Video Telephony for End-consumers: Measurement Study of Google+, iChat, and Skype

Yang Xu, Chenguang Yu, Jingjiang Li and Yong Liu
Department of Electrical and Computer Engineering
Polytechnic Institute of New York University
Brooklyn, NY, USA 11201
{yxu10, cyu03, jli12}@students.poly.edu, yongliu@poly.edu

ABSTRACT

Video telephony requires high-rate and low-delay voice and video transmission. It is challenging to deliver high-quality video telephony to end-consumers through the best-effort Internet. In this paper, we present our measurement study on three popular video telephony systems: iChat, Google+, and Skype. Through a series of carefully designed passive and active measurements, we are able to unveil important information about their design choices and performance, including application architecture, video generation and adaptation schemes, loss recovery strategies, end-to-end voice and video delays, resilience against bursty losses. Obtained insights can be used to guide the design of applications that call for high-rate and low-delay data transmissions under a wide range of "best-effort" network conditions.

1. INTRODUCTION

The Internet has fundamentally changed the way people communicate, ranging from emails, text-messages, blogs, tweets, to Voice-over-IP (VoIP) calls, etc. We are now experiencing the next big change: *Video Telephony*. Video telephony was originally conceived in 1920s. Due to its stringent bandwidth and delay requirements, for years, business customers have been paying high prices to utilize specialized hardware and software for video encoding, mixing and decoding, and dedicated network pipes for video distribution.Video telephony had little success in the end-consumer market, until very recently. The proliferation of video-capable consumer electronic devices and the penetration of increasingly faster residential network accesses paved the way for the wide adoption of video telephony. Two-party video chat and multi-party video conferencing services are now being offered for free or at low price to end-consumers on various platforms. Notably, Google+ Hangout [11], Apple iChat [15], and Skype Video Calls [26] are among the most popular ones on the Internet.

Video conferencing is more challenging than VoIP and video streaming. Compared with voice, video is much more bandwidth-demanding. While Skype encodes high quality voice at $40kbps$, a Skype video call can easily use up $900kbps$ [32]. Compared with video streaming, video conferencing has much tighter delay constraints. While seconds of buffering delay is often tolerable even in live video streaming, in video conferencing, user Quality-of-Experience (QoE) degrades significantly if the one-way end-to-end video delay goes over 350 milli-seconds [17]. To deliver good conferencing experience to end-consumers over the best-effort Internet, video conferencing solutions have to cope with user terminal and access heterogeneity, dynamic bandwidth variations, and random network impairments, such as packet losses and delays. All these have to be done through video generation and distribution *in realtime*, which makes the design space extremely tight. This motivates us to conduct a measurement study on three existing solutions: iChat, Google+, and Skype. Our objective is to investigate *how they address the previously mentioned challenges*, and *how well they do it on the Internet?*. We study the following questions:

1. *What are their video conferencing topologies: peer-to-peer, server-centric, or hybrid of the two?*
2. *How do they generate and adapt video to cope with user heterogeneity and bandwidth variations? Do they encode multiple versions of a user's video and send different versions to different receivers, or they generate multiple video layers and send different subsets of layers to different receivers?*
3. *How are voice and video transmitted to simultaneously meet the high-rate and low-delay requirements? How are voice and video of different users mixed and synchronized?*
4. *How robust are these systems against network impairments? Do they user Forward-Error-Correction (FEC) or Automatic Repeat-reQuest (ARQ)? What is their resilience against bursty packet losses?*
5. *Ultimately, how good is user conferencing experience? what are their perceived video quality, end-to-end voice and video delay, and the synchronization of voice and video?*

It is admittedly challenging and ambitious to come up with conclusive answers for all these questions. It is definitely NOT our intention to benchmark those three systems. And it is NOT of our interest to promote any of them. Instead, the driving-force for our study is to investigate how to deliver video telephony on the Internet through measuring working systems. We are happy to share with the community the findings and insights obtained from our study, which can hopefully shed some lights on the design of an increasing array of applications that call for high-rate and low-delay data transmissions under a wide range of "best-effort" network conditions. The rest of the paper is organized as follows. We briefly talk about related work in Sec. 2. The measurement platform is introduced in Sec. 3. Then we study the application architectures of three systems in Sec. 4. The video generation and adaptation strategies are investigated in Sec. 5. Their voice and video delay performance is measured in Sec. 6. In Sec. 7, we investigate the loss recovery strategies of the three systems, and measure their resilience against bursty losses and long delays. The paper is concluded with a summary of findings in Sec. 8

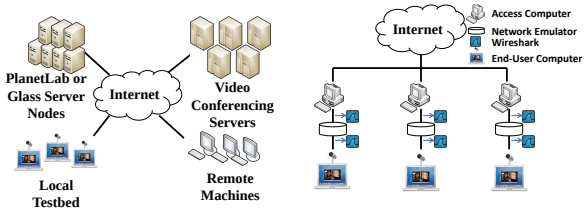
2. RELATED WORK

Most previous measurement work on Skype focused on its VoIP service. Baset *et al* [1] first analyzed Skype's P2P topology, call establishment protocols, and NAT traversal mechanism. Since then, a lot of papers have been published on Skype's overlay architecture, P2P protocol, and VoIP traffic [2, 12]. Some other studies [3, 14, 31] focused on the quality of Skype's voice-over-IP (VoIP) calls. Huang *et al.* investigated Skype's FEC mechanism and its efficiency for voice streaming [14, 31]. In [3], the authors proposed a USER Satisfaction Index model to quantify VoIP user satisfaction. Cicco *et al.* [4] proposed a congestion control model for Skype VoIP traffic. All of these studies only focused on the voice service of Skype, did not consider its video service.

More recently, there are some measurement work on video telephony. Cicco *et al.* [5] measured the responsiveness of Skype video calls to bandwidth variations. They conclude that Skype's response time to bandwidth increase is long. In [32], we conducted an extensive measurement of Skype two-party video calls under different network conditions. Based on the measurements, we propose the models for Skype video calls' rate control, FEC redundancy, and video quality. According to a 2010 survey [18], most of the eighteen surveyed multi-party video conferencing systems before Skype and Google+ are sever-based. The maximum number of conferencing participants supported by each system ranges from 4 to 24.

3. MEASUREMENT PLATFORM

Our video telephony measurement effort was initiated in December 2010, right after Skype introduced its beta service of multi-party video conferencing. Since then, we have been continuously taking measurement of Skype. In June 2011, Google+ Hangout introduced video conferencing service for users in *friend circles*. To obtain a broad view, we extended our study to Google+, as well as, Apple's iChat service. All three systems use proprietary protocols and encrypt data and signaling. There is very limited public information about their architecture, video encoding and distribution algorithms. Through a combination of sniffing packets, capturing statistics from application windows, and monitoring end-to-end video performance, we unveiled important information about their key design choices.



(a) Overall Testbed (b) Local Testbed

Figure 1: Measurement Testbed

As illustrated in Fig 1, our measurement platform consists of three major components: local testbed within NYU-Poly campus network, remote machines of friends distributed over the Internet, and Planetlab [22] and Glass Server nodes [27] at selected locations. Experimental video calls are established either between our local machines, or between local machines and remote friends' machines. Planetlab and Glass Server nodes are employed for active probing to geolocate video conferencing servers. To emulate a video call, we choose a standard TV news video sequence "Akiyo" from JVT (Joint Video Team) test sequence pool. The sequence has mostly head and shoulder movements. It is very similar to a video-call scenario. We inject the video sequence into video conferencing systems using a virtual video camera tool [7]. This ensures the transmitted video content are consistent and repeatable. To emulate a wide range of network conditions, we install software-based network emulator, NEWT[20], on our local machines. It emulates a variety of network attributes, such as propagation delay, random packet loss, and available bandwidth. As illustrated in Fig 1, we use Wireshark [30] to capture detailed packet-level information on both ends of network emulators.Skype, Google+ and iChat all report technical information about video quality through their application windows, such as video rates, frame rates, RTT, et al, [21]. We use a screen text capture tool [23] to capture these information periodically. The sampling interval is 1 second.

