THE CHALLENGES INVOLVED IN GENERATING ST 2110 STREAMS ON STANDARD IT HARDWARE

Kieran Kunhya

Founder and CEO, Open Broadcast Systems, United Kingdom

ABSTRACT

ST 2110 has been promoted by the industry as a new standard for video-over-IP (Internet Protocol), allowing for live broadcast production to take place using high bandwidth IP networks instead of SDI (Serial Digital Interface) using coaxial cables. However, as it as hardware-centric standard, it has extremely tight timing requirements that make it very difficult to generate streams on standard IT hardware. Achieving this requires solving a large number of technical challenges. This paper will explore these challenges and how they were solved in one of the few practical software implementations.

INTRODUCTION

Whilst new Uncompressed video-over-IP standards such as ST 2110 use technologies from the IT industry such as IP and Precision Time Protocol (PTP), there are numerous technical challenges involved in complying with a standard that has exceptionally tight timing requirements as well as high data rates when using standard IT hardware and software. Solving these challenges requires a detailed understanding of entire research fields such as high-performance networking, high-performance pixel processing and broadcast engineering as well as creative problem solving in this area. All in all, this requires several years of painstaking work to solve and its complexity cannot be overstated.

As the author can attest to, making the wrong design decisions can cause months of work to be scrapped. Often the correct design decisions to solve the aforementioned problems fly in the face of existing orthodoxy of software engineering practices. This paper will explain these in detail.

Arguably, this is the biggest blocker towards the broadcast industry gaining the flexibility to move to IT-based live production in order to compete with FAANG media giants such as Netflix. Without solving these technical challenges, the industry will depend on fixed-function, inflexible hardware that does not allow it to scale to audience needs without substantial effort such as rewiring a facility.

The move to IP for live broadcast is comparable to the move to electric vehicles. But in the same way that an electric engine cannot be just bolted onto existing vehicle designs (this of course does not stop manufacturers), adding IP capabilities often requires a full product redesign in order to achieve the maximum benefits.

TECHNICAL BACKGROUND

ST 2110 streams involve separate flows of Uncompressed Video and Audio and Ancillary data flows. In particular the Uncompressed Video flows are high bitrate, of the order 1-10Gbps, a very high data rate. Whilst Audio flows have modest bitrates of 10-20Mbps, depending on the profile this could nonetheless have a high packet rate of up to 8000 packets-per-second. The timing of these packets uses PTP, meaning that video and audio data is released N frame periods after the PTP epoch of 1st January 1970. Furthermore, in practical facilities, each stream is duplicated across two networks, known as 2022-7. Complying with the above requirements is a challenging problem.

Standard IT Operating Systems (OSs) are not designed to send high packet rate UDP flows as this requires a single system call per packet in contrast to TCP where batching of packets can occur. A single 1080p50 stream consists of around 200,000 packets per second, a very large number of system calls per second. Furthermore after the Spectre and Meltdown vulnerabilities the system call overhead is increased, further complicating the problem.

Packet Timing Requirements

There is an additional problem in that in ST 2110 Uncompressed Video must comply with 2110-21 Traffic Shaping and Delivery Timing for Video specification. This paper will not go into the specifics of the timing model as this is explained in detail in Kostiukevych et al (1) and has large quantities of arcane mathematical variable names. Instead this paper will go into high-level principles behind compliance with the timing model.

This means complying with the Gapped Narrow (referred to as N) profile as this is what is deployed in real world facilities and will continue to be in the foreseeable future. In particular this profile is very hardware-centric, with only small packet bursts of the order of 20-40us allowed as well as including a packet “gap” where the legacy Vertical Blanking Interval (VBI) used to exist in SDI. This allows for usage of hardware receivers with small amounts of RAM. Packet bursts of the order 20-40us are exceptionally small for a traditional software application which broadly operate in the order of 500-1000us timescale for its operations.

As a result it is very challenging to send a packet that is cotimed with the PTP epoch plus the gap as the timescales software operates on are much larger than the requirements of the Narrow Profile. For audio, a similar set of problems apply, although the audio specification is relatively vague, mandating buffering of 3x the frame duration (around 375us), still a tight constraint for a software sender.

As discussed previously, virtually all practical facilities use 2022-7 and thus the timing requirements must also be met for 2022-7 such that a packet consisting of the same payload (but different headers) must be cotimed across the two different network links.

