Skype Video Congestion Control: an Experimental Investigation

Luca De Cicco[a], Saverio Mascolo[a], Vittorio Palmisano[a]

[a] *Politecnico di Bari, Via Orabona 4, 70124, Bari, Italy*

Abstract

The Internet is facing a significant evolution from being a delivery network for static content to an efficient platform for multimedia content delivery. Well-known examples of applications driving this evolution are YouTube Video on Demand, Skype Audio/Video conference, IPTV and P2P video distribution. While YouTube streams videos using the Transmission Control Protocol (TCP), time-sensitive applications, such as Skype Audio/Video conference, employ the UDP because they can tolerate small loss percentages but not delays due to TCP recovery of lost packets via retransmissions. Since, differently from the TCP, the UDP does not implement congestion control, these applications must implement congestion control at the application layer in order to avoid congestion and preserve network stability. In this paper we investigate Skype Video congestion control in order to assess to what extent this application is able to throttle its sending rate to match the unpredictable Internet bandwidth while preserving resource for co-existing best-effort TCP traffic. We have found that: 1) Skype Video adapts its sending rate by varying frame rate, frame quality and video resolution; 2) in many scenarios a Skype video call refrains from fully utilizing all available bandwidth thus not sending videos at the highest possible quality; 3) Skype Video employs an adaptive FEC action that is proportional to the experienced loss rate; 4) the sending rate matches a changing available bandwidth with a transient time as large as a hundred of seconds; 5) the minimum bandwidth required for a video call is 40kbps at 5 frames per second.

1. Introduction

The introduction of new multimedia services such as video on demand, video broadcasting, personal communication and IPTV is pushing the Internet, which was originally conceived to transport time insensitive elastic type data traffic, towards a platform for delivering an ever increasing amount of delay-sensitive multimedia traffic. Examples of such driving applications are Voice over IP, video conferencing (such as Skype), Video on Demand (such as YouTube, DailyMotion), IPTV, peer-to-peer video distribution systems such as, to name few, Coolstreaming [24] and Hulu[1].

A key difference between time-insensitive data traffic and time-sensitive traffic generated by applications such as VoIP or real-time video is that, while data sending rate can be modulated to match the network available bandwidth, real-time audio/video sending rate must follow the source rate. For these reasons data traffic is elastic and is carried over the TCP, which implements congestion control, whereas real-time traffic is inelastic and is carried over the UDP.

Although, in principle, time-sensitive audio/video applications generate inelastic traffic because, due to time-constraints, flows cannot reduce their bandwidth requirements in the presence of congestion as TCP does, in practice well-designed time-sensitive applications must adapt to network available bandwidth at least to some extent. The way this goal can be achieved is by using a congestion control algorithm along with an adaptive video codec that adjusts video quality, frame rate and picture size to match both QoS requirements and network available bandwidth [23], [19].

Differently from TCP flows that continuously probe for network capacity via the Additive Increase Multiplicative Decrease (AIMD) paradigm,

Email addresses: ldecicco@gmail.com (Luca De Cicco), mascolo@poliba.it (Saverio Mascolo), vpalmisano@gmail.com (Vittorio Palmisano)

[1] http://www.hulu.com/

the throughput of the flows originated by means of an adaptive video codec is always bounded by the maximum and minimum bitrate achievable by the specific codec.

YouTube is the most relevant example of video distribution system and employs the TCP to generate elastic traffic. In particular, the video stream is buffered at the receiver for a while before the playing is started. In this way, short-term mismatch between the source video rate and the network available bandwidth are averaged out and masked by the playout buffer. On the other hand, Skype is one of the most prominent example of applications providing unicast Audio/Video calls over UDP. Skype Audio/Video is a closed source application. Skype Audio employs several audio codecs such as G729, SVOPC, iSAC, iLBC, SILK, whereas Skype Video employs the VP7 codec provided by On2[2].

In the literature several papers proposed to design new transport protocols tailored to transport multimedia content. A review of these protocols along with a proposed one can be found in [12]. Among these protocols, the only congestion control for multimedia flows that has been proposed for IETF standardization is the TCP Friendly Rate Control (TFRC) [14], [18]. Implementation of TFRC is complex since it requires ad-hoc tuning of many parameters. For instance, to enable a VoIP application, it has been necessary to propose the small packet variant [11]. For these considerations, the state of art of today running real-time applications such as Skype Audio/Video employs the UDP. Since the UDP does not implement congestion control, it is mandatory for a well-designed multimedia application to include an efficient congestion control algorithm [9], otherwise the Internet would experience a congestion collapse as the one happened in the eighties before the introduction of TCP congestion control [22].

Skype today counts over 40 millions active users of which 17 millions are concurrent users[3]. In a recent report, Skype claims that more than 25 billions minutes of Skype video calls have been generated [1], resulting in the most used desktop video conference application. For this reason, it is increasingly important to assess if and how Skype contributes to network congestion and how it affects TCP responsive traffic, which still contributes the most part of the Internet traffic [9, 14]. Moreover, it is of interest to study if there is room for improving design and implementation of adaptive videoconference applications.

This work investigates how a Skype Video flow behaves when sharing the Internet with other TCP and Skype Video flows. The goal is to determine the responsiveness of Skype Video to the unpredictable time-varying Internet bandwidth in terms of transient times needed to match the available bandwidth, fairness with respect to coexisting TCP and Skype flows, frames per second and packet loss rate.

At the best of authors' knowledge, this is the first investigation of Skype Video congestion control.

The rest of the paper is organized as follows: in Section 2 we summarize the related work; in Section 3 we summarize the knowledge made available to the public on the adaptive video codec used by Skype; in Section 4 we briefly describe the experimental testbed and the tools we have developed in order to carry out the experiments; in Section 5 we present and discuss the experimental results; Section 6 describes the adaptive FEC algorithm employed by Skype. Finally, Section 7 draws the conclusions of the paper.

2. Related Work

It is well-known that the best-effort Internet cannot provide guaranteed resources for real-time multimedia applications. The first attempts to address this problem date back to early '90s and show the benefits of using congestion control schemes to control the rate generated by a video source [17], [4]. In particular, in [17] authors show that by using explicit feedback information provided from the network it is possible to implement a control algorithm that achieves graceful degradation when congestion occurs. In [4], authors show the benefit of implementing a very basic congestion control scheme in conjunction with the adaptive video codec H.261 in a video conferencing system. In the past years, the idea of applying congestion control to multimedia systems [10] has consolidated itself and it has led to several design efforts [12],[14],[18],[21].

One of the most prominent and successful applications which implement real-time audio/video transmission over the Internet is Skype.

Recently, an experimental investigation has revealed that Skype VoIP implements some sort of congestion control by varying the sending rate to

[2]On2 Truemotion VP7 codec, http://www.on2.com/

[3]http://share.skype.com/stats_rss.xml

match the network available bandwidth to some extent [7]. Moreover, in [8] a mathematical model of Skype VoIP flows is provided, revealing that the main driver of the congestion control algorithm is the estimated loss ratio.

Other relevant papers on Skype can be grouped in the following categories: i) P2P network characterization; ii) perceived quality of the Skype VoIP flows.

First papers on Skype mainly focused on the characterization of the P2P network built by Skype in order to enlight, at least partially, interesting details on its architecture and on the NAT traversal techniques [3],[13].

Moreover, several studies have been carried out on the quality provided by the Skype VoIP calls in different scenarios by using metrics such as mean opinion score (MOS) and Perceptual Evaluation of Speech Quality (PESQ) [2], [6], [15], [16] or by defining metrics based on packet level measurements such as round trip time, input rate and duration of the calls [5].

