Practical Evaluation of H.264 Video Streaming over IEEE 802.11e Devices by Cross-Layer Signaling

Gabriel Lazăr, Virgil Dobrotă, *Member, IEEE*,
Tudor Blaga, *Member, IEEE*

Agenda

I. Introduction
II. Traffic Differentiation over WLAN
III. Cross-Layer and QoS Linux Issues
IV. CL Node Implementation
V. Video Streaming Scenarios
VI. Conclusions

I. Introduction

- Proliferation of real-time multimedia traffic
- Heterogeneous Internet
- E.g. – Video streaming over hybrid, wireless-wired IP networks
- Different nature of real-time traffic - not handled well in *best-effort* networks
- Over-provisioning – prohibitive due to limited resources in wireless environments
- Users expect high service quality independent of underlying access technologies
- => Reliable, real-time multimedia streaming:
  - challenging task
  - requires cooperation of all involved "actors": *all* networks, nodes and layers must collaborate and optimize functionality to achieve this common goal

I. Introduction (cont.)

- Strong demands imposed on video codecs and wireless links => new paradigm in network architecture design: the *cross-layer design* (CLD)
- Applying cross-layer optimization to multiple layers (application, network, data link) – optimal adaptation of the network
- MAC-centric CL architecture
  - traffic information sent from application layer to the MAC => flow/datagram priority is obtain based on QoS requirements
  - Linux based H.264 streaming server, capable of CL signaling between sender application and wireless card driver
- Focus
  - practical evaluation of video streaming performance over IEEE 802.11e devices
  - description of implementation issues

II. Traffic Differentiation over WLAN

- High-speed multimedia communication over wireless links: IEEE 802.11g (54 Mbps) and 802.11n (100 Mbps)
- Data Link Layer QoS: IEEE 802.11e
- DCF (distributed coordination function)
  - all stations compete to channel access with the same priority
  - in case of collision, the size of CW is doubled until it reaches a maximum value *CW[max]*; if the transmission is successful, CW is reset at the minimum value *CW[min]*.
- IEEE 802.11e - the enhanced DCF channel access (EDCA):
  - service differentiation using access categories (ACs)
  - traffic assigned to a category - sent to a separate transmission queue
  - each AC has different channel access parameters: *CW[min]*, *CW[max]*, arbitrary interframe space (*AIFS*) and transmission opportunity duration limit (*TXOP[lim]*)
  - traffic packets from a class with smaller *CW[min]*, *CW[max]* or *AIFS* will wait less, on average, before being sent than traffic from a category with bigger values
  - 4 ACs: from AC0 (lowest priority) to AC3 (highest priority), AC3 and AC2 - for multimedia transmission, AC1 and AC0 - *best effort* and background traffic

