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-bandwidth and low-delay voice and video transmissions between geographically distributed users. 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 on the Internet: Google+, iChat, and Skype. Through a series of carefully designed active and passive measurements, we are able to unveil important information about their key design choices and performance, including application architecture, video generation and adaptation schemes, loss recovery strategies, end-to-end voice and video delays, resilience against random and bursty losses, etc. Obtained insights can be used to guide the design of applications that call for high-bandwidth 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 prices to end-consumers on various platforms. Notably, Apple iChat [15], Google+ Hangout [11], and Skype Video Calls [26] are among the most popular ones on the Internet.

Video conferencing requires high-bandwidth and low-delay voice and video transmission. While Skype encodes high-quality voice at data rate of $40kbps$, a Skype video call with good quality can easily use up bandwidth of $900kbps$ [33]. 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 device and network access heterogeneities, 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 of three existing solutions: iChat, Google+, and Skype, to investigate *how they address video conferencing challenges and how well they do it on the Internet.* Specifically, our study is focused on the following issues about their key design choices and their delivered user conferencing experiences.

- **System Architecture:** A natural conferencing architecture is Peer-to-Peer (P2P), where users send their voice and video to each other directly. Skype employs P2P for VoIP and two-party video chat. While P2P achieves low-delay, it comes short at achieving high-bandwidth: a residential user normally cannot upload multiple high-quality video streams simultaneously. While video conferencing servers can be employed to relay users' voice and video, data relay often incurs longer delay than direct transfer. Conferencing servers have to be strategically located and selected to balance the bandwidth and delay performance of voice and video relay between geographically distributed users.

- **Video Generation and Adaptation:** To cope with receiver heterogeneity, a source can generate single video version at a rate *downloadable* by the weakest receiver. One-version design unnecessarily limits the received video quality on other stronger

receivers. Alternatively, multiple video versions can be generated, either directly by the source or by relay servers through transcoding. Different video versions will be sent to different receivers, matching their download capacities. In a simple multi-version design, each video version is encoded and transmitted separately. It incurs high encoding and bandwidth overhead. *Scalable Video Coding* (SVC) encodes video into multiple layers. It is appealing to adopt SVC in video conferencing to realize multi-version design with greatly reduced overhead.

- **Packet Loss Recovery:** To achieve reliability in realtime streaming, the conventional wisdom is to use Forward Error Correction (FEC) coding, instead of retransmission. However, in video conferencing, video has to be encoded and decoded in realtime. To avoid long FEC encoding and decoding delays, FEC blocks have to be short. This largely reduces FEC's coding efficiency and robustness against bursty losses. Meanwhile, retransmission is viable if the network delay between sender and receiver is small. 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. The choice between FEC and retransmission is tightly coupled with system architecture and video generation.

- **User Quality-of-Experience:** Ultimately, the performance of a video conferencing system is evaluated by the delivered user conferencing experiences, which are highly sensitive to various voice and video quality metrics, such as end-to-end voice and video delay, synchronization between voice and video, video resolution, frame-rate and quantization, etc. To provide stable conferencing services to end consumers over the Internet, it is extremely important for a conferencing system to be adaptive to varying network conditions and robust against random network impairments. We systematically study the delivered user conferencing experiences of each system in a wide range of real and emulated network scenarios. We further investigate the implications of the design choices made by each system on their delivered user experiences.

It is admittedly challenging and ambitious to come up with conclusive answers for these questions. All three systems use proprietary protocols and encrypt data and signaling messages. There is very limited public information about their architectures and protocols. To address these challenges, we undertake an extensive measurement campaign and systematically measure the three systems as *black-boxes*. We observe and analyze their behaviors and performances through a set of carefully designed active and passive measurement experiments. Extrapolating from the measurement results, we are able to unveil important information about their system architecture, video encoding and adaptation, loss recovery and delivered user QoE. Our major findings are summarized as following.

1. While P2P is promising for voice conferencing and two-party video calls, P2P alone is not sufficient to sustain high-quality multi-party video conferencing. The design choices and performance of multi-party video conferencing systems are largely affected by the availability of bandwidth-rich server infrastructures.

2. Conferencing server locations not only impact the delivered user delay performance, but also affect the design of loss recovery mechanism and the achieved loss resilience. When servers are located close to end users, retransmission is more preferable than FEC to recover from random and bursty losses. Data relays through well-provisioned server networks can deliver low-delay and high-bandwidth voice and video transfers between geographically distributed users.

3. To deliver high-quality video telephony over the Internet, realtime voice and video generation, protection, adaptation, and distribution have to be *jointly* designed. Various voice and video processing delays, incurred in capturing, encoding, decoding and rendering, account for a significant portion of the end-to-end delays perceived by users.

4. Compared with multi-version video coding, layered video coding can efficiently address user access heterogeneity with low bandwidth overhead. With layered coding, content-aware prioritized selective retransmissions can further enhance the robustness of conferencing quality against random and bursty losses.

The rest of the paper is organized as follows. We briefly discuss the related work in Section 2. Our measurement platform is introduced in Section 3. Then we study the application architectures of the three systems in Section 4. Video generation and adaptation strategies are investigated in Section 5. Voice and video delay performances are measured in Section 6. In Section 7, we investigate their loss recovery strategies, and measure their resilience against bursty losses and long delays. The paper is concluded in Section 8.

2. RELATED WORK

Most of the previous measurement studies of realtime communications over the Internet were focused on Skype's VoIP service. Baset *et al* [1] first analyzed

Skype's P2P topology, call establishment protocol, 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, 32] 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, 32]. 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.

