HTTP as the Narrow Waist of the Future Internet

Lucian Popa
U.C. Berkeley / ICSI

Ali Ghodsi
U.C. Berkeley

Ion Stoica
U.C. Berkeley

ABSTRACT

Over the past decade a variety of network architectures have been proposed to address IP's limitations in terms of flexible forwarding, security, and data distribution. Meanwhile, fueled by the explosive growth of video traffic and HTTP infrastructure (*e.g.*, CDNs, web caches), HTTP has became the de-facto protocol for deploying new services and applications. Given these developments, we argue that these architectures should be evaluated not only with respect to IP, but also with respect to HTTP, and that HTTP could be a fertile ground (more so than IP) for deploying the newly proposed functionalities. In this paper, we take a step in this direction, and find that HTTP already provides many of the desired properties for new Internet architectures. HTTP is a content centric protocol, provides middlebox support in the form of reverse and forward proxies, and leverages DNS to decouple names from addresses. We then investigate HTTP's limitations, and propose an extension, called S-GET that provides support for low-latency applications, such as VoIP and chat.

1. INTRODUCTION

During the past decade, a plethora of new Internet architectures have been proposed to address the shortcomings of IP in terms of flexibility, scale, and security (*e.g.*, [10, 20, 27, 28, 31, 42, 47, 49]). Some of these limitations have been traced to IP's inability to decouple the concepts of address and identity, its lack of explicit support for middleboxes, mobility, and content distribution.

Meanwhile, industry has been pushing through changes that are having a profound impact on the Internet. In particular, we are witnessing an explosive growth of HTTP traffic [29, 39]. This trend is driven by the prevalence of the existing HTTP infrastructure (*e.g.*, CDNs, HTTP proxies, and caches), the ease of deploying new functionality on the data path via reverse and forward proxies, and the ability of HTTP to penetrate corporate firewalls. In turn, the growth of HTTP traffic pushes infrastructure providers to expand their HTTP footprint, creating a positive feedback loop, which further accelerates HTTP traffic growth.

Hotnets '10, October 20–21, 2010, Monterey, CA, USA.

In this paper, we take this trend to its logical conclusion and consider the scenario where HTTP becomes the de facto "narrow waist" of the Internet—that is, the vast majority of traffic runs over HTTP instead of directly over IP, and HTTP itself might run on top of network layers other than IP.

Given such scenario, we argue that we should start evaluating HTTP with respect to the existing Internet architecture proposals, and, in the process, answer the following questions: What are the properties aimed by these architectures that HTTP already provides, and what are the ones it does not? What are HTTP's main drawbacks? Can these drawbacks be addressed by extending HTTP, or are they the result of fundamental limitations of HTTP?

In this paper, we find that HTTP addresses many of the limitations of IP, limitations which have been the target of several recently proposed network architectures. First, HTTP is a content-centric protocol, as each HTTP method specifies the name of the resource (content) it operates on. This allows proxies along a request's path to cache the content, or to redirect the request to the closest or least loaded server storing a copy of the content. Building a content-centric network, albeit at the network layer, has been one of the main goals of recently proposed architectures such as DONA [28] and CCN [27].

Second, HTTP supports both reverse and forward proxies [3, 16]. This allows senders and receivers to add middleboxes on the data path, functionality proposed by many to be incorporated in the Internet [10, 20, 28, 42, 47].

Third, HTTP uses DNS names to refer to content. This enables data "mobility" in the context of a DNS name, and basic anycast functionality via DNS round-robin or modified DNS resolution (typically done by CDNs).

However, HTTP is not without drawbacks. HTTP does not address network-level DoS attacks, nor is HTTP a good fit for low-latency services, such as VoIP, chat, and real-time applications. To alleviate some of these drawbacks, we propose a new HTTP GET method, called S-GET, which enables *datagram services* on top of HTTP. S-GET can be used to provide low-latency communication between clients, end-host mobility, and implement delay tolerant networks. Moreover, the HTTP datagram communication model inherently enables users to be default-off, shielding them from unwanted traffic. We show that such an extension achieves high throughput and incurs low overhead.

