Measuring the Impact of Network Performance on Cloud-Based Speech Recognition

An Empirical Study of Apple Siri and Google Speech Recognition

Mehdi Assefi*, Guangchi Liu†, Mike P. Wittie‡, Clemente Izurieta§
Montana State University, Bozeman, MT 59715, USA

Abstract

Cloud-based speech recognition systems enhance Web surfing, transportation, health care, etc. For example, using voice commands helps drivers search the Internet without affecting traffic safety risks. User frustration with network traffic problems can affect the usability of these applications. The performance of these type of applications should be robust in difficult network conditions. We evaluate the performance of several client-server speech recognition applications, under various network conditions. We measure transcription delay and accuracy of each application under different packet loss and jitter values. Results of our study show that performance of client-server speech recognition systems is affected by jitter and packet loss, which commonly occur in WiFi and cellular networks.

An experimental study on client-server speech recognition applications is reported in *Impact of the network performance on cloud-based speech recognition systems*, in which a solution that uses network coding to improve the performance of cloud-based speech recognition applications has been proposed. The aforementioned paper is published in ICCCN 2015 [8]. Designing and implementing of experimental testbeds by using TCP and UDP connections and also designing and implementing another testbed that uses fountain codes on UDP connection has been introduced in the paper. In this paper, we design and implement an extensive experimental evaluation of five client-server speech recognition applications to compare the performance of these applications under different network conditions.

Key Words: Cloud Speech Recognition, Quality of Experience, Software Measurement, Streaming Media, Real-time Systems.

1 Introduction

Performance evaluation of cloud-based speech recognition systems under different network conditions has received much less attention than other streaming systems. Although Apple Siri and Google Speech Recognition (GSR) are popular applications that help users to interact with search engines using voice commands, an experimental evaluation of these applications is noticeably missing.

Delay and accuracy of the voice recognition process is an important parameter that affects the quality of a user's experience with cloud-based speech recognition applications. Streaming voice from the client to the server and converting it to text are two phases of this process and should have the low delay and high accuracy in order to satisfy the quality of a user's experience. Delays of this process should also be consistent under all different network conditions.

In this paper, we describe the design and implementation of an experimental evaluation of Siri and GSR. We also evaluate three client-server speech recocgintion systems using TCP, UDP, and network coding over UDP. We evaluate these applications under different packet loss and jitter values and measure the delay and accuracy of each under different network conditions. Specifically, we employ four statistical models to evaluate the effects of packet loss and jitter, respectively. Each model is designed to evaluate two factors (jitter and packet loss) with two blocking variables on the response variable - delay and accuracy. The blocking variable is the application for all experiments. The ANOVA test is used to evaluate effects of packet loss and jitter for each experiment respectively. Results of our study show that delays in all applications are affected by packet loss and jitter. Results also show that the accuracy of three applications is affected by packet loss and jitter.

The remainder of this paper is organized as follows: In Section II we explore related work. In Section III we describe our experimental methods. In Section IV we describe overall results. In Section V we describe our experimental design and the mathematical model used to analyze experimental data. Section VI discusses results. Finally, in Section VII we discusses threats to validity of our experiment and conclude in Section VIII.

2 Related Work

Yang Xu *et al.* performed a measurement study on Google+, iChat, and Skype [31]. They explored the architectural features

*Department of Computer Science, Email: mehdi.assefi@msu.montana.edu.
†Department of Computer Science, Email: guangchi.liu@msu.montana.edu.
‡Department of Computer Science, Email: mwittie@montana.edu.
§Department of Computer Science, Email: clemente.izurieta@montana.edu.

of these applications. Using passive and active experiments, the authors unveiled some performance details of these applications such as video generation and adaption techniques, packet loss recovery solutions, and end-to-end delays. Based on their experiments the server location had a significant impact on user performance and also loss recovery in server-based applications. They also argued that using batched re-transmissions was a good alternative for real time applications instead of using Forward Error Correction (FEC) –an error control technique in streaming over unreliable network connections.

Te-Yuan Huang *et al.* did a measurement study on the performance of Skype's FEC mechanism [21]. They studied the amount of the redundancy added by the FEC mechanism and the trade-offs between the quality of the users' experience and also the resulting redundancy due to FEC. They tried to find an optimal level of redundancy to achieve the maximum quality of the users' experience.

Te-Yuan Huang *et al.* also performed a study on voice rate adaption of Skype under different network conditions [20]. Results of this study showed that using public domain codecs was not the ideal choice for users' satisfaction. In that study, they considered different levels of packet loss in their experiments and created a model to control the redundancy under different packet loss conditions.

Kuan-Ta Chen *et al.* proposed a framework for user QoE measurement [11]. Their proposed framework, OneClick, provided a dedicated key that could be pressed by users whenever they felt unsatisfied by the network conditions with streaming media. OneClick was implemented on two applications – instant messaging applications and shooter games.

