Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures

Subhabrata Sen
AT&T Labs-Research
Florham Park, NJ 07932
sen@research.att.com

Oliver Spatscheck
AT&T Labs-Research
Florham Park, NJ 07932
spatsch@research.att.com

Dongmei Wang
AT&T Labs-Research
Florham Park, NJ 07932
mei@research.att.com

ABSTRACT

The ability to accurately identify the network traffic associated with different P2P applications is important to a broad range of network operations including application-specific traffic engineering, capacity planning, provisioning, service differentiation, etc. However, traditional traffic to higher-level application mapping techniques such as default server TCP or UDP network-port based disambiguation is highly inaccurate for some P2P applications.

In this paper, we provide an efficient approach for identifying the P2P application traffic through application level signatures. We first identify the application level signatures by examining some available documentations, and packet-level traces. We then utilize the identified signatures to develop online filters that can efficiently and accurately track the P2P traffic even on high-speed network links.

We examine the performance of our application-level identification approach using five popular P2P protocols. Our measurements show that our technique achieves less than 5% false positive and false negative ratios in most cases. We also show that our approach only requires the examination of the very first few packets (less than 10 packets) to identify a P2P connection, which makes our approach highly scalable. Our technique can significantly improve the P2P traffic volume estimates over what pure network port based approaches provide. For instance, we were able to identify 3 times as much traffic for the popular Kazaa P2P protocol, compared to the traditional port-based approach.

Categories and Subject Descriptors

C.2.3 [**Computer-Communication Networks**]: Network operations—*Network management, Network monitoring*; D.2.8 [**Software Engineering**]: Metrics—*Performance measures*

General Terms

Measurement, Performance, Design

Keywords

Traffic Analysis, P2P, Application-level Signatures, Online Application Classification

1. INTRODUCTION

Peer-to-peer (P2P) file sharing applications have dramatically grown in popularity over the past few years, and today constitute a significant share of the total traffic in many networks. These applications have proliferated in variety and have become increasingly sophisticated along a number of dimensions including increased scalability, more functionality, better search capabilities and download times, etc. In particular the newer generation P2P applications are incorporating various strategies to avoid detection.

Access networks as well as enterprise networks require the ability to accurately identify the different P2P applications and their associated network traffic, for a range of uses, including network operations and management, application-specific traffic engineering, capacity planning, provisioning, service differentiation and cost reduction. For example, enterprises would like to provide a degraded service (via rate-limiting, service differentiation, blocking) to P2P traffic to ensure good performance for enterprise critical applications, and/or enforce corporate rules guiding running of peer-to-peer. Broadband ISPs would like to limit the P2P traffic to limit the cost they are charged by upstream ISPs. All these require the capability to accurately identify P2P network traffic.

Application identification inside IP networks, in general, can be difficult. In an ideal situation, a network administrator would possess precise information on the applications running inside the network, along with unambiguous mappings between each application and its network traffic (e.g., by port numbers used, IP addresses sourcing and receiving the particular application data, etc.). However, in general, such information is rarely available, up-to-date or complete, and identifying either the applications or their associated traffic is a challenging proposition. In addition, traditional techniques like network port-based classification of applications have now become problematic. Although the earlier P2P systems mostly used default network ports for communication, we have found that substantial P2P traffic nowadays is transmitted over a large number of non-standard ports, making default port-based classification less accurate.

In this paper, we report on our exploration of online, in-network P2P application detection based on application signatures. The following are some key requirements for such an application-level filter. It must be accurate, have low overheads, and must be robust to effects like packet losses, asymmetric routing, etc. (details in Sections 2 and 3) that make it difficult/impossible for a monitoring point to observe all the application-level data in a connection flowing by.

We designed a real-time classification system which operates on individual packets in the middle of the network, and developed application-level signatures for a number of popular P2P applications. Our signatures can be used directly to monitor and filter P2P traffic.