We are not the first to argue that the narrow waist of the Internet is changing. Several previous works have argued that the transport layer should be incorporated in the Internet's narrow waist [17, 36]. However, we believe that HTTP

would represent a far more drastic change of the narrow waist, as it provides a data centric abstraction which is fundamentally different from the unicast and multicast abstractions, and there is already a tremendous HTTP infrastructure (*e.g.*, caches, proxies, servers) deeply integrated in the Internet fabric.

While in this paper we argue that HTTP achieves many of the properties targeted by the recently proposed Internet architectures, we emphasize that it is not our intention to underestimate the importance of research on clean-slate designs. On the contrary, we strongly believe that such research is necessary to better understand the limitations of today's Internet, and explore the solution space. Moreover, we do not argue that HTTP is the right layer to implement these properties. Instead, in this paper we merely take note of the seemingly inevitable trend of HTTP becoming the de facto protocol used by new services and applications, and argue that: (1) HTTP could be a fertile ground (more so than IP) for deploying the solutions and techniques provided by the clean slate proposals, and (2) new functionalities proposed at the IP layer (already implemented by HTTP or not) should consider their interaction with HTTP and should be evaluated with respect to HTTP.

2. HTTP TAKES OVER THE WORLD (AGAIN)

With the advent of the web in mid-90's, HTTP became the dominant traffic in the Internet [33]. This led to the rapid development of the Internet infrastructure to support HTTP traffic. Content distribution networks as well as HTTP proxies greatly increased the distribution scale and the availability of the HTTP content.

However, the emergence of video and audio traffic at the end of 90's and the beginning of this decade challenged the dominance of HTTP. Real Networks, Microsoft, and Adobe employed streaming protocols, such as RTSP and RTMP, to deliver media content. More recently, peer-to-peer technologies saw an explosive growth, with their traffic being dominated by video and audio content. These trends seemed to indicate that HTTP would lose its dominant position in the Internet traffic.

However, today we are witnessing a resurgence of HTTP traffic. Ironically, this resurgence has been driven by the growing popularity of video traffic (in a recent report, Cisco forecasts that by 2013, 90% of the consumer traffic will be video [4, 38]). To sustain such growth, the content providers and aggregators have recently turned their attention to HTTP for video distribution.

Several companies, including Move Networks and Swarmcast, have pioneered *HTTP chunking*, which enables the delivery of video and audio over HTTP instead of traditional streaming protocols. The basic idea is to chunk a video stream into blocks of a few seconds each, and then distribute these blocks as individual files by leveraging existing CDNs and HTTP proxies. In turn, a client downloads the chunks, stitches them together, and plays the original stream.

HTTP chunking has several advantages over traditional streaming protocols. First, it increases the distribution scale and reduces the cost, as CDNs have more HTTP servers than streaming servers, and they do not incur licensing costs for the HTTP servers (these servers are typically based on open-source software, unlike the streaming servers). Furthermore, using HTTP to distribute video can leverage the HTTP caching proxies deployed by ISPs and enterprises. Second, it improves availability: if an HTTP server fails, the client can mask such a failure by requesting the subsequent chunks from a different server or CDN. Third, it improves quality, as a client can request multiple chunks simultaneously, which leads to aggregating the throughput of multiple TCP connections. In contrast, traditional streaming protocols use one TCP connection for data transfer. Fourth, it improves penetration. Unlike streaming protocols such as RTPS and RTMP that are blocked by some firewalls, HTTP traffic is almost universally allowed.

These advantages have pushed HTTP chunking to the forefront of distribution technologies for both video-on-demand (VoD) and live streaming content. Indeed, Microsoft used HTTP chunking to stream the Beijing Olympics for `NBC.com`, and used a second generation HTTP-based technology, called Smooth Streaming [30], to stream the Vancouver Winter Olympic Games. Apple uses an HTTP-chunking solution to stream video to the iPhone, and Adobe, which dominates the video market, recently announced their HTTP-based solution [5].