Another framework that quantified the quality of a user's experience was proposed by Kuan-Ta Chen *et al* [12]. The proposed system was able to verify participants' inputs, so it supported crowd-sourcing. Participation is made easy in this framework. The framework generates interval-scale scores. They argue that researchers can use this framework for measuring the quality of a users' experience without affecting quality of the results and achieve a higher level of diversity in users' participation while also keeping cost low.

Budzisz *et al.* proposed and developed a delayed-based congestion control system [10]. The proposed system offers low standing queues and delay in homogeneous networks, and balanced delay-based and loss-based flows in heterogeneous networks. They argue that this system can achieve these properties under different loss values, and outperform TCP flows. Using experiments and analysis, they demonstrate that this system guarantees aforementioned properties.

Hayes *et al.* proposed an algorithm which tolerates non-congestion related packet loss [18]. They proved experimentally that the proposed algorithm improves the throughput by 150% under packet loss of 1% and improves the ability to share the capacity by more than 50%.

Akhshabi *et al.* proposed an experimental evaluation of rate adaption algorithms for streaming over HTTP [4, 5]. They experimentally evaluated three common video streaming applications under a range of bandwidth values. Results of this study showed that congestion control of TCP and its reliability requirement does not necessarily affect the performance of such streaming applications. Interaction of rate-adaption logic and TCP congestion control is left as an open research problem.

Chen *et al.* experimentally studied performance of multipath TCP over wireless networks [13]. They measured the latency resulting from different cellular data providers. Results of this study show that Multipath TCP offers a robust data transport under various network traffic conditions. Studying the energy costs and performance trade-offs should be considered as a possible extension of this study.

Google is currently working on a new transport protocol for the Internet called QUIC (Quick UDP Internet Connections) [25]. QUIC uses UDP and solves problems of packet delay under different packet loss values in TCP connections. QUIC solves this problem by multiplexing and FEC.

An experimental investigation on the Google Congestion Control (GCC) in the RTCWeb IETF WG was performed by Cicco *et al.* [14]. They implemented a controlled testbed for their experiment. Results of this experimental study show that the proposed algorithm works well but it does not utilize the bandwidth fairly when it is shared by two GCC flows or a GCC and a TCP flow.

Cicco *et al.* have also experimentally investigated the High Definition (HD) video distribution of Akamai [15]. They explained details of Akamai's client-server protocol which implements the quality adaption algorithm. Their study shows that the proposed technique encodes any video at five different bit rates and stores all of them at the server. The server selects the bit rate that matches the bandwidth that is measured based on the signal received from the cilent. The bitrate level adaptively changes based on the available bandwidth. Authors of the paper also evaluated the dynamics of the algorithm in three scenarios.

Winkler *et al.* ran a set of experiments to asses quality of experience on television and mobile applications [28, 29]. Their proposed subjective experiment considers different bitrates, contents, codec, and network traffic conditions. Authors of the paper used Single Stimulus Continous Quality Evaluation (SSCQE) and Double Stimulus Impairment Scale (DSIS) on the same set of materials and compared these methods and analyzed results of experiments in view of codec performance.

A mesh-pull-based P2P video streaming using Fountain codes is proposed by Oh *et al.* [24]. The proposed system offers fast and smooth streaming with low complexity. Experimental evaluations show that the proposed system has better performance than existing buffer-map-based video streaming systems under packet loss values. Considering jitter as another important factor and evaluation of behavior of the proposed system considering jitter values can be a potential extension of this study.

Application of Fountain Multiple Description Coding (MDC) in video streaming over a heterogeneous peer to peer network is

considered by Smith *et al.* [26]. They conclude that Fountain MDC codes are favorable in such cases, but there are some restrictions in real-world P2P streaming systems.

Finally, Vukobratovic *et al.* proposed a novel multicast streaming system that is based on Expanding Window Fountain (EWF) codes for real-time multicast [27]. Using Raptor-like precoding has been addressed as a potential improvement in this area.

3 Experimental Testbeds

We design and implement our experimental testbed to study the performance of cloud-based speech recognition systems under loss and jitter. Clients of such systems transmit voice data through a network traffic shaper, where we change jitter and packet loss values in the communication network. We set a bandwidth to 2Mbps which is typical on 3G connections [19]. The server receives voice data, translates the voice into text, and sends the text and search results based on the converted text to the client. The client calculates the delay of the server response. To calculate the accuracy of transcription we use Levenshtein distance [32]. Accuracy is measured as the match percentage of the original string used to generate the voice and the resulting transcription. The client uses Wireshark Version 1.12.4 to timestamp the traffic of voice transmission to and from the server [6]. We developed a Windows application using Visual C# to timestamp the voice playback. All experiments are performed on a Windows 7 platform for GSR, and on iOS 7.0 for Siri. The traffic shaper is a `netem` box which runs the Fedora Linux operating system. We ran our experiment 30 times for each value of loss and jitter and for each cloud speech recognizer.

3.1 Experimental Testbed for GSR

We use the GSR service available in Google Chrome. There is also another alternative for using Google voice recognition. Google offers a voice recognition Web service that can be used in Windows applications. Figure 1 shows the architecture of our experimental setup.

