Cache Me If You Can

**BUILDING A
DECENTRALIZED
WEB-DELIVERY
MODEL**

JACOB LOVELESS

Years ago I squandered most of my summer break locked inside my apartment, tackling an obscure problem in network theory (the bidirectional channel capacity problem). I was convinced that I was close to a breakthrough. (I wasn't.) Papers were everywhere, intermingled with the remnants of far too many 10¢ Taco Tuesday wrappers.

A good friend stopped by to bring better food, lend a mathematical hand, and put an end to my solitary madness. She listened carefully while I jumped across the room grabbing papers and incoherently babbling about my “breakthrough.”

Then she somberly grabbed a pen and wrote out the

flaw that I had missed, obliterating the proof.

I was stunned and heartbroken. She looked up and said, “But this is great, because what you’ve done here is define the problem more concretely.” She continued with a simple truth that I’ve carried with me ever since:

“Most times, defining the problem is harder and more valuable than finding the answer. The world needs hard, well-defined problems. Generally, only one or two people work together on an answer—but hundreds work on a well-defined problem.”

And so, dear reader, this is where I would like to begin. Unlike most articles in *acmqueue*, this one isn’t about a new technique or an exposition on a practitioner’s solution. Instead, this article looks at the problems inherent in building a more decentralized Internet. Audacious? Yes, but this has become a renewed focus in recent years, even by the father of the Web himself (see Tim Berners-Lee’s Solid project⁴). Several companies and open-source projects are now focusing on different aspects of the “content-delivery” problem. Our company Edgemesh (<https://edgemesh.com>) is working on peer-enhanced client-side content acceleration, alongside other next-generation content-delivery networks such as Peer5 (<https://peer5.com>) and Streamroot (<https://streamroot.io>), both of which are focused on video delivery. Others, such as the open-source IPFS (InterPlanetary File System; <https://ipfs.io>) project are looking at completely new ways of defining and distributing and defining “the web.”

Indeed, the concept of a better Internet has crept into

popular media. In season 4 of HBO's sitcom *Silicon Valley*, the protagonist Richard Hendricks devises a new way to distribute content across the Internet in a completely distributed manner using a P2P (peer-to-peer) protocol. "If we could do it, we could build a completely decentralized version of our current Internet," Hendricks says, "with no firewalls, no tolls, no government regulation, no spying. Information would be totally free in every sense of the word." The story line revolves around the idea that thousands of users would allocate a small portion of their available storage on their mobile devices, and that the Piper software would assemble the storage across these devices in a distributed storage "cloud." Then, of course, phones explode and hilarity ensues.

The core idea of a distributed Internet does make sense, but how would it be built? As I learned in my self-imposed solitary confinement so long ago, before diving into possible solutions, you need to define the problems more clearly.

PROBLEMS OF A DISTRIBUTED INTERNET

In his 2008 *acmqueue* article "Improving Performance on the Internet," Tom Leighton, cofounder of Akamai, the largest content-distribution network in the world, outlined four major architectures for content distribution: centralized hosting, "big data center" CDNs (content-delivery networks), highly distributed CDNs, and P2P networks. Of these, Leighton noted that:

"P2P can be thought of as taking the distributed architecture to its logical extreme, theoretically

providing nearly infinite scalability. Moreover, P2P offers attractive economics under current network pricing structures.”¹¹

He then noted what others have found in the past, that although the P2P design is *theoretically* the most scalable, there are several practical issues, specifically *throughput*, *availability*, and *capacity*.

Throughput

The most commonly noted issue is the limited uplink capacity of edge devices, as noted by Leighton in his 2008 article:

P2P faces some serious limitations, most notably because the total download capacity of a P2P network is throttled by its total uplink capacity. Unfortunately, for consumer broadband connections, uplink speeds tend to be much lower than downlink speeds: Comcast’s standard high-speed Internet package, for example, offers 6 Mbps for download but only 384 Kbps for upload (one-sixteenth of download throughput).¹¹