3. Video Codec Employed by Skype

In this Section we summarize all the public information concerning the video codec used by Skype Video that are reported in [20]. Since 2005, Skype employs the proprietary Video Codec TrueMotion VP7 provided by On2 in order to manage one-to-one videoconferencing. The codec supports real-time video encoding and decoding using a "datarate control" which adjusts frame quality, video resolution and number of frame per seconds to adapt to bandwidth variations. Moreover, the white paper [20] states that a model of the client buffer level is employed in order to control those variables, but no further details are provided. Regarding the bitrates produced by VP7, On2 claims to provide video transport starting from bitrates as low as 20 kbps, whereas no information is given on the maximum bitrate.

4. Experimenting with Skype Video: the Skype Measurement Lab

In order to investigate how Skype Audio/Video connections behave when network bandwidth changes over time, we have developed a set of tools that allows real network experiments to be deployed over one or more hosts and to measure and log

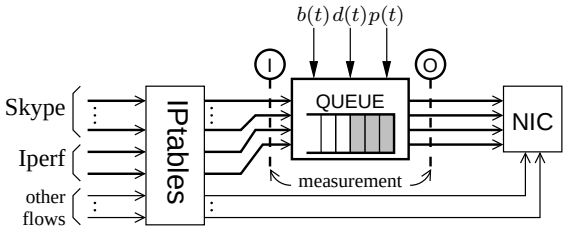


Figure 2: IPQshaper functional diagram: filtered traffic (thick arrows) is routed to a userspace queue whose bandwidth $b(t)$, delay $d(t)$ and packet drop probability $p(t)$ can be set

Skype key variables. Figure 1 shows the testbed set-up which is made of two real hosts: on each host one or more Skype applications are started with or without concurrent Iperf generated TCP traffic[4].

On each host we deployed IPQshaper, which is a software we have developed to perform per-flow measurements. The tool, whose schematic is shown in Figure 2, allows us choosing a set of processes and filtering the generated traffic using IPTables rules. The filtered traffic is then routed to a userspace queue using the IPtables QUEUE target[5]. At the input of this queue, marked with an "I" in Figure 2, the per-flow incoming rate $r_i(t)$ is measured. At the output of the queue, that is marked with an "O" in Figure 2, the per-flow outgoing rate $r_o(t)$ is measured, so that the loss rate $l(t)$ experienced by the flow can be computed as $r_i(t) - r_o(t)$.

In order to emulate a LAN or WAN scenarios, the packets in the queue are drained at configurable rate $b(t)$, which models the bandwidth available at the bottleneck drop tail queue. Finally, the tool allows packets to be delayed of an amount $d(t)$ and dropped with probability $p(t)$.

The throughput is defined as $\Delta sent/\Delta T$, the loss rate as $\Delta loss/\Delta T$ and the goodput as $(\Delta sent - \Delta loss)/\Delta T$, where $\Delta sent$ is the number of bits sent in the period ΔT, $\Delta loss$ is the number of bits lost in the same period. We have considered $\Delta T = 0.4$ s in our measurements.

It is of fundamental importance to perform experiments in a controlled environment in order to allow tests be reproducible. We provide reproducibility by employing a controlled LAN as a testbed and using the same video sequence as input. In fact, using the input obtained by a webcam

[4] http://dast.nlanr.net/Projects/Iperf/
[5] NetFilter: http://www.netfilter.org/

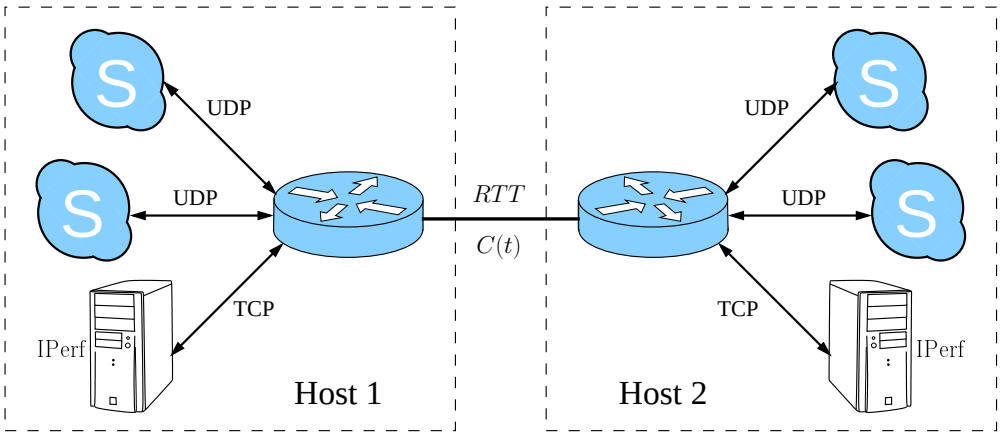


Figure 1: Experimental testbed

would generate an encoded bitrate that depends on the particular video content, thus not allowing experiments to be reproduced.

To this end we have developed a software called *Skype Measurement Lab (SML)*, which allows a desired video source to be injected as input to Skype. In particular, we have modified the `GStreamer` plug-in `gst-fakevideo`[6], which generates a fake `/dev/video` device that simulates a video source (like a webcam) using a technique similar to the one employed by Skype Audio Dsp Hijacker[7]. Another important feature of the SML is the automatic logging of all the information contained in the Skype technical call information tooltip, which is displayed when the "Technical Call Infos" option is enabled in the preferences. To the purpose, we have modified the QT 4.X user interface library[8] that is used by this client (freely available as source code) in order to periodically log all information contained in the call tool-tip, which includes among others: RTT, jitter, video resolution, video frame rate, estimated sent and received loss percentages.

The experiments have been run using the Linux Skype client version and the standard *Foreman* YUV test sequence[9]. The audio input has been muted in order to analyze only the network traffic generated by video flows. From now on, the *RTT* of the connection is set at 50 ms and the queue size at the two hosts is set equal to the bandwidth delay product unless otherwise specified.

[6] http://code.google.com/p/gstfakevideo/
[7] Skype DSP hijacker: http://195.38.3.142:6502/skype/
[8] QT 4.3: http://trolltech.com/products/qt
[9] http://www.cipr.rpi.edu/resource/sequences/sif.html

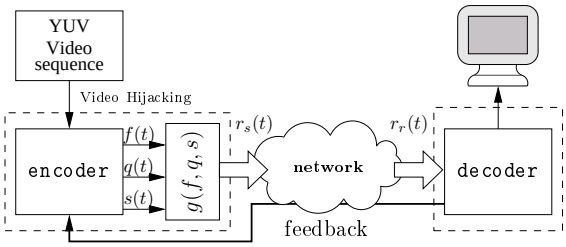


Figure 3: Model of the Skype Video rate adaptation scheme

5. Experimental investigation of Skype Video dynamics

In this Section we aim at investigating how Skype Video flows throttle their sending rates when changes of available bandwidth occur and how Skype flows behave when concurrent TCP flows share the bottleneck. To the purpose we consider step-like variations of the available bandwidth since this is a widely used and efficient practice in control theory when testing the dynamic response of a system to a stimulus. Indeed, the step response of a system reveals key features of the system dynamics such as transient time and degree of stability.

In this case we are interested in revealing the transient dynamics of the Skype flows in response to bandwidth increase/decrease or to joining/leaving of TCP flows.

Figure 3 shows the overall scheme of a desktop video conferencing system. An encoder adapts the video flow sending rate $r_s(t)$ by throttling the frame quality $q(t)$, the video resolution $s(t)$ and the number of frames per second (fps) $f(t)$ based on feed-

back reports sent by the receiver. It is reasonable to conjecture that the feedback variables used that throttle $q(t)$, $s(t)$ and $f(t)$ are the available bandwidth, loss rate and jitter [8]. Throughout the discussion of the experimental results we will illustrate the effect of variable network conditions on the three control variables throttled by Skype.