Evaluations using large packet traces at different Internet loca-

WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-844-X/04/0005.

tions show that the individual signature-based classification (i) has good accuracy properties (low false positives and negatives), even in situations where not all packets in a connection are observed by the monitoring point, (ii) can scale to handle large traffic volumes in the order of several Gbps (GigaBits per second), and (iii) can significantly improve the P2P traffic volume estimates over what pure network port based approaches provide. Our filter has been successfully deployed and is currently running at multiple network monitoring locations.

A lot of existing research on P2P traffic characterization has only considered traffic on default network ports (e.g., [11, 18, 17]). A recent work [12] uses application signatures to characterize the workload of Kazaa downloads. But they do not provide any evaluation of accuracy, scalability or robustness features of their signature. Signature based traffic classification has been mainly performed in the context of network security such as intrusion and anomaly detection (e.g. [5, 4, 19, 14]) where one typically seeks to find a signature for an attack. In contrast our approach identifies P2P traffic for network planning and research purposes. This work, is therefore, more closely related to [8] which provides a set of heuristics and signatures to identify Internet chat traffic. There is also a large body of literature on extracting information from packet traces (e.g., [9]); however, none of these works provides and evaluates application layer P2P signatures.

The remainder of this paper is organized as follows. Section 2 highlights the issues involved in identifying P2P traffic in real time inside the network. Section 3 discusses some of the design choices we made in our approach. Section 4 derives the actual signatures used for P2P detection, and Section 5 describes our implementation of an online P2P application classifier using these signatures. Section 6 presents the evaluation setting, and Section 7 describes the evaluation results. Finally, Section 8 concludes the paper.

2. PROBLEM STATEMENT

We first outline some key requirements of any mapping technique for identifying traffic on high speed links inside the network.

Accuracy: The technique should have low false positives (identifying other traffic as peer-to-peer) and low false negatives (missing peer-to-peer traffic).

Scalability: The technique must be able to process large traffic volumes in the order of several hundred thousand to several million connections at a time, with good accuracy, and yet not be computationally expensive.

Robustness: Traffic measurement in the middle of the network has to deal with the effects of asymmetric routing (2 directions of a connection follow different paths), packet losses and reordering.

The above requirements indicate there are tradeoffs in terms of the level of accuracy, scalability and robustness that can be achieved. On one end of this spectrum is the current practice of TCP/UDP port number based application identification. Port number based application identification uses known TCP/UDP port numbers to identify traffic flows in the network. It is highly scalable since only the UDP/TCP port numbers have to be recorded to identify an application. It is also highly robust since a single packet is sufficient to make an application identification.

Unfortunately port number based application identification is becoming increasingly inaccurate in identifying P2P traffic. For example, we observed in our traffic traces that a large amount of Kazaa traffic is not using the default Kazaa port numbers most likely — we speculate — to avoid detection.

To address this problem we developed and evaluated a set of application layer signatures to improve the accuracy of P2P traffic detection. In particular this approach tries to determine common signatures in the TCP/UDP payload of P2P applications.

A key challenge in realizing such signatures is the lack of openly available reliable, complete, uptodate and standard protocol specifications. This is partly due to developmental history and partly a result of whether the protocols are open or proprietary. First, the protocols are mostly not standardized and they are evolving. For some protocols (e.g., Gnutella), there exists some documentation, but it is not complete, or uptodate. In addition, there are various different implementations of Gnutella clients which do not comply with the specifications in the available documentation, raising potential inter-operability issues. For a user, this will manifest itself in the form of sometimes poor search performance. For an application classifier to be accurate, it is important to identify signatures that span all the variants or at least the dominantly used ones. At the other end of the spectrum is a protocol like Kazaa, which is developed by a single organization and therefore exhibits a more homogeneous protocol deployment, but is a proprietary protocol with no authoritative protocol description openly available. Finally, just access to the protocol specification is not sufficient - we need signatures that conform to the design decisions outlined above.

Our approach to signature identification has involved combining information available documentation, with information gleaned from analysis of packet-level traces to develop potential signatures. Multiple iterations were used to evaluate the signatures against network traffic data to improve the accuracy and computation overheads.

