Differential Congestion Notification: Taming the Elephants

Long Le Jay Aikat Kevin Jeffay F. Donelson Smith
Department of Computer Science
University of North Carolina at Chapel Hill
http://www.cs.unc.edu/Research/dirt

Abstract — Active queue management (AQM) in routers has been proposed as a solution to some of the scalability issues associated with TCP's pure end-to-end approach to congestion control. A recent study of AQM demonstrated its effectiveness in reducing the response times of web request/response exchanges as well as increasing link throughput and reducing loss rates [10]. However, use of the ECN (explicit congestion notification) signaling protocol was required to outperform drop-tail queuing. Since ECN is not currently widely deployed on end-systems, we investigate an alternative to ECN, namely applying AQM differentially to flows based on a heuristic classification of the flow's transmission rate. Our approach, called differential congestion notification (DCN), distinguishes between "small" flows and "large" high-bandwidth flows and only provides congestion notification to large high-bandwidth flows. We compare DCN to other prominent AQM schemes and demonstrate that for web and general TCP traffic, DCN outperforms all the other AQM designs, including those previously designed to differentiate between flows based on their size and rate.

1. Introduction

Congestion control on the Internet has historically been performed end-to-end with end-systems assuming the responsibility for detecting congestion and reacting to it appropriately. Currently, TCP implementations detect instances of packet loss, interpret these events as indicators of congestion, and reduce the rate at which they are transmitting data by reducing the connection's window size. This congestion reaction (combined with a linear probing congestion avoidance mechanism) successfully eliminated the occurrence of congestion collapse events on the Internet and has enabled the growth of the Internet to its current size.

However, despite this success, concerns have been raised about the future of pure end-to-end approaches to congestion control [1, 5]. In response to these concerns, router-based congestion control schemes known as active queue management (AQM) have been developed and proposed for deployment on the Internet [1]. With AQM, it is now possible for end-systems to receive a signal of incipient congestion prior to the actual occurrence of congestion. The signal can be implicit, realized by a router dropping a packet from a connection even though resources exist to enqueue and forward the packet, or the signal can be explicit, realized by a router setting an explicit congestion notification (ECN) bit in the packet's header and forwarding the packet.

In a previous study of the effects of prominent AQM designs on web performance, we argued that ECN was required in order to realize the promise of AQM [10]. This was a positive result that showed a tangible benefit to both users and service providers to deploying AQM with ECN. When compared to drop-tail routers, the deployment of particular AQM schemes with ECN (and with ECN support in end-system protocol stacks) allowed users to experience significantly reduced response times for web request/response exchanges, and allowed service providers to realize higher link utilization and lower loss rates. Without ECN, certain AQM schemes could realize modest performance improvements over simple drop-tail queue management, but the gains were small compared to those achievable with ECN.

The positive ECN results, however, beg the question of whether or not all AQM inherently requires ECN in order to be effective, or if it simply is the case that only existing AQM designs require ECN in order to be effective. This is a significant issue because ECN deployment requires the participation of both routers and end-systems and hence raises a number of issues including the cost and complexity of implementing and deploying ECN, the incremental deployability of ECN, and the (largely unstudied) issue of dealing with malicious end-systems that advertise ECN support but in fact ignore ECN signals or simply have not been configured appropriately. Other deployment issues include the fact that many firewalls and network address translators intentionally or unintentionally drop all ECN packets or clear ECN bits. In a study of TCP behavior, Padhye and Floyd found that less than 10% of the 24,030 web servers tested had ECN enabled, of which less than 1% had a compliant implementation of ECN [14]. More recent results (August 2003) showed that only 1.1% of 441 web servers tested had correctly deployed ECN [15]. This clearly points to obvious difficulties in deploying and properly using ECN on the end-systems. Thus, AQM could be significantly more appealing if ECN were not required for effective operation.

