# Feedback Control for Adaptive Live Video Streaming

L. De Cicco, S. Mascolo and V. Palmisano

**Dipartimento di Elettronica ed Elettrotecnica**

**Politecnico di Bari – Italy**

***ACM Multimedia Systems 2011 - February 23-25, 2011***
***San Jose, California***

# Motivation

- "The sum of all forms of video will exceed 91% of global consumer traffic by 2014" (source: *Cisco Visual Networking Index 2009-2014*)
- Video Streaming feeds this bandwidth demanding growth (with full HD videos bitrate up to several Mbps)
- Adding adaptivity to classic progressive download streaming (*a là* YouTube) is a

key challenge: to provide live video in real-time

# *Outline of the work*

1. Video rate adaptation techniques
2. The Akamai HD Video Streaming stream-switching controller
3. The proposed Quality Adaptation Controller
4. Testbed
5. Experimental results
6. Conclusions

# Progressive Download Streaming
## *YouTube, Dailymotion*

- The video is a static file sent via HTTP over a greedy TCP connection
- A buffer at the client absorbs mismatches between available bandwidth and encoding bitrate
- Buffer eventually gets empty and playback interruptions occur when the available bandwidth is less than encoding bitrate
- Easy deployment with standard HTTP servers, supports proxies and CDNs

# *Adaptive Streaming*

1. **Transcoding**: on-the-fly encoding at a desired bitrate to match the available bandwidth (high processing and deployment costs)

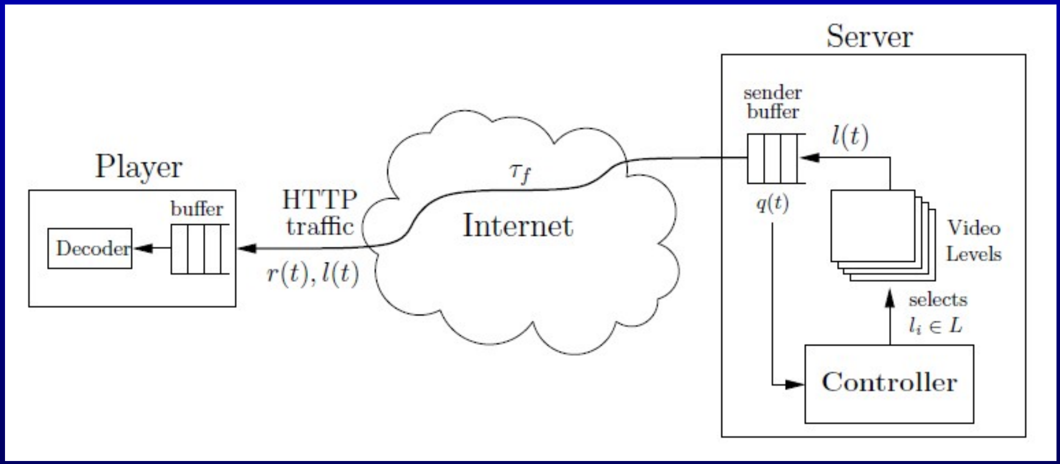
2. **Scalable codecs**: encoding once using temporal and spatial scalability. The encoded video is adapted without transcoding (low processing costs)

3. **Stream-switching**: encoding at several bitrates. The level that matches the available bandwidth is chosen (low processing costs, increased storage, simple to be deployed on CDNs).

4. Stream-Switching is gaining a wide acceptance in the industry: **Adobe Dynamic Streaming**, **HTTP Adaptive Live Streaming** (Apple), **Move Networks, IIS Smooth Streaming (Microsoft)**:

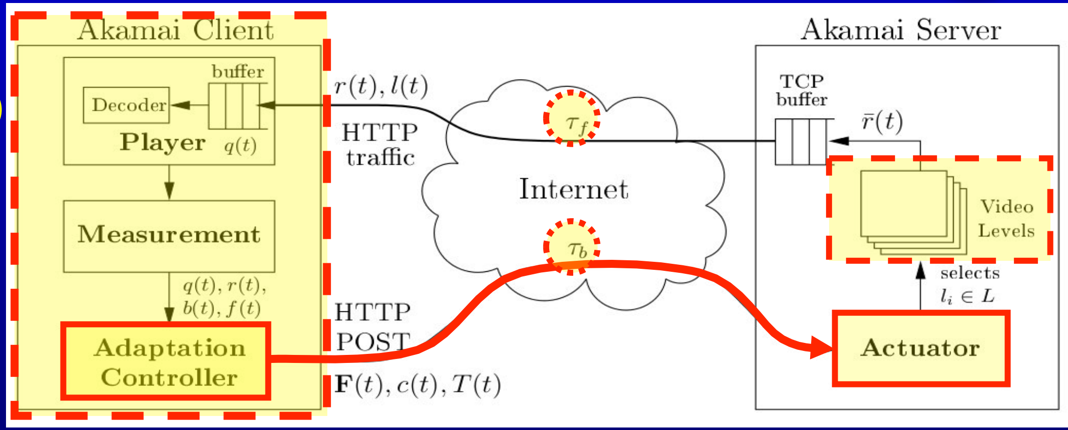
# The Architecture of the proposed Quality Adaptation Controller (QAC)

- The control logic is implemented at the **server**, no feedback from the client
- The control loop is **delay-free**
- Controller is designed using **feedback control**
- The goal of the control is to keep the sender buffer at a desired target
- Dynamical properties of the system can be **mathematically analyzed**, control parameters rigorously tuned
- Settling time, steady state errors, can be set



# Akamai stream-switching algoritmh at a glance

- Five video levels from 300 kbps (320x180) up to 3500 kbps (1280x720)
- Adaptation logic is client side
- Client sends POST commands to the server specifying several feedbacks




- **Adaptation logic** coupled modules:
  1. **Buffer level controller**: controls the client buffering time using a proportional controller
  2. **Stream-switching heuristic**: selects the video level based on measurement of variables such as the estimated bandwidth
- Control algorithm is distributed (the actuator is at the server) and affected by a **time-variant delay** equal to an RTT
- The overall system dynamics is difficult to be predicted and mathematically analyzed

