Stream Annotations for Energy Trade-offs in a Video Decoder for Multimedia Applications *

Radu Cornea Alex Nicolau Nikil Dutt
Donald Bren School of Information and Computer Science
University of California, Irvine, CA 92697-3425
{radu,nicolau,dutt}@ics.uci.edu

CECS Technical Report #06-09

May 2006

Abstract

Recent applications for distributed mobile devices, including multimedia video/audio streaming, typically process streams of incoming data in a regular, predictable way. Profiling shows that the runtime behavior of these applications can be accurately predicted most of the time by analyzing the data to be processed and annotating the stream with information collected. We introduce an annotation-based approach to power-quality trade-offs and demonstrate its application on CPU frequency scaling during video decoding, for an improved user experience on portable devices. Our experimental results show that savings of up to 50% can be achieved for the processor power consumption on typical PDAs during MPEG video decoding.

*This work was partially supported by NSF award ACI-0204028.

Contents

List of Figures

List of Tables

1 Introduction

Recent developments in PDAs and cellphones have made possible a range of new applications for mobile devices, such as multimedia streaming (audio and video), online gaming, voice over IP. All these applications have one thing in common: they process incoming streaming data using known algorithms, in a repetitive fashion. The main power consuming components of a handheld device are the CPU, display and network interface. Running multimedia applications further aggravates the situation, as these programs are known to be both CPU-intensive and to require higher network bandwidth.

Research has shown that most streaming applications follow a regular pattern of phases during their execution. They start with an initiation phase followed by a repeated main loop that contains the main phases of the algorithm and ends with a finalization phase. In case of streaming applications (video, audio or gaming), the algorithms are typically composed of different filters (e.g. motion estimation, quantization, DCT), which are applied on the incoming data (a sequence of frames in the case of multimedia streaming) in a regular, repetitive way. This regular behavior is confirmed by recent research [10].

This regular behavior is only affected by the nature of data to be processed. Data stream and its characteristics introduce a degree of variation within the execution of these applications, like timings, load, power consumption and other metrics ([8]). Therefore, by analyzing the data either off-line or online before it is processed allows us to gain knowledge its content which makes an entire range of data-aware optimizations possible at the data stream level.

By profiling incoming data and annotating it with the information collected, we can predict the behavior of the application and use this knowledge for improving either the performance or energy consumption. These optimizations are especially useful in the case of mobile devices where battery capacity is a limiting factor. For these applications, annotations can substantially improve battery life of mobile devices and therefore increase user experience.

Moreover, data annotation and profiling could be integrated in the future with application profiling (execution phases), for a unified view of the application's behavior during runtime and further improvements in both power savings and performance.

In this paper we focus on data annotations for multimedia streaming, more specifically MPEG decoding. We show that pre-analysis of video streams allows for more aggressive DVS techniques, while still meeting timing constraints. These translate into power savings of up to 50% as compared to a no DVS approach or up to 30-40% as compared to a simple DVS based on worst case execution time.

2 Data Annotation

2.1 General Considerations

Data annotation is the process of analyzing a stream of data and supplementing it with a summary of the information collected, information which will be use later for data-aware optimizations to be performed at run-time. Annotations typically focus on patterns or trends in the data that can be exploited later for power or performance benefits.

The steps for annotating data stream are:

- Profile data stream (offline or online by a proxy server)
- Embed profiling information into data stream (annotate)
- Use annotations at runtime for optimizations

The annotations can be performed either statically (offline annotation/profiling for example in the case of media serving, where information is preprocessed and saved on media servers) or dynamically (in case of live streams, through on-the fly annotation done at an intermediary proxy node). In this work, we assume that the annotations we are added offline, through profiling of an extensive data set, representative for the application domain. We focus on multimedia streaming and annotations that are relevant to this domain.

Annotations can be used either at the client side, for optimal playing, or at a proxy node, which performs various operations on the data stream to adapt it to client's capabilities (transcoding, etc.). These annotations may proved useful in estimating the required bandwidth for communication, estimating computation (for task balancing between transcoding at the proxy vs the client), or applying more aggressive QoS trade-offs based on image content.

The advantage of annotating the data in advance is two-fold. First, there is no overhead for doing all the work at runtime. Second, because the information is available even before decoding the data, more optimizations are possible, which would not be possible with a runtime analysis. For example, annotation in the MPEG stream allow optimizations at the network level because the information is stored in the MPEG header itself. Other optimizations (like DVS) can be applied earlier, even before decoding is finished, because the annotations are immediately available from the data stream. Without annotations, the client application would first need to first decode the data and analyze it or use some history based predictions, with a more limited knowledge because only a small window of data can be analyzed at one time.

2.2 Applications for Annotations

The idea of annotation is not new. Annotations have been previously used in other domains. For example, in compilers, annotations are sometimes used for maintaining as much from the original semantics as possible. Annotations are also used by programmers in their code

to provide hints to the compiler [7]. For example, register assignment for variable in C falls in this category. Other examples include the use of 'pragma' directives (C, C++, Ada).

Similarly, we see the process of annotating the data stream as either automated (performed statically, by an analysis step) or under user supervision, where the user can, for example, specify which parts or objects of the video stream are more important and should be best preserved in a power-quality trade-off.

Multimedia streaming applications can be analyzed at different abstraction layers: application, middleware/network, OS, hardware. Each of these layers benefits from additional information on the data to be processed.

The video streams provide a perfect opportunity to apply data annotations. Video streaming and video playing are especially strenuous for the processor and network subsystem, where most of the possible optimization are located. Therefore, annotations can prove helpful at both these levels by providing more information on the stream's content, trends and patterns.

At network level, annotations help streamlining or optimizing the transmission based on the content (i.e. encoding parameters directly translate into communication bandwidth). Network interfaces typically have low power modes, which can be exploited, with information on the variation in the transfer rates.

Most operating system for mobile devices perform various power slowdown/shutdown optimizations to conserve battery energy. Without information on the data to be processed, the OS takes a conservative approach and considers the worst case when applying these power saving measures. Data annotation improves the decisions that are made at this level, resulting in larger power savings and longer battery life.

At hardware level, annotations may help reconfiguring the hardware to perfectly match the requirements for the data to be processed (e.g. cache size and associativity can be tuned based on the encoding parameters of the video stream).

In this paper we look at data annotation in the context of MPEG video decoding. The idea is to estimate processor requirements for decoding video frames. In general, frames in a MPEG stream have different distribution of I, P and B macroblocks. As each macroblock type require a certain amount of computation, there are huge differences between decoding times for different frames. The annotation would store the processing requirements for each individual frame through a high level estimation based on macroblock distribution and frame size.

The analysis is performed at frame level, as each frame has different macroblock distribution. The estimation can be applied later in the decoding step for a more aggressive, data-aware DVS (for power savings, especially for B frames, which require less computation).

There has been previous work for estimating frame decoding time for video streams. However, in these cases, the decoding time was known only after the frame was extracted completely from the video stream, which delays the time when power savings schemes can be used. In our approach, frame complexity information is available right after the frame is received, since the information is precomputed and stored in the MPEG header. This allows other optimizations at network/OS level, not possible with the previous techniques.

2.3 System architecture

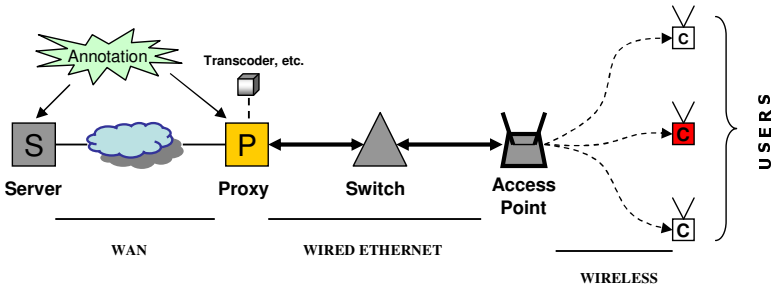

Figure 1: System model

We assume the distributed system model depicted in Fig. 1. The system entities include a multimedia server, a proxy node that can perform various optimizations on the stream (e.g. transcoding), the users with low-power wireless devices and other network equipment along the way. The multimedia servers store media content and stream videos to clients upon requests issued by the users on their handheld devices. The communication between the handheld device and the servers can be routed through a proxy server – a high-end machine that has the ability to process the video stream in real-time. The proxy node can also perform in-line profiling/annotation for real-time video streaming (videoconferencing is an example of such an application).

The annotations can be generated and stored in the video stream (headers) at either the server side or the proxy node, therefore saving the client of any additional work. Another option is to send the annotations over a separate control channel.

3 MPEG Background

A typical MPEG video stream is composed of sequences of frames (I, P, B), some of which are entirely encoded in the stream (intracoded), some which are predicted based on either previous, subsequent frames or a combination(Fig. 2).

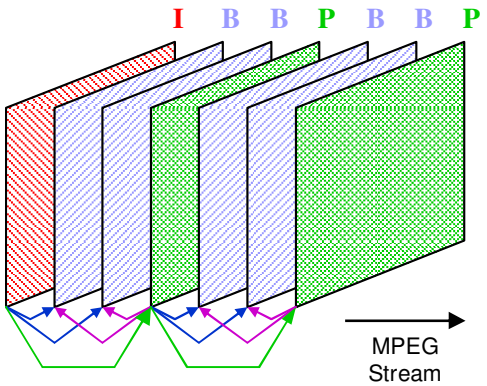

Figure 2: Frames in a MPEG stream

The 3 types of frames in a MPEG video stream are composed of macroblocks (16x16 pixels) as follows:

- I frames: only I (intracoded) macroblocks
- P frames: I and P (forward predicted) macroblocks (P predominate)
- B frames: I, P and B (backward predicted and interpolated forward-backward) macroblocks (B predominate, followed by P and I)

Macroblock distribution varies with the amount of motion in video clip.

If we plot the distribution of macroblock types in a video clip we observe that the relative percentage of I, P, B macroblocks varies depending on the nature of the video stream and the amount of motion in it (e.g. Fig.3). Therefore, for accurate prediction we need to separately profile video clips from different categories (clips with dynamic action vs mostly static, news type).

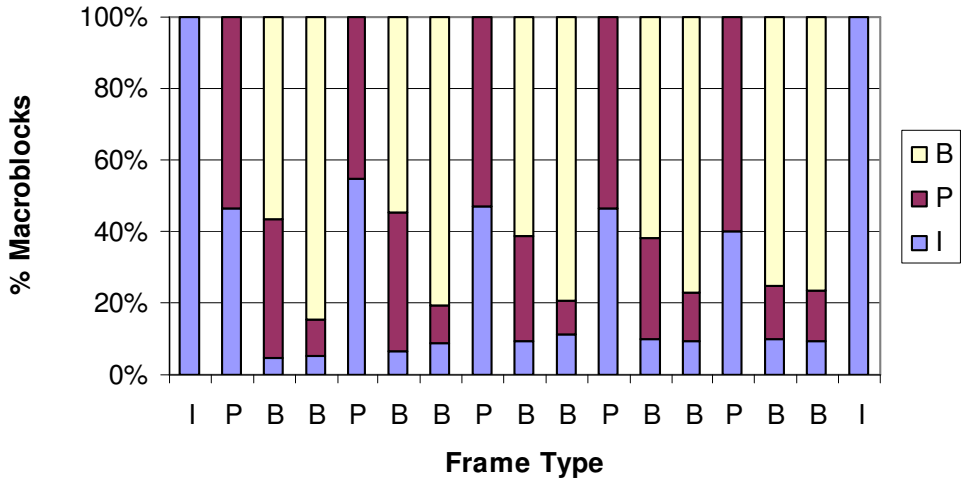

Figure 3: Macroblock Distribution(action clip)

Because I macroblocks are completely intracoded, they require IDCT for decoding and are typically much larger than the other macroblocks. On the other hand, P and B macroblocks are predicted from other frames. Therefore they are smaller in size and require both motion compensation and IDCT. Normally, IDCT is more computationally intensive than motion compensation and depends on the size of the block to be processed.

In MPEG, the processing requirements for decoding a macroblock is typically such that $\boldsymbol{P_i > P_p > P_b}$. The distribution of different macroblocks in a frame allows us to roughly estimate the required time for decoding the entire frame (which always contains a constant number of macroblocks, depending on its pixel size). As each frame can potentially have a different distribution, we store this annotation at frame level.

There is a clear delimitation between decoding times for different frame types (I/P/B), as can be seen in Fig. 4 where we show the distribution of frame decoding times during the

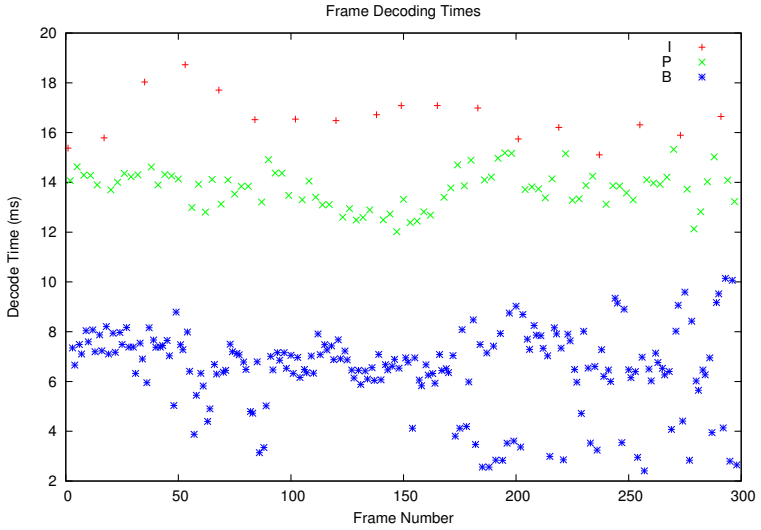

Figure 4: Frame Decoding Times (action clip)

execution of a video clip (action). Moreover, the relative variation of the points follows the patterns of activity (motion) in the video clips.

In our approach, the annotations are used to help estimating the computation needed for decoding frame (decoding time). The prediction is then applied in a power management scheme (DVS for slowing down the processor on a frame by frame basis). Nevertheless, there are other levels in an application where annotations on the incoming data could be useful (one such example is the network subsystem).

In case of MPEG decoding, saving are mainly due to the inequality between decoding I/P versus B frames, where the CPU is idle for longer periods of time, allowing more aggressive power management schemes.

