Multiport Streaming Of Matroska (MKV) Video Over IP

Ioannis Tsakos and Yiannis Andreopoulos

University College London

Abstract: Several popular IP-based services today involve entertainment media streaming. In this paper we propose (and make available online) Java-based media streaming over IP utilizing the Matroska (MKV) container, which is rapidly gaining popularity within online user communities. Fast and accurate access to individual MKV elements is ensured by a hinting mechanism tailored to the container. The achieved throughput and continuous playback capability is examined under RTSP/TCP with UDP transmission over Wi-Fi, which is setup via a *single* port or via *multiple* ports. New avenues for media traffic differentiation with the proposed framework are highlighted.

1. Introduction[1]

Media streaming tends to be the "stress-test" of modern networking infrastructure. The combination of the bursty, high-bandwidth, characteristics of media traffic with the stringent requirements for continuous playback and low start-up lag make practical deployments a very challenging task [1]. However, due to the ever-increasing user interest for media streaming over IP, such deployments are amongst the most popular online services today[2].

In this paper we are experimenting within the last (but not least!) part of the delivery networking infrastructure, i.e. the home or enterprise LAN, which is typically setup using Wi-Fi (e.g. using IEEE 802.11b/g). We start with the conventional basis of a Java-based client-server media streaming architecture based on rudimentary RTSP over TCP and with media payload transported via UDP. This connects (over Wi-Fi) the media server (e.g. home or office gateway) to the destination media platform, where content is decoded and viewed. Building upon this basic testbed, we propose:

- A simple and efficient hinting mechanism for pre-encoded (entertainment) media content encapsulated by the MKV (Matroska) container[3].
- A multiport connection mechanism for a single media flow. This allows for numerous possibilities, the rudimentary of which is traffic smoothing that is explored in this paper.
- Real-time signalling via RTSP in order to monitor connection and multiport flow status in real time.

Hinting of media bitstreams provides compact description metadata for individual stream elements allowing for efficient packetization from the streaming server with minimal stream parsing [2][3]. However, most hint proposals (besides Apple's QuickTime and Darwin streaming) stay at simulation level without real deployment. Importantly, to the best of our knowledge, no hint mechanism has been proposed for the popular MKV container, whose combination with H.264/AVC coding is becoming the *de-facto* standard for user-generated online video[3]. One of the streaming challenges with MKV is that it was not designed for streaming applications[3]. The proposed hinting attempts to remedy this.

Multitrack streaming can be defined as the separation of a single video flow into multiple "subflows" [2], which are transported from the server to the client using different connections and are then aggregated at the client for decoding and playback. Existing proposals tend to stay at simulation level and make certain assumptions for control of delay and queue length at the lower layers (e.g. at Link/MAC/PHY) that are infeasible without cumbersome customization. This tailors the proposed solutions to a particular cross-layer scenario. To the contrary, it is the aim of this work to be deployable to all IP infrastructures without requiring cross-layer customization. In fact, our choice of Wi-Fi LANs is purely for test purposes and our proposal is by no means restricted to such technology.

This work was supported by EPSRC EP/F020015/1.

[1] **Acronym synopsis:** IP: Internet Protocol; MKV: Matroska Video; RTSP: Real-time Streaming Protocol; TCP: Transmission Control Protocol; UDP: User Datagram Protocol; Wi-Fi: wireless systems certified by the Wi-Fi Alliance; LAN: Local Area Network; AVC: Advanced Video Coder; MAC: Medium Access Control; PHY: Physical layer; QoS: Quality of Service; VLC: VideoLAN Client; APP layer: Application layer.

[2] e.g.: BTVision, BBC iPlayer, Channel 4 4oD, LOVEFiLM "watch online", YouTube, Akamai, PPLive, PPStream.

[3] http://www.matroska.org/technical/guides/faq/index.html; A search for "MKV x264" on Google provides more than 16 million hits, indicating that the container's bundling with H.264 coding is enormously popular.

2. Matroska for Media Streaming: Hinting Mechanism

An outline of the Matroska container is given in the "Matroska Info" and "Matroska Summary" parts of Figure 1 (each derived from mkvinfo – mkvtoolnix[4]) based on: *(i)* the internal layering into Clusters, Block groups, Blocks and Frames; *(ii)* a frame-by-frame description[5]. Different tracks (e.g. audio, video, subtitles) are distinguishable by their track id. Each element includes the playout time. While the "Matroska Info" of Figure 1 gives the stream dependencies hierarchically (Clusters, Block Groups, etc.), "Matroska Summary" gives the dependencies in a linear, element-by-element manner.