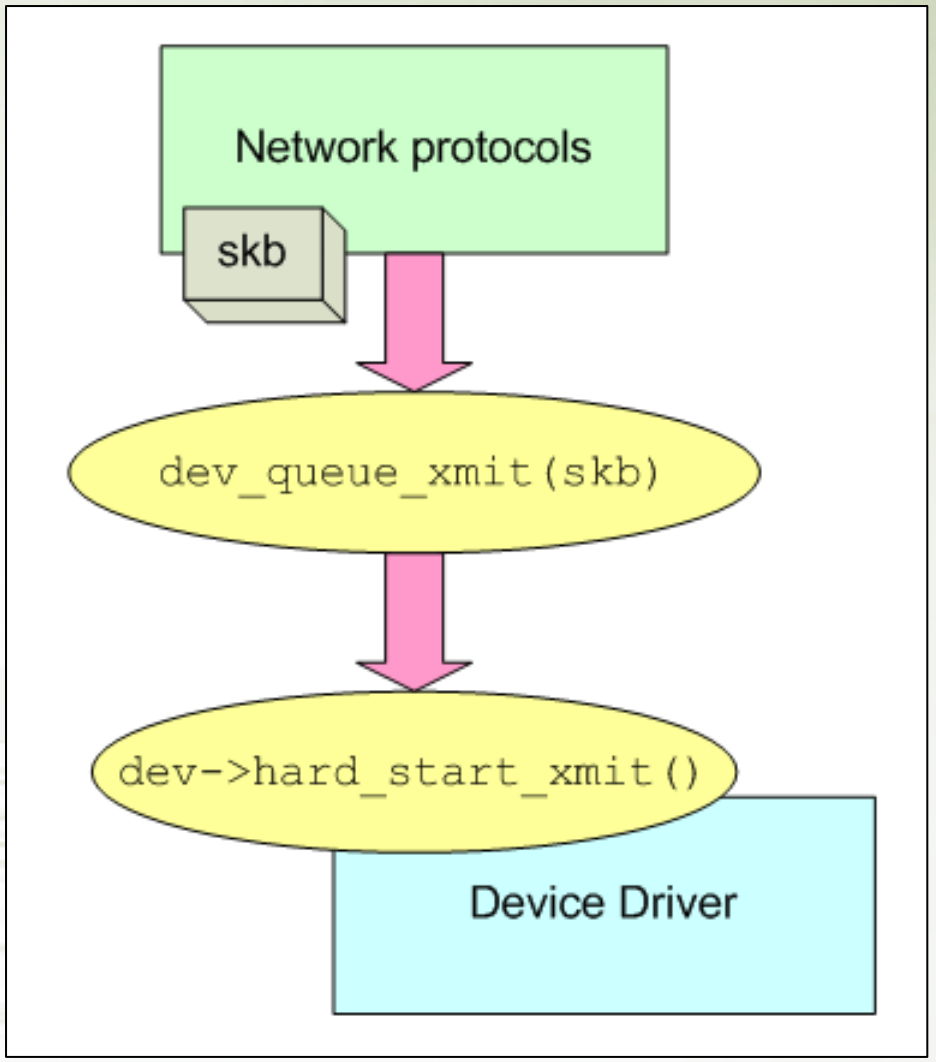
II. Cross-Layer and QoS Linux Issues

A. Linux Network Subsystem

- A *socket kernel buffer* – `skb` – wraps a packet during its processing lifetime at each layer inside the kernel space
- Network device (`struct net_device`) - uniform interface between software-based protocols and network adapters
- All registered devices are connected into a doubly linked list
- One of the pointers stored in `net_device` - the queuing discipline (`qdisc`) associated with packets transmitted through this network card