4 Annotation Based DVS

In the previous section we showed that video decoding has a very regular and predictive behavior most of the time. Next, we use these observation and attempt to use it in a prediction scheme based on annotation data, which allows us to apply more aggressive power saving schemes.

4.1 Decoding Time Estimation

If we plot decoding time for a frame as a function of the frame size, we observe a strong correlation between the two in practically all video clips. For example for the same video clip as in Fig. 4 the plot is shown in Fig. 5.

In order to estimate decoding time based on frame size $\boldsymbol{Dec(f) = F(size(f))}$ we apply regression fitting algorithms for defining a function between frame sizes and decoding times.

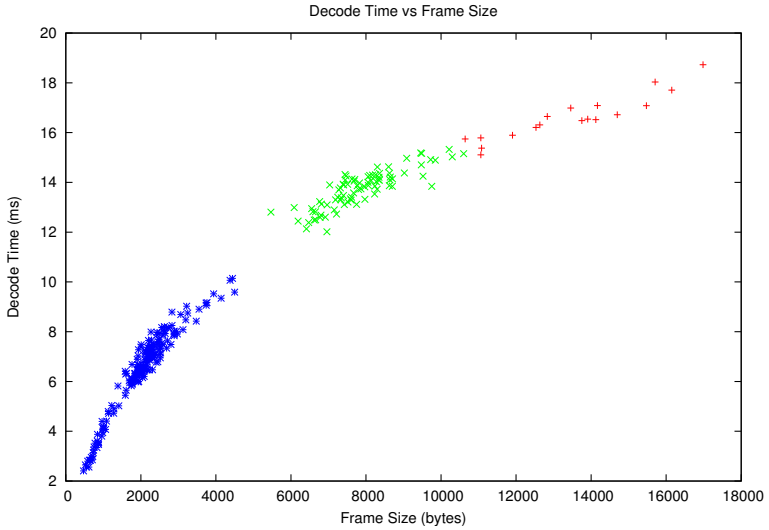

Figure 5: Decode Time Variation with Frame Size

Both linear and and quadratic regressions are tested against profiled data. Results are evaluated in terms of estimation errors.

Next, we devise an estimation heuristic for frame decoding time and use it to control a DVS scheme.

4.2 DVS Control

In this section we describe our prediction algorithm and dynamic frequency scaling. Since current iPaq PDAs do not support voltage scaling, we limit our results to frequency scaling with the mention that voltage scaling would even further improve the results. Therefore in the reminder of the paper the term DVS will refer to "frequency scaling" only.

In general, the power consumed by a CMOS circuit can be approximated through the formula $\boldsymbol{P = CfV_{dd}^2}$, where $\boldsymbol{C}$ is the switching capacitance of the circuit, $\boldsymbol{f}$ is the clock frequency and $\boldsymbol{V_{dd}}$ is the supply voltage level.

Our technique slows down the processor (saving power) during each frame decode to the lowest frequency that still allows it to finish before the frame deadline.

Fig. 6 shows the frequency and time involved in the decoding of a single frame. Originally, the processor runs at full frequency ($\boldsymbol{f_o}$) and the frame is decoded in $\boldsymbol{t_o}$ time. The deadline for decoding a frame is given by $\boldsymbol{t_d}$ and depends on the frame rate at which the video is encoded.

Based on the size of the frame, we use an approximation function $\boldsymbol{F_{estim}}$ and predict the decoding time as

$\boldsymbol{t_{estim} = F_{estim}(s)}$

Because of estimation errors, the computed decode time can vary from the actual decoding time by a ε value. The smaller ε, the better our prediction is:

$\boldsymbol{t_{estim} - \varepsilon < t_{orig} < t_{estim} + \varepsilon}$

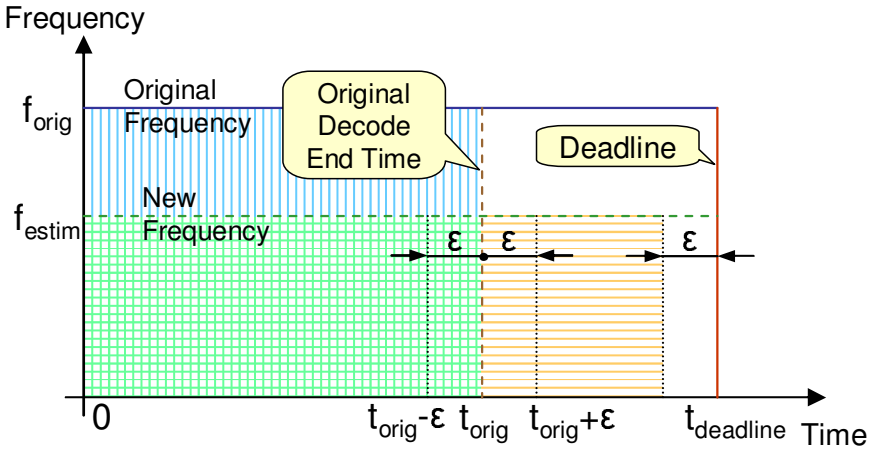

Figure 6: Frequency Scaling for Frame Decoding

This allows us to run the processor at a lower frequency given by the equation:
$(t_{estim} + \varepsilon) f_{orig} = t_{deadline} f_{estim}$
Original power consumption is determined by the original frequency:
$P_{orig} = C f_{orig} V_{dd}^2$
Power consumption using the predictive approach becomes:
$P_{estim} = C f_{estim} V_{dd}^2$

To evaluate the power savings, we compare our approach to the original case, where no DVS technique is in effect and the processor runs at full power all the time. We also compare our technique against a simple heuristic, which we call "simple WCET DVS". This heuristic assumes a constant processor frequency for the duration of the entire clip, chosen so that all frames can be decoded before the deadline (slows down the processor such that the worst case decoding time is still before the deadline). Power consumption in this case is:

$P_{wcet} = C f_{wcet} V_{dd}^2$, where the frequency f_{wcet} is determined from:

$t_{max} f_{orig} = t_{deadline} f_{wcet}$, with t_{max} bounding the largest (WCET) decode time during the entire clip (or a large window in the execution).

The "simple WCET heuristic" is similar to what a current DVS-capable device would actually perform and does not take advantage of the time difference between decoding different frames. In contrast, our estimation based approach tries to compute (within a reasonable error) the decode time for each individual frame based on the annotations stored in its header. An illustration of these three cases (no DVS, "simple WCET DVS" and estimation-based DVS) is depicted in Fig. 7.

The energy savings in our technique come from the difference between the original CPU power (or WCET power) and our estimation-based power, which is always lower or equal to the original. We lower the CPU operating frequency by taking advantage of the slack time in the decoding phase, which we more accurately estimate through annotations and profiling.

We estimate the CPU power savings of our annotation-based technique over both the original power consumption, with no DVS scheme applied, and the power consumption in the "simple WCET DVS" case. The results are computed for both offline annotation (streaming

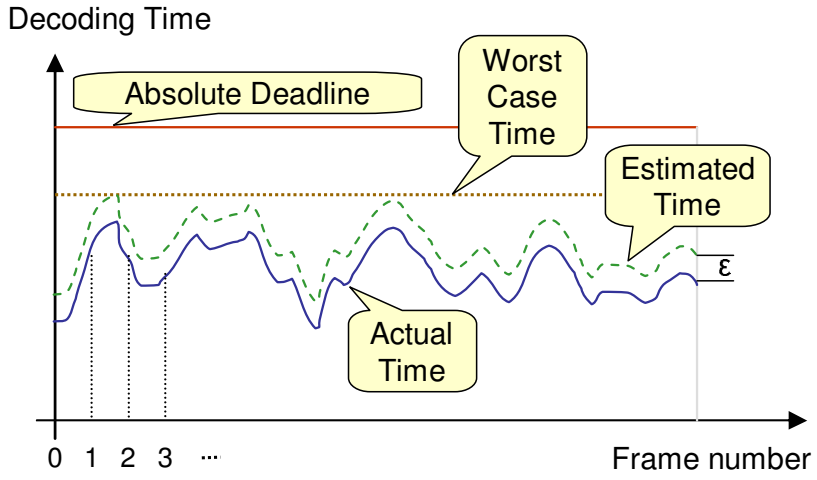

Figure 7: Different DVS heuristics

of stored material) and real-time video streaming (e.g. videoconferencing).

Annotations can be stored to the stream in different ways: inside empty packets in the MPEG stream or in a dedicated control channel. In our experiment we assume that only the parameters of the fitting curve are transmitted for each clip, which requires just a number of bytes per clip (the evaluation of the function is performed at runtime, being a very simple quadratic function of the frame size). Another possibility would be to precompute the estimated decode time and store it with each frame (saving even more processing time at runtime, but requiring an additional number of bytes in the stream, though a minimal overhead compared to the frame size overhead compared to the frame sizes).

5 Experimental Results

In the previous section we have shown that there is a strong correlation between frame sizes and decoding times in the case of video decoding algorithm. We have also demonstrated how in theory data annotations can be applied to video streaming. In particular we looked at how annotations help prediction when we perform dynamic voltage scaling during decoding phase.

We now apply our data annotation strategy in a real application of video decoding. First, we derive an estimation function by analyzing profiled data from a set of videos from with similar encoding parameters. Then, we use the resulted function to predict the decoding time for each frame in the MPEG stream and we apply DVS in order to fully take advantage of the remaining slack time by slowing down the processor while meeting the deadlines.

5.1 Experimental Setup

We used Sim-Panalyzer[1], a power architecture simulator, based on SimpleScalar. Sim-Panalyzer models an ARM processor architecture and performs cycle accurate simulation.

At the end of execution the simulator reports a number of statistics, including power consumption by the internal units of the processor.

The simulated architecture is a StrongArm processor (200MHz), found in typical iPaq PDAs. The overhead for switching frequency is in the order of microseconds for a StrongArm processor, as opposed to tens of milliseconds for a frame decoding time and therefore is assumed negligible in our experiments. For measuring decoding times, we run the MPEG decoder, which is part of Berkeley MPEG Tools. The predictor is written in Octave.

As data input, we use video clips of 10 second each (300 frames) with a framerate of 30fps. Eight of the clips are encoded in a CIF (352x288) pixel format, and are taken from the multimedia community (akiyo, coastguard, container, foreman, hall, mobile, news, silent); they cover the entire range from very still to very dynamic. The other two clips are encoded in a 320x240 format (action, news).

Our experiments assume a frame-based DVS. Other approaches are possible too: for example DVS applied to a group of frames. In this latter case, buffers are required between the decoder and the display, to smoothen the variation between decoding times and avoid frame delays. Also, the power savings would be slightly lower, but the technique could be applied to even more devices (those with less DVS steps, more overhead).

5.2 Frame Decoding Time

We start by profiling the video clips in our simulator and generating the information required for data annotation. The next step is to derive the approximation (estimation) function (decode time vs frame size).

For curve fitting, we use linear and quadratic functions. For a linear regression we try to approximate the decoding time as a linear function:

$\boldsymbol{f(x) = a * x + b}$

For a quadratic regression:

$\boldsymbol{f(x) = a * x^2 + b * x + c}$

$\boldsymbol{a, b}$ and $\boldsymbol{c}$ are determined using the "least square error" method. To evaluate the quality of the results, we compute the coefficient of determination ($\boldsymbol{R^2}$ value) for the fit, which indicates the percent of the variation in the data to be expected. The closer that $\boldsymbol{R^2}$ is to 1, the better the model "fits" the data.

We also compute the maximum residual value, which bounds the maximum variation from the predicted value we can expect ($\boldsymbol{\varepsilon}$ in our discussion above). This bound limit is used for assuring that almost no deadlines are missed, even in the presence of variations, thereby maximizing quality.

In our experiments we computed this threshold such that virtually no deadlines were missed. If the user can tolerate some quality degradation (frame loss), we could lower the threshold even further, significantly improving power savings. This is the trade-off between the quality of service delivered and the maximum power savings possible.

Example:

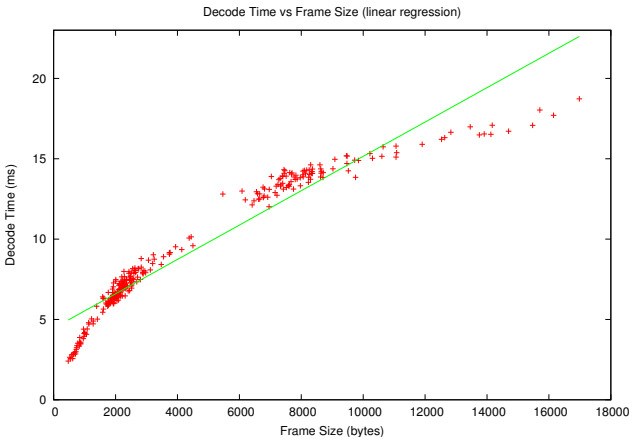

Figure 8: Linear Regression Fitting (action clip)

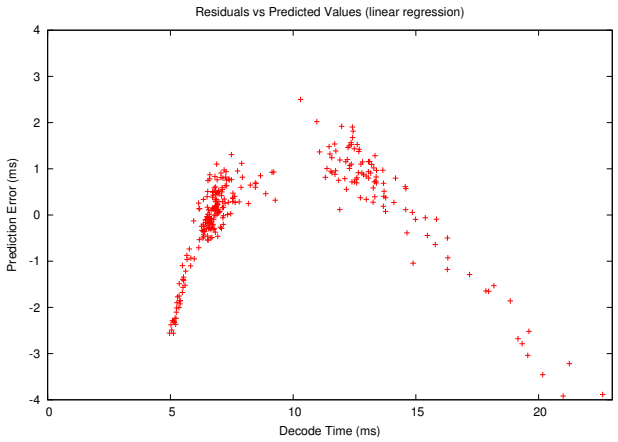

Figure 9: Errors for Linear Fitting (action clip)

For a video clip (action), Fig. 8 shows a linear regression model, while Fig. 9 displays the errors with this model (spread between -3.7 and 3).

The results for the linear regression model are:

$\boldsymbol{a = 0 \quad b = 4.46}$

$\boldsymbol{R^2 = 0.92 \quad MaxResid = 4.02}$

For the quadratic model, Fig. 10 and Fig. 11 present the curve fitting and errors (between -1.5 and 1.5 in this case).