3. DESIGN CHOICES

Our main goal is to derive application layer signatures for P2P protocols which achieve high accuracy and robustness while being able to apply them at least at Gigabit Ethernet speeds in real time. As we will discuss in Section 7 we achieved these goals by making the following high level design choices.

UDP versus TCP: P2P traffic in principle can flow over UDP and TCP. Since currently most P2P protocols transmitted their data via TCP we focus on signatures found within TCP based P2P traffic. Obviously our signatures could be extended to UDP if so desired.

Packets versus Streams: The P2P application layer signatures can be applied to individual TCP segments or to fully reassembled TCP connection data streams. The advantage of applying them to TCP data streams is that duplicate data has been removed and that signatures can match data which is transmitted in multiple TCP segments. However, the drawback of applying the signatures to TCP data streams is that the TCP segments have to be reassembled in real time on the monitoring device. In our current design we chose to apply the signatures to individual TCP segments which allows us to achieve higher speeds. We therefore focus on developing signatures that do not span multiple TCP packet boundaries. As we will demonstrate we still achieve high accuracy for the 5 applications with the signatures that we develop.

Location of Signature: Again to improve performance we focus on finding signatures which appear in the beginning of the file downloads. Using this approach allows us to focus our signature evaluation on the first few packets of a TCP connection. We will study how many packets our signatures required in Section 7.

Robustness to network effects: We also aim to develop signatures that can independently identify each direction of an application-level communication. This is to enhance the potential of identifying connections for which the filter does not observe one direction of the traffic (due to asymmetric network routing), or misses some signature-carrying packets in one or both directions (caused by either router-based load splitting [16] or other routing instabilities). Independent identification of each direction also serves to decrease the potential of misclassification, by either reinforcing the marking (if both directions identify the same application) or flagging a potential discord (if the 2 directions are identified with different applications). Note that for some usages, such as accounting for total P2P traffic or identifying if some P2P communication is being used, where it is more important to identify that some P2P communications is being used, the last potential (of multiple classifications of the directions) is not an issue.

Early Discard: For efficiency reasons, we shall consider both signatures that identify an application as well as those that indicate that a connection does not belong to an application. The latter category of signatures allows us to quickly identify packets that are not likely application packets, and thereby frees up resources for examining more promising candidates.

Signaling versus Transport: Since the bulk of P2P traffic is related to file downloads and not due to file searches (signaling) we chose to concentrate our efforts on identifying signatures for file downloads rather than the signaling part of P2P protocols.

4. P2P PROTOCOLS AND SIGNATURES

Historically in the client/server model content is stored on the server and all clients download content from the server. One drawback of this model is that if the server is overloaded, the server becomes the bottleneck. The P2P file sharing model addresses this problem by allowing peers to exchange content directly. To perform these file sharing tasks, all popular P2P protocols allow a random host to act as both a client and a server to its peers, even though some P2P protocols do not treat all hosts equally.

Typically the following two phases are involved if a requester desires to download content:

Signaling: During the signaling phase a client searches for the content and determines which peers are able and willing to provide the desired content. In many protocols this does not involve any direct communication with the peer which will eventually provide the content.

Download: In this phase the requester contacts one or multiple peers directly to download the desired content.

In addition to the two phases described above many P2P protocols also exchange keep-alive messages or synchronize the server lists between servers.

In the remainder of the paper we focus on the download phase of the five most popular P2P protocols (*Kazaa, Gnutella, eDonkey, DirectConnect*, and *BitTorrent*). We decided to only track the download phase since it allows us to capture the majority of P2P traffic. We will also only classify the first download in a TCP connection. This simplification is reasonable since it is highly unlikely that two different applications will share a single TCP connection. In the remainder of this Section we will discuss the signatures we discovered for these five protocols. Unless otherwise specified, all the identified signatures are case insensitive.

