Context-Based Adaptive Binary Arithmetic Coding in the H.264/AVC Video Compression Standard

Detlev Marpe, *Member, IEEE,* Heiko Schwarz, and Thomas Wiegand

***Abstract*—Context-based Adaptive Binary Arithmetic Coding (CABAC) as a normative part of the new ITU-T | ISO/IEC standard H.264/AVC for video compression is presented. By combining an adaptive binary arithmetic coding technique with context modeling, a high degree of adaptation and redundancy reduction is achieved. The CABAC framework also includes a novel low-complexity method for binary arithmetic coding and probability estimation that is well suited for efficient hardware and software implementations. CABAC significantly outperforms the baseline entropy coding method of H.264/AVC for the typical area of envisaged target applications. For a set of test sequences representing typical material used in broadcast applications and for a range of acceptable video quality of about 30 to 38 dB, average bit-rate savings of 9 to 14% are achieved.**

***Index Terms*—CABAC, entropy coding, context modeling, binary arithmetic coding, H.264, MPEG-4 AVC.**

I. Introduction

NATURAL camera-view video signals show non-stationary statistical behavior. The statistics of these signals largely depend on the video content and the acquisition process. Traditional concepts of video coding that rely on a mapping from the video signal to a bitstream of variable length-coded syntax elements exploit some of the non-stationary characteristics but certainly not all of it. Moreover, higher-order statistical dependencies on a syntax element level are mostly neglected in existing video coding schemes. Designing an entropy coding scheme for a video coder by taking into consideration these typically observed statistical properties, however, offers room for significant improvements in coding efficiency.

Context-based Adaptive Binary Arithmetic Coding (CABAC) is one of the two entropy coding methods of the new ITU-T | ISO/IEC standard for video coding, H.264/AVC [1],[2]. The algorithm was first introduced in a rudimentary form in [7] and evolved over a period of successive refinements [8]–[17]. In this paper, we present a description of the main elements of the CABAC algorithm in its final, standardized form as specified in [1]. Unlike the specification in [1], the presentation in this paper is intended to provide also some information on the underlying conceptual ideas as well as the theoretical and historical background of CABAC.

Manuscript received May 21, 2003.

The authors are with the Fraunhofer Institute for Communications – Heinrich Hertz Institute, Berlin, Germany.

Entropy coding in today's hybrid block-based video coding standards such as MPEG-2 [3], H.263 [4], and MPEG-4 [5] is generally based on fixed tables of variable length codes (VLC). For coding the residual data in these video coding standards, a block of transform coefficient levels is first mapped onto a one-dimensional list using an inverse scanning pattern. This list of transform coefficient levels is then coded using a combination of run-length and variable length coding. Due to the usage of variable length codes, coding events with a probability greater than 0.5 cannot be efficiently represented and hence, a so-called *alphabet extension* of "run" symbols representing successive levels with value zero is used in the entropy coding schemes of MPEG-2, H.263, and MPEG-4. Moreover, the usage of fixed VLC tables does not allow an adaptation to the actual symbol statistics, which may vary over space and time as well as for different source material and coding conditions. Finally, since there is a fixed assignment of VLC tables and syntax elements, existing inter-symbol redundancies cannot be exploited within these coding schemes.

Although, from a conceptual point-of-view, it is well known for a long time that all these deficiencies can be most easily resolved by *arithmetic codes* [23], little of this knowledge was actually translated into practical entropy coding schemes specifically designed for block-based hybrid video coding. One of the first hybrid block-based video coding schemes that incorporate an adaptive binary arithmetic coder capable of adapting the model probabilities to the existing symbol statistics was presented in [6]. The core of that entropy coding scheme was inherited from the JPEG standard (at least for coding of DCT coefficients) [25], and an adjustment of its modeling part to the specific statistical characteristics of typically observed residual data in a hybrid video coder was not carried out. As a result, the performance of this JPEG-like arithmetic entropy coder in the hybrid block-based video coding scheme of [6] was not substantially better for inter-coded pictures than that of its VLC-based counterpart.

