp the time the first and last character created by Dragon. We call these timestamps ffr and lfr (stand for first text file character received and last text file character received), respectively. Every time a new character is created by dragon, our text editor sends that character to the sender and a program on the sender collects the received characters and stamp the time of the first and the last received character, we call these timestamps fcr, and lcr (stand for first client received and last client received), respectively. Figure 4 shows the data flow from the client to the server and also the data flow from Dragon's output to the client. This Figure also shows the relative order of the timestamp variables used for our evaluation.

The recorded timestamps for each round of experiment, we monitor the behavior of different parts of the testbed. (ffr - fsr) shows response time of the Dragon, (lfr - fsr) shows the total time of the speech recognition on the server, (lcr - fct) shows the total time of each round of experiment. We used (lcr - lct) as the delay of the remote speech recognition system.

D. Experimental testbed for UDP

TCP waits for each packet to be received and retransmits lost packets. Reliable transmission is not necessarily a good choice for real-time communications, in which transmission delay reduces the feeling of interactivity. UDP is a good alternative when the application tolerate moderate packet loss. We changed our TCP testbed to send UDP packets to observe the effect of packet loss and jitter on delay and accuracy of the speech recognition software. The UDP testbed has the same

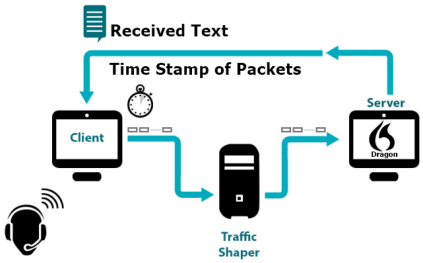


Figure 3: Experimental testbed for TCP.

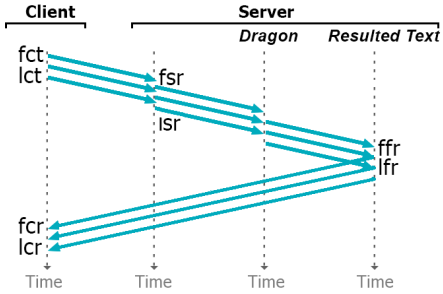


Figure 4: Timestamp Variables.

architecture as TCP, but streaming part of the testbed has been changed to UDP packets. We ran the UDP testbed with the same conditions as the TCP.

III. Evaluation

In this section we present results of experimental evaluation on each testbed. The metrics reported are transcription accuracy and delay under different values of packet loss and jitter. We ran our experiment under jitter in range of [0, 200] ms, and packet loss in range of [0, 5] percent and 30 times of experiment for each setting. We ran all experiments of each testbed under the same configuration. For measurement consistency, we used a recorded audio file 26.6 seconds long which belongs to a male with a North American accent. We also eliminated the effect of environmental noise by using virtual audio cables [4]. Here we analyze effect of our variables on each configuration.

A. Effect of packet loss

Figure 5 shows boxplots of the delay resulting by different values of packet loss in each setup. As we mentioned before GSR and Siri both use TCP. With the TCP connection sender retransmits lost packets and receiver waits to deliver packets after reordering. That means if we increase the packet loss rate, round trip time of the process increases due to retransmission delays. We observe this behavior in GSR, Siri, and TCP setups. There is a noticeable difference between Siri and two other TCP setups because of the different operation of Siri, as it does not start the translation before it receives the last voice packet. GSR and our TCP testbed, on the other hand, start to translate as soon as they receive the first voice packet. Generally we can say that delay of all three setups increases under packet loss. As we can see in Figure 5(d), UDP has relatively lower delay under packet loss in this measurement, because UDP does not tolerate lost packets.

Packet loss does not affect the accuracy of GSR and Siri and the accuracy was always 100% with all values of loss due to TCP retransmissions. We expect to get the same accuracy with our TCP testbed but, Figure 7(a) shows this is not the case with bigger loss values. As we mentioned before TCP does not lose any packet, but server waits to receive lost packets and this delay can affect on the speech recognizer output. This problem has been solved in GSR and Siri. Siri does not start to translate voice until it receives the last packet, and so packet loss and retransmission just increases the delay and without affecting accuracy. GSR has also solved this problem by full interaction between the speech recognizer and receiver.

Figure 7(b) shows that the accuracy decreases rapidly by increasing the packet loss in UDP, as we anticipated. Losing voice packets has direct impact on the quality of voice playback on receiver. As the result, we can say that packet loss affects the accuracy of UDP and delay of TCP connection.

