Adaptivity in Multimedia Flows

a Feedback Control Approach

Luca De Cicco
ldecicco@gmail.com
http://c3lab.poliba.it/index.php/LDC

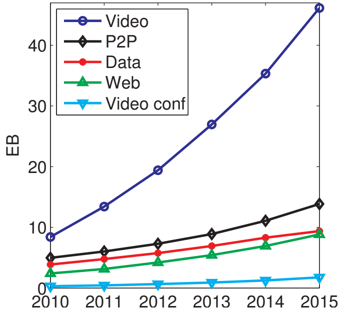
Politecnico di Bari, Dipartimento di Elettrotecnica ed Elettronica

LAAS - Toulouse
16th February 2012

Motivation

Two ongoing trends

- *Video is booming*: video applications will account for more than half of the global traffic in 2012
- *Mobile is growing*: mobile data traffic was 1% of total IP traffic in 2010, and will be 8% in 2015

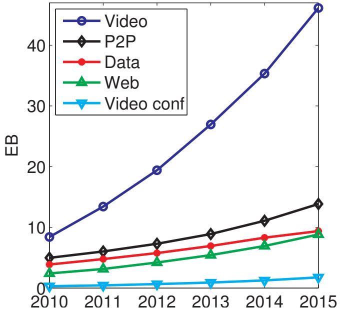


Data Source: CISCO VNI, June 2011

Motivation

Two ongoing trends

- *Video is booming*: video applications will account for more than half of the global traffic in 2012
- *Mobile is growing*: mobile data traffic was 1% of total IP traffic in 2010, and will be 8% in 2015



Data Source: CISCO VNI, June 2011

Issues

- Internet was not designed for delay-sensitive traffic
- Bandwidth is unpredictable in best-effort Internet
- Mobile devices have limited CPU, battery, screen, and network bandwidth resources

The challenge

Content producer PoV

Provide a seamless multimedia experience maximizing the QoE obtainable given devices and network constraints

The challenge

Content producer PoV

Provide a seamless multimedia experience maximizing the QoE obtainable given devices and network constraints

The building block

The key technological enabler is **Adaptivity**

The challenge

Content producer PoV

Provide a seamless multimedia experience maximizing the QoE obtainable given devices and network constraints

The building block

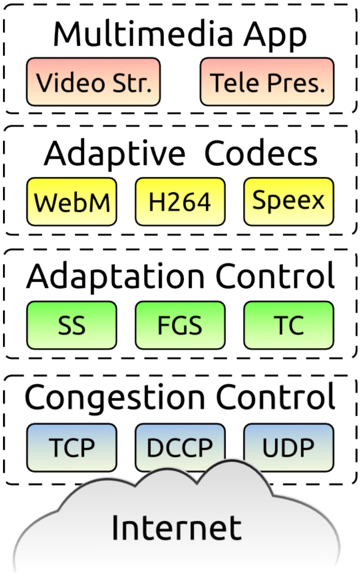
The key technological enabler is **Adaptivity**

Adaptive multimedia application requirements

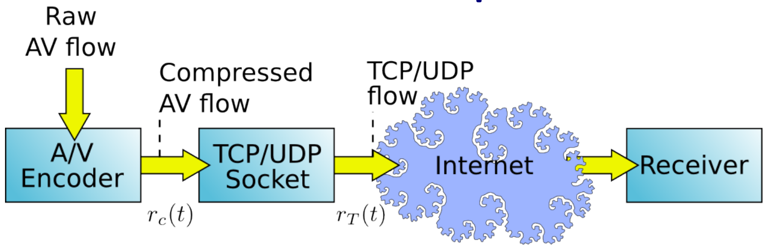
1. It should not harm the network, i.e. no congestion collapse $\Rightarrow$ Effective **congestion control** (TCP or application layer)
2. It should provide the maximum audio/video quality possible $\Rightarrow$ Flows must be made elastic by means of **adaptive codecs** together with an effective quality adaptation controller

Outline of the talk

The stack


Multimedia App
Video Str.
Tele Pres.
Adaptive Codecs
WebM
H264
Speex
Adaptation Control
SS
FGS
TC
Congestion Control
TCP
DCCP
UDP
Internet

The data flow path

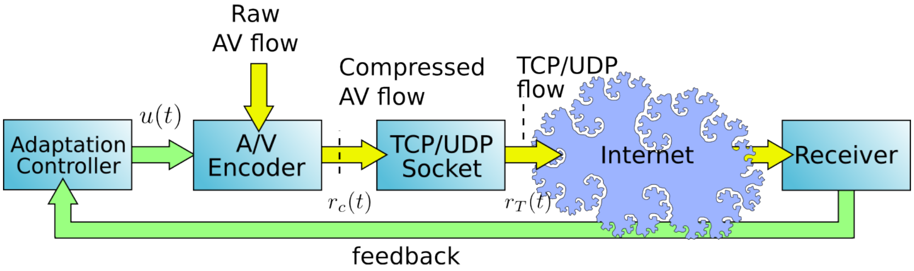


The multimedia flow path

1. The A/V Encoder compresses the raw A/V flow at a bitrate $r_c(t)$
2. The compressed A/V flow is sent via a TCP/UDP over the Internet at a rate $r_T(t)$
3. The receiver decodes the flow

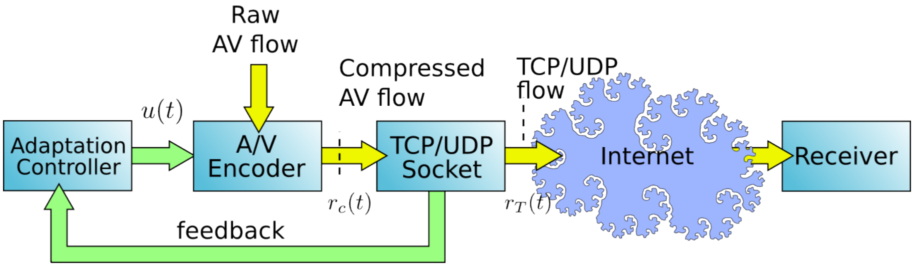
How to implement adaptivity?

End-to-end Architecture



- Adaptation controller regulates the encoding rate $r_c(t)$ based on receiver's feedback. Controller logic may be implemented at the receiver.
- Feedback could be playout buffer, lost frame rate, etc
- **Drawback**: the feedback loop is affected by a time delay due to the RTT of the connection

Local Feedback Architecture



- The feedback is local, control loop is not affected by delays
- Local feedback can be network available bandwidth (a-la TCP Westwood+), sender buffer size, etc
- **Drawback**: receiver is not in the loop, CPU overload cannot be taken into account

Leading applications: are they doing it right?

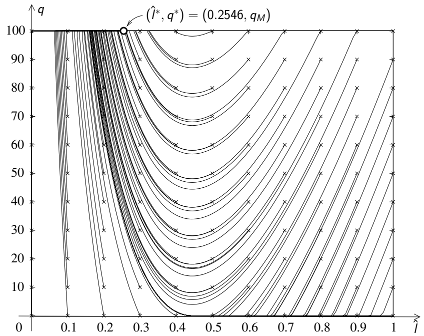
Video Streaming

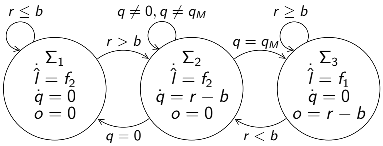
- **YouTube**: no adaptivity; each video is encoded into a number of different bitrates and resolutions, the user picks the version making a trade off between quality and his prediction about QoE
- **NetFlix**: implements adaptivity; each video is encoded into a number of different versions and the client automatically selects the most suitable version
- **Akamai HD Net**: heuristic-based stream-switching similar to NetFlix approach.

Video-conference

Skype: it implements adaptivity by throttling encoding bitrate to match available bandwidth [WWIC07, NOSSDAV08, CDC08, TAC10, COMNET11]

Skype VoIP Adaptation Algorithm





- De Cicco, Mascolo, "A Mathematical Model of the Skype VoIP Congestion Control Algorithm", *IEEE Trans. on Automatic Control*, vol. 55, n. 3, pp 790-795, Mar 2010
- De Cicco, Mascolo, Palmisano, "An Experimental Investigation of the Congestion Control Used by Skype VoIP", in Proc of *WWIC 2007*, May 2007

Introduction

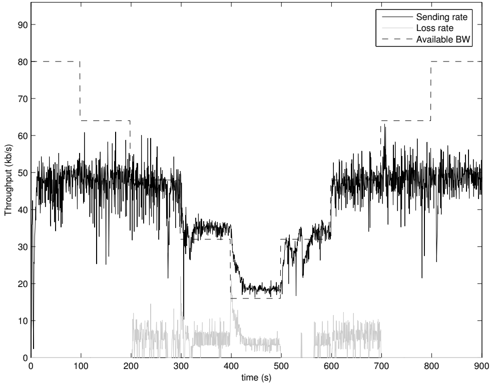
- TCP is not suitable for real-time multimedia communication (retransmissions)
- Several multimedia congestion control algorithms have been proposed: TFRC, DCCP, ARC, RAP
- Commercial applications still use proprietary congestion control algorithms (if any) over the UDP
- Skype is the leading VoIP application, generating a very large amount of UDP flows