This limitation is not as acute today as it was nearly a decade ago when upload speeds in the U.S. hovered around .5 Mbps. Figure 1 shows the current upload and download as taken from Speedtest.net (<http://www.speedtest.net/reports/>). These data points show that global “last-mile” throughput rates are nearly 30 times their 2008 counterparts. Is this enough? Would a peer with an upload rate at the lower quartile of these metrics (~4 Mbps)

1

FIGURE 1: **BANDWIDTH RATES (UPLOAD/DOWNLOAD, BROADBAND/MOBILE)**

REGION	TERRESTRIAL		YOY %		MOBILE		YOY %	
	DOWN-LOAD (MBPS)	UPLOAD (MBPS)	DOWN-LOAD	UPLOAD	DOWN-LOAD (MBPS)	UPLOAD (MBPS)	DOWN-LOAD	UPLOAD
United States	54.97	18.88	42%	51%	19.61	7.94	794%	28%
United Kingdom	40.12	40.12	18%	23%	24.19	10.03	38%	31%
Singapore	190.9	178.69	18%	20%	45.61	17.93	16%	13%
Hong Kong	153.53	146.69	13%	0%	18.9	8.78	5%	1%
Brazil	16.53	6.08	8%	11%	11.31	4.27	37%	35%

suffice? This question has been thoroughly explored in regard to actual web page load time.

When Mike Belshe (of Google SPDY fame) looked at the relationship between end-client bandwidth and page-load time, he discovered that “bandwidth doesn’t matter (much).”³ Once the client’s available bandwidth reached 5 Mbps, the impact on the end user’s page load is negligible. Figure 2 shows page-load time as a function of client bandwidth, assuming a fixed RTT (round-trip time) of 60 ms.

Availability

The next major hurdle for the distributed Internet is peer availability. Namely, are there enough peers, and are they online and available for enough time to provide value? In the past ten years the edge device count has certainly increased, but has it increased enough? Looking at “Internet Trends 2017,”¹⁴ [compiled by Mary Meeker of the venture capital firm KPCB], you can see how much the “available peer” count has increased over the past ten years from mobile alone [see figure 3].

2

FIGURE 2: PAGE LOAD TIME AS A FUNCTION OF CLIENT BANDWIDTH

Today roughly 49 percent of the world's population is connected¹⁰—around 3.7 billion people, many with multiple devices—so that's a *big* pool of edge devices. Peter Levine of the venture capital firm Andreessen Horowitz has taken us out a few years further and predicted that we will shortly be going beyond billions and heading toward *trillions* of devices.¹²

You can get a sense of scale by looking at an Edgemesher network for a single e-commerce customer's website with a global client base, shown in figures 4 and 5.

It's probably safe to say there are enough devices online, but does the average user stay online long enough to be

3

FIGURE 3: MARY MEEKER'S 2017 SMARTPHONE ANALYTICS

4

FIGURE 4: **EDGEMESH SAMPLE CUSTOMER MESH NETWORK (GLOBAL)**

FIGURE 5: **EXAMPLE P2P NETWORK**

5

available? What is “long enough” for a peer to be useful?

A sensible place to start might be to want peers to be online long enough for any peer to reach any other peer anywhere on the globe. Given that, we can set some bounds.

The circumference of the earth is approximately 40,000 km. The rule of thumb is that light takes 4.9 microseconds to move 1 km through fiber optics. That would mean data could circumnavigate the globe in about one-fifth of a second (196 milliseconds). Oh, if wishing only made it so, but as Stanford University’s Stuart Cheshire points out in “It’s the Latency, Stupid,”⁶ the Internet operates at least a factor of two slower than this. This 2x slowdown would mean it would take approximately 400 milliseconds to get around the globe. Unfortunately, I have spent some time in telecom—specifically in latency-optimized businesses¹³—and I think this time needs to be doubled again to account for global transit routing; thus, the data can go around the world in some 800 milliseconds. If users are online and available for sub-800 millisecond intervals, this may become problematic. Since most decentralized solutions would require the user to visit the content (e.g., be on the website), the real question is, what is the average page-view time for users across the globe?

Turns out it is 3 minutes 36 seconds,²⁴ or 216,000 milliseconds.