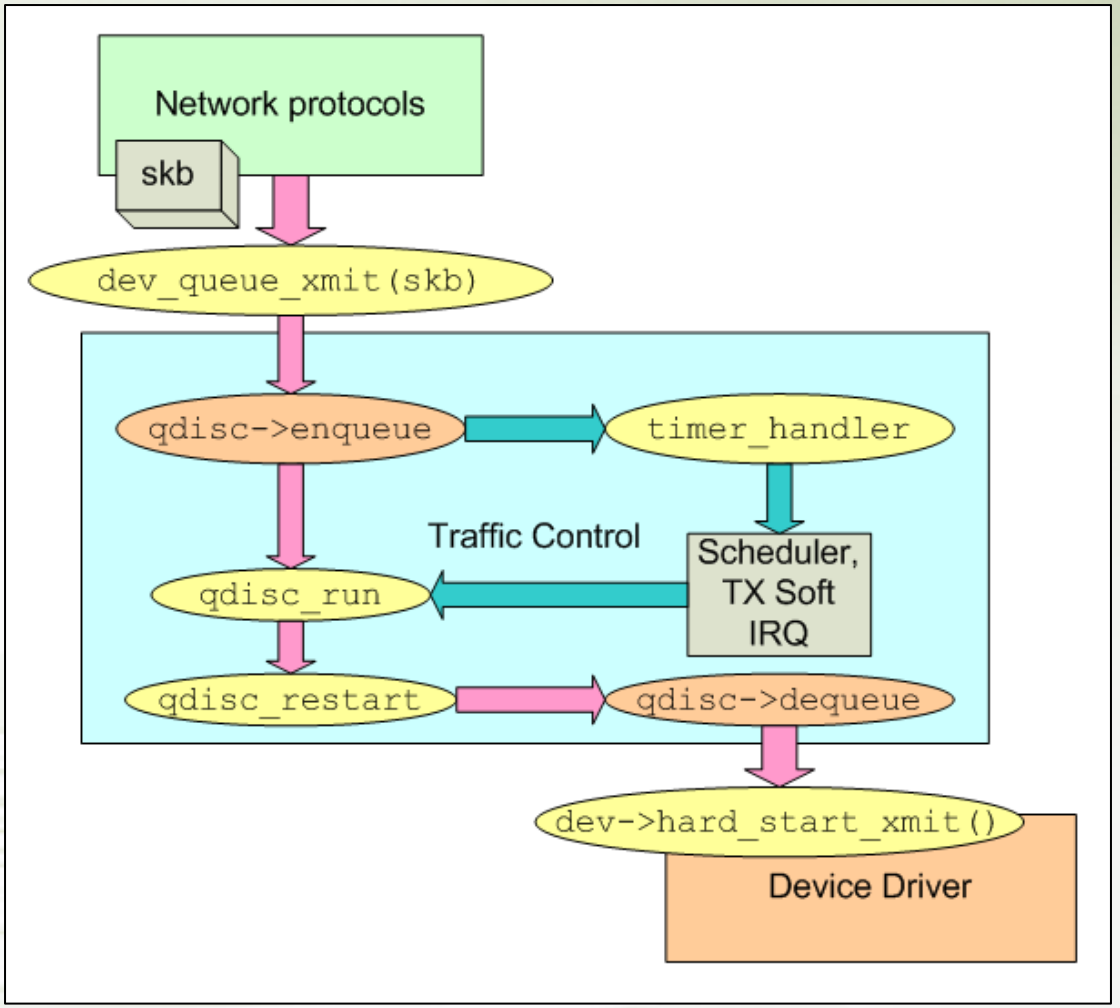
A. Linux Network Subsystem (cont.)

- Packet transmission for devices with no `qdisc` attached (`.enqueue` pointer is NULL)
- Protocol instances of higher network layers call `dev_queue_xmit(skb)`
- Network device to be used is specified in `skb->dev` parameter
- The packet is put immediately on the hardware queue through method `hard_start_xmit()`