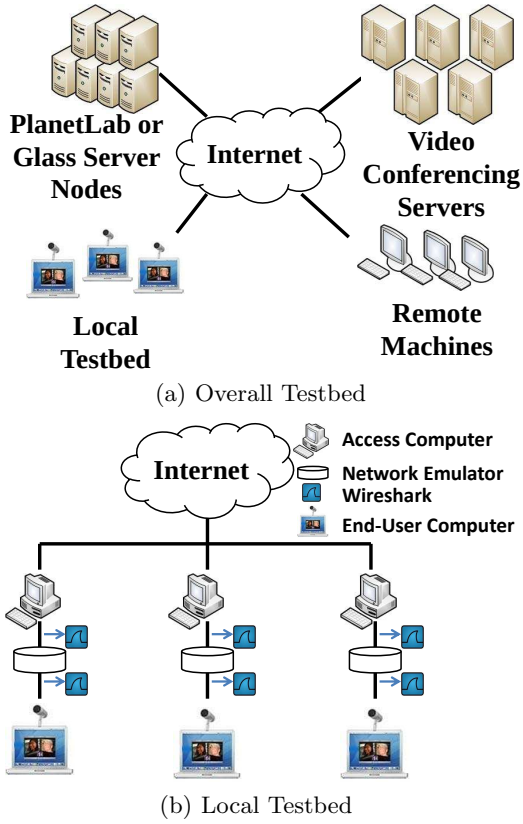
More recently, there are some measurement work on video telephony. Cicco *et al.* [5] measured the responsiveness of Skype video calls to bandwidth variations. They conclude that Skype's response time to bandwidth increase is too long. In [33], we conducted an extensive measurement study of Skype two-party video calls under different network conditions. Based on the measurement results, we propose models for Skype video calls' rate control, FEC redundancy, and video quality. According to a year 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 its video conferencing service for users in *friend circles*. To obtain a broader view, we extended our study to Google+, as well as Apple's iChat.

As illustrated in Fig 1(a), our measurement platform consists of three major components: local testbed within NYU-Poly campus network, remote machines of friends distributed all over the world, and Planetlab [22] and Glass Server [27] nodes at selected locations. Experimental video calls are established either only 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 contents are consistent and repeatable.

As illustrated in Fig 1(b), 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 or bursty packet loss, and available network bandwidth. We use Wireshark [31] to capture detailed packet-level information before and after the network emulators. iChat, Google+ and Skype all report technical information about video quality through application window, such as video rates, frame rates, RTT, etc. [21] We use a screen text capture tool [23] to capture these information periodically. The sampling interval is 1 second. Most of the measurement experiments are automated. All the reported statistics in the rest of the paper are drawn from a large number of samples.



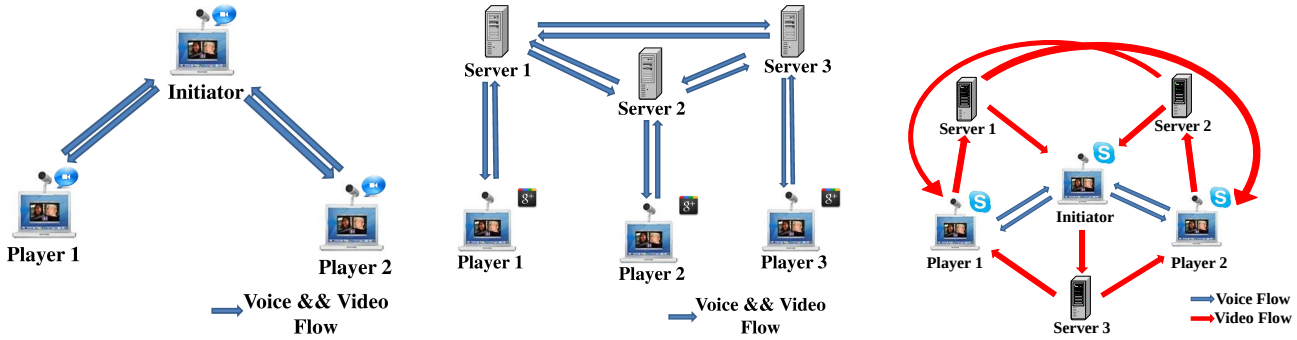
(a) Overall Testbed

(b) Local Testbed

Figure 1: Testbed to Measure the Three Video Conferencing Systems

4. SYSTEM ARCHITECTURE

There are three main architectures for networked applications: client-server, 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 that of voice, and might have to send the encoded video to multiple receivers. From the 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.



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

Figure 2: System Architectures of the iChat, Google+, and Skype

4.1 Methodology

For each system, we set up video calls and use Wireshark 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 trace analysis, we capture those TCP and UDP sessions, whose durations are long enough (at least half of the conference session time) and data rates are significant (larger than $5kbps$), as voice or video flows. To get the right topology and differentiate between signaling packets, voice packets and video packets, we conduct three stages of experiments. In the first stage, each user sets his voice and video on to form a video conference. In the second stage, users only set voice on to form a voice conference. In the third stage, all users shut down their videos and mute their voices. We compare the recorded flow sessions in the three stages. Consequently, we identify voice flows, video flows, and signaling flows.

4.2 iChat is P2P

iChat employs a P2P architecture and users in a conference form a star topology at the application layer as shown in Fig. 2(a). The central hub is the conference initiator, i.e., 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 uploads his voice and video data to the initiator through one UDP flow, and also downloads other participants' voice and video data from the initiator through another UDP flow. Participants use UDP port 16402. Voice and video are transported using the RTP protocol [13]. Normal users only connect to the initiator. There is no direct connection between two normal users. iChat cannot work if UDP traffic is blocked.

4.3 Google+ is Server-centric

Google+ video calls, both two-party and multi-party, always use server-centric topology, as illustrated in Fig. 2(b). Each user sends his voice and video to a dedicated proxy server, and also receives other users' voice and video from that server. There is no direct transmissions between users. Generally, different users choose different proxy servers. Thus, the proxy servers need to communicate with each other to exchange users' voice and video. 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. TCP is used only when we deliberately block UDP traffic. There is a trick [21] to access various statistics of Google+. The statistics show that two of the four flows carry voice and video respectively. Google+ also uses RTP protocol to transmit voice and video. 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 [33]. 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 $50kbps$, 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 voice stream. For video transmission, each user uploads his video to a Skype server, which relays the video flow directly to all other users in the conference, without going through another relay server. Different users normally choose different relay servers.

Similar to Google+, Skype mostly uses UDP to transmit voice and video, and only switches to TCP if UDP is blocked. Different from Google+ and iChat, Skype does not use RTP and RTCP. In addition to voice and video flows, we also observe some low-rate flows between servers and users. We conjecture that these small flows

are for signaling.

4.5 Conferencing Server Placement

Google+ and Skype employ relay servers. If servers are not properly located and selected for users, voice and video relay will incur long delays that significantly degrade users' experience of realtime interaction in a conference. To determine the server locations of Skype and Google+, we set up video conferences with our friends all over the world. Table 1 shows the Google+ server IP addresses used by our friends from different locations. When using a popular geo-location tool Maxmind [19] to geo-locate those IP addresses, we were surprised that all Google+ server IP addresses are geo-located to Mountain View, CA, USA. However, this seems incorrect. In Table 1, we also report the measured RTT between our friends' machines and their corresponding proxy Google+ servers. The RTT results suggest that most of our friends are assigned to Google+ servers within short network distances, except for the two friends in Australia. Thus, we infer that Google+ hangout deploy conferencing servers all over the world.