To double-check this, I took all peer-session times (amount of time Edgемesh peers were online and connected to the mesh) across the Edgемesh user base for the past six months (figure 6). The average was right in line at 3 minutes 47 seconds.

FIGURE 6: HISTOGRAM OF PEER SESSION DURATION

In either case, if the node stays online just long enough to download a single web page, that would be enough time for the data to circumnavigate the globe 270 times, certainly long enough to contact a peer anywhere on Earth.

Capacity

If enough users are online for a long enough duration, and they have an acceptable egress throughput (upload bandwidth), all that remains is the question of whether

7

there is enough spare capacity (disk space) available to provide a functional network.

If we assume a site has 20 percent of its users on mobile and 80 percent of its users on desktops—and further expand this to 500 MB of available capacity per desktop user and 50 MB per mobile user (the lower end of browser-available storage pools)—we can extract an estimated required mesh size to achieve a given cache hit rate if the requested content follows a Zipf distribution.¹ Figure 7 shows estimated mesh size required for variable cache hit rates for .5 TB, 1 TB, and 2 TB active caches. These numbers certainly seem reasonable for a baseline. Essentially, a website with 500 GB of static content (about 16 million

FIGURE 7: **ESTIMATED MESH SIZE REQUIRED VARIABLE CACHE HIT RATES**

average web images] would need an online capacity of 2 million distinct nodes to achieve a theoretical offload of 100 percent of its traffic to a P2P mesh (approximately an 8:1 ratio of images to users).

ENABLING A DISTRIBUTED INTERNET

Now that we've better defined the problems and established the *theoretical* feasibility of a new solution, it's time to look at the technology available to bring to bear on the problem. To start, we can constrain our focus a bit. Implementations such as IPFS focus on distributing the entire content base, allowing you to free yourself from the restrictions of web servers and DNS entirely. This is a fantastic wholesale change, but the tradeoff is that it will require users to dramatically modify how they access content.

Since a peer-to-peer design is dependent on the total network size, this model has difficulty growing until it reaches critical mass. At Edgemesh we wanted to focus on enhancing existing web-content delivery transparently (e.g., in the browser) without requiring any changes to the user experience. This means ensuring that the technology abides by the following three restrictions:

- For users, the solution should be transparent.
- For developers, the solution should require zero infrastructure changes.
- For operations, the solution should be self-managing.

The next question is where exactly to focus.

Fully enabling peer-enhanced delivery of *all* content is difficult and dangerous (especially allowing for peer-delivered JavaScript to be executed). Is there an 80

percent solution? Trends posted by HTTP Archive reveal that static components (images/video/fonts/CSS) make up roughly 81 percent of the total page weight,⁹ as shown in figure 8.

Given these details, let's narrow the focus to enabling/enhancing edge delivery of these more traditional CDN assets and the associated challenges of moving and storing data.

Moving data: building a new network (overlay)

To support peer-to-peer distribution, an overlay network needs to be developed to allow the peer-to-peer connections to operate within the larger existing Internet infrastructure. Luckily, such a stack is available: WebRTC

FIGURE 8: AVERAGE PAGE WEIGHT % BY ASSET TYPE

[Real-Time Communications¹⁹]. Started in earnest in 2011 by Google, WebRTC is an in-browser networking stack that enables peer-to-peer communication. WebRTC is primarily employed by voice and video applications (Google Hangouts/Dual/Allo, Slack, Snapchat, Amazon Chime, WhatsApp, Facebook Messenger) to facilitate peer-to-peer video- and audioconferencing.

WebRTC is a big deal; in June 2016 (only five years later) Google provided several key milestones⁷ from stats it collected (with some additional updates at the end of 2016²⁵):

- ➔ Two billion Chrome browsers with WebRTC.
- ➔ One billion WebRTC audio/video minutes per week on Chrome.
- ➔ One petabyte of DataChannel traffic per week on Chrome (0.1 percent of all web traffic).
- ➔ 1,200 WebRTC-based companies and projects (it was 950 in June).
- ➔ Five billion mobile app downloads that include WebRTC.