Absolute results:

$\boldsymbol{a = -6.7e-08 \quad b = 0.002 \quad c = 2.77}$

$\boldsymbol{R^2 = 0.98289 \quad MaxResid = 2.2647}$

As we can see, the quadratic model gives a better match both from $\boldsymbol{R^2}$ point of view (0.99 vs 0.95) and maximum residual value (1.5 vs 3.7). This tells us that the maximum error we can expect is around 1.5 ms, so we can plan the DVS slowdown accordingly, so as not to miss deadlines where possible.

A problem arise when we apply this technique to different video clips: one curve fitting cannot guarantee good results for all. Therefore, we separate I prediction from P/B, like in Fig. 13. (I macroblocks have different behavior than P and B)

By analyzing a large number of video clips from different domains we concluded that the best fit (best $\boldsymbol{R^2}$ value) is achieved when using the following two fitting curves: a linear function for I-frames, and a quadratic function for P and B-frames (example in Fig. 13). A single quadratic function would not capture the curve variations present in some of the clips between the different types of frames (see Fig. 12). The variation is introduced by the extra motion compensation step, which is present only in P and B-frames).

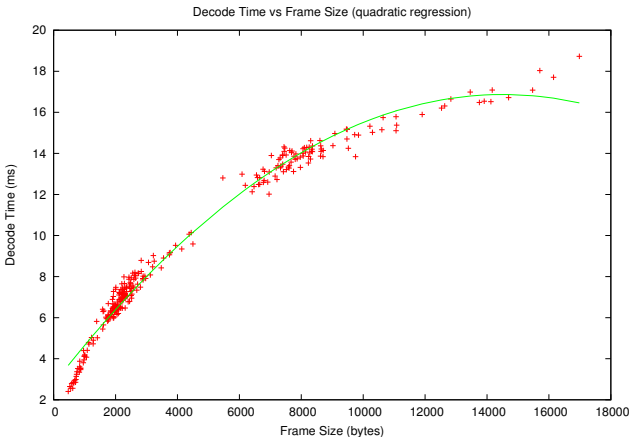

Figure 10: Quadratic Regression Fitting (action clip)

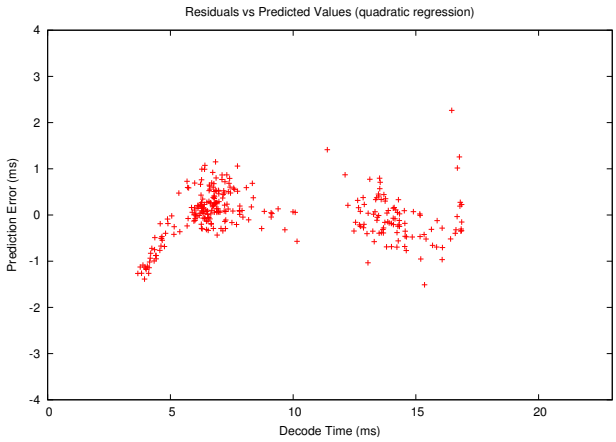

Figure 11: Errors for Quadratic Fitting (action clip)

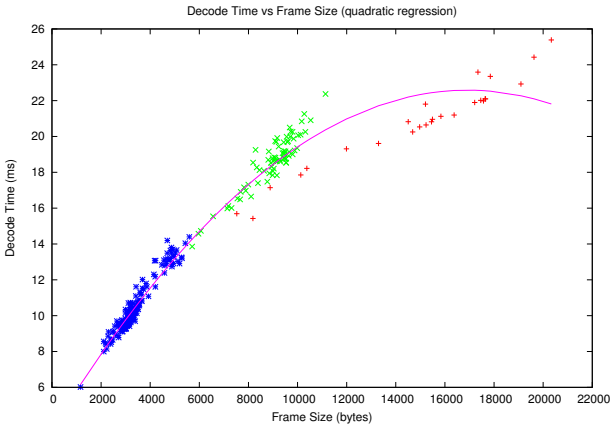

Figure 12: Fitting Curve ('foreman' CIF clip)

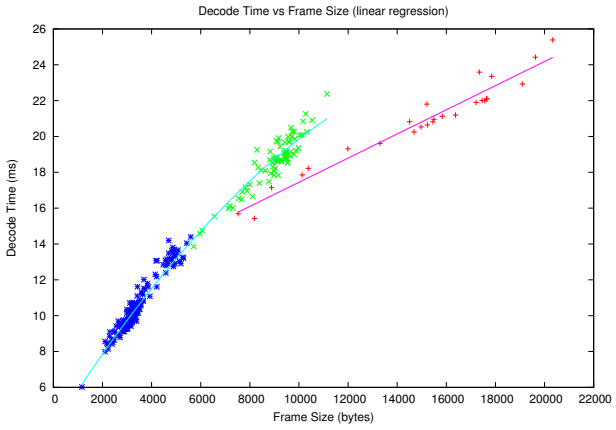

Figure 13: Fitting Curves ('foreman' CIF clip)

5.3 Power Savings

To simulate both an off-line scenario (stored media, annotations are performed off-line) and a real-time streaming (videoconferencing, annotations are performed in-line, by the proxy node) we experimented with two different setups:

(a) **Individual estimation**: the case of stored material
Because the profiling is performed off-line in this case, we applied the profiling and estimation steps individually on each clip (i.e. each clip is first profiled, annotated, then its own estimator is used for DVS). Results are shown in Table 1.

(b) **Global estimation**: simulates a real-time scenario
For an in-line annotation, we first profile globally all clips from the same domain and with similar encoding parameters and derive a single global estimator. Next, we run this estimator on each clip separately. This is a more realistic approach for live video

streams, where we may not want or be able to separately profile each clip, instead use a set of similar clips from the same video domain for the off-line profiling step. The results are presented in Table 2.

Video Clip	Savings Over no DVS	Savings Over Simple DVS
action	65.2033	44.2539
akiyo_cif	55.4909	48.5246
coastguard_cif	51.4923	36.118
container_cif	56.8692	50.9896
foreman_cif	50.5261	41.534
hall_cif	48.5585	32.8528
mobile_cif	52.6674	44.4279
news_cif	54.4522	45.244
news	59.3555	41.6503
silent_cif	56.562	52.5924

Table 1: Power Savings (individual)

Video Clip	Savings Over no DVS	Savings Over Simple DVS
action	49.4689	19.0467
news	52.6133	31.971
akiyo_cif	54.1925	47.023
coastguard_cif	51.3742	35.9625
container_cif	56.8142	50.927
foreman_cif	46.5873	36.8793
hall_cif	47.0726	30.9133
mobile_cif	51.7005	43.2927
news_cif	47.5397	36.9341
silent_cif	53.1341	48.8512

Table 2: Power Savings (global)

The power consumption results for both scenarios are presented in a graphical way in Fig. 14, against the original (no DVS) and the "simple WCET DVS" case. The numbers are normalized to the original power consumption (i.e. 100% means no DVS).

We observe that the power savings for the processor are up to 50% over no DVS and up to 40% over "simple WCET DVS" (which already performs much better that the original case), for the individual estimation. Even for the global estimation case where the savings

are slightly lower, the power gains are still high when compared to the other approaches. Our savings translate to around 15% for the entire device, with virtually no quality loss. More savings are possible if a lower quality of service is allowed.

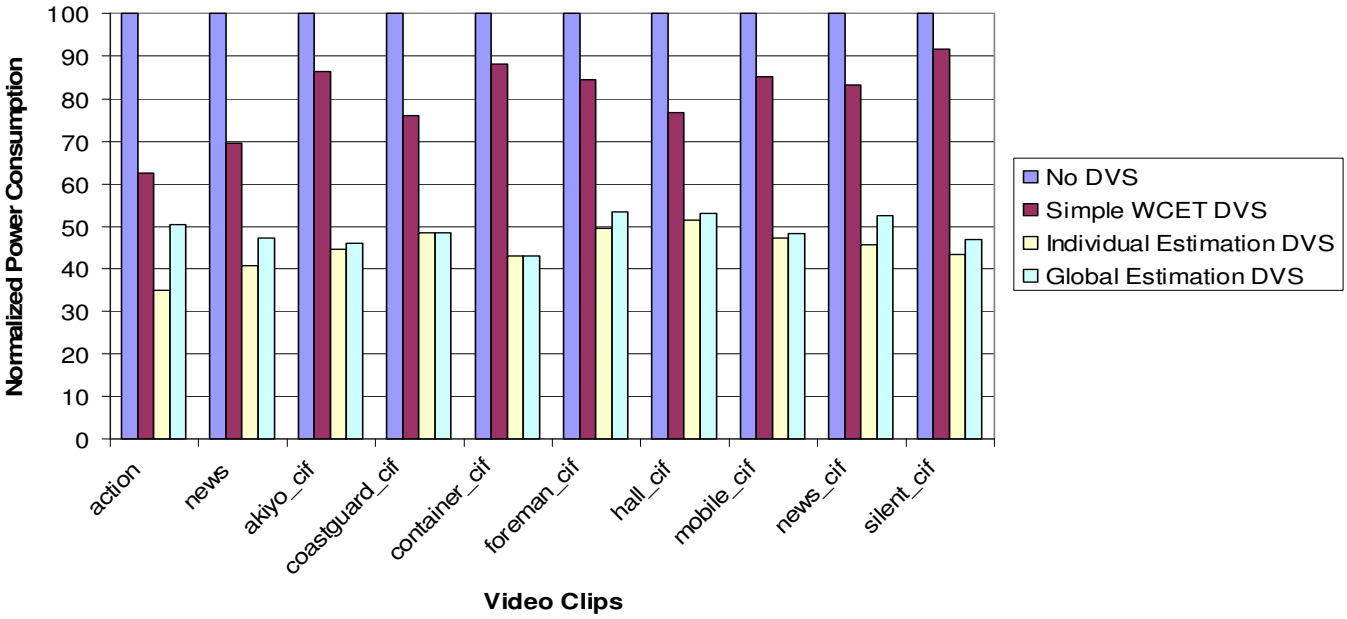

Figure 14: Normalized CPU power consumption results

Other results are presented in the Appendix.

6 Related Work

There has been recent work that looks at the processed data and tries to derive various heuristics for improving either communication or computation in streaming applications.

The Aspire research group studies various data-shaping for mobile multimedia communication. They profile and annotate still images for improving transmission over a wireless channel usage (bandwidth, latency). In [6] the image data is compressed according to dynamic conditions and requirements. Content adaptation is classified depending on time (static, dynamic), content (to determine optimal compression) and goals of technique or metrics (constrained bandwidth, display size, response time).

Chandra performs an informed quality aware transcoding in [3], based on image characteristics. He finds that a change in JPEG quality factor (compression metric controlled by quantization steps) directly corresponds to information quality lost. A prediction for computational overhead is applied, which approximates number of basic computation blocks based on image size, color depth and can predict output size for a particular transcoding.

In [4], the authors analyze the characteristics of images available on web sites (distribution of gif or jpg images, size, colors and quality). They clasify images in: bullets, lines,

icons, banners, trueimages based on heuristics and analyze various transcoding techniques for redugind image size (reducing spatial geometry or thumbnailing, reducing the number of unique colors, changing the image format or compression)

Avanish Tripathi and Mark Claypool study different ways to reduce bandwidth in network transmission in [11], by either temporal scaling (dropping frames), quality scaling (reducing quality of frames) or spatial scaling (changing the size of frames). The quality degradation is evaluated through an user study.

[5] presents a DVS technique for MPEG decoding to reduce energy consumption while at the same time maintaining the quality of service. The approach is to separate decoding time into a frame-dependent, which varies with the frame and a frame independent part, constant regardless of the frame. The independent part is used to compensate the error that can appear during the depending frame part.

Bavier et al in [2] present a set of experiments to measure the CPU processing required for decoding a MPEG frame in software. The algorithm predicts the number of cycles required for a given frame by constructing a linear model between frame type, size and time. The accuracy is within 25% of the actual decode times.

In [9], the authors describe a feedback based controller for video decoding that apply DVS for individual frames. The complexity of frames is estimated using a simple correlation between frame length and decoding time.

In contrast, our technique performs an off-line profiling and annotation, which allows us to more accurately estimate the frame decoding times with a minimal overhead at runtime.

7 Conclusions and Future Work

In this paper we presented a new approach toward optimization for multimedia streaming, using data annotation. We have shown how annotation can be used for predicting runtime behavior. We showe our experiments where we used data annotations for prediction based DVS.

Our results show good power savings when prediction is performed on similar video clips: up to 50% savings over no DVS and up to 40% over simple DVS (based on worst case assumption)

As future work, we plan to extend our algorithm to include dynamic prediction when the prediction error cannot be minimized with an offline approach. The dynamic algorithm would maintains a history window for future prediction. Another extension is for prediction on video clips with different encoding parameters(size, framerate, bitrate).

References

[1] Sim-panalyzer 2.0 reference manual. Technical report, University of Michigan, University of Colorado.

[2] Andy Bavier, Brady Montz, and Larry L. Peterson. Predicting mpeg execution times. Technical Report 97-15, University of Arizona, 1997.

[3] Surendar Chandra and Carla Schlatter Ellis. JPEG compression metric as a quality-aware image transcoding. In *USENIX Symposium on Internet Technologies and Systems*, 1999.

[4] Surendar Chandra, Ashish Gehani, Carla Schlatter Ellis, and Amin Vahdat. Transcoding characteristics of web images. In Martin Kienzle and Wu chi Feng, editors, *Multimedia Computing and Networking (MMCN'01)*, volume 4312, pages 135–149, San Jose, CA, jan 2001. SPIE - The International Society of Optical Engineering.

[5] Kihwan Choi, Karthik Dantu, Wei-Chung Cheng, and Massoud Pedram. Frame-based dynamic voltage and frequency scaling for a mpeg decoder. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design*, pages 732–737. ACM Press, 2002.

[6] D.G.Lee, D.Panigrahi, and S.Dey. Network-aware image data shaping for low-latency and energy-efficient data services over the palm wireless network. In *World Wireless Congress (3G Wireless)*, 2003.

[7] Samuel Z. Guyer and Calvin Lin. An annotation language for optimizing software libraries. In *Domain-Specific Languages*, pages 39–52, 1999.

[8] Christopher J. Hughes, Praful Kaul, Sarita V. Adve, Rohit Jain, Chanik Park, and Jayanth Srinivasan. Variability in the execution of multimedia applications and implications for architecture. In *International Conference on Computer Architecture*, pages 254–265, 2001.