4. VIDEO CONFERENCING TOPOLOGY

There are three architectures for networked applications: server-client, peer-to-peer, and hybrid of the two. Skype delivers a scalable VoIP service using P2P, where users connect to each other directly as much as possible. In a video conference, a source encodes his video at a rate much higher than voice, and might have to send the video to multiple receivers. From bandwidth point of view, a pure P2P design might not be sustainable. We now investigate the application architectures adopted by the three video telephony systems.

4.1 Methodology

For each system, we set up video calls and use Wireshark [30] to capture packets sent out and received by each computer. Each user in the conference generates voice and video packets constantly at significant rates. In our Wireshark analysis, we capture those TCP and UDP sessions, whose durations are long enough (at least half of the conference session time) with significant flow rates (larger than 5kbps), as voice or video flow sessions. To get the right topology and differentiate between signaling packets, video packets and voice packets, we conduct three types of experiments. For the first experiment, every 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. This way, we can identify voice flows, video flows, and signaling flows.

4.2 iChat is P2P

iChat is P2P and employs a star topology as shown in Fig. 2(a), where the central hub is the conference initiator, the user who initiated the conference. Only the conference initiator has the right to add people into the conference or close the entire conference. A normal user sends one UDP flow, combining his video and voice information together, to the initiator. At the same time, that the normal user receives other participants' video and voice information through one UDP flow from the initiator. Voice and video are transported using RTP protocol [13]. Participants use UDP port 16402. Normal users only have connection to the initiator. No direct UDP flow exists between normal users.

4.3 Google+ is Server-centric

Google+ video calls, both two-party and multi-party, always use server-centric topology illustrated in Fig. 2(b). Each user sends his video and voice to a dedicated proxy server and also receives others' video and voice from that server. There is no direct transmissions between users. Generally, users choose different dedicated proxy servers. Thus, the proxy servers need to communicate with each other to exchange user's video and voice data. Each user opens four connections with his proxy server on the same server port (Port 19305). Most of the time, these four connections all use UDP. In very rare cases, TCP is used. There is a trick [21] that allows us to access various statistics of Google+. The statistics show that two of the four flows carry video and voice respectively. Google+ also uses RTP protocol to transmit video and voice. The other two flows' payloads conform to the format of RTCP protocol. We infer that those two flows carry signaling information.

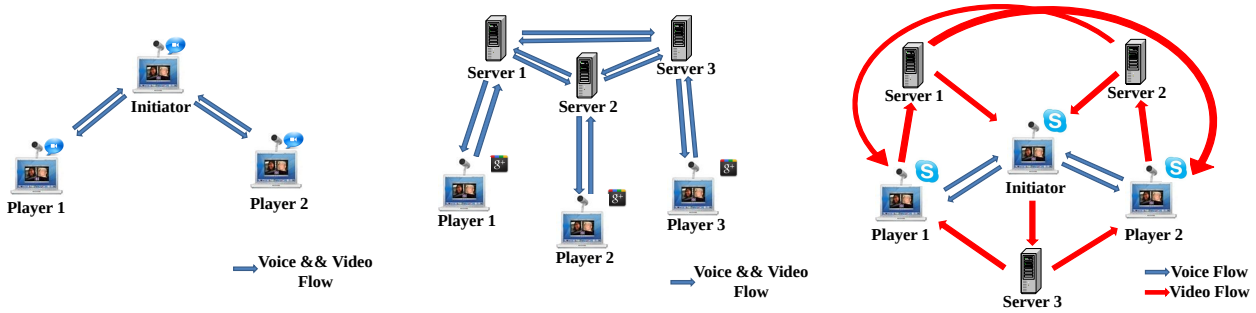
4.4 Skype is Hybrid

For two-party video calls, Skype uses direct P2P transmission for voice and video if the two users can establish a direct connection [32]. When three or more users are involved, the network topology is shown in Fig. 2(c). Voice is still transmitted using P2P. Similar to iChat, the conference initiator acts as the central hub. A normal user uploads his voice to the initiator and downloads other users' voice from the initiator. In a conference with three users, the upload flow rate from a normal user to the initiator is around $40kbps$. And the download rate from the initiator to that user is around $50kbps$, which is less than the total rate of two separate voice flows. This indicates that the initiator might use *sound mixing technique* to combine voices of multiple users into one flow. For video transmission, each user uploads his video to a Skype server, which relays the video flow to all other users in the conference. Different users normally choose different relay servers. All Skype servers encountered in our measurements are all in the same subnet of 208.88.186.00/24.

Skype mostly uses UDP flows for voice and video transmission. TCP is used in very rare cases. Different from Google+ and iChat, Skype does not use RTP and RTCP. In addition to voice and video flows, we also observe some small rate flows between servers and users. We conjecture that these small flows are for signaling.

4.5 Conferencing Server Placement

High video data rates drive Google+ and Skype to employ servers for video relay. If servers are not properly provisioned and selected for users, video relay will incur long delays that significantly degrade the experience of realtime interaction between conferencing users. To determine the server location of Skype and Google+, we set up video conferences with our friends all around the world. Table 1 shows the Google+ server IP addresses used by our friends from different locations. When using a popular geolocation tool Maxmind [19] to locate those IP addresses, we were surprised that all Google+ server IP addresses are geolocated to Mountain View, USA. However, this seems not correct. In Table 1, we also report the measured RTT between our friends' locations and their corresponding Google+ servers. The result shows that those servers are not far from our



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

Figure 2: Video Conferencing Topology of Three Systems

remote friends' locations, except for two cases in Australia. Thus, we infer that Google+ hangout locates servers all around the world.

Table 1: Google+ Hangout Server IP Addresses

Friend Location	Server IP	RTT (ms)
Hong Kong, CHN	74.125.71.127	3.49
Armagh, UK	173.194.78.127	8.88
Rio de Janeiro, BRA	64.233.163.127	9.02
New York, USA	173.194.76.127	14.2
Aachen, DE	173.194.70.127	20.00
Toronto, CA	209.85.145.127	26.76
San Jose, USA	173.194.79.127	28.89
Brisbane, AU	72.14.203.127	147
Canberra, AU	74.125.31.127	147

We did similar experiments for Skype. We found that the server IP addresses from the above locations all locate in 208.88.186.00/24. Maxmind [19] states that those servers all locate in Estonia. We select nodes across the world in PlanetLab to measure RTT to Skype servers in Table 2. Even if we assume that bits propagation speed is $3 \cdot 10^8 meters/sec$, the round-trip propagation time between New York and Estonia is about 44.4ms (11,314km)[16], which is even larger than the RTT got by using Ping in Table 2. Thus, Skype servers can't be located in Estonia. From the ping result in Table 2, we infer that Skype servers are located in some place around New York City. From nodes located in New York and Hong Kong, we also pinged all possible IP addresses in the subnet of 208.88.186.00/24. The RTT results are consistent with the corresponding values from these two locations listed in Table 2. This suggests that those Skype servers are likely in the same physical location.