WebRTC support exists in the major browsers (Chrome, Firefox, Edge, and now Safari²). Comparing WebRTC's five-year adoption against other VoIP-style protocols shows the scale (see figure 9).⁸

WebRTC is a user-space networking stack. Unlike HTTP, which is dependent on TCP for transfer, WebRTC has its roots in a much older protocol—SCTP (Stream Control Transmission Protocol)—and encapsulates this in UDP (User Datagram Protocol). This allows for much lower latency transfer, removes head-of-line blocking, and, as a separate network stack, allows WebRTC to use significantly more bandwidth than HTTP alone.

SCTP is a little like the third wheel of the transport

FIGURE 9: VOIP USERS, BY TECHNOLOGY, AFTER 5 YEARS

layer of the OSI (Open Systems Interconnection) model—we often forget it’s there but it has some very powerful features. Originally introduced to support signaling in IP networks,²² SCTP quickly found adoption in next-generation networks (IMS and LTE).

WebRTC leverages SCTP to provide a reliable, message-oriented delivery transport (encapsulated in UDP or TCP, depending on the implementation⁵). Alongside SCTP, WebRTC leverages two additional major protocols: DTLS (Datagram Transport Layer Security) for security (a derivative of SSL) and ICE (Interactive Connectivity Establishment) to allow for support in NAT (network address translation) environments (e.g., firewall traversal).

The details of the ICE protocol and how it works with signaling servers (e.g., STUN and TURN) are beyond the

scope of this article, but suffice it to say that WebRTC has all the necessary plumbing to enable real peer-to-peer networking.

A simple example is a WebRTC Golang implementation by Serene Han.²¹ Han's chat demo allows you to pass the SDP (Session Description Protocol) details between two machines (copy-paste signaling) to enable peer-to-peer chat. To run this yourself (assuming you have a Docker instance locally), simply do the following:

```
docker run -it golang bash
```

Then in the Docker instance, this one-liner will get you set up:

```
apt-get update && apt-get install libx11-dev -y  
&& \  
go get github.com/keroserene/go-webrtc && \  
cd /go/src/github.com/keroserene/go-webrtc && \  
go run demo/chat/chat.go
```

If you prefer a browser-native starting point, look at simple-peer module,²⁰ originally from Feross Aboukhadijeh's work with WebTorrent (<https://webtorrent.io>).

Storing data: browser storage options and asset interception

The next step is finding a method both to intercept standard HTTP requests and to develop a system for storing peer-to-peer delivered assets. For the request-intercept problem, you have to look no further than the service worker.¹⁸ The service worker is a new feature

10

available in most browsers that allows for a background process to run in the browser. Like a web worker (which can be used as a proxy for threads), a service worker has restrictions on how it can interact and exchange data with the DOM [Document Object Model].

The service worker does, however, have a powerful feature that was originally developed to support offline page loads: the Fetch API¹⁶. The Fetch API allows a service worker to intercept request and response calls, similar to an HTTP proxy. This is illustrated in figure 10.

FIGURE 10: **SERVICE WORKER ENABLED CACHE VIA MOZILLA**

- 1 After registering a service worker to control a scope, each time a client in that scope requests a resource from **anyplace** in the network...

- 2 The service worker for that scope enters the game and **hijacks** the request.

- 3 Through **event.respondWith()** and passing a response object, the service worker can answer the client's request with a custom response made out of different sources: the network, storage or its own code.

With the service worker online, you can now intercept traditional HTTP requests and offload these requests to the P2P network. The last remaining component will be a browser local storage model where P2P-accelerated content can be stored and distributed. Although no fewer than five different storage options exist in the browser, the IndexedDB¹⁷ implementation is the only storage API available within a service-worker context and the DOM context (where the WebRTC code can execute, which is why Edgemesh chose it as the storage base). Alternatively, the CacheStorage API may also be used within the service-worker context.¹⁵

IMPLEMENTING A DISTRIBUTED INTERNET

We have a theoretically viable model to support peer-to-peer content delivery. We have a functional network stack to enable ad-hoc efficient peer-to-peer transfer and access to an in-browser storage medium. And so, the game is afoot!

Figure 11 is a flowchart of the Edgemesh P2P-accelerated content-delivery system. The figure shows where the service-worker framework will enable asset interception, and WebRTC (aided by a signal server) will facilitate browser-to-browser asset replication.