# Akamai: the client-server protocol

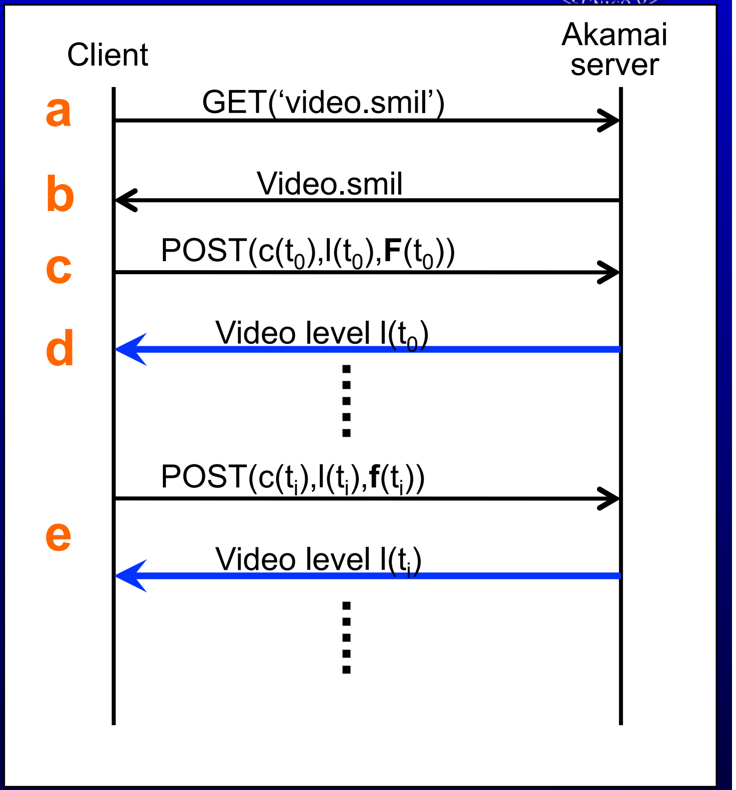
a) The client clicks on a video thumbnail and a HTTP GET message is sent requesting a SMIL file

b) The SMIL file is sent to the client. All video levels with encoding bitrates are exposed in such file

c) At time $t_0$ the adaptation algorithm starts. HTTP POST messages are sent specifying a command $c(t_0)$, the video level $l(t_0)$and several feedback variables $\mathbf{F}(t_0)$

d) The video level requested at time $t_0$ is sent to the client

e) The algorithm repeats at time $t_i$



# The commands (`cmd`)

The POST messages specify two arguments: **`cmd`** and **`lvl1`**

**`cmd`**: specifies **commands** to be issued on the server

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`)*

The **lvl1** argument provides the following feedback variables to the server:

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 Akamai buffer level controller

**Goal**: steer the client buffering time q(t) to a set point $q_T(t)$

**Identified control law:**

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

T(t) is used to throttle the rate X(t) at which Akamai fills the TCP buffer with the current video level l(t) as follows:

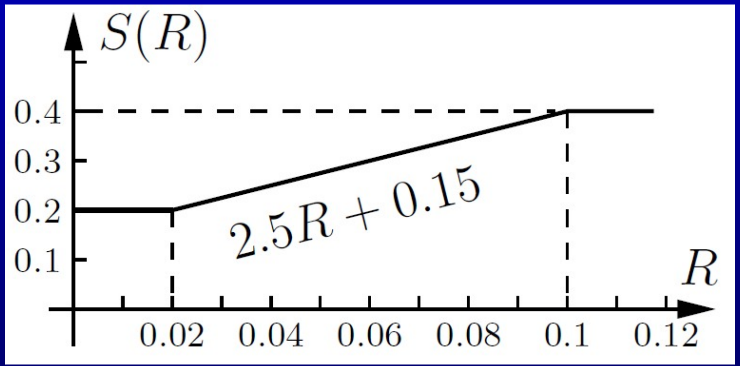
$$X(t) = l(t)\,T(t)/100$$

When the error $q_T(t)-q(t)>0$, $T(t)>100$ so that $X(t)>l(t)$. This allows to send the video at a rate higher than l(t) letting the receiver buffer to fill.

# The stream switching heuristics

When `rtt-test` commands are 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

We have identified a **safety factor** *S* computed as a function of R(t):