Clients transmit voice packets to the Google server through the `netem` box that changes network traffic performance. We used a recorded voice with a length of 26.4 seconds for all experiments in order to have a consistent measurement. Google starts to recognize voice as soon as it receives the first voice packet, and sends converted text back to the client. The client records the time of each packet and also voice transmission time to calculate the transcription time of the experiment. The client also compares the resulting text to the original string which was used to generate the voice command and calculates transmission accuracy using the Levenshtein distance [32].

3.2 Experimental Testbed for Siri

The experimental setup for Siri is similar to GSR. We use an iPhone as the client. A client is connected to the Internet through a WiFi router then to a `netem` box. Here we also used Wireshark to timestamp the transmission of voice packets and reception of results from the Siri server. Figure 2 depicts this setup.

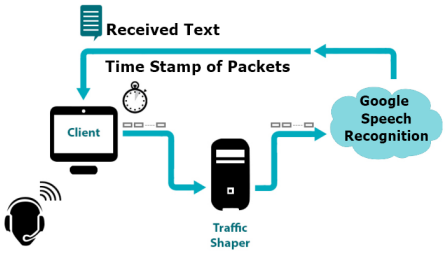


Figure 1: Experimental testbed for GSR.

3.3 Experimental Testbed for Nuance Dragon

We consider key characteristics of Siri and GSR to design an in-lab testbed that shows the same behavior. Siri and GSR both use TCP transport protocol [7, 17]. 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. The 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 the 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 (first client transfer and last client transfer), respectively. Network traffic conditions are controlled by `netem`. A logger on the server is responsible for keeping the timestamp of packets and storing the first and last timestamps in a file. We call these timestamps fsr and lsr (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 when the first and last character was created by Dragon. We call these timestamps ffr and lfr (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 stamps the time of the first and the last received character. We call these timestamps fcr, and lcr (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.

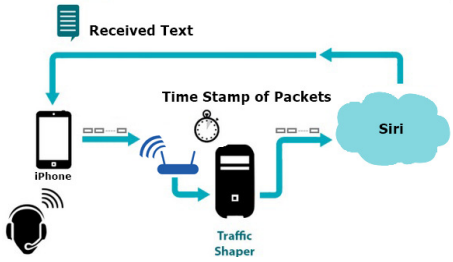


Figure 2: Experimental testbed for Siri.

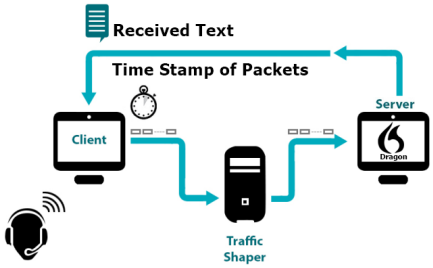


Figure 3: Experimental testbed for TCP.

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

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

3.5 Experimental Testbed for UDP with Network Coding

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

3.5.1 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 [23]. Coded packets sent to the receiver are combinations

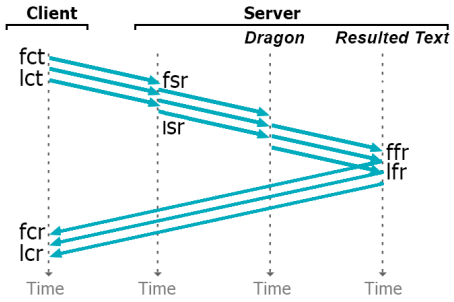


Figure 4: Timestamp Variables.

of original packets. Once the receiver receives enough coded packets it is able to decode and extract the original packets. Figure 5 illustrates the mechanism behind the fountain codec that is used in our solution [26]. Sender takes a group of packets, creates a number of coded packets, and sends them to the receiver along with information needed for their decoding. The receiver extracts the original packets after receiving enough coded packets by solving a linear equation created by the received information.

3.5.2 Fountain Encoder

The Fountain encoder generates an 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 the group, and we will find this number later in this section. Each encoded packet is a bit-wise sum of packets of group:

$$EP_n = \sum_{x=1}^{X} P_x G_{xn}, \quad (1)$$

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 the decoder to be able to extract original packets after receiving enough number of coded packets. Figure 6 demonstrates the process of coding and sending packets over a lossy network. Grey shaded packets are not received. The 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 the receiver to have probability 1-δ of decoding success is $\approx$ X+log_2 1/δ [23].

3.5.3 Fountain Decoder

With enough number of received packets, the receiver is able to extract original packets. Lets say there are X original packets and the 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

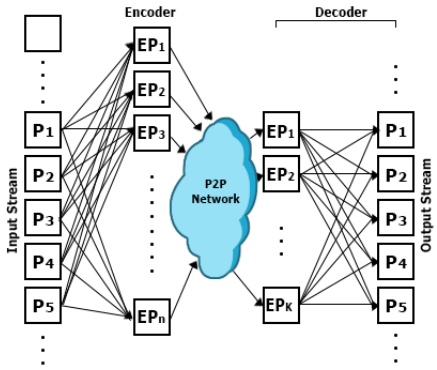


Figure 5: Coding and sending packets over a lossy network [8].

