Measurement Study of Commercial Video Conferencing Systems

Yang Xu, Chenguang Yu, Jingjiang Li, Hao Hu, Yong Liu and Yao Wang
Department of Electrical and Computer Engineering
Polytechnic Institute of NYU
Brooklyn, New York

1. INTRODUCTION

In this technical report, we present our recent measurement study of commercial video conferencing systems. In our experiments, we focus on investigating three questions:
Q1: What is the service architecture of each system?
Q2: How is video quality adapted to different network conditions in those systems?
Q3: What is the audio and video delays perceived by users?
To answer these questions, in the following sections, we will present our experiment settings, the obtained results and insights derived from them. The rest of paper is organized as follows. Section II describes out testbeds for the experiment. Then we will answer Q1,Q2,Q3 in Section III.

2. EXPERIMENTS SET-UP

Four systems are investigated in our study:

- Skype Group Video Call (version 5.5.32.117): Server-Client(S/C) based
- Google+ Video Hangout: web-browser based, Server-Client(S/C)
- iChat: centralized P2P
- VSee (version 11.0.0.1129): decentralized full-mesh P2P

To study them, we develop the following measurement testbeds.

2.1 Studying Video Conferencing Topology

To get video conference topology, as the testbed described in Fig. 1(a), three people set their videos on and form a video conference. During the video call session, we use Wireshark[1] to capture network packets sent out and received by each computer. Each player in the conference generates voice and video packets constantly at significant rates.

In our Wireshark analysis, we capture those TCP and UDP sessions, whose duration is long enough (at least half of the conference session time) and at the same time flow rate is high enough (larger than 5kbps), as voice or video flow sessions. Besides, these applications all have technical information display windows. We also use information from those windows to help us identify the topology.

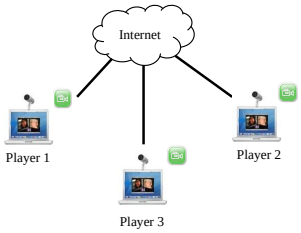
2.2 Studying Video Adaptation under Different Network Conditions

In the testbed(Fig. 1(b)) for studying video adaptation under different network conditions, only one user, referred as the sender, sets his video on and the other two users are purely video receivers. We denote such case as a video sub-conference. As sub-conference is a basic component of the whole video conference, focusing on this can simplify our analysis. We can inject video sequence into softwares using a virtual video camera tool [3]. This ensures the transmitted video contents are consistent and repeatable, and let us focus on the impact of network conditions. To emulate a wired or wireless network, we employ a software-based network emulator, NEWT [4]. It can emulate a variety of network attributes, such as propagation delay, random packet loss, and available bandwidth. In the testbed, we add one emulator in the uplink of the video sender and one emulator in the downlink of one video receiver.

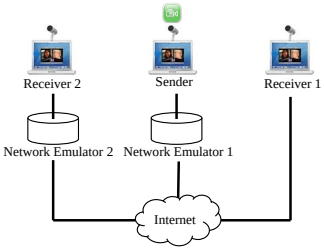
NEWT can only run on Windows machine. In addition, when packet loss is added using that tool, Wireshark can only capture those packets after the loss. In order to use the emulator in different operation systems and capture both packets before packet loss and after packet loss, we set up an ad-hoc wireless connection in our testbed. As in Fig. 1(c), we use one computer as the access point to the Internet. Another computer, referred as player computer, running the video conference software, employs wireless ad-hoc mode to connect to that access point. Thus, the input flow and output flow of the player all go through the access computer. And we can choose to install NEWT in player computer or in access computer to fully observe packets before and after packet loss.

2.3 Studying One-way Video Display Delay

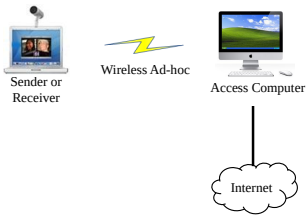
In the testbed in Fig. 1(d), we run a stopwatch program [6] on a sender computer, and focus the video camera on the stopwatch window so that the stopwatch window is transmitted to the video receiver by the conferencing software. Thus, the video of this stopwatch would show in the screen of video receiver. We put the monitors of these two computers side-by-side, and use camera to take the pictures of two monitors three times per second. At any given time, the difference between the clocks of these two monitor is the one-way end-to-end video display delay perceived by that user.



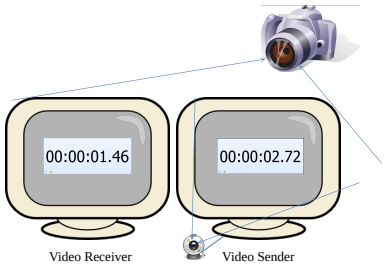
(a) Studying Video Conference Topology



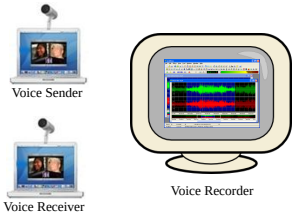
(b) Studying Video Adaptation



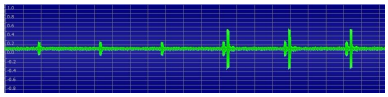
(c) Ad-hoc Connection



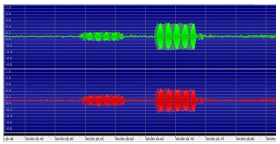
(d) Studying One Way Video Display Delay