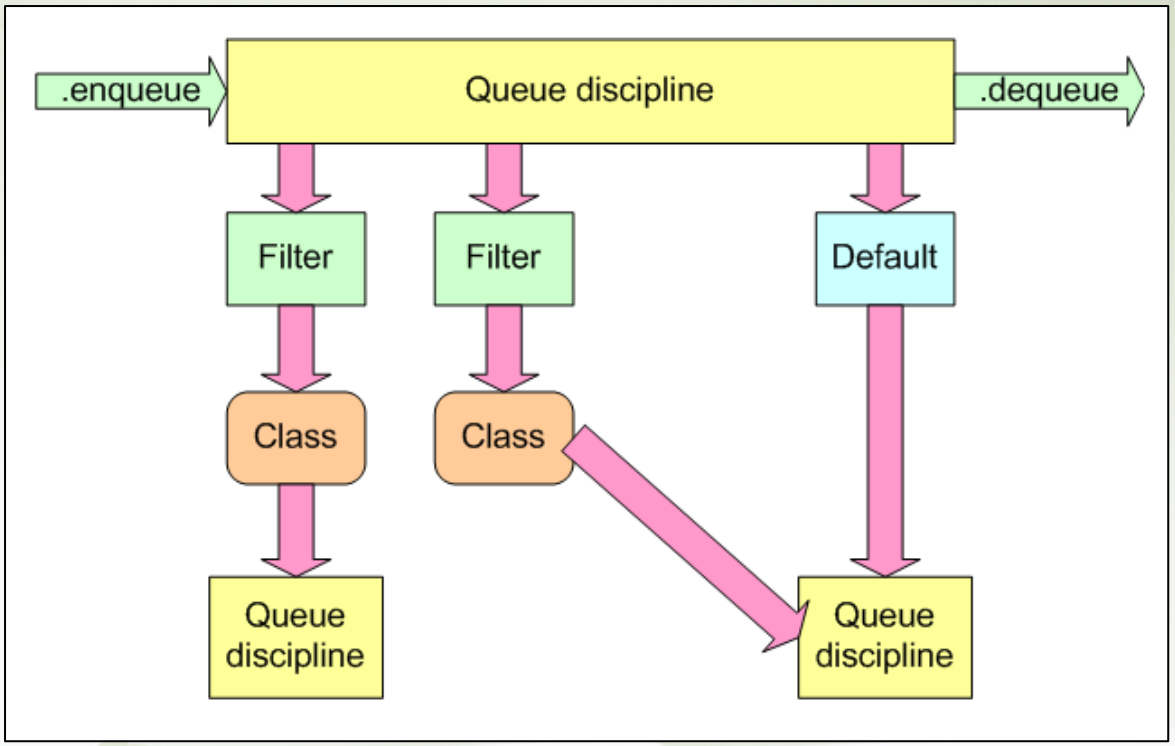


B. Linux Traffic Control

- Each device has normally associated a `qdisc`
- Advantages:
  - additional buffer space (software queues)
  - an entire kernel QoS implementation: custom schedulers, queue managers and detailed classification



B. Linux Traffic Control (cont.)



- `qdisc` – the basic building block of Linux traffic control subsystem
  - Simple – one queue with FIFO scheduling
  - Complex – a *classful qdisc* contains *classes* where traffic is sent according to custom filters; each class can contain another class or qdisc => powerful hierarchy of queues and schedulers
- User space configuration: `tc` tool from the *iproute2* package.

C. Link Characteristics and Traffic Control Efficiency

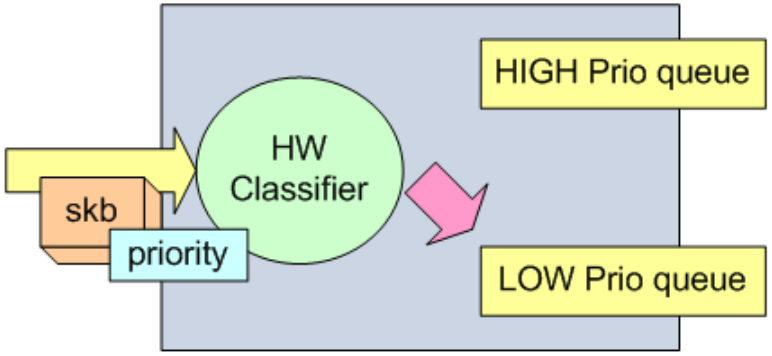
- Traditional network adapters
  - One Tx hardware queue (FIFO)
  - QoS – enabled in software (on top of the device driver) – e.g. traffic control in Linux
  - Intention
    - keep the HW queue empty
    - rely on the customizable queue disciplines configured in software
  - Method (constant link rate)
    - shaper configured at traffic control level, with shaping rate slightly lower than the actual link rate
  - Linux traffic control limitation: static QoS rules do not work well for links with variable capacity
    - shaping is hard to obtain with statically configured rates
    - device hardware queue not empty, or inefficient usage of the link rate
    - QoS mechanism that works – traffic prioritization (PRIO)

III. Devices with Multiple HW Queues

1) Linux Network Design Limitation

- General network driver rules:
  - When HW queue is full, driver calls `netif_stop_queue`
  - Scheduler of associated `qdisc` stops sending packets to the adapter
  - If `dev->hard_start_xmit` is still called, the packet is refused by the adapter and has to be requeued at the above layer; returned status code is NETDEV_TX_BUSY
  - After transmission takes place, HW queue not full anymore – driver calls `netif_start_queue`.

- Design Limitation: if driver has multiple HW queues, feedback is not sufficient
  - Example: Device with two HW queues, for packets of different priorities; if one of the queues is full, should the driver call `netif_stop_queue` ?