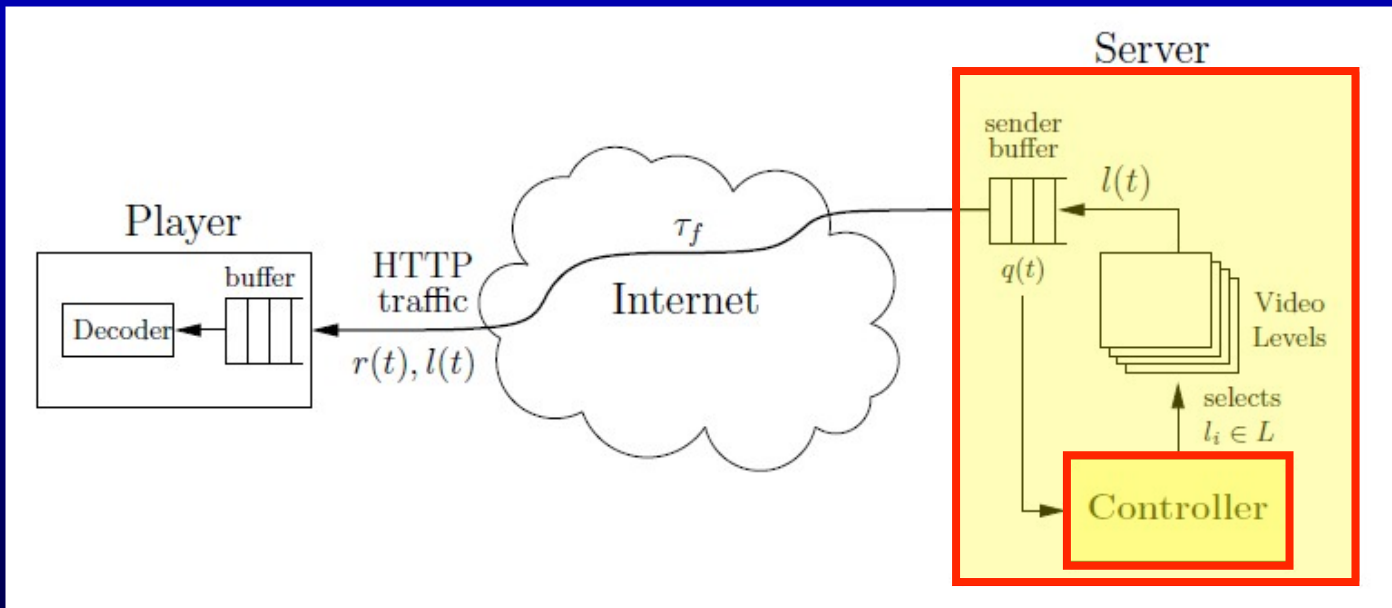

Two thresholds are maintained for each video level $l_i$:

$$L_i^H(t) = l_i\,(1+S(t)) \quad ; \quad L_i^L(t) = 1.2\, l_i$$

**`POST(SWITCH_UP(l_j))`**: when $B(t) > L_i^H(t)$, the highest video level $l_j$ satisfying $B(t) > L_j^H(t)$ is sent via POST and the command is actuated by the server after an average delay of $\tau_{su}$=14s

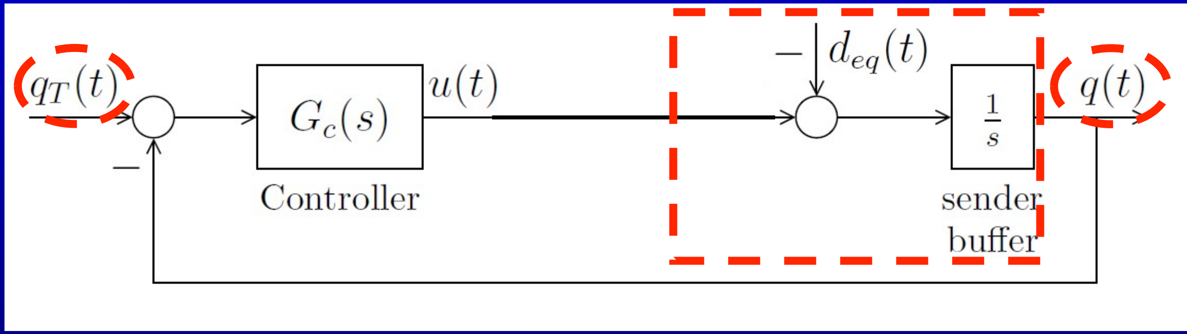
**`POST(BUFFER_FAIL(l_j))`**: when $q(t) < q_L(t)$ (*low threshold*), the highest video level $l_j$ satisfying $B(t) > L_j^L(t)$ is selected. The command is actuated after an average delay of $\tau_{sd}$=7s

# Feedback control for Quality Adaptation



# QAC – The control loop

- **Goal of the controller:** steer the sender buffer at a desired target $q_T>0$
- The queue is filled at rate l (t) and drained at available bandwidth b(t)




- l(t) belongs to a discrete set L, thus the control signal is quantized
- To make the control loop linear, we introduce the equivalent 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

$$G_c(s) = \frac{U(s)}{E(s)} = K_p + \frac{K_i}{s}$$

## Controller tuning and implementation

- **Closed loop transfer function**: second order system

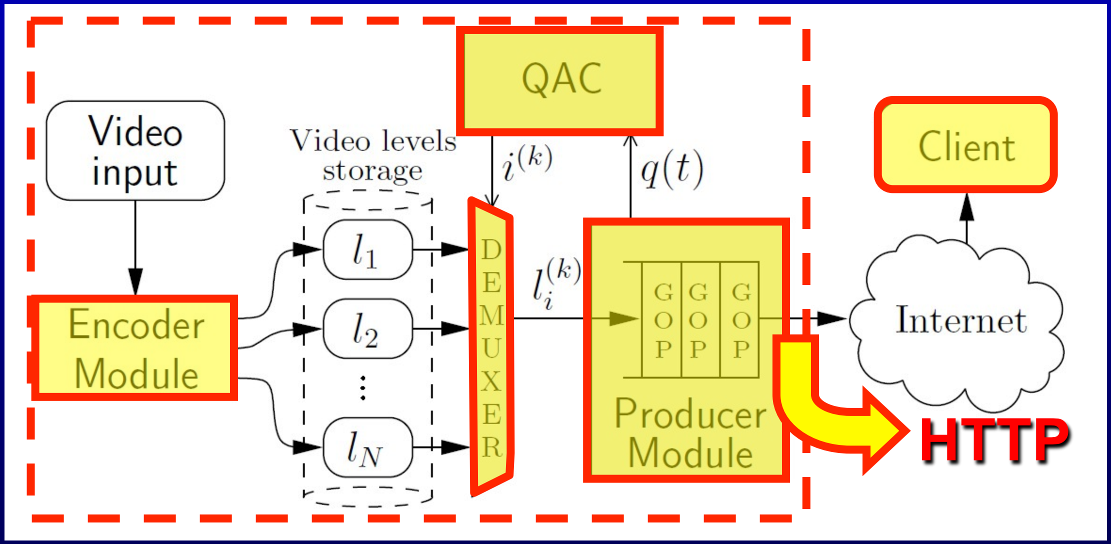
$$G_0(s) = \frac{K_p s + K_i}{s^2 + K_p s + K_i}$$

- Desired **settling time** $T_s$=30s (corresponding to a system bandwidth of 0.06 Hz in order to reduce oscillations) **damping factor** $\delta$=0.707
- It turns out: $K_i$ = 0.0356 and $K_p$ = 0.2667
- **Discretized control law** with sampling time $\Delta$T=0.5s