(e) Studying One Way Voice Delay



(a) Recorded Voice Wave



(b) Enlarged Two Adjacent Voice Waves

(f) Voice Waves Detail

Figure 1: Testbeds in the experiment

For example, in Fig. 1(d), the stopwatch program running at the sender side shows time "00:00:02.72" and at the same time, the received video in video receiver shows time "00:00:01.46". Thus, the video display delay under such case is 1260 ms. After getting these pictures, we use softwares [8][9] to first turn pictures into mono color, and then extract stopwatch parts from the achieved mono color pictures. Finally, these stopwatch pictures will be changed to text file by using Optical character recognition (OCR) software [10]. It should be noted that sometimes the photo we get shows the transition state. Under such state, the last two numbers in the stopwatch are in the display transition state and we can not decide the exact numbers they represent. Thus, we omit the data under such condition.

2.4 Studying One-way Voice Delay

In order to record one-way voice delay of the conference software, we employ a repeatable "tick" sound as the voice source. In the corresponding testbed in Fig. 1(e), we use a dedicated computer as voice recorder. A software[7] installed in that computer records both the sound injected to the voice sender and the sound coming out of the voice receiver. Thus, the time difference between that two sound waves are the one way voice delay.

Fig. 1(f) shows one instance of the experiment. In subfigure (a), the impulse with smaller amplitude corresponds to the repeated voice signal injected to the voice sender. In the implementation, we set the volume of sound out of the receiver to the maximum. Thus, the impulse with larger amplitude is the voice signal out of the voice receiver. Then, in the enlarged figure of two adjacent voice waves (sub-figure (b)), the temporal difference between the first peaks of two adjacent voice waves is the one-way voice delay.

Let D_{encode} be the video or voice encoding time at sender, $D_{oneWayTrans}$ be one way transmission delay, $D_{process}$ be server or super node process time (0 if there is no server or super node) and D_{decode} be the video or voice decoding time at receiver. Thus, the one way video display delay or one way voice delay we measure from the experiment is the summation of these four parts. We subtract $D_{oneWayTrans}$ from the experiment result, then one-way delay $D_{oneWayDelay}$ can be expressed as following:

$$D_{oneWayDelay} = D_{encode} + D_{process} + D_{decode} \quad (1)$$

3. MEASUREMENT RESULTS

3.1 Network Topology

To get the right topology and differentiate between signalling packets, video packets and voice packets, we conduct three types of experiments. The first experiment is like the example in Fig. 1(a). Every involved user set his video and voice on to form a video conference. In the second experiment, users choose to only set their microphone on to form a voice conference. For the third experiment, all users shut down videos and mute their voices.

Skype: Skype two-party video call uses direct Peer-to-Peer connection. When three or more users are involved, the network topology is shown in Fig. 2(a). Voice flow transmission uses centralized P2P architecture. The initiator, either the one who initializes video conference or the one who first adds people to video conference, serves as the super node. In the conference, only initiator has the right to add people into the conference or close the entire conference session. Normal user uploads his voice flow to the initiator and download other user's voice information from the initiator. When three users participate in the conference, the upload flow rate from a normal player to the initiator is around

40kbps. And the download rate from the initiator to that player is around 50kbps, which is less than two times of the single voice flow upload rate. This indicates that initiator might use sound mix technique to reduce voice flow rates. For the video transmission, each user uploads his video flow to a server and that server relays video flows to other users. Most of the time, each user choose different relay servers. From our experiments, those servers are all in the same subnetwork 208.88.186.00/24, which locates in Estonia. Usually network flows in the transmission use UDP. Sometimes they choose to use TCP. In addition to voice and video flows, we also observe some small rate flows between servers and players, players and players. These small flows might carry feedback or control information.

Google+ Hangout: Whenever two or more users are involved in the conference, the network topology of Google+ Hangout is all like the case in Fig. 2(b). No direct network flow exists between users. Each user sends his video and voice flows to a dedicated proxy server and also receives others' video and voice flows from that server. Generally, users choose different dedicated proxy servers. Thus, the proxy servers need to communicate with each other to exchange user's video and voice information. Each user has four different connection sessions with the proxy server and those four sessions all connect to the same server port (Port 19305). Most of the time, these four flows all use UDP for transmission, although the rare case of using TCP exists. There is a trick that we could access the detailed statistics information about Google+ Hangout.[11]. The statistics information shows that two of the four sessions are video and voice flows respectively. And Google use RTP protocol[13] to transmit its video and voice network flows. The other two flows' payloads conform to the format of RTCP protocol and we infer that those two flows carry signal information for voice and video flows. From our observation, the proxy servers involved are all in California.

iChat: Like Skype, the concept of initiator exists in iChat. It employs a centralized P2P architecture as shown in Fig. 2(c). Normal user sends one UDP flow, combining his video and voice information together, to the initiator. At the same time, that normal user receives other participants' video and voice information through one UDP flow from the initiator. Participants choose to use port 16402 in his transmission. Normal users only have network connection to the initiator. No direct UDP flows exists between normal users.

VSee: As Fig. 2(d) shows, VSee uses a decentralized full-mesh P2P architecture. One user sends one UDP flow which combines his video and voice information to each other user separately. In the case of Fig. 2(d), Player 1 uses one UDP flow $Flow_{12}$ for the information transmission to Player 2 and another UDP flow $Flow_{13}$ for the information transmission to Player 3.