In this paper, we present an AQM design that signals congestion based on the size and rate of the flow and does not require ECN for good performance. Our approach is to differentially signal congestion to flows (through the dropping of packets) based upon a heuristic classification of the length and rate of the flow. We classify traffic into "mice," short connections that dominate on many Internet links (more than 84% of all flows in some cases [21]), and "elephants," long connections that, while relatively rare, account for the majority of bytes transferred on most links (more than 80% of all bytes [21]). Our AQM design attempts to notify only high-bandwidth "elephants" of congestion while allowing "slower" (and typically shorter) connections to remain unaware of incipient congestion. The motivation for this approach, borne out by our analysis of AQM schemes, is that providing early congestion notifications to mice only hurts their performance by forcing these short TCP connections to simply wait longer to transmit their last few segments. These short flows are often too short to have a meaningful transmission rate and are so short that slowing them down does not significantly reduce congestion. In contrast, providing early congestion notification to elephants can lead to abatement of congestion and more efficient use of the network. Our form of differential congestion notification, called DCN, significantly improves the performance of the vast majority of TCP transfers and provides response times, link utilizations, and loss ratios that are better than those of existing AQM schemes including those that also attempt to differentiate between flows based on their size and rate.

The remainder of the paper makes the case for differential congestion notification based on classification of flow-rate. Section 2 discusses previous related work in AQM schemes in general and in differential AQM specifically. Section 3 presents our DCN scheme. Section 4 explains our experimental evaluation methodology and Section 5 presents the results of a performance study of DCN and several prominent AQM schemes from the literature. The results are discussed in Section 6.

2. Background and Related Work

Several AQM designs have attempted to achieve fairness among flows or to control high-bandwidth flows. Here we give a description of the AQM designs most related to ours.

The Flow Random Early Drop (FRED) algorithm protects adaptive flows from unresponsive and greedy flows by implementing per-flow queueing limits [11]. The algorithm maintains state for each flow that currently has packets queued in the router. Each flow is allowed to have up to min_q but never more than max_q packets in the queue. Packets of flows that have more than min_q but less than max_q packets in the queue are probabilistically dropped. When the number of flows is large and a high-bandwidth flow consumes only a small fraction of the link capacity, FRED maintains a large queue. However, a large queue results in high delay. Furthermore, the algorithm becomes less efficient with a large queue since the search time for flow state is proportional to queue length.

The Stabilized Random Early Drop (SRED) algorithm controls the queue length around a queue threshold independent of the number of active connections [13]. The algorithm keeps the header of recent packet arrivals in a "zombie list." When a packet arrives, it is compared with a randomly chosen packet from the zombie list. If the two packets are of the same flow, a "hit" is declared. Otherwise, the packet header in the zombie list is probabilistically replaced by the header of the new packet. The number of active connections is estimated as the reciprocal of the average number of hits in a given interval (a large number of active connections results in a low probability of hits and vice versa). The drop probability of a packet is a function of the instantaneous queue length and the estimated number of active connections. Hits are also used to identify high-bandwidth flows. Packets of high-bandwidth flows are dropped with a higher probability than other packets.

The CHOKe algorithm heuristically detects and discriminates against high-bandwidth flows without maintaining per flow state [17]. The algorithm is based on the assumption that a high-bandwidth flow is likely to occupy a large amount of buffer space in the router. When a new packet arrives, CHOKe picks a random packet in the queue and compares that packet's header with the new packet's header. If both packets belong to the same flow, both are dropped, otherwise, the new packet is enqueued. As with FRED, CHOKe is not likely to work well on a high-speed link and in the presence of a large aggregate of flows.

The Stochastic Fair BLUE (SFB) algorithm detects and rate-limits unresponsive flows by using accounting bins that are organized hierarchically [4]. The bins are indexed by hash keys computed from a packet's IP addresses and port numbers and used to keep track of queue occupancy statistics of packets belonging to the bin. High-bandwidth flows can be easily identified because their bins' occupancy is always high. These high-bandwidth flows are then rate-limited. SFB works well when the number of high-bandwidth flows is small. When the number of high-bandwidth flows increases, more bins become occupied and low bandwidth flows that hash to these bins are incorrectly identified as high-bandwidth and penalized.