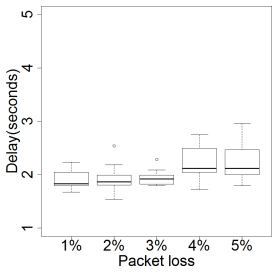
B. Effect of jitter

Jitter is a variation in the delay of arrived packets. This variation causes packets be received out of order. TCP connection does packet reordering at the receiver in this case. UDP, on the other hand delivers packets with the same order it receives them.

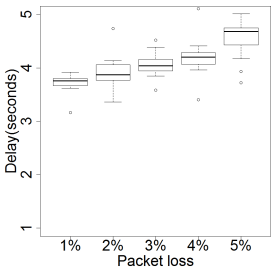
We ran our testbeds under different jitter values to see how does it affect the performance. Figure 6 shows boxplots of the effect of jitter on delay of each system. GSR and Siri show more stability with increasing jitter, but the overall delay value is higher compared to TCP and UDP testbeds. Figures 7(c) and 7(d) show that TCP accuracy does not change under jitter, until high values of jitter. Delay of transcription does not change on the average, based on results for TCP. Jitter increases the delay of UDP, because of the additional processing time of the speech recognizer with the missing audio data. Figure 7(c) shows the effect of jitter on the accuracy of TCP and UDP testbeds. Again, jitter does not affect on accuracy of GSR and Siri. As we can see, UDP performs poorly in this case. Jitter on UDP voice stream results noisy output and we can see this fact in the graph for UDP. With jitter of 200 milliseconds accuracy goes less than 50%. Jitter also affects TCP. We expect TCP performs better with jitter by packet reordering. The fact is that TCP delivers voice packets in order, but the time spent for packet reordering increases with higher amount of jitter and this delay affects the performance of the speech recognizer. This is why we see voice transmission in the TCP system is less accurate with jitter values higher than 140 milliseconds.

IV. Improving the Performance of the UDP Connection with Network Coding

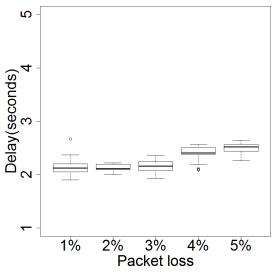
Our experiments show that using UDP can solve the problem of the transcription delay under packet loss, but packet loss and jitter affect the accuracy in this case. In order to increase the accuracy we propose to use network coding over UDP.



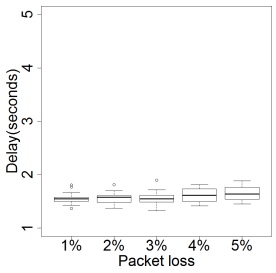
(a) Delay of GSR under packet loss.



(b) Delay of Siri under packet loss.

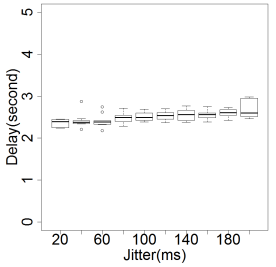


(c) Delay of Dragon with TCP under packet loss.

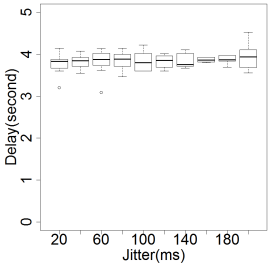


(d) Delay of Dragon with UDP under packet loss.

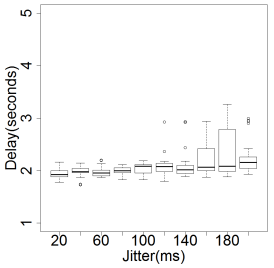
Figure 5: Effect of packet loss on round-trip delay



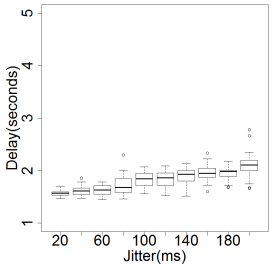
(a) Delay of GSR under jitter.



(b) Delay of Siri under jitter.

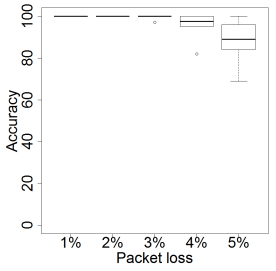


(c) Delay of Dragon with TCP under jitter.