4.1 Gnutella protocol

Gnutella is a completely distributed protocol. In a Gnutella network, every client is a server and vice versa. Therefore the client and server are implemented in a single system, called *servent*. A servent connects to the Gnutella network through establishing a TCP connection to another servent on the network. Once a servent has connected successfully to the network, it communicates with other servents using Gnutella protocol descriptors for searching the network - this is the signaling phase of the protocol. The actual file download is achieved using a HTTP-like protocol between the requesting servent and a servent possessing the requested file.

To develop the Gnutella signature we inspected multiple Gnutella connections and observed that the request message for Gnutella TCP connection creation assumes following format:

```
GNUTELLA CONNECT/<protocol version string>\n\n
```

And the response message for Gnutella TCP connection creation assumes:

```
GNUTELLA OK\n\n
```

We also observed that there is an initial request-response handshake within each content download. In the download request the servent uses the following HTTP request headers:

```
GET /get/<File Index>/<File Name>
/HTTP/1.0 \r \n
Connection: Keep-Alive\r\n
Range: byte=0-\r\n
User-Agent: <Name>\r\n
\r\n
```

The reply message contains the following HTTP response headers:

```
HTTP 200 OK\r\n
Server: <Name>\r\n
Content-type: \r\n
Content-length: \r\n
\r\n
```

Based on these observations and performance consideration, we recommend the following signatures for identifying *Gnutella* data downloads:

- The first string following the TCP/IP header is ‘GNUTELLA’, ‘GET’, or ‘HTTP’.

- If the first string is ‘GET’ or ‘HTTP’, there must be a field with one of following strings:

  ```
  User-Agent: <Name>
  UserAgent: <Name>
  Server: <Name>
  ```

where $< Name >$ is one of the following: LimeWire, BearShare, Gnucleus, MorpheusOS, XoloX, MorpheusPE, gtk-gnutella, Acquisition, Mutella-0.4.1, MyNapster, Mutella-0.4.1, MyNapster, Mutella-0.4, Qtella, AquaLime, NapShare, Comeback, Go, PHEX, SwapNut, Mutella-0.4.0, Shareaza, Mutella-0.3.9b, Morpheus, FreeWire, Openext, Mutella-0.3.3, Phex.

Generally it is much cheaper to match a string with a fixed offset than a string with varying locations. Hence we include 'GET' and 'HTTP' here to help early discard the packets, which do not start with 'GNUTELLA', and also are non-HTTP packets. For robustness, we included the signatures for the request and response header. This way, we can identify Gnutella traffic even if we only see one direction of the traffic.

4.2 eDonkey protocol

An eDonkey network consists of clients and servers. Each client is connected to one main server via TCP. During the signaling phase, it first sends the search request to its main server. (Optionally, the client can send the search request directly to other servers via UDP - this is referred to as extended search in eDonkey.) To download a file subsequently from other clients, the client establishes connections to the other clients directly via TCP, then asks each client for different pieces of the file.

After examining eDonkey packets, we discovered that both signaling and downloading TCP packets have the following common eDonkey header directly following the TCP header:

```
 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8
+-+-+-+-+-+-+-+-+
|     Marker    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               packet Length (4 Bytes)                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Message type  |
+-+-+-+-+-+-+-+-+
```

where the marker value is always 0xe3 in hex, the packet length is specified in network byte order and the value is the byte length of the content of the eDonkey message excluding the marker 1 byte and the length field 4 bytes.

Utilizing these discoveries, we recommend the following signatures for identifying eDonkey packets:

For TCP signaling or handshaking data packets, we use two steps to identify eDonkey packets.

- The first byte after the IP+TCP header is the eDonkey marker.
- The number given by the next 4 bytes is equal to the size of the entire packet after excluding both the IP+TCP header bytes and 5 extra bytes.

Since the accuracy for identifying the P2P connections is proportional to the length of the signatures, we tend to include as many fields as we can so long as they do not increase the computational complexity significantly. Here both marker and length fields have a fixed offset, therefore the computational complexity is the same (O(1)) for matching one of them or both, but the accuracy is improved by 2^{32} times compared with matching the marker field alone.

We have also identified the signatures for UDP handshaking messages. However, since UDP is only used for extended searching, and is rare compared with TCP communications, we do not report it in this study.