Matroska Info: +	Matroska Summary: =	Proposed Matroska Hint Track:
|+Header, pos, size	I: track, time, pos, size	**H** pos size
|+Cluster, time, pos, size	P: track, time, pos, size	**E** time-range pos size #I #P #B #O
| +Block group, track no, time, pos, size	I: track, time, pos, size	...
| +Block, time, pos, size	P: track, time, pos, size	**E** time-range pos size #I #P #B #O
| +Frame, time, pos, size	I: track, time, pos, size	**D** time-range pos size #I #P #B #O
...	P: track, time, pos, size	**D** time-range pos size #I #P #B #O
| +Block, time, pos, size	...	**E** time-range pos size #I #P #B #O
...		...
| +Block group, time, pos, size		Definitions:
...		**H**: header; **E**: element; **D**: dependent element
|+Cluster, time, pos, size		#I: number of intra frames (video) in this element
...		#P: number of predicted frames (video) in this element
// codes for efficient random access		#B: number of bi-predicted frames (video) in this element
// appear at the end of the file		#O: number of <audio, subtitles> frames in this element

Figure 1. Left: Outline of information extracted from the MKV container; Right: Proposed hinting.

The first stage of constructing hinting mechanism for streaming consists of identifying what can be dropped from the container while allowing for uninterrupted playback at the client (decoder) side. We confirmed that dropping individual Clusters, Block Groups, or Blocks *in their entirety* from an MKV file allows for continuous playback within popular media players, such as VLC. However, this leads to visual (or audio) artifacts, the severity of which depends on the type and number of elements contained within each dropped component. For example, dropping intra frames of video creates the well-known "ghosting" drift effect from missing the low-frequency data during the temporal prediction of several decoded frames. Importantly, if the drops are partial, e.g. a partial Block or Block Group is passed to the decoder without obeying the MKV spec, the playback is abruptly terminated. Finally, as expected, drops of header information invariably lead to the stream not being decodable.

Based on these observations, we construct the MKV hinting mechanism by aggregating all sequential elements smaller than the smallest UDP packet size into *hint elements*. We indicate an aggregated header and hint element explicitly by the H and E tags, as shown in the right part of Figure 1. In addition, apart from playout time range of the hint elements, we indicate their position and size information within the MKV stream. We also indicate the number of intra (I), predicted (P) and bi-directionally predicted (B) video frames aggregated within each hint element. The number of audio or other frames within each hint element is indicated by the O tag. This was derived by combining the information within "MKV Info" and "MKV Summary" by using the positional and timecode information to identify each individual MKV element. The derived hinting is rapidly parsed by the streaming server once the session is setup (and, of course, during the actual live streaming) and the related stream truncation points, as well as the importance (and timing information) of each individual element is readily identified. No hint data is transmitted to the client. Since high-bitrate video encoding may derive I frames larger than the maximum allowable IP datagram, we allow for truncation of elements in two or more parts and indicate all dependent parts by the D tag, as shown in Figure 1. This requires that the client ensures that all co-dependent elements have been received before accepting them in the received MKV stream, or else continuous playback may be compromised.

3. Multiport Client-Server Architecture

Once the RTSP session between client and server is established, we setup N UDP ports for N concurrent UDP sessions, $N \in \{1,2,3,4,5,6\}$. We did not experiment beyond N=6 in this paper.

[4] http://www.bunkus.org/videotools/mkvtoolnix/

[5] Within MKV, the notion of "frame" is generic and includes audio, video, or other media components.

Server Side – During the actual streaming, we can prioritize the MKV traffic in a number of ways and according to the number of live ports.