HTTP traffic is also increasing at the expense of peer-to-peer (P2P) traffic, as indicated by recent reports [4, 32, 38]. The promise of P2P has been to provide highly scalable, low cost (in some cases free) content distribution. However, the CDN delivery cost has decreased dramatically in the past few years (*e.g.*, by a factor of 10 between 2006 and 2010), which had considerably decreased the appeal of P2P distribution. Thus, it should come as no surprise, that today virtually all major content providers, including Youtube, Hulu and MLB use CDNs instead of P2P for content delivery. With the advent of HTTP chunking and with a continuous expansion of the HTTP infrastructure, we expect that this trend will only intensify.

We therefore project that HTTP traffic will dominate (at least in volume) Internet traffic.

3. HTTP VS. THE BRAVE NEW WORLD

Given the rapid growth of HTTP, we argue that the proposals for new Internet architectures should be evaluated, not only in the context of IP, but also in the context of HTTP. Next, we contrast HTTP to several of these research proposals.

For the clarity of the comparison, we classify the numerous research proposals to improve the Internet architecture into five categories[1]: (a) proposals to transform the Internet into a content centric network, *e.g.*, [10, 13, 27, 28]; (b) proposals to enable the explicit use of middleboxes, *e.g.*, [10, 20, 28, 42, 47]; (c) proposals to enable more flexible communication patterns, such as mobility, anycast and multicast,

[1]Note that some proposals belong to multiple categories.

Property	HTTP support
Content centric network [10, 13, 27, 28]	Yes, via named resources and caching
Middlebox support [10, 20, 28, 34, 40, 42, 47]	Yes, via proxies
Additional communication patterns – Mobility [10, 25, 42, 47] – Multicast [37] – Anycast [27, 28, 42, 47] – Multipath / Multihomed [10, 47, 48] – DTN [15]	Yes, via proposed S-GET extension (see §4), caching, CDNs and DNS
Security extensions – Data Authenticity [27, 28] – DoS Protection [9, 26, 49]	Yes/Partial via proposed S-GET extension (see §4), HTTPS and adding authentication field in header (see [12, 18, 35])
Routing policy extensions [19, 22–24, 40]	No, but can be implemented mostly independently of HTTP

Table 1: HTTP's support for the main types the functionalities proposed in the Internet architecture literature.

e.g., [10, 15, 20, 25, 31, 34, 37, 42, 44, 47]; (d) proposals to increase network security, *e.g.*, [**?**, 8, 9, 11, 20, 26, 40, 49]; and (e) proposals to extend the Internet routing policies and add QoS policies, *e.g.*, [19, 22–24, 40, 48].

Table 1 presents a summary of our findings, which we discuss in more detail next.

Content centric networks: Several research proposals have advocated for content centric network architectures [13, 27, 28], where the communication revolves around data instead of end-points. HTTP is already a content centric protocol, as HTTP requests deal primarily with retrieving, storing, and updating content. In particular, each HTTP request contains the name of the content, similarly to the way the packets in these proposals contain the content name [13, 27, 28]. Moreover, the massive HTTP caching infrastructure deployed in the Internet exhibits another common property of the aforementioned architectures, the pervasive use of caches.

One aspect in which HTTP and some of the content centric proposals differ is naming: while some of the proposals use global and semantic-free names [28], HTTP binds content names to DNS names (in the form of URLs). We note that content-based architectures are likely to require a resolution mechanism to translate human readable names into content identifiers [27, 41, 46], though that service might be outside of the architecture [28]. HTTP uses DNS for this purpose and names content directly with human readable names (similar to CCN [27]). There are five concerns associated with DNS names: (i) persistence, (ii) latency, (iii) fast updates, (iv) availability, and (v) security. We note, however, that recent developments have alleviated these concerns to some degree. The emergence of third party DNS infrastructures, such as OpenDNS and Google Public DNS, improve latency and availability. In addition, they alleviate the persistence concern, as a user can obtain names from these services, instead of her own organization, and preserve these names when moving from an organization to another. Dynamic DNS has been proposed to address the fast update problem [45], and more recently, OpenDNS has provided low latency updates [2]. Still, despite these developments, more research is needed to fully address the persistence and fast updates of DNS names. Finally, DNSSEC addresses the security of the name resolution (we discuss content integrity later in this section).