[9] J. Pouwelse, K. Langendoen, R. Lagendijk, and H. Sips. Power-aware video decoding, 2001.

[10] T. Sherwood, S. Sair, and B. Calder. Phase tracking and prediction. In *International Symposium on Computer Architecture*, 2003.

[11] A. Tripathi and M. Claypool. Improving multimedia streaming with content-aware video scaling, 2001.

A Macroblock Distribution

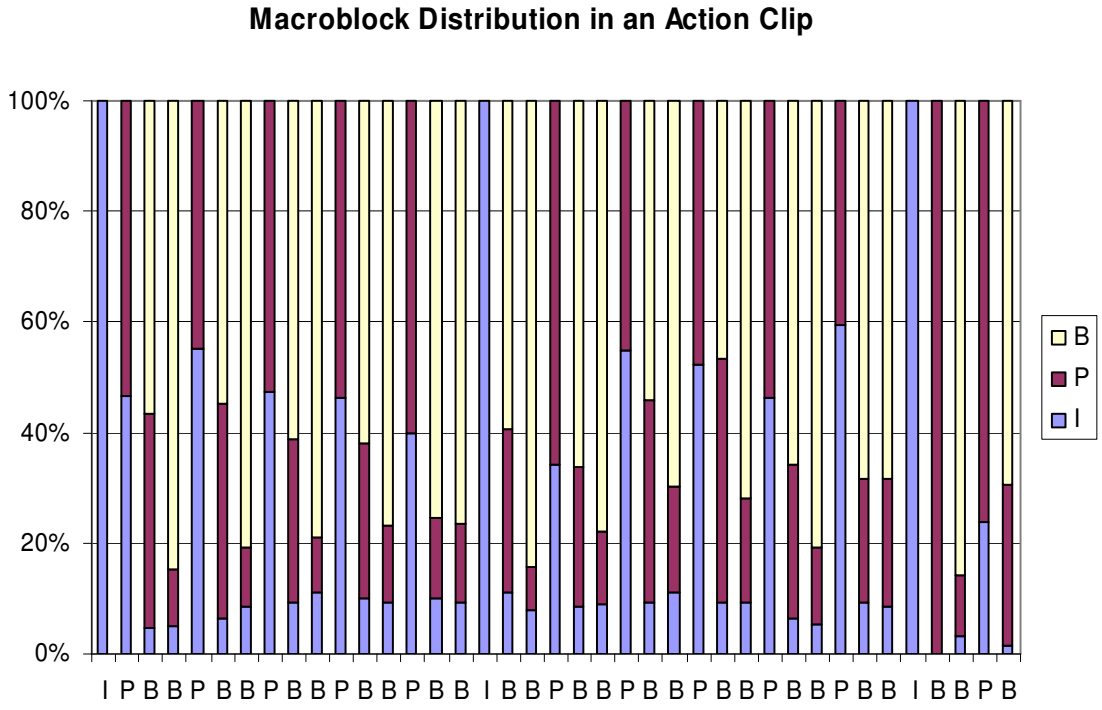

Figure 15: Macroblock Distribution(action clip)

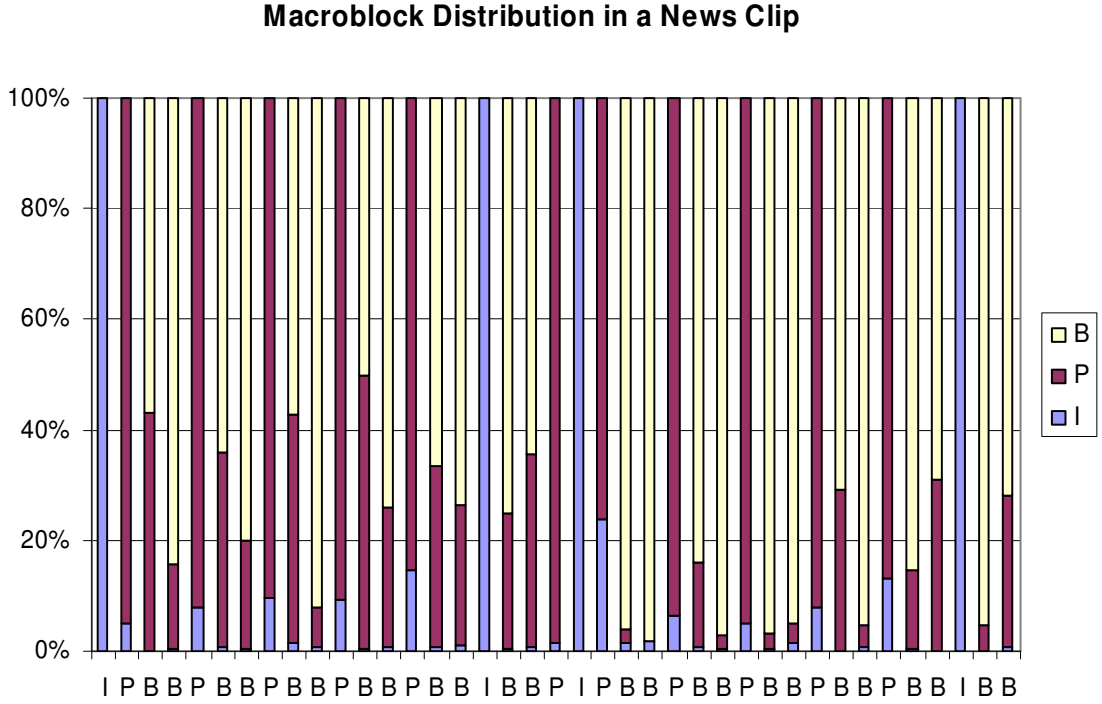

Figure 16: Macroblock Distribution(news clip)

B Frame Decoding Times (one fitting curve)

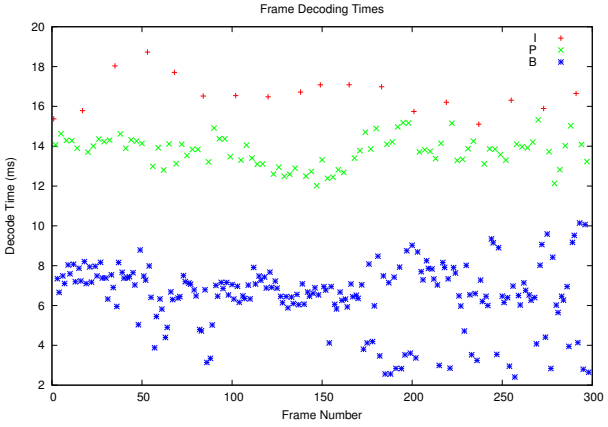

Figure 17: Frame Decoding Times ('action' clip)

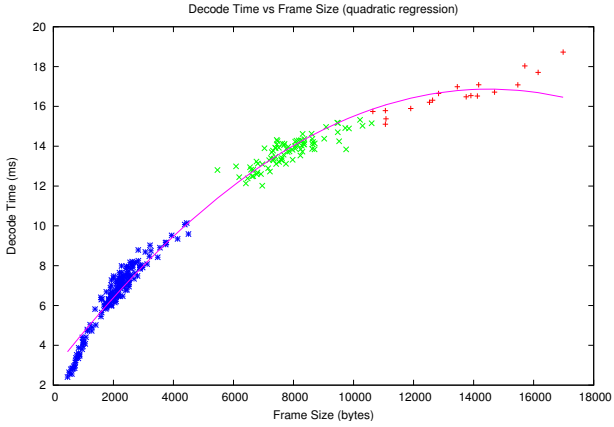

Figure 18: Fitting curve ('action' clip)

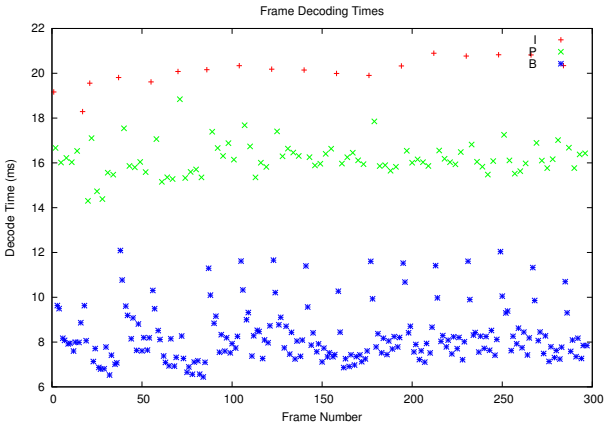

Figure 19: Frame Decoding Times ('news' clip)

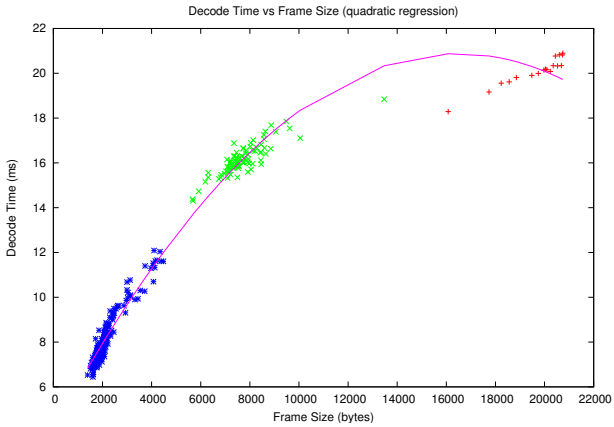

Figure 20: Fitting curve ('news' clip)

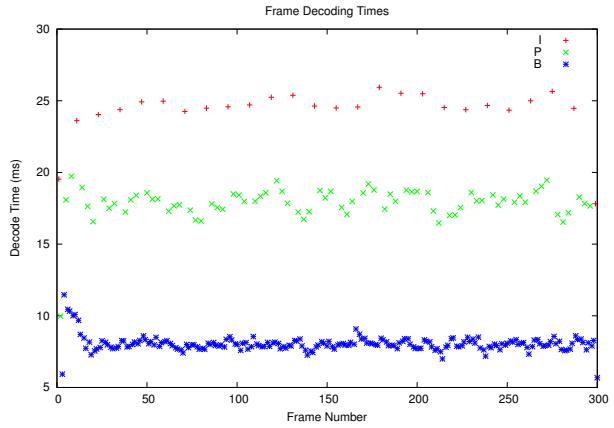

Figure 21: Frame Decoding Times ('akiyo' CIF clip)

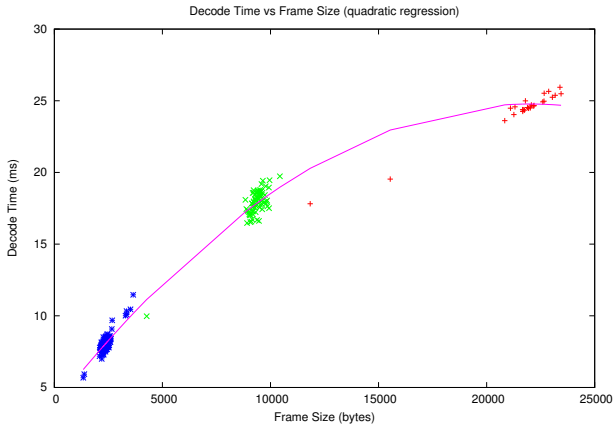

Figure 22: Fitting curve ('akiyo' CIF clip)

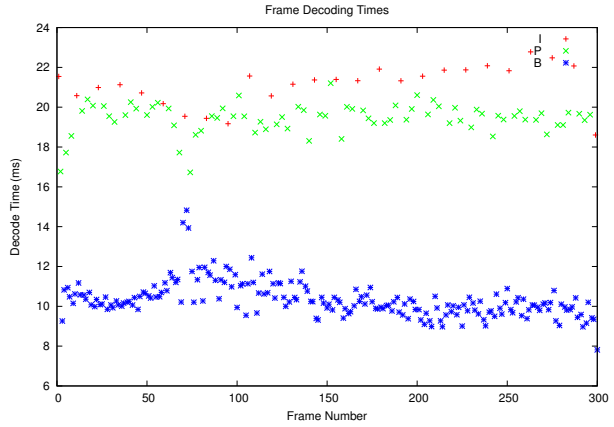

Figure 23: Frame Decoding Times ('coastguard' CIF clip)

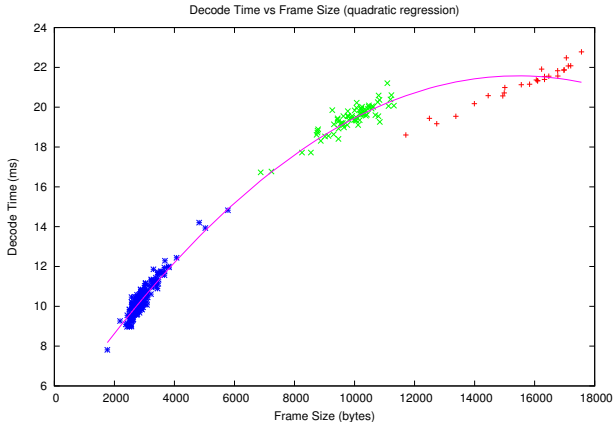

Figure 24: Fitting Curve ('coastguard' CIF clip)

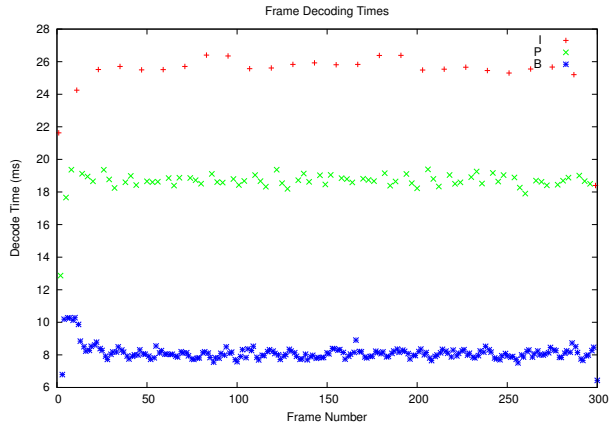

Figure 25: Frame Decoding Times ('container' CIF clip)