- *Quality-driven*: For example, I frames can be transported from port 1, P frames from port 2, O-tagged frames from port 3 and B frames from port 4. Finally, ports 5 and 6 can occasionally transport duplicate MKV packets, such as duplicate copies of elements with dependencies.
- *Traffic-driven*: By using a running average for the MKV packet sizes streamed per port within the last T milliseconds, we can assign each incoming MKV packet into the port with the minimum average traffic (i.e. a traffic fairness strategy). Alternatively, for MKV flows with large bitrate variation across time, we can assign the range of packet sizes accommodated by each port. Such approaches lead to lower layers experiencing smoother traffic per UDP port that is easier to handle.
- *Content-driven:* Based on the capability of the Matroska container to multiplex several tracks (e.g. multiple video/audio/subtitles) as well as include chapters, tags and menus in the format (amongst others), each port can be used to transport different media elements according to real-time RTSP requests made by the client. This creates truly dynamic media-on-demand MKV streaming.
- *Connection-driven*: By utilizing network or transport layer information fed back from the client in real time via RTSP with respect to routing delay estimates, packet losses, and inter-packet arrival times as an indication of jitter (see below for the feedback mechanism), the server can assign transport-quality indicators to each UDP port in real time and appropriately select appropriate stream components for each UDP connection, either in quality-driven or in traffic-driven mode.

Importantly, all lower layers will see each UDP connection as a separate UDP flow. Hence, by controlling the inter-packet transmission times for each UDP flow, we can create traffic differentiation in terms of traffic volume transmitted, minimum and maximum burst size, etc. This can be combined with lower-layer protocols providing for network-layer QoS [4], or MAC-layer QoS [5], in order to allow for graceful degradation according to congestion status, error rates experienced at the PHY, etc.

Client Side – At the APP, we utilize a single buffer of 20 packets to store and reorder all incoming MKV packets via the N connections. The buffer is periodically emptied ("flushed") with the packets reordered according to their number in the MKV stream fed to VLC, in order to ensure correct playback. Packets arriving significantly out of order may be dropped at this stage. The client collects: inter-packet arrival times for each port, VLC's playout point, number of last packet received, and any related link quality metric and feedbacks them to the server every 100 received packets by writing in the RTSP socket with request type REPORT, which was created explicitly for this purpose.

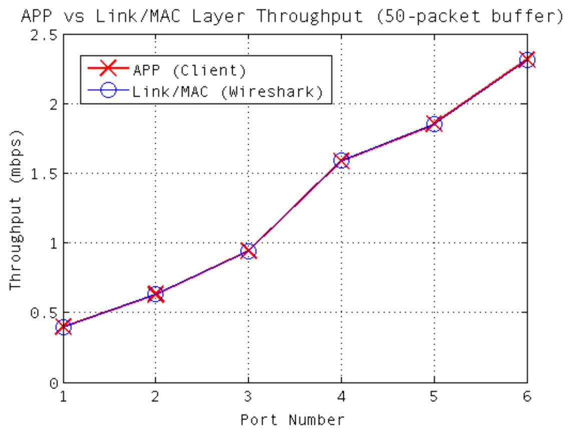
4. Results

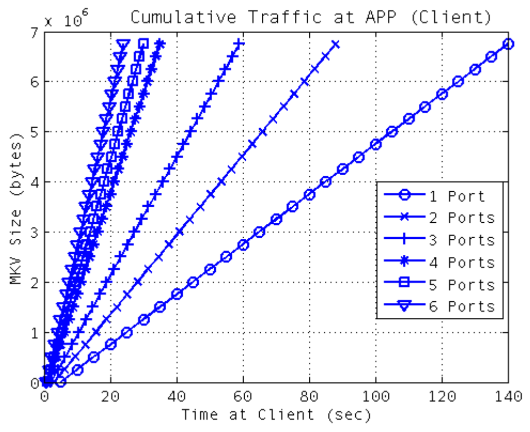
We only include indicative results in this paper as our full implementation is available online[6]. In the first round of experiments, we validate the behaviour of our streaming client-server design under increased number of UDP ports. To this end, we set fixed UDP packet size of 1250 bytes and fixed inter-packet transmission time of 25ms per UDP flow. Standard-definition (SD) 90-second videos were used for these experiments, compressed with H.264/AVC and encapsulated in MKV (using VideoLAN's x264). The sender run on a low-end Windows netbook and the client run on a medium-range Ubuntu laptop – they were both connected as access points to an IEEE802.11b/g router.