III. Devices with Multiple HW Queues (cont.)

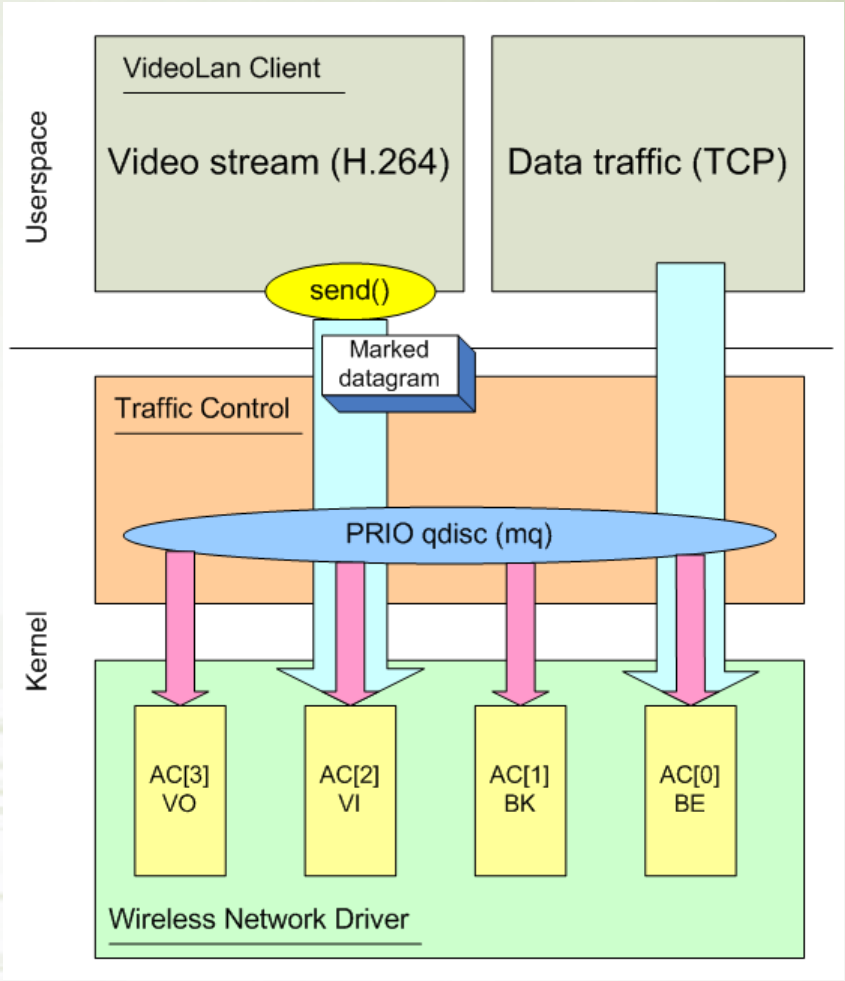
2) Cross-Layer QoS Issues

- If hardware QoS is present on device, it should be used (e.g. Wireless devices with IEEE 802.11e support – important packets have higher chances of transmission)
- Lack of correlation between QoS policies implemented at different layers can nullify its effects
- Linux traffic control is not aware of multiple hardware queues and Layer-2 QoS: the `qdisc .dequeue` operation does not have enough information from the device driver, in order to find out what `skb` to extract from its buffers

