Fork Redirects edited

How to synchronize web data with multiple partners in a single call.

An important part of web technologies, particularly AdTech, is exchanging real-time browser data. Often this is in the form of a web **cookie** or other user/session identifier. Because many of the tech enabled solutions depend on multiple partners and vendors working together, it is critical to **synchronize** this data across partners.

One common example is that websites, advertisers, plugins, etc often want to have a shared understanding of a user. When a website sees someone, and an advertiser sees someone, how do they know whether they are the same person? This type of user identification on the internet is based on cookies. Due to web browser protocols, specific user cookies are only shared with website/domains that a cookie originates from. In order to get a shared understanding of a user, this cookie data must be synchronized across partners. One simplistic way of achieving this is by building a redirect chain.

A redirect chain is when one HTTP call (to a server) is then redirected to another, executed by the user's browser. A redirect chain can be fired by placing a pixel on a page, such as a simple IMG tag. For example:

```
<img url="https://lasso-os.com/identify" />
```

A user's browser will perform an HTTP GET request when the page is loaded and as a response to this request, a backend can return a response with 3XX code and `Location` header. This will contain a new URL with some piggyback identifier for the next sequential GET call that browser will perform automatically. For example:

`https://partner-url.com/sync?lasso-os.com/identify/store?lasso_id=some_id&partner_id={PARTNER_ID}`.

In the example above, a partner will fulfill their identifier placeholder and redirect the user back to some endpoint with both identifiers. This is what allows the two partners to synchronize their users.

So if we want to synchronize a user's id with multiple partners, we can build a following chain of redirects.