The Approximate Fairness through Differential Dropping (AFD) algorithm approximates fair bandwidth allocation by using a history of recent packet arrivals to estimate a flow's transmission rate [16]. AFD uses a control theoretic algorithm borrowed from PI [7] to estimate the "fair share" of bandwidth that a flow is allowed to send. Packets of a flow are marked or dropped with a probability that is a function of the flow's estimated sending rate. The algorithm uses a *shadow buffer* to store recent packet headers and uses these to estimate a flow's rate. The estimated rate of a flow is proportional to the number of that flow's headers in the shadow buffer. When a packet arrives, its header is copied to the shadow buffer with probability $1/s$, where s is the sampling interval, and another header is removed randomly from the shadow buffer. Note that while sampling reduces implementation overhead, it also reduces the accuracy in estimating flows' sending rate. This problem can be severe when most flows only send a few packets per RTT.

The RED with Preferential Dropping (RED-PD) algorithm provides protection for responsive flows by keeping state for just the high-bandwidth flows and preferentially dropping packets of these flows [12]. RED-PD uses the history of recent packet drops to identify and monitor high-bandwidth flows. The algorithm is based on the assumption that high-bandwidth flows also have a high number of packet drops in the drop history. Packets of a high-bandwidth flow are dropped with a higher probability than other packets. After being identified as high-bandwidth, a flow is monitored until it does not experience any packet drop in a certain time period. The absence of packet drops of a high-bandwidth flow in the drop history indicates that the flow has likely reduced its sending rate. In this case, the flow is deleted from the list of monitored flows.

The RIO-PS scheme (Red with In and Out with Preferential treatment to Short flows) gives preferential treatment to short flows at bottleneck links [6]. With preferential treatment, short flows experience a lower drop-rate than long flows and can thus avoid timeouts. In RIO-PS, edge routers maintain per-flow state for flows entering the network. The first few packets of a flow are marked as "short" or "in." Subsequent packets of that flow are marked as "long" or "out." Core routers use the standard RIO algorithm [2] and drop long or out packets with a higher probability than short or in packets.

Our DCN algorithm, described next, is an amalgam of existing AQM mechanisms. Like AFD it uses a control theoretic algorithm for selecting packets to drop and like RED-PD it maintains state for only the suspected high-bandwidth flows. However, we show empirically that our particular choice and construction of mechanisms results in better application and network performance than is possible with existing differential and non-differential AQM designs.

3. The DCN Algorithm

The design of DCN is based on the observation that on many networks, a small number of flows produce a large percentage of the traffic. For example, for the web traffic we have used to evaluate AQM designs, Figure 1 shows the cumulative distribution function (CDF) of the empirical distribution of HTTP response sizes [18]. Figure 2 shows a CDF of the percentage of total bytes transferred in an hour-long experiment as a function of HTTP response size. Together, these figures show that while approximately 90% of web responses are 10,000 bytes or less, these responses account for less than 25% of the bytes transferred during an experiment. Moreover, responses greater than 1 megabyte make up less than 1% of all responses, but account for 25% of the total bytes.

These data suggest that providing early congestion notification to flows carrying responses consisting of a few TCP segments (*e.g.*, flows of 2,000-3,000 bytes, approximately 70% of all flows), would have little effect on congestion. This is because these flows comprise only 6-8% of the total bytes and because these flows are too short to have a transmission rate that is adaptable. By the time they receive a congestion notification signal they have either already completed or have only one segment remaining to be sent. Furthermore, since short flows have a small congestion window, they have to resort to timeouts when experiencing a packet loss. Thus giving these flows a congestion signal does not significantly reduce congestion and can only hurt the flows' performance by delaying their completion. In contrast, high-bandwidth flows carrying large responses are capable of reducing their transmission rate and hence can have an impact on congestion. Unlike short flows, high-bandwidth flows do not have to resort to timeouts and instead can use TCP mechanisms for fast retransmission and fast recovery to recover from their packet losses. Our approach will also police high-bandwidth non-TCP or non-compliant TCP flows that do not reduce their transmission rate when congestion occurs.

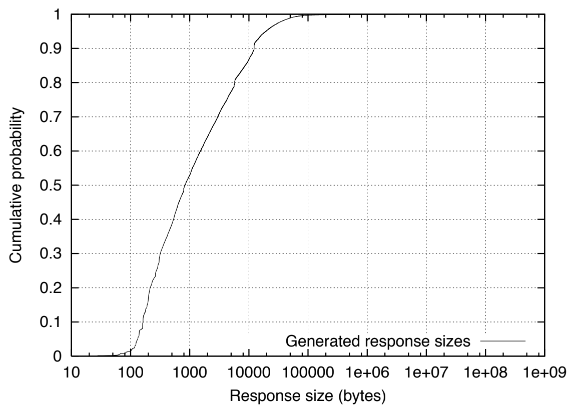


Figure 1: CDF of generated response sizes.