5. VIDEO GENERATION AND ADAPTATION

In a video conference, each source encodes his video in realtime. To cope with network bandwidth variations, the encoded video rate has to be adapted to the available bandwidth. In multi-party video conferencing, each source has multiple receivers, potentially with different downloading capacities. In a *one-version* design, a source generates single video version that can be

Table 2: RTT between Nodes and Skype Server

Location	RTT (ms)
New York, USA	25.9
Washington, USA	31.3
Vancouver, CA	61.41
San Jose, USA	67.5
London, UK	99.2
Brisbane, AU	266
Canberra, AU	222
Rio de Janeiro, BRA	176
Hong Kong, CHN	226
Saarbrucken, DE	138.00
Guayaquil, EC	111
Oulu, FIN	150.96
Tel Aviv, IL	207
West Bengal, IN	324

downloaded by the weakest receiver, and sends that version to all receivers. It is more desirable for a source to send different receivers different video qualities, maximally matching their download capacities. In a simple *multi-version* design, a source uses different video encoding parameters to generate multiple versions of the same video, and simultaneously upload those versions to the server/receivers. One obvious drawback is that, as end-host, a source might not have enough bandwidth to upload multiple versions. Alternatively, a source can send one copy of his video to a server, which then transcodes it into multiple versions at lower qualities. The third option is the *multi-layer* design, where a source encodes a video into multiple layers, using the recent scalable video coding techniques, such as SVC [25] or MDC [28]. A receiver's perceived video quality increases as more video layers are received. While multiple-layer coded video incurs coding overhead, recent advance in SVC coding has brought down the overhead to 10% [29]. With multi-layer coding, a source only needs to send out all layers using upload bandwidth slightly higher than the one-version design, and realizes the effect of multi-version design by allowing different receivers download different numbers of layers, matching their download capacities. It does not require server transcoding. It is robust against bandwidth variations and packet losses: a basic quality video can be still decoded as long as the base layer is received reliably.

5.1 Methodology

Since there is no public information about their video generation and adaption strategies, we design experiments to trigger video adaptation by manipulating source upload bandwidth and receiver downloading bandwidth. In our local testbed, for all users involved in the conference, we set up one user as the sender, turn on his video and use the other users as purely receivers. We denote such a setting as a *sub-conference*, which is a basic component of the whole conference. As illustrated in Fig. 1, the bandwidth of the sender and receivers can all be set by using network emulator NEWT. We collect video information from the application window. We also record the packets using Wireshark [30] for offline video payload analysis.

5.2 Video Encoding Parameters

To deal with bandwidth variation, these three systems all have the abilities to produce video flows in a large rate range, as shown in Table 3. From the perspective of a viewer, the perceived video quality is mostly determined by three video encoding parameters: resolution, frame rate, and quantization. In our experiments, the ranges of the observed resolution values for all systems are listed in Table 4. The frame rates can vary from 1 FPS (Frame-Per-Second) to 30 FPS for each system. In our experiments, we find that not all systems change all parameters for video adaptation. Table 5 lists which parameters are adapted in each system. Skype adapts all three parameters. We didn't observe the change of quantization in Google+ Hangout and iChat. 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 the resolution is set at the beginning, it will not be changed no matter how we change the bandwidth setting.

Table 3: Video Rate Ranges

System	Range
iChat	49 kbps - 753 kbps
Google+	28 kbps - 890 kbps
Skype	5 kbps - 1200 kbps

Table 4: Resolution Ranges

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

Table 5: Varied Parameters

System	Resolution	FPS	Quantization
Skype	✓	✓	✓
Google+	✓	✓	
iChat		✓	

5.3 Adaptation for Receiver Heterogeneity

When the upload link bandwidth of a sender varies, the video rate out of the sender changes correspondingly. Generally, for these three systems, the higher the upload link bandwidth, the larger the sending rate. When the upload bandwidth is too small, those systems will automatically shutdown the video. This shows that those systems have their own network probing algorithms to determine the video quality to be encoded for transmission.

Next, we set receivers' download bandwidths to different levels. We found that iChat uses *one-version encoding:* heterogenous receivers always receive the same video version, and the receiver with download bandwidth limit set determines the video quality sent out by the sender. No video trans-coding, nor layered coding, is employed.

For Skype, when receivers' bandwidth differences are large, it employs *source-side multi-version encoding.* For example, in a video conference involving one sender and three receivers, receiver 1, receiver 2 and receiver 3's download bandwidths are set to be 150 kbps, 400 kbps and 2000 kbps respectively. The sender generates three video versions: version 1 with rate 70 kbps, version 2 with rate 200 kbps and version 3 with rate 500 kbps. Relay server gets these three video versions from the sender. Then, it distributes one of these three video versions to the appropriate receiver. In this case, receiver 1, 2, 3 get video version 1, 2, 3 respectively. No server-side transcoding was observed. In our experiments, the largest video version number that sender can generate is 3. When receiver bandwidth diversity is small, Skype use one-version encoding and sends to all receivers the same video quality.

Google+ always give different video qualities to heterogenous receivers. For example, we only limit the download bandwidth of one receiver, say 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 frame rate is 30 FPS. For video flow to receiver 1, the received rate is 386.9 kbps, video resolution is $640 * 360$, and frame rate is 14 FPS. And for video flow to receiver 2, the received rate is 168.6 kbps, video resolution is $320 * 180$, and frame rate is 14 FPS. The experiment environment is not fully controlled, as flows between our experiment machines and Google+ servers traverse the Internet, have to compete with cross traffic. This explains that the received quality on receiver 1 is lower than the original video sent out by the sender, although we didn't add any bandwidth constraint on receiver 1. At this time, both receiver 1 and receiver 2 receive consistent and acceptable video quality.

To gain more insight about how the two video quali-

ties are generated, we examine the packets captured on the sender and receivers. Both Google+ and iChat use RTP[13] protocol for their video and voice transmission. Even though video payload cannot be directly decoded to reveal how video is generated, several fields in RTP headers enable us to infer the coding strategy employed by Google+. According to RTP header format specification [24], "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+ Hangout, packets from the same sender in one flow form a unique sequence. "Timestamp" field reflects the sampling instant of the first octet in the RTP data packet. In Google+ RTP packet traces, we observe that a flight of packets with consecutive sequence numbers carry the same timestamp. We infer that those packets belong to the same video frame. The common timestamp is indeed the generation time of the frame. If a packet is the last one for a frame, then the "Marker" field for that packet is set to 1; otherwise the value is 0. Thus, we infer that the "Marker" field can used to identify boundary between video frames. Since different machines use different initial sequence numbers, to match video frames between the sender and receivers, we instead use the "Timestamp" and "Marker" field.

In Table 6, we match the RTP packets sent out by the sender and received by the two receivers based on their timestamps, marker (labeled 'M') and length. (Packets on the same row have the same values in these three fields.) In aligned table, the sender sends out 5 frames, both receiver 1 and receiver 2 only receive 2 frames. While receiver 1 receives all packets of those two frames, 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 consecutive. The lost packets/frames are not due to packet losses. Instead, it is the video relay server who decides not to send those frames/packets to receivers based their bandwidth conditions. Note that, both receivers can decode the video with decent quality suggests that Google+ employs multi-layer coding. The fact that receiver 1 can decode the video even though it lost some frames suggests that layered coding used by Google+ achieves *temporal scalability*: video frames are temporally grouped into layers, such that a high frame-rate video generated by the sender can be still decoded at a lower frame-rate by a receiver even if a subset of frames/layers are dropped. The fact that receiver 2 can decode video even though it lost some packets within each frame suggests that layered coding used by Google+ also achieves *spatial scalability*: a video frame is encoded into spatial layers at the sender, such that even if some spatial layers are lost, the receiver can still decode the frame at lower resolution and larger quantization.