We did many experiments and also tried the case of involving four or more users. The architectures are also the same as above.

3.2 Video Adaptation

Different video sources would generate different transmission rates in conferencing systems. Table. 1 shows transmission rates of each system under a highly varied video sequence "Football" and a much smoother video sequence "Akiyo". In the result, Skype maintains transmission rate at the same level. iChat and VSee produce much more rates under "Football" sequence. With that sequence, in the opposite trend, Google+ has less rates. In the following section, we use "Akiyo" as the video source to simulate a practical video telephony. The sequence is encoded using a H.264/AVC video codec [2]. And as described in the set-up section, we mainly focus on one sub-conference.

Table 1: Video Transmission Rate under Different Standard Video Sequences

Softwares		Akiyo	Football
Skype	Two-party	1061.2 kbps	1134.7 kbps
	Three-party	523.4 kbps	513.8 kbps
Google Plus		835.7 kbps	314.9 kbps
iChat	Two-party	752.8 kbps	1392.2 kbps
	Three-party	365.8 kbps	1173.4 kbps
VSee		206.5 kbps	648.5 kbps

Table 2: Skype Video States Definition

Video States	Definition
One-version	Video sender only generate one video version
Multi-version	Video sender generates multiple video versions
Kick-out	Video receivers in bad condition is kicked out of video conference
Shutdown	Sender decides to close its video sub-conference

3.2.1 Video States Definition

For the clarity of presentation, we first define *video states* that we encountered in the experiments. All video states are enumerated in Table. 2.

One-version state is the usual video operation state. Under that state, at any given time, the video sender only generates one video version and uploads that version to servers (S/C architecture) or receivers (P2P architecture). When the network condition is not very good (either the link bandwidth is small or link loss is high), other three video states might emerge.

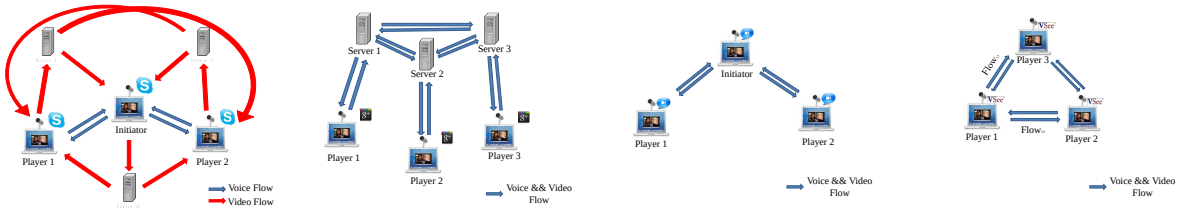
Multi-version state means that the sender generates several different video versions. Each video version has a unique video rate. Such a state happens when network conditions of receivers are heterogenous and the system doesn't support layered video coding. Then, multiple video versions are produced for different receivers' conditions.

Kick-out state will happen when one receiver's download link is very bad. Under such state, the receiver in bad condition is kicked out of video conference. That receiver can at most receive voice information.

Shutdown state means that the sender closes its video sub-conference. It happens when the upload link condition of the video sender is very bad. In addition, when only download link of one receiver is bad, we may also observe such state.

3.2.2 RTP protocol

Google Plus Hangout and iChat use RTP[13] protocol for their video and voice transmission. Thus, understanding the



(a) Skype multi-party (b) Google Plus Hangout (c) iChat (d) VSee

Figure 2: Network Topology of the softwares

meanings of the fields in RTP protocol are helpful for us to investigate functions of these two systems. The description of all fields in the standard RTP protocol can be found in [14]. We just select some useful fields that can help us in understanding the mechanisms of those two softwares to discuss here.

"Version" field always sets to 2 in RTP protocol. Google Plus Hangout uses one flow to transmit video information and one flow to transmit voice information. 97.2% packets in the video flow and 85.9% packets in the voice flow are RTP packets. iChat transmits video and voice information using one flow. 99.9 % packets in that flow are RTP packets. The non-RTP packets in those flows might be signal or feedback packets.

"SSRC" filed identifies the synchronization source. The value is chosen randomly, with the intent that no two synchronization sources within the same RTP session have the same SSRC. In Google Plus Hangout, for a receiver, packets from multiple other users merge together in one RTP session. This field is used to identify which user generated the packets received. In iChat, regardless of the number of users involved in the conference, we can only observe two different "SSRC" values. Packet length for one "SSRC" value is very low (around 70 Bytes). Those packets occupy 44.5% of the total flow. For the other "SSRC" value, packet length is kind of high (> 560 Bytes). We infer those two kind of packets are voice and video packets respectively. And iChat use the SSRC field to differentiate video packets from voice packets.

"Sequence Number" field increments by one for each RTP data packet sent, and could be used by the receiver to detect packet loss and to restore packet sequence. In Google Plus Hangout, packets from the same sender in one flow form a unique sequence. In iChat, packets of video and packets of voice form two different sequences.

"Timestamp" field reflects the sampling instant of the first octet in the RTP data packet. We infer data in that field means the sample time of the frame. And all packets belonging to the same frame have the same timestamp value.

If the packet is the last one for one frame, then the "Marker" field for that packet would be 1, otherwise the value is 0. Thus, we infer that such "Marker" field means whether the packet is frame boundary or not.