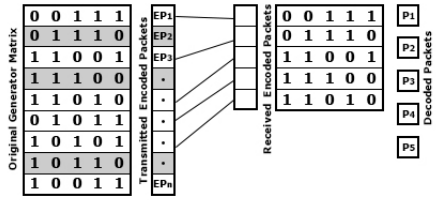


Figure 6: The generator matrix of the linear code [8].

packets. If k=X, it is possible to recover packets. If the resulting 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 [23]. The 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 a thousand [23]. Using faster versions of fountain codes, like the LT code or Raptor codes offers less complexity [16].

4 Overall Results

To investigate the effect of packet loss and jitter on delay and accuracy, we generate packet loss from 1% to 5% and jitter from 20 ms to 200 ms respectively on our testbeds and observe the resulting accuracy and delay. Siri and GSR both keep 100% accuracy under high values of packet loss and jitter, so we just consider accuracy values for the other three testbeds. The effect of packet loss and jitter on roundtrip delay of applications is shown in Figures 7 to 16, where the y axis displays delay(s), and the x axis displays packet loss (percentile). and jitter (ms), respectively. There are increasing trends as packet loss and jitter increases, for all applications. For GSR, an increase of 1 packet loss unit (percentile), leads to delay increases in the range of 0-100 ms. An increase of 1 unit (20 ms) in jitter leads to increases in delay from 0-100 ms. In addition, the variance of delay also increases as packet loss and jitter increase, indicating a trend of instability. For Siri, the increase in 1 unit (percentile) packet loss leads to increases in delay of 200 ms; which is worse than GSR. On the other hand, jitter has less impact on delay. In addition, the variance of delay is unchanged, compared to GSR. For Dragon under TCP, packet loss and jitter both affect the roundtrip delay. Variance of delay increases as jitter increases, indicating a trend of instability in the case of high values of jitter. For Dragon under UDP, packet loss does not affect the roundtrip delay, but variance of delay increases as we increase the packet loss. Jitter, on the other hand, affects the roundtrip delay of Dragon testbed under UDP. Variance of delay also increases with increasing jitter. Figures 19 and 22 show that using the network coding with UDP improves the accuracy of UDP when packet loss increases. Comparing the results from Figure 18, we can say that the accuracy of Dragon under UDP with using Fountain codes has been improved by about 30% in the existence of high values of packet loss. Comparing the results from Figures 22 and 20 shows that the accuracy of Dragon under UDP improves under different values of jitter, if we apply Fountain coding. Results show that the accuracy of Dragon has been improved to 85% with 200ms jitter, and this value is 30% when we do not use Fountain coding. From Figures 17 and 20, we can see the effect of increasing packet loss and jitter on the accuracy of Dragon under TCP. Accuracy of Dragon under TCP decreases by 15% when jitter is 200ms. Figure 20 also shows that the accuracy does not change when jitter is between 0 to 100ms and after this point, system starts to lose the accuracy. Variances of accuracy also start to increase from this point.

5 Experiment Design

5.1 Model

Since our data is collected by varying jitter and packet loss respectively, we designed four statistical models to assess the effect of jitter and packet loss on delay and accuracy, respectively. Also, since our data is collected from five applications (i.e. Siri, GSR, fontain, TCP and UDP), we take the application as a blocking variable. Each model contains one factory and one blocking variable. For the first model, the response variable is delay, the independent variable is jitter and the blocking variable is application. Also, to guarantee the assumptions still hold for the following ANOVA tests, we apply log transformation on the response variable. Hence, the first model can be expressed as:

$$log(y_{dij}) = \mu + \alpha_i + \beta_j + e_{ij} \quad (3)$$

where α is jitter, β is application.

For the second model, the response variable is delay, the independent variable is packet loss and the blocking variable is application. The model can be expressed as:

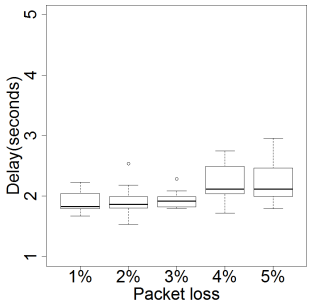


Figure 7: Impact of packet loss on delay of GSR

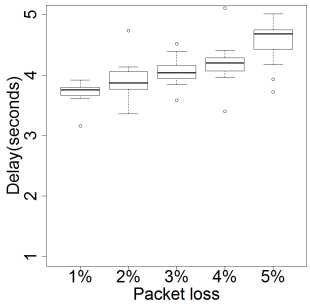


Figure 8: Impact of packet loss on delay of Siri

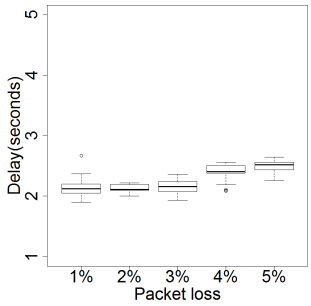


Figure 9: Impact of packet loss on delay of Dragon with TCP

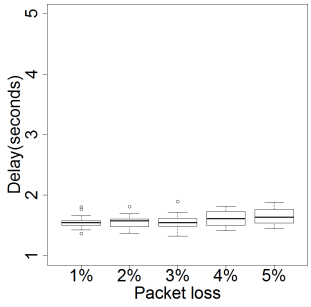


Figure 10: Impact of packet loss on delay of Dragon with UDP

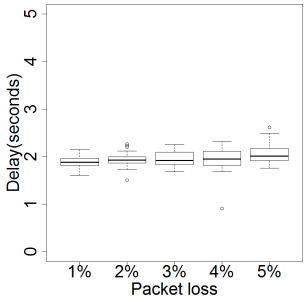


Figure 11: Impact of packet loss on delay of Dragon with UDP by using network coding

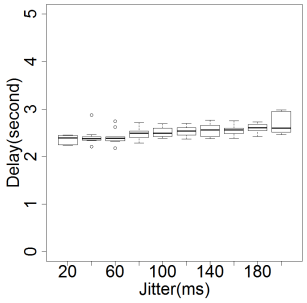


Figure 12: Impact of jitter on delay of GSR

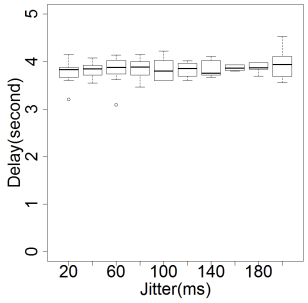


Figure 13: Impact of jitter on delay of Siri

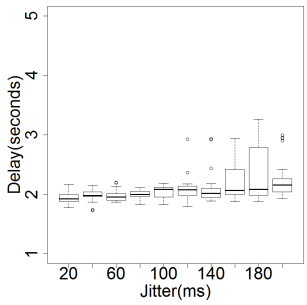


Figure 14: Impact of jitter on delay of Dragon with TCP

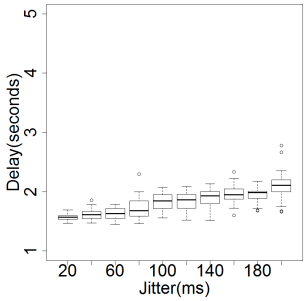


Figure 15: Impact of jitter on delay of Dragon with UDP

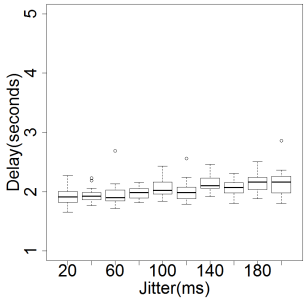


Figure 16: Impact of jitter on delay of Dragon with UDP by using network coding

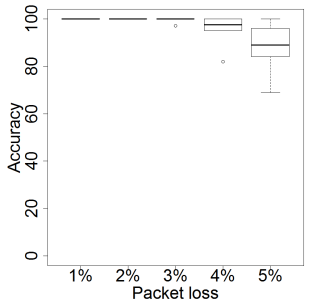


Figure 17: Impact of packet loss on accuracy of Dragon with TCP

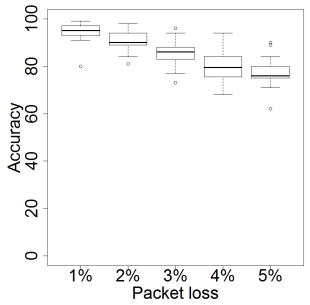


Figure 18: Impact of packet loss on accuracy of Dragon with UDP

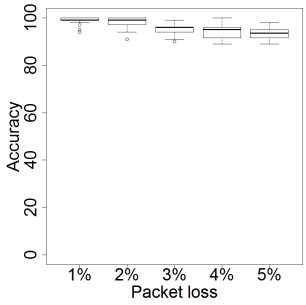


Figure 19: Impact of packet loss on accuracy of Dragon with UDP by using network coding

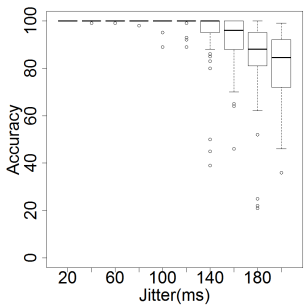


Figure 20: Impact of jitter on accuracy of Dragon with TCP

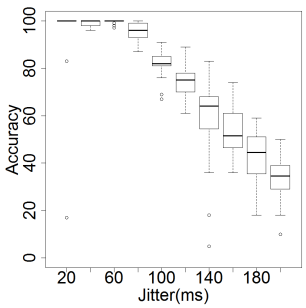


Figure 21: Impact of jitter on accuracy of Dragon with UDP

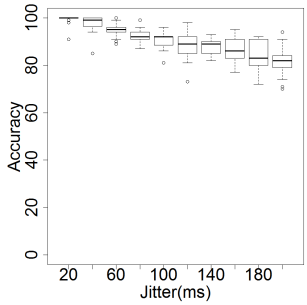


Figure 22: Impact of jitter on accuracy of Dragon with UDP by using network coding

$$log(y_{dij}) = \mu + \gamma_i + \beta_j + e_{ij} \quad (4)$$

where γ is packet loss, β is application.