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

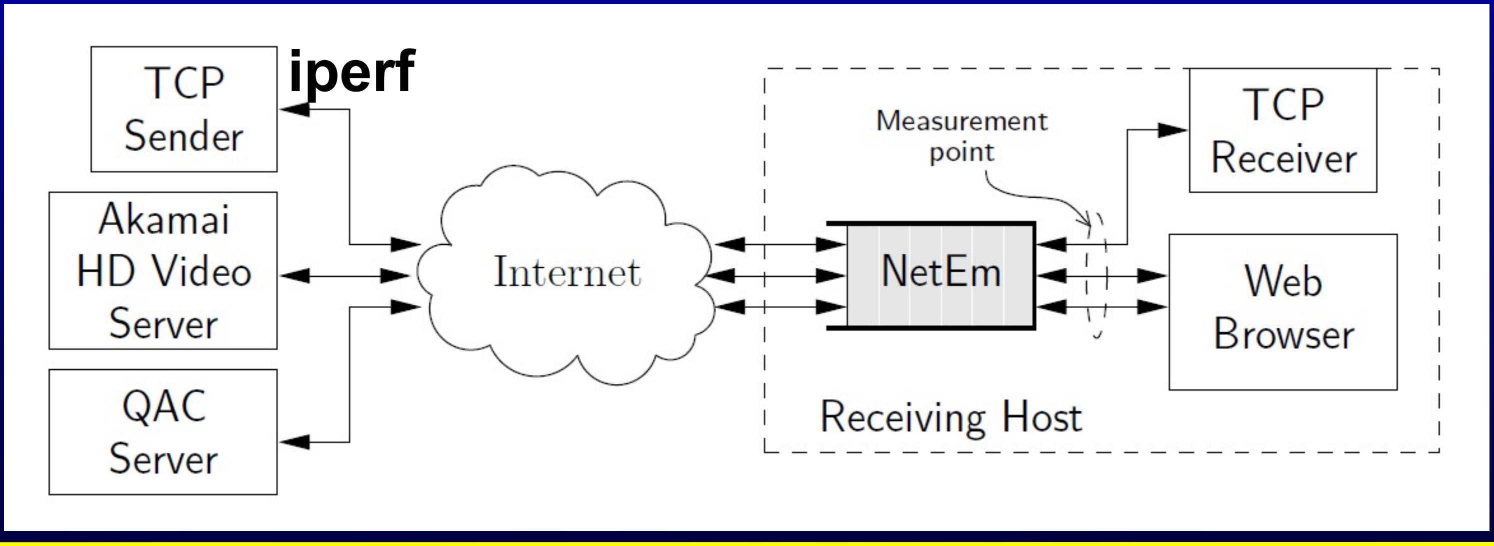
# Implementation of the proposed Streaming Server

- **Development environment**: Python + gstreamer + Twisted framework
- **Encoder module**: H264 or WebM (ex On2's VP8) GOP 1s, same video levels of Akamai (30 fps)
- **Producer module**: standard HTTP server, serving the adapted video to the client
- **QAC**: selects the video level $l_i$ to be streamed to the k-th user according to the control law
- **Client**: any client which is able to decode the video stream. A pre-buffering of 15s is recommended to avoid interruptions. In the experiment the client is a Flash application.



# Experimental Testbed

- Client: web browser connected to the Internet via our campus wired connection (100 Mbps)
- NetEm and IFB are used to set bandwidth b(t) at the receiver
- Traffic is sniffed after NetEm using **tcpdump**
- Python script to analyze the client-server protocol employed by Akamai



# *Experimental Scenarios and metrics*

**Experimental scenarios**:

1. A video flow over an available bandwidth varying as a step function
2. A video flow over an available bandwidth varying as a square wave
3. A video flow sharing a bottleneck with one TCP flow
4. Two video flows sharing the same bottleneck

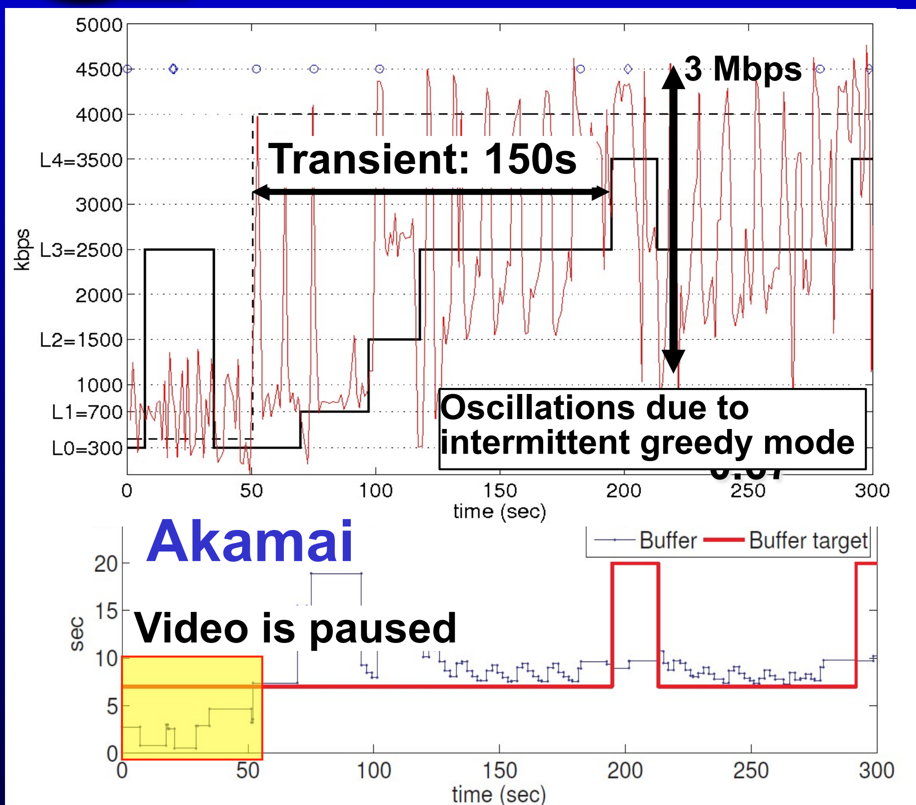
**efficiency index**: $\eta = E[l(t)]/\min(b, l_M)$