Pixel Packing Format

Most of live broadcast production is using the YUV 4:2:2 10-bit pixel format. In 2110 this uses a particularly software unfriendly packing, known as a pgroup as shown in Figure 1, where four 10-bit samples (two pixels) are packed into 40 bytes.

It can be seen from Figure 1 that none of the samples are in a byte-aligned position (a bit position divisible by 8) and as such the pgroup is a very software unfriendly format.

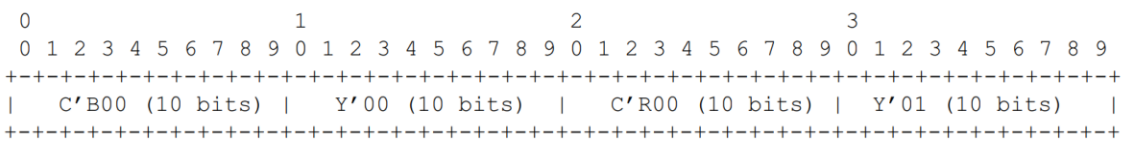

Figure 1 – YUV 4:2:2 pgroup

SOLVING THE PROBLEMS

High Data Rates

As the IT industry uses much higher bandwidths than broadcast in areas such as big-data and streaming, kernel-bypass has taken over as a technique to overcome performance improvements in OSs. As the name suggests it allows the programmer to bypass the OS and read or write data directly to the Network Interface Card (NIC). Examples of which include DPDK, Netmap and Registered IO (Windows). This allows for video data to be directly rendered to the NIC, in a similar fashion to the way a video player renders directly to the Graphics Processing Unit (GPU). In addition to the costly system calls required, the OS performs a number of costly memory copies before the pixel data reaches the network card.

The netmap framework was chosen as the kernel bypass mechanism as it had a simple but powerful Application Programming Interface (API) that could be iterated on quickly as well as being vendor neutral. As explained by Rizzo (2), there are three main gains that netmap provides. It reduced per-packet system calls by batching packets such that per-packet overhead became small, it reduced dynamic memory allocations by using static allocations and it also removed costly memory copies. In addition netmap also supported partial kernel-bypass, which allowed using only certain queues of the NIC for acceleration. This means that the OS could keep most of the NIC queues for its purposes, e.g for PTP, ping, ARP etc. As a naïve implementation was not implemented, we estimate the performance improvement owing to kernel bypass was 10-20x.

There are of course complexities to using kernel bypass in that the application, not the OS, is responsible for creating packets. This means that Ethernet, IP, UDP headers must be created by the application. For unicast, this also requires implementing ARP. As the OS is bypassed, features such as firewalling, routing etc. are not available.

Pixel Packing

Even in a “fast” language like C, converting pixels to/from the 2110 pgroup format is a slow operation. It is also often the case that incoming data is natively 8-bit (such as when decoded from an MPEG Transport Stream) and so in a naïve implemention this would require a conversion to 10-bit pixels, a double step. Of course, there is a substantial speed improvement from going directly from 8-bit pixels to the pgroup format.

More importantly, however, is the use of Single Instruction Multiple Decision (SIMD) functions to accelerate conversion between pixel formats, of which there can be many to many. SIMD instructions allow multiple samples to be operated on a single instruction. With some careful function design, this means that numerous samples can be converted in a few instructions. This is in contrast to per-samples operations that would take place in C.

In our implementation, just for YUV 4:2:2 alone, there is a 5x5 matrix of pixel formats as well as a few other special cases (not shown):

- Planar 10b - main working format
- Planar 8b - preview quality
- UYVY 10b (16-bit aligned) - SDI datastream
- Apple v210 - some hardware
- Contiguous 10-bit - pgroup packing

We have painstakingly implemented by hand (not using any aids such as intrinsics) every combination of conversion such that every possible code path is accelerated. Measured on an Intel i7-6700TE processor with C code compiled with gcc (Ubuntu 7.5.0-3ubuntu1~18.04) 7.5.0, the speed difference of the SIMD vs C conversion of planar 4:2:2 10-bit to pgroup is 20 times faster and the corresponding 8-bit conversion is 21 times faster. This is a massive speed improvement over a basic implementation. Care has to be taken for some combinations of resolution and pixel formats which may not have the right alignment for certain SIMD implementations.

Coupled with kernel bypass, this allows for highly accelerated pixel conversions to/from the NIC memory directly, leading to an estimated overall speed improvement of 100-200x. As a naïve implementation was not implemented, we are not able to measure the exact speed improvement.