In the experiment, we also observe that no matter how low the download bandwidth of a receiver is, the received video resolution could only be one quarter of the resolution of the original video. This shows that the encoded video only has two spatial layers. This is reasonable as spatial layers inducing much higher encoding overheads compared to temporal layers. Thus, the number of spatial layers could not be too large.

Table 6: Packet Payloads in Google+

M	Timestamp	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	

6. VOICE AND VIDEO DELAY

To facilitates realtime interactions between users, video conferencing systems have to deliver voice and video data with short delays. User conferencing experience degrades significantly if the one-way end-to-end video delay goes over 350 milli-seconds [17]. In this section, we develop measurement to study the user delay performance in the three systems.

6.1 Methodology

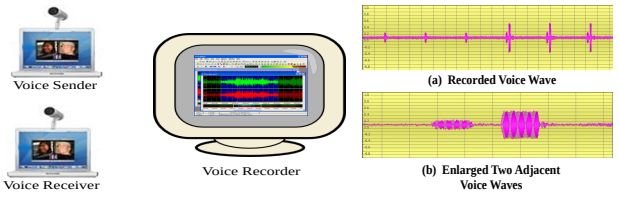
The end-to-end voice/video delay perceived by a user is the sum of delays incurred by realtime voice/video capturing, encoding, transmission, decoding, and rendering. We can divide the end-to-end delay into four portions. Let T_e be the video/voice capturing and encoding delay at sender, T_n be the one-way transmission delay on the network path between sender and receiver, T_s be server or super node process time (0 if there is no server or super node involved) and T_d be the video or voice decoding and playback delay at the receiver. Thus, the one-way voice (video) delay is:

$$T = T_e + T_n + T_s + T_d. \quad (1)$$

Obviously, it is not sufficient to just measure the network transmission delay T_n. We therefore develop our

end-to-end delay measurement by emulating a real user's experience.

6.1.1 One-way Voice Delay

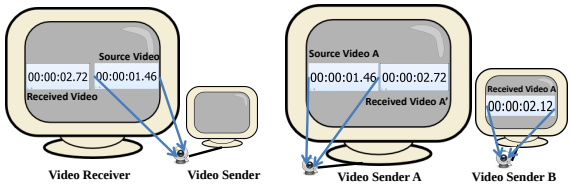


(a) One-way Voice Delay (b) Voice Waves Detail

Figure 3: Testbeds in the experiment

To record one-way voice delay, we employ a repeatable "Tick" sound as the voice source. In our local testbed, we set up three computers close to each other, as illustrated in Fig. 3(a). One computer is the voice sender, another one is the voice recorder. The sender repeatedly sends the "Tick" sound to the receiver. A third computer emulates a user and "hears" (records) the sound injected to the voice sender and the sound coming out of the receiver using a sound recording software[10]. We can visually analyze captured sound signal in that software. Fig. 3(b) shows a recorded sound wave sample. In the upper subfigure, we observe two sequences of impulses at different amplitudes. Since we set the volume of the sender's speaker significantly lower than the receiver's speaker, the impulses with smaller amplitude correspond to the repeated "Ticks" sent by the sender, the impulses with larger amplitude are the received "Ticks" on the receiver. On the sender, we set the time interval between two "Tick"s to be 1,000 ms, larger than the expected voice delay, so that we can match a large impulse with the preceding small impulse. The lower subfigure is the zoom-in view of the two adjacent sound impulses, with each impulse is indeed a waveform segment with rich frequency components. We use the time-lag between the first peaks of two adjacent segments as the one-way voice delay.

6.1.2 One-way Video Delay



(a) One-way Video Delay (b) Round-trip Video Delay

Figure 4: Testbeds in the experiment

To measure video delay, we similarly emulate a user's video experience by simultaneously viewing (recoding) the original video on a sender and the received video on a receiver using a video capturing program. As illustrated in Fig. 4(a), we set up two computers side-by-side in our local testbed, one as sender, the other as receiver. We run a stopwatch program [8] on the **receiver** side, and focus sender's video camera on the stopwatch window on the receiver's screen. The sender transmits the captured stopwatch video from the receiver screen back to the receiver using one of the three conferencing systems. There are two stopwatch videos on the receiver's screen: the original stopwatch video, and the copy captured by the sender then transmitted back through the video conferencing system. At any given time, the difference between the two clocks is the one-way end-to-end video delay perceived by the receiver. For the example in Fig. 4(a), the original stopwatch shows time "00:00:02.72" second and the received copy of the stopwatch shows time "00:00:01.46" second. Thus, the video delay in this case is 1260 ms. We use programming script running on the receiver side to capture snapshots of those two clocks 10 times each second. After getting these pictures, we use softwares [6] to first turn the 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 [9]. It should be noted that if the received video quality is not good, OCR cannot decode the numbers correctly. We skip those undecodable samples in our delay calculation. The undecodable picture ratio can be also used to infer the received video quality.

6.1.3 Round-trip Video Delay

The previous method does not work if the sender and the receiver are not in the same location. We develop another method to measure the *Round-trip video delay* between two geographically distributed computers. As in Fig. 4(b), we run a stopwatch program on user A. A uses his camera to capture the stopwatch as its source video, and transmits it to user B by using one of the three conferencing systems. The received stopwatch video is now displayed on B's screen. Then B focus its camera on the received video from A, and send the captured video it back to A by using the same conferencing system. Then on A's screen, we can observe two stopwatch videos: the original one, and the one that is first sent to B, then recaptured and sent back to A by B. The clock difference between the two stopwatches on A reflects the summation of one-way video delay from A to B, and the one-way video delay from B back to A, in other words, the *Round-trip video delay*. If we assume the two directions are symmetric, then the one-way video delay between A and B is roughly half of the Round-trip delay. After the cameras are A and B are

properly positioned, we only need to take screen shots on A. We can further use the same approach as in the one-way video display case to process the data to get the delay samples.

6.2 One-way Delay Performance

We first test the one-way delay between users in our local testbed. All user computers are connected to the same router. The transmission delay between them is almost negligible. The voice delay and video delay

Table 7: One-way Delay Performance (ms)

Systems		Video	Voice
Google+		180	100
Skype Two-Party		156	110
Skype Multi-party	initiator to normal	230	130
	normal to normal	230	190
iChat Two-Party		220	220
iChat Multi-party	initiator to normal	220	220
	non-initi. to non-initi.	270	270

performances for three systems are shown in Table 7. For Google+, the average video delay is 180 ms, larger than the voice delay 100 ms. Since Google+ use server-centric architecture, users' video and audio are always first sent to the Google+ server, then come back. The round trip network delay between our local testbed and the allocated Google+ server is 14 **ms**. The addition delays is due to voice and video processing on both ends. It is reasonable that video processing takes longer.

Skype two-party call takes direct P2P transmission for voice and video. The network delay is almost zero. The one-way delay is mostly due to the processing delay. It suggests that voice and video processing can take a significant portion of delay budget of good quality video conferencing.

As studied in Section 4.4, Skype multi-party call employs different network topologies for voice and video transmissions. Voice takes direct P2P transmission, while video has to be relayed by servers. For P2P voice transmission, conference initiator is the central hub. Voice from a non-initiator to another non-initiator has to be transmitted to the initiator first. The initiator has to do some processing, like voice mixing and recoding. The processing occupies a lot of time. Thus, the delay between an initiator and a normal user is shorter than the delay between two normal users. Since video has to be sent to Skype servers first then come back, the video delay is larger then voice. Video and voice are unsynchronized. And the gap is as large as 200ms when we consider the case from the initiator to a normal user.

iChat combines video and voice in one RTP flow. From its two-party result, we find that its video delay and voice delay are synchronized. For multi-party function, it employs centrialized P2P architecture for video and voice transmission. When voice and video flows are transmitted from the initiator to normal users, delay performance is the same as the two-party case. But when communications happens between non-initiators, saying between user 1 and user 2, the packet flow from user 1 should first go to initiator. Initiator needs to combine all packets destined to user 2 in one RTP flow. These processing needs some extra time. Thus, the delay between non-initiators are larger.