For the third model, the response variable is accuracy, the independent variable is jitter and the blocking variable is application. The model can be expressed as:

$$log(y_{aij}) = \mu + \alpha_i + \beta_j + e_{ij} \quad (5)$$

where α is jitter, β is application.

For the fourth model, the response variable is accuracy, the independent variable is packet loss and the blocking variable is application. The model can be expressed as:

$$log(y_{aij}) = \mu + \gamma_i + \beta_j + e_{ij} \quad (6)$$

where γ is packet loss, β is application.

For model 3, the factor (jitter) has 10 alternatives, which are the jitter duration ranging from 20 to 200 ms. For model 4, the factor (packet loss) has 5 alternatives, which are the proportion of lost packet ranging from 1% to 5%. The blocking variable for both models 4 and 3 have 5 alternatives, which are SiRi, GSR, fontain, TCP and UDP, respectively. For model 5, the factor (jitter) has 10 alternatives, which are the jitter duration ranging from 20 to 200 ms. For model 6, the factor (packet loss) has 5 alternatives, which are the proportion of lost packet ranging from 1% to 5%. The blocking variable for both models 6 and 5, however, have only three alternatives, which are fontain, TCP and UDP, respectively. The reason is that for SiRi and GSR, the accuracy is always 100%, no matter how the factor changes. Therefore, we ignore them for accuracy.

5.2 Assumptions

To guarantee the effectiveness of ANOVA test, some assumptions should be checked before conducting the ANOVA tests. In this paper, the interaction of dependent variables, the normality of errors and the constant variance of errors are tested. All of these assumptions hold for an effective ANOVA test on the collected data and models (Eq. 3, 4, 5, and 6). Hence, we can conduct ANOVA tests, whose result will be shown in the next section.

Table 1: Statistical Findings of Effect on Delay: Jitter and Packet Loss

Jitter	**Df**	**Sum Sq**	**Mean Sq**	**F value**	**Pr(<F)**
Jitter	9	2.30	0.255	29.97	<2e-16
App	4	49.02	12.255	1437.63	<2e-16
Residuals	905	7.71	0.009	–	–
Packet Loss	**Df**	**Sum Sq**	**Mean Sq**	**F value**	**Pr(<F)**
Packet Loss	4	1.87	0.467	60.17	<2e-16
App	4	40.04	10.009	1290.49	<2e-16
Residuals	535	4.15	0.008	–	–

Table 2: Statistical Findings of Effect on Accuracy: Jitter and Packet Loss

Jitter	**Df**	**Sum Sq**	**Mean Sq**	**F value**	**Pr(<F)**
Jitter	9	29.16	3.24	62.84	<2e-16
App	2	23.94	11.972	232.21	<2e-16
Residuals	920	47.42	0.052	–	–
Packet Loss	**Df**	**Sum Sq**	**Mean Sq**	**F value**	**Pr(<F)**
Packet Loss	4	0.7102	0.1775	53.52	<2e-16
App	2	1.0846	0.5423	163.49	<2e-16
Residuals	299	0.9918	0.0033	–	–

6 ANOVA: Results and Conclusions

As can be seen in Table 1 there is conclusive evidence that delay is affected by both jitter and packet loss. More specifically, the f-values of jitter and application are 29.97 (df. = 9, p-value = 2e-16) and 1437.63 (df. = 4, p-value = 2e-16) while the f-values of packet loss and application are 60.17 (df. = 4, p-value = 2e-16) and 1290.47 (df. = 4, p-value = 2e-16),

respectively. The underlying reasons are as follows. As we mentioned before, jitter causes packets to arrive out of order and TCP needs to reorder packets before delivering them to the application layer. TCP also re-transmits lost packets. Both packet loss and jitter reduce the voice stream quality and this affects the performance of the speech recognition. On the other hand, the application affects the delay more seriously. The f-values of application for jitter and packet loss are 1437.63 and 1290.47, respectively. In other words, Siri causes much more delay than GSR. This is because Siri generates accurate transcription by starting the speech recognition process just after receiving the whole voice. That means Siri needs to receive the whole stream before starting to generate the text. That increases the delay in processing the whole text and accounts for the majority of total delay. GSR, on the other hand, keeps the result accurate by interaction between the transport and application layers and so it offers less delay even under high values of packet loss and jitter, compared to Siri.

As can be seen in Table 2, there is conclusive evidence that accuracy is affected by both jitter (p-value = 2e-16, f-value =62.84 on 9 df.) and packet loss (p-value = 2e-16 , f-value =53.52 on 4 df.). More specifically, the f-values of jitter and application are 62.84 (df. = 9, p-value = 2e-16) and 232.21 (df. = 4, p-value = 2e-16) while the f-values of packet loss and application are 53.52 (df. = 4, p-value = 2e-16) and 163.49 (df. = 4, p-value = 2e-16), respectively. Interestingly, for accuracy, the impact of jitter and packet loss begin is greater than that for delay. However, their impacts are still less than application.