Goal

To identify Skype VoIP adaptation algorithm mathematical model and to analyze its most relevant properties, i.e. to what extent it is able to adapt to time variable available bandwidth

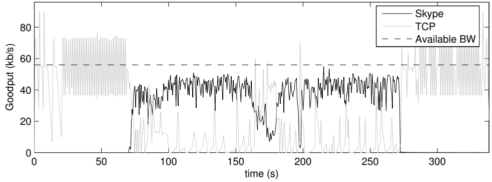
State of the art

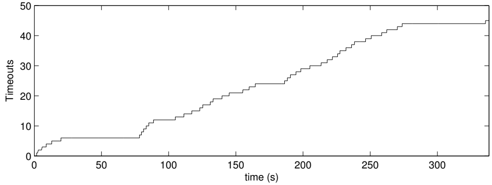
- Skype flows adapt its rate to bandwidth variations to some extent (losses occur) [WWIC07, NOSSDAV08]



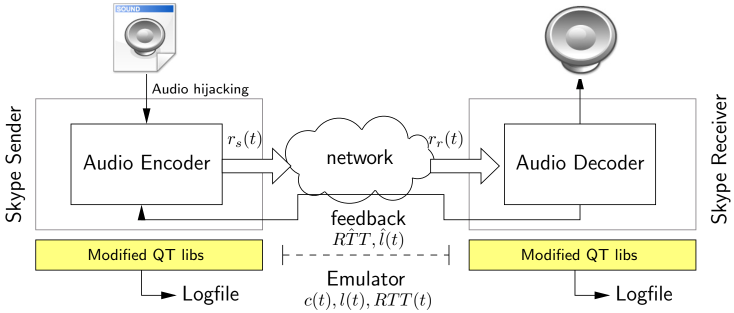
State of the art

- Skype flows adapt its rate to bandwidth variations to some extent (losses occur) [WWIC07, NOSSDAV08]
- The Skype VoIP adaptation exhibits remarkable unfairness *wrt* TCP concurrent flows due to a very slow responsiveness to sudden bandwidth variations (transient lasts $\sim 40\,\text{s}$)



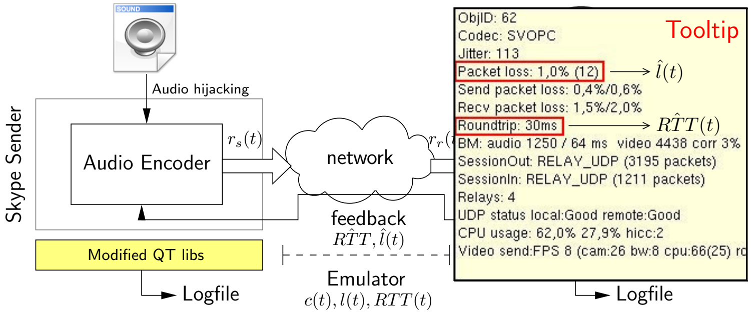


Testbed for experimental evaluation



- Three hosts: two Skype, one emulates a wide area network
- The emulator can impose available bandwidth, RTT, and loss ratio

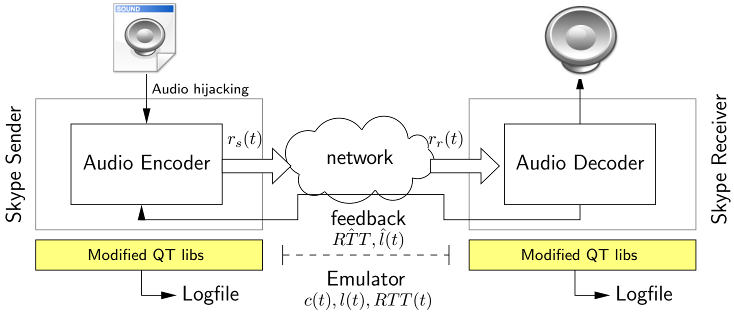
Testbed for experimental evaluation



ObjID: 62
Codec: SVOPC
Jitter: 113
Packet loss: 1,0% (12)
Send packet loss: 0,4%/0,6%
Recv packet loss: 1,5%/2,0%
Roundtrip: 30ms
BM: audio 1250 / 64 ms video 4438 corr 3%
SessionOut: RELAY_UDP (3195 packets)
SessionIn: RELAY_UDP (1211 packets)
Relays: 4
UDP status local:Good remote:Good
CPU usage: 62,0% 27,9% hicc:2
Video send:FPS 8 (cam:26 bw:8 cpu:66(25) rc

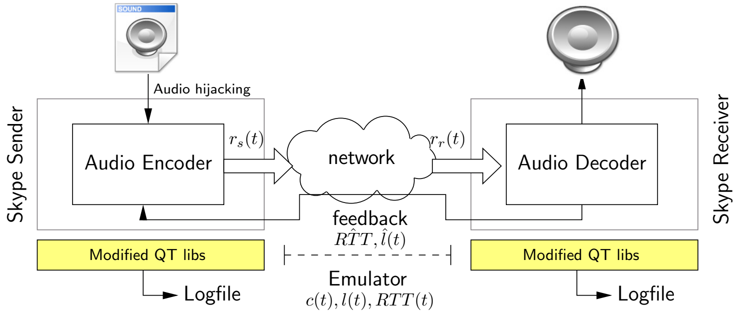
- Three hosts: two Skype, one emulates a wide area network
- The emulator can impose available bandwidth, RTT, and loss ratio
- We modified QT libraries to log the Skype tooltips (measured loss ratio $\hat{l}(t)$, $\hat{RTT}(t)$, etc)

Testbed for experimental evaluation



- Three hosts: two Skype, one emulates a wide area network
- The emulator can impose available bandwidth, RTT, and loss ratio
- We modified QT libraries to log the Skype tooltips (measured loss ratio $\hat{l}(t)$, $\hat{RTT}(t)$, etc)
- We expect *loss ratio* $\hat{l}(t)$ and *round trip time* $\hat{RTT}(t)$ measured by Skype are candidate inputs of the controller

Testbed for experimental evaluation



- Three hosts: two Skype, one emulates a wide area network
- The emulator can impose available bandwidth, RTT, and loss ratio
- We modified QT libraries to log the Skype tooltips (measured loss ratio $\hat{l}(t)$, $\hat{RTT}(t)$, etc)
- We expect *loss ratio* $\hat{l}(t)$ and *round trip time* $\hat{RTT}(t)$ measured by Skype are candidate inputs of the controller
- We cannot inspect packets since Skype encrypts flows

Experiments

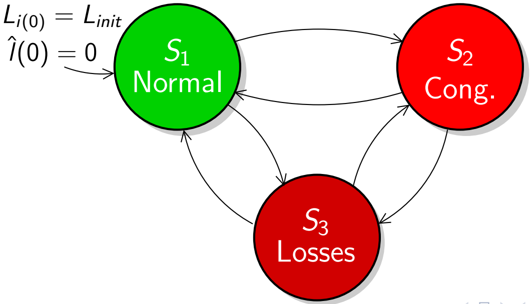
To identify the mathematical model of the rate $r_s(t)$ generated by Skype VoIP flow we conducted several experiments using the testbed:

1. Variable RTT $RTT(t)$: **we found it has no effect on** $r_s(t)$
2. Variable loss ratio $l(t)$ (lossy link): Skype uses FEC to counteract losses when losses are not due to congestion
3. Variable bandwidth $b(t)$: we found that Skype uses a low-passed version of $l(t)$ as the main driver for $r_s(t)$

Experiments

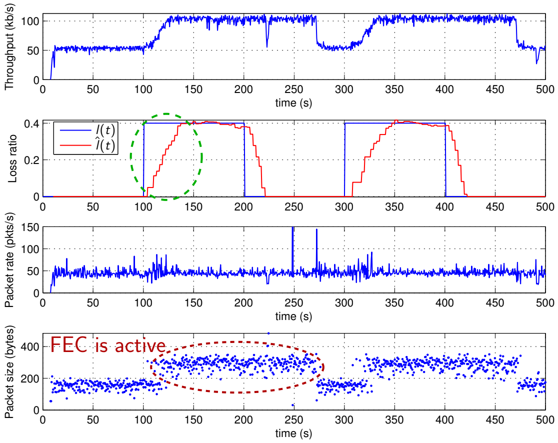
To identify the mathematical model of the rate $r_s(t)$ generated by Skype VoIP flow we conducted several experiments using the testbed:

1. Variable RTT $RTT(t)$: **we found it has no effect on** $r_s(t)$
2. Variable loss ratio $l(t)$ (lossy link): Skype uses FEC to counteract losses when losses are not due to congestion
3. Variable bandwidth $b(t)$: we found that Skype uses a low-passed version of $l(t)$ as the main driver for $r_s(t)$



Skype Audio over a variable loss rate link

Loss ratio $l(t)$: square wave, period 200 s, max value 0.4, min value 0



Results

1. When $l(t)$ increases, the sending rate increases⇒ *not a loss based algorithm*
2. The measured loss ratio $\hat{l}(t)$ (red) is a *low pass filtered* version of $l(t)$ (blue)
3. packet size increases when $l(t)$ increases ⇒ FEC to counteract persistent packet losses (state S_3 - Losses)