Table 8: 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

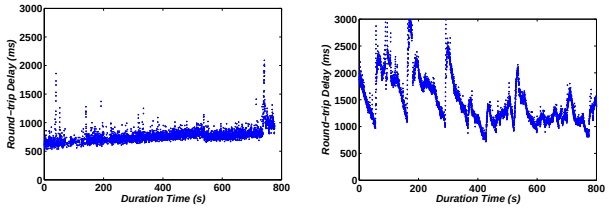
Table 8 shows one-way video delay of Skype two-party call in our local WiFi network. In the experiment, four computers are all connected wirelessly to a 802.11b router. Two computers set up a Skype two-party video call. The other two computers induce background traffic **to** video call. We experiment with both UDP and TCP background traffic. As the rate of UDP or the number of TCP connections increases, the packet transmission delay increases in our local WiFi network. The one-way video delay increases consequently. Video delay deviation becomes very large when TCP is used as background traffic. But when lots of TCP flows are used, the deviation of video delay becomes even smaller because of the traffic multiplexing effect.

6.3 Round-trip Video Delay

We also measure the Round-trip Video Delay (video RTT) between Hong Kong and New York when using Google+ Hangout and Skype Multi-party Systems. Now video round-trip time is the outcome of video encoding, video decoding, packet transmission, error correction, etc. From Figure. 5, we can observe that video RTT using Google+ is much smaller than the RTT using Skype Multi-party. Google+'s mean video RTT is only about 776 ms, stand deviation is about 123ms. To the contrary, Skype's mean video RTT is about $1,467$ms, the standard deviation is 473.6ms. When we consider the video One-way delay by simply dividing video RTT by 2, Google+'s one-way video delay is only 388 ms.

Skype's delay performance is much worse. From the topology analysis in Sec. 4, we know that for each video source, Skype just uses a single server to do packet relay. Video flow between relay server and all users need to go through the public Internet. To the contrary, Google+ have servers located all around the world, and a user is mostly connected to a close-by server. The transmission between Google+ video servers likely go through their own private network with good QoS guarantee. This

design can makes the end-to-end network loss and delay much smaller. In addition, Google+'s error correction method is more efficient than Skype upon packet losses, which also lead to shorter video delay. We will discuss this in more detail in the following section.



(a) Google+ Hangout (b) Skype Multi-party

Figure 5: Video Round-Trip-Time between Hong Kong and New York

7. ROBUSTNESS AGAINST LOSSES

One of the main challenge of delivering video telephony over the best-effort Internet is to cope with unpredictable network impairments, such as congestion delay, random or bursty packet losses. To achieve reliability in realtime streaming, the conventional wisdom is to use packet level forward error correction (FEC) coding, instead of retransmissions, which would incur too much delay. Unfortunately, in video conferencing, to avoid long FEC encoding and decoding delays, FEC blocks have to be short. This largely reduces the FEC coding efficiency and its robustness against bursty losses. If the round-trip network delay between two machines is short, e.g. 20 ms between our local testbed and Google+ servers, retransmissions might be affordable within a end-to-end video delay budget of 350ms. Unlike FEC, retransmission adds redundancy only as needed, and hence is more bandwidth-efficient. Redundant retransmissions can also be used to protect important packets against bursty losses. In this section, we investigate how the three systems recover from packet losses, how robust they are against random and bursty losses.

7.1 Methodology

We set up multi-party conferencing using machines in our local testbed. We conduct two types of experiments by injecting packet losses using network emulators. One is to add upload losses on the video sender side. The other one is to add download losses on only one of the receivers. Because we capture packets both before and after packet losses, 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. At the same, we monitor the application window to collect video rate and total rate statistics to figure out the redundant transmission ratio.

7.2 Skype uses FEC

In our Skype experiments, we never identified any retransmission of lost packets. From Skype's technical window, we can easily observe a gap between the total data rate and video rate. Previous studies [31, 32] suggests that Skype employs aggressive Forward Error Correction (FEC) coding for VoIP and two-party video calls. 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. We define the FEC redundancy ratio ρ as the ratio between the redundant traffic rate and throughput:

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

The experiment results of adding random upload losses is shown in Table 9. Surprisingly, the sender always add significant redundancy even if we don't introduce any additional packet losses. (Note, since the upload flow traverses the Internet to reach Skype server, the sender might still see some packet losses. Since we don't know the actual loss rate, we can't get a precise FEC model for Skype's multi-party calls similar to the model for the two-party calls in [32].) On the other hand, the redundancy ratios on the two receivers are pretty low, even though the downloading flows also have to traverse the Internet. One possible explanation is that Skype tries hard to protect the video uploaded by the source, because any lost video during the upload has quality implication on all receivers. As we introduce additional losses, the video rate goes down significantly, and the FEC ratio increases. However, due to the initial high FEC ratio, the increase trend is not obvious when we further increase the loss rate. When the loss rate is really high, the Skype significantly drops its sending rate. This is consistent with the observation for Skype two-party call in [32].

The results for download losses are shown in Table. 10. We only add random download losses to receiver 1. The FEC ratio is much less aggressive than in the upload loss case. But there is still the trend that as the loss rate becomes higher, more FEC packets are added into the video flow of receiver 1. In Table 9, we can also observe that quite often the download flow rates on the two receivers are lower than the upload rate of the sender. This scenario suggests that the relay server first removes FEC packets of uploaded video, and then adds new FEC packets for each download flow. Results of Table 10 shows that the two receivers in different conditions receive the same video rate with different FEC ratio. Thus, we infer that relay server monitors network conditions of receivers and calculate the different FEC ratios for different receivers.

7.3 Google+ uses Selective Persistent Retransmission

In Section 4.3, we showed that a Google+ user only

Table 9: FEC Adaptation at Skype Sender Side

Video Rate (kbps)	Video Sender Side			Video Receiver 1		Video Receiver 2	
	Upload Loss Ratio	Upload Rate (kbps)	FEC Ratio ρ_s	Received Rate (kbps)	FEC Ratio ρ_r	Received Rate (kbps)	FEC Ratio ρ_r
513.48	0	974.00	0.47	542.62	0.05	542.56	0.05
474.26	0.02	1108.00	0.57	519.61	0.09	522.05	0.09
460.37	0.05	1019.50	0.55	487.84	0.06	488.19	0.06
225.05	0.08	496.57	0.55	241.80	0.07	241.32	0.07