Frequency Lock

In a simple implementation (ignoring the complexities of gapped transport) it is required to send a packet approximately every 7-8us. A naïve implementation is theoretically possible using carefully tuned hardware such as those used by the High Frequency Trading (HFT) industry. However the HFT industry is usually performing relatively small amounts of processing on the data so is able to isolate specific CPU cores and has limited memory bandwidth usage. It is likely that the system scheduling delays and other jitter would make this approach with video data complex, if not impossible, as it would require a very tight busy loop.

Instead of doing this, the approach taken was the canonical approach proposed by Grace (3), where a hardware rate limiter is used to rate limit output packets (packet pacing). On the NICs used, initially Intel and later Mellanox, this rate limiter is per queue. The intended purpose of this rate limiter is to forcibly limit Virtual Machines (VMs), each of which would be allocated a queue, to a maximum data rate. But for broadcast this is useful because it allows for video flows of different rates (e.g. SD, HD 25fps, HD 29.97fps) on the same device as well as allowing for dynamic changes (discussed later).

However, on both Intel and Mellanox hardware there were significant challenges with using the rate limiter. Most obviously, the rate limiter on these devices is an integer, yet we have the requirement to send at a non-integer rate. As a result, a careful design of clock drift compensation algorithm is needed to make sure that video remains at the correct rate and that buffers do not under or overflow.

Furthermore, there were a number of practical problems with accuracy of the rate limiters on various NICs up to an error of 1000ppm. In practice, this meant that the measured output rate was slightly larger than the reported claimed rate. It was measured on the Intel NIC, that the rate was accurate when using a “nice” rate limit such as 1Gbps (note: SI

giga- prefix). But when using a rate from the broadcast industry that may not be a “nice” number, the rate was seen to be inaccurate. The drift compensation algorithm needed to adaptively measure this drift as it was unclear whether a logical relationship existed between the requested and actual output rate.

So far, all discussion has been made on a free-running clock but for ST 2110 compliance, frequency lock with PTP is required. Linux has a standard mechanism for locking the NIC clock to PTP. However, in spite of the corrections made above, the actual output rate also drifts relative to the PTP clock. It is likely that this is because there is an internal free-running clock in the hardware which is controlling the output pacing rate. In addition to the drift compensation above, an implementation needs to also compensate for the Internal Clock vs PTP drift. All of these drifts require continual measurement and compensation.

Phase Lock

So far, we have only discussed frequency lock, such that video frames are released at the same rate as the PTP clock. But ST 2110 requires us to maintain phase lock N frame durations after the PTP epoch, essentially releasing the first packet at this time (ignoring the gap). The approach we have taken is to start transmitting, wait and measure how far off we are from the desired transmission point and to drift forward to the next “tick”. At its limit this is either perfect from the beginning (unlikely) or in the worst case requires just under a frame period of drift. Extreme care must be taken at the limiting value to not drift too quickly such that the other drift compensation above breaks and causes a buffer underflow. This is a coupled problem, as we care about frequency lock, phase lock and packet bursts. If we change one of these variables, the other two also change and so we must continually measure and compensate for any of these variables being out. In addition, at higher frame-rates, the reaction time must be faster, further complicating the problem.

It is also worth bearing in mind that the time between the act of transmitting in software vs the packet actually reaching the wire is unknown but the algorithm is also innately compensating for this too (likely a normally distributed value with a low standard deviation). There are methods for measuring this time but this is largely an academic exercise.

It was noted that some receivers could not tolerate a stream sent initially out of phase and then gradually drifted into phase, especially when the drift value to the epoch was over half a frame period. It is postulated that these receivers are generating output based on packet arrival times and outputting data at the correct frequency but wrong initial phase. As the phase drifts to the correct value, the receiver’s buffer underflows and it is unable to resync to the correct phase.

Minding The Gap

The gap for the Vertical Blanking also causes challenges as in a software application it is practically impossible to stop transmitting and start transmitting in the timescale of 20-40us. Using the packet pacing method mentioned does not allow for this at all as it requires a constant input of data. The only solution to this problem is to use the gap to put other traffic on the network such as audio or ancillary data. With enough care it is possible to create a clean edge that is compliant with 2110-21.

2022-7 Packet Output Skew

In a software application it is generally impossible to call a function that can act on both network ports at the same time. And it does not yet seem possible to send one packet and have the NIC duplicate the packet and rewrite the headers. As a result there is a substantial delay between sending packets on one NIC and sending packets on the other NIC. Depending on the vendor/driver of the NIC, this can range from 60us to 500us, well outside of the desired goal of sending packets on both links at the same time. Unsurprisingly, the answer is to implement yet another drift compensation algorithm that measures the skew between links and manually delays the stream that is delayed. Note, that there are now numerous drift compensation algorithms that have the potential to compete with each other.