FEC action $f(t)$ and $\hat{l}(t)$

FEC action model $f(t)$

- Skype adds at most one redundant frame for each new sent frame
- when FEC action is maximum rate doubles
- $f(t) \in [0, 1]$ fraction of redundant frames: $f(t) = 0$ FEC is off, $f(t) = 1$ FEC is max

FEC action $f(t)$ and $\hat{l}(t)$

FEC action model $f(t)$

- Skype adds at most one redundant frame for each new sent frame
- when FEC action is maximum rate doubles
- $f(t) \in [0, 1]$ fraction of redundant frames: $f(t) = 0$ FEC is off, $f(t) = 1$ FEC is max

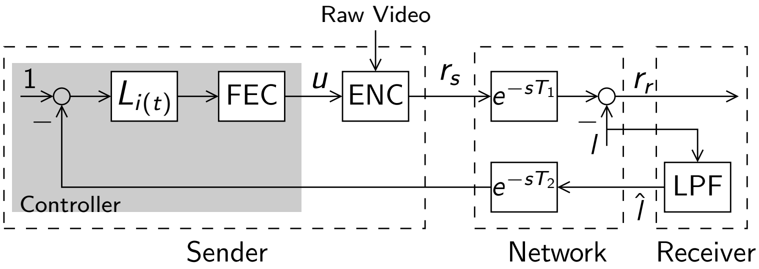
Measured loss ratio $\hat{l}(t)$

- Loss ratio: $l(t) = o(t)/r(t)$, where $o(t)$ is the overflow rate, $r(t)$ is the sending rate
- Skype measured loss rate: $l(t)$ filtered by a first order filter with time constant τ ($\tau \cong 11\,\mathrm{s}$)

$$\dot{\hat{l}}(t) = -\frac{1}{\tau}\hat{l}(t) + \frac{1}{\tau}l(t)$$

Proposed Model

Hypothesis: the encoder is multi-rate, it can choose among N levels of encoding $\mathscr{L} = \{L_1, \ldots, L_N\}$

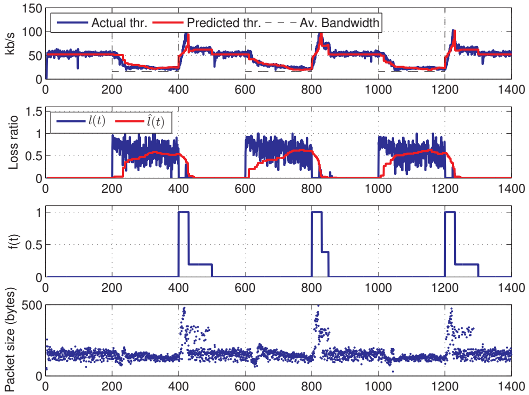


Proposed model: controller+encoder

$$r_s(t) = (1 - \hat{l}(t)) \underbrace{(1 + f(t))}_{\text{FEC action}} L_{i(t)}$$

$L_{i(t)}$ is the encoding level selected at time t

Validation of the proposed model



The long transient time exhibited by the CC algorithm is due to the filtering of the loss ratio

- The sending rate predicted by the model follows the rate produced by Skype
- $f(t) = 0$ during congestion (after a bandwidth drop occurs)
- FEC is activated after the available bandwidth increases to its maximum value (packet size doubles) to counteract packet losses in the "probing phase"

A Skype flow accessing a bottleneck

Setting: available bandwidth $b(t)$, drop-tail queue length q_M, fw path delay T_1, bw path delay T_2. State $[q(t), \hat{l}(t)]^T$

Queue length model $q(t)$

$$\dot{q} = \begin{cases} 0 & q = 0, r \leq b \text{ or } q = q_M, r \geq b \\ r - b & \text{otherwise} \end{cases} \quad (1)$$

where r is the queue input rate. The overflow rate is given by:

$$o = \begin{cases} r - b & q = q_M, r > b \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Measured loss ratio $\hat{l}(t)$ model

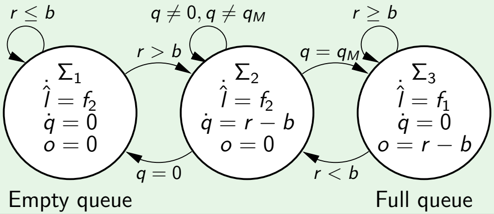
Recalling the model for the measured loss ratio is $\dot{\hat{l}} = -\frac{1}{\tau}\hat{l} + \frac{1}{\tau}\frac{o_{T_2}}{r_{T_2}}$ and plugging (2) it turns out:

$$\dot{\hat{l}} = \begin{cases} f_1 = \frac{1}{\tau} - \frac{\hat{l}}{\tau} - \frac{b_{T_2}}{\tau(1-\hat{l}_{T_2})(1+f_{T_2})L_{T_2}} & q = q_M, \;\; r > b \\ f_2 = -\frac{1}{\tau}\hat{l} & \text{otherwise} \end{cases} \quad (3)$$

Skype accessing a bottleneck: hybrid model

It is simple to show that the following hybrid automaton models the considered system (equations (1) and (3)):

Hybrid Automaton $\mathcal{H}$

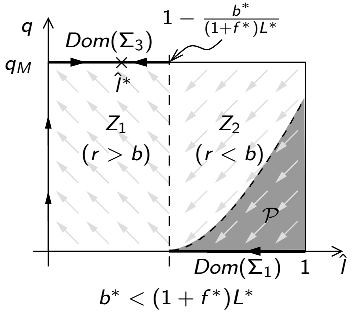


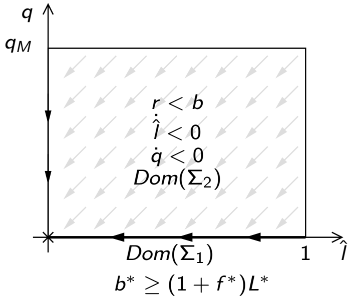
Characterizing the equilibria of $\mathcal{H}$

- Characterization of the equilibrium states of $\mathcal{H}$
- Steady state characterization of a Skype Audio flow accessing a bottleneck with capacity b
- The time delay is neglected since $\tau \gg RTT$

Qualitative Phase Portrait of $\mathcal{H}$ [TAC10]

Two different phase portraits depending on equilibrium inputs b^*, f^*, L^*



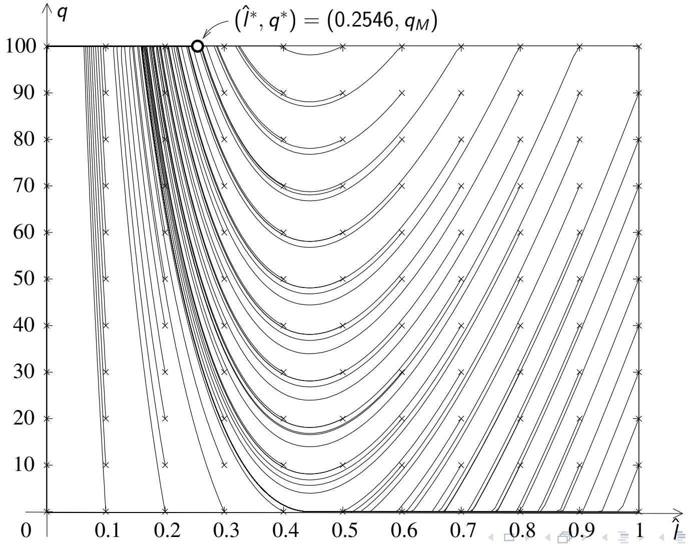


- Unique equilibrium:
  $\hat{l}^* = 1 - \sqrt{\frac{b^*}{L^*(1+f^*)}}$; $q^* = q_M$
- $\forall(\hat{l}(t_0), q(t_0))$ only reachable state:
  Σ_3 (full queue) $\Rightarrow$
  Persistent congestion

- Unique equilibrium:
  $\hat{l}^* = 0$; $q^* = 0$
- $\forall(\hat{l}(t_0), q(t_0))$ only reachable state
  Σ_1 (empty queue) $\Rightarrow$
  No congestion

Phase Portrait $b^* < (1 + f^*)L^*$

Example $b^* = 30\,\mathrm{kb/s}$, $f^* = 0$, $L^* = 56\mathrm{kb/s} \Rightarrow l^* = 0.2546$, $q^* = q_M$



Characterization of the steady state

Proposition

Let us consider the equilibrium inputs b^*, L^* and f^* the automaton $\mathcal{H}$ is characterized by the following equilibria that are **globally asymptotically stable***:*

$$\hat{l}^* = 1 - \sqrt{\frac{b^*}{L^*(1+f^*)}}\,;\; q^* = q_M \quad \text{iff} \quad b^* < (1+f^*)L^* \tag{4}$$

$$\hat{l}^* = 0\,;\; q^* = 0 \quad \text{iff} \quad b^* \geq (1+f^*)L^* \tag{5}$$

Corollary

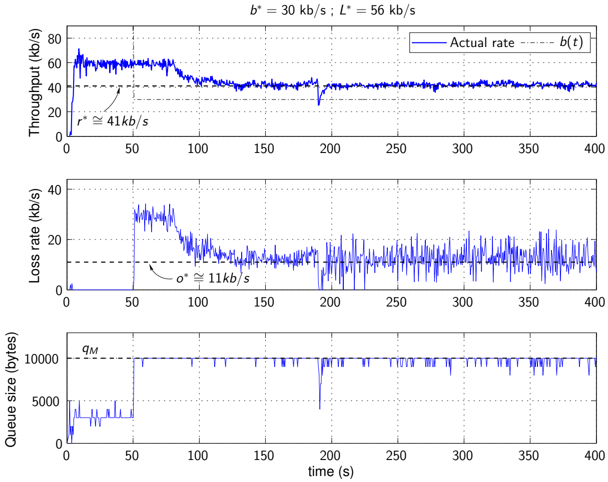
The controller employed by Skype is not able to avoid congestion unless $b^* > L^*(1+f^*)$.