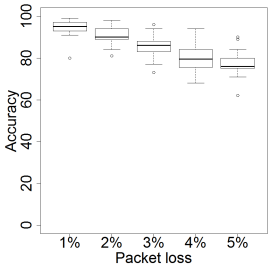


(d) Delay of Dragon with UDP under jitter.

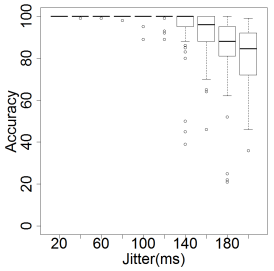
Figure 6: Effect of jitter on round-trip delay



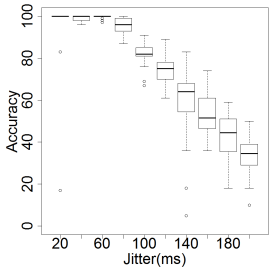
(a) Accuracy of Dragon with TCP under packet loss.



(b) Accuracy of Dragon with UDP under packet loss.



(c) Accuracy of Dragon with TCP under jitter.



(d) Accuracy of Dragon with UDP under jitter.

Figure 7: Accuracy of Dragon on TCP and UDP

A. Fountain Codes

Fountain codes are used in erasure channels such as the Internet. Channels with erasure transmit files in multiple small packets and each packet is either received without error or is lost [14]. Coded packets sent to the receiver are combinations of original packets. Once receiver receives enough coded packets it is able to decode and extract the original packets. Figure 8 illustrates the mechanism behind the fountain codec that is used in our solution [16]. Sender takes a group of packets, creates a number of coded packets, and send them to receiver along with information needed for their decoding. Receiver extract the original packets after receiving enough coded packets by solving a linear equation created by the received information.

B. Fountain Encoder

The Fountain encoder generates unlimited number of encoded packets using original ones. In order to decode packets of a stream , we group every X consecutive original packets together. Fountain encoder generates enough number of coded packets using original packets of group, we will find this number later in this section. Each encoded packet is a bitwise sum of packets of group:

$$EP_n = \sum_{x=1}^{X} P_x G_{xn}, \tag{1}$$

where G_{xn} is a random binary number consisting of X bits and P's are original packets. The sum operation is done by XOR-ing packets. The resulting packet is sent to the receiver and G_{xn} is also put in the header for decoder to be able to extract original packets after receiving enough number of coded packets. Figure 9 demonstrates the process of coding

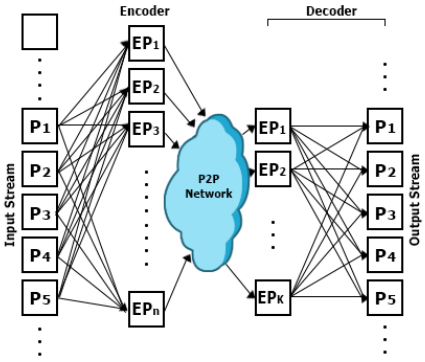


Figure 8: Coding and sending packets over a lossy network

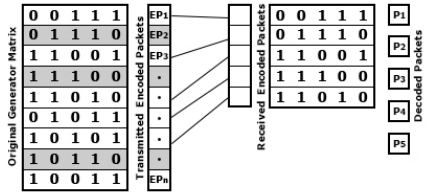


Figure 9: The generator matrix of the linear code

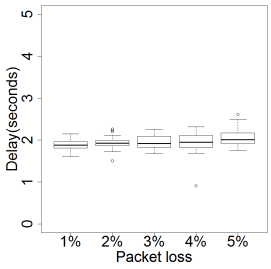
and sending packets over a lossy network. Grey shaded packets are not received. Sender creates and sends n coded packets from each group. In order to have enough information to extract the original packets, n should be greater than X. The number of coded packets required to be received by receiver to have probability 1-δ of decoding success is $\approx$ X+log_2 1/δ [14].

C. Fountain Decoder

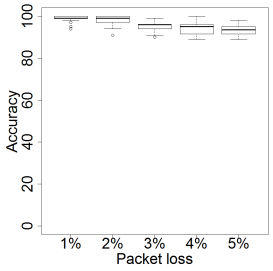
With enough number of received packets, receiver is able to extract original packets. Lets say there are X original packets and receiver has received K packets. The binary numbers that we used in our encoder make a K-by-X matrix. If K<X, the decoder does not have enough information to extract the original packets. If k=X, it is possible to recover packets. If the resulted K-by-K matrix is invertible, the decoder is able to calculate the inverse of G^{-1} by Gaussian elimination and recover

$$t_x = \sum_{k=1}^{K} t_k G_{kx}^{-1}. \quad (2)$$

The probability that a random K-by-K matrix is invertible is 0.289 for any K greater than 10 [14]. Decoder should receive extra packets to increase the probability of having an inversible matrix. The time complexity of encoding and decoding of linear Fountain codes are quadratic and cubic in number of encoded packets but this is not important when working with packets less than thousand [14]. Using faster versions of fountain codes, like the LT code or Raptor codes offers less complexity[10].