The first and – until H.264/AVC was officially released – the only standardized arithmetic entropy coder within a hybrid block-based video coder is given by Annex E of H.263 [4]. Three major drawbacks in the design of that optional

arithmetic coding scheme can be identified. First, Annex E is applied to the same syntax elements as the VLC method of H.263 including the combined symbols for coding of transform coefficient levels. Thus, one of the fundamental advantages of arithmetic coding that a non-integer code length can be assigned to coding events is unlikely to be exploited. Second, all probability models in Annex E of H.263 are non-adaptive in the sense that their underlying probability distributions are assumed to be static. Although, multiple probability distribution models are defined and chosen in a frequency-dependent way for the combined symbols of run, level and "last" information, this conditioning does not result in a significant gain in coding efficiency, since an adaptation to the actual symbol statistics is not possible. Finally, the generic *m*-ary arithmetic coder used in Annex E involves a considerable amount of computational complexity, which may not be justified in most application scenarios, especially in view of the typically observed, small margins of coding gains.

Entropy coding schemes based on arithmetic coding are quite frequently involved in the field of non block-based video coding. Most of these alternative approaches to video coding are based on the discrete wavelet transform (DWT) in combination with disparate methods of temporal prediction, such as overlapped block motion compensation, grid-based warping or motion-compensated temporal filtering [18],[19],[20]. The corresponding entropy coding schemes are often derived from DWT-based still image coding schemes like SPIHT [21] or other predecessors of JPEG2000 [35].

In our prior work on wavelet-based hybrid video coding, which led to one of the proposals for the H.26L standardization [19], the entropy coding method of *partitioning, aggregation and conditional coding* (PACC) was developed [22]. One of its main distinguishing features is related to the partitioning strategy: Given a source with a specific alphabet size, for instance, quantized transform coefficients, it was found to be useful to first reduce the alphabet size by partitioning the range according to a binary selector, which e.g. in the case of transform coefficients would be typically given by the decision whether the coefficient is quantized to zero or not. In fact, range partitioning using binary selectors can be viewed as a special case of a *binarization scheme*, where a symbol of a non-binary alphabet is uniquely mapped to a sequence of binary decisions prior to further processing.

This (somehow) dual operation to the aforementioned alphabet extension, which in the sequel we will therefore refer to as *alphabet reduction*, is mainly motivated by the fact that it allows the subsequent modeling stage to operate more efficiently on this maximally reduced (binary) alphabet. In this way, the design and application of higher-order conditioning models is greatly simplified and, moreover, the risk of "overfitting" the model is reduced. As a positive side effect, a fast table-driven binary arithmetic coder can be utilized for the final arithmetic coding stage.

The design of CABAC is in the spirit of our prior work. To circumvent the drawbacks of the known entropy coding schemes for hybrid block-based video coding such as Annex E of H.263, we combine an adaptive binary arithmetic coding technique with a well-designed set of context models. Guided by the principle of alphabet reduction, an additional binarization stage is employed for all non-binary valued symbols. Since the increased computational complexity of arithmetic coding in comparison to variable length coding is generally considered as its main disadvantage, great importance has been devoted to the development of an algorithmic design that allows efficient hardware and software implementations.

For some applications, however, the computational requirements of CABAC may be still too high given today's silicon technology. Therefore, the baseline entropy coding method of H.264/AVC [1] offers a different compression-complexity trade-off operating at reduced coding efficiency and complexity level compared to CABAC. It mostly relies on a single infinite-extended codeword set consisting of zero-order Exp-Golomb codes, which are used for all syntax elements except for the residual data. For coding the residual data, a more sophisticated method called *Context-Adaptive Variable Length Coding* (CAVLC) is employed. In this scheme, inter-symbol redundancies are exploited by switching VLC tables for various syntax elements depending on already transmitted coding symbols [1],[2]. The CAVLC method cannot provide an adaptation to the actually given conditional symbol statistics. Furthermore, coding events with symbol probabilities greater than 0.5 cannot be efficiently coded due to the fundamental lower limit of 1 bit/symbol imposed on variable length codes. This restriction prevents the usage of coding symbols with a smaller alphabet size for coding the residual data, which could allow a more suitable construction of contexts for switching between the model probability distributions.

The remainder of the paper is organized as follows. In Section II, we present an overview of the CABAC framework including a high-level description of its three basic building blocks of binarization, context modeling and binary arithmetic coding. We also briefly discuss the motivation and the principles behind the algorithmic design of CABAC. A more detailed description of CABAC is given in Section III, where the individual steps of the algorithm are presented in depth. Finally, in Section IV we provide experimental results to demonstrate the performance gains of CABAC relative to the baseline entropy coding mode of H.264/AVC for a set of interlaced video test sequences.