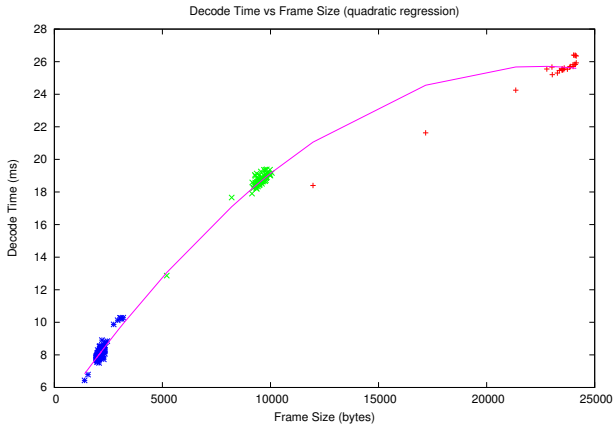

Figure 26: Fitting Curve ('container' CIF clip)

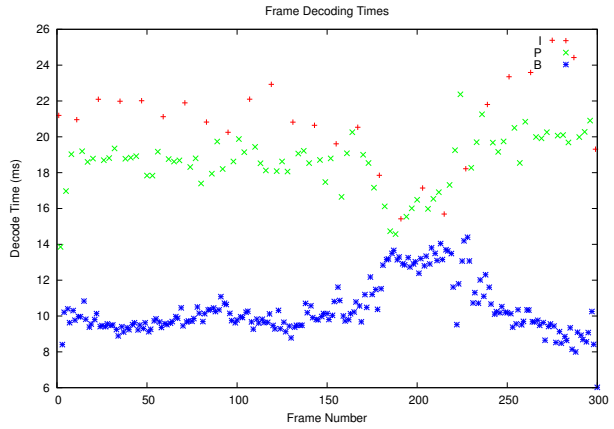

Figure 27: Frame Decoding Times ('foreman' CIF clip)

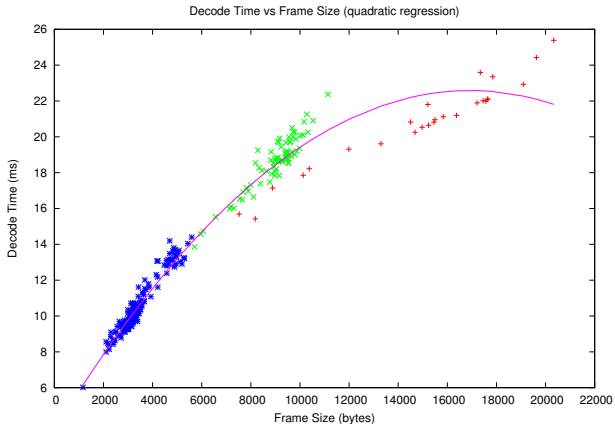

Figure 28: Fitting Curve ('foreman' CIF clip)

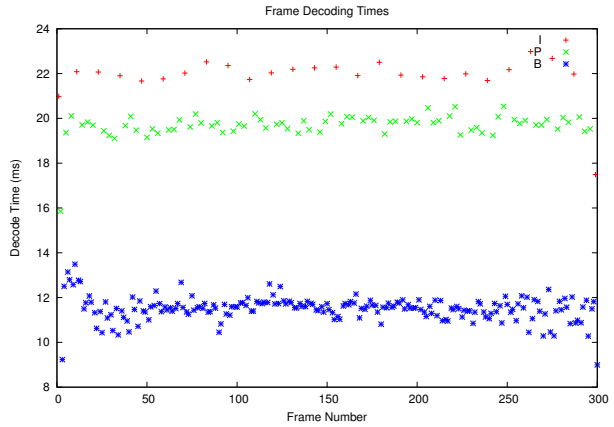

Figure 29: Frame Decoding Times ('hall' CIF clip)

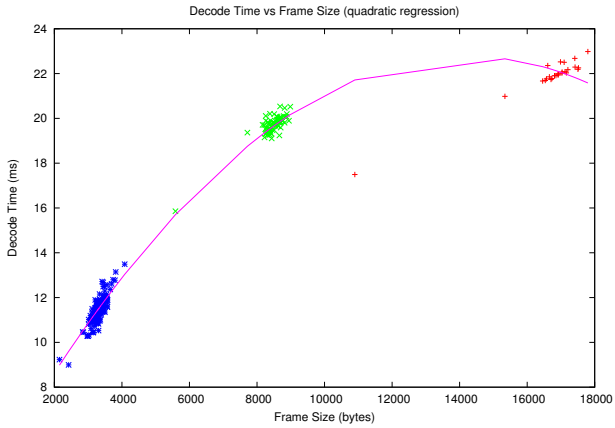

Figure 30: Fitting Curve ('hall' CIF clip)

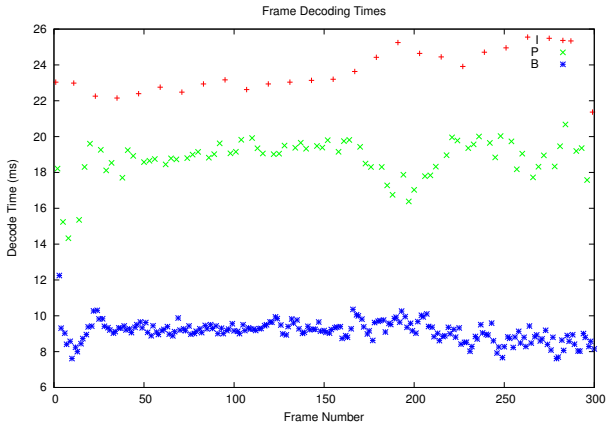

Figure 31: Frame Decoding Times ('mobile' CIF clip)

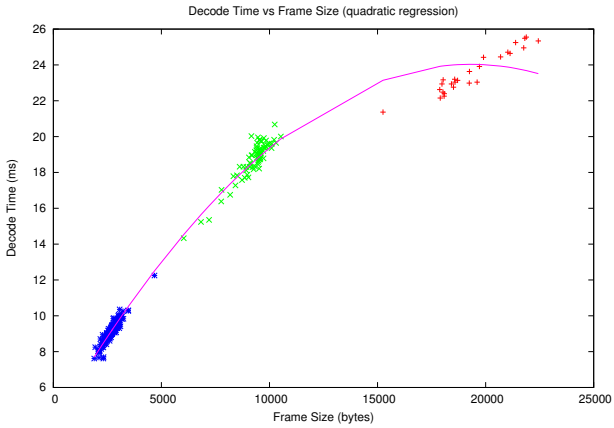

Figure 32: Fitting Curve ('mobile' CIF clip)

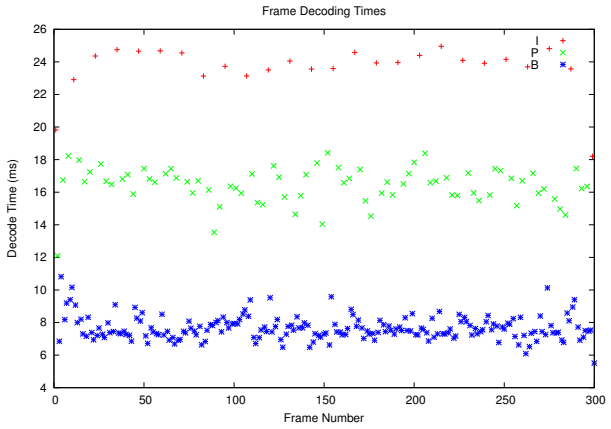

Figure 33: Frame Decoding Times ('news' CIF clip)

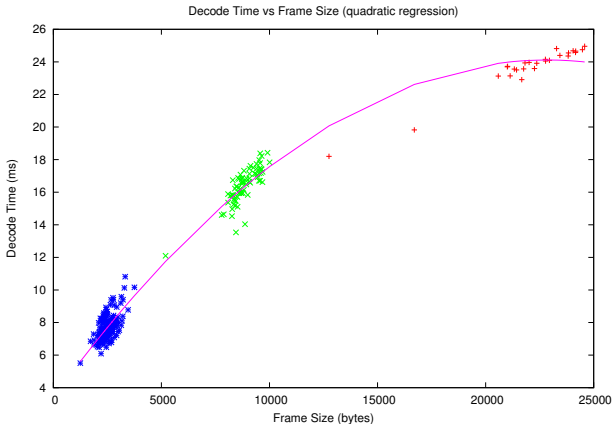

Figure 34: Fitting Curve ('news' CIF clip)

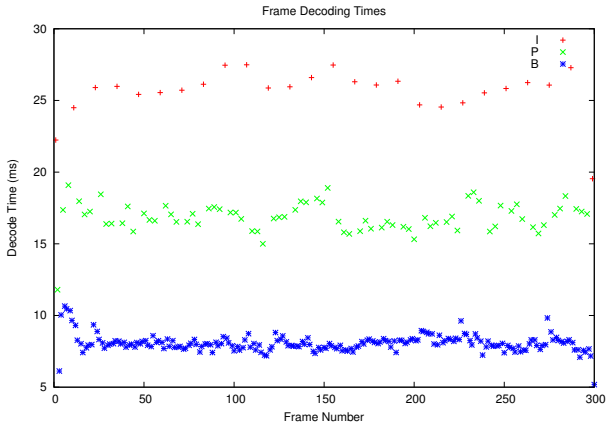

Figure 35: Frame Decoding Times ('silent' CIF clip)

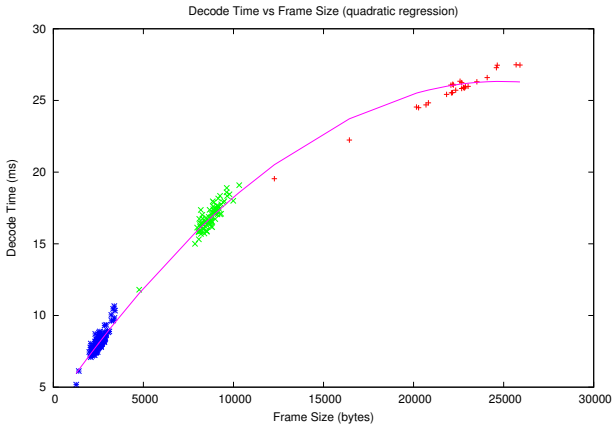

Figure 36: Fitting Curve ('silent' CIF clip)

C Frame Decoding Times (two fitting curves)

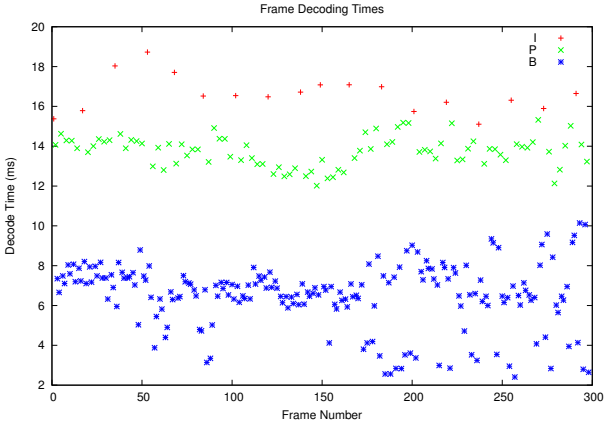

Figure 37: Frame Decoding Times ('action' clip)

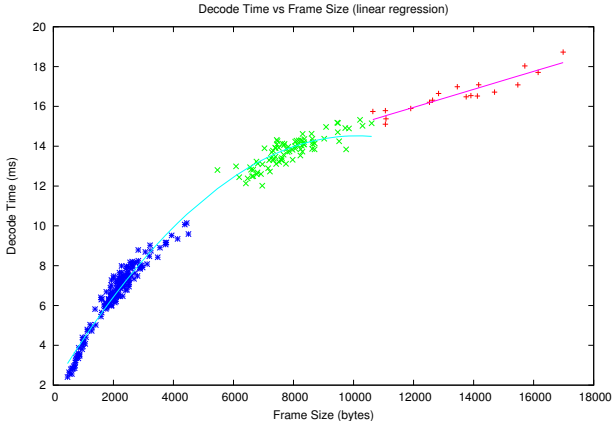

Figure 38: Fitting Curves ('action' clip)

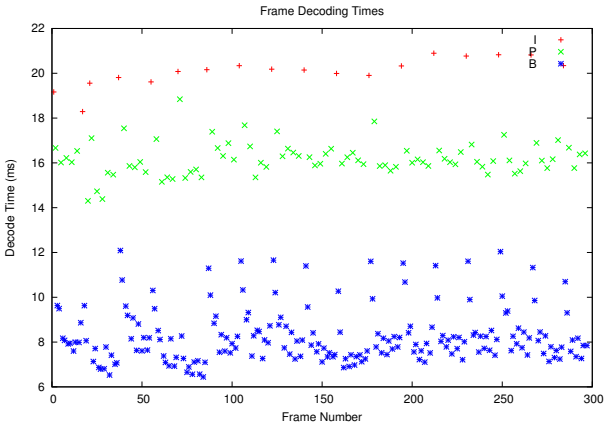

Figure 39: Frame Decoding Times ('news' clip)

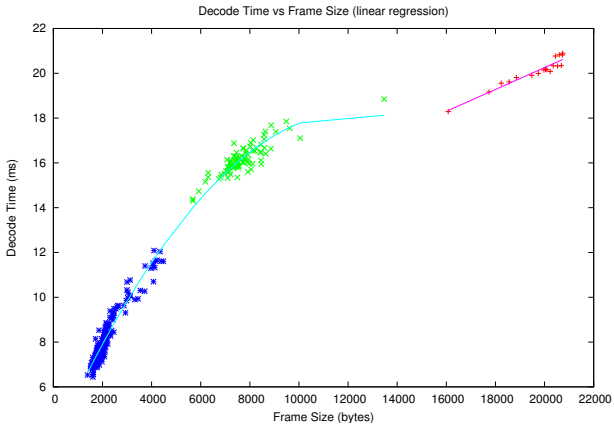

Figure 40: Fitting Curves ('news' clip)

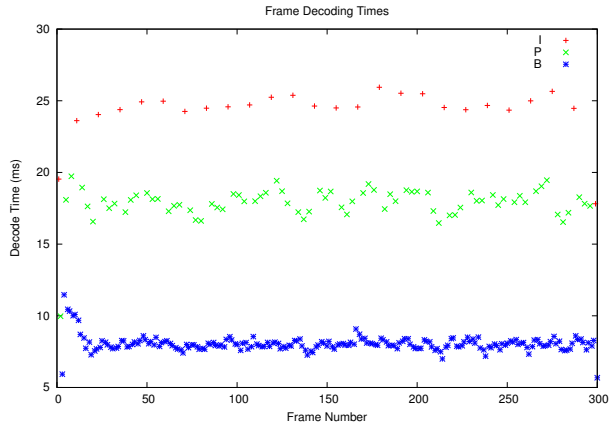

Figure 41: Frame Decoding Times ('akiyo' CIF clip)