Returning, then, to Mike Belshe's research, we can start to dig into some of the key areas to be optimized. Unlike bandwidth, where adding incrementally more bandwidth above 5 Mbps has negligible impact on page-load time, latency (RTT) dramatically increases page-load time, as shown in figure 12.³

11

FIGURE 11: FLOW DIAGRAM FOR WEBRTC ENHANCED ASSET DELIVERY

12

FIGURE 12: **LINEAR DECREASE IN PAGE LOAD TIME AS FUNCTION OF LATENCY**

WebRTC is already an efficient protocol, but the peer-selection process presents opportunities for further latency reduction. For example, if you are located in New York, providing a peer in Tokyo is likely a nonoptimal choice. Figure 13 shows a sampling of WebRTC latency distributions for a collection of sessions across the Edgemesh networks. Can we do better?

A simple optimization might be to prefer peers that reside in the same network, perhaps identified by the AS

13

FIGURE 13: WEBRTC LATENCY HISTOGRAM IN PRODUCTION

[autonomous system]²³ number of each peer. Even this simple optimization can cut the average latency by a factor of two. Figure 14 shows performance increase by intra-AS routing preference.

Another optimization is choosing which assets to replicate into a peer. For example, if a user is currently browsing the landing page of a site, we can essentially precache all the images for the next pages, effectively eliminating the latency altogether. This is a current area of research for the team at Edgemesh, but early solutions have already shown significant promise. Figure

14

FIGURE 14: PERFORMANCE INCREASE BY ROUTING PREFERENCE

15 shows the effective render time for Edgemesh-enabled clients (accelerated) and non-Edgemesh enabled clients (standard) for a single customer domain. The average page-load time has been reduced by almost a factor of two.

This is most clearly seen when most of the page content can be effectively precached, as shown in the page-load time statistics of figure 16.

CONCLUSION

It had been a few days since I'd been outside, and a few more since I would make myself presentable. For the

15

FIGURE 15: EFFECT OF P2P ENHANCED BACKGROUND CACHING ON LOAD TIME

accelerated	standard
4249 page views	7674 page views
4248 visitors	4235 visitors
2.568 average load time (seconds)	4.838 average load time (seconds)
55.658 max load time (seconds)	56.775 max load time (seconds)
0.403 average net time (seconds)	0.852 average net time (seconds)
9.361 max net time (seconds)	36.465 max net time (seconds)
803 views over 4 seconds	3524 views over 4 seconds
1779 views over 2 seconds	5912 views over 2 seconds

previous few weeks the team and I had been locked inside the office around the clock, essentially rewriting the software from scratch. We thought it would take a week, but we were now three months into 2017. The growing pile of empty delivery bags resting atop the ad-hoc whiteboard tables we were using was making it difficult to make out exactly what the big change was. We were convinced this was the breakthrough we had been looking for (turns out it was), and that this version would be the one that cracked

16

FIGURE 16: **PAGE LOAD TIME COMPONENTS: ACCELERATED VS. STANDARD**

the problem wide open. I was head down trying to get my parts to work, and I was lost. Then I heard the knock at the door. She came in and sat down, patiently moving aside the empty pizza boxes while I babbled on about our big breakthrough and how I was stuck.

Then, just like she had nearly two decades earlier, she grabbed the marker and said:

“Honey, I think I see the issue. You haven’t properly defined the problem. Most times, defining the problem is harder and more valuable than finding the answer. So, what exactly are you trying to solve?”

The world is more connected than it ever has been before, and with our pocket supercomputers and IoT

(Internet of Things) future, the next generation of the web might just be delivered in a peer-to-peer model. It's a giant problem space, but the necessary tools and technology are here today. We just need to define the problem a little better.