7 Threats to Validity

7.1 Conclusion Validity

Conclusion validity makes sure that there is a statistical relationship between the experiment and results, with a given significance [30]. A perfect experiment would be conducted in randomly selected locations around the world and using randomly selected Internet providers. The experiment should repeat many times in each location. Our testbeds for Apple Siri and Google Speech Recognition were limited to a campus network, thus limiting the statistical strength of the results. Selecting the location and Internet provider randomly, as well as increasing the number of sites can increase the conclusion validity.

7.2 Internal Validity

Internal Validity refers to the causal effects between independent and dependent variables, and for any relationship to exist, we should make sure that it is not as a result of a factor that there is no control over or that it has not been measured [30]. One of the possible threats to internal validity is the hardware limitations of the devices running GSR and Siri. More specifically, the processing speed of memory and CPU will affect the processing of data streams in a PC. Another possible threat is the status of the PC. For example, when the OS is busy, it does not have enough time to respond to the interruptions generated from GSR or Siri, hence generating and thus affecting delay.

7.3 Construct Validity

Construct validity refers to the relationship between theory and study. Experiments need to be set up such that to the highest degree possible, they are representative of the theory under test. The experiments reflect the construct of cause and results reflect the construct of effects well [30]. Since the delay generated by the Internet (e.g., router, DNS, etc.) is complicated and unpredictable, it is hard to say the extent to which packet loss and jitter impact delay. Also, the transportation and routing layers employ self-adaptive mechanisms to adjust the performance of specific applications, e.g., GSR and Siri. In the end, both the jitter and the packet loss are generated by a specific program (i.e., simulated), rather than real network conditions. It is hard to know whether the simulated impact has the same effects of real jitter or packet loss.

7.4 External Validity

The external validity is all about generalization. Can we generalize the result of the treatment outside the scope of our study in case of a causal relationship between cause and the construct [30]? All of the experiments were conducted in our lab and through our campus network. It is likely that the configuration of our campus network is different from other networks, such as firewalls and TCP/UDP controls. Hence, the conclusion obtained from the experiment cannot be generalized to common network environments. In addition, the available bandwidth of different regions in United States is different. It is possible that this diversity affects the conclusion that it cannot be applied to the other regions in United States. Finally, the sample is small (the evaluation is run on one desktop in a laboratory setting). A larger scale experiment running on more desktops, as well as laptops and smart phones, will lessen external threats.

8 Conclusions and Future Work

We designed and implemented experimental evaluations of Siri and GSR, and Dragon. Using experiment data, we designed four models to evaluate the effects of jitter and packet loss separately. After conducting ANOVA tests for each experiment, we found that the effects of packet loss and jitter on delay are statistically significant but the impact is not important compared to the one that comes from the application, because from the tables we can see that the application generated most of the impact. In addition, we found that GSR performs better than Siri in respect to delay. Results from the Dragon testbeds shows that the effects of packet loss and jitter on delay and also accuracy are statistically significant but the impact is not important compared to the one that comes from the application.

Statistical findings of effect of jitter and loss on the accuracy and delay show that the application generated most of the impact.

Delay of all applications is affected by packet loss and jitter. In order to design and implement real-time cloud speech recognition applications for more critical tasks, there should be mechanisms to measure loss/jitter tolerant systems. Network coding is a possible solution to reduce the effect of packet loss and jitter [8, 24, 26, 27]. Using TCP keeps these applications accurate under packet loss and jitter values, but as we saw in our results, it affects the roundtrip delay. By using UDP and network coding, we can keep the system accurate under different values of jitter and packet loss while we reduce the resulting delay. Future cloud based speech recognition applications that use cellular networks are still required to overcome this problem; which is due to the presence of jitter from packet transmission over different paths.

This experiment can also be extended by running Siri and GSR over different cellular networks, and adding the celluar data provider as another blocking variable.

Running the experimental setup over a wide geographical range of clients and also using different cellular data providers can result in more accurate results. Considering clients with a diversity of hardware and software configurations can be another extension for this research.

Acknowledgements

We thank the National Science Foundation for supporting this work through grant NSF CSR-1527097.

References

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

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

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

[4] Saamer Akhshabi, Ali C. Begen, and Constantine Dovrolis. An experimental evaluation of rate-adaptation algorithms in adaptive streaming over http. In *Proceedings of the Second Annual ACM Conference on Multimedia Systems*, MMSys '11, pages 157–168, New York, NY, USA, 2011. ACM.

[5] Saamer Akhshabi, Sethumadhavan Narayanaswamy, Ali C Begen, and Constantine Dovrolis. An experimental evaluation of rate-adaptive video players over http. *Signal Processing: Image Communication*, 27(4):271–287, 2012.

[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] Mehdi Assefi, Mike P. Wittie, and Allan Knight. Impact of network performance on cloud speech recognition. *ICCCN*, IEEE. Aug. 2015.