5.1. Skype Video response to a step variation of available bandwidth

We start by investigating the behaviour of one Skype flow accessing a bottleneck link whose bandwidth capacity changes following a step function with minimum value $A_m = 160\,\text{kbps}$ and maximum value $A_M = 2000\,\text{kbps}$. The aim of this experiment is to show how Skype flows behave when the network available bandwidth suddenly increases; this is particularly important to assess Skype responsiveness in grabbing the available bandwidth.

In this experiment no concurrent traffic is injected. Figure 4 shows throughput and frame rate dynamics obtained by repeating four experiment runs. The video flow starts sending at a very low rate and achieves a steady state sending rate of roughly 80 kbps, well below the available bandwidth of 160 kbps. When the available bandwidth increases at $t = 50\,\text{s}$, the sending rate reaches an average bitrate slightly below 450 kbps, after a long transient time of roughly 100 s.

Now, let us focus our attention on the three variables $f(t)$, $q(t)$ and $s(t)$ that are throttled by the video codec to match the network available bandwidth. In the four experiments the resolution $s(t)$ of the videos produced by Skype was set at 320×240 pixels and kept unchanged throughout all the experiments; the frame rate $f(t)$ decreases from an initial value of 15 fps to a value of around 10 fps in less than 10 s. After the step increment of the available bandwidth at $t = 50$ s, $f(t)$ starts to increase at roughly $t = 85\,\text{s}$ and then it oscillates around the value of 15 fps; the sending rate $r_s(t)$ starts to increase at $t = 50\,\text{s}$ whereas the value of $f(t)$ remains roughly constant in the time interval $[50, 85]\,\text{s}$ which means that the quality $q(t)$ is increased.

A further insight can be obtained by looking at Figure 5 that shows packet sizes and cumulative losses of the four experiment runs: the packet size increases in the time interval $[50, 85]\,\text{s}$ whereas $f(t)$ is left almost unchanged, which means that the increment of the sending rate is due to an improved quality $q(t)$.

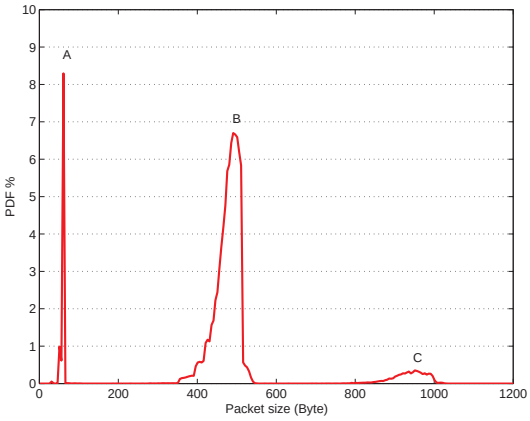


Figure 6: Packet size probability density function (PDF): packets of type A contain feedback information, packets of type B are video packets without redundancy, and packet of type B are packets with redundancy.

Figure 5 reveals also an interesting correlation between packet size and packet losses: every time a large loss event occurs (marked by a large step in the cumulative line shown in the Figure 5) the packet size doubles, thus meaning that Skype employs a FEC scheme to counteract packet losses. On the other hand, Skype does not trigger a packet size increment when the entity of the loss is considered negligible as it can be inferred by looking at the S3 plot at time $t = 177\,s$, which shows that a small step increase in the cumulative loss curve does not induce a doubling of the packet size.

In order to provide a further insight, Figure 6 shows the packet size probability density function computed for time $t > 150$s when the transient due to the bandwidth variation at $t = 50$s ends. The probability density function shows three peaks: the first one (A) occurs at packet size equal to 61 bytes and we conjecture that such packets contain feedback information sent to the other peer of the communication; the second peak (B) occurs at packet size equal to 491 bytes and we conjecture that it reveals that the size of the packets containing the video are in the range $[350, 550]$ bytes; the third peak (C), occurring at packet size equal to 961 bytes, reveals FEC packets in the size $[720, 1035]$ bytes that indeed is roughly two times the normal packet size.

To summarize, the main result of this first experiment is that a Skype Video flow produces a sending rate that achieves the maximum value of

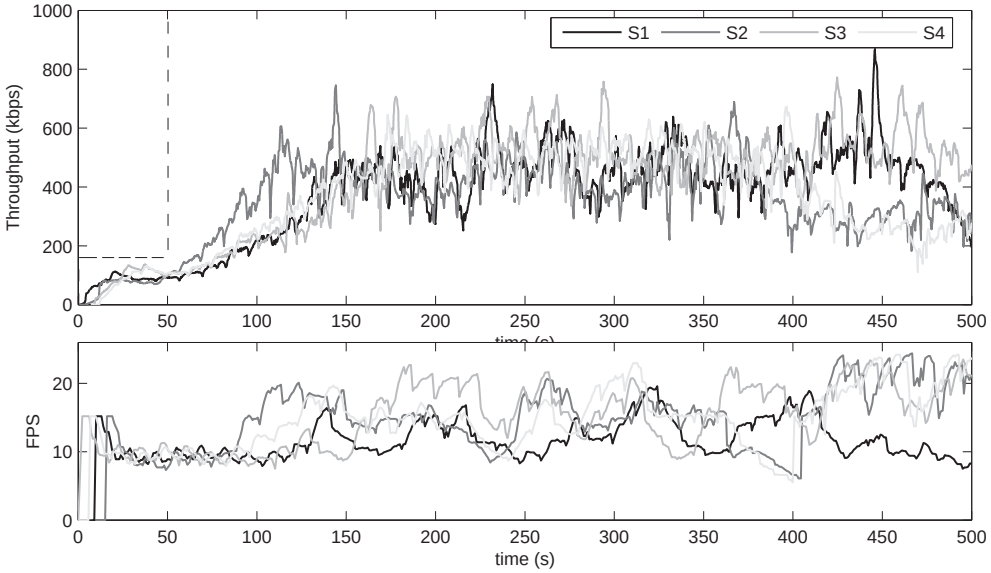


Figure 4: Skype Video response to a step change of available bandwidth at $t = 50\,\mathrm{s}$

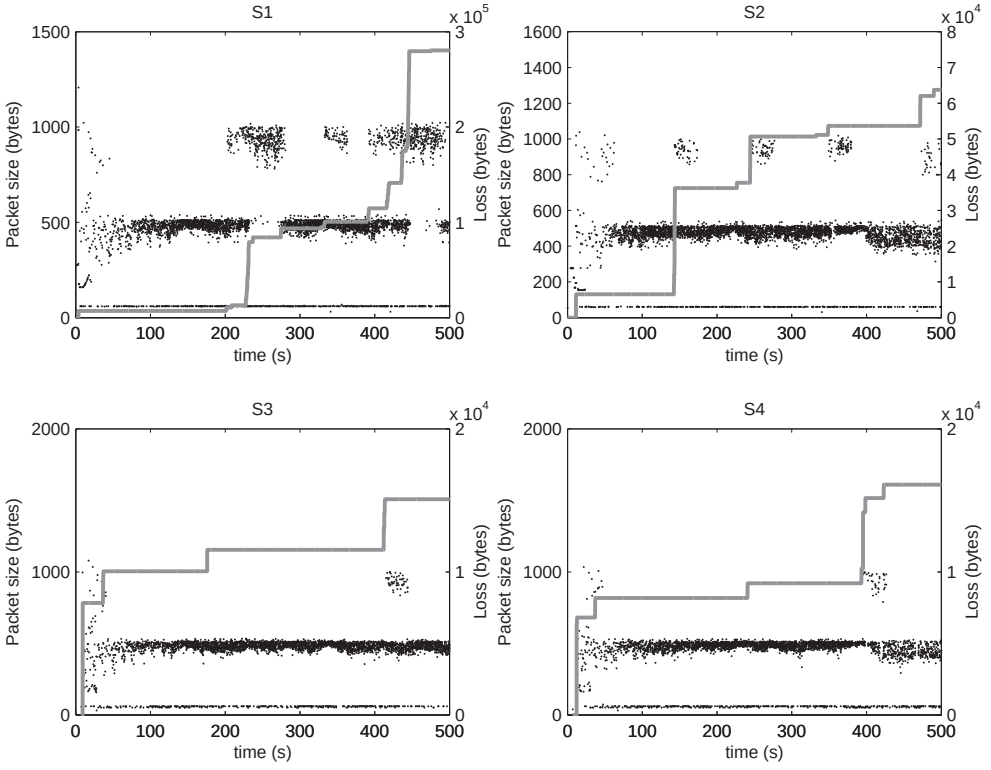


Figure 5: Packet size (black points) and cumulative bytes lost (gray lines) of the four Skype flows in response to a step change of available bandwidth at $t = 50\mathrm{s}$