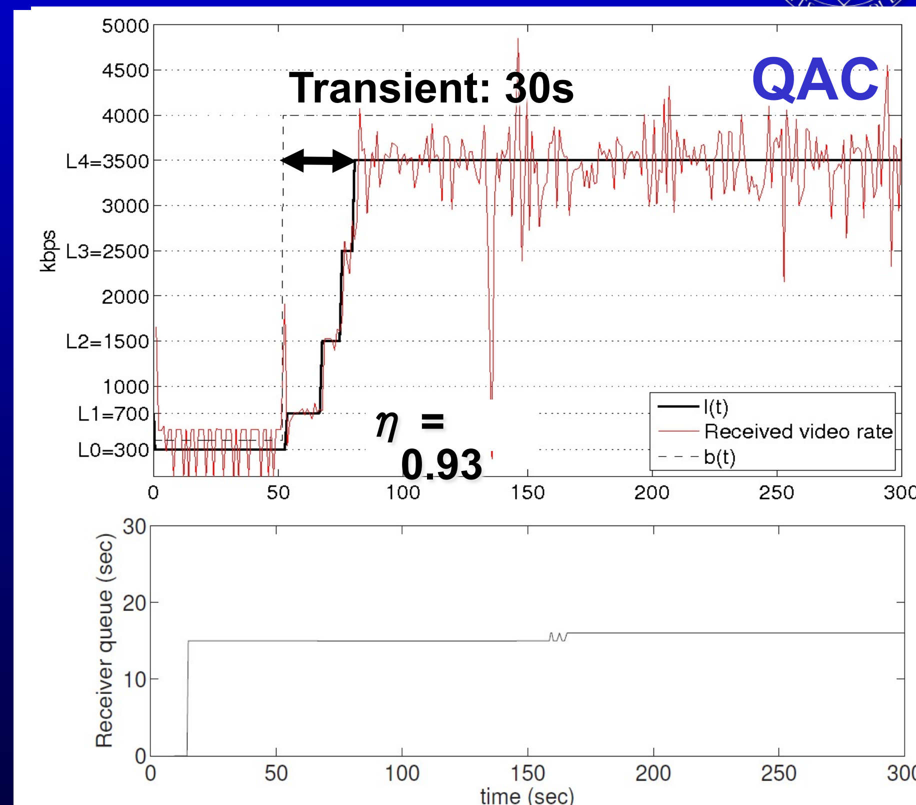
$E[l(t)]$ = average value of the video level

$l_M$ maximum video level, b is the available bandwidth

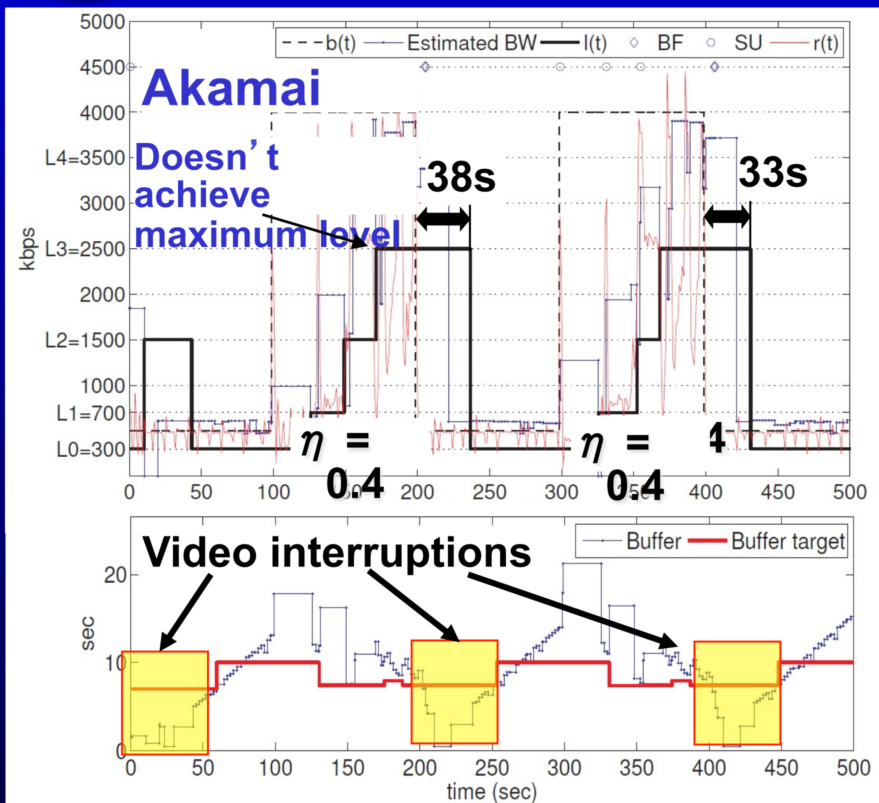
When $\eta = 1$ the maximum efficiency is obtained