Table 10: FEC Adaptation at Skype Relay Server Side

Video Rate (kbps)	Video Sender Side		Video Receiver 1			Video Receiver 2	
	Upload Rate (kbps)	FEC Ratio ρ_s	Download Loss Ratio	Relay Send Rate (kbps)	FEC Ratio ρ_r	Received Rate (kbps)	FEC Ratio ρ_r
513.48	974.00	0.47	0	542.62	0.05	542.56	0.05
478.52	1163.87	0.59	0.02	653.40	0.27	505.43	0.05
440.94	955.72	0.54	0.05	949.73	0.54	465.82	0.05
343.73	821.43	0.58	0.08	824.39	0.58	363.88	0.06

interact with a Google+ server for video upload and download. Thus, we only focus on one video sender and one video receiver in our experiments. We first set up a machine as video sender and inject random packet losses to its uplink. Then we capture the packets before and after the network emulator. Surprisingly, Google+ still offers reasonable video quality under random download losses of 40%. Since we get packets both before and after the loss module, we can identify the lost packets. In our packet trace analysis, if two packets have the same "Timestamp" and "Sequence Number", we treat them as redundant transmissions of the same packet. (Payload analysis shows that the bit information of those matched packets are almost the same except for several bits). Thus, we can identify retransmissions of lost packets. Detailed results are presented in Table. 11. As more losses are injected, more retransmissions appear in the flow. However, the sender video FPS doesn't change too much when we increase upload loss from 0% to 20%. If we calculate video rate as total rate minus retransmission rate, Google+ maintains stable video rate under a wide range of packet losses. Google+ strives to maintain a high video upload rate even under high packet losses. This is similar to Skype's high FEC protection for upload video. The *recovery ratio* is defined as the fraction of lost packets eventually received by the receiver. It shows that Google+ is implementing selective retransmission, only half of lost packets are recovered. The persistent ratio defines the fraction of packets that get retransmitted at least once eventually received by the receiver. A persistent ratio of 1 means that if Google+ decides to retransmit a packet, it will persistently retransmit it until it is received successfully. Since Google+ uses layered video coding, where packets from lower layer video packets are more important for decoding. We conjecture Google+'s *selective persistent retransmission* will be applied to packets from lower video layers first.

The results of adding download losses on the receiver is shown in Table 12. It shows a different behavior. The received video frame rate, the total rate, and video rate inferred from retransmission rate all decrease as the download loss rate increases. This suggests that Google+ servers use packet loss as a network congestion signal. As packet loss rate increases, it not only retransmits important lost packets, it also proactively reduce the number of video layers to be sent to a receiver. The recovery ratio and persistent ratio are more or less the same as the upload loss case. Because we couldn't control the network condition of Internet, retransmission under packet loss of 0% is still being observed.

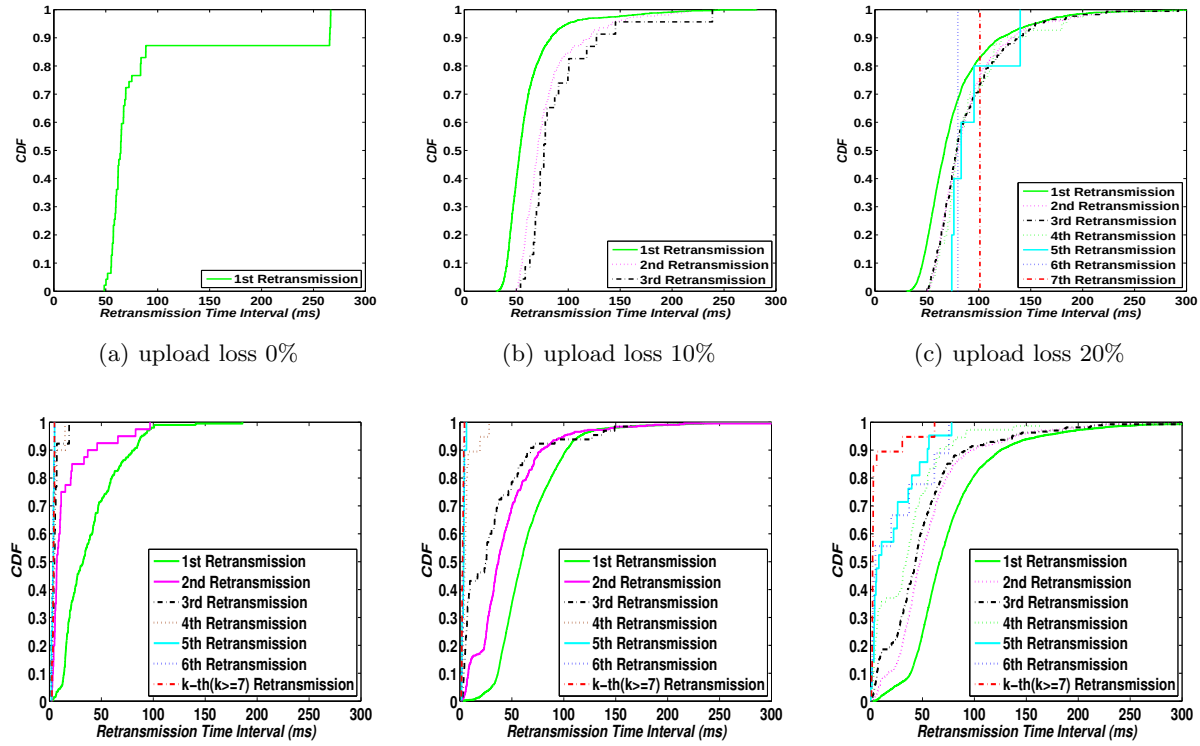
Table 11: Retransmission in the Upload Link of Google+ Hangout

Loss	Frame Rate	Total Rate (kbps)	Retrans Rate (kbps)	Recovery Ratio	Persistent Ratio
0	29.97	826.0	0.9	-	-
0.05	29.98	836.9	24.2	0.46	1.00
0.10	29.97	885.7	52.1	0.46	1.00
0.20	29.98	857.2	101.4	0.45	1.00

Table 12: Retransmission in the Download Link of Google Plus Hangout

Loss	FPS Rate	Total Rate (kbps)	Retrans Rate (kbps)	Recovery Ratio	Persistent Ratio
0	27.91	810.8	4.4	-	-
0.05	27.07	827.3	35.2	0.51	1.00
0.10	20.158	744.3	67.4	0.60	1.00
0.20	19.231	677.7	116.4	0.64	1.00

To gain more insights about Google+'s loss recovery strategy, we collect more statistics about its packet retransmissions. Table 13 lists how many retransmissions Google+ attempted to reliably transfer a packet at different packet loss rates. Most of the time, the retransmission will be successful with one or two tries. Sometimes, we do observe it tries to retransmit a packet many times, with the highest up to 18 times. We define the *k-th retransmission interval* as the time lag between the kth retransmission of and $k-1$th retransmission of the same packet (the original transmission is considered as the 0th retransmission). Table 14 presents the mean and standard deviation of retransmission intervals. For



(a) upload loss 0% (b) upload loss 10% (c) upload loss 20%

(d) download loss 0% (e) download loss 10% (f) download loss 20%

Figure 6: Retransmission Time Interval under loss variation for Google Plus Hangout

Table 13: Retransmission Probability in Google+

Case	1st	2nd	3rd	4th	5th	6th	k-th ($k \geq 7$)
Downlink loss 0	0.8058	0.1311	0.0146	0.0097	0.0049	0.0146	0.0194
Downlink loss 0.05	0.8845	0.0958	0.0101	0.0032	0.0032	0.0005	0.0027
Downlink loss 0.10	0.8677	0.1125	0.0140	0.0027	0.0003	0.0003	0.0024
Downlink loss 0.20	0.7691	0.1793	0.0376	0.0100	0.0023	0.0008	0.0010
Uplink loss 0	1.00						
Uplink loss 0.05	0.9492	0.0477	0.0023	0.0008			
Uplink loss 0.10	0.8963	0.0947	0.0090				
Uplink loss 0.20	0.7996	0.1620	0.0293	0.0080	0.0009	0	0.0002

reference, the RTT between our machines and Google+ server is 14 ms. And the CDFs of retransmission intervals are shown in Fig. 6. 60% of the first 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 the server does retransmission to video receivers, the N-th ($N \geq 5$) retransmission happens only about 5ms after the previous retransmission. This batch retransmission strategy is likely used to deal with bursty packet losses.