4.3 DirectConnect Protocol

The *DirectConnect* network is composed of hubs, clients, and a single superhub with multiple servers. All of them listen on TCP port 411 to connect and exchange commands such as search request. Clients (peers) store files and respond to search requests for those files. The single superhub acts as a name service for all the hubs. All hubs register with the superhub and clients discover hubs by asking the superhub. Each of the clients has a username (a.k.a. nick). Normally the clients listen at port 412 for client connections. If the port 412 is already in use, clients will use ports 413, 414 and so on. *DirectConnect* uses TCP for client to server and client to client communication, while UDP is used for communication between servers. The TCP/UDP data is a series of commands or a public chat message. In this study, we focus on the TCP commands. The TCP commands are identified with following form:

```
$command_type  field1 field2 ...|
```

which starts with character '\$', and ends with character '|'. The list of valid command types for TCP communications are: MyNick, Lock, Key, Direction, GetListLen, ListLen, MaxedOut, Error, Send, Get, FileLength, Canceled, HubName, ValidateNick, ValidateDenide, GetPass, Mypass, BadPass, Version, Hello, Logedin, MyINFO, GetINFO, GetNickList, NickList, OpList, To, ConnectToMe, MultiConnectToMe, RevConnectToMe, Search, MultiSearch, SR, Kick, OpForceMove, ForceMove, Quit.

To improve the evaluation performance we evaluate this signature in the following two steps:

1. The first byte after the IP+TCP header is '\$', and the last byte of the packet is '|'.
2. Following the '\$', the string terminated by a space is one of the valid TCP commands listed above.

Although we are matching a list of strings which can be an expensive operation, we shall only perform the string match on packets which pass the first test.

4.4 BitTorrent Protocol

The *BitTorrent* network consists of clients and a centralized server. Clients connect to each other directly to send and receive portions of a single file. The central server (called a tracker) only coordinates the action of the clients, and manages connections. Unlike the protocols discussed above, the *BitTorrent* server is not responsible for locating the searching files for the clients, instead the *BitTorrent* network client locates a *torrent* file through the Web, and initiates the downloading by clicking on the hyperlink. Hence there is no signaling communication for searching in the *BitTorrent* network. To identify *BitTorrent* traffic, we focus on the downloading data packets between clients only since the communication between the client and server is negligible.

The communication between the clients starts with a handshake followed by a never-ending stream of length-prefixed messages. We discovered that the *BitTorrent* header of the handshake messages assumes following format:

```
<a character(1 byte)><a string(19 byte)>
```

The first byte is a fixed character with value '19', and the string value is 'BitTorrent protocol'. Based on this common header, we use following signatures for identifying *BitTorrent* traffic:

- The first byte in the TCP payload is the character 19 (0x13).
- The next 19 bytes match the string 'BitTorrent protocol'.

The signatures identified here are 20 bytes long with fixed locations, therefore they are very accurate and cost-effective.

4.5 Kazaa protocol

The *Kazaa* network is a distributed self-organized network. In a Kazaa network, clients with powerful connections, and with fast computers are automatically selected as Supernodes. Supernodes are local search hubs. Normal clients connect to their neighboring Supernodes to upload information about files that they share, and to perform searches. In turn Supernodes query each other to fulfill the search.

The request message in a Kazaa download contains the following HTTP request headers:

```
GET /.files HTTP/1.1\r\n
Host: IP address/port\r\n
UserAgent: KazaaClient\r\n
X-Kazaa-Username: \r\n
X-Kazaa-Network: KaZaA\r\n
X-Kazaa-IP: \r\n
X-Kazaa-SupernodeIP: \r\n
```

The Kazaa response contains the following HTTP response headers:

```
HTTP/1.1 200 OK\r\n
Content-Length: \r\n
Server: KazaaClient\r\n
X-Kazaa-Username: \r\n
X-Kazaa-Network: \r\n
X-Kazaa-IP: \r\n
X-Kazaa-SupernodeIP: \r\n
Content-Type: \r\n
```

For higher Kazaa version (v1.5 or higher), a peer may send an encrypted short message before it sends back above response. Note that both messages include a field called X-Kazaa-SupernodeIP. This field specifies the IP address of the supernode to which the peer is connected including the TCP/UDP supernode service port. This information could be used to identify signaling using flow records of all communication.