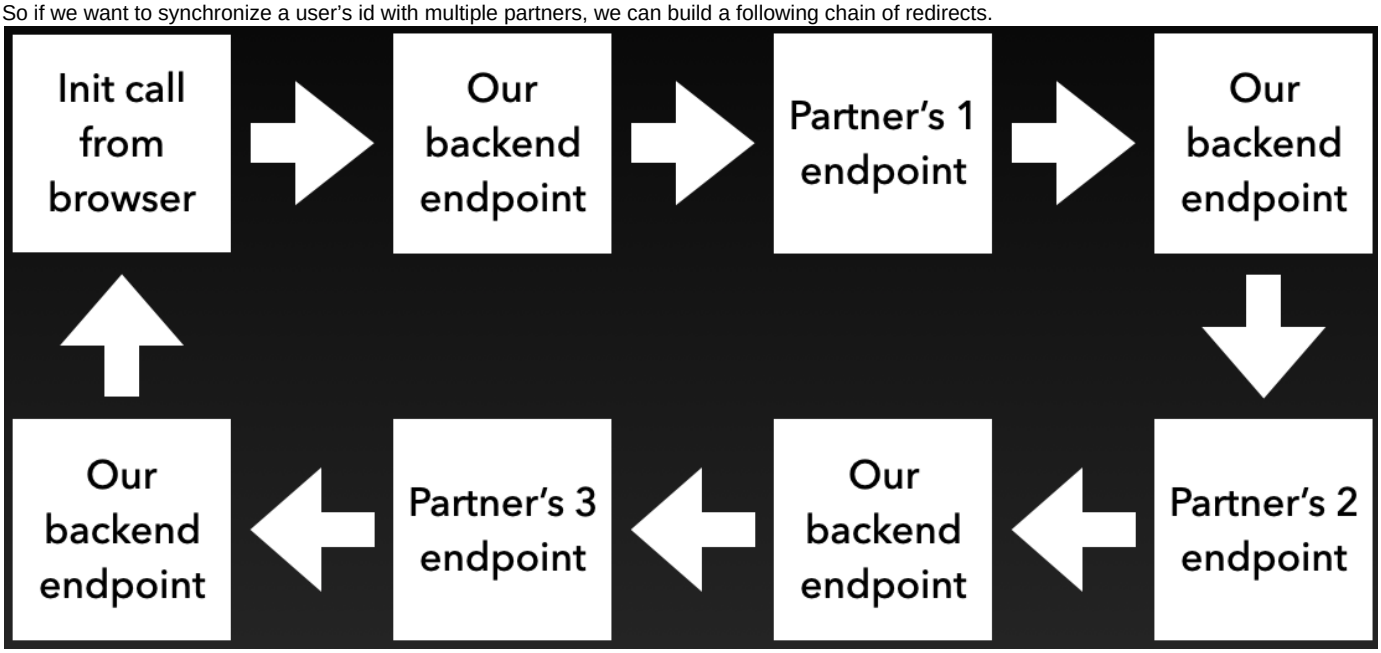


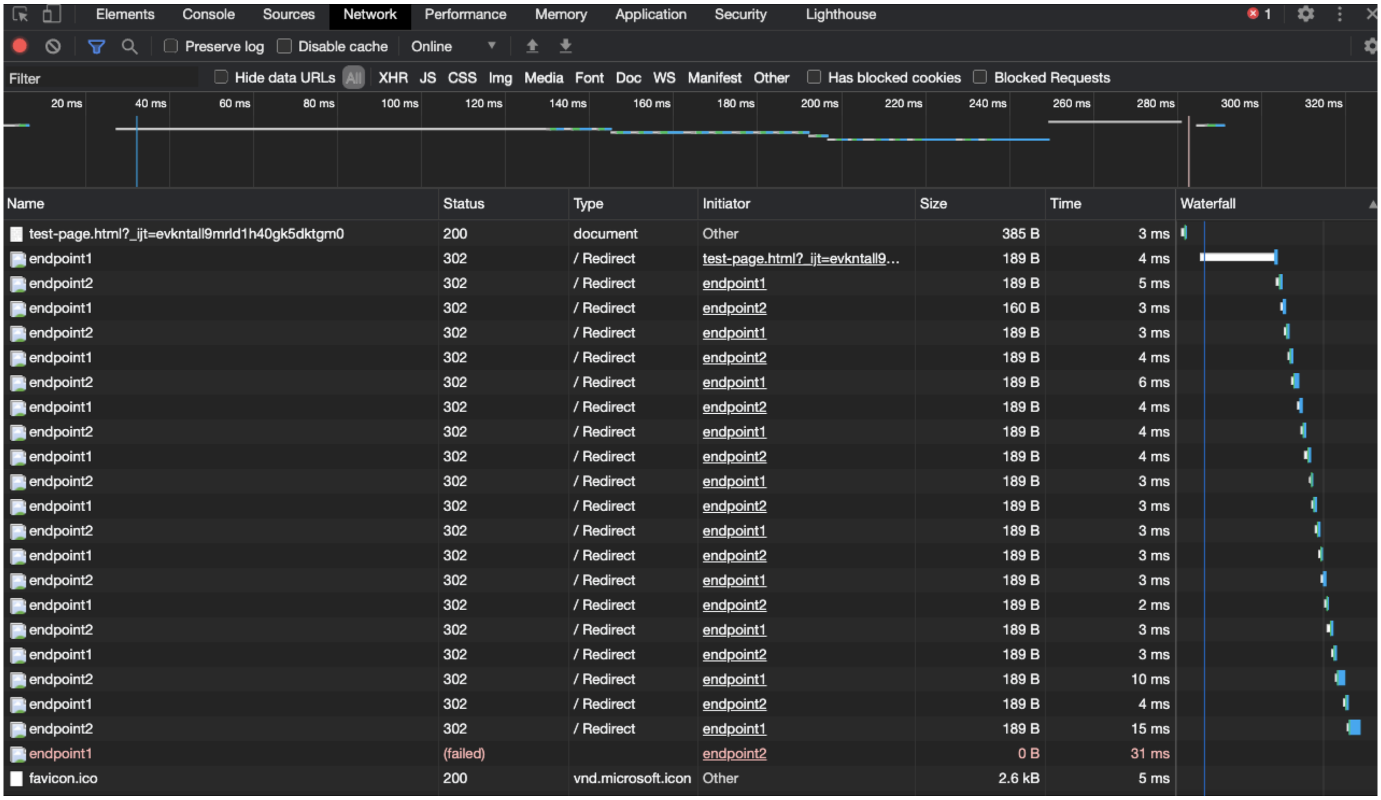
Fig. 1 - A sample of a redirect chain

But what issues we're facing with this approach?

That's a lot of redirects.