Table 1: RTT to Google+ Hangout Servers

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 for our friend machines are all located in the subnet of 208.88.186.00/24. Maxmind [19] reports that those servers are in Estonia. To verify this, we select Planetlab [22] and Glass Server [27] nodes to ping Skype servers randomly picked from the identified subnet and report the RTT results in Table 2. From nodes located in New York and Hong Kong, we also pinged all possible IP addresses in the subnet. 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. Even if the bit propagation speed is exactly the speed of light ($3 \cdot 10^8$ meters/sec), the RTT between New York and Estonia is about 44.4ms (11,314km) [16], which is even larger than the measured RTT in Table 2. Thus, Skype servers can't be located in Estonia. To geolocate Skype servers, we did additional traceroute experiments from different locations. We found that the last-hop router IP addresses are all located near New Jersey area. This is consistent with Table 2, where the location with the smallest RTT is New Haven. Thus, we infer that Skype servers are located near the New Jersey/New York area. As will be shown in the following sections, server locations not only directly impact users' delay performance, but also have implications on the design and performance of loss recovery mechanisms of Google+ and Skype.

Table 2: RTT to Skype Video Relay Servers

PlanetLab/Glass Server Locations	RTT (ms)
New Haven, USA	17.4
New York, USA	25.9
Washington, USA	31.3
Vancouver, CA	61.41
San Jose, USA	67.5
London, UK	99.2
Guayaquil, EC	111
Saarbrucken, DE	138
Oulu, FIN	151
Rio de Janeiro, BRA	176
Tel Aviv, IL	207
Canberra, AU	222
Hong Kong, CHN	226
Brisbane, AU	266
West Bengal, IN	324

5. VIDEO GENERATION AND ADAPTATION

In a video conference, each source encodes and uploads his video in realtime. To cope with network bandwidth variations, the encoded video rate on a source first has to be adapted to his available uplink bandwidth. Additionally, in multi-party video conferencing, each source has multiple receivers, potentially with different download capacities. The source video encoding rate also has to be adapted to receivers' download capacities. In an *one-version* design, a source generates single video version that can be downloaded by the weakest receiver, and sends that version to all receivers. This unnecessarily limits the received video quality on stronger 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 uploads those versions to the server/receivers. One obvious drawback is that, as an 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 [29]. A receiver can decode a basic quality video after receiving the base layer. Higher video quality can be obtained as more video layers are received. Recent advances in SVC coding has brought down the layered video encoding overhead to 10% [30]. 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 still be decoded as long as the base layer is received reliably.

5.1 Methodology

Since there is no public information about their video generation and adaptation strategies, we design experiments to trigger video adaptation by manipulating source upload bandwidth and receiver download 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 purely as 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 the network emulator NEWT. We collect video information from the application window. We also record the packets using Wireshark [31] for offline video payload analysis.

5.2 Video Encoding Parameters

To cope with bandwidth variations, all three systems encode video using a wide range of parameters, 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. The frame rates can vary from 1 FPS (Frame-Per-Second) to 30 FPS for each system. In our experiments, the ranges of the observed resolution values are also listed in Table 3. Skype and Google+ adapt video resolution to network bandwidth. iChat's video resolution is determined by the number of users in the conference. For example, the resolution is always set to be 640×480 in the case of two-party call. When three or more users involved in the conference, the resolution becomes 320×240 or 160×120. And once the resolution is set at the beginning, it will not be changed afterward, no matter how we change the bandwidth setting. We cannot directly measure the quantization adaptation for any of the three systems.

Table 3: Video Rate and Resolution Ranges

System	Rate (kbps)	Resolutions
iChat	$49 \sim 753$	640*480,320*240,160*120
Google+	$28 \sim 890$	640*360,480*270,320*180 240*135,160*90,80*44
Skype	$5 \sim 1200$	640*480,320*240,160*120

5.3 Video Adaptation

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, they will automatically shutdown the video. This shows that the three systems have their own available network bandwidth probing algorithms to determine the video quality to be encoded. To study video adaptation to receiver bandwidth heterogeneity, we set receivers' download capacities to different levels.

5.3.1 iChat uses One-version Encoding

We found that iChat uses *one-version encoding:* heterogenous receivers always receive the same video version, and the receiver with the lowest download capacity determines the video quality sent out by the sender. No video trans-coding, nor layered coding, is employed.

5.3.2 Skype uses Multi-version Encoding

When there is a large variability in receivers' download capacities, Skype employs *source-side multi-version encoding*. For example, in a video conference of one sender and three receivers, receiver 1, receiver 2 and receiver 3's download capacities are set to 150 kbps, 400 kbps and 2000 kbps respectively. The sender generates three video versions and uploads them to the relay server, which distributes one of the three video versions to the appropriate receiver. In this case, the received video rates on receiver 1, 2, and 3 are 70 kbps, 200 kbps, and 500 kbps, respectively. Server-side transcoding was not observed. In our experiments, the largest number of video versions that a sender can generate is three. When receiver capacity variability is small, or the sender upload capacity is low, Skype uses one-version encoding and sends to all receivers the same video quality.

5.3.3 Google+ uses Multi-layer Encoding

Google+ always give different video qualities to heterogenous receivers. For example, we only limit the download capacity of one receiver, say receiver 2, to 500 kbps. On the sender side, the encoded video rate is 835.7 kbps, with resolution of $640 * 360$ and frame rate of 30 FPS. On receiver 1, the received video rate is 386.9 kbps, with resolution of $640 * 360$, and frame rate of 14 FPS. On receiver 2, the received video rate is 168.6 kbps, with resolution of $320 * 180$, and frame rate of 14 FPS. The perceptual video quality on both receiv-

er 1 and receiver 2 are consistent and acceptable. The experiment environment is not fully controlled. Video flows between our experiment machines and Google+ servers traverse the Internet, and 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.

To gain more insight about how the two video qualities are generated, we examine the packets captured on the sender and receivers. Both Google+ and iChat use RTP[13] protocol for voice and video 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 video 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 can be used by the receiver to detect packet loss and restore packet sequence. In Google+, video packets generated by one source form a unique sequence. "Timestamp" field records 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 in a frame, then its "Marker" field is set to 1; otherwise the value is 0. Thus, we infer that the "Marker" field can be used to identify the boundary between two adjacent video frames. Since different machines use different initial sequence numbers, to match video frames between the sender and receivers, we instead use the "Timestamp", "Marker" and "Length" fields.