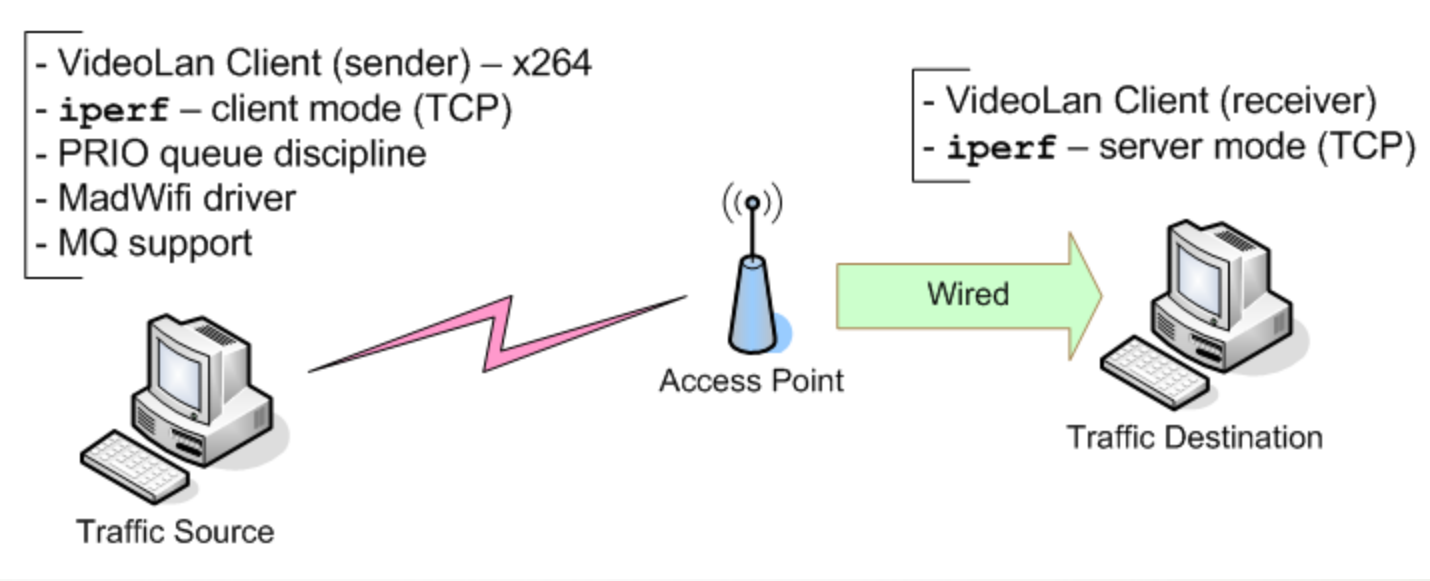
- Possible solutions
  - *Generic:* add multiqueue support to the Linux network subsystem and to the device drivers:
    - Expose the hardware queues to the `net_device` structure (recently implemented in new Linux kernels - `PRIO qdisc`)
    - Enable manipulation of each queue's state, e.g. `netif_stop_subqueue` (has to be implemented in each multiqueue device driver)
  - *Local:* creation of queue disciplines designed and tied to a particular multiqueue device driver

IV. CL Node Implementation

- CL-enabled video streaming server
  - *Userspace:* modified Linux build of VideoLan Client (VLC), capable of packet marking (based on importance in the video stream)
  - *Kernel (traffic control)*: recent kernel (2.6.24.3) with MQ support in the network subsystem (`PRIO qdisc`)
  - *Kernel (driver):* modified MadWifi driver (WLAN network cards with Atheros chipset)
    - Feedback at subqueue level
    - Packets are classified according to their marking and sent to the corresponding AC for transmission



V. Video Streaming Scenarios



- Details common to all scenarios
  - One H.264 video stream (x264 codec)
  - One TCP background flow (background traffic)
  - Physical rate set on the wireless adapter – 9 Mbps
  - PRIO qdisc – packets marked with TOS 0xB8 were serviced with higher priority than Best-Effort packets
  - Modified MadWidif driver also sends important packets through the AC2 (Video) HW queue.
  - 10-seconds Foreman sequence, encoded on the fly by VLC
  - Depending on scenario, none, some or all of the video packets were marked for the Video AC.
  - TCP traffic – sent as Best-Effort (no marking)