3.2.3 *Video Parameters*

From the perspective of viewer, the perceived video quality is determined by video encoding parameters. Scopes of resolution values for all systems are listed in Table. 3. FPS can vary from almost 1 to 30 for each system.

In our experiments, we find that some systems only change some parameters for video adaptation. We consider three major parameters: resolution, FPS and quantization, and list whether they vary in different systems in Table. 4.

Table 3: Resolution Values

Skype	Google	iChat	VSee
640*480, 320*240, 160*120	640*360,480*270, 320*180,240*135, 160*90,80*44	640*480, 320*240, 160*120	1280*720,640*480, 320*240,160*120

Table 4: Varied Parameters

Softwares	Resolution	FPS	Quantization
Skype	✓	✓	✓
Google Plus	✓	✓	
iChat		✓	✓
VSee		✓	

Skype adapts all three parameters. We didn't observe the change of quantization in Google Plus Hangout. iChat's video resolution is determined by the number of users in the conference. For example, the resolution of video is always set to be 640×480 in the case of two-party call. When three or more users involved in the conference, resolution of video becomes 320×240 or 160×120. And once it is decided in the beginning, that value will not be changed. VSee let users decide video resolution and FPS level that they want to use. It has three FPS level options: Low, Adaptive and High (30fps). In the experiment, we set the resolution to be 1280×720 and FPS to be Adaptive. Then, we only observe variations of FPS in the video adaptation process.

3.2.4 *Video Adaptation under Bandwidth Variation*

When the upload link bandwidth of a sender varies in a video sub-conference, the video rate out of the sender changes correspondingly. Generally, for these four systems, the higher the upload link bandwidth, the larger the sending rate. When the upload bandwidth is too small, those systems may enter into Shutdown state. This shows that those softwares have their own network probing algorithms to determine the video quality to send out.

Then, we change download bandwidth to observe video adaptation. In the experiment, we only set the download bandwidth constraint to one of the receivers. Mostly, those four softwares are all running in One Version state. Under such state, for iChat, VSee and Skype, all receivers receive the same video version. At this time, receiver in the worst condition determines the video quality sent out by the sender. The result also shows that these three softwares don't employ layered video coding or video trans-coding. Google Plus shows a distinct behavior from the others. Heterogenous receivers can receive different video versions. For example, we only limit the download bandwidth of one re-

ceiver, saying receiver 2, to be 500 kbps. The result shows that for video flow out of the sender, the sending rate is 835.7 kbps, video resolution is 640*360 and video FPS is 30. For video flow to receiver 1, received rate is 386.9 kbps, video resolution is 640*360, video FPS is 14. And for video flow to receiver 2, received rate is 168.6 kbps, video resolution is 320*180, video FPS is 14. The experiment environment is not fully controlled, as flows involved in Google Plus Hangout have to compete with unknown flows in the Internet. That might be the reason that the received quality on receiver 1 is lower than the original video sent out by the sender, although we didn't add any constraint in receiver 1. At this time, both receiver 1 and receiver 2 receive consistent and acceptable video quality. We show the payloads of some packets from the experiment in Table. 5. In the table, the sender sends out 5 frames, both receiver 1 and receiver 2 only receive 2 frames. Besides, receiver 2 only receives the first two packets of these two frames. It should be noted that sequence numbers in receiver 1 and receiver 2 are consistent. Thus, the loss of frames or packets is decided by the server. **Payload analysis** shows that video codec of Google Plus Hangout has both temporal and spatial scalability. In the experiment, we also find that no matter how low the download bandwidth of receiver is, received video resolution could only be one quarter of the original sent video resolution. This shows that encoded video only has two spatial layers. It is reasonable as spatial layers inducing much more overheads compared to temporal layers. Thus, the number of spatial layers could be too large. Without layered coding, to deal with heterogenous receivers, senders in Skype enter into Multi-version state. In the experiment, it generate as many as three different resolution video versions when four heterogenous receivers involved in the conference. The sender of other two systems always only generate one video version. These four systems may enter into Kick-out state when one of its receivers is in bad condition.

In the experiment, VSee doesn't show a good network probing capability. We find a high loss ratio when we limit the bandwidth. VSee only automatically changes one video parameter, the FPS. Thus, it only have a very coarse adaptation granularity.

3.2.5 Video Adaptation under Loss Variation

To find how those softwares work under packet losses, two types of experiments are executed. One is to add upload losses on the video sender. The other one is to add download losses on only one of the receivers. Because we capture flows both before and after packet loss, we can figure out which packets are lost. In the analysis, we can check if there exists any other packets that have similar packet payloads like the lost ones to see whether softwares use retransmission or not.