Proof We are under the hypothesis of Prop 1 $\Rightarrow$ (4) is asymptotically stable. At steady state it results $o^* = (1-\hat{l}^*)L^*(1+f^*) - b^*$, thus substituting (4) in the equation just written we have:

$$o^* = \sqrt{b^* L^*(1+f^*)} - b^* > 0$$

Proposition validation

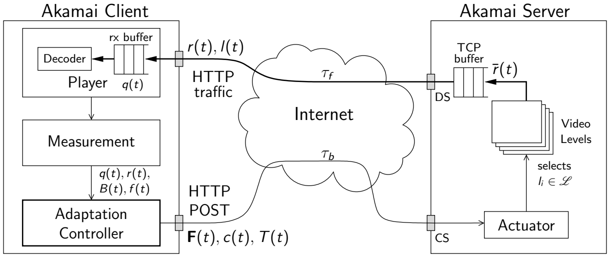
Available bandwidth drop at $t = 50$, $b^* = 30\,\text{kb/s}$, $f^* = 0$, $L^* = 56\text{kb/s} \Rightarrow r^* = 41\text{kb/s}$, $o^* = 11$ kb/s



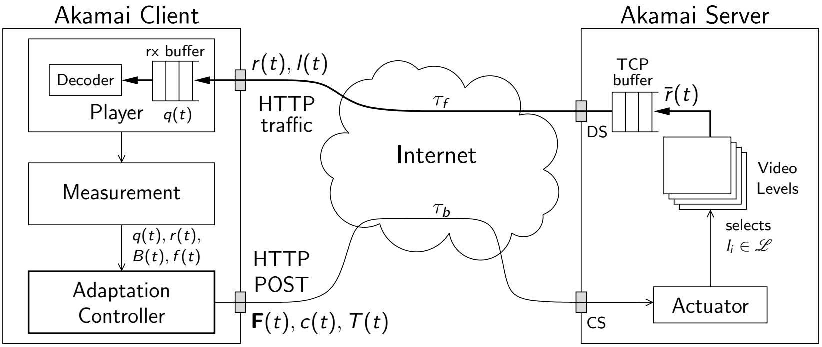
Conclusions

The steady state values for $r(t)$ and $o(t)$ recover the ones predicted by the model.
Persistent overflow rate is present: Skype VoIP is not able to avoid congestion.

Akamai Adaptive Streaming Algorithm



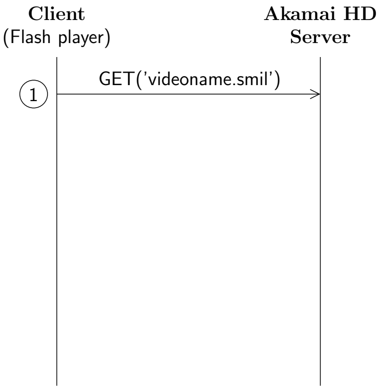
Akamai stream-switching at a glance



- Five video levels $\mathscr{L} = \{l_0, \ldots, l_4\}$ from 300 kbps (320x180) up to 3500 kbps (1280x720)
- Adaptation logic is client side
- Control loop is distributed: time-delay (RTT) affects the loop
- Adaptation logic is made of two coupled modules: 1) a **buffer level controller** and 2) a **stream-switching heuristics**

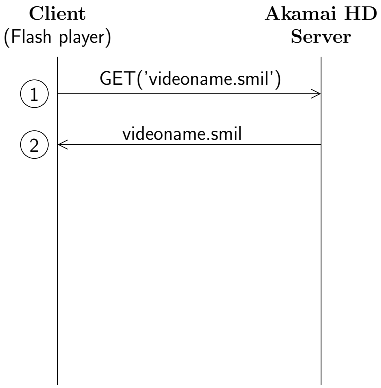
The client-server protocol

1. Client clicks on video, a HTTP GET request is sent to download a SMIL file



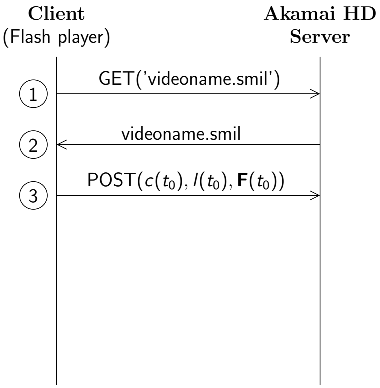
The client-server protocol

1. Client clicks on video, a HTTP GET request is sent to download a SMIL file
2. The SMIL file is sent to the client. The client parses the file which contains information on video levels with encoding bitrates



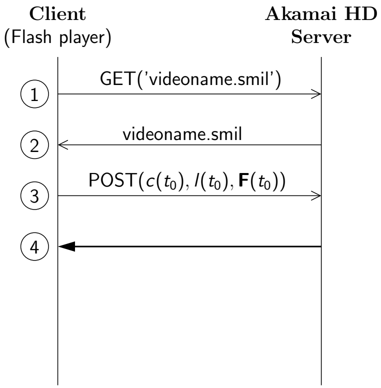
The client-server protocol

1. Client clicks on video, a HTTP GET request is sent to download a SMIL file
2. The SMIL file is sent to the client. The client parses the file which contains information on video levels with encoding bitrates
3. At time t_0 the adaptation algorithm starts. HTTP POST requests are sent specifying a command $c(t_0)$, the video level $l(t_0)$ and several feedback variables $\mathbf{F}(t_0)$



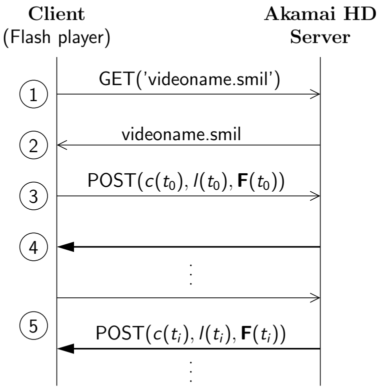
The client-server protocol

1. Client clicks on video, a HTTP GET request is sent to download a SMIL file
2. The SMIL file is sent to the client. The client parses the file which contains information on video levels with encoding bitrates
3. At time t_0 the adaptation algorithm starts. HTTP POST requests are sent specifying a command $c(t_0)$, the video level $l(t_0)$ and several feedback variables $\mathbf{F}(t_0)$
4. The server sends to the client the video level requested at time t_0



The client-server protocol

1. Client clicks on video, a HTTP GET request is sent to download a SMIL file
2. The SMIL file is sent to the client. The client parses the file which contains information on video levels with encoding bitrates
3. At time t_0 the adaptation algorithm starts. HTTP POST requests are sent specifying a command $c(t_0)$, the video level $l(t_0)$ and several feedback variables $\mathbf{F}(t_0)$
4. The server sends to the client the video level requested at time t_0
5. The algorithm repeats at time t_i


Client
(Flash player)
Akamai HD Server
1
GET('videoname.smil')
2
videoname.smil
3
POST($c(t_0), l(t_0), \mathbf{F}(t_0)$)
4
5
POST($c(t_i), l(t_i), \mathbf{F}(t_i)$)

The commands (cmd)

POST /control/Fname?cmd=c_i,$a_1, \ldots, a_n$&v=1.0&r=ABCDE&g=token&lvl1=$F_1, \ldots, F_{12}$

POST arguments

1. cmd: command c_i to be issued on the server
2. lvl1: client feedback variables $F_1, \ldots, F_{12}$ sent to the server

Commands c_i

1. throttle: issued periodically on average each 2s to adjusts the receiver buffer using a feedback control loop
2. rtt-test: issued periodically, on average each 11s, triggers greedy send mode (lasts 5 seconds) to estimate available bandwidth and RTT under congestion
3. SWITCH_UP: asks the server to switch the video level up
4. BUFFER_FAIL: asks the server to switch the video level down

The feedbacks (lvl1)

1. Receiver buffer size $q(t)$ [s]
2. Receiver buffer set point $q_T(t)$ [s]
3. Decoded frame rate $f(t)$ [fps]
4. Estimated bandwidth $B(t)$ [kbps]
5. Received goodput $r(t)$ [kbps]
6. Current video level $l(t)$ [kbps]
7. Round trip time $R(t)$ [s]

The buffer level controller (server)

Goal of the controller

To steer the client buffer length $q(t)$ to the set point $q_T(t)$

The throttle percentage

The throttle command specifies as the only argument the **throttle percentage** $T(t)$. We have identified the following control law:

$$T(t) = \max\left(100(1 + \frac{q_T(t) - q(t)}{q_T(t)}), 10\right)$$

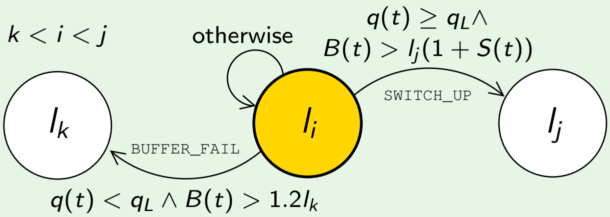
Akamai server throttles the TCP send buffer filling rate $\bar{r}(t)$ by using $T(t)$:

$$\bar{r}(t) = l(t)\frac{T(t)}{100}$$

When the error $e(t) = q_T(t) - q(t) > 0$ (buffer below the threshold) $T(t) > 100$ so that $\bar{r}(t) > l(t)$. This allows to send the video at a higher rate than $l(t)$ letting the receiver to fill the buffer.