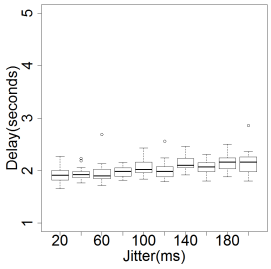


(a) Delay of Dragon with Fountain codec under packet loss.

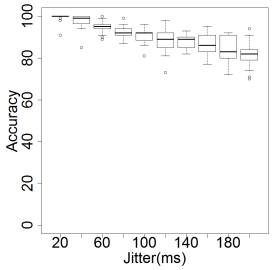


(b) Accuracy of Dragon with Fountain codec under packet loss.

Figure 10: Effect of packet loss on the network coding streaming.



(a) Delay of Dragon with Fountain codec under jitter.



(b) Accuracy of Dragon with Fountain codec under jitter.

Figure 11: Effect of jitter on the network coding streaming

D. Evaluation of Network Coding Over UDP

We implemented P2P streaming system using linear fountain and replaced it with the standard UDP stream, other parts of the testbed are the same.

Results show that by using the linear fountain increasing the packet loss does not affect the accuracy and delay comparing to our experimental testbeds. Figure 10(a) shows the resulting delay under the packet loss. We removed the effect of retransmission delay on the accuracy and delay by using network coding. This property makes network coding on UDP transport a very good alternative for streaming over lossy networks. Accuracy of our soloution is also high. Figure 10(b) shows that the accuracy is 93% at the worst case scenario, i.e. packet loss of the 5%. Comparing these graphs with TCP, we can see using network coding with UDP connection on a lossy network improves the delay and also keeps the accuracy in an acceptable range. We also ran our proposed system under different jitter values. Figure 11(a) shows improvement of delay by about 16% comparing to TCP. Figure 11(b) shows the accuracy of result under different jitter values. Our proposed solution is not perfect with jitter in this case but as we can see in Figure 11(a) delay is much lower compared to TCP, GSR, and Siri. So we can still argue that Fountain codec voice transmission improves accuracy and delay under jitter.

V. Related Work

In this paper, we focused on effect of the network traffic conditions on performance of cloud speech recognition.

An experimental evaluation of rate adaption algorithms streaming over HTTP has been done by Akhshabi *et al* [5]. They evaluated three popular video streaming applications under different bandwidth values. They argue that TCP's congestion control and reliability requirement does not necessarily hurt the performance of streaming. Understanding the interaction between rate-adaption logic and TCP congestion control is left as future work.

A measurement-based study of multipath TCP performance over wireless networks has been done by Chen *et al* [8]. They have measured the latency over different cellular data providers and shown that MPTCP offers a data transport that is robust under various traffic conditions. Although, studying on energy cost and performance trade-off has not been considered.

An study of Skype's voice rate adaption under different network conditions is also done by Te-Yuan Huang *et al*. [13]. They also argue that Skype's technique of using public domain codecs is not ideal for user satisfaction. Based on the results of their experiments on quality of users' experience under different levels of packet loss, they propose a model for redundancy control based on the level and burstiness of packet loss.

Cicco *et al*. have done an experimental investigation on the Google Congestion Control (GCC) proposed in the RTCWeb IETF WG [9]. They set up a controlled testbed for their evaluation. Results of this study shows that the proposed algorithm works as expected but it does not provide a fair bandwidth utilization when bandwidth is shared by GCC flow and another GCC or a TCP flow.

Oh *et al*. demonstrate a mesh-pull-based P2P video streaming by using Fountain codes [15]. The proposed system provides fast and smooth streaming with minimum computational complexity. Evaluations on this system shows that the proposed system outperforms existing buffer-map-based video streaming systems under different packet loss values. Studying on the behavior of this system under different jitter values is missing.

Smith *et al*. have focused on the practical utility of using Fountain Multiple Description Coding (MDC) in video streaming over a heterogeneous P2P network. They show that although using Fountain MDC codes is favorable in this case, benefits are restricted in real P2P streaming systems.