around 450 kbps after a transient of 100 s and employs FEC mechanism to counteract large packet losses. Moreover, in this experiment we have found that around 10% of packets sent in a Skype Video flow are feedback packets (type A), around 83% are packets without redundancy (type B), and the remaining 7% are FEC packets containing redundancy (type C).

5.2. Skype response to a staircase variation of available bandwidth

In this scenario we aim at investigating how a Skype Video flow adapts to small step-like increments/decrements of the available bandwidth. To the purpose we start by allowing the available bandwidth to vary in the range $[160, 1000]$ kbps. By using the knowledge on transient times that we have gathered in the previous scenario, we set bandwidth variations to occur every 100 s in order to let sending rates to extinguish their transients. In particular, in the first half of the experiment, the available bandwidth increases every 100 s of 168 kbps, whereas, in the second half, it decreases of the same amount every 100 s.

Figure 7 shows that Skype Video flow is somewhat slow in reaching the steady state since the maximum sending rate is achieved only at $t = 700$ s, when the second half of the experiment is already started. In the first half of the experiment, losses are negligible and the average throughput is around 300 kbps, a value that is well below the available bandwidth that goes up to 1000 kbps.

Regarding the frame rate, after an initial value of $f(t)$= 15 fps, it decreases down to 5 fps at $t = t_A$ when it suddenly increases its value again to 15 fps. This sudden increase in the frame rate occurs in correspondence to a reduction in the video resolution $s(t)$ from 320×240 to 160×120. The frame rate is kept unchanged until $t = t_B$ when the resolution switches back to 320×240 and the frame rate is set again to 15 fps.

We have run a similar experiment in which the available bandwidth varies from 160 kbps down to 20kbps in order to investigate how Skype flows are able to match a thin link capacity. Figure 8 shows that the sending rate follows bandwidth reductions until the capacity drops to 40 kbps. In this condition a minimum frame rate around 5 fps is measured. When the available bandwidth shrinks at 20 kbps, which is the minimum declared bitrate of the Skype video codec [20], the video call is dropped at $t = 375$ s probably because Skype detects a very large packet loss percentage.

In this test, even though the available bandwidth reaches the value of 500 kbps in 200 s and then outpaces this value, the Skype video sending rate does not exceed an average value of only 300 kbps. This means that Skype is not effective to take all the available bandwidth thus losing the possibility of delivering videos at the highest possible quality. The test has also shown that Skype Video is able to shrink the sending rate to match a thin available bandwidth as low as 40 kbps.

5.3. Skype Video response to a square wave available bandwidth

This scenario aims at showing how one Skype Video flows reacts to variable network conditions such as sudden drops/increases of the available bandwidth. To the purpose we evaluate the Skype response to a square wave available bandwidth with a period $T = 400$ s, a maximum value $A_M = 1000$ kbps, which is well above the maximum average sending rate we have measured in the first scenario, and a minimum value $A_m = 160$ kbps. The Skype response is evaluated by measuring the sending rate, the loss rate and the frame rate.

Figure 9 shows that, in the first half of the period, the sending rate reaches an average value of 232 kbps, whereas the frame rate is between 10 and 15 fps with negligible losses. When the first available bandwidth drop occurs at $t = 200$ s, the Skype flow suffers persistent losses which lasts for 19 s. During this interval roughly 128000 bytes are lost which corresponds to an average loss rate of 54 kbps. During the time interval $[217, 400]$ s, the Skype sending rate shrinks at 100 kbps, which is well below the available bandwidth A_m, thus experiencing no packet loss. The frame rate is kept almost unchanged, except for a short transient time during which it is reduced, which means that the encoder decreases the quality in the time interval $[200, 400]$ s. The flow starts to increase its rate when the bandwidth is up again in the interval $[400, 600]$s reaching an average sending rate of 238kbps. During this interval the video flow experiences significant losses due to a high burstiness of the sending rate in the interval [480, 540] s. During the last interval [600,800]s, the sending rate achieves an average value of 78kbps with a frame rate that increases up to 15fps at $t = 735$ s, when the video resolution is reduced from 320×240 to 160×240 so that the resulting sending rate is kept unchanged.

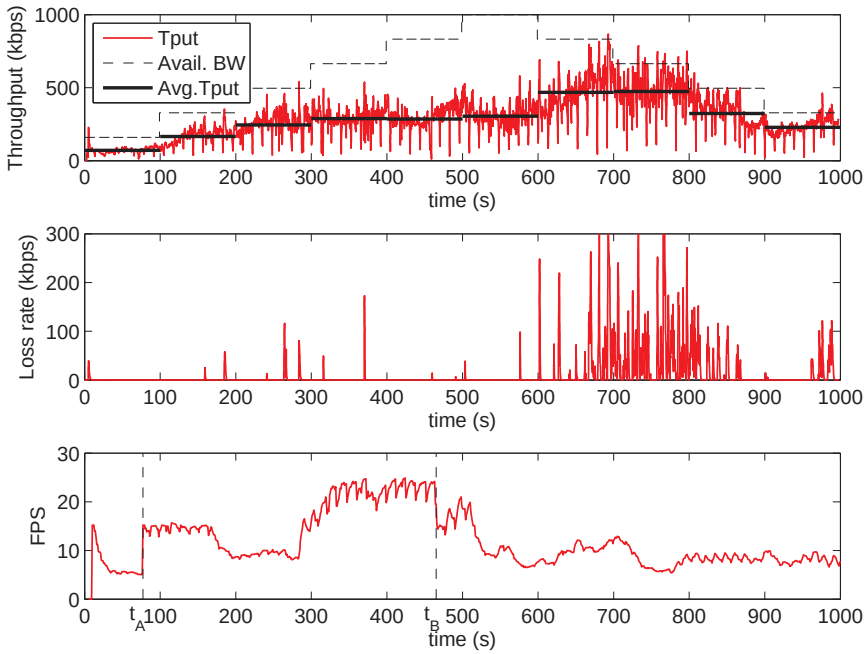


Figure 7: Skype Video flow dynamics in response to a time-varying available bandwidth