Figure 2(a) shows the throughput scaling measured at the client APP and Link/MAC layer when increasing the number of ports, while Figure 2(b) shows the cumulative traffic at the client's APP. These results demonstrate that *(i)* with a 50-packet buffer for incoming packets and sufficient computational power to run the Java client software, no drops are observed from Link to APP layer at the client; *(ii)* the cumulative traffic scaling saturates beyond four ports in this experimental testbed. The 50-packet buffer is for incoming UDP data and is separate from the 20-packet buffer at the client's APP that is used for sorting prior to MKV stream output to the player.

In the second round of experiments, we tested our entire design with variable packet sizes ranging from 1250 bytes to 64kbps and thus increased the inter-packet transmission time to 50ms per UDP flow. SD-resolution entertainment movies encoded in their entirety into variable-bitrate (VBR) H.264/AVC+MKV streams were used. This corresponds to the content provider giving "best-quality" VBR streams and posing the challenge of low-delay streaming without transcoding.

[6] http://www.ee.ucl.ac.uk/~iandreop/UNV.html





(a) **(b)**

Figure 2. (a) Throughput at client APP and Link/MAC layer under IEEE 802.11g (without residual traffic), measurements at the lower layers were done using Wireshark; (b) Cumulative MKV stream at the client's APP across time.

To stress-test our deployment, we selected a particularly challenging 90-second section of an action movie (SD resolution) with very rapid and irregular camera motion, explosions, flashes, etc., that led to VBR-coded video with very high rate variation, such as going from 1.5mbps to 8mbps within 3 sec. We deployed 1-port, 3-port and 5-port versions of our design; in the multiport cases, we assigned MKV hint elements to each port in real time according to their size, thereby creating multiple subflows with smoother traffic characteristics. Table 1 reports our observations at the client. Traffic smoothing provides for efficient delivery within the 3-port case, while the 1-port and 5-port versions lead to worse results due to insufficient flow rate (1-port) or due to consecutive UDP or client APP drops (5-port) – in both cases continuous playback is impossible for this challenging segment. This simple case exemplifies trade-offs for media prioritization in the manner described in Section 3.

Setup	Streaming Time	Received Packets	Flushed Packets	Behaviour
1-port	140sec	2845	2845	Some artifacts, video playback stops after a few seconds
3-port	72sec	2768	2767	Some visual artifacts, uninterrupted playback
5-port	61sec	2720	2704	Severe audio-video artifacts, video freezes after ~1min

Table 1. Summary of 1-port, 3-port and 5-port streaming stress-testing with a 90-sec MKV stream. Flushed packets indicate the packets send to VLC from the client; drops may occur during this process.

5. Conclusion

We propose practical multiport hinting and streaming using Matroska MKV. Our experiments demonstrate that straightforward throughput scaling is achievable within Wi-Fi LANs via our approach. In combination with RTSP-based feedback, many options for traffic prioritization become readily available without the need to invoke complex cross-layer interaction. Further implementation details are available in the first author's MSc dissertation [6].

References

[1] E. Mykoniati, R. Landa, S. Spirou, R. G. Clegg, L. Latif, D. Griffin and M. Rio, "Scalable peer-to-peer streaming for live entertainment content," *IEEE Comm. Mag.*, vol. 46, no. 12, pp. 40-46, Dec. 2008.

[2] P. Amon, T. Rathgen and D. Singer, "File format for scalable video coding," *IEEE Trans. on Circ. Syst. for Video Technol.*, vol. 17, no. 9, pp. 1174-1185, Sept. 2007.

[3] *Information Technology—Coding of Audio-Visual Objects—Part 12: ISO Base Media File Format* (Technically Identical to ISO/IEC 15444-12), ISO/IEC 14496-12: 2005.

[4] S. Georgoulas, *et al*, "An integrated bandwidth allocation and admission control framework for the support of heterogeneous real-time traffic in class-based IP networks," *Computer Comm.*, vol. 31, no. 1, Jan. 2008.

[5] *IEEE 802.11e/D5.0, Draft Supplement to Part 11*: Wireless Medium Access Control (MAC) and physical layer (PHY) specifications: MAC Enhancements for Quality of Service (QoS), June 2003.

[6] I. Tsakos, "Multiport video streaming system over IP," *MSc Dissertation*, Dept. of Electronic and Electrical Engineering, University College London, online at: http://www.ee.ucl.ac.uk/~iandreop/Tsakos_MSc.pdf