Explicit middlebox support: Numerous proposals argue for explicit middlebox support in the Internet, *e.g.*, [10, 20, 28, 42, 47]. Since IP does not expose a middlebox-like abstraction to end-hosts, this leaves one no choice but to physically place the middlebox on the IP data path. This operation is not only complex, but also fails to guarantee correctness in the presence of IP path changes. Furthermore, implementing sophisticated middlebox functionality (*e.g.*, caching, web acceleration) that changes the number, the size, or the content of the packets may require to violate the end-to-end semantics of IP.

In contrast, HTTP does provide support for middleboxes via explicit forward proxies and via reverse proxies [16]. Clients and servers can leverage these proxies to insert a variety of functionalities on the data path, including caching, web acceleration, content filtering, intrusion detection, load balancing, and anonymization.[2]

Flexible communication: During the past two decades, many solutions have been proposed to extend IP to support mobility, multicast, anycast, multi-path and delay tolerant networks (DTN), [15, 25, 34, 37, 42, 44, 47, 48]. HTTP can directly offer some of these services. As we have discussed in § 2, HTTP has been successfully used to provide large scale single-source multicast (*e.g.*, the live transmissions of the last two Olympic games, and the 2009 US presidential inauguration). HTTP can provide anycast by leveraging the DNS anycast functionality, or through reverse HTTP proxies, such as Squid [3]. Through DNS updates, HTTP supports single host mobility. However, HTTP does not support simultaneous end-host mobility, or multi-path communications.

Security: A large body of research is concerned with improving the security of the Internet by providing defense against DoS attacks [9, 26, 49], or by ensuring data authenticity and integrity [27, 28]. While HTTP does not protect against IP-level DoS attacks, the widespread caching infrastructure and proprietary HTTP-based DoS mitigation products [1] do improve the status quo of DoS protection. On the other hand, data authenticity and integrity guarantees can be implemented within the existing HTTP in several ways: by embedding content-hashes and digital signatures in the HTTP header [12, 18], by using self-certified URLs, or by using HTTPS (see our extended TR [35]).

Routing and QoS: Many proposals have aimed to improve the robustness, efficiency, and security of inter/intra domain routing, as well as to provide QoS guarantees [19, 22–24]. With few exceptions[3], HTTP does not address any of these

[2]It is also possible to chain multiple HTTP proxies (*e.g.*, the cache_peer option in Squid [3]).

[3]HTTP can be used to implement loose source routing via multiple proxies by using the CONNECT method. Since each proxy can control which other proxies it is willing to relay connections

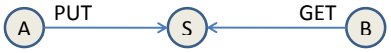

Figure 1: Two HTTP clients A and B exchanging data through an HTTP server S, which acts as a relay

challenges directly. Thus, this remains an area of research largely unaffected by HTTP.

In summary (see Table 1), HTTP already provides support for middleboxes and content-centric networking, and partial support for mobility, anycast and multicast. Many security properties can be achieved by using HTTPS or by leveraging the flexibility of the the HTTP header to implement new security mechanisms.

Despite its many strengths, HTTP is not perfect. First, HTTP is a pull oriented protocol where receivers need to explicitly ask for new data. As we will discuss in the next section, this is not a good fit for datagram and connection oriented services, such as VoIP, video-conferencing, and real-time applications. Second, HTTP incurs a non-trivial overhead when compared to IP. Third, as discussed in this section, HTTP does not provide or provides limited support for QoS, network layer DoS protection, and naming persistence. In the reminder of this paper, we focus on addressing the first limitation, and leave the others for future work.

4. DATAGRAM SERVICES OVER HTTP

In this section we describe a datagram communication model on top of HTTP. In this paper, we use liberally the term of "datagram" to denote a communication service in which two or more clients exchange application data units (ADUs) and does not provide end-to-end reliability or in-order delivery. The main goal of this datagram service is to support low-latency applications such as VoIP, chat, and video conferencing, which are hard to implement in the current client-server model employed by HTTP.