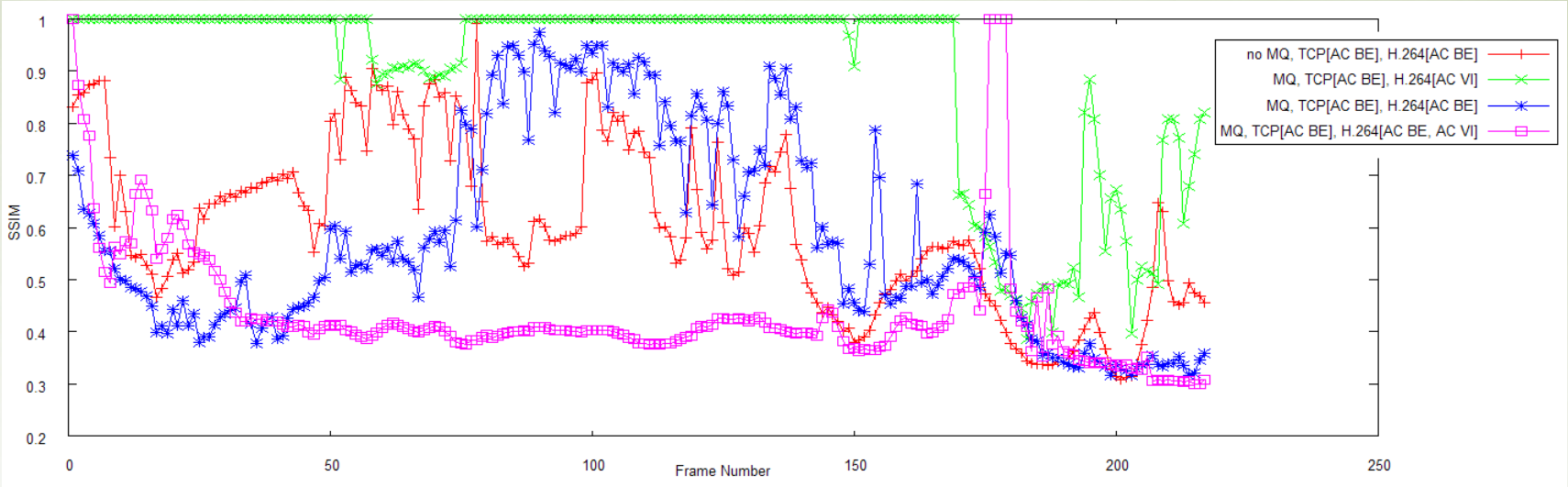
Scenario 1

- Comparative performance of the following four experiments:

  1) Original MadWifi driver (no MQ in the driver) – all packets sent through the *Best-Effort AC* hardware queue; full-queue event not signaled (driver accepts packets at any time)
  2) MQ support enabled, all video packets marked as high priority (TOS = 0xB8) and sent through the Video AC hardware queue
  3) MQ support enabled, but no marking for video packets; full-queue event is signaled
  4) MQ support enabled; only important packets in the video stream are marked for high priority transmission (based on x264 encoder hints)

- Results
  - Per frame SSIM evolution
  - Average SSIM and PSNR between sent and received h.264 encoded videos for each experiment

Scenario 1 (cont.)