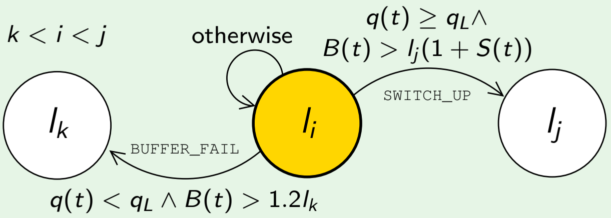
The stream-switching heuristic (client)

Identified stream-switching heuristics



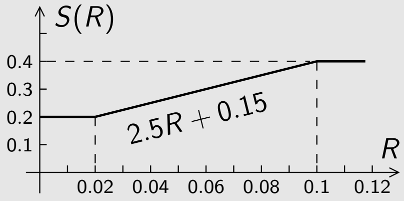
The stream-switching heuristic (client)

Identified stream-switching heuristics

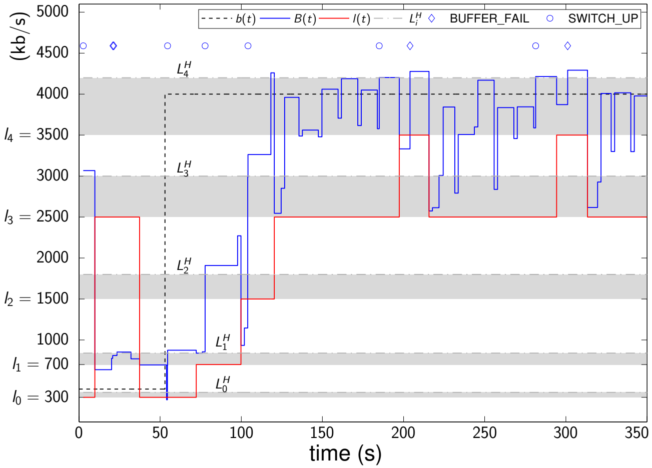


Safety factor S(t)

When `rtt-test` is issued, $T(t) = 500$ allowing the server to send in greedy mode and to probe for the available bandwidth and measure the RTT $R(t)$ under congestion

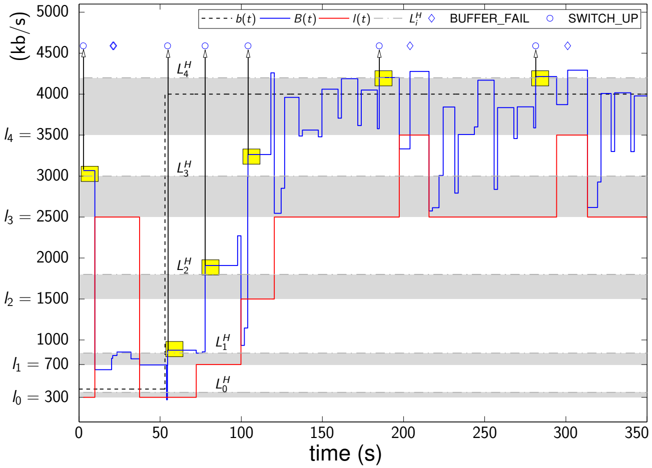


Step Response (1/2)



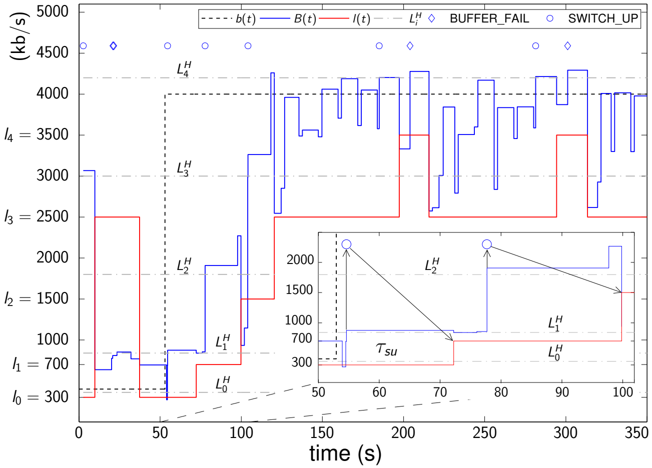
- Step-like available bandwidth from 0.5Mb/s to 4 Mb/s

Step Response (1/2)



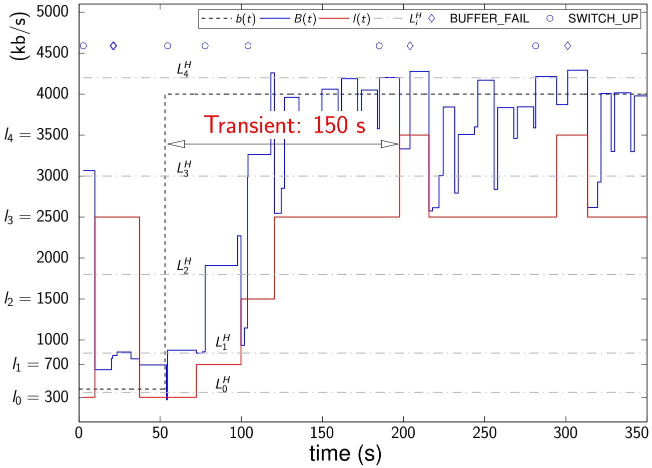
- Step-like available bandwidth from 0.5Mb/s to 4 Mb/s
- Each time $B(t) > l_i(1 + S(t))$ a `SWITCH_UP` is sent

Step Response (1/2)



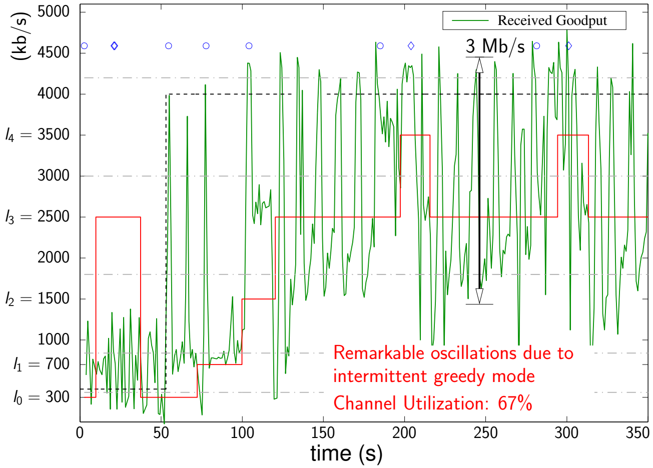
- Step-like available bandwidth from 0.5Mb/s to 4 Mb/s
- Each time $B(t) > l_i(1 + S(t))$ a `SWITCH_UP` is sent
- Video levels are switched up after a delay of around 15s

Step Response (1/2)



- Step-like available bandwidth from 0.5Mb/s to 4 Mb/s
- Each time $B(t) > l_i(1 + S(t))$ a `SWITCH_UP` is sent
- Video levels are switched up after a delay of around 15s
- Sluggish response due to conservativeness

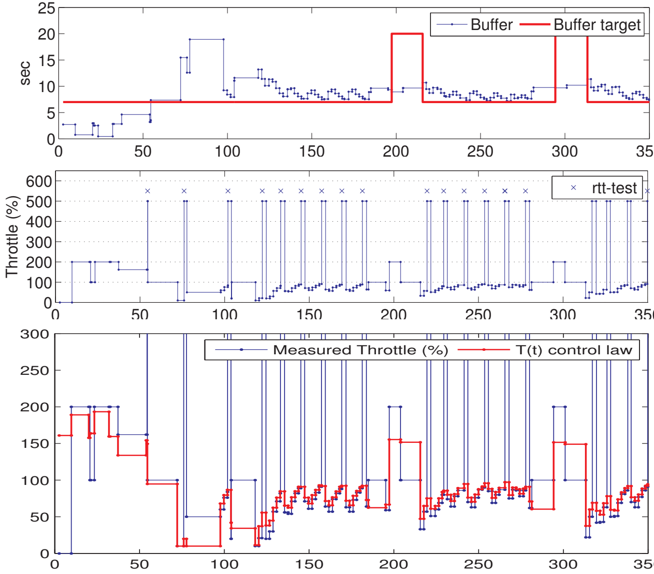
Step Response (1/2)