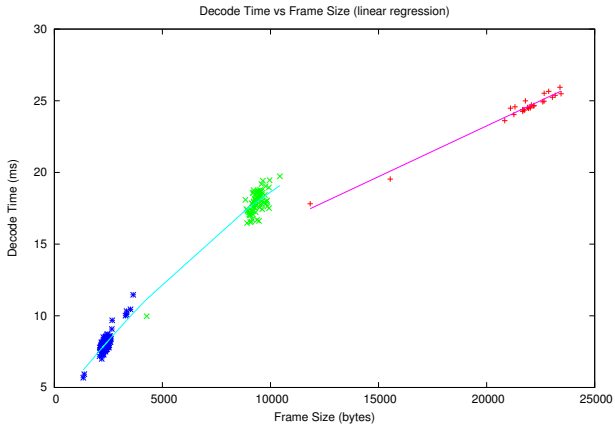

Figure 42: Fitting Curves ('akiyo' CIF clip)

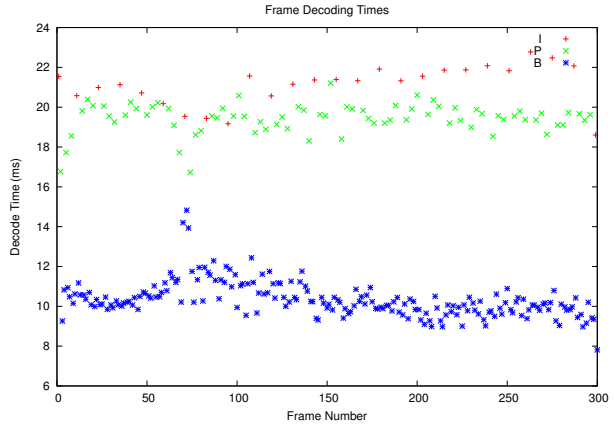

Figure 43: Frame Decoding Times ('coastguard' CIF clip)

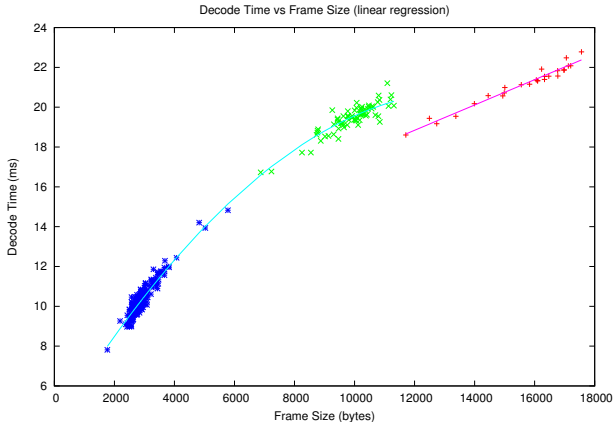

Figure 44: Fitting Curves ('coastguard' CIF clip)

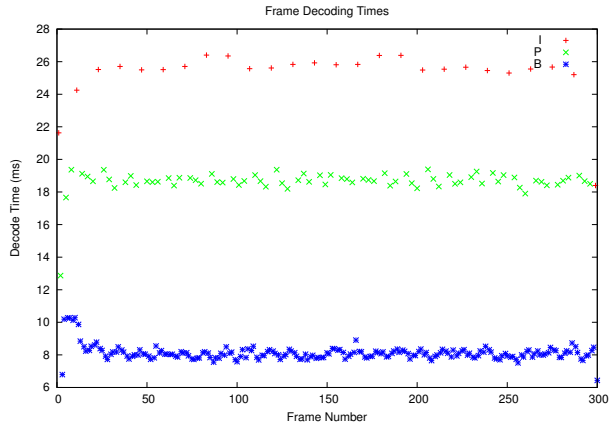

Figure 45: Frame Decoding Times ('container' CIF clip)

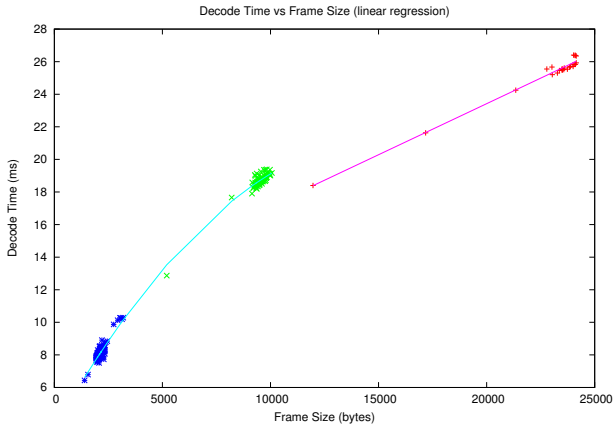

Figure 46: Fitting Curves ('container' CIF clip)

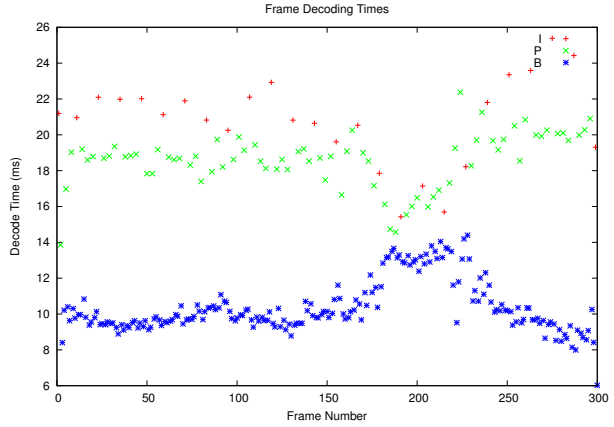

Figure 47: Frame Decoding Times ('foreman' CIF clip)

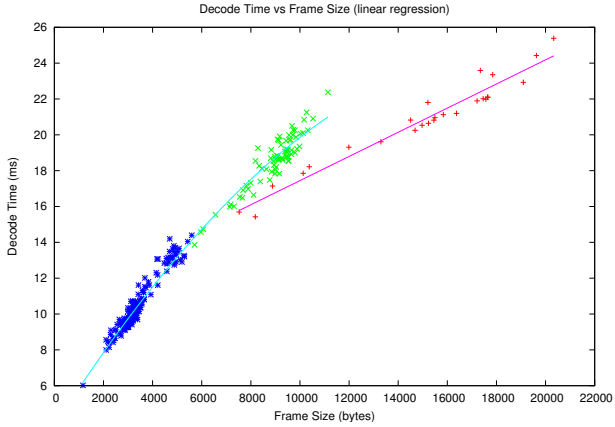

Figure 48: Fitting Curves ('foreman' CIF clip)

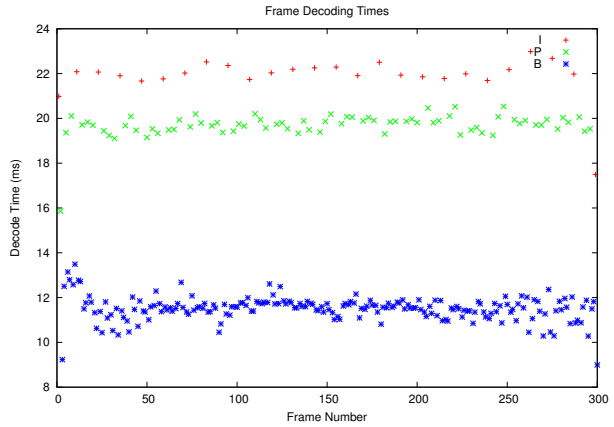

Figure 49: Frame Decoding Times ('hall' CIF clip)

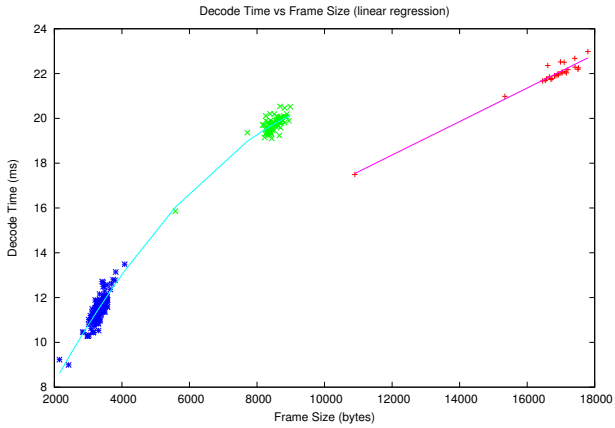

Figure 50: Fitting Curves ('hall' CIF clip)

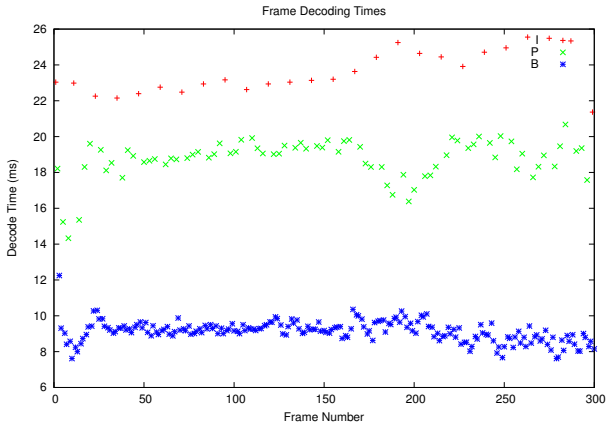

Figure 51: Frame Decoding Times ('mobile' CIF clip)

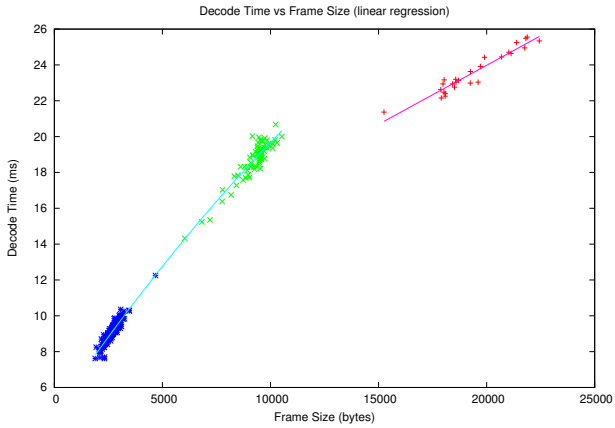

Figure 52: Fitting Curves ('mobile' CIF clip)

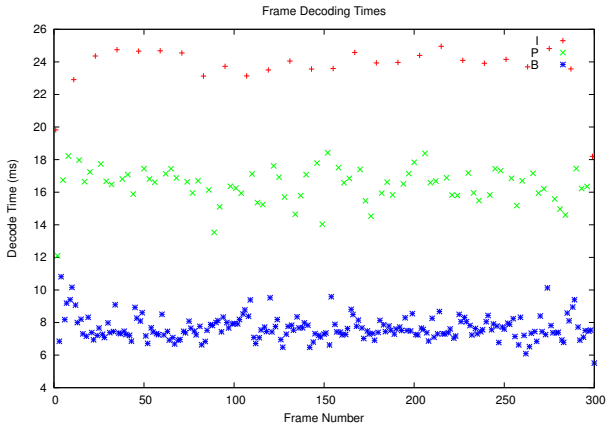

Figure 53: Frame Decoding Times ('news' CIF clip)

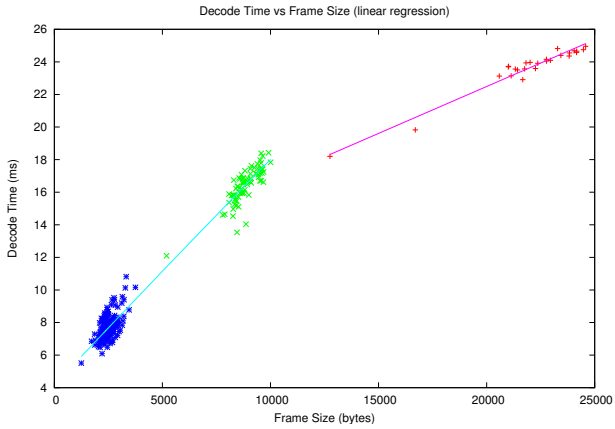

Figure 54: Fitting Curves ('news' CIF clip)

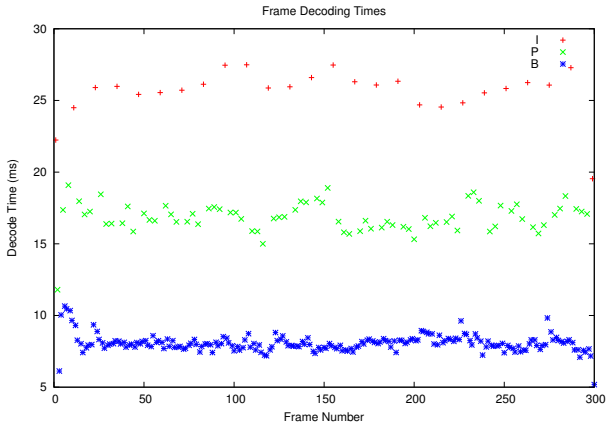

Figure 55: Frame Decoding Times ('silent' CIF clip)

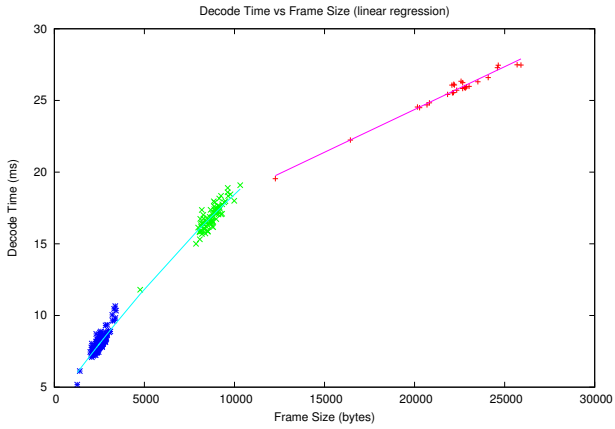

Figure 56: Fitting Curves ('silent' CIF clip)