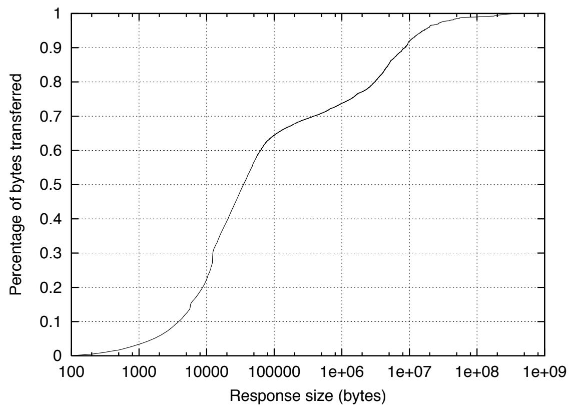


Figure 2: CDF of percentage of total bytes transferred as a function of response sizes.

Our observation about traffic characteristics is also confirmed by other studies of Internet traffic. For example, Zhang *et al.* found that small flows (100KB or less) accounted for at least 84% of all flows, but carried less than 15% of all bytes [21]. They also found that large flows accounted for a small fraction of the number of flows, but carried most of the bytes. Moreover, the flows that are "large" and "fast" (*i.e.*, high-bandwidth, greater than 10KB/sec) account for less than 10% of all flows, but carrying more than 80% of all the bytes.

The Differential Congestion Notification (DCN) scheme is based on identifying these "large" and "fast" flows, and providing congestion notification to only them. While the idea is simple, the challenge is to design an algorithm with minimal state requirements to identify the few long-lived, high-bandwidth flows from a large aggregate of flows and provide them with a congestion signal when appropriate. An important dimension of this problem is that of all the flows carrying large responses, we most want to signal flows that are also transmitting at a high-rate. These are the flows that are consuming the most bandwidth and hence will produce the greatest effect when they reduce their rate. Additionally, we must ensure that flows receiving negative differential treatment are not subject to undue starvation.

Our DCN AQM design has two main components: identification of high-bandwidth flows and a decision procedure for determining when early congestion notification is in order.

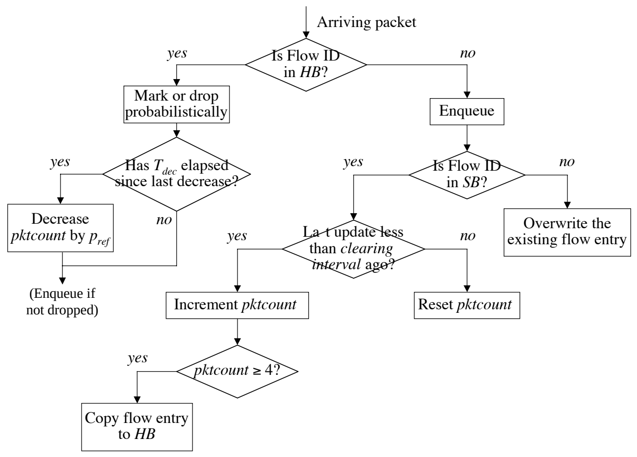
3.1 Identifying High-bandwidth Flows

Our approach to identifying high-bandwidth, long-lived flows is based on the idea that packets of high-bandwidth flows are closely paced (*i.e.*, their interarrival times are short) [3]. DCN tracks the number of packets that have been recently seen from each flow. If this count exceeds a threshold, the flow is considered to be a "long-lived and high-bandwidth" flow. The flow's rate is then monitored and its packets are eligible for dropping. As long as a flow remains classified as high-bandwidth, it remains eligible for dropping. If a flow reduces its transmission rate, it is removed from the list of monitored flows and is no longer eligible for dropping.

DCN uses two hash tables for classifying flows: HB ("high bandwidth") and SB ("scoreboard"). The HB table tracks flows that are considered high-bandwidth and stores each flow's flow ID (IP addressing 5-tuple) and the count of the number of forwarded packets. The SB table tracks a fixed number of flows not stored in HB. For these flows SB stores their flow ID and "recent" forwarded packet count.