II. The CABAC Framework

Fig. 1 shows the generic block diagram for encoding a single syntax element in CABAC.[1] The encoding process

[1] For simplicity and for clarity of presentation we restrict our exposition of CABAC to an encoder only view. In the text of the H.264/AVC standard

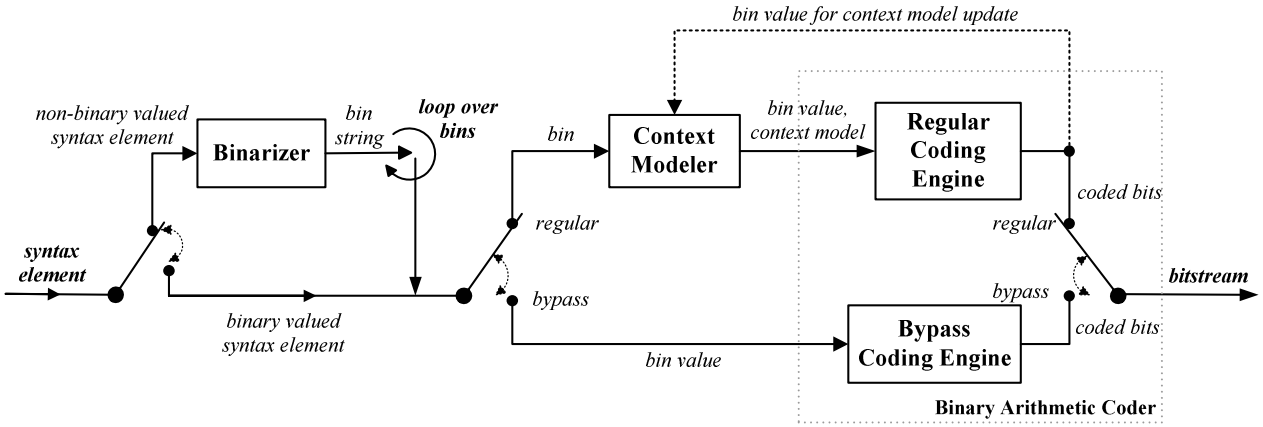

Fig. 1. CABAC encoder block diagram.

consists of at most three elementary steps:

1) binarization
2) context modeling
3) binary arithmetic coding

In the first step, a given non-binary valued syntax element is uniquely mapped to a binary sequence, a so-called *bin string*. When a binary valued syntax element is given, this initial step is bypassed, as shown in Fig. 1. For each element of the bin string or for each binary valued syntax element, one or two subsequent steps may follow depending on the coding mode.

In the so-called *regular coding mode*, prior to the actual arithmetic coding process the given binary decision, which, in the sequel, we will refer to as a *bin*, enters the context modeling stage, where a probability model is selected such that the corresponding choice may depend on previously encoded syntax elements or bins. Then, after the assignment of a context model the bin value along with its associated model is passed to the regular coding engine, where the final stage of arithmetic encoding together with a subsequent model updating takes place (see Fig. 1).

Alternatively, the *bypass coding mode* is chosen for selected bins in order to allow a speedup of the whole encoding (and decoding) process by means of a simplified coding engine without the usage of an explicitly assigned model, as illustrated by the lower right branch of the switch in Fig. 1.

In the following, the three main functional building blocks, which are binarization, context modeling, and binary arithmetic coding, along with their interdependencies are discussed in more detail.

A. Binarization

1) General Approach

For a successful application of context modeling and adaptive arithmetic coding in video coding we found that the following two requirements should be fulfilled:

i) a fast and accurate estimation of conditional probabilities must be achieved in the relatively short time interval of a slice coding unit
ii) the computational complexity involved in performing each elementary operation of probability estimation and subsequent arithmetic coding must be kept at a minimum to facilitate a sufficiently high throughput of these inherently sequentially organized processes.

To fulfill both requirements we introduce the important "pre-processing" step of first reducing the alphabet size of the syntax elements to encode. Alphabet reduction in CABAC is performed by the application of a binarization scheme to each non-binary syntax element resulting in a unique intermediate binary codeword for a given syntax element, called a bin string. The advantages of this approach are both in terms of modeling and implementation.