Skype: We haven't observed any retransmission packets when packet losses are induced. From Skype's technical window, we can easily observe a gap between packet rate and video rate. Previous study[12] shows that Skype Two-party Video Call employs Forward Error Correction (FEC) coding. We infer that here the gap is also due to FEC. Let r_v be the actual video rate, r_s be the actual sending rate and we define the FEC redundancy ratio ρ as the ratio between the redundant traffic rate and throughput:

$$\rho = \frac{r_s - r_v}{r_s} \tag{2}$$

Thus when ρ becomes larger, there are more redundant bits. In the experiment, we set the upload bandwidth of video sender to be 400kbps. The experiment results of adding upload losses and adding download losses are showed in Table. 6 and Table. 7 respectively. When doing experiment about download loss, we only add download loss to receiver 1. Because of the non-controlled effects in the Internet, we can't get a precise model on Skype's behavior like [12]. But there is still the trend that as loss rate becomes higher, more FEC packets are added into video flow. In Table. 6, we can also observe that when upload loss rate is high, the received flow rate is smaller than the flow rate sent from the video sender. This scenario indicates that the relay server first removes FEC packets of incoming network flow and then adds new FEC packets to the network flow according to the receiver condition. Result of Table. 7 shows that two receivers in different conditions receive a same video version with different FEC ratio. Thus, we infer that relay server monitors network conditions of receivers and decide corresponding FEC policies to different receivers.

Google+ Hangout: Previous sections show that Google+ Hangout employs SVC and each receiver's dedicated server decides the number of layers he received. Each receiver has no correlation with each other. Thus, we only focus on one video sender and one video receiver here. Surprisingly, sometimes Google+ Hangout still survive under random loss of 40%. When loss is induced, retransmission packets are observed. We list the experiment results in Table. 8 and Table. 9. For both two types of experiments, the more losses are induced in the link, the more retransmissions appear in the flow. However, Sender video FPS doesn't change too much when we increase upload loss from 0% to 20%. If we consider the video rate after subtracting retransmission rate, we find that number doesn't change too much. The results of adding download losses in the receiver shows a different behavior. Received video FPS decreases as download loss rate increases. The upload video quality decides QOE of all receivers. That might be the reason that Google+ Hangout tends to maintain the upload video quality. Because we couldn't control the network condition of Internet, retransmission under packet loss of 0% is still being observed. We denote retransmission time interval as the temporal difference between the last transmission and retransmission. And the CDFs of retransmission time interval for the experiments are shown in Fig. 3. Most of the time, Google Plus Hangout just try to do retransmission once or twice after a packet is lost. Sometimes, we can observe it tries to retransmit many times, with the highest is up to 18 times. 60% of the retransmission happens within 70ms after the first original packet transmission, which is about 3.75 times of RTT. The retransmissions on the video uploader side all need to wait such a long time. When server does retransmission to video receivers, the N-th ($N \geq 5$) retransmission happens only about 5ms after the previous retransmission. We infer such duplicate retransmissions can solve the problem of bursty loss. And the statistics of packets show that Google Plus Hangout do retransmissions only to protect some packets, which might be the lower layer video packets in SVC. And using retransmission, it manages to transmit those packets under protection successfully almost 100%.

iChat: The experiment results are shown in Table. 10 and Table. 11. When doing the experiment of download loss, we only add download loss to receiver 2. We observe the existence of retransmission when loss is induced. The

Table 5: Packet Payloads in Google Plus Hangout

Marker	Timestamp	Packet Length (bytes)	Sequence Number		
			Sender	Receiver 1	Receiver 2
0	2063696701	1269	61603	44445	52498
0	2063696701	1113	61604	44446	52499
0	2063696701	1278	61605	44447	
0	2063696701	1234	61606	44448	
0	2063696701	1283	61607	44449	
0	2063696701	1277	61608	44450	
0	2063696701	1077	61609	44451	
1	2063696701	989	61610	44452	
0	2063699269	621	61611		
1	2063699269	560	61612		
0	2063703362	1086	61613		
0	2063703362	485	61614		
0	2063703362	1167	61615		
1	2063703362	1048	61616		
0	2063706604	543	61617		
1	2063706604	914	61618		
0	2063709620	1276	61619	44453	52500
0	2063709620	1067	61620	44454	52501
0	2063709620	1272	61621	44455	
0	2063709620	1267	61622	44456	
0	2063709620	1279	61623	44457	
0	2063709620	1276	61624	44458	
1	2063709620	736	61625	44459	

Table 6: FEC Adaptation at Skype Sender Side

Upload Loss rate	Video Rate (kbps)	Video Sender Side		Video Receiver 1		Video Receiver 2	
		Sent Rate (kbps)	FEC Ratio ρ_s	Sent Rate (kbps)	FEC Ratio ρ_r	Sent Rate (kbps)	FEC Ratio ρ_r
0	209.657	248.304	0.1556	306.536	0.3160	309.464	0.3225
0.02	119.726	190.256	0.3833	160.192	0.2526	164.472	0.2721
0.05	71.227	160.464	0.5783	97.168	0.2670	102.176	0.3029
0.08	85.119	209.096	0.6255	134.736	0.3683	135.000	0.3695

Table 7: FEC Adaptation at Skype Relay Server Side

Download Loss rate	Video Rate (kbps)	Video Sender Side		Video Receiver 1		Video Receiver 2	
		Sent Rate (kbps)	FEC Ratio ρ_s	Sent Rate (kbps)	FEC Ratio ρ_r	Sent Rate (kbps)	FEC Ratio ρ_r
0	209.657	248.304	0.1556	306.536	0.3160	309.464	0.3225
0.02	196.971	217.128	0.0928	306.000	0.3692	215.952	0.0879
0.05	187.722	208.184	0.0983	442.680	0.5971	209.000	0.1018
0.08	131.039	148.144	0.1155	312.688	0.6145	209.072	0.3732