References

1. Adamic, L. A., Huberman, B. A. 2002. Zipf's law and the Internet. *Glottometrics* 3: 143-150; <http://www.hpl.hp.com/research/idl/papers/ranking/adamicglottometrics.pdf>.
2. Apple Inc. Safari 11.0; https://developer.apple.com/library/content/releasenotes/General/WhatsNewInSafari/Safari_11_0/Safari_11_0.html.
3. Belshe, M. 2010. More bandwidth doesn't matter [much]; <https://docs.google.com/a/chromium.org/viewer?a=v&pid=sites&srcid=Y2hyb21pdW0ub3JnfGRldnxneDoxMzcyOWI1N2I4YzI3NzE2>.
4. Berners-Lee, T. Solid; <https://solid.mit.edu/>.
5. BlogGeek.Me. 2014. Why was SCTP selected for WebRTC's data channel; <https://bloggeek.me/sctp-data-channel/>.
6. Cheshire, S. 1996-2001. It's the latency, stupid; <http://www.stuartcheshire.org/irants/latency.html>.
7. Google Groups. 2016. WebRTC; <https://groups.google.com/forum/#!topic/discuss-webrtc/IOGqzwfKJfQ>.
8. Hart, C. 2017. WebRTC: one of 2016's biggest technologies no one has heard of. WebRTC World; <http://www.webrtcworld.com/topics/webrtc-world/articles/428444-webrtc-one-2016s-biggest-technologies-no-one-has.htm>.

9. HTTP Archive; <http://httparchive.org/trends.php>.
10. Internet World Stats. 2017. World Internet usage and population statistics—March 31, 2017; <http://www.internetworldstats.com/stats.htm>.
11. Leighton, T. 2008. Improving performance on the Internet. *acmqueue* 6(6); <http://queue.acm.org/detail.cfm?id=1466449>.
12. Levine, P. 2016. The end of cloud computing. Andreessen Horowitz; <http://a16z.com/2016/12/16/the-end-of-cloud-computing/>.
13. Loveless, J. 2013. Barbarians at the gateways. *acmqueue* 11(8); <http://queue.acm.org/detail.cfm?id=2536492>.
14. Meeker, M. 2017. Internet trends 2017—code conference. KPCB; <http://www.kpcb.com/internet-trends>.
15. Mozilla Developer Network. 2017. CacheStorage; <https://developer.mozilla.org/en-US/docs/Web/API/CacheStorage>.
16. Mozilla Developer Network. 2017. FetchAPI; https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API.
17. Mozilla Developer Network. 2017. IndexedDB API; https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API.
18. Mozilla Developer Network. 2017. Using service workers; https://developer.mozilla.org/en-US/docs/Web/API/Service_Worker_API/Using_Service_Workers.
19. Real-time communication in web browsers (rtcweb) Charter for Working Group; <http://datatracker.ietf.org/wg/rtcweb/charter/>.
20. Simple WebRTC video/voice and data channels. Github; <https://github.com/edgimesh/simple-peer>.

21. WebRTC for Go. Github; <https://github.com/keroserene/go-webrtc>.
22. Wikipedia. Signalling System No. 7; https://en.wikipedia.org/wiki/Signalling_System_No._7.
23. Wikipedia. Autonomous system; [https://en.wikipedia.org/wiki/Autonomous_system_\(Internet\)](https://en.wikipedia.org/wiki/Autonomous_system_(Internet)).
24. Wolfgang Digital. 2016. E-commerce KPI benchmarks; https://www.wolfgangdigital.com/uploads/general/KPI_Infographic_2016.jpg.
25. YouTube. 2016. WebRTC; <https://youtu.be/OUfYFMGtPQO?t=16504>.

Jacob Loveless is chief executive officer at Edgemesh Corporation, the premier edge-network acceleration platform. Prior to Edgemesh, Loveless served as CEO of Lucera Financial Infrastructures, a global network service provider for financial institutions. He was a partner at Cantor Fitzgerald and Company, responsible for running the firm's global low-latency trading operations for nearly 10 years. Prior to Wall Street, he was a senior engineer and consultant for the Department of Defense, focused on large-scale data-analysis programs and distributed networks. Loveless focuses primarily on low-latency networks and distributed computing. His prior ACM articles are available at <http://queue.acm.org/detail.cfm?id=2536492> and <http://queue.acm.org/detail.cfm?id=2534976>.

Copyright © 2017 held by owner/author. Publication rights licensed to ACM.