First, it is important to note that nothing is lost in terms of modeling, since the individual (non-binary) symbol probabilities can be recovered by using the probabilities of the individual bins of the bin string. For illustrating this aspect, let us consider the binarization for the syntax element *mb_type* of a P/SP slice.

As depicted in Fig. 2 (left), the terminal nodes of the binary tree correspond to the symbol values of the syntax element such that the concatenation of the binary decisions for traversing the tree from the root node to the corresponding terminal node represents the bin string of the corresponding symbol value. For instance, consider the value "3" of mb_type, which signals the macroblock type "P_8x8", i.e., the partition of the macroblock into four 8×8 submacroblocks in a P/SP slice. In this case the corresponding bin string is given by "001". As an obvious consequence, the symbol probability p("3") is equal to the product of the probabilities $p^{(C0)}$("0"), $p^{(C1)}$("0") and $p^{(C2)}$("1"), where *C0*, *C1* and *C2* denote the (binary) probability models of the corresponding internal nodes, as shown in Fig. 2. This relation is true for any symbol represented by any such binary tree, which can be deduced by the iterated application of the Total Probability Theorem [26].

[1] itself, the converse perspective dominates – the standard normatively specifies only how to decode the video content without specifying how to encode it.

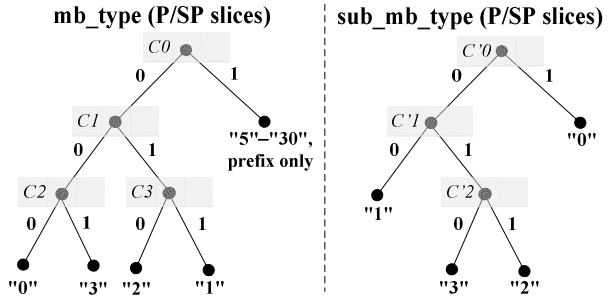

Fig. 2. Illustration of the binarization for mb_type (left) and sub_mb_type (right) both for P/SP slices.

Although at this stage nothing seems to be gained, there is already the advantage of using a binary arithmetic coding engine on the bin string instead of an m-ary arithmetic coder operating on the original m-ary source alphabet. Adaptive m-ary arithmetic coding (for $m > 2$) is in general a computationally complex operation requiring at least two multiplications for each symbol to encode as well as a number of fairly complex operations to perform the update of the probability estimation [36]. In contrast to that, there are fast, multiplication-free variants of binary arithmetic coding, one of which was specifically developed for the CABAC framework, as further described below. Since the probability of symbols with larger bin strings is typically very low, the computational overhead of coding all bins of that bin string instead of using only one pass in an m-ary arithmetic coder is fairly small and can be easily compensated by using a fast binary coding engine.

Finally, as the most important advantage, binarization enables context modeling on a sub-symbol level. For specific bins, which, in general, are represented by the most frequently observed bins, conditional probabilities can be used, whereas other, usually less frequently observed bins can be treated using a joint, typically zero-order probability model. Compared to the conventional approach of using context models in the original domain of the source with typically large alphabet size (like e.g. components of motion vector differences or transform coefficient levels) this additional freedom in the design offers a flexible instrument for using higher-order conditional probabilities without suffering from context "dilution" effects. These effects are often observed in cases, where a large number of conditional probabilities have to be adaptively estimated on a relatively small (coding) time interval, such that there are not enough samples to reach a reliable estimate for each model.[2]

For instance, when operating in the original alphabet domain, a quite moderately chosen 2nd order model for a given syntax element alphabet of size $m = 256$ will result in the intractably large number of $256^2 \cdot (256 - 1) \approx 2^{24}$ symbol probabilities to be estimated for that particular syntax element only. Even for a zero-order model, the task of tracking 255 individual probability estimates according to the previous example is quite demanding. However, typically measured probability density functions (*pdf*) of prediction residuals or transformed prediction errors can be modeled by highly peaked Laplacian, generalized Gaussian or geometric distributions [28], where it is reasonable to restrict the estimation of individual symbol statistics to the area of the largest statistical variations at the peak of the *pdf*. Thus, if, for instance, a binary tree resulting from a Huffman code design would be chosen as a binarization for such a source and its related *pdf*, only the nodes located in the vicinity of the root node would be natural candidates for being modeled individually, whereas a joint model would be assigned to all nodes on deeper tree levels corresponding to the "tail" of the *pdf*. Note that this design is different from the example given in Fig. 2, where each (internal) node has its own model.