- Step-like available bandwidth from 0.5Mb/s to 4 Mb/s
- Each time $B(t) > l_i(1 + S(t))$ a `SWITCH_UP` is sent
- Video levels are switched up after a delay of around 15s
- Sluggish response due to conservativeness

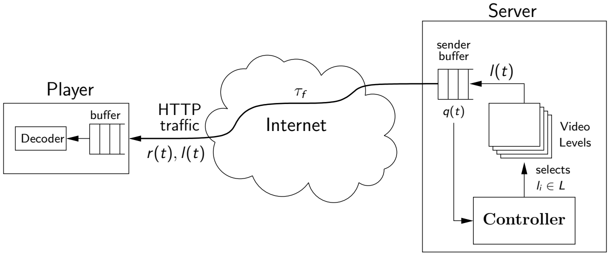
- Large oscillations due to the greedy/non-greedy phases triggered by rtt-test
- Poor channel utilization

Step response (2/2)

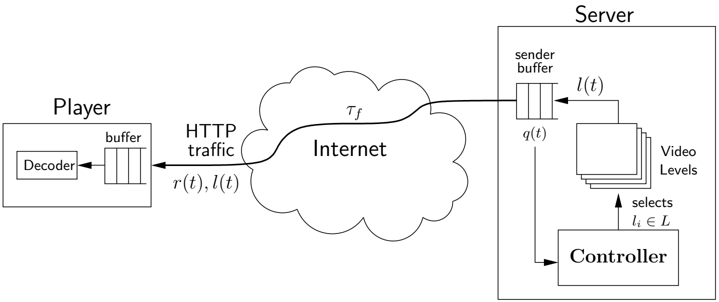


- Buffer tracks the target with a steady state error
- Each time a rtt-test command is sent throttle is set to 500
- The identified law for the throttle $T(t)$ fits well the measured throttle signal

Feedback-based Adaptive Streaming Algorithm



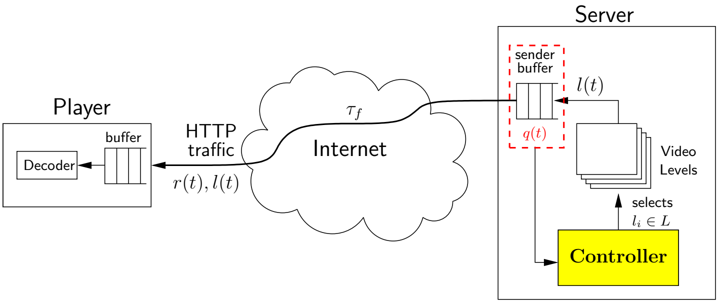
The proposed quality adaptation controller (QAC)



Design choices

- Controller is **server-side**, no feedback from the client required

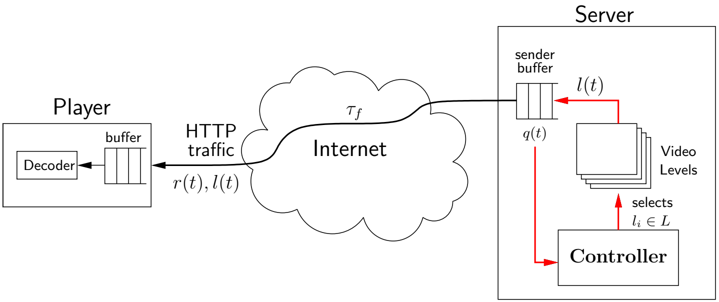
The proposed quality adaptation controller (QAC)



Design choices

- Controller is **server-side**, no feedback from the client required
- Control architecture: local feedback using the sender buffer

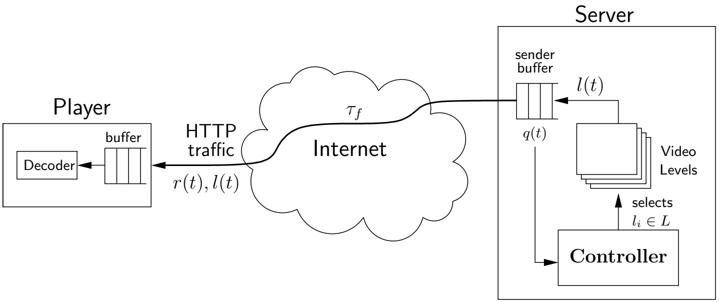
The proposed quality adaptation controller (QAC)



Design choices

- Controller is **server-side**, no feedback from the client required
- Control architecture: local feedback using the sender buffer
- The control loop is not distributed thus the system is **delay-free**

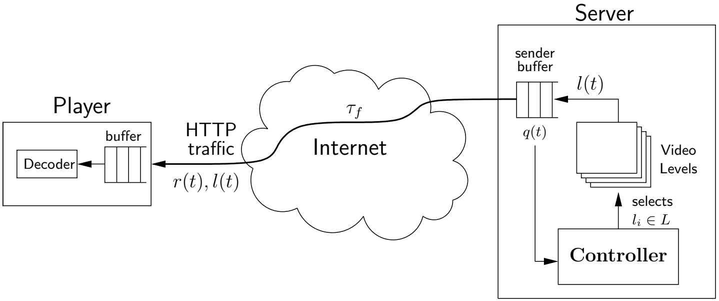
The proposed quality adaptation controller (QAC)



Design choices

- Controller is **server-side**, no feedback from the client required
- Control architecture: local feedback using the sender buffer
- The control loop is not distributed thus the system is **delay-free**
- The controller is designed using classic **feedback control**

The proposed quality adaptation controller (QAC)

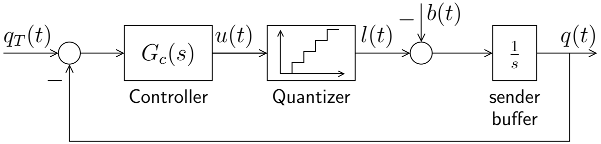


Design choices

- Controller is **server-side**, no feedback from the client required
- Control architecture: local feedback using the sender buffer
- The control loop is not distributed thus the system is **delay-free**
- The controller is designed using classic **feedback control**
- Any player can be employed

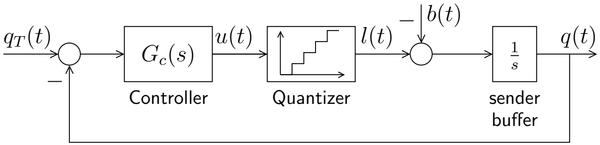
QAC - The control loop

- **Goal of the controller**: steer the sender buffer at a desired target $q_T(t) > 0$



QAC - The control loop

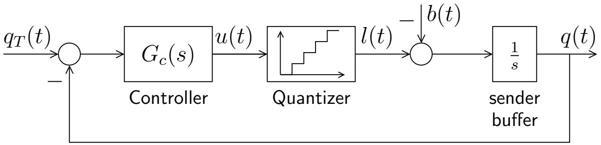
- **Goal of the controller**: steer the sender buffer at a desired target $q_T(t) > 0$



- The sender buffer is filled at rate $l(t)$ and drained by the available bandwidth $b(t)$

QAC - The control loop

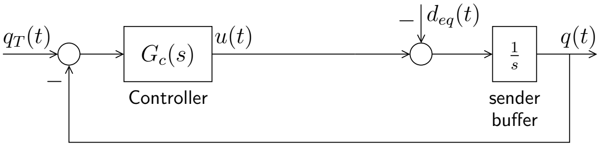
- **Goal of the controller**: steer the sender buffer at a desired target $q_T(t) > 0$



- The sender buffer is filled at rate $l(t)$ and drained by the available bandwidth $b(t)$
- $l(t)$ belongs to a discrete set $\mathscr{L} = \{l_0, \ldots, l_4\}$, thus control signal $u(t)$ is quantized

QAC - The control loop

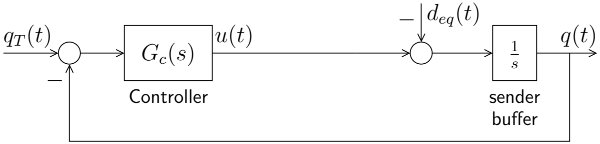
- **Goal of the controller**: steer the sender buffer at a desired target $q_T(t) > 0$



- The sender buffer is filled at rate $l(t)$ and drained by the available bandwidth $b(t)$
- $l(t)$ belongs to a discrete set $\mathscr{L} = \{l_0, \ldots, l_4\}$, thus control signal $u(t)$ is quantized
- To make the control loop linear, we introduce the equivalent (bounded) disturbance $d_{eq}(t) = d_q(t) + b(t)$ where $d_q(t)$ is the mismatch between $u(t)$ and $l(t)$

QAC - The control loop

- **Goal of the controller**: steer the sender buffer at a desired target $q_T(t) > 0$



- The sender buffer is filled at rate $l(t)$ and drained by the available bandwidth $b(t)$
- $l(t)$ belongs to a discrete set $\mathscr{L} = \{l_0, \ldots, l_4\}$, thus control signal $u(t)$ is quantized
- To make the control loop linear, we introduce the equivalent (bounded) disturbance $d_{eq}(t) = d_q(t) + b(t)$ where $d_q(t)$ is the mismatch between $u(t)$ and $l(t)$
- To get zero steady state error and reject step disturbances we employ a PI controller: $G_c(s) = K_p + K_i/s$