Using the special HTTP headers found in the Kazaa data download we recommend the following two steps to identify Kazaa downloads:

1. The string following the TCP/IP head is one of following: 'GET', and 'HTTP'.
2. There must be a field with string: X-Kazaa.

Similar to our Gnutella signatures we include 'GET' and 'HTTP' to early discard non-HTTP packets, so that we can avoid searching through the whole packet to match 'X-Kazaa' if the packet has a low probability to contain HTTP request or response headers.

5. SIGNATURE IMPLEMENTATION

As stated earlier we concentrate on P2P application detection in TCP traffic. In particular we decomposed our P2P signatures into fixed pattern matches at fixed offsets within a TCP payload and variable pattern matches with variable offset within a TCP payload. The fixed offset operation can be implemented cheaply whereas variable pattern matches are substantially more expensive.

To be able to execute the decomposed signatures on real network traffic we implemented them in the context of the Gigascope [7] high speed traffic monitor. In this section we will first discuss the issues involved in evaluating fixed and variable offset signatures and then discuss how we implement them in the context of Gigascope.

5.1 Fixed Offset Match

Implementing a fixed pattern match at a fixed offset within a TCP payload is rather trivial. The complexity of this operation in the worst case is the size of the pattern matched. Despite this simplicity it is useful to provide multiple library functions which perform this operation using slightly different parameters to allow for the easy implementation of diverse signatures. For example, in the context of P2P signatures the offset could be specified from the beginning or end of the TCP payload and the pattern matches could be a byte, a word in little endian byte order, a word in big endian byte order, or a string. Therefore, we implemented a library which provides the following functions:

byte_match_offset: returns true if a byte matches the byte in the TCP payload on a given offset. If the offset is negative it is calculated from the end of the TCP payload.

word_match_offset: similar to byte match offset, except that a word is compared. This function takes as additional argument a flag indicating the byte order of the data in the TCP payload.

string_match_offset: similar to byte match offset, except that a fixed length sequence of bytes (string) is compared.

5.2 Variable Offset Match

There are multiple ways to implement matches at variable offsets in an input stream that involve variable length strings. As discussed in Section 3 we decided to perform the matches on a per packet basis, trading off higher performance against matching strings which span multiple packets.

Using this approach all variable matches we need to perform can be expressed as a regular expression match over TCP payloads. For example, the Gnutella data download signature can be expressed as:

```
'^(Server:|User-Agent:)[ \t]*(LimeWire|
BearShare|Gnucleus|Morpheus|XoloX|
gtk-gnutella|Mutella|MyNapster|Qtella|
AquaLime|NapShare|Comback|PHEX|SwapNut|
FreeWire|Openext|Toadnode)'
```

Due to the fact that it is expensive to perform full regular expression matches over all TCP payloads we exploit the fact that the required regular expression matches are of a limited variety. In particular all of the signatures we need to evaluate can be expressed as stringset1.*stringset2 where stringset1 and stringset2 contain a list of possible strings. This allows us to use the following algorithms for our signatures:

- Standard regex (SR): This is the regular expression match function found in the standard c library on FreeBSD 4.7.
- AST regex (AR): Part of the AST library [10], this code is based on the Boyer Moore string search algorithm [6] extended to handle alternation of fixed strings. To search for an m character long string in a $n \geq m$ character sequence, the Boyer-Moore algorithm has worst case time complexity $O(m+n)$, but often runs in $O(n/m)$-time on natural-language text for small values of m.
- Karp-Rabin (KR): This is a probabilistic string matching technique [13] that compares the hash value of the pattern against the hash value of the sub text of a given search text. The worst case complexity of Karp-Rabin is $O(mn)$, but for many situations is often $O(m+n)$.

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.

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

fastcase®
Smarter legal research.