There are further problems with handling 2022-7 skew. Kernel bypass frameworks such as netmap operate a transmit buffer, usually in the order of thousands of packets. However, if the switch port is disabled, simulating a switch failure, the buffer on the stopped port contains stale data that can't be transmitted. When the port is reactivated, this buffer with stale data must be emptied. But it leads to the problem where the second port's buffer cannot just be automatically refilled to the same level as the first as the first is dynamically moving as well. Therefore, the 2022-7 skew correction algorithm must also account for this.

Multiple Flows Of Different Rates

In the author's product it is very likely that the output flows may be of different resolutions and frame-rates and this is something not known at any given time. These could be over a dozen different flows with theoretically every combination of possible resolution and framerate, and thus output bitrate. Each of these flows do not exist in their own vacuum, and the numerous drift compensation algorithms on each one need to make sure they don't interact with the other flows. For strict compliance, it is likely that a large number of combinations would need to be checked to make sure there are no interactions between the drift compensation algorithm.

Audio

Initially audio seemed like a simple problem compared to the complexity of video. It is relatively low bitrate in the tens of Mbit/s and so it was thought the complexities of kernel bypass could be avoided. In the 125us profile this led to numerous problems with system call overhead and buffer referencing and unreferencing (discussed more below). Initially a very high CPU usage was seen in the audio thread and so this was analysed using the Flame Graph tool, a common tool used in the IT industry and created by a Netflix engineer. This is shown in Figure 2:

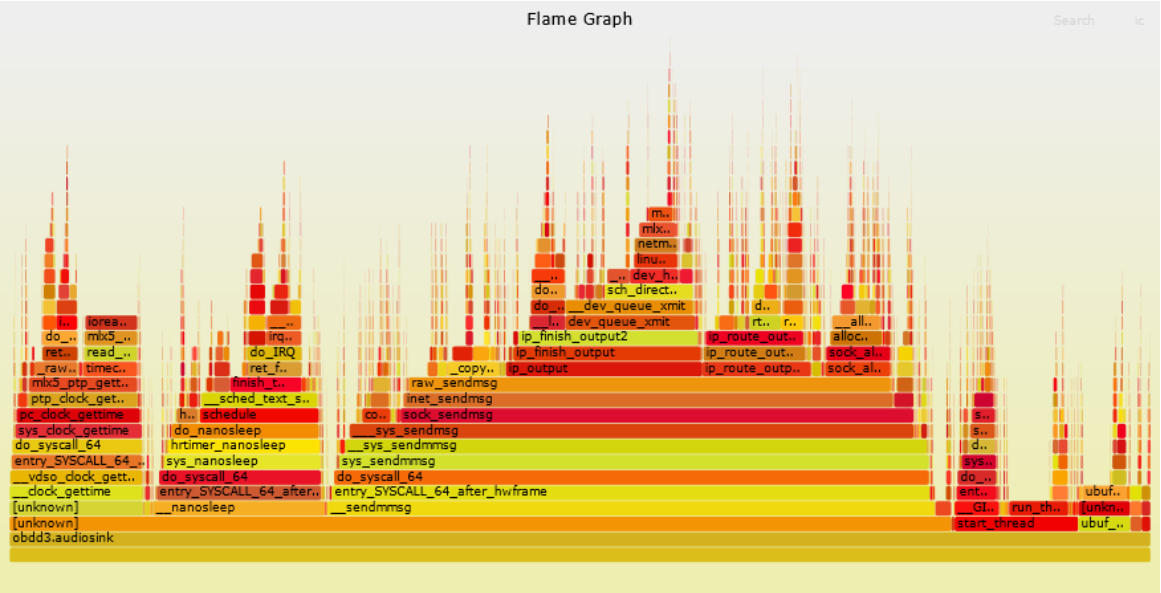

Figure 2 – Audio thread flame graph

Audio ran in a tight busy loop that slept every 125us so the existence of PTP and sleep functions can be ignored. Usually, analysis would show that the top of the flame would be wide and so this would be a function that could be easily optimised. However, it can be seen the flames are very thin and there is nothing that could be optimised. At large numbers of flows the system call overhead of audio packets being sent 8000 times per second was very high. Thus, after several months of attempted optimisation it was reluctantly decided that audio would also use kernel bypass alongside the video.