The natural approach to implement a datagram service on top of today's HTTP is for a sender, A, to publish data to an HTTP server and for receiver B to get the piece of data from the server (see Fig. 1). However, this pull communication abstraction is not appropriate for low-latency applications, as the receiver (client) does not know when new data has become available. The only way to reduce the latency from the time A publishes the content to the time B fetches it is for B to periodically check for the content availability as often as possible. Assuming the receiver checks for content every T ms, the end-to-end latency may exceed T ms. If an application wants to achieve an end-to-end latency on par with cross country latencies, it needs to pull about every 50 ms. Such high pulling frequencies can be prohibitive both for the client and the server, especially when the receiver does not know *when* and *which* sender starts sending data.

To reduce the end-to-end delay, we argue that HTTP should also support a push communication abstraction, *i.e.*, provide a mechanism that allows GET requests for content that is not yet available at the server. In particular, we propose to extend HTTP with a new type of GET request, called *Subscribe-GET* (S-GET), that "waits" for content at the server, instead of having the server return an error when content is not available.

Note that while one could continue to use IP for latency-sensitive applications, providing a datagram service over HTTP has the advantage of leveraging all the HTTP benefits, as we discuss in Sec. §4.2.

4.1 Subscribe-GET (S-GET)

The format of an S-GET request is similar to that of a traditional GET. However, unlike GET requests, HTTP servers *store* S-GET requests up to an expiration timeout associated to the request. As long as the S-GET is stored at the server, any updates (through PUTs) to the URI of the S-GET are sent to the client that issued the S-GET. A server removes an S-GET request only after the timeout expires. Each update is sent to the client through a regular HTTP response.

S-GET only returns content published after the S-GET has been received by the server. Hence, upon receiving an S-GET request for a URI, the server does not match it against the content already stored under that URI. Proxies never cache the content returned by S-GET.

Note that S-GET provides the abstraction of a named pipe, where the client opens a pipe through an S-GET request, and the sender writes data (using either POST or PUT) to the pipe. The use of S-GET represents just another instantiation of the publish-subscribe paradigm, which has been strongly advocated by previous work [14].

S-GET represents a departure from HTTP's stateless model, raising concerns about performance, memory requirements and failure resilience. Our evaluation shows that the performance impact of S-GET is not significant. S-GET also uses soft state, which leaves the failure semantics of HTTP based protocols largely unchanged.

The S-GET request contains the desired timeout as a header attribute. For security purposes, the server may not accept large timeout values, in which case it returns an error response containing the maximum allowed timeout. To extend their duration, S-GET requests need to be "refreshed" before their expiration.

S-GET is useful beyond the datagram abstraction we have discussed so far. Many websites today attempt to implement "HTTP push", a similar functionality where the server sends data to the client without the need for the client to query the data explicitly. These solutions are implemented on top of HTTP, typically using CGI or Javascript scripts [6, 7]. In this context, the S-GET primitive can be seen as an effort to standardize these ad-hoc mechanisms. One advantage of standardizing S-GET is that proxies would appropriately cache and handle S-GETs (see the next paragraph). In addition, ad-hoc methods suffer from portability issues, since different clients/servers use their own implementations and APIs.

Caching Proxies: Today, proxies cache GET *responses* and deliver them to subsequent GETs for that URI. To support

to, security concerns traditionally associated with IP loose source routing are mitigated.

S-GET, proxies should cache S-GET *requests* rather than responses, *i.e.*, the dual of what is done today. If the proxy receives multiple S-GETs for the same URI, it should only forward the first to the server. The rest of the S-GETs could be satisfied by relaying content that is anyway being fetched for the first request; this behavior is similar to that of an IP multicast router.

Usage Example: Consider a point-to-point communication between two hosts A and B, where A sends B a sequence of ADUs. In the HTTP datagram model (see Fig. 1) B registers an S-GET for a URI and A sends all its ADUs to that URI through HTTP PUTs. For example, the URI can be `S/from/A/to/B/x`, where S is the server's host name, and x is for distinguishing different sessions between two hosts (similar to TCP ports). Since publishing a new ADU through PUT modifies the content at `S/from/A/to/B/x`, the server will forward each ADU to B.