Controller tuning and implementation

Closed loop transfer function

$$G_0(s) = \frac{K_p s + K_i}{s^2 + K_p s + K_i}$$

Controller tuning

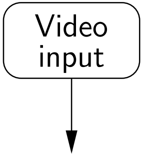
- **Settling time** $T_s = 30$ s (system bandwidth 0.06Hz in order to reduce video level switches)
- **damping factor** $\delta = 0.707$
- It turns out $K_p = 0.2667$, $K_i = 0.0356$

Implementation

Discretized control law with sampling time $\Delta T = 0.5$ s

$$u(t_k) = K_p e(t_k) + K_i \sum_{j=0}^{k} \Delta T e(t_j)$$

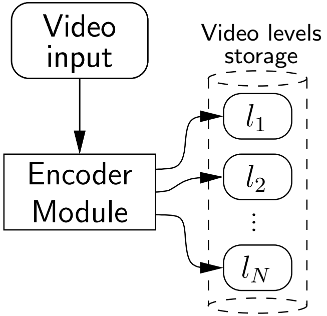
From theory to practice: QuavStreams



Implementation

①Raw video

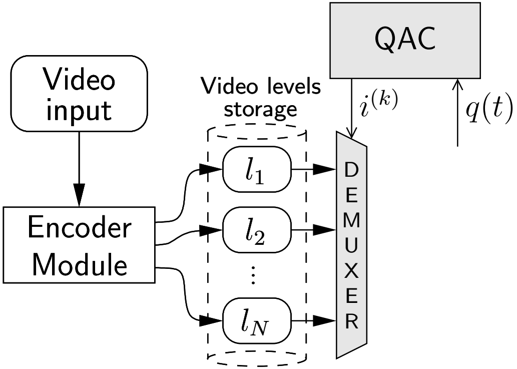
From theory to practice: QuavStreams



Implementation

①Raw video→② Multiple Encoding

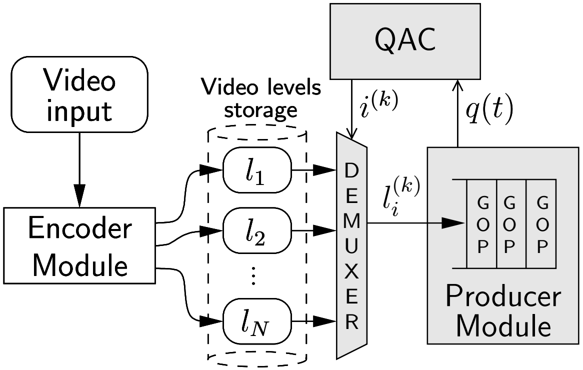
From theory to practice: QuavStreams



Implementation

①Raw video→② Multiple Encoding→③ QAC selects video level l_i

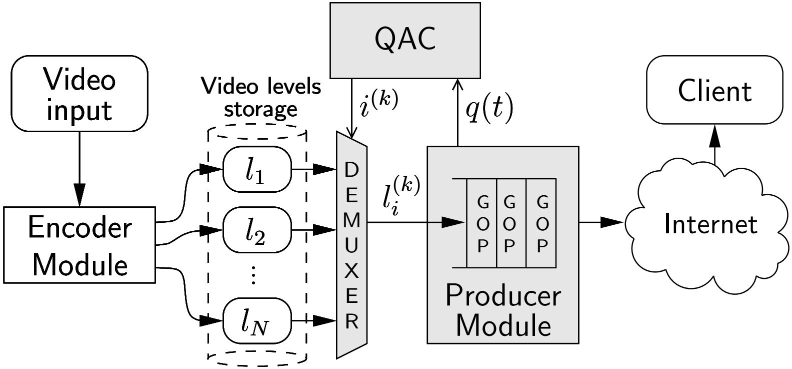
From theory to practice: QuavStreams



Implementation

①Raw video→② Multiple Encoding→③ QAC selects video level l_i→
④ i-th file version is stored in the producer's sender queue

From theory to practice: QuavStreams



Implementation

①Raw video→② Multiple Encoding→③ QAC selects video level l_i→
④ i-th file version is stored in the producer's sender queue→
⑤ Producer sends the video through HTTP to the client

Experimental comparison: Akamai vs QAC

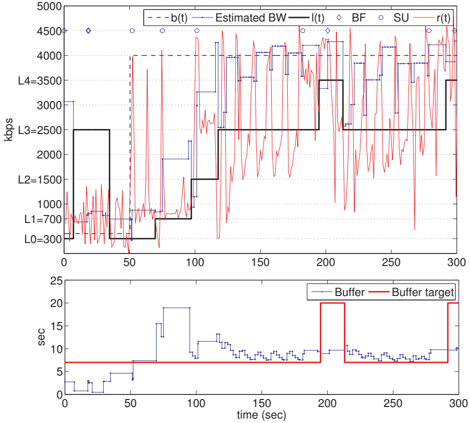
Controlled testbed scenario

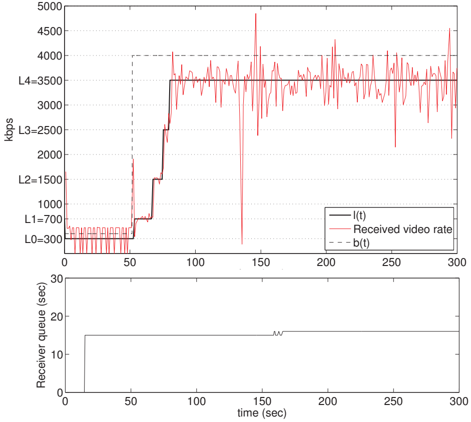
- Investigate systems dynamics
- Evaluate network utilization

High Speed Data Packet Access scenario

- How the algorithms work in a mobile scenario in terms of video continuity and video quality achieved
- Investigate fairness with concurrent TCP traffic

Step response





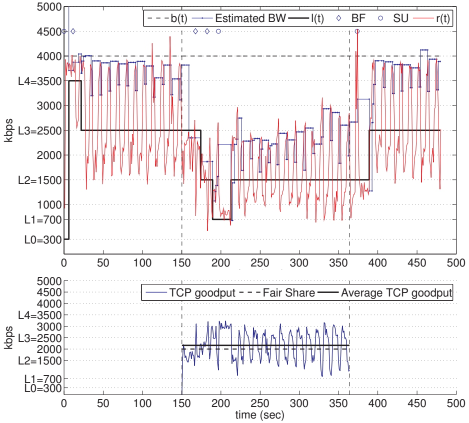
Akamai

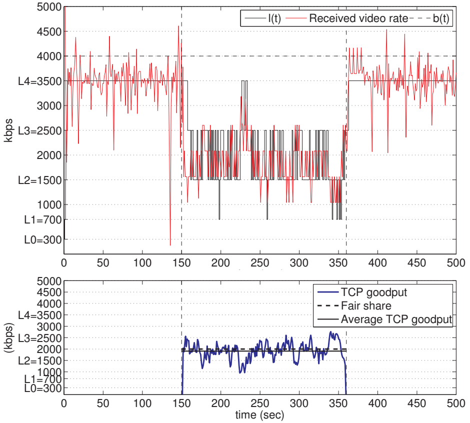
- Transient: 150s
- Ch. utilization: 67%
- Complex dynamics

QAC

- Transient: 30s (settling time)
- Ch. utilization: 93%
- Predictable dynamics

Concurrent TCP flow: Akamai vs QAC





Akamai

- Remarkable TCP oscillations
- Ch. utilization: 76%
- Takes 62 s to switch down

QAC

- Typical TCP burstiness
- Ch. utilization: 99%
- Switches between l_2 and l_3

Experimental campaign over HSDPA

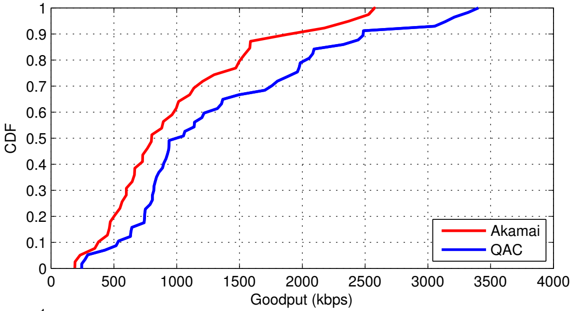
Comparing Akamai and QAC over a HSDPA network

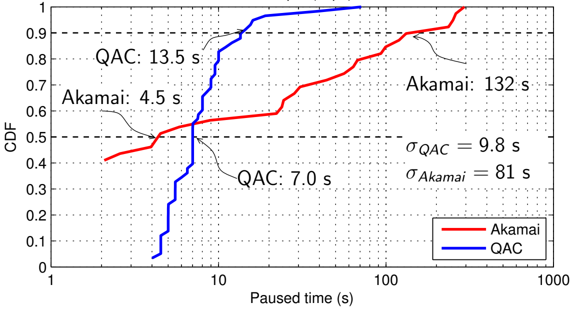
- For fair comparison QAC use the same video with the same video levels set of Akamai
- Two considered scenarios: 1) One video flow, 2) one video flow vs one TCP flow
- More than 200 experiments in a two months time-span