7.4 iChat's Retransmission Strategy

iChat also uses retransmission to recover from packet losses. We did similar measurement for iChat as for Google+. iChat doesn't employ persistent retransmission. Most of the time, it just tries to do retransmission once as shown in Table 15.

The mean retransmission intervals are reported in Table 16. Since we set machines in the same subnetwork, the RTTs between them are only about $2-4$ ms. Even though iChat waits for 30+ ms for the first retransmission, the second retransmission happens only $3ms$ after the first retransmission. Due to space limit, more results about iChat retransmissions are available from our technical report.

Table 15: Retransmission Probability in iChat

Case	1st	2nd
Downlink loss 0.02	1.00	
Downlink loss 0.10	1.00	
Uplink loss 0.02	0.9960	0.0040
Uplink loss 0.10	0.9975	0.0025

7.5 Robustness of Conferencing Quality

We use end-to-end video delay defined in Section 6 as a measure of the delivered conferencing quality. If the video can be continuously delivered to the receiver in realtime fashion, the measured video delay from the

Table 14: Mean Time and Standard Deviation(SD) for Retransmission in Google+

Case	1st		2nd		3rd		4th		5th		6th		k-th ($k \geq 7$)	
	Mean (ms)	SD (ms)	Mean (ms)	SD (ms)	Mean (ms)	SD (ms)	Mean (ms)	SD (ms)	Mean (ms)	SD (ms)	Mean (ms)	SD (ms)	Mean (ms)	SD (ms)
Downlink loss 0	39.35	27.52	15.97	21.55	5.54	4.47	4.86	3.87	3.41	1.11	3.40	1.08	3.91	0.85
Downlink loss 0.05	80.35	122.62	40.72	36.66	16.33	34.32	4.56	2.70	4.76	6.00	2.69	1.18	2.17	0.90
Downlink loss 0.10	66.03	34.10	45.14	42.45	32.36	36.21	6.70	6.37	3.41	1.39	3.44	0.89	2.64	0.83
Downlink loss 0.20	77.84	57.05	58.59	57.97	53.16	64.31	35.14	31.59	20.86	22.85	23.06	28.58	6.87	14.83
Uplink loss 0	90.14	68.66												
Uplink loss 0.05	64.69	31.12	70.47	14.90	61.56	9.13	56.88	0						
Uplink loss 0.10	60.28	26.82	79.62	30.05	89.63	39.44								
Uplink loss 0.20	77.49	38.00	89.63	36.00	90.15	36.32	90.89	34.34	93.72	27.15	79.83	0	101.08	0

Table 16: Mean Time and Standard Deviation(SD) for Retransmission in iChat

Case	1st		2nd	
	Mean (ms)	SD (ms)	Mean (ms)	SD (ms)
Downlink loss 0.02	39.06	26.31		
Downlink loss 0.10	35.27	26.66		
Uplink loss 0.02	40.09	29.79	4.45	0.15
Uplink loss 0.10	39.20	31.69	6.86	9.05

stopwatch video should be consistently low. Since the original stopwatch video is continuously advance, losses of video frames on the receiver side will lead to large measured video delays. Additionally, if the received stopwatch video has bad video quality. Our text recognition tool, OCR, cannot decode the clock reading. We can use recognition ratio of OCR as an indirect measure of the received video quality. In this section, we compare the robustness of the three systems against bursty loss and long delays.

Packet losses can be bursty, especially on wireless links. We use emulator to generate bursty losses to the download link. For each loss burst, $4-5$ packets will be dropped in batch. Table 17 compares the clock recognition probability under different bursty loss rates. The clock video transmitted in iChat quickly become uncodable, this is consistent with our own video perception. The recognition probability remains high in Google+ and Skype, indicating that their delivered quality is stable up to 4% bursty losses. Figure 7 compares the one-way video delay performances of Google+ and Skype.

Table 17: Digital Clock Recognition Probability

Case	iChat	Google+	Skype
No Loss	0.90	0.86	0.88
2% Bursty loss	0.46	0.90	0.90
4% Bursty loss	0.10	0.76	0.92

When we don't introduce any additional loss, both Google+ and Skype are able to maintain a low and stable video delay, with average about 200 ms. As we introduce additional bursty losses, Skype incurs a large and highly-varying one-way video delays, as shown in Figure. 7(e) and Figure. 7(f). This suggests that the FEC scheme adopted by Skype cannot efficiently recover from bursty losses. At frame rate of 30 frames/second, each lost frame leads to an additional video delay of 33 ms. Consecutive lost frames leads to delay pikes in the figures. Google+'s selective and persistent retransmissions can recover from bursty losses well. It can always do batch retransmissions for packets from the base layers upon bursty losses. The receiver can decode the video even if only the base layer is received. As a result, Google+ is able to maintain low video delays up to 4% bursty losses in Figure. 7(b) and Figure. 7(c). Compared with employing FEC, one main drawback of using retransmission is that it does not work well when the network delay between sender and receiver is large. To investigate the impact of network delay on the efficiency of retransmission, we set constant random loss to the download link of Skype and Google+, then we change RTT by introducing propagation delays using network emulator. Figure 8 compares the mean and variance of one-way video delay of both systems at different RTTs. Skype's one-way video delay curves are almost the same at no loss and 8% random losses. As RTT increases by Δ, the one-way video delay increases by roughly $\frac{\Delta}{2}$. This suggests that Skype's FEC efficiency is almost independent of RTT. To the contrary, when no loss is induced, Google+'s one-way video delay curve is almost the same as Skype's curve. Almost no retransmission is needed. When the random loss is set to be 8%, the video delay curve ramps up quickly, with increasingly high variances. This suggests that FEC is preferable over retransmissions if RTT is large, loss is random, and loss rate is not too high.

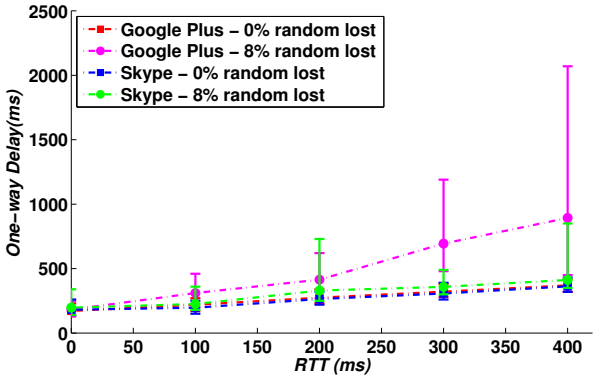
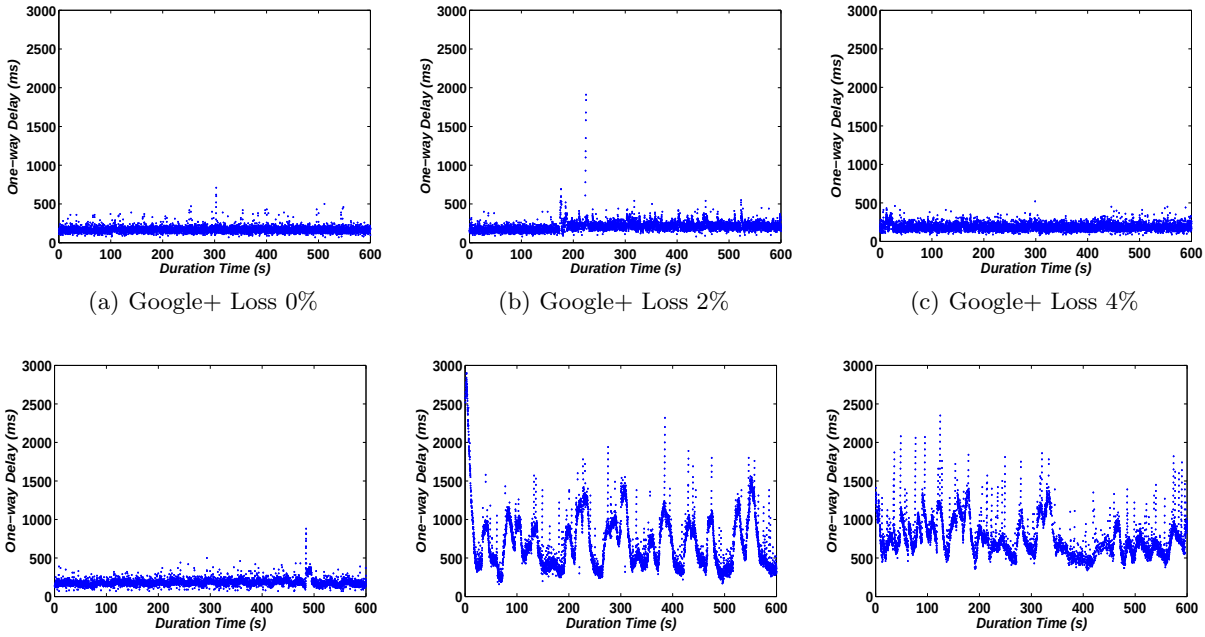