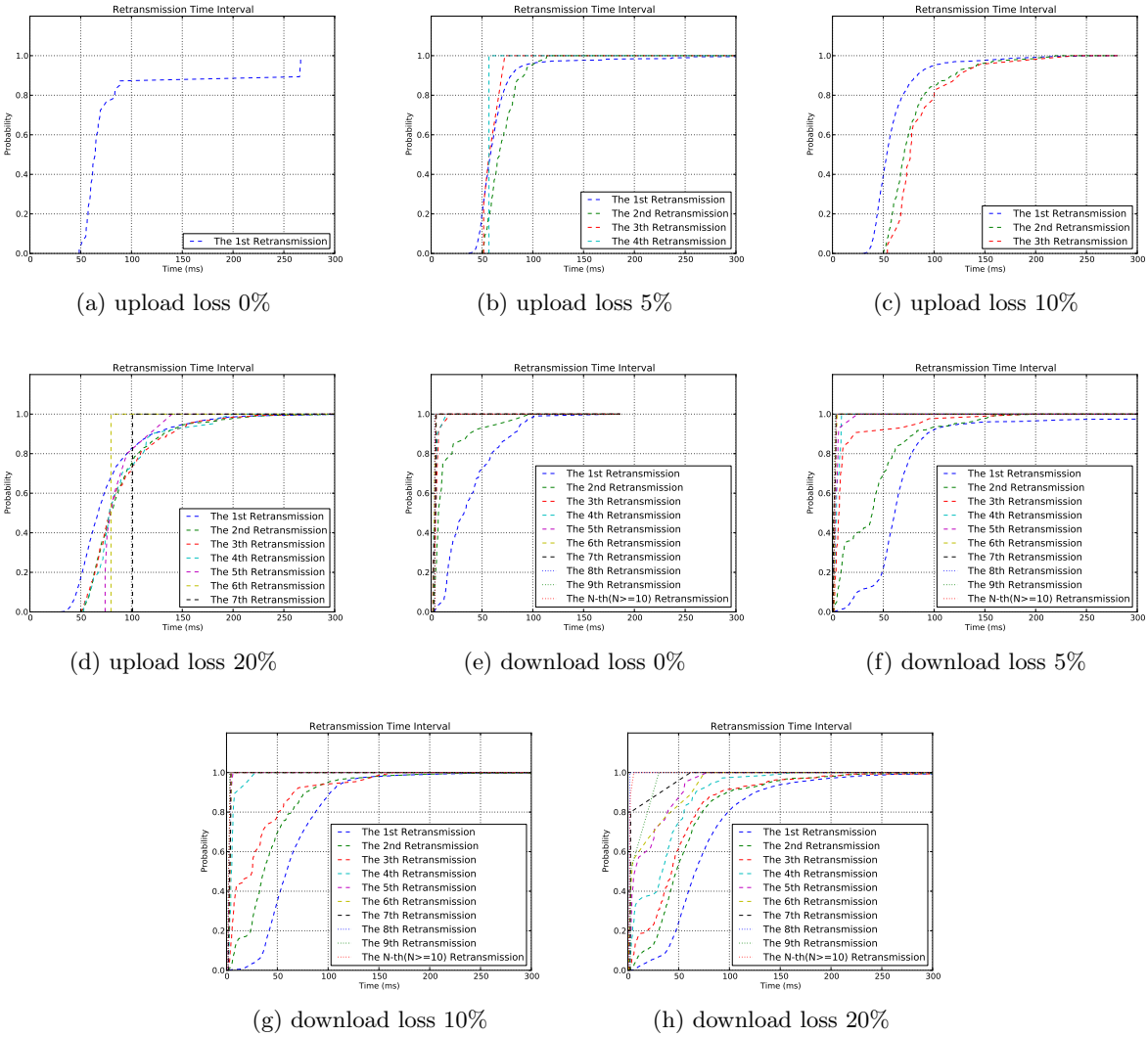
same as Google, retransmission rate increases as loss rate increases. Unlike Google, video FPS decreases when upload losses are induced. However, that value doesn't change too much under download loss. And iChat doesn't use a duplicate retransmission policy. Most of the time, it just try to do retransmission once. As the retransmission time inter-

Table 8: Retransmission in the Upload Link of Google Plus Hangout

Upload Loss	Video Flow Loss Ratio	FPS	Total Video Upload Rate (kbps)	Retransmission Rate (kbps)
0	0	29.973	826.0	0.9
0.05	0.0476	29.975	836.9	24.2
0.10	0.0937	29.973	885.7	52.1
0.20	0.1672	29.977	857.2	101.4

Table 9: Retransmission in the Download Link of Google Plus Hangout

Download Loss	Video Flow Loss Ratio	FPS	Total Video Download Rate (kbps)	Retransmission Rate (kbps)
0	0	27.91	810.8	4.4
0.05	0.0495	27.07	827.3	35.2
0.10	0.0889	20.158	744.3	67.4
0.20	0.1695	19.231	677.7	116.4



(a) upload loss 0% (b) upload loss 5% (c) upload loss 10%

(d) upload loss 20% (e) download loss 0% (f) download loss 5%

(g) download loss 10% (h) download loss 20%

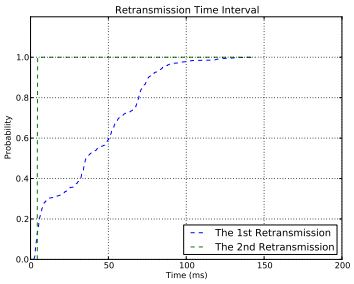
Figure 3: Retransmission Time Interval under loss variation for Google Plus Hangout