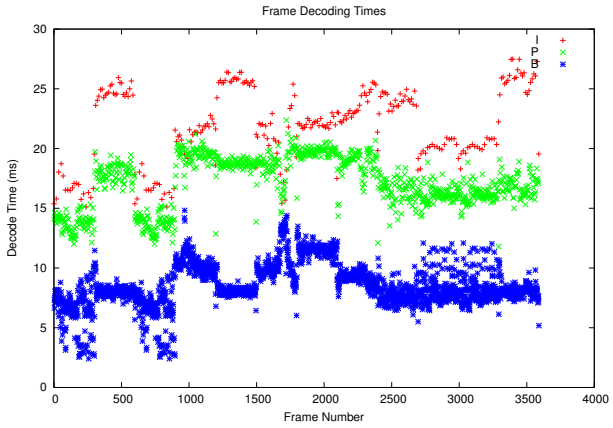

Figure 57: Frame Decoding Times (all clips)

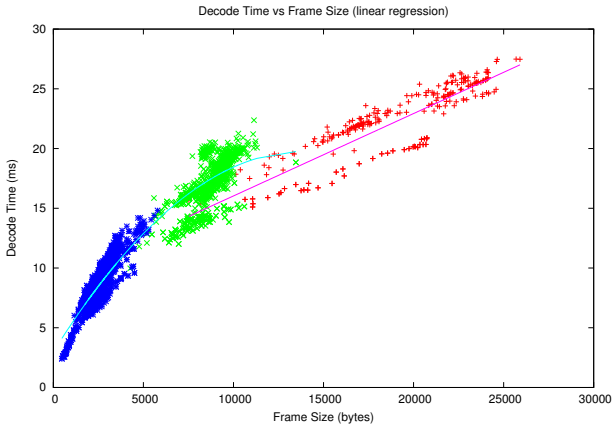

Figure 58: Fitting Curves (all clips)

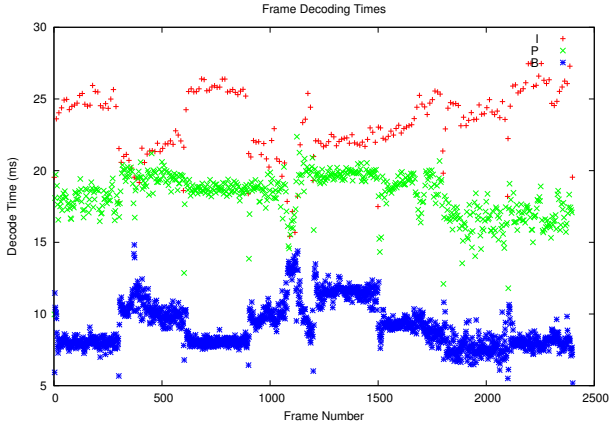

Figure 59: Frame Decoding Times (all CIF clips)

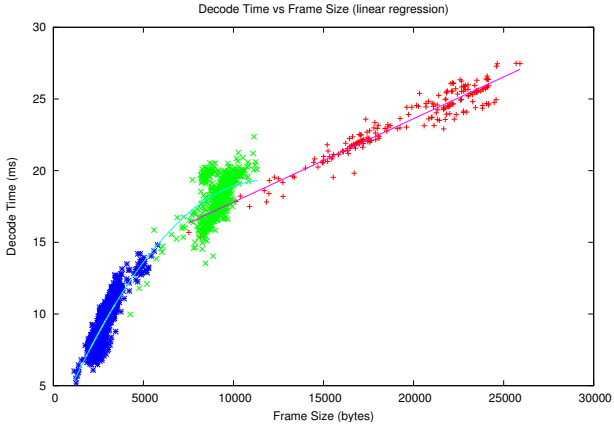

Figure 60: Fitting Curves (all CIF clips)

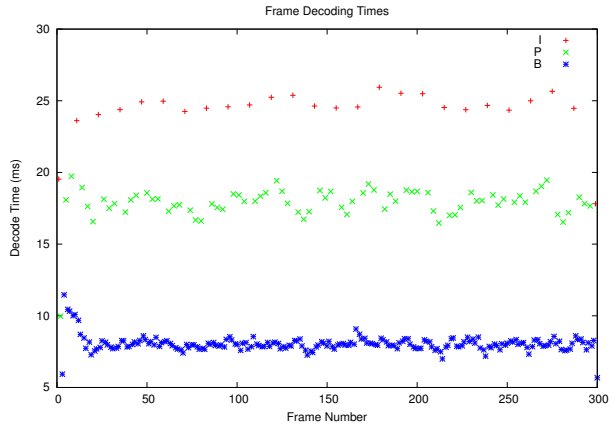

Figure 61: Frame Decoding Times ('akiyo' CIF clip)

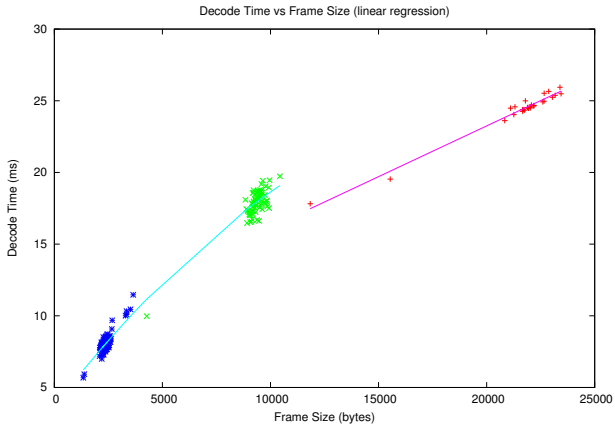

Figure 62: Fitting Curves ('akiyo' CIF clip)

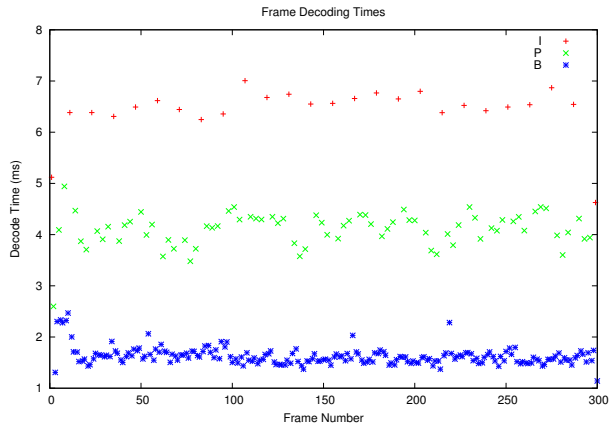

Figure 63: Frame Decoding Times ('akiyo' QCIF clip)

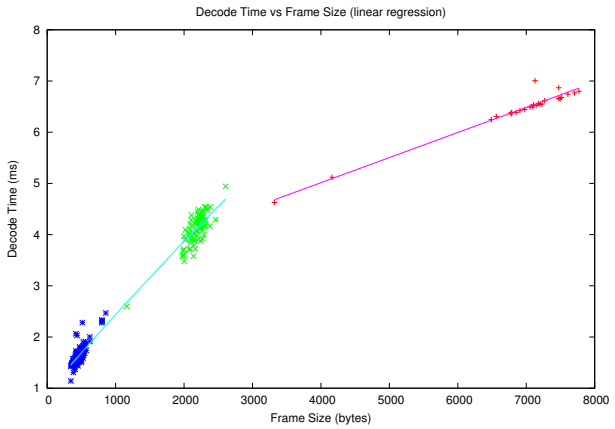

Figure 64: Fitting Curves ('akiyo' QCIF clip)

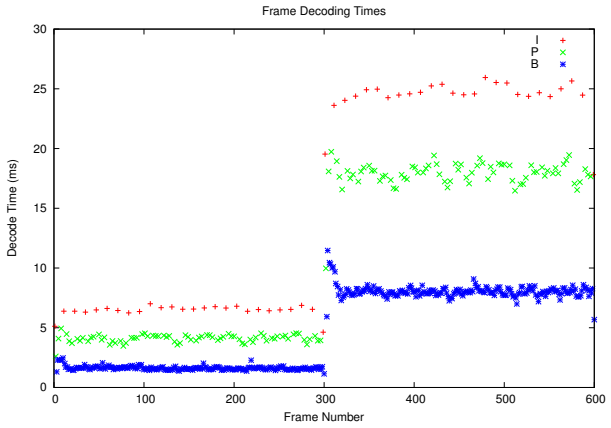

Figure 65: Frame Decoding Times ('akiyo' QCIF+CIF clip)

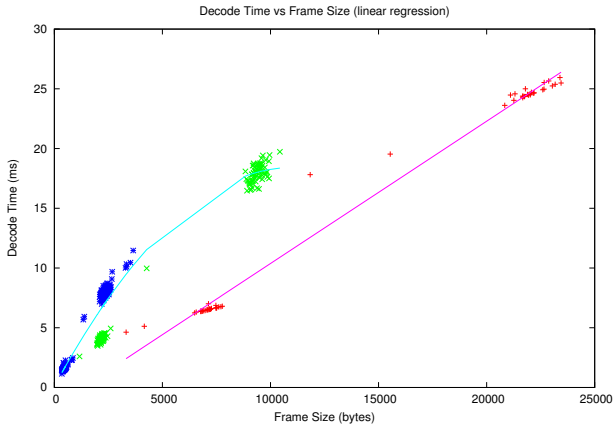

Figure 66: Fitting Curves ('akiyo' QCIF+CIF clip)

D Frame Decoding Times - global estimation (all)

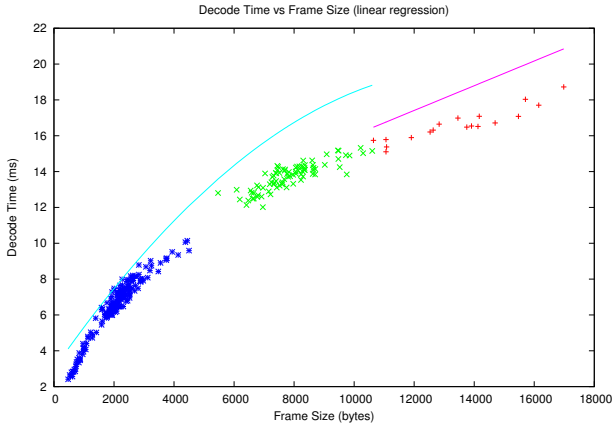

Figure 67: Fitting Curves ('action' clip)

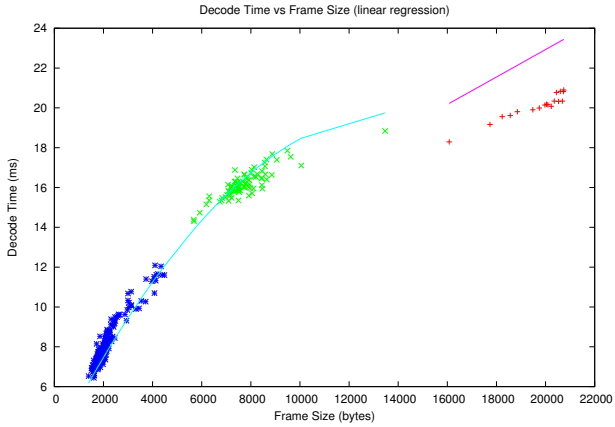

Figure 68: Fitting Curves ('news' clip)

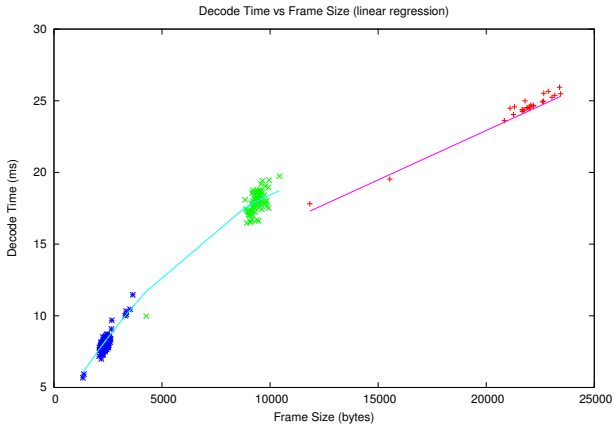

Figure 69: Fitting Curves ('akiyo' CIF clip)

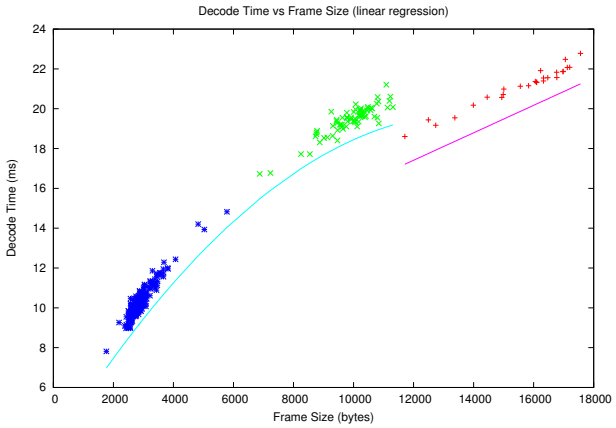

Figure 70: Fitting Curves ('coastguard' CIF clip)

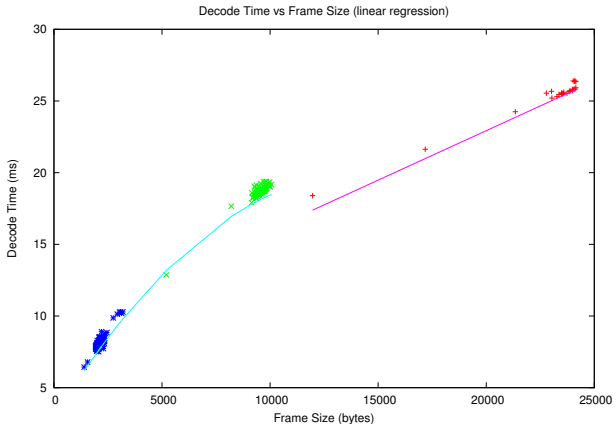

Figure 71: Fitting Curves ('container' CIF clip)

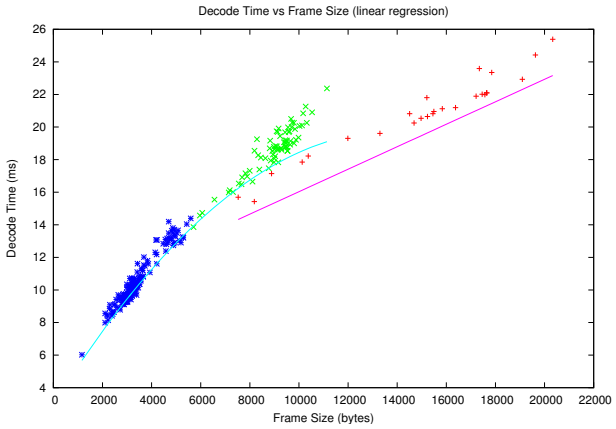

Figure 72: Fitting Curves ('foreman' CIF clip)