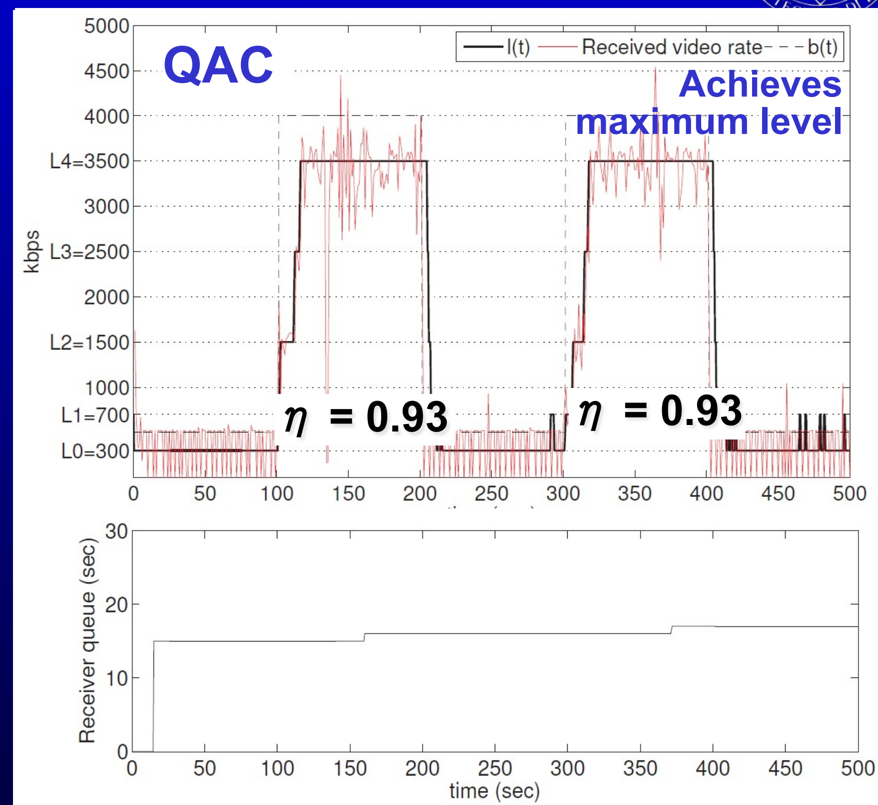
## *A video flow over an available bandwidth varying as a step function b(t): min value 500kbps max value 4000kbps*



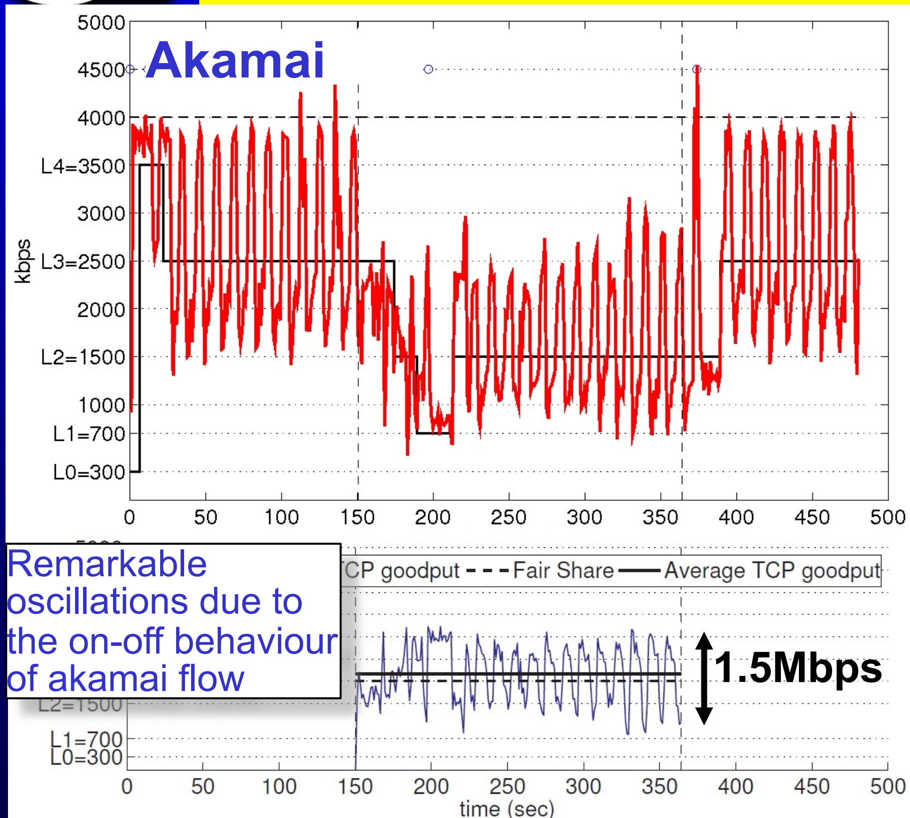



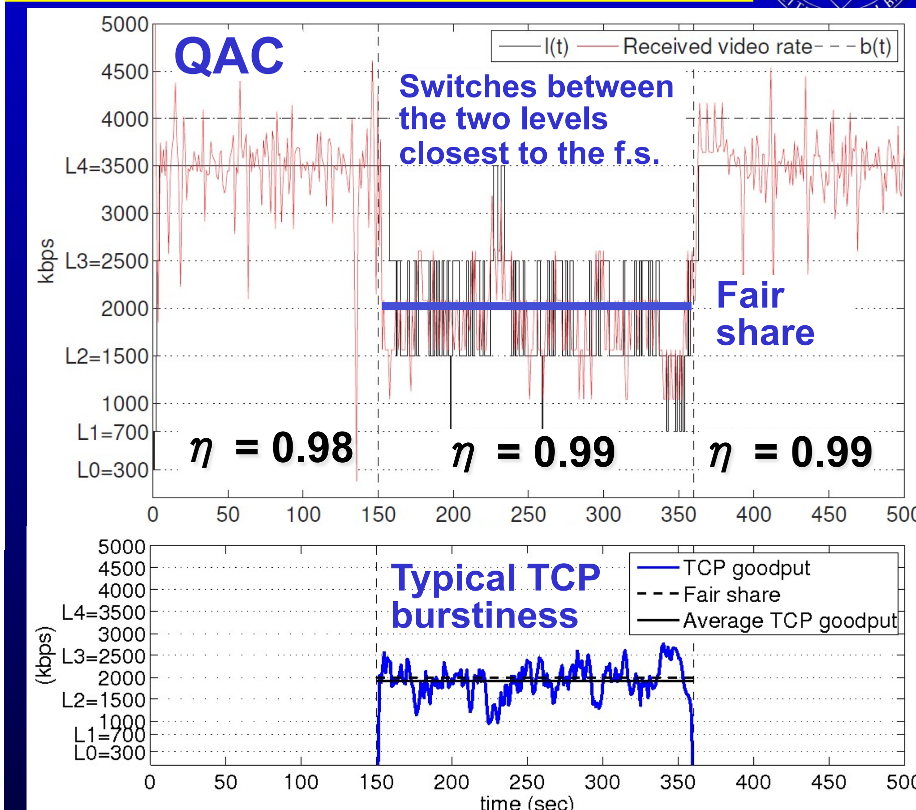
## *A video flow over an available bandwidth varying as a step function*
## *period 200s, min value 500kbps max value 4000kbps*