In Table 4, 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 in the same row have the same values in these three fields.) In the aligned table, the sender sent out five video frames, both receiver 1 and receiver 2 only received two frames. While receiver 1 received all packets of those two frames, receiver 2 only received the first two packets of those 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 on their bandwidth conditions.

Both receivers can decode the video with decent quality. This suggests that Google+ employs multi-layer coding. Since receiver 1 can decode the video even though some frames are missing, the 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 still be decoded at a lower frame-rate by a receiver even if a subset of frames/layers are dropped. Since receiver 2 can decode video even though it lost some packets within each frame, the 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 observe that no matter how low the download bandwidth of a receiver is, the received video resolution is at least one quarter of the resolution of the original video. This shows that the encoded video only has two spatial layers. This is reasonable, because spatial layers incur much higher encoding overhead than temporal layers. Thus, the number of spatial layers could not be too large.

We also developed a script and analyzed other RTP packet traces of Google+. It is verified that Google+ employs layered video coding with temporal and spatial scalability. Different layers have different importances for video decoding. Consequently, one should use different realtime transmission and protection strategies for different layers. As will be shown in the later sections, prioritized loss recovery scheme can be designed for layered video coding to achieve high robustness against packet losses.

Table 4: RTP Packet Headers in Google+

M	Timestamp	Length (bytes)	Sequence Number		
			Sender	Receiver1	Receiver2
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 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

study the delay performance of the three systems.

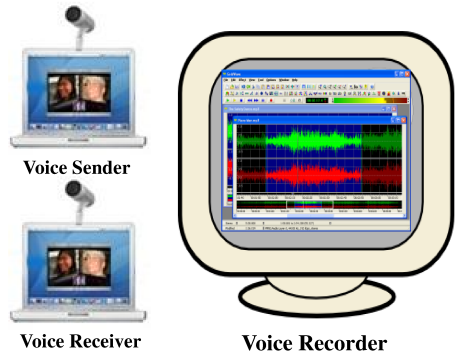
6.1 Methodology

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 voice/video capturing and encoding delay at the sender, T_n be the one-way transmission and propagation delay on the network path between the sender and the receiver, T_s be the server or super-node processing time ($T_s = 0$, 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:

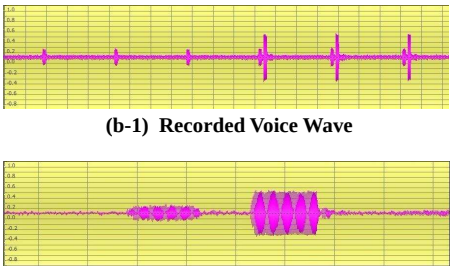
$$T = T_e + T_n + T_s + T_d. \quad (1)$$

Obviously, it is not sufficient to just measure the network transmission delay T_n. We therefore measure end-to-end delay by emulating a real user's experience.

6.1.1 One-way Voice Delay



(a) One-way Voice Delay

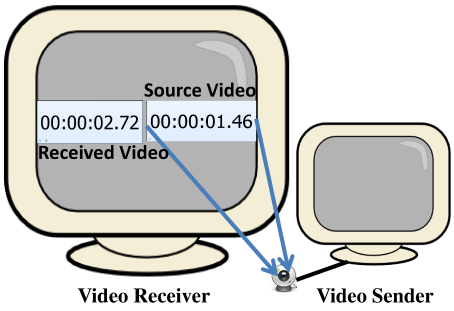


(b) Voice Waves Detail

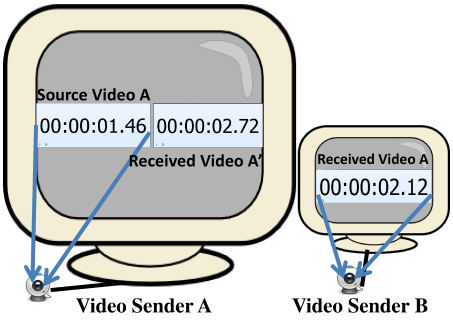
Figure 3: Testbed to Measure Vioce Delay

To record one-way voice delay, we employ a repeatable "Tick" sound as the voice source and inject it to the sender using a virtual microphone [28]. In our local testbed, we set up three computers side by side, as illustrated in Fig. 3(a). One computer is the voice sender, another one is the voice receiver. The sender repeatedly sends the "Tick" sound to the receiver. The 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]. [1] We can visually analyze the 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 amplitudes correspond to the repeated "Ticks" sent by the sender, the impulses with larger amplitudes 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. Each impulse is indeed a waveform segment with rich frequency components. We use the time-lags between the first peaks of two adjacent segments as the one-way voice delays.

6.1.2 One-way Video Delay



(a) One-way Delay



(b) Round-trip Delay

Figure 4: Testbed to Measure Video Delay

To measure video delay, we similarly emulate a user's video experience by simultaneously "viewing" (recoding) the original video on the sender and the received video on the receiver using a video capturing program. As illustrated in Fig. 4(a), we set up two computer-

[1] The sender uses a virtual microphone to inject the "Tick" sound to conferencing system. Consequently, the received sound on the receiver will NOT be looped back to system.

s side-by-side in our local testbed, one as the sender, the other as the receiver. We run a stopwatch program [8] on the *receiver* side, and focus the sender's video camera on the stopwatch window on the receiver's screen. The sender transmits the stopwatch video captured 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 stopwatches 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 $1,260$ ms. We use a software program running on the receiver to capture the snapshots of the receiver screen 10 times every second. We then use a software [6] to convert the captured pictures into mono-color, and extract stopwatch images from the mono-color pictures. Finally, the stopwatch readings will be decoded from stopwatch images using an Optical Character Recognition (OCR) software [9]. It should be noted that if the received video quality is bad, the OCR software cannot decode the stopwatch readings correctly. We skip those undecodable samples in our delay calculation. The undecodable picture ratio can also be 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 illustrated 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 using one of the three conferencing systems. The received stopwatch video is now displayed on B's screen. Then B focuses his camera on the received video from A, and sends the captured video 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 is roughly half of the round-trip delay. After the cameras on 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 delay case to process the data and get a large number of delay samples.