In the CABAC framework, typically, only the root node would be modeled using higher-order conditional probabilities. In the above example this would result for a 2nd order model in only 4 different binary probability models instead of m^2 different m-ary probability models with $m = 256$.

2) *Design of CABAC Binarization Schemes*

As already indicated above, a binary representation for a given non-binary valued syntax element provided by the binarization process should be close to a minimum-redundancy code. On the one hand, this allows to easily accessing the most probable symbols by means of the binary decisions located at or close to the root node for the subsequent modeling stage. On the other hand, such a code tree minimizes the number of binary symbols to encode on the average, hence minimizing the computational workload induced by the binary arithmetic coding stage.

However, instead of choosing a Huffman tree for a given training sequence, the design of binarization schemes in CABAC (mostly) relies on a few basic code trees, whose structure enables a simple on-line computation of all code words without the need for storing any tables. There are four such basic types: the *unary code*, the *truncated unary code*, the *k^{th} order Exp-Golomb code* and the *fixed-length code*. In addition, there are binarization schemes based on a concatenation of these elementary types. As an exception of these structured types, there are five specific, mostly unstructured binary trees that have been manually chosen for the coding of macroblock types and sub-macroblock types. Two examples of such trees are shown in Fig. 2.

In the remaining part of this section, we explain in more detail the construction of the four basic types of binarization and its derivatives.

Unary and Truncated Unary Binarization Scheme: For each unsigned integer valued symbol $x \geq 0$ the unary code word in CABAC consists of x "1" bits plus a terminating "0" bit. The truncated unary (*TU*) code is only defined for x with $0 \leq x \leq S$, where for $x < S$ the code is given by the unary code, whereas for $x = S$ the terminating "0" bit is neglected such that the *TU* code of $x = S$ is given by codeword

[2] A more rigorous treatment of that problem can be found in [23][24].

TABLE I
UEG0 BINARIZATION FOR ENCODING OF ABSOLUTE VALUES OF TRANSFORM COEFFICIENT LEVELS

Abs. value	Bin string																		
	TU prefix													*EG0* suffix					
1	0																		
2	1	0																	
3	1	1	0																
4	1	1	1	0															
5	1	1	1	1	0														
...	.	.	.	.	.	.													
...	.	.	.	.	.	.	.	.	.	.	.	.							
13	1	1	1	1	1	1	1	1	1	1	1	0							
14	1	1	1	1	1	1	1	1	1	1	1	1	0						
15	1	1	1	1	1	1	1	1	1	1	1	1	1	0					
16	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0			
17	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1			
18	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	0	
19	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	0	1	
20	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	
...	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.
bin	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	...

consisting of x "1" bits only.

k^{th} order Exp-Golomb Binarization Scheme: Exponential Golomb codes were first proposed by Teuhola [29] in the context of run-length coding schemes. This parameterized family of codes is a derivative of Golomb codes, which have been proven to be optimal prefix-free codes for geometrically distributed sources [30]. Exp-Golomb codes are constructed by a concatenation of a prefix and a suffix code word. Fig. 3 shows the construction of the *k^{th} order Exp-Golomb (EGk)* code word for a given unsigned integer symbol x. The prefix part of the *EGk* code word consists of a unary code corresponding to the value $l(x) = \lfloor \log_2(x/2^k + 1) \rfloor$.

The *EGk* suffix part is computed as the binary representation of $x + 2^k (1 - 2^{l(x)})$ using $k + l(x)$ significant bits, as can be seen from the pseudo-C code in Fig. 3.

Consequently, for the *EGk* binarization the number of symbols having the same code length of $k + 2 \cdot l(x) + 1$ is geometrically growing. By inverting Shannon's relationship between ideal code length and symbol probability, we can e.g. easily deduce that *EG0* is the optimal code for a *pdf* $p(x) = ½ \cdot (x + 1)^{-2}$ with $x \geq 0$. This implies that for an appropriately chosen parameter k the *EGk* code represents a fairly good first-order approximation of the ideal prefix-free code for tails of typically observed *pdf*s, at least for syntax elements that are representing prediction residuals.