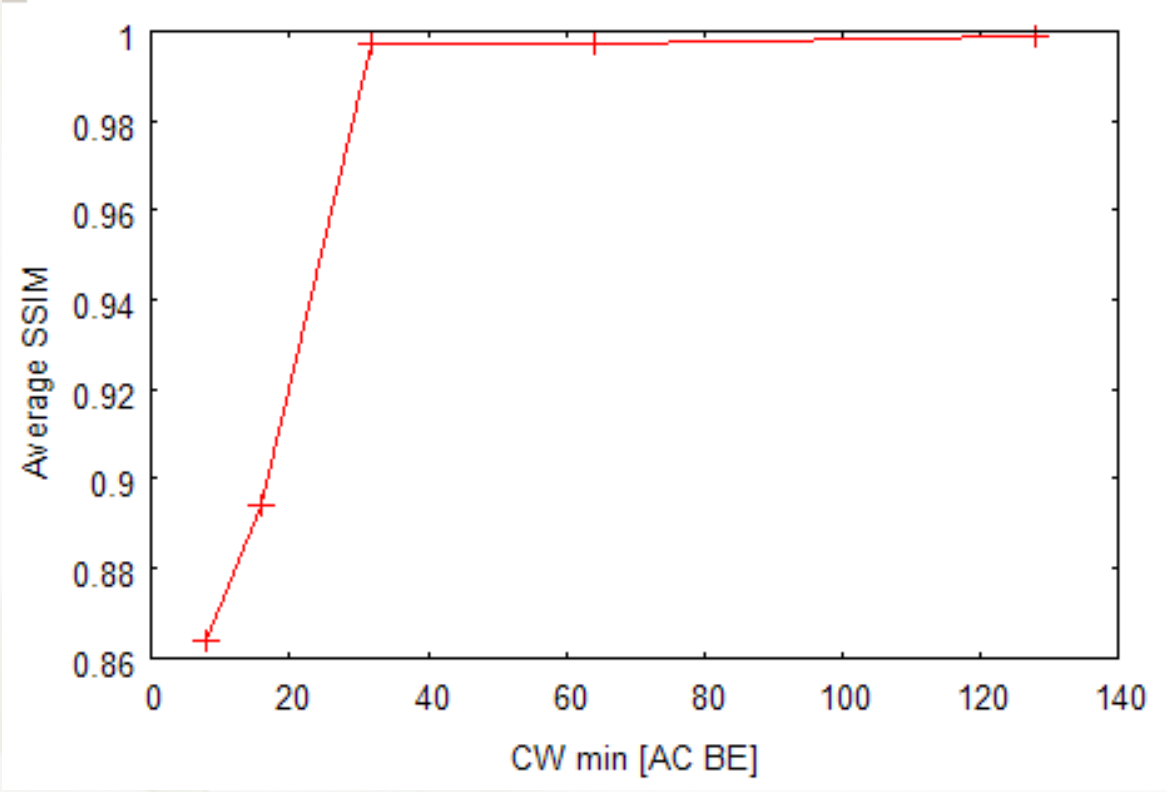


- Test 1 vs. 2 – Prioritized video packets clearly benefit from the MQ addition
- Test 1 vs. 3 – similar results, but difference in behavior
  - `send()` routine in VLC can provide feedback (Test 3)
  - No excessive `.requeue` operations (Test 3)
- Test 4 – worst results; VLC cannot handle packet reordering

Test Nr.	Average SSIM index	Average PSNR [dB]
1	0.60	18.52
2	0.89	71.66
3	0.58	18.06
4	0.44	16.41

Scenario 2

- Effect of different *CW min [AC BE]* values on video streaming quality (with all packets marked for *Video AC*)
- Default *CW min [AC VI]* = 8 (constant)
- Result: Average SSIM between sent and received h.264 streams



Scenario 3

- Effect of different H.264 encoding parameters (number of I, P and B frames) on the rate and quality (Average SSIM) of the video transmission
- Video data – sent at higher priority (AC2) than TCP background traffic (AC0) (similar with previous experiments)
- Possible CL optimization – if link rate is greater than video stream rate – increase number of I,P frames instead of increasing the target encoding rate (better protection vs. better overall quality at source)

I [%]	P [%]	B [%]	H.264 rate [Mbps]	Avg. SSIM
4.30	22.97	72.73	3.43	0.73
4.24	25.85	69.91	4.63	0.75
4.07	33.33	62.60	4.81	0.76
4.00	96.00	0.00	4.95	0.89
4.80	95.2	0.00	5.00	0.92

VI. Conclusions

- Practical implementation and evaluation of CL-enabled video streaming server is feasible
- CL signaling between userspace (application) and kernel (traffic control subsystem, driver) is possible
- Video flows can avoid competing with background traffic (=> better multimedia streaming) by packet prioritization
- Bottom-up signaling could help video streaming to adapt to the current HW buffer availability
- In practice, datagrams from a single flow should be sent through a single HW queue, in order to avoid packet reordering (legacy applications)

VI. Conclusions (cont.)

- Difference between *CW[min]* values for *Video AC* and *Best-Effort AC* can also improve the quality of the video transmission
- Changing H.264 parameters (percentages of I,P and B frames), the Avg. SSIM can be increased to a high quality level

- Future work
  - Try to adapt video encoding based on bottom-up signaling (e.g. current HW queue status, or current link rate)
  - Larger number of nodes (senders and receivers)
  - UDP background traffic
  - Background traffic measurements (in addition to video)