6.2 One-way Voice and Video Delay

We first test the one-way delay between users in our local testbed. All user computers are connected to the same router. The network delay along the direct connection between them is almost negligible.

Table 5: 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

The voice delay and video delay performances for three systems are listed in Table 5. For Google+, the average video delay is $180ms$, larger than the voice delay of $100ms$. As studied in Section 4.3, Google+ uses server-centric architecture. The round trip network delay between our local testbed and the allocated Google+ server is $14ms$. The addition delays are due to voice and video processing on both ends. It takes longer to process video than voice. Skype two-party calls use direct P2P transmissions for voice and video. Since network delays between machines in local testbed are almost zero, the measured one-way delays are mostly due to voice/video processing. *This suggests that voice and video processing can take a significant portion of the delay allowance of good-quality video conferencing.*

As studied in Section 4.4, Skype multi-party call employs different overlay topologies for voice and video transmissions. The voice flow from a non-initiator to another non-initiator has to be transmitted to the initiator first. The initiator will do some processing, like voice mixing and recoding. Thus, the voice delay between an initiator and a non-initiator is shorter than the voice delay between two non-initiators. Since video has to be first sent to Skype servers then relayed to receivers, Skype video delay is larger than voice delay. Video and voice are *unsynchronized*. The voice and video delay gap is as large as $100ms$ when we consider the case from an initiator to a normal user.

iChat transmits video and voice in one RTP flow. For two-party calls, video and voice are synchronized. For multi-party conferences, when voice and video flows are transmitted from the initiator to non-initiators, delay performance is the same as the two-party case. The voice and video delays between two non-initiators are longer than those between the initiator and non-initiators. Since the network delays are negligible, the additional delays are mostly for the initiator to combine packets generated by different sources into one RTP flow before sending it to a receiver.

6.3 Round-trip Video Delay

We also measure the round-trip video delays (video RTTs) between New York City and Hong Kong, New York City and Singapore when using Google+ Hangout and Skype Multi-party conferencing systems respectively. From Figure 5, in both cases, video RTTs of Google+ are much smaller than those of Skype. For the case between New York and Singapore, Google+'s mean video RTT is only about $579ms$, with standard deviation of $44ms$. We can approximate the one-way video delay by half of the video RTT. Google+'s one-way video delay is about $290ms$, within the $350ms$ delay allowance. To the contrary, Skype's mean video RTT is about $1,048ms$, with standard deviation of $239ms$. The corresponding one-way delay is over the delay allowance. Between New York City and Hong Kong, the average one-way video delay of Google+ is $374ms$, again much shorter than Skype's average delay of $788ms$.

From the topology study in Section 4, in Skype, video generated by a source is directly relayed by a server to all receivers. Video flows from a source to its assigned Skype relay server and from the relay server to the receivers all have to traverse the public Internet. They have to compete with background network traffic. To the contrary, Google+ deploy relay servers located all around the world, and a user is mostly connected to a close-by server. The transmission between Google+ relay servers likely traverse their own private backbone network with good QoS guarantee. This way Google+ voice and video relays incur much less random network losses and delays than Skype's voice and video relays over the public Internet. Additionally, as will be shown in the following section, Google+'s loss recovery scheme is more efficient than Skype, which also leads to shorter end-to-end video delay.

7. ROBUSTNESS AGAINST LOSSES

One main challenge of delivering video telephony over the best-effort Internet is to cope with unpredictable network impairments, such as congestion delays, 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 network delay between the sender and the receiver is short, e.g. RTT of $20ms$ between our local testbed and Google+ servers, retransmissions might be affordable within an end-to-end delay allowance of $350ms$. Unlike FEC, retransmission adds in 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, and how robust they are against random and bursty losses.

7.1 Methodology

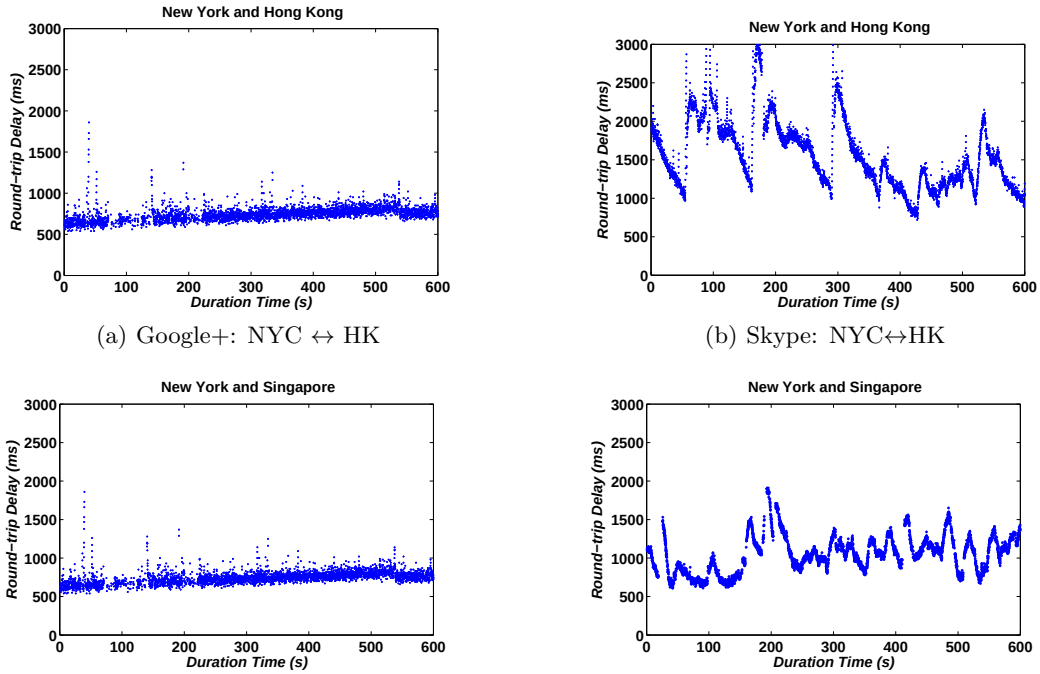
We set up multi-party conferences using machines in our local testbed. We conduct two sets of experiments by injecting packet losses using network emulators. In the first set of experiments, we add upload losses on the video sender side. In the second set, we add download losses on only one of the receivers. Because we capture packets before and after the loss module, we can figure out which packets are lost. Through packet trace analysis, we can also search for packets that have similar payloads as the lost ones to check whether retransmission is employed or not. At the same time, we monitor the application window to collect statistics on video rates and total data rates to calculate redundant transmission ratios.