Figure 8: Google+ and Skype Delay Performances under RTT variation



(a) Google+ Loss 0% (b) Google+ Loss 2% (c) Google+ Loss 4%

(d) Skype Loss 0% (e) Skype Loss 2% (f) Skype Loss 4%

Figure 7: One-way Delay For Google+ and Skype under Different Bursty Loss Probabilities

8. CONCLUSION

In this paper, we present our measurement study on three popular video telephony systems for end-consumers. Through a series of carefully designed passive and active measurements, we were able to unveil important design choices of the three systems. Our major findings are summarized below.

- While P2P is promising for voice conferencing and two-party video calls, multi-party video conferencing still relies heavily on bandwidth-rich server infrastructures.
- In server-based solution, server location plays an important role not only in the delivered user delay performance, but also in loss recovery and resilience.
- Compared with multi-version video coding, scalable video coding can simultaneously address user access heterogeneity and network bandwidth variations with low bandwidth overhead.
- Various voice/video processing delays, incurred in capturing, encoding, decoding and rendering, account for a significant portion to the end-to-end delays perceived by users.
- When voice and video are relayed by servers, per-hop retransmissions along the relay path can effectively recover from high packet loss rate. Content-aware selective retransmissions further enhance the robustness of conferencing quality.
- FEC is not efficient for realtime conferencing with bursty losses. Batched retransmissions can be used to recover from bursty losses.

Our study demonstrated the benefits of jointly designing video generation, adaptation and distribution to deliver high-quality video telephony over the best-effort Internet. Motivated by our first-hand experience with the three measured systems, we are interested in developing new joint design solutions in the extremely tight and challenging design space.

9. REFERENCES

[1] S. A. Baset and H. G. Schulzrinne. An analysis of the skype peer-to-peer internet telephony protocol. In *Proceedings of IEEE INFOCOM*, pages 1–11, April 2006.

[2] D. Bonfiglio, M. Mellia, M. Meo, N. Ritacca, and D. Rossi. Tracking down skype traffic. In *Proceedings of IEEE INFOCOM*, pages 261–265, Apr 2008.

[3] K. Chen, T. Huang, P. Huang, and C. Lei. Quantifying skype user satisfaction. In *Proceedings of ACM SIGCOMM*, volume 36, Oct 2006.

[4] L. D. Cicco, S. Mascolo, and V. Palmisano. A mathematical model of the skype voip congestion control algorithm. In *Proceedings of IEEE Conference on Decision and Control*, Dec 2008.

[5] L. D. Cicco, S. Mascolo, and V. Palmisano. Skype video congestion control: an experimental

investigation. *Computer Networks*, 55(3):558 – 571, Feb 2011.

[6] Comic Enhancer Pro. Homepage. http://www.comicer.com/stronghorse/.

[7] e2eSoft. Vcam: Webcam emulator. `http://www.e2esoft.cn/vcam/`.

[8] Free Stopwatch. Homepage. http://free-stopwatch.com/.

[9] FreeOCR. Homepage. http://www.freeocr.net/.

[10] GoldWave. Homepage. http://www.goldwave.com/.

[11] Google+. Homepage. `https://plus.google.com/`.

[12] S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proceedings of the 5th International Workshop on Peer-to-Peer Systems*, pages 1–6, Santa Barbara, CA, February 2006.

[13] H. Schulzrinne, S. Casner, R. Frederick and V. Jacobson. Rtp: A transport protocol for real-time applications. http://tools.ietf.org/html/rfc3550.

[14] T. Huang, K. Chen, and P. Huang. Tuning skype redundancy control algorithm for user satisfaction. In *Proceedings of IEEE INFOCOM*, pages 1179–1185, April 2009.

[15] iChat. Homepage. `http://www.apple.com/macosx/apps/all.html#ichat`.

[16] Infoplease. Homepage. http://www.infoplease.com/atlas/calculate-distance.html.

[17] J. Jansen, P. César, D. C. A. Bulterman, T. Stevens, I. Kegel, and J. Issing. Enabling composition-based video-conferencing for the home. *IEEE Transactions on Multimedia*, 13(5):869–881, 2011.

[18] Y. Lu, Y. Zhao, F. A. Kuipers, and P. V. Mieghem. Measurement study of multi-party video conferencing. In *Networking*, pages 96–108, 2010.

[19] MaxMind. Homepage. http://www.maxmind.com/.

[20] Microsoft Resarch Asia. Network Emulator for Windows Toolkit (NEWT). `http://blogs.msdn.com/b/lkruger`.

[21] Paul Spoerry. Hidden features in google+ hangouts. http://plusheadlines.com/hidden-features-googleplus-hangouts/1198/.

[22] PlanetLab. Homepage. http://www.planet-lab.org/.

[23] Renovation Software. Text grab for windows. `http://www.renovation-software.com/en/text-grab-sdk/textgrab-sdk.html`.

[24] RTP protocol Fields. Homepage. http://www.networksorcery.com/enp/protocol/rtp.htm.

[25] H. Schwarz, D. Marpe, and T. Wiegand. Overview of the Scalable Video Coding Extension of the H.264/AVC Standard. *IEEE Trans. Circuit and System for Video Technology*, 17:1103–1120, Sep. 2007.

[26] Skype. Homepage. `http://www.skype.com`.

[27] Traceroute.org. Homepage. http://www.traceroute.org/.

[28] Y. Wang, A. Reibman, and S. Lin. Multiple Description Coding for Video Delivery. *Proceedings of the IEEE*, 93, Jan. 2005.

[29] M. Wien, H. Schwarz, and T. Oelbaum. Performance Analysis of SVC. *IEEE Trans. Circuit and System for Video Technology*, 17:1194–1203, Sep. 2007.

[30] Wireshark. Homepage. http://www.wireshark.org/.

[31] T. yuan Huang, K. ta Chen, and P. Huang. Could skype be more satisfying? a QoE-Centric study of the fec mechanism in an internet-scale voip system. *IEEE Network*, 24(2):42, Mar 2010.

[32] X. Zhang, Y. Xu, H. Hu, Y. Liu, Z. Guo, and Y. Wang. Profiling Skype Video Calls: Rate Control and Video Quality. In *Proceedings of IEEE INFOCOM*, March 2012.