When a packet arrives at a DCN router, the HB table is checked to see if this packet belongs to a high-bandwidth flow. If the packet's flow is found in HB, then it is handled as described below. If the packet's flow ID is not in HB, then the packet is enqueued and its flow is tested to see if it should be entered into HB. The SB table is searched for the flow's ID. If the flow ID is not present, it is added to SB.[1] If

[1] If a flow ID hashes to an entry in SB for another flow, then the new flow overwrites the entry for the previous flow. This ensures that all SB operations can be performed in constant time. Thus, SB stores the packet counts for all currently active non-high-bandwidth flows, modulo hash function collisions on flow IDs.



Arriving packet
Is Flow ID in HB?
yes
no
Mark or drop probabilistically
Enqueue
Has T_{dec} elapsed since last decrease?
yes
no
Decrease pktcount by p_{ref}
(Enqueue if not dropped)
Is Flow ID in SB?
yes
no
Overwrite the existing flow entry
Last update less than clearing interval ago?
yes
no
Increment pktcount
Reset pktcount
pktcount ≥ 4?
yes
Copy flow entry to HB

Figure 3: High-level DCN flowchart.

the flow ID is present in SB, the flow's packet count is incremented.

A flow is classified as long-lived and high-bandwidth if the number of packets from the flow arriving within a "clearing interval," exceeds a threshold. Once the flow's packet count in SB has been incremented, if the count exceeds the threshold, the flow's entry in SB is added to HB.[2] If no packets have been received for the flow within a clearing interval, the flow's packet count is reset to 0.

A high-level flow chart of the DCN algorithm is given in Figure 3. All operations on the SB table are performed in $O(1)$ time. Since the number of flows identified as high-bandwidth is small (*e.g.*, ~2000 for traffic generated during experiments reported herein), hash collisions in HB are rare for a table size of a few thousand entries. Thus, operations on the HB table are also usually executed in $O(1)$ time.

3.2 Early Congestion Notification

Packets from a high-bandwidth flow are dropped with a probability $1 - p_{ref} / pktcount$, where *pktcount* is the number of packets from that flow that have arrived at the router within a period of T_{dec}, and p_{ref} is the current "fair share" of a flow on a congested link. When congestion is suspected in the router we target high-bandwidth flows for dropping in proportion to their deviation from their fair share (p_{ref} packets within an interval T_{dec}) [16, 19].

DCN uses a simple control theoretic algorithm based on the well-known proportional integral controller to compute p_{ref}. The instantaneous length of the queue in the router is periodically sampled with period T_{update}. A flow's fair share of the queue at the k^{th} sampling period is given by:

$$p_{ref}(kT_{update}) = p_{ref}((k-1)T_{update}) + a \times (q(kT_{update}) - q_{ref}) - b \times (q((k-1)T_{update}) - q_{ref})$$

[2] If a collision occurs in HB when trying to insert the new flow, then a hash chain is used to locate a free table entry.

where a and b, $a < b$, are control coefficients (constants) that depend on the average number of flows and the average RTT of flows (see [7] for a discussion), $q()$ is the length of the queue at a given time, and q_{ref} is a target queue length value for the controller. Since $a < b$, p_{ref} decreases when the queue length is larger than q_{ref} (an indication of congestion) and hence packets from high-bandwidth flows are dropped with a high probability. When congestion abates and the queue length drops below q_{ref}, p_{ref} increases and the probability of dropping becomes low. Pan *et al.* and Misra *et al.* use the same equation in the design of AFD and PI respectively [16, 7].

The flow ID of a high-bandwidth flow is kept in the HB table as long as the flow's counter *pktcount* is positive. After each interval T_{dec}, the counter *pktcount* is decreased by p_{ref}. If a high-bandwidth flow's packet count becomes negative, the flow is deleted from HB. We set T_{dec} to 800 *ms* in our experiments because the maximum RTT in our network can be up to 400 *ms*. Furthermore, we want to avoid situations where a new high-bandwidth flow is detected at the end of an interval T_{dec} and immediately removed from HB. We experimented with different parameter settings for a, b, T_{update}, and T_{dec} and here report only the results for our empirically determined best parameter settings.