7.2 Skype uses FEC

In our Skype experiments, we never found any retransmission of lost packets. From Skype's technical window, we can easily observe gaps between the total data rates and video rates. Previous studies [32, 33] suggest that Skype employs aggressive Forward Error Correction (FEC) coding for VoIP and two-party video calls. We infer that the observed rate gaps are also due to FEC. Let r_v be the actual video rate, r_s be the actual sending rate. FEC redundancy ratio ρ is calculated as the ratio between the redundant traffic rate and the total sending rate:

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

The experiment results of adding random upload losses are shown in Table 6. The reported video rates are the same on the sender and two receivers. Surprisingly, the sender always adds significant redundancy even if we don't introduce any additional packet losses. (Note that, since the upload flow traverses the Internet to reach Skype server, the sender might still incur 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 [33].) On the other hand, the redundancy ratios on the two receivers are pretty low, even though the download flows also have to traverse the Internet. One possible explanation is that Skype tries hard to protect video uploaded by a source, because any lost video during the upload has quality implications 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.



(a) Google+: NYC ↔ HK

(b) Skype: NYC↔HK

(c) Google+: NYC↔SG

(d) Skype: NYC↔SG

Figure 5: Round-trip Video Delays of Google+ and Skype

Table 6: 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

When the loss rate is really high, Skype significantly drops its video rate and the total sending rate. This is consistent with the observation for Skype two-party call in [33].

The results for download losses are shown in Table 7. Again, the reported video rates are the same on the sender and two receivers. 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 a trend that as the loss rate increases, more FEC packets are added into the video flow of receiver 1. In Table 6, 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 suggests that the relay server first removes FEC packets from the received data, and then adds new FEC packets to each download flow. Results of Table 7 shows that the two receivers in different conditions receive the same video rate with different FEC ratios. Thus, we infer that relay server monitors network conditions of receivers and calculate different FEC ratios for different receivers.

7.3 Google+ uses Selective Persistent Retransmission

As shown in Section 4.3, a Google+ user only interacts with a Google+ server for video upload and download. In our experiments, we only focus on one video sender and one video receiver. We first set up a machine as a video sender and inject random packet losses to its uplink. Then we capture the packets before and after the network emulator so that 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 8. 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%. We calculate video rate as

Table 7: 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

total rate minus retransmission rate. Google+ strives to maintain a high video upload rate even under high packet losses. This is similar to Skype's high FEC protection for video uploading. The *recovery ratio* is defined as the fraction of lost packets that are eventually received by the receiver. It shows that Google+ implements *selective retransmission*, only half of the lost packets are recovered. The *persistent ratio* defines the fraction of packets retransmitted at least once that are eventually received by the receiver. A persistent ratio of 1 means that if Google+ attempts to retransmit a packet, it will persistently retransmit the packet until it is received successfully. Since Google+ uses layered video coding, where packets from the lower video layers are more important for decoding. We conjecture Google+ applies *selective persistent retransmission* to packets of the lower video layers first. In all experiments, Google+ consistently offers good video quality, even under 20% uplink loss.

Table 8: Google+ Retransmission: Uplink

Loss Rate	Video FPS	Total Rate (kbps)	Retran. Rate (kbps)	Rcvry. Ratio	Persis. 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

The results of adding download losses on the receiver is shown in Table 9. It shows a different trend. 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 signal to infer network congestion. As packet loss rate increases, it not only retransmits important lost packets, but also proactively reduces 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 cannot control packet loss outside of our local testbed, we still observe packet retransmissions even if the injected packet loss rate is zero. We also tried downlink loss rate of 40%. Surprisingly, Google+ still offers reasonably good perceptual video quality.

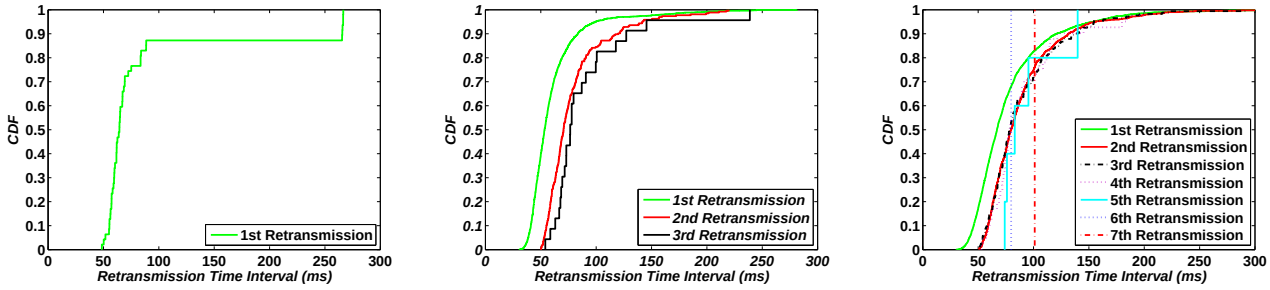
Table 9: Google+ Retransmission: Downlink

Loss Rate	Video FPS	Total Rate (kbps)	Retran. Rate (kbps)	Rcvry. Ratio	Persis. 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