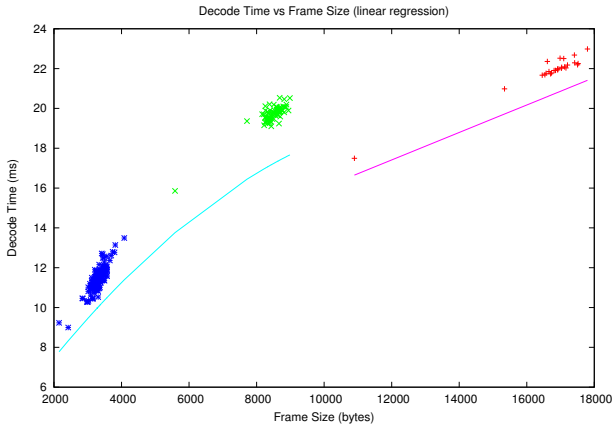

Figure 73: Fitting Curves ('hall' CIF clip)

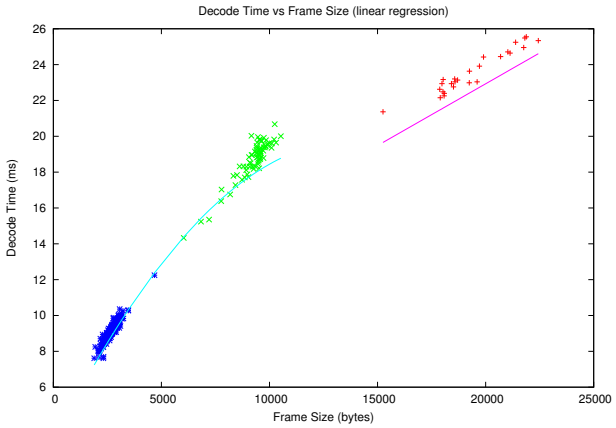

Figure 74: Fitting Curves ('mobile' CIF clip)

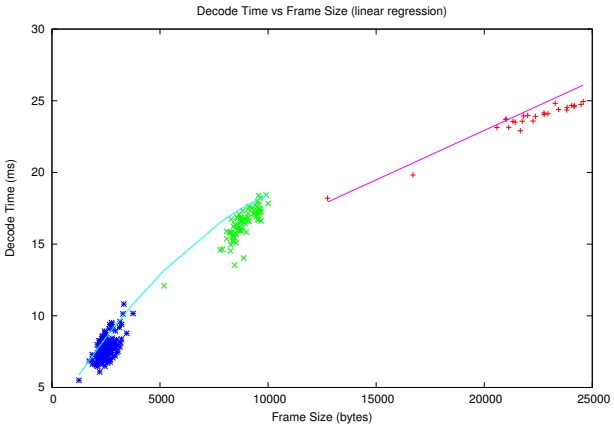

Figure 75: Fitting Curves ('news' CIF clip)

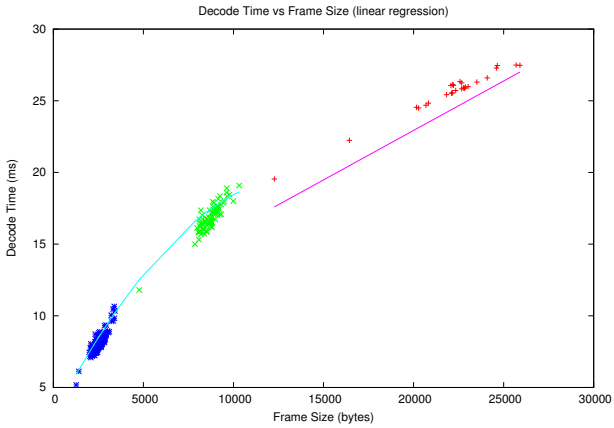

Figure 76: Fitting Curves ('silent' CIF clip)

E Frame Decoding Times - global estimation (all CIF)

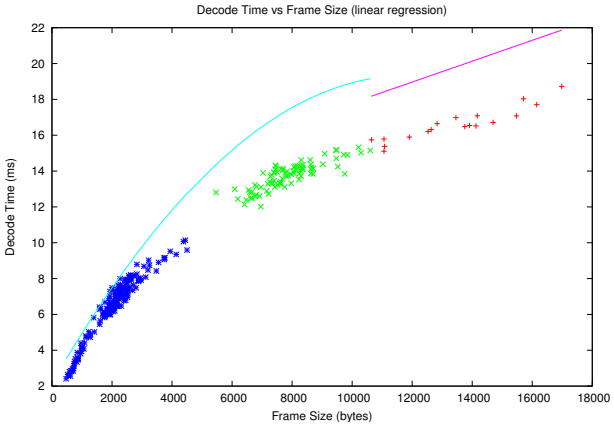

Figure 77: Fitting Curves ('action' clip)

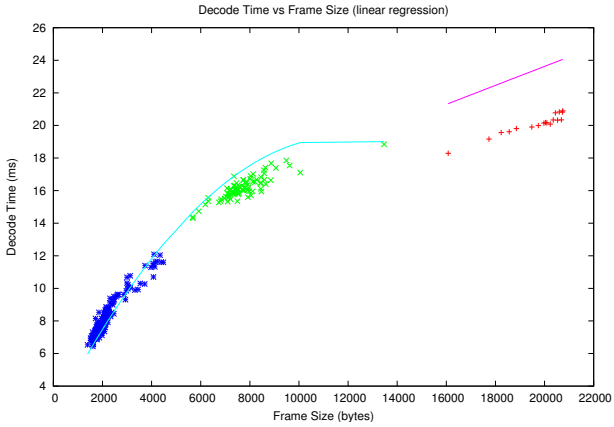

Figure 78: Fitting Curves ('news' clip)

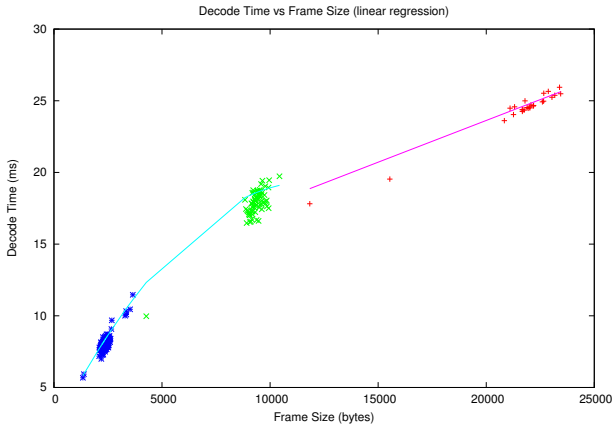

Figure 79: Fitting Curves ('akiyo' CIF clip)

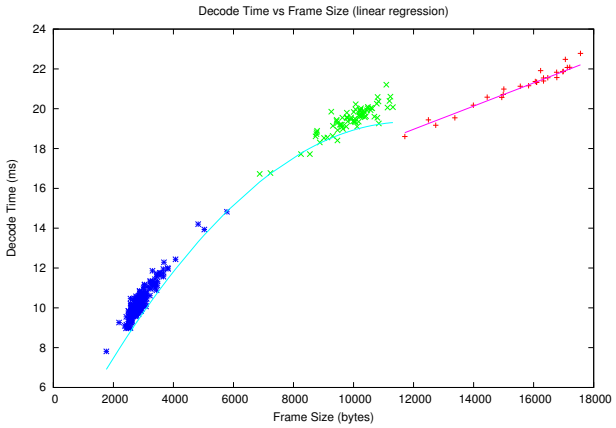

Figure 80: Fitting Curves ('coastguard' CIF clip)

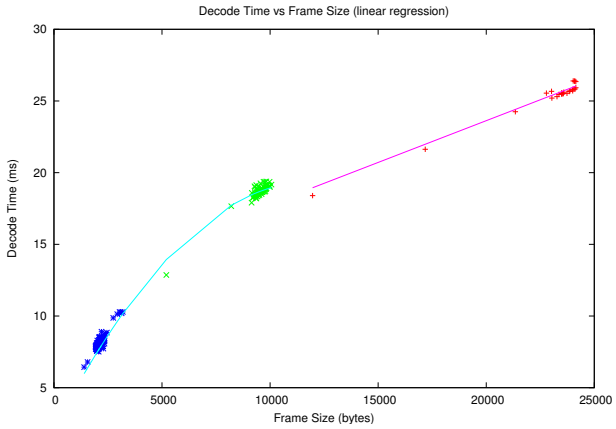

Figure 81: Fitting Curves ('container' CIF clip)

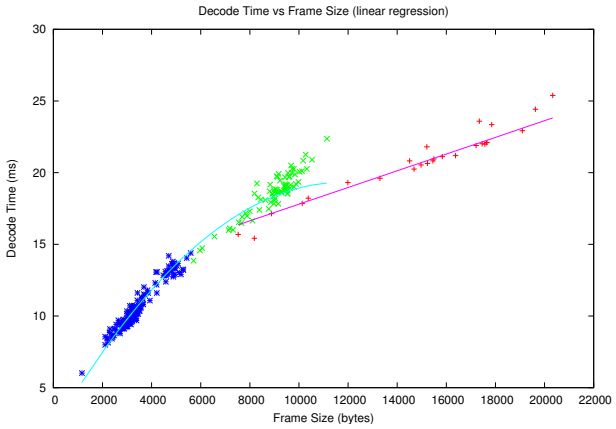

Figure 82: Fitting Curves ('foreman' CIF clip)

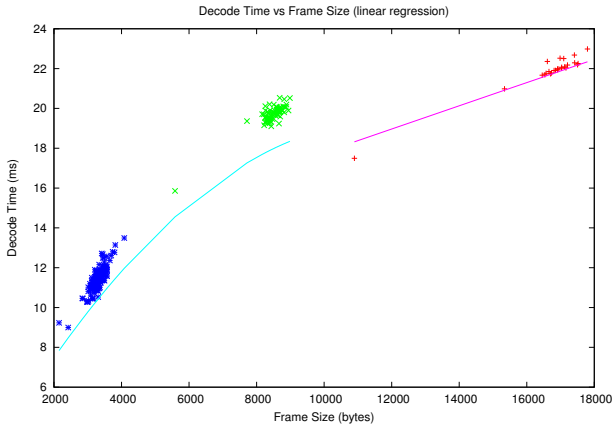

Figure 83: Fitting Curves ('hall' CIF clip)

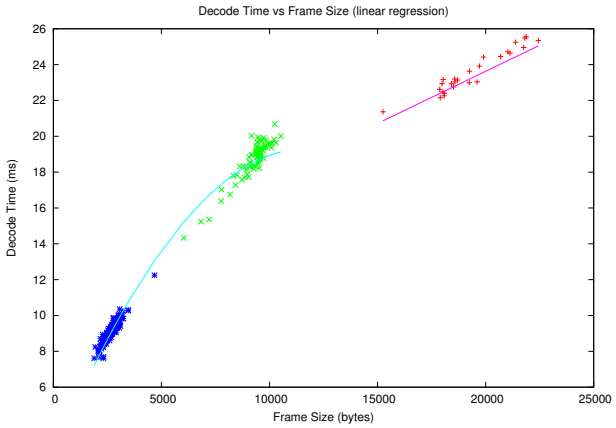

Figure 84: Fitting Curves ('mobile' CIF clip)

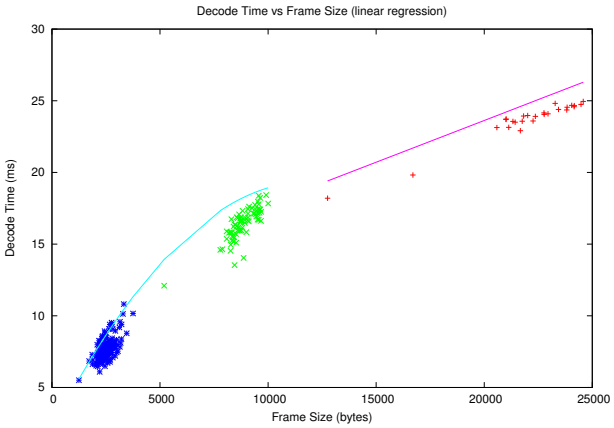

Figure 85: Fitting Curves ('news' CIF clip)

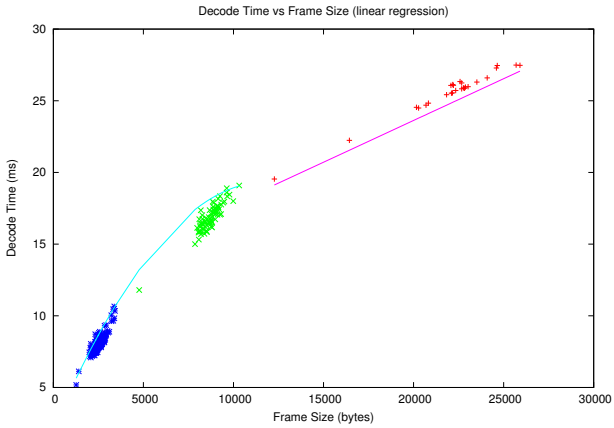

Figure 86: Fitting Curves ('silent' CIF clip)

F Power Savings

Video Clip	Individual Estimation		Global Estimation	
	Savings Over no DVS	Savings Over Simple DVS	Savings Over no DVS	Savings Over Simple DVS
action	65.2033	44.2539	46.636	14.5081
news	59.3555	41.6503	49.5052	27.5091
akiyo_cif	55.4909	48.5246	51.1881	43.5483
coastguard_cif	51.4923	36.118	50.6199	34.9691
container_cif	56.8692	50.9896	56.3697	50.422
foreman_cif	50.5261	41.534	45.4058	35.4831
hall_cif	48.5585	32.8528	47.6642	31.6854
mobile_cif	52.6674	44.4279	52.3581	44.0647
news_cif	54.4522	45.244	43.9385	32.6049
silent_cif	56.562	52.5924	50.4433	45.9146

Table 3: Power Savings for the 2 Curve Fitting