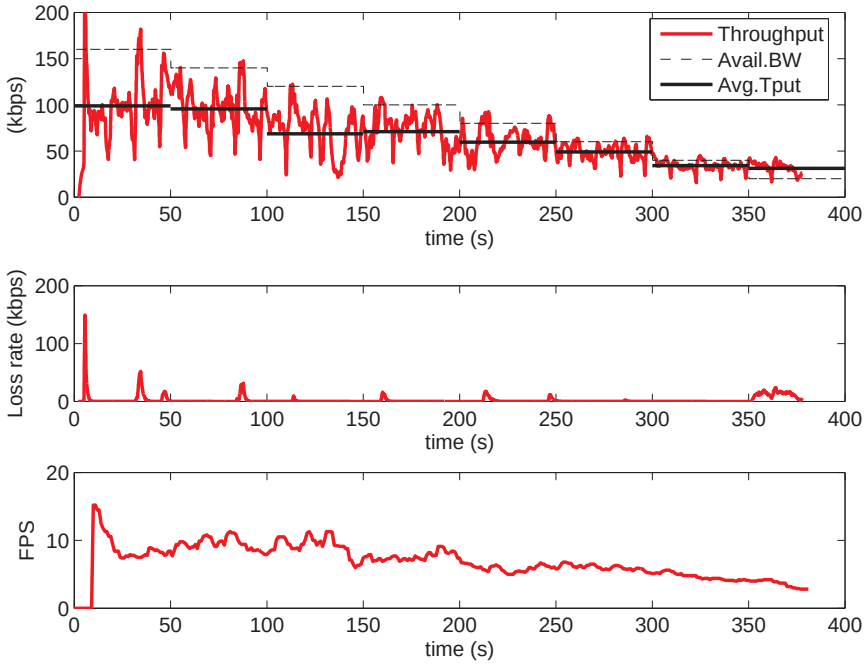


Figure 8: Skype Video response to an available bandwidth starting at 160 kbps and decreasing down to 20 kbps

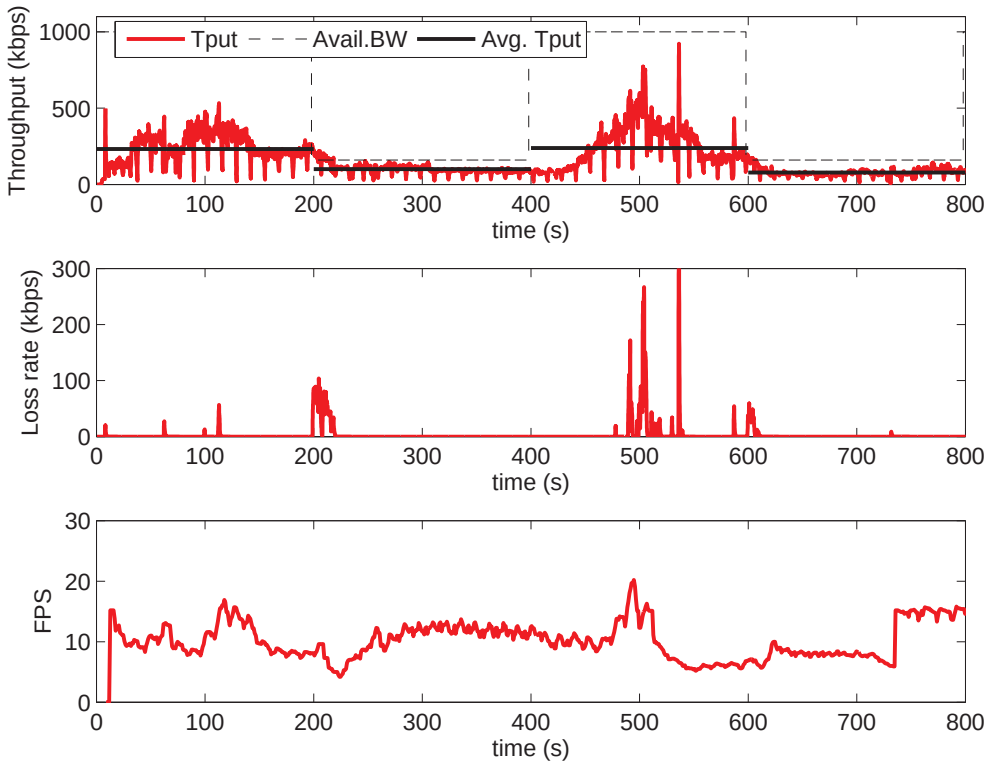


Figure 9: Skype Video response to a square wave available bandwidth with period $T = 400\ s$

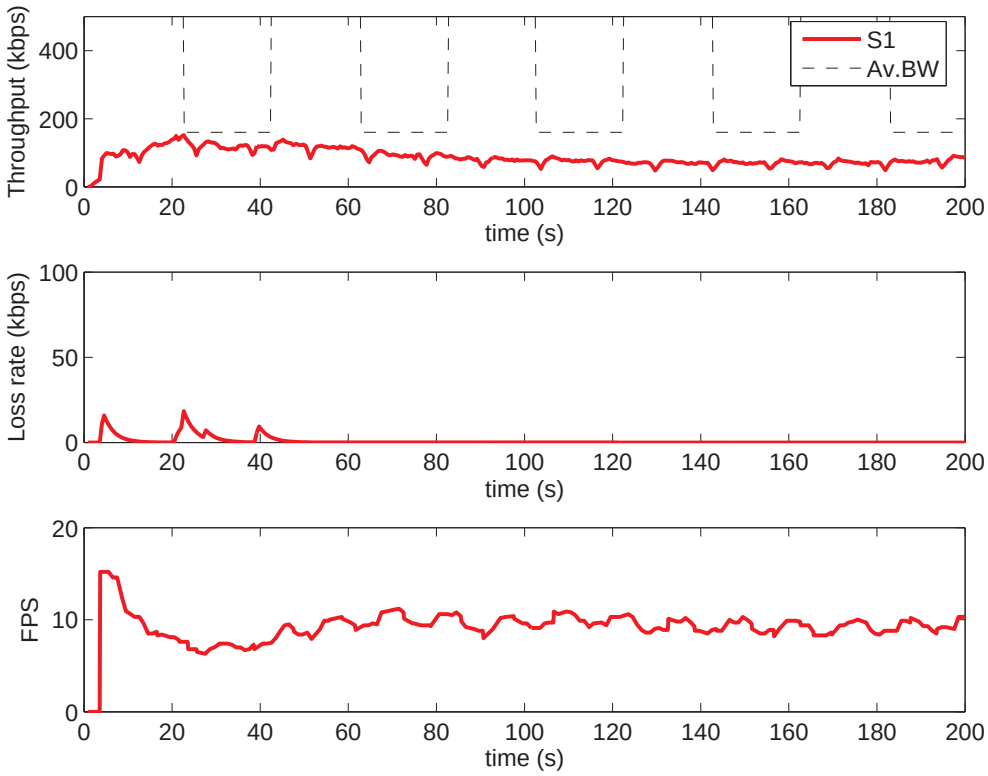


Figure 10: Skype Video response to a square wave available bandwidth with period $T = 40\ s$

We have also run a similar experiment by using a square wave with a period of 40 s in order to asses the reaction speed of the control algorithm. Figure 10 shows throughput, loss rate and frame rate in this scenario. We have found that the input rate never exceeds A_m, which means that the reaction speed of the control algorithm is too slow with respect to bandwidth variations. As a consequence the frame rate is not able to reach values higher than 12fps.

Again, the conclusion of this test is that Skype is not effective to grab all the available bandwidth thus losing the possibility of delivering the video at the highest possible quality.

5.4. Two Skype Video flows over a square wave available bandwidth

In this subsection we aim at investigating the effect of multiple video flows on the stability of the network. To the purpose, we set up a scenario in which one Skype Video flow S_1 is started at $t = 0$ and a second flow S_2 is started at $t = 50$ s . The available bandwidth varies as a square wave of period $T = 400$ s with a maximum value $A_M = 384$ kbps and a minimum value $A_m = 160$ kbps. We have selected $A_M = 384$ kbps since this is the downlink capacity of an UMTS link and is smaller than the maximum average sending rate of Skype Video, which we have measured is around 450 kbps. In this scenario, the two Skype Video flows will create a congested bottleneck. Again, we have set $A_m = 160$ kbps, since with a lower value calls are dropped.