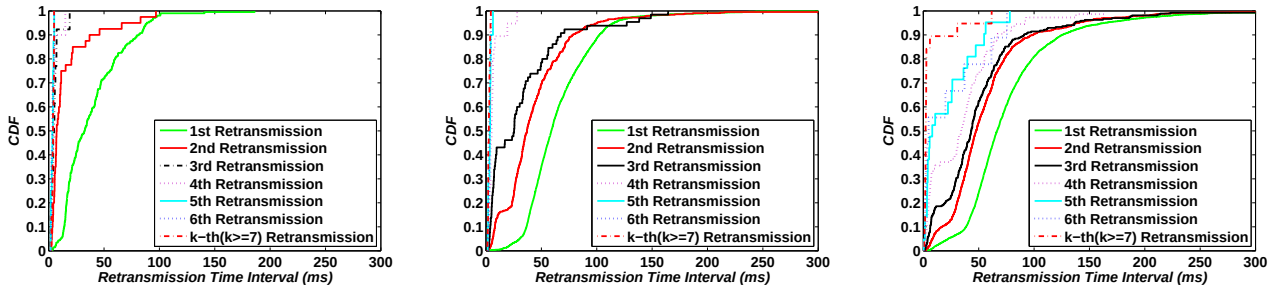
To gain more insights about Google+'s loss recovery strategy, we collect more statistics about its packet retransmissions. Table 10 lists how many retransmissions Google+ attempted to reliably transfer a packet at different packet loss rates. Most of the time, retransmission is done within one or two tries. Sometimes, a packet is retransmitted many times, with the highest up to 18 times. We define the *k-th retransmission interval* as the time lag between the kth retransmission and the $(k-1)th$ retransmission of the same packet (the original transmission is considered as the $0th$ retransmission). Table 11 presents the mean and standard deviation of retransmission intervals. For reference, the RTT between our machines and Google+ server is 14 ms. And the CDFs of retransmission intervals are plotted in Figure 6. In general, the retransmission intervals on the uplink are longer than the downlink. On the uplink side, 60% of the first retransmissions happen within $70ms$ after the first original packet transmission, which is about 3.75 times of the RTT. The retransmissions on the video uploader side all need to wait for such a long time. This is to avoid wasting valuable user upload bandwidth in retransmitting delayed, instead of lost, packets. On the downlink side, Google+ servers retransmit lost packets more aggressively with shorter intervals. Aggressive retransmissions might waste some server bandwidth, but recover faster from losses. Many of the N-th ($N \geq 5$) retransmissions are attempted within $5ms$ after the previous retransmission. This batch retransmission strategy is likely used to deal with severe and/or 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+. The results for its retransmissions are shown



(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: CDF of Retransmission Time Intervals for Google+ Hangout

Table 10: Retransmission Probability in Google+

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

in Table 12 and Table 13. Unlike Skype and Google+, video FPS decreases when upload losses are induced. However, that value doesn't change too much under download loss. Different from Google+, iChat does not do selective retransmission. It always retransmit lost packets. But, its retransmission is not *persistent*. Most of the time, it just tries to retransmit lost packet once as shown in Table 14. There is no guarantee that a lost packet will be successful recovered. Even though the recovery ratios reported in Table 12 and Table 13 are higher than those reported in Table 8 and Table 9, since iChat does not employ layered coding and prioritized retransmission, a lost video packet may prevent decoding of multiple video frames. Consequently, iChat's video quality under losses is much worse than Google+, even though they both use retransmissions to recover from packet losses. This suggests that layered video coding not only adapts to user heterogeneity well, but also enhances video quality resilience against losses.

Table 12: iChat Retransmission: Uplink

Upload Loss	Video FPS	Sending Rate	Retrans. Rate	Rcvry. Ratio
0	25.1	365.7	0	-
0.02	19.8	363.0	7.0	0.95
0.10	19.6	411.5	34.4	0.81

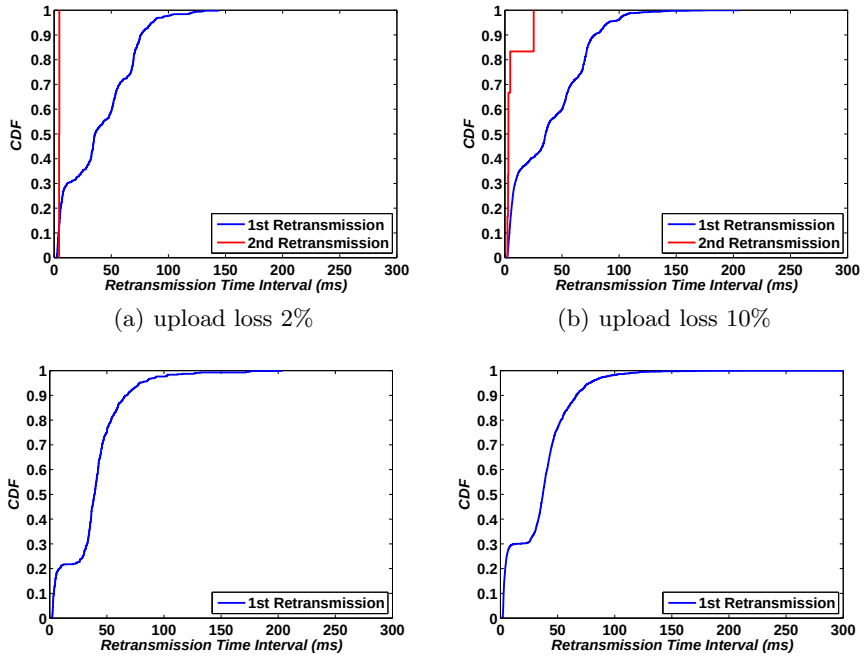
The CDFs of retransmission intervals are shown in Fig. 7. The mean retransmission intervals are reported in Table 15. Since we set machines in the same subnetwork, the RTTs between them are only about $2-4$ ms. Even though iChat waits for 30+ ms for the first retransmission, the second retransmission happens only $3ms$ after the first retransmission.

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

Table 11: Mean and Standard Deviation(SD) for Retransmission Intervals in Google+

Loss Rate	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)
Uplink: 0	90.14	68.66												
Uplink: 0.05	64.69	31.12	70.47	14.90	61.56	9.13	56.88	0						
Uplink: 0.10	60.28	26.82	79.62	30.05	89.63	39.44								
Uplink: 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
Downlink: 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: 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: 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: 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



(a) upload loss 2%

(b) upload loss 10%

(c) download loss 2%

(d) download loss 10%

Figure 7: Retransmission Time Interval under Loss Variation for iChat

Table 13: iChat Retransmission: Downlink

Loss Rate	Video FPS	Total Rate	Retrans. Rate	Rcvry. Ratio
0	25.1	365.8	0	-
0.02	24.3	401.7	7.8	0.95
0.10	24.2	447.3	37.7	0.81

the video can be continuously delivered to the receiver in realtime, the measured video delay from the stopwatch video should be consistently low. Since the original stopwatch continuously advances, losses of video frames on the receiver side will lead to large measured video delays. Additionally, if the received stopwatch video quality is bad, the text recognition tool, OCR, cannot decode the clock reading. The recognition ratio of OCR is an indirect measure of the received video quality. In this section, we compare the robustness of the three systems against bursty losses and long delays.