Fixed-Length Binarization Scheme: For the application of fixed-length (*FL*) binarization, a finite alphabet of values of the corresponding syntax element is assumed. Let x denote a given value of such a syntax element, where $0 \leq x < S$. Then, the *FL* code word of x is simply given by the binary representation of x with a fixed (minimum) number $l_{FL} = \lceil \log_2 S \rceil$ of bits. Typically, *FL* binarization is applied to syntax elements with a nearly uniform distribution or to syntax elements, where each bit in the *FL* binary representation represents a specific coding decision as e.g. in the part of the coded block pattern symbol related to the luminance residual data.

Concatenation of Basic Binarization Schemes: From the basic binarization schemes as described above three more binarization schemes are derived. The first one is a concatenation of a 4-bit *FL* prefix as a representation of the luminance related part of the coded block pattern and a *TU* suffix with $S = 2$ representing the chrominance related part of coded_block_pattern.

Both the second and third concatenated scheme are derived from the *TU* and the *EGk* binarization. These schemes, which are referred as *Unary / k^{th} order Exp-Golomb* (*UEGk*) binarizations, are applied to motion vector differences and absolute values of transform coefficient levels. The design of these concatenated binarization schemes is motivated by the following observations. First, the unary code is the simplest prefix-free code in terms of implementation cost. Secondly, it permits a fast adaptation of the individual symbol probabilities in the subsequent context modeling stage, since the arrangement of the nodes in the corresponding tree is typically such that with increasing distance of the internal nodes from the root node the corresponding binary probabilities are less skewed.[3] These observations are only accurate for small values of the absolute motion vector differences and transform coefficient levels. For larger values, there is not much use of an adaptive modeling leading to the idea of concatenating an adapted truncated unary tree as a prefix and a static Exp-Golomb code tree as a suffix. Typically, for larger values, the *EGk* suffix part represents already a fairly good fit to the observed probability distribution, as already mentioned above. Thus, it is reasonable to speedup the encoding of the bins related to the *EGk* suffix part in CABAC by using the fast bypass coding engine for uniformly distributed bins, as further described in Section III.D.

For motion vector differences *UEGk* binarization is constructed as follows. Let us assume the value *mvd* of a motion vector component is given. For the prefix part of the *UEGk* bin string, a *TU* binarization is invoked using the absolute value of *mvd* with a cut-off value of $S = 9$. If *mvd* is equal to zero, the bin string consists only of the prefix code word "0". If the condition $|mvd| \geq 9$ holds, the suffix is constructed as an *EG3* codeword for the value of $|mvd| - 9$, to which the sign of *mvd* is appended using the sign bit "1" for a negative *mvd* and the sign bit "0" otherwise. For *mvd* values with $0 < |mvd| < 9$, the suffix consists only of the sign bit. Noting that the component of a motion vector difference represents the prediction error at quarter-sample accuracy, the prefix part always corresponds to a maximum error component of ±2 samples. With the choice of the Exp-Golomb parameter $k = 3$, the suffix code words are given

[3] The aspect of a suitable ordering of nodes in binary trees for optimal modeling and fast adaptation has been addressed in [31], although in a slightly different context.

DOCKET ALARM

Explore Litigation Insights

Docket Alarm provides insights to develop a more informed litigation strategy and the peace of mind of knowing you're on top of things.

Real-Time Litigation Alerts

Keep your litigation team up-to-date with **real-time alerts** and advanced team management tools built for the enterprise, all while greatly reducing PACER spend.

Our comprehensive service means we can handle Federal, State, and Administrative courts across the country.

Advanced Docket Research

With over 230 million records, Docket Alarm's cloud-native docket research platform finds what other services can't. Coverage includes Federal, State, plus PTAB, TTAB, ITC and NLRB decisions, all in one place.

Identify arguments that have been successful in the past with full text, pinpoint searching. Link to case law cited within any court document via Fastcase.

Analytics At Your Fingertips

Learn what happened the last time a particular judge, opposing counsel or company faced cases similar to yours.

Advanced out-of-the-box PTAB and TTAB analytics are always at your fingertips.

API

Docket Alarm offers a powerful API (application programming interface) to developers that want to integrate case filings into their apps.

LAW FIRMS
Build custom dashboards for your attorneys and clients with live data direct from the court.

Automate many repetitive legal tasks like conflict checks, document management, and marketing.

FINANCIAL INSTITUTIONS
Litigation and bankruptcy checks for companies and debtors.

E-DISCOVERY AND LEGAL VENDORS
Sync your system to PACER to automate legal marketing.