Figure 11 (a) shows that, at the beginning, the first flow increases its sending rate similarly to what we have shown in previous experiments. Moreover, the rate is kept increasing also when the second Skype flow joins the bottleneck at $t = 50$ s. However, for $t > 90$ s the first flow S_1 starts to leave bandwidth to S_2 that in turn increases its sending rate until the first bandwidth drop occurs at $t = 200$ s. It can be seen that S_2 generates a high and persistent loss rate at around 80 kbps which lasts for around 30 s.

Figure 11 (a) also shows the average throughput in each time interval during which the bandwidth is kept constant. In particular, the channel link utilization is 68% for $t \in [0, 200]$ s, 83% for $t \in]200, 400]$ s, 46% for $t \in]400, 600]$ s and 61% for $t \in]600, 800]$ s.

It is important to note that when the available bandwidth increases again up to 384 kbps at $t = 400$ s, the two Skype flows do not increase their sending rate thus not taking the opportunity to send video at the best possible quality. For what concerns fairness issues, the two flows share the bottleneck in a fair way (the Jain fairness index is 0.97).

Figure 11 (b) shows packet size (dots) of the two flows and lost bytes (continuous line). The Figure confirms what we have reported in Section 5.1, i.e. Skype Video increases the FEC action when packets are lost.

Again, this test shows that Skype Video is not efficient in getting full bandwidth utilization thus losing the possibility of delivering a video with a higher quality.

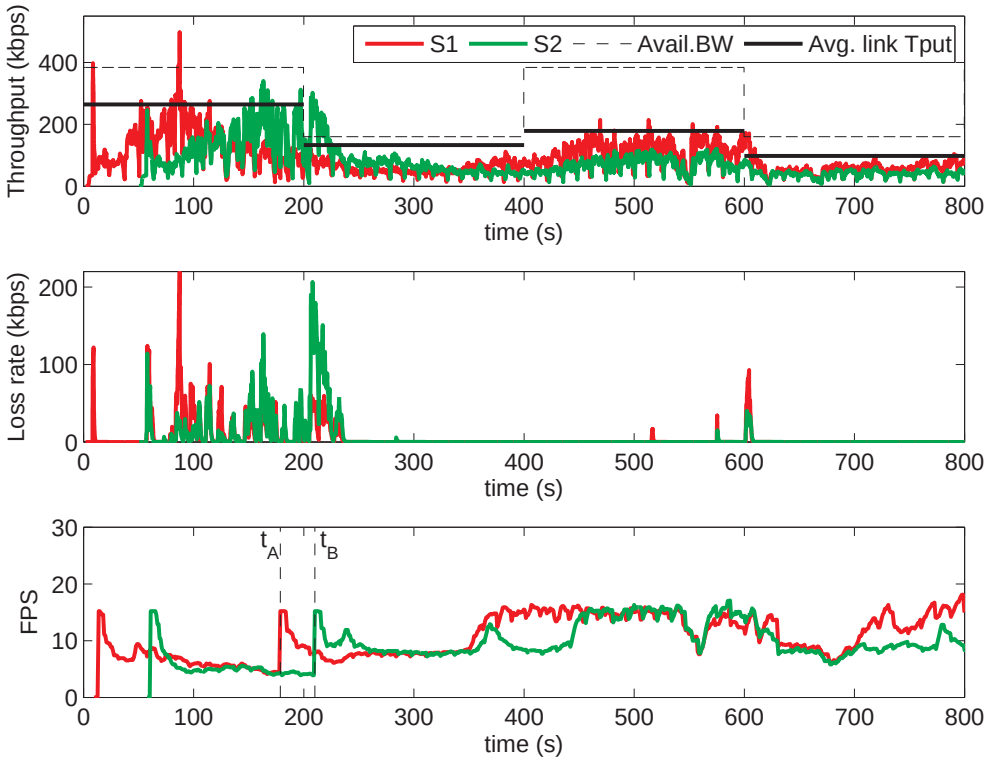
5.5. One Skype Video flow with concurrent TCP flows

In this subsection we investigate the Skype Video behaviour when the network bandwidth is shared with TCP flows. We consider a link with a constant capacity of 384 kbps. A Skype Video call starts at $t = 0$, the first TCP flow starts at $t = 200$ s and a second one starts at $t = 400$ s. Figure 12 (a) shows throughput whereas Figure 12 (b) shows cumulative losses of Skype and TCP flows along with packet size and frame rate of the Skype flow.

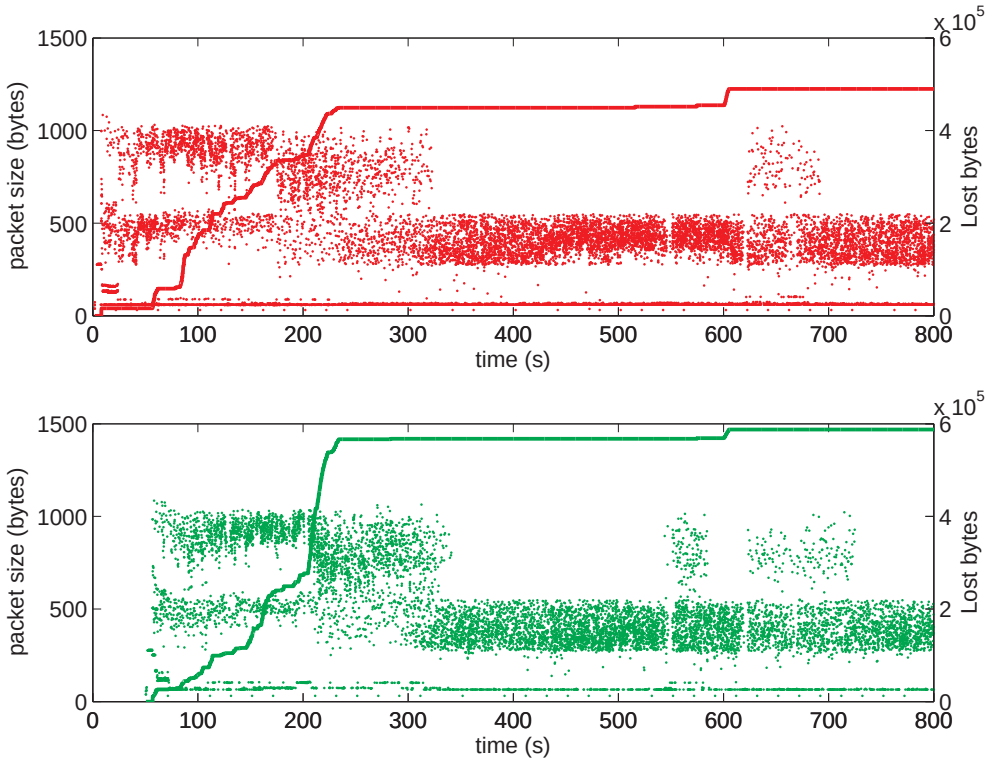
When TCP1 enters the bottleneck, the Skype Video flow releases bandwidth by decreasing its sending rate. The two flows share the bandwidth fairly until $t = 250$ s when the Skype flow starts decreasing its sending rate leaving bandwidth to TCP1. Figure 12 (a) shows that the steady state is still not reached when TCP2 flow starts. After the TCP2 flow is started, the bandwidth is shared in a somewhat fair way among the flows in the time interval $[400, 1000]$ s, except during the interval $[550, 700]$ s during which the Skype flow increases its bandwidth obtaining a significantly larger bandwidth share.

In order to understand the reason that triggers the increasing of the sending rate of the Skype flow, let us look at the packet size evolution shown in Figure 12 (b).