Table 10: Retransmission in the Upload Link of iChat

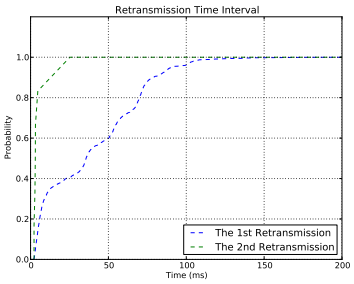
Upload Loss	Video Flow Loss Ratio	Video FPS	Total Video Rate (kbps)	Video Retransmission Rate (kbps)
0	0	25.096	365.7	0
0.02	0.020	19.844	363.0	7.0
0.10	0.091	19.598	411.5	34.4

Table 11: Retransmission in the Download Link of iChat

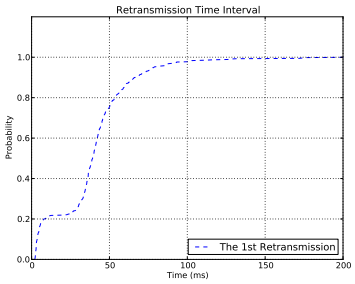
Download Loss	To Receiver 1	To Receiver 2					
	Sent Rate (kbps)	Sent Rate (kbps)	Video Flow Loss Ratio	Video FPS	Total Video Rate (kbps)	Video Retransmission Rate (kbps)	Voice Rate (kbps)
0	389.6	390.1	0	25.096	365.8	0	24.0
0.02	389.7	426.3	0.0188	24.260	401.7	7.8	24.4
0.10	388.0	474.3	0.092	24.187	447.3	37.7	26.7



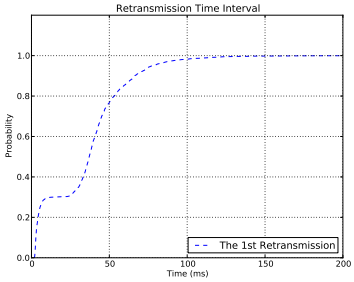
(a) upload loss 2%



(b) upload loss 10%



(c) download loss 2%



(d) download loss 10%

Figure 4: Retransmission Time Interval under upload loss variation for iChat

val shown in Fig. 4, 60% of a retransmission happens within 50ms after the original transmission. And at this time, RTT between machines is only about $2-4$ ms, as we set machines are in the same subnetwork. The second retransmission happens only 3ms later than the first retransmission. Also iChat just choose some lost packets to do retransmission. And some packets are still lost even if it does retransmission to protect them.

VSee: The experiment results are shown in Table. 12 and Table. 13. When doing experiment of download loss, we add losses to receiver 1. As loss rate becomes higher, we can observe the increase of the flow rate. And we haven't observed any retransmission packets in the experiment. In order to understand the behavior that sender did under packet losses, we try to analyze the payload of packets. We know that VSee doesn't employ SVC and the video versions from sender to receivers are all the same. Thus, the general video part of two flows to different receivers should be the same. In the experiment, we use Windows 7 to run VSee. Windows has the feature that the IPID of packets generated from the same port number should increase linearly. Coincidentally, VSee chooses just one port number for all its transmission. Using such feature, we list the packet information under no loss condition in Table. 14. In that table, we could see that the IP ID of those packets increase one by one. And mostly the length of the corresponding packets in that table is the same. For example, the lengths of packets with IP ID 26799 and 26800 are the same. We infer those packets should convey the same content. Looking into the payload of those two packets, they are totally different. That might

Table 12: Video Adaptation under Upload Packet Loss for VSee

Upload Loss	Rate from Sender to Receiver 1 (kbps)	Rate from Sender to Receiver 2 (kbps)
0	206.543	204.875
0.05	216.542	212.672
0.10	222.856	225.003
0.20	240.663	242.200

Table 13: Video Adaptation under Download Packet Loss for VSee

Download Loss	Rate from Sender to Receiver 1 (kbps)	Rate from Sender to Receiver 2 (kbps)
0	206.543	204.875
0.02	206.392	202.550
0.05	214.954	203.126
0.10	235.719	194.57

Table 14: Packet Information under No Loss Condition For VSee

Sender to Receiver 1			Sender to Receiver 2		
Capture Time(ns)	IP ID	Packet Length(KB)	Capture Time(ns)	IP ID	Packet Length(KB)
1325521587318240	26799	462	1325521587318863	26800	462
1325521587377320	26801	110	1325521587377495	26803	76
1325521587377355	26802	556	1325521587377532	26804	556
1325521587377617	26805	556	1325521587377731	26806	556
1325521587378374	26807	556	1325521587378522	26808	556
1325521587379384	26809	556	1325521587379515	26810	556
1325521587380386	26811	108	1325521587380543	26813	108
1325521587380407	26812	556	1325521587380571	26814	556
1325521587381394	26815	556	1325521587381566	26816	556
			1325521587433555	26817	256
1325521587441696	26818	288	1325521587451146	26820	462
1325521587450977	26819	556	1325521587451182	26821	556

be the reason of encryption. For different flow session, VSee may choose different public key and private key combination. Some corresponding packet lengths are not the same. Or one flow has more packets than another flow, like the existence of packet with IP ID 26817. We infer such packet is for error correction. Different flows endure different network conditions. Thus, their packets for error correction are not the same. But generally, under no loss condition, the packets are almost the same. Then We look into the case of just setting download link loss of receiver 1 to be 10%. Like Table. 15, at that time, compared to receiver 2, receiver 1 receives much more packets. We didn't observe retransmission of the lost packets. Thus, we infer that VSee uses FEC for error correction.