4.2 Benefits of HTTP Datagrams

Mobility: The proposed datagram communication on top of HTTP enables both end-hosts to move at the same time, thus supporting simultaneous mobility.

Multicast and Large Scale Data Distribution: Multi-sender multicast can be implemented by having each participant register an S-GET request for the same URI, which plays the role of an IP multicast address. This implements an open group multicast model similar to IP multicast, but access control can be implemented on top (see [35] for details).

HTTP has already been proven highly successful for streaming data to large audiences (see § 2 and § 3). The proposed S-GET method further improves this service for *live* streaming, in which the availability of new chunks needs to be signaled to receivers. S-GET can then be used as a control channel for signaling the availability of new chunks.

Multi-homing and Multiple paths: Multi-homed clients can setup different S-GET receive channels for each interface, using multiple links for one communication. To further increase reliability and throughput, clients can use multiple intermediary servers, *e.g.*, receiver B sends S-GET requests to servers S1 and S2 and sender A alternates between sending to S1 and S2. This way, even clients with a single interface can use multiple paths.

Delay Tolerant Networking (DTN): The high level idea for supporting DTN is that during disconnection periods, the intermediary HTTP servers act as a buffer to store ADUs. Our DTN solution uses a combination of S-GET and GET to fetch and resume data transfers (see our TR [35]).

NAT/Firewall Penetration and Default-Off: All HTTP requests are client-initiated and hence HTTP datagrams traverse today's NATs and firewalls. More fundamentally, HTTP datagrams would enable an architecture with two types of entities, clients, which are default-off and servers, which are reachable by everyone. This emulates the architecture envisioned in [21], with the addition that in our proposal, clients can still communicate among themselves. This way, DDoS attacks can be alleviated since resource-weak hosts can be default-off and receive data through HTTP datagram channels opened at multiple resourceful servers and data centers.

4.3 Other Considerations

Server Selection: The placement of the HTTP server has crucial impact on the performance of the HTTP datagram service. For good performance, each host may chose a nearby server to receive messages, similar to $i3$ [42]. This way, end-hosts can avoid the risk of picking a server far away from both the sender and the receiver, which may lead to inefficient routing. It has been previously observed that communicating through a one-hop overlay using a carefully selected Akamai server often outperforms a direct connection between two end-points [43]. Thus, CDNs are already in a good position to offer HTTP datagram services. For other considerations such as DNS names corresponding to multiple IP addresses see our TR [35].

Connection Establishment: HTTP datagrams can be used to provide functionalities provided by traditional connection-oriented services: `listen`, `connect`, and reliability. Listen can be implemented using S-GET on a *listen URI*, *e.g.*, `S/listen/B` and a connection can be set up by having one URI per unidirectional channel, *e.g.*, `S/from/B/to/A`. Please see our TR for details [35].

Security: The HTTP datagram service faces three types of attacks: impersonation, eavesdropping and DoS attacks. First, a malicious node could impersonate a sender by putting ADUs at the receiver's S-GET URI. Second, an attacker could eavesdrop by issuing an S-GET request to the receiver's end-point URI. With a similar attack mounted on the receiver's listen channel, the attacker can hijack connections. Finally, a denial-of-service (DoS) attack could be launched against the server by sending many PUT requests (an attack that can also occur today) or by registering many S-GETs, to exhaust its resources.

The impersonation and eavesdropping attacks can be prevented by the use of randomly selected URIs and encryption, *e.g.*, end-point channels contain random strings and listening channels use encryption. Our analysis suggests that HTTP datagrams do not fundamentally increase attackers' power to DoS HTTP servers (see our TR [35]).

Interaction with Transport: In our implementation, the entire S-GET communication is performed over a single TCP connection. However, note that since S-GET does not provide reliability or in-order delivery, in theory, it can be implemented over other protocols than TCP.

5. PRELIMINARY EVALUATION

The performance of an architecture based on HTTP depends on several factors: the performance of CDNs, the pervasiveness of caches, the hit rate of caches, the scalability of HTTP proxies and the efficiency of the proposed datagram service. In this section, we summarize the preliminary evaluation results restricted to the datagram service (for more evaluation results, see [35]).

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.