The Figure 12 (b) shows that the Skype Video packet size increases in the time intervals $[120, 180]$s, $[375, 494]$ s and $[590, 681]$ s which means that Skype has activated the FEC action. This is confirmed by the frame rate dynamics that does not follow the sending rate increase in those intervals. It is worth noticing that the step change in the frame rate evolution that occurs at $t = 436$ s corresponds

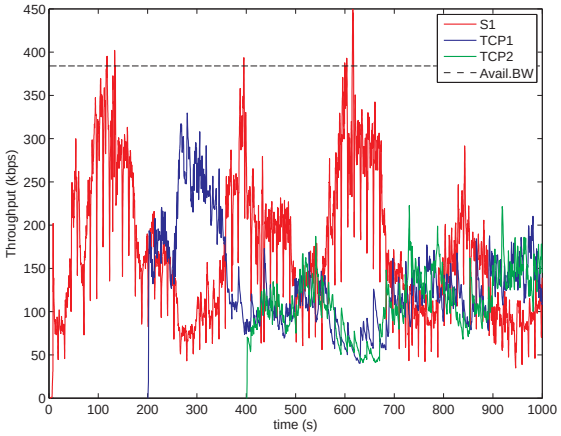


(a) Throughput, loss rate and frame rate time evolutions

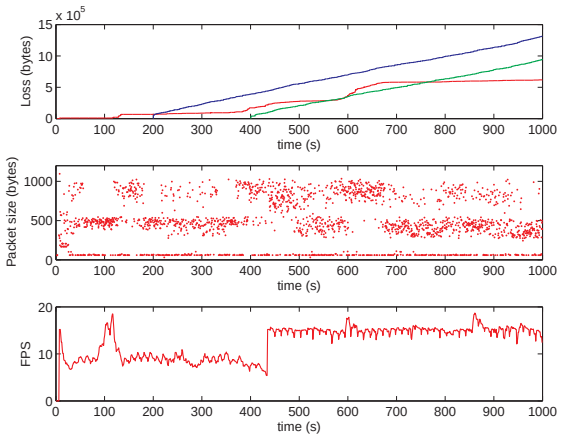


(b) Packet size (dots) and lost bytes (continuous line) evolutions

Figure 11: Dynamics of two concurrent Skype Video flows in the presence of a square wave available bandwidth



(a) Throughput of Skype Video and two TCP flows



(b) Cumulative losses, packet size and frame rate evolutions

Figure 12: One Skype Video flow over a link with 384 *kbps* capacity sharing the bottleneck with two concurrent TCP flows started at $t = 200\,s$ and $t = 400\,s$

Table 1: Throughput, loss rate, loss ratio and channel utilization for the Skype and the two TCP flows

	Tput (kbps)	Loss rate (kbps)	Loss ratio	Channel util.
S1	162.5	6.0	3.7%	42.3%
TCP1	101.6	12.3	12%	26.4%
TCP2	102.3	12.6	12%	26.6%

to a decrease in the video resolution from 320×240 to 160×120. The cumulative losses graph shown in Figure 12 (b) clearly suggests that the increments in the FEC are triggered by the increasing of lost bytes (see also Section 5.1). In particular, the Skype flow loses 258000 bytes in the interval $[590, 681]$s and exhibits an unfair behaviour with respect to the TCP flows.

In order to evaluate how the Skype flow behaves when sharing the link with other TCP flows, Table 1 reports average values of throughput, loss rates, loss percentages and channel utilizations of all the flows for $t > 400\,s$. Results show that Skype takes a larger share of channel capacity, whereas the two TCP flows share the left over bandwidth equally.

The overall conclusion here is that Skype Video seems more aggressive than the TCP, because of the FEC action that seems to unresponsively increase the sending rate when losses are experienced.

5.6. Effect of reverse traffic on a Skype Video flow

This scenario aims at showing the effect on a Skype Video flow when congestion is present on the reverse path. To the purpose, in this experiment, the available bandwidth is set at 2000 kbps so that the Skype Video flow is not be able to generate congestion on the forward path of the bottleneck. A Skype Video flow is started at $t = 0\,s$ and three TCP connections start along the reverse path at $t = 200\,s$ and leave at $t = 400\,s$.

Figure 13 shows that when the TCP flows join the path at time $t = 200$ s the Skype sending rate decreases from a steady state value of around 450 kbps to a value of around 190 kbps (corresponding to a frame rate of 9 fps) even though the available bandwidth on the forward path does not vary. By looking at the *RTT* evolution shown in Figure 13, the decreasing in the sending rate seems to be triggered by the increased *RTT* on the reverse path that is due to the slow start phase of the TCP flows. After the TCP slow start phase ends, the *RTT* decreases and Skype starts increasing the sending rate again

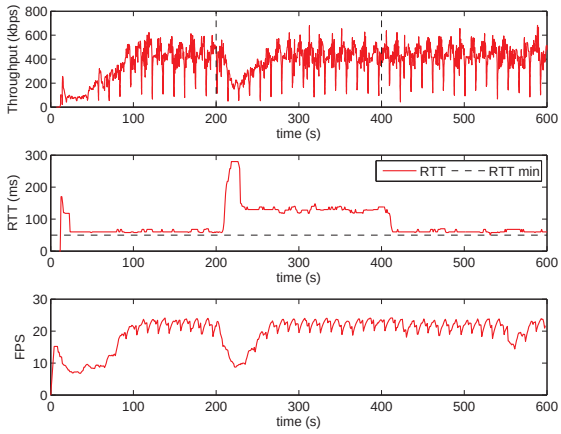


Figure 13: One Skype Video flow with TCP reverse traffic present for $t \in [200, 400]$ s

up to its steady state of 450 kbps. In this scenario the sending rate is driven mainly by the frame rate as it is can be inferred by noting that the frame rate dynamics follows the sending rate dynamics.

In conclusion, the control algorithm employed by Skype Video is sensitive to the congestion on the reverse path.

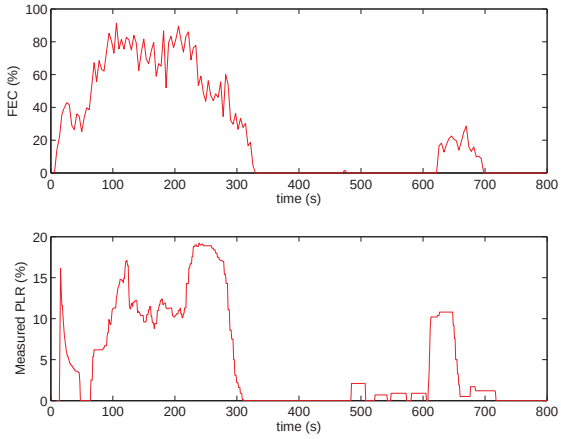
6. Skype Video adaptive FEC algorithm

In Section 5 we have discussed the main properties of Skype Video flows in several different scenarios. In particular, we have shown that packets generated by Skype Video (see Figure 5 and Figure 6) can be of three different types: control packets, video data packets and video data packets with redundancy. We also noticed that packets with redundancy are sent, i.e. FEC action is on, when Skype detects packet losses (see Figure 5 and Figure 11 (b)).

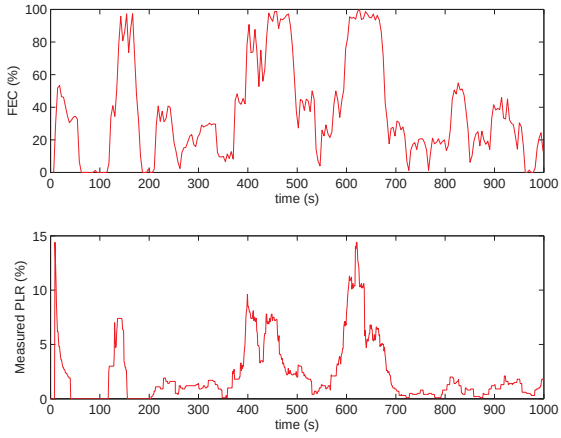
In this Section we aim at explaining how, and to what extent, FEC action is controlled by Skype.