3.3 Delay Performance

The voice delay and video delay performances for those softwares are shown in Table. 16. As Skype employs different network topology for its voice and video transmission, its video and voice are unsynchronized. And the gap is as large as 200ms when we consider the case from the initiator to none-initiator. In centralized P2P architecture, flow from a none-initiator to another none-initiator has to be transmitted to the initiator first. The initiator has to do some process work, like voice mixing, combining video packets from different source into one flow, etc. The result shows that delay from none-initiator to none-initiator is larger than the delay from initiator to none-initiator.

When used in wired condition, one-way video delay of Skype two-party call is about 200ms. Table. 17 shows one-way video delay of Skype two-party call under 802.11b set-up. In the experiment, four computers are all connected to a wireless router which is running in the mode of 802.11b. Two computers are using Skype two-party video call: one serves as pure video sender and the other one serves as pure video receiver. The other two computers induce background flow traffic in the uplink of video call. For the background traffic, with rate of UDP being higher or number of TCP connections being more, one-way video delay becomes higher. Video delay deviation becomes very large when TCP is used as background traffic. Because rate of background flow varies all the time. But when lots of TCP are used, the deviation of video delay becomes smaller because of the multiplex effect.

Table 15: Packet Information under Download Loss 10% For VSee

Sender to Receiver 1			Sender to Receiver 2		
Capture Time(ns)	**IP ID**	**Packet** Length(KB)	**Capture Time(ns)**	**IP ID**	**Packet** Length(KB)
1325542081617251	21852	108	1325542081617139	21850	76
1325542081617251	21853	556	1325542081617140	21851	556
1325542081617435	21855	556	1325542081617369	21854	556
1325542081667571	21857	428	1325542081643944	21856	494
1325542081667572	21858	556			
1325542081667684	21859	556			
1325542081668725	21860	556			
1325542081668810	21861	556			
1325542081669715	21862	556			
1325542081669791	21863	556			
1325542081670704	21864	556			
1325542081670786	21865	556			
1325542081671709	21866	556			
1325542081671778	21867	556			
1325542081672696	21868	476			
			1325542081681239	21869	110
1325542081681312	21871	556	1325542081681241	21870	556
1325542081682219	21873	556	1325542081681425	21872	556
1325542081683218	21875	108			
1325542081683218	21876	556	1325542081682323	21874	556

Table 16: Delay Performance

Softwares		**One-way Video Delay (ms)**	**One-way Voice Delay (ms)**
Google+		180	100
Skype	initiator to none-initiator	230	100
	none-initiator to none-initiator	230	200
iChat	initiator to none-initiator	160	200
	none-initiator to none-initiator	230	270
VSee		235	215

Table 17: One-way Video Delay of Skype Two-party Call in 802.11b

Background Flow Condition	**Delay(ms)**	**Deviation(ms)**
No Background	220	65
UDP 200kbps	220	50
UDP 500kbps	310	155
UDP 1Mbps	320	160
UDP 2.5Mbps	760	175
TCP 1 connection	290	145
TCP 5 connection	630	400
TCP 15 connection	690	350
TCP 30 connection	720	150

4. REFERENCES

[1] *Wireshark*, `http://www.wireshark.org/`.

[2] FFMPEG group, *Ffmpeg project*, `http://www.ffmpeg.org`.

[3] e2eSoft, *VCam: Webcam Emulator*, `http://www.e2esoft.cn/vcam`.

[4] Microsoft Asia Resarch, *Network Emulator for Windows Toolkit (NEWT)*, http://blogs.msdn.com/b/lkruger.
[5] Renovation Software, *Text Grab for Windows*, http://www.renovation-software.com/en/text-grab-sdk/textgrab-sdk.html.
[6] Comfort Software Group, *Free Stopwatch*, http://free-stopwatch.com/
[7] GoldWave Inc., *GoldWave*, http://www.goldwave.com/
[8] Jian Ma, *Comic Enhancer Pro*, http://www.comicer.com/stronghorse/
[9] FlyingSpace, *FlyingSpace PhotoBatch*, http://www.flyingspace.com/
[10] Ralph Richardson, *FreeOCR*, http://www.freeocr.net/
[11] Paul Spoerry, *Hidden features in Google+ Hangouts ĺC updated*, http://plusheadlines.com/hidden-features-googleplus-hangouts/1198/
[12] X. Zhang, Y. Xu, H. Hu, Y. Liu, Z. Guo, Y. Wang, "Profiling Skype Video Calls: Rate Control and Video Quality", to appear in *INFOCOM*, 2012
[13] H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson, *RTP: A Transport Protocol for Real-Time Applications*, http://tools.ietf.org/html/rfc3550
[14] *RTP protocol fields*, http://www.networksorcery.com/enp/protocol/rtp.htm

APPENDIX