4. Experimental Methodology

To evaluate DCN we ran experiments in the testbed network described in [10]. The network, illustrated in Figure 4, emulates a peering link between two Internet service provider (ISP) networks. The testbed consists of approximately 50 Intel processor based machines running FreeBSD 4.5. Machines at the edge of the network execute one of a number of synthetic traffic generation programs described below. These machines have 100 Mbps Ethernet interfaces and are attached to switched VLANs with both 100 Mbps and 1 Gbps ports on 10/100/1000 Ethernet switches. At the core of this network are two router machines running the ALTQ extensions to FreeBSD [9]. ALTQ extends IP-output queuing at the network interfaces to include alternative queue-management disciplines. We used the ALTQ infrastructure to implement DCN, AFD, RIO-PS, and PI.

Each router has sufficient network interfaces to create either a point-to-point 100 Mbps or 1 Gbps Ethernet network between the two routers. The Gigabit Ethernet network is used to conduct calibration experiments to benchmark the traffic generators on an unloaded network. To evaluate DCN and compare its performance to other AQM schemes, we create a bottleneck between the routers by altering the (static) routes between the routers so that all traffic flowing in each direction uses a separate 100 Mbps Ethernet segment. This setup allows us to emulate the full-duplex behavior of a typical wide-area network link.

So that we can emulate flows that traverse a longer network path than the one in our testbed, we use a locally-modified

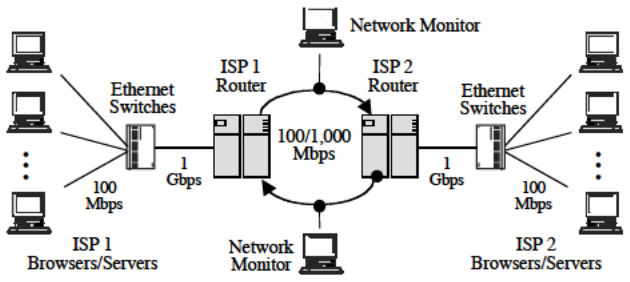


Figure 4: Experimental network setup.

version of *dummynet* [8] to configure out-bound packet delays on machines on the edge of the network. These delays emulate different round-trip times on *each* TCP connection (thus giving *per-flow* delays). Our version of *dummynet* delays all packets from each flow by the same randomly-chosen minimum delay. The minimum delay in milliseconds assigned to each flow is sampled from a discrete uniform distribution on the range [10, 150] with a mean of 80 *ms*. The minimum and maximum values for this distribution were chosen to approximate a typical range of Internet round-trip times within the continental U.S. and the uniform distribution ensures a large variance in the values selected over this range.

A TCP window size of 16K bytes was used on the end systems because widely used OS platforms, *e.g.*, most versions of Windows, typically have default windows of 16K or less.

4.1 Synthetic Generation of TCP Traffic

Two synthetically generated TCP workloads will be used to evaluate DCN. The first is an HTTP workload derived from a large-scale analysis of web traffic [18]. Synthetic HTTP traffic is generated according to an application-level description of how the HTTP 1.0 and 1.1 protocols are used by web browsers and servers today. The specific model of synthetic web browsing is as described in [10], however, here we note that the model is quite detailed as it, for example, includes the use of persistent HTTP connections and distinguishes between web objects that are "top-level" (*e.g.*, HTML files) and objects that are embedded (*e.g.*, image files).

The second workload is based on a more general model of network traffic derived from measurements of the full mix of TCP connections present on Internet links. For the experiments here we emulate the traffic observed on an Internet 2 backbone link between Cleveland and Indianapolis [20]. Thus in addition to generating synthetic HTTP connections, this model will also generate synthetic FTP, SMTP, NNTP, and peer-to-peer connections. Details on this model can be found in [20].

For both workloads, end-to-end response times for TCP data exchanges will be our primary measure of performance. Response time is defined as the time interval necessary to complete the exchange of application-level data units be-

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 inter-face) 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.

WHAT WILL YOU BUILD? | **sales@docketalarm.com** | **1-866-77-FASTCASE**

fastcase® Smarter legal research.