A novel multicast streaming system based on Expanding Window Fountain (EWF) codes for real-time multicast is proposed by Vukobratovic *et al* [17].

VI. Conclusion

We showed that using fountain codes with UDP connection is a good alternative for cloud speech recognition systems and using this configuration has a considerable effect on accuracy and delay under packet loss and it also improves the delay under jitter.

TCP connection is currently used by cloud speech recognition applications to provide the maximum accuracy. Our experimental evaluation shows that these applications offer the accurate result in cost of delay. Quality of users' experience can be improved by applying network coding on UDP connection. Using this novel solution applications improve the delay while maintaining the accuracy. Future cloud speech recognition systems can provide more interactivity by improving the delay using this technique.

We did not consider the effect of available bandwidth and transmission delay on the performance of cloud speech recognition. Cellular network configuration is not considered in our study and we used WiFi connection to evaluate Apple Siri. Evaluation of Siri under cellular connection should be considered in future studies. Implementing more sophisticated streaming systems using advanced rateless codes and comparing the overhead of the solution with streaming systems that use packet loss classification (PLC) algorithm and Forward Error Correction (FEC), and also considering the effect of available bandwidth and transmission delay are potential extensions of this research.

References

[1] *Nuance Communications*, 2014. http://www.nuance.com/news/pressreleases/2009/20091005_ecopy.asp.

[2] *Nuance Technologies*, Dec. 2014. http://research.nuance.com/category/speech-recognition/.

[3] *Siri*, Dec. 2014. https://support.apple.com/en-us/ht4992.

[4] *Virtual Audio Cable*, Dec. 2014. http://www.ntonyx.com/vac.htm.

[5] Saamer Akhshabi, Ali C Begen, and Constantine Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *ACM conference on Multimedia systems*, Feb. 2011.

[6] Jay Beale Angela Orebaugh, Gilbert Ramirez and Joshua Wright. Wireshark and ethereal network protocol analyzer toolkit. *Syngress Media Inc*, 2007.

[7] Apple. *iOS: Multipath TCP Support in iOS 7*, 2014. http://engineering.purdue.edu/~mark/puthesis.

[8] Yung-Chih Chen, Yeon-sup Lim, Richard J Gibbens, Erich M Nahum, Ramin Khalili, and Don Towsley. A measurement-based study of multipath tcp performance over wireless networks. In *ACM IMC*, Oct. 2013.

[9] Luca De Cicco, Gaetano Carlucci, and Saverio Mascolo. Experimental investigation of the google congestion control for real-time flows. In *SIGCOMM workshop on Future human-centric multimedia networking*, Aug. 2013.

[10] M Eittenberger, Todor Mladenov, and Udo R Krieger. Raptor codes for p2p streaming. In *Parallel, Distributed and Network-Based Processing (PDP)*, Feb. 2012.

[11] Google. *Performing speech recognition over a network and using speech recognition results*, Dec. 2014. http://www.google.com/patents/US8335687.

[12] Junxian Huang, Qiang Xu, Birjodh Tiwana, Z Morley Mao, Ming Zhang, and Paramvir Bahl. Anatomizing application performance differences on smartphones. In *ACM Mobile systems, applications, and services*, Jun. 2010.

[13] Te-Yuan Huang, Kuan-Ta Chen, and Polly Huang. Tuning skype's redundancy control algorithm for user satisfaction. In *INFOCOM, Apr. 2009*.

[14] David JC MacKay. Fountain codes. *IEE Proceedings-Communications*, 152(6):1062–1068, Dec. 2005.

[15] Hyung Rai Oh and Hwangjun Song. Mesh-pull-based p2p video streaming system using fountain codes. In *Computer Communications and Networks (ICCCN)*, Jul. 2011.

[16] Guillaume Smith, P Tournoux, Roksana Boreli, Jérôme Lacan, and Emmanuel Lochin. On the limit of fountain mdc codes for video peer-to-peer networks. In *World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, Jun. 2012.

[17] Dejan Vukobratovic, Vladimir Stankovic, Dino Sejdinovic, Lina Stankovic, and Zixiang Xiong. Scalable video multicast using expanding window fountain codes. *IEEE Transactions on Multimedia*, 11(6):1094–1104, Oct. 2009.

[18] Li Yujian and Liu Bo. A normalized levenshtein distance metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(6):1091–1095, Jun. 2007.