Programming Paradigms

It is worth commenting that owing to the tight software timescales involved that many of today's fashionable programming languages that have garbage collection that could take tens of milliseconds of runtime cannot be used. This timescale is likely too long to function optimally.

The other main challenge is that the complexity of the application does not favour a clean, modular software design. It is often the case that modules have to perform numerous different operations (e.g pixel processing, timing operations etc in the same loop). In particular the use of high packet rates for video and audio precludes the use of reference counting of small buffers. Modern multithreaded applications make extensive use of reference counting, however, the cost of referencing and unreferencing hundreds of thousands of times per second is very high. This further precludes a clean, modular design.

As a result of this the author has had to lead numerous substantial product redesigns in order to solve the problems described in this paper. Unlike in the broadcast industry, there is a strong focus in the IT industry to not accumulate technical debt, but this has proven to be very difficult to stop in this case. It is why the addition of Uncompressed IP functionality is comparable to making an electric car, it might seem obvious to bolt on an electric engine

to an existing vehicle (as many manufacturers do) but this causes numerous problems such that redesigning the entire vehicle is often easier.

Other Issues

There are other practical challenges that this paper does not cover such as internal multicast routing. If a receiver needs to receive a stream that it itself is sending, the switch will not return a copy of the same stream (in contrast to an SDI router). Therefore, it is vital that an application sending and receiving 2110 streams at the same time is able to perform internal routing, a process that is very nontrivial.

As has been seen, the use of packet pacing has proven to be extremely complex, as we have to use a sophisticated algorithm to compensate for its problems. It is likely that in the long-run we will move to a system where we just send line rate across the NIC and multiplex ourselves. This has its own complexities but in its totality is less complex than the complexity of using packet pacing.

INTEROPERABILITY

Numerous problems were seen with interoperability of third-party receivers. The most time consuming was discovering that a certain receiver required a source port to match its destination port. This is something nontrivial to do on standard OSs and completely incorrect behaviour. It was also apparent that numerous receivers were not using Real Time Protocol (RTP) timestamps correctly, but instead just using packet arrival times (more like SDI). This was easily seen by setting timestamps to nonsense values but seeing said devices work as normal. This is highly problematic behaviour, contrary to the principles of IP video. There were also often basic issues such as not verifying IP header checksums. Some receivers were also only able to handle a single audio flow and many were limited to only 8 channels.

CONCLUSIONS

It has taken nearly five years of substantial engineering effort to reach a position where we can generate production-ready 2110 video. This has often used novel and creative approaches to comply with the very tight requirements. It is worth pointing out this is not a normal software application; it is a very sophisticated application with demanding hardware requirements that will never be replicable in everyday software applications. To continue with the car analogy, it is a Formula 1 car vs a road car. This will almost certainly make software-based ST 2110 practical for only the tiniest proportion of developers willing to spend years developing such a complex solution as well as using specially-tuned IT hardware, far from being a plug-and-play solution.

As discussed in the introduction, it is important that the broadcast industry moves away from fixed-function hardware to IT/cloud-centric production that its FAANG competitors such as Netflix are used to. However, 2110 is not the solution to this problem, it requires extraordinarily sophisticated techniques to implement. It is hoped that future broadcast standards are more IT-centric since at the moment the industry lacks technical solutions for live production with the same flexibility as the workflows that FAANG use internally. It is disappointing to have a standard based around very complex timing requirements entirely for historical reasons. That said, this project has been a massively interesting engineering challenge, pushing IT hardware right to its limits. It does appear that the move to the cloud is leading to more practical approaches to uncompressed video production, e.g Amazon CDI (Cloud Digital Interface) and the future unnamed GCCG (Ground Cloud - Cloud

Ground) protocol which bodes well for the future but arguably too late as viewers move quickly to OTT services from linear broadcast.

REFERENCES

1. Kostiukevych, Vermost, Ferreira, 2017. An Open-source Software Toolkit for Professional Media over IP (ST 2110 and more)
2. Rizzo, 2012 USENIX ATC 2012. netmap: a novel framework for fast packet I/O
3. Grace, 2016. UK Network Operators Forum 2016, IP Networks in the TV Studio: Recent work by BBC R&D

ACKNOWLEDGEMENTS

The author would like to thank his colleagues for actually writing the code and delivering the huge leaps needed for the most complex thing ever built in his career. It was especially pleasing to watch numerous live services using this development whilst writing this paper and thanks must also go to the customers who put their faith in us to deliver this.