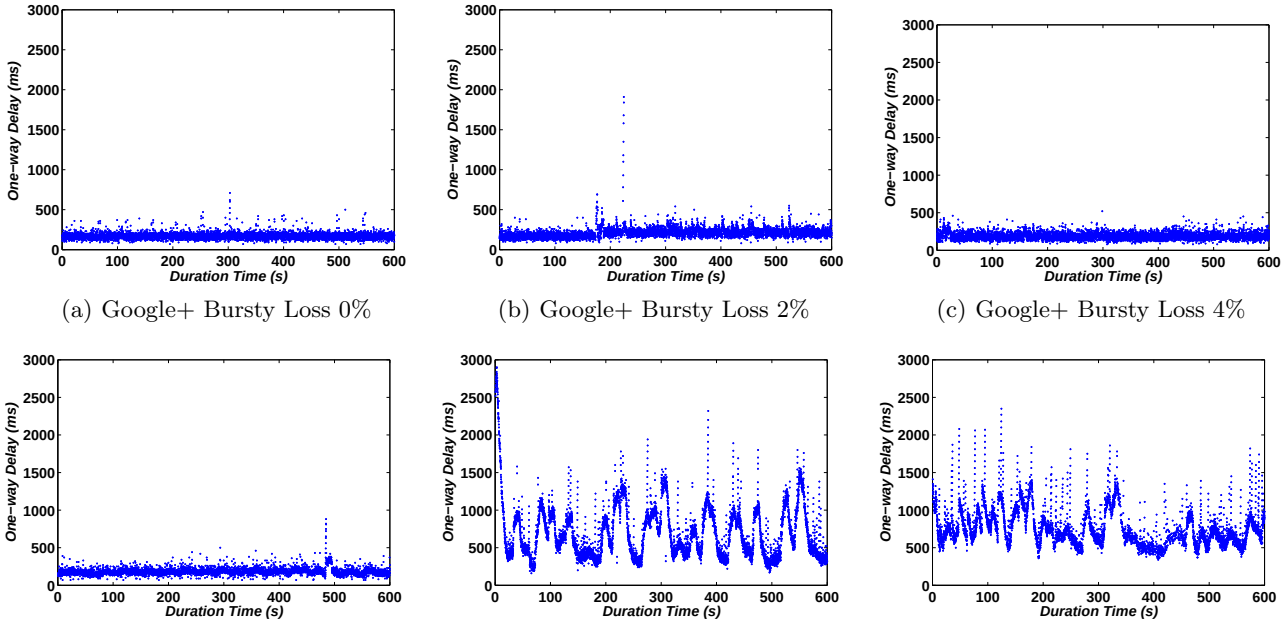
Packet losses can be bursty, especially on wireless

Table 14: 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

links. We use the network emulator to generate bursty losses on the download link. For each loss burst, $4 \sim 5$ packets are dropped in batch. Table 16 compares the stopwatch recognition probabilities of the three systems under different bursty loss rates. The stopwatch 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 8 compares the one-way video delay performances of Google+ and Skype. When we don't introduce any additional loss, both Google+ and Skype are



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

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

Figure 8: One-way Video Delay For Google+ and Skype under Different Bursty Loss Probabilities

Table 15: 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

Table 16: 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

able to maintain a low and stable video delay, with average about 200 ms. As we introduce additional bursty losses, Skype incurs large and highly-variable one-way video delays, as shown in Figure 8(e) and Figure 8(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 $33ms$. 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 of the base layers upon bursty losses. The receiver can decode video as long as all packets of the base layer are received. As a result, Google+ is able to maintain low video delays up to 4% bursty losses in Figure 8(b) and Figure 8(c).

Compared with employing FEC, one main drawback of using retransmission is that it does not work well when the network delay between the sender and the 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 delay on the access link of our local machines using the network emulator. Figure 9 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 higher variances. This suggests that FEC is preferable over retransmissions if RTT is large, loss is random, and loss rate is not too high.

8. CONCLUSION

In this paper, we present our measurement study of three popular video telephony systems for end-consumers. Through a series of carefully designed active and passive measurements, we were able to unveil important information about their design choices and performances. Our study demonstrated that pure P2P architecture cannot sustain high-quality multi-party video conferencing services on the Internet, and bandwidth-rich server infrastructure can be deployed to signifi-

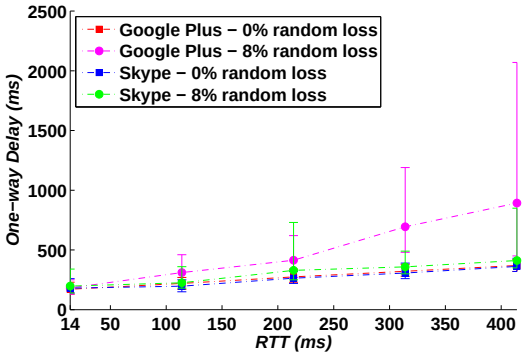


Figure 9: Google+ and Skype Delay Performances under Different Network Delays

cantly improve user conferencing experiences. We also demonstrated that, in the extremely tight design space of video conferencing system, video generation, protection, adaptation and distribution have to be jointly considered. Various voice/video processing delays, incurred in capturing, encoding, decoding and rendering, account for a significant portion of the end-to-end delays perceived by users. Compared with multi-version video coding, layered video coding can more efficiently address user access heterogeneity. When relay servers are well-provisioned and selected, per-hop retransmission is more preferable than FEC to recover from random and bursty losses. With layered video coding, prioritized selective retransmissions can further enhance the robustness of conferencing quality against various network impairments. Our findings can be used to guide the design of new video conferencing solutions, as well as an increasing array of network applications that call for high-bandwidth and low-delay data transmissions under a wide range of "best-effort" network conditions.

9. ACKNOWLEDGMENT

The authors would like to thank Prof. Yao Wang, Dr. Hao Hu and Prof. Xinggong Zhang for discussions on video coding techniques. They also would like to thank the anonymous reviewers and the Shepherd Dr. Ming Zhang for their valuable feedbacks and suggestions to improve the paper quality. The work is partially supported by USA NSF under contracts CNS-0953682 and CNS-0916734.

10. 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] Virtual Audio Cable. Homepage. http://software.muzychenko.net/eng/vac.htm/.
[29] Y. Wang, A. Reibman, and S. Lin. Multiple Description Coding for Video Delivery. *Proceedings of the IEEE*, 93, Jan. 2005.
[30] M. Wien, H. Schwarz, and T. Oelbaum. Performance Analysis of SVC. *IEEE Trans. Circuit and System for Video Technology*, 17:1194–1203, Sep. 2007.
[31] Wireshark. Homepage. http://www.wireshark.org/.
[32] 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.
[33] 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.