Metrics considered for each experiment

- Average goodput $\overline{g}_i$
- Paused time: total time the client is paused due to rebuffering

One video flow





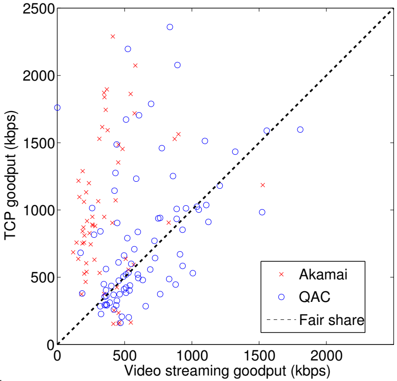
Goodput

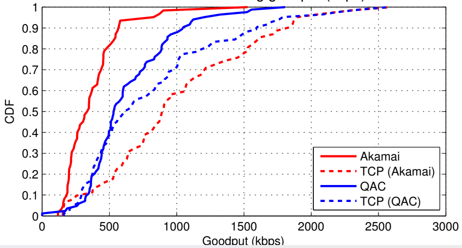
Akamai goodput is around 40% less wrt QAC due to its conservativeness (switch up/down safety bands $\max(S) = 0.4$)

Paused time

QAC 90-th percentile is < 15s, Akamai 90-th percentile is > 100s. Akamai performance is hard to be predicted (high standard deviation)

One video flow vs one TCP flow

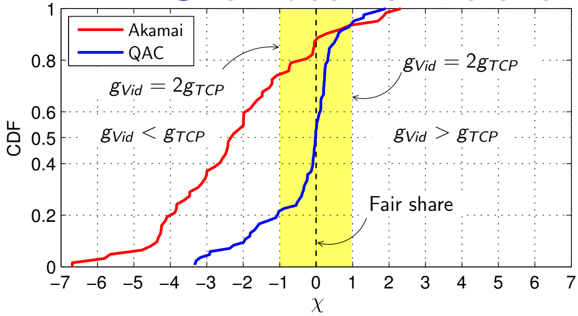




Goodput

Akamai does not get a fair share due to on-off greedy phases (TCP is greedy)
QAC obtains the fair share since it is always greedy ($q_T > 0$)

One video flow vs one TCP flow



Relative distance from fair share:

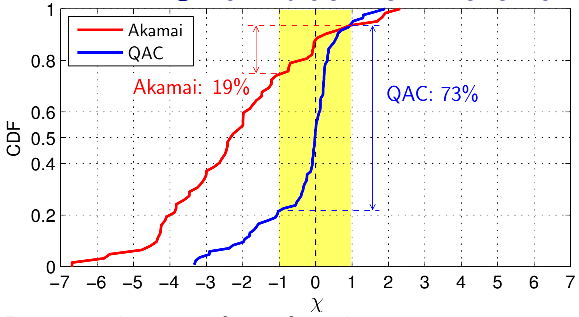
$$\chi(g_{vid}, g_{TCP}) = \begin{cases} \frac{g_{vid}}{g_{TCP}} - 1 & g_{vid} \geq g_{TCP} \\ -\frac{g_{TCP}}{g_{vid}} + 1 & g_{vid} < g_{TCP} \end{cases}$$

In yellow the "fair region" video or TCP gets at most twice the goodput of the concurrent flow ($\chi = \pm 1$)

Goodput

Akamai does not get a fair share due to on-off greedy phases (TCP is greedy)
QAC obtains the fair share since it is always greedy ($q_T > 0$)

One video flow vs one TCP flow



Relative distance from fair share:

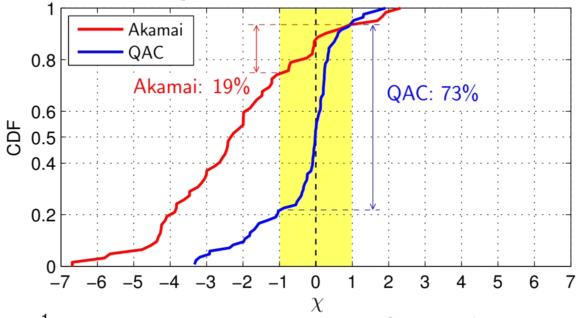
$$\chi(g_{vid}, g_{TCP}) = \begin{cases} \frac{g_{vid}}{g_{TCP}} - 1 & g_{vid} \geq g_{TCP} \\ -\frac{g_{TCP}}{g_{vid}} + 1 & g_{vid} < g_{TCP} \end{cases}$$

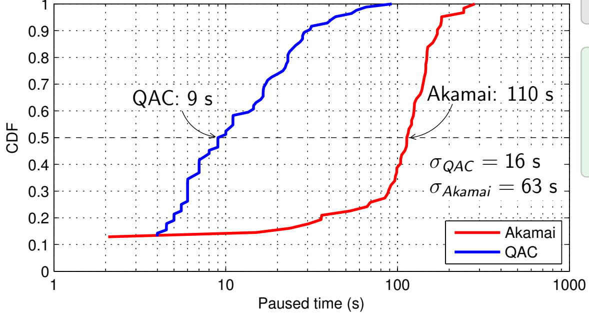
In yellow the "fair region" video or TCP gets at most twice the goodput of the concurrent flow ($\chi = \pm 1$)

Goodput

Akamai does not get a fair share due to on-off greedy phases (TCP is greedy)
QAC obtains the fair share since it is always greedy ($q_T > 0$)

One video flow vs one TCP flow





Goodput

Akamai does not get a fair share due to on-off greedy phases (TCP is greedy)
QAC obtains the fair share since it is always greedy ($q_T > 0$)

Paused time

QAC median is $<$ 10s, Akamai median is $>$ 100s

Conclusions

- The design of effective control algorithms to adapt multimedia sources to the available bandwidth is crucial to ensure good QoE both in terms of video quality and video continuity

Conclusions

- The design of effective control algorithms to adapt multimedia sources to the available bandwidth is crucial to ensure good QoE both in terms of video quality and video continuity
- Even though automatic control techniques have been successfully applied to the networking field since almost two decades, these tools are hardly used in the industry and complex heuristics are used

Conclusions

- The design of effective control algorithms to adapt multimedia sources to the available bandwidth is crucial to ensure good QoE both in terms of video quality and video continuity
- Even though automatic control techniques have been successfully applied to the networking field since almost two decades, these tools are hardly used in the industry and complex heuristics are used
- Skype Audio adaptation law is not able to avoid congestion and Akamai Adaptive Video Streaming is based on heuristics that are not able to efficiently adapt to bandwidth variations

Conclusions

- The design of effective control algorithms to adapt multimedia sources to the available bandwidth is crucial to ensure good QoE both in terms of video quality and video continuity
- Even though automatic control techniques have been successfully applied to the networking field since almost two decades, these tools are hardly used in the industry and complex heuristics are used
- Skype Audio adaptation law is not able to avoid congestion and Akamai Adaptive Video Streaming is based on heuristics that are not able to efficiently adapt to bandwidth variations
- We have proposed a simple adaptation controller (QAC) based on classic control theory which outperforms Akamai Video Streams both in terms of dynamics and QoE

Ongoing and future research

CPU overload control for mobile devices

- design a cascade control architecture to control both bandwidth variations and CPU overload
- propose standardization in the recent WebRTC (Web real time communication) IETF standardization effort

Cloud-based adaptive live multimedia distribution network

- develop a model of the overall system (considering all nodes of the distribution tree)
- design of optimal cloud resource management algorithm minimizing costs and maximizing QoE using optimal control framework

Bibliography

Skype

- De Cicco, Mascolo, Palmisano, "Skype Video Congestion Control: an Experimental Investigation", *Computer Networks, Elsevier*, vol. 55, issue 3, pp. 558-571, Feb. 2011
- De Cicco, Mascolo, "A Mathematical Model of the Skype VoIP Congestion Control Algorithm", *IEEE Trans. on Automatic Control*, vol. 55, n. 3, pp 790-795, Mar 2010
- De Cicco, Mascolo, Palmisano, "Skype Video Responsiveness to Bandwidth Variations", in Proc. of *ACM NOSSDAV '08*, Germany, May, 2008
- De Cicco, Mascolo, Palmisano, "An Experimental Investigation of the Congestion Control Used by Skype VoIP", in Proc of *WWIC 2007*, May 2007

Adaptive Video Streaming

- De Cicco, Mascolo, Abdallah, "An Experimental Evaluation of Akamai Adaptive Video Streaming over HSDPA networks", *in Proc. of MCSC 2011* , Denver, USA, Sept. 2011
- De Cicco, Mascolo, Palmisano, "Feedback Control for Adaptive Live Video Streaming", *in Proc. of ACM Multimedia Systems Conference 2011*, San Jose, California, USA, Feb 2011
- De Cicco, Mascolo, "An Experimental Investigation of the Akamai Adaptive Video Streaming" , *in Proc. of USAB 2010*, pp. 447-464, Klagenfurt, Austria, Nov. 2010