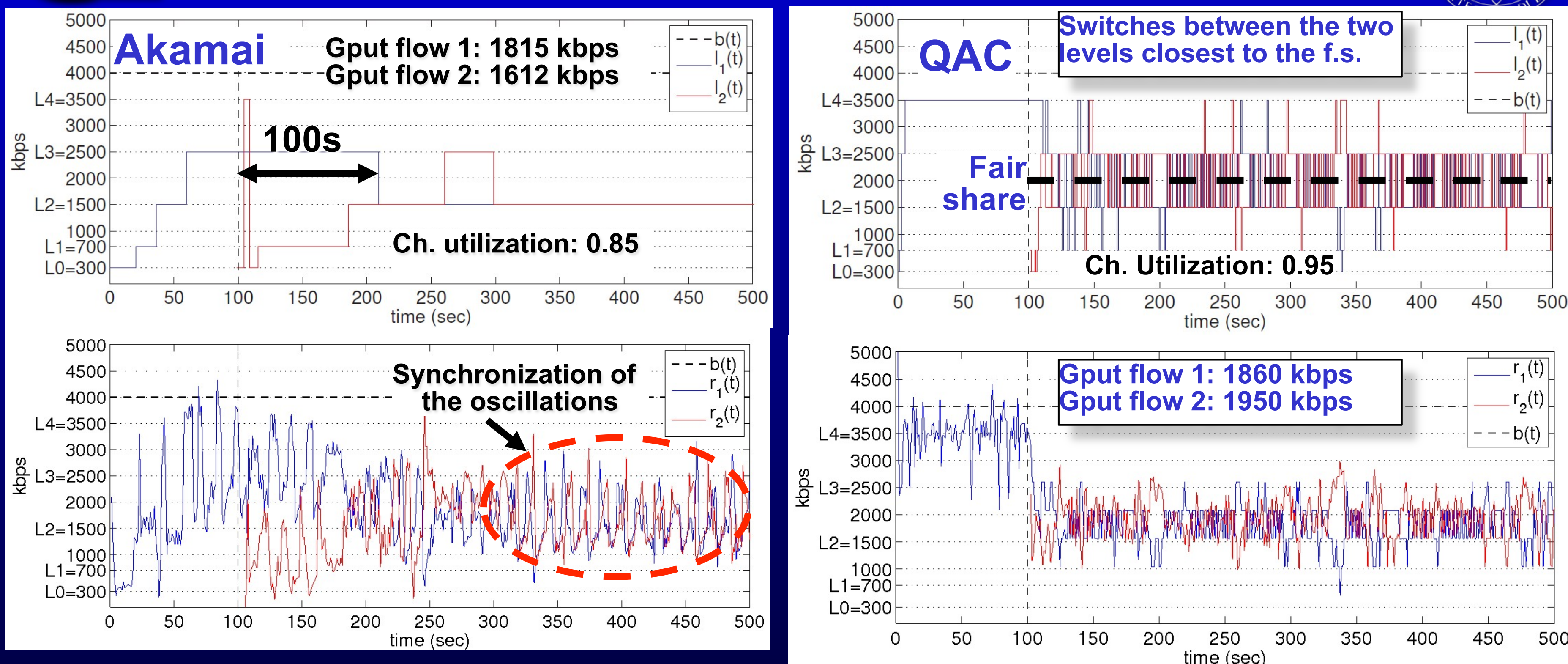



## A video flow with a concurrent TCP greedy connection
### Bottleneck 4000 kbps, fair share 2000 kbps. TCP starts at t=150s stops at t=360s