[9] Mehdi Assefi, Mike P. Wittie, Guangchi Liu, and Clemente Izurieta. An experimental evaluation of apple siri and google speech recognition. *SEDE*, ISCA. Oct. 2015.

[10] Ł Budzisz, Rade Stanojević, Arieh Schlote, Fred Baker, and Robert Shorten. On the fair coexistence of loss-and delay-based tcp. *IEEE/ACM Transactions on Networking (TON)*, 19(6):1811–1824, 2011.

[11] Kuan-Ta Chen, Cheng-Chun Tu, and Wei-Cheng Xiao. Oneclick: A framework for measuring network quality of experience. In *INFOCOM 2009, IEEE*, pages 702–710. IEEE, 2009.

[12] Kuan-Ta Chen, Chen-Chi Wu, Yu-Chun Chang, and Chin-Laung Lei. A Crowdsourceable QoE Evaluation Framework for Multimedia Content. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 491–500. ACM, 2009.

[13] 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.

[14] 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.

[15] Luca De Cicco and Saverio Mascolo. *An experimental investigation of the Akamai adaptive video streaming*. Springer, 2010.

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

[17] Google. *Performing speech recognition over a network and using speech recognition results*. `http://www.google.com/patents/US8335687`.

[18] David A Hayes and Grenville Armitage. Improved coexistence and loss tolerance for delay based tcp congestion control. In *Local Computer Networks (LCN), 2010 IEEE 35th Conference on*, pages 24–31. IEEE, 2010.

[19] 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.

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

[21] Te-Yuan Huang, Polly Huang, Kuan-Ta Chen, and Po-Jung Wang. Could skype be more satisfying? a qoe-centric study of the fec mechanism in an internet-scale voip system. *Network, IEEE*, 24(2):42–48, 2010.

[22] Xin Lei, Andrew Senior, Alexander Gruenstein, and Jeffrey Sorensen. Accurate and compact large vocabulary speech recognition on mobile devices. In *INTERSPEECH*, pages 662–665, 2013.

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

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

[25] J. Roskind. *QUIC: Design Document and Specification*, Dec. 2013. `https://docs.google.com/a/chromium.org/document/d/1RNHkxVvKWyWg6Lr8SZ-saqsQx7rFV-ev2jRFUoVD34`.

[26] 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.

[27] 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.

[28] Stefan Winkler and Ruth Campos. Video quality evaluation for internet streaming applications. In *Electronic Imaging 2003*, pages 104–115. International Society for Optics and Photonics.

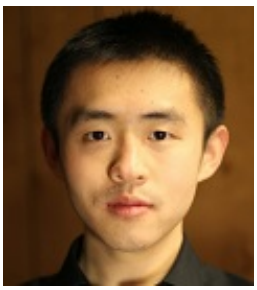
[29] Stefan Winkler and Frédéric Dufaux. Video quality evaluation for mobile streaming applications. In *Visual Communications and Image Processing 2003*, pages 593–603. International Society for Optics and Photonics, 2003.

[30] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. Experimentation in Software Engineering. pages 102–104. Springer Science & Business Media, 2012.

[31] Yang Xu, Chenguang Yu, Jingjiang Li, and Yong Liu. Video telephony for end-consumers: measurement study of google+, ichat, and skype. In *Proceedings of the 2012 ACM conference on Internet measurement conference*, pages 371–384. ACM, 2012.

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

Mehdi Assefi is a graduate student in the Computer Science department at Montana State University. Born in Mashhad, Iran, his research interests include cloud computing, network performance evaluation, and real-time streaming. Mehdi Assefi has approximately 11 years teaching and research experience.



Guangchi Liu is currently a Ph.D candidate in the department of computer science, Montana State University, Bozeman, MT, USA. He received his B.E. in biomedical engineering, and M.E. in electrical and computer engineering from Southeast University, China in 2009 and 2012, respectively. His research interests include Internet of things, trust assessment, social network, and wireless sensor network.

Dr. Mike P. Wittie is a RightNow Technologies Assistant Professor and a co-director of the Networks+Algorithms Lab at the Montana State University Computer Science Department since Fall 2011. His research interests focus on latency reduction, network measurement, and content delivery in wide-area networks. He received his PhD in Computer Science from the Computer Science Department at the University of California, Santa Barbara, and MSE in Computer Science and BA in Cognitive Science from the University of Pennsylvania. He worked professionally on military datalink integration for Anzus Inc. (now Rockwell Collins).

Dr. Clemente Izurieta is an Assistant Professor in the Computer Science department at Montana State University and holds a PhD from Colorado State University. Born in Santiago, Chile, his research interests include empirical software engineering, design and architecture of software systems, the measurement of software quality, and technical debt. Dr. Izurieta has approximately 16 years experience working for various RD labs at Hewlett Packard and Intel Corporation and currently directs the Software Engineering Laboratories (SEL) at Montana State. SEL has funding from NSF, DoD, and the State of Montana and supports 3 PhD and 6 MS students.