To the purpose, we reconsider two experiments: 1) two Skype Video flows over a square wave available bandwidth (Section 5.4); 2) one Skype Video flow with concurrent TCP flows (Section 5.5). We have chosen these experiments because they exhibited the highest number of lost packets, thus triggering the FEC action for a prolonged amount of time.

In order to evaluate the percentage of video packets with redundancy at a time instant t, we employ the following algorithm. For each time $t_k = k\Delta T$



(a) Two Skype flows over a square wave available bandwidth



(b) One Skype Video flow with concurrent TCP flows

Figure 14: FEC action vs Measured packet loss ratio

(with $\Delta T = 4$s) we evaluate the number of video packets without redundancy $n_v(t_k)$ and the number of video packets with redundancy $n_{vr}(t_k)$ contained in the time interval t_k. We define the FEC action $FEC(t_k)$ at time t_k as the ratio between video packets with redundancy n_{vr} and the total number of packets $n_v(t_k) + n_{vr}(t_k)$ sent in the current time interval:

$$FEC(t_k) = \frac{n_{vr}(t_k)}{n_v(t_k) + n_{vr}(t_k)} \quad (1)$$

By comparing the loss ratio $\hat{l}(t)$ estimated by Skype as shown in the "technical tooltip" and the FEC action $FEC(t)$ computed using (1) we have found a proportionality between those two signals. Figure 14 (a) and Figure 14 (b) show a comparison between the packet loss ratio measured by Skype Video and the FEC action $FEC(t)$ computed using (1) in both considered scenarios.

Both the figures show that Skype Video adaptively throttles the FEC action $FEC(t)$ roughly proportionally to the estimated packet loss ratio.

7. Conclusions

We have carried out an experimental investigation of Skype Video flows behaviour in the presence of time varying network conditions and TCP traffic. We have found that a Skype Video call uses the frame rate, the packet size and the video resolution in order to throttle its sending rate to match the network available bandwidth. The obtained results have shown that a Skype Video call roughly requires a minimum of 40 kbps available bandwidth to start and it is able to fill in a bandwidth up to 450 kbps. Thus it can be said that a video flow is made elastic through congestion control and adaptive codec within that bandwidth interval.

We have also measured that a Skype Video sending rate exhibits a large transient time when it increases to match an increment of the available bandwidth. Moreover, we have found that in many scenarios a Skype video call refrains from fully utilizing all available bandwidth, which means that a video call is not sent at the best quality that a network would permit. Regarding coexistence with TCP flows, Skype Video seems more aggressive than the TCP because of the FEC action that unresponsively increases the bandwidth even when losses are experienced. Furthermore, we have found that when congestion is present on the reverse path, Skype Video unduly reduces its sending rate. Finally, we have shown that Skype Video employs an adaptive FEC action that is roughly proportional to the measured packet loss ratio.

8. Acknowledgment

This work has been partially supported by Financial Tradeware plc.

References

[1] Skype fast facts - q4 2008. Available online: http://ebayinkblog.com/wp-content/uploads/2009/01/skype-fast-facts-q4-08.pdf.

[2] R. Barbosa, C. Kamienski, D. Mariz, A. Callado, S. Fernandes, and D. Sadok. Performance evaluation of P2P VoIP application. In *Proc. of ACM NOSSDAV '07*, 2007.

[3] S. Baset and H. Schulzrinne. An Analysis of the Skype Peer-to-Peer Internet Telephony Protocol. In *Proc. of IEEE INFOCOM '06*, Apr. 2006.

[4] J.-C. Bolot and T. Turletti. A rate control mechanism for packet video in the internet. In *Proc. of IEEE INFOCOM '94*, pages 1216–1223, 1994.

[5] K. Chen, C. Huang, P. Huang, and C. Lei. Quantifying Skype user satisfaction. In *Proc. of ACM SIGCOMM '06*, Sept. 2006.

[6] W. Chiang, W. Xiao, and C. Chou. A Performance Study of VoIP Applications: MSN vs. Skype. In *Proc. of MULTICOM '06*, 2006.

[7] L. De Cicco, S. Mascolo, and V. Palmisano. An Experimental Investigation of the Congestion Control Used by Skype VoIP. In *Proc. of WWIC '07*, May 2007.

[8] L. De Cicco, S. Mascolo, and V. Palmisano. A Mathematical Model of the Skype VoIP Congestion Control Algorithm. In *Proc. of IEEE Conference on Decision and Control '08*, Cancun, Mexico, Dec. 9–11, 2008.

[9] L. Eggert and G. Fairhurst. UDP Usage Guidelines for Application Designers. *RFC 5405*, Nov. 2008.

[10] S. Floyd and K. Fall. Promoting the use of end-to-end congestion control in the Internet. *IEEE/ACM Trans. on Networking (TON)*, 7(4):458–472, 1999.

[11] S. Floyd and E. Kohler. TCP Friendly Rate Control (TFRC): The small-packet (sp) variant. *RFC 4828, Experimental*, 2007.

[12] Grieco, L.A. and Mascolo, S. Adaptive Rate Control for streaming flows over the Internet. *ACM Multimedia Systems Journal*, 9(6):517–532, Jun 2004.

[13] S. Guha, N. Daswani, and R. Jain. An Experimental Study of the Skype Peer-to-Peer VoIP System. In *Proc. IPTPS '06*, Feb. 2006.

[14] M. Handley, S. Floyd, and J. Pahdye. TCP Friendly Rate Control (TFRC): Protocol Specification. *RFC 3448, Proposed Standard*, Jan. 2003.

[15] T. Hoßfeld and A. Binzenhöfer. Analysis of Skype VoIP traffic in UMTS: End-to-end QoS and QoE measurements. *Computer Networks*, 52(3):650–666, 2008.

[16] T. Huang, K. Chen, and P. Huang. Tuning the redundancy control algorithm of Skype for user satisfaction. In *Proc.of IEEE INFOCOM '09*, Apr. 2009.

[17] H. Kanakia, P. Mishra, and A. R. Reibman. An Adaptive Congestion Control Scheme for Real Time Packet Video Transport. In *Proc. of ACM SIGCOMM '93*, San Francisco, USA, 1993.

[18] E. Kohler, M. Handley, and S. Floyd. Designing DCCP: congestion control without reliability. In *Proc. of ACM SIGCOMM '06*, Sept. 2006.

[19] J. McCarthy, M. Sasse, and D. Miras. Sharp or smooth?: comparing the effects of quantization vs. frame rate for streamed video. In *Proc. SIGCHI conference on Human factors in computing systems*, pages 535–542. ACM New York, NY, USA, 2004.

[20] On2 Technologies. TrueMotion VP7 Video Codec White Paper. 10 Jan. 2005.

[21] H. Schulzrinne, S. Casner, S. Frederick, and V. Jacobson. Rtp: A transport protocol for real-time applications. *RFC 3550, Standard*, 2003.

[22] V. Jacobson. Congestion avoidance and control. In *Proc. of ACM SIGCOMM '88*, pages 314–329, 1988.

[23] S. Wenger. H. 264/AVC over IP. *IEEE Trans. Circuits and Syst. Video Technol.*, 13(7):645–656, 2003.

[24] X. Zhang, J. Liu, B. Li, and Y. Yum. CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming. In *Proc. of IEEE INFOCOM '05*, 2005.