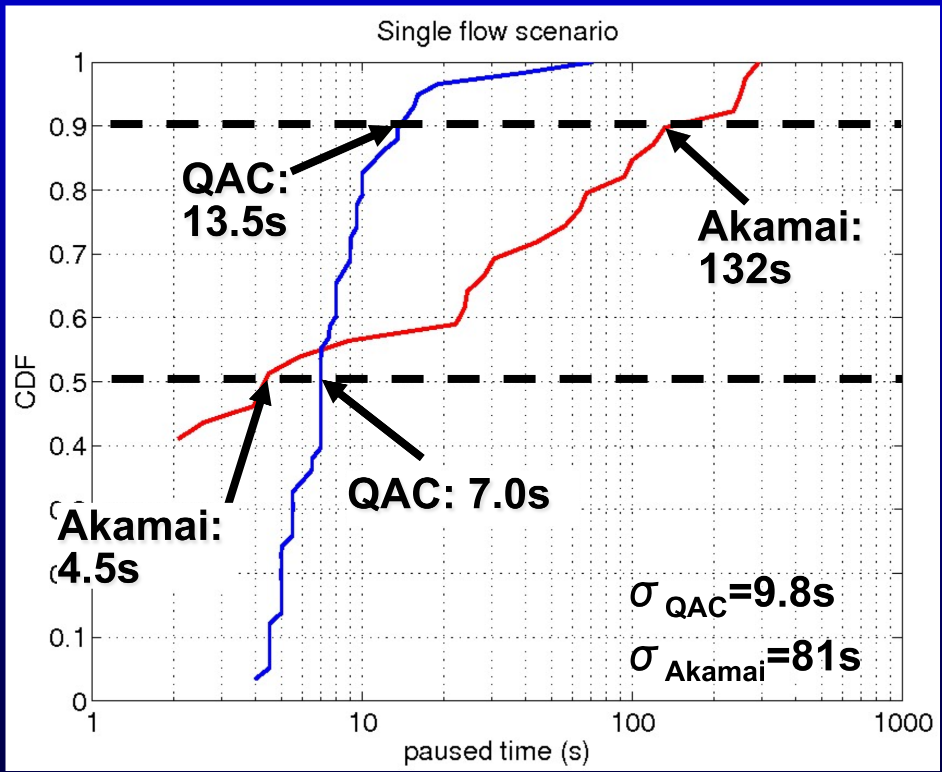



## Two concurrent video streaming sessions
## Bottleneck 4000 kbps, fair share 2000 kbps. Second flow starts at t=100s

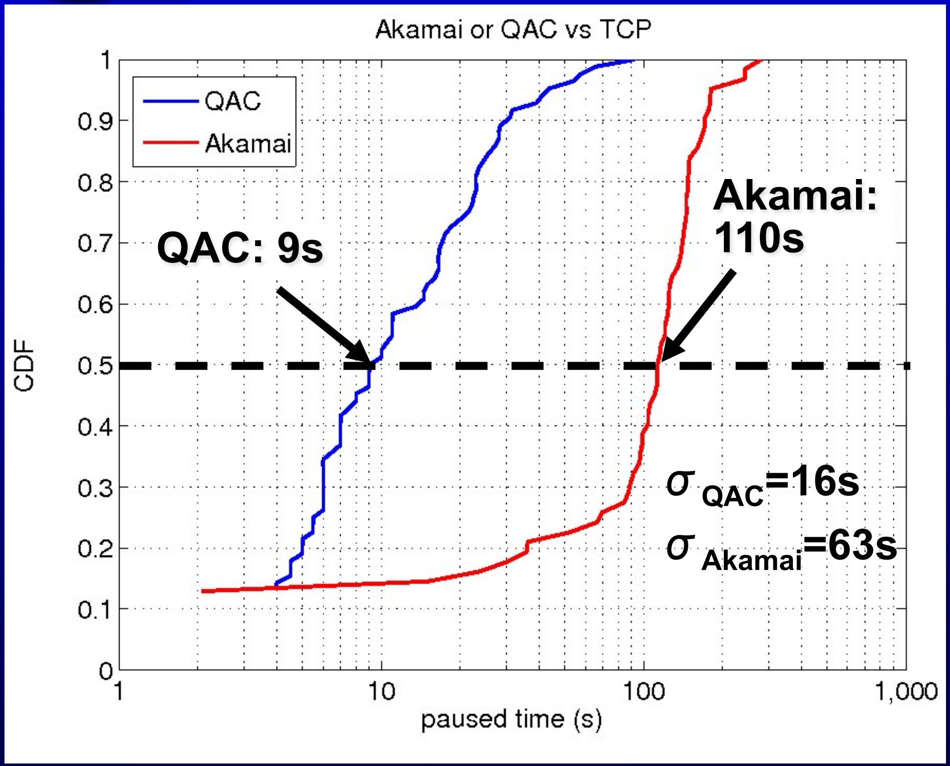


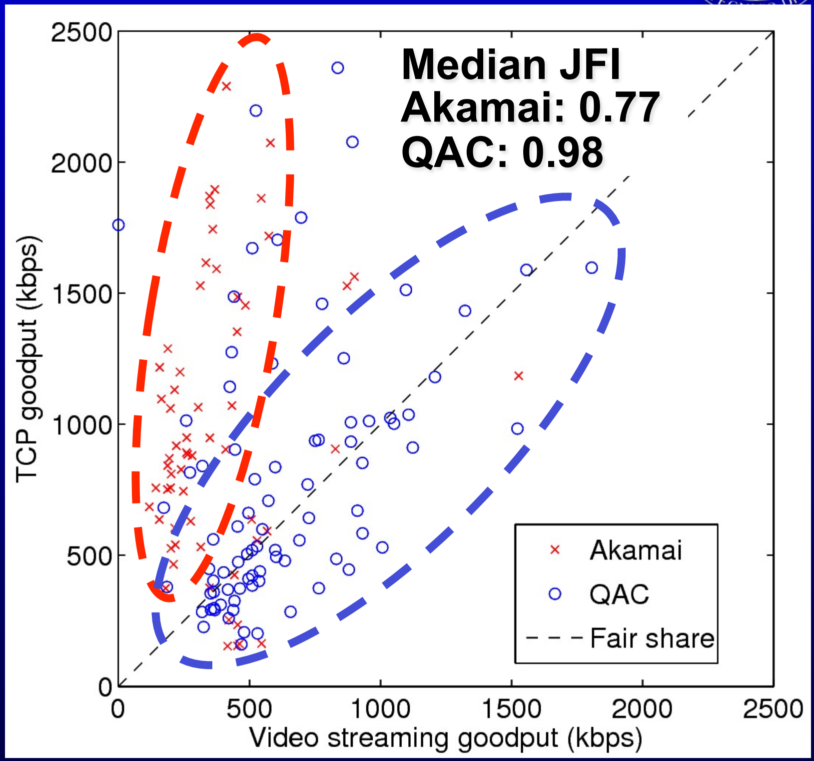
# Akamai vs QAC over a HSDPA link – Basic scenario

## Preliminary results



# Akamai vs QAC over a HSDPA link – Concurrent TCP
## Preliminary results



# Conclusions

- Akamai employs a stream-switching adaptation algorithm executed at the client
- The two control laws employed by Akamai to adapt the video level to Internet variable bandwidth are affected by a time-delay.
- We have proposed a Quality Adaptation Controller which is delay-free
- QAC is able to control the video level to match the available bandwidth with a transient time that is less than 30s, whereas Akamai with a transient that is around 150s.
- The proposed controller is able to share in a fair way the available bandwidth both in the case of a concurrent greedy connection and a concurrent video streaming flow
- Akamai underutilizes the available bandwidth due to the conservativeness of its algorithm based on heuristics and with abrupt reductions of the available bandwidth the video reproduction is affected by interruptions

# *Thank You!*