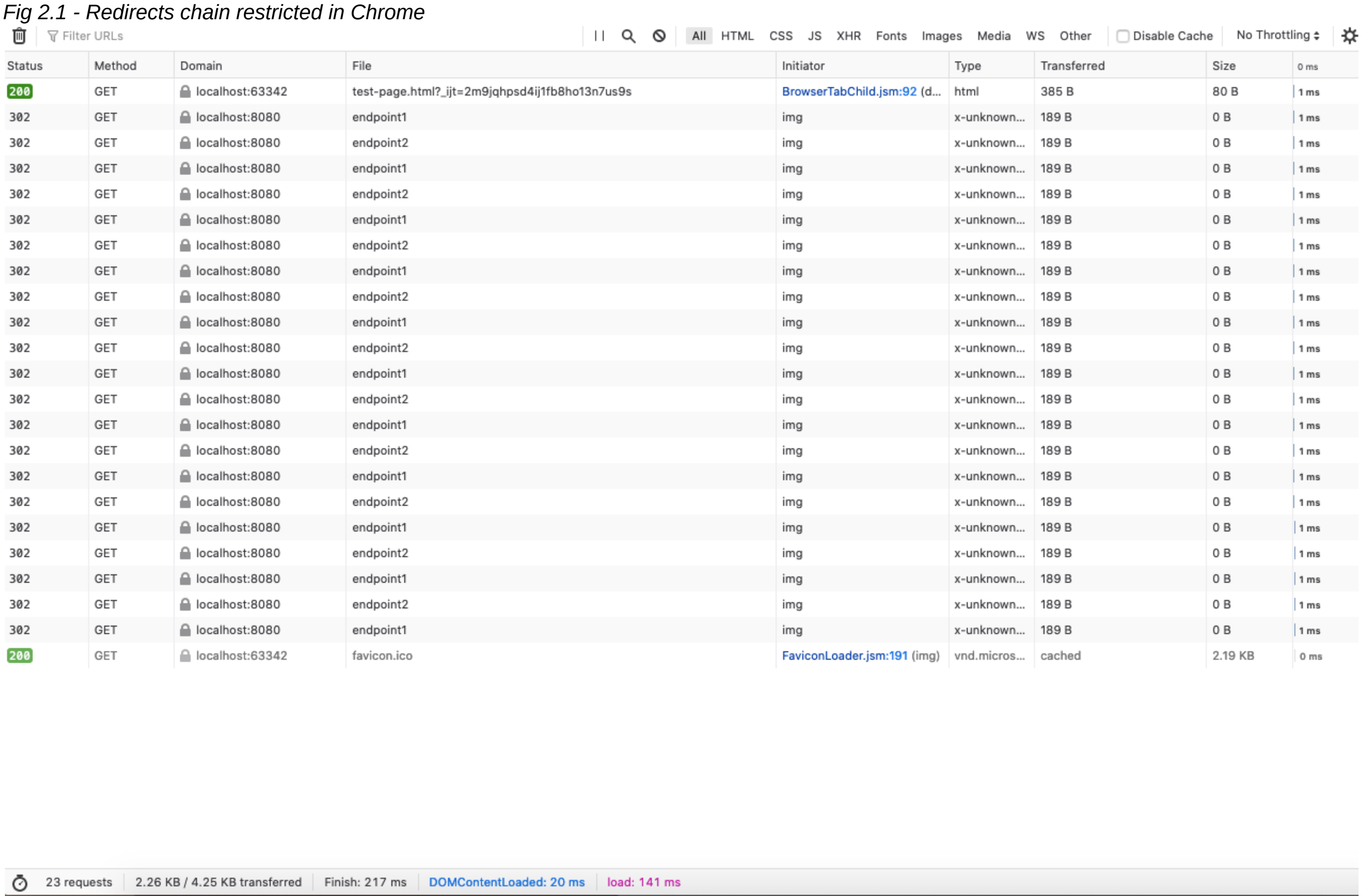
With only 3 partners in the chain we had on Fig. 1, we already have 7 redirects. And the bad news - browsers have restrictions on the number of redirects and will finally stop them with the following error:

ERR_TOO_MANY_REDIRECTS



Name	Status	Type	Initiator	Size	Time
test-page.html?_ijt=evkntall9mrld1h40gk5dktgm0	200	document	Other	385 B	3 ms
endpoint1	302	/ Redirect	test-page.html?_ijt=evkntall9…	189 B	4 ms
endpoint2	302	/ Redirect	endpoint1	189 B	5 ms
endpoint1	302	/ Redirect	endpoint2	160 B	3 ms
endpoint2	302	/ Redirect	endpoint1	189 B	3 ms
endpoint1	302	/ Redirect	endpoint2	189 B	4 ms
endpoint2	302	/ Redirect	endpoint1	189 B	6 ms
endpoint1	302	/ Redirect	endpoint2	189 B	4 ms
endpoint2	302	/ Redirect	endpoint1	189 B	4 ms
endpoint1	302	/ Redirect	endpoint2	189 B	4 ms
endpoint2	302	/ Redirect	endpoint1	189 B	3 ms
endpoint1	302	/ Redirect	endpoint2	189 B	3 ms
endpoint2	302	/ Redirect	endpoint1	189 B	3 ms
endpoint1	302	/ Redirect	endpoint2	189 B	3 ms
endpoint2	302	/ Redirect	endpoint1	189 B	3 ms
endpoint1	302	/ Redirect	endpoint2	189 B	2 ms
endpoint2	302	/ Redirect	endpoint1	189 B	3 ms
endpoint1	302	/ Redirect	endpoint2	189 B	3 ms
endpoint2	302	/ Redirect	endpoint1	189 B	10 ms
endpoint1	302	/ Redirect	endpoint2	189 B	4 ms
endpoint2	302	/ Redirect	endpoint1	189 B	15 ms
endpoint1	(failed)		endpoint2	0 B	31 ms
favicon.ico	200	vnd.microsoft.icon	Other	2.6 kB	5 ms

Fig 2.1 - Redirects chain restricted in Chrome



Status	Method	Domain	File	Initiator	Type	Transferred	Size	0 ms
200	GET	localhost:63342	test-page.html?_ijt=2m9jqhpsd4ij1fb8ho13n7us9s	BrowserTabChild.jsm:92 (d...	html	385 B	80 B	1 ms
302	GET	localhost:8080	endpoint1	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint2	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint1	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint2	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint1	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint2	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint1	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint2	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint1	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint2	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint1	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint2	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint1	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint2	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint1	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint2	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint1	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint2	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint1	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint2	img	x-unknown...	189 B	0 B	1 ms
302	GET	localhost:8080	endpoint1	img	x-unknown...	189 B	0 B	1 ms
200	GET	localhost:63342	favicon.ico	FaviconLoader.jsm:191 (img)	vnd.micros...	cached	2.19 KB	0 ms

Fig 2.2 - Redirects chain restricted in Firefox

To demonstrate this issue, I created two backend endpoints and pointed (haha 🙂) them one on another so they create an infinite loop. As you can see, browsers allow only 20 redirects and then cuts the loop.

It’s hard to maintain a chain of redirects

Every time our partner redirects a user back to our backend, we have to redirect it further to the next partner, or finally return 2XX and stop the chain. But how do we know which redirect should be the last in the chain? And what should be done if one of our partner’s services is down and the chain breaks?

That requires a lot of supportive logic on our end, and of course, we would like to avoid it!

Why not add more tags on our client’s page explicitly?

It will reduce the redirects chain length down to 2 calls (for every partner), but we want to have as few explicit details of the process, exposed to our clients, as possible. Not to mention, the more logic we have explicitly embedded in our client’s pages - the harder it is to change. Every new partner we want to exchange our user's data with will require a new tag on the page. And this:

```
<img url=”https://lasso-os.com/identify?partner=partner1” />

<img url=”https://lasso-os.com/identify?partner=partner2” />
```

looks terrible, because at some point we might want to modify our API, and we definitely don’t want to share any sort of our partner’s identifiers with our clients (even if it would be hashed).

And if we have hundreds of client pages, thousands of users, interacting with each page daily, and multiple partners we synchronize user’s data with, we would like to reduce the number of direct calls straight to our API for every redirect chain, so the first call in each chain would be direct to our partner and then redirected back to our service. The question is - how to manage all the calls to our partners' APIs from the client’s page if we want to form those calls right at the moment when a user hits the page?

So before answering the question let’s name our requirements:

1. We don’t actually want to build and maintain a long redirects chain
2. We want to have as much control as possible over the user's data exchange process
3. We want to expose as fewer details to our customers as it is possible

And ask ourselves again: can we synchronize our user’s data with all of our partners with just a single API call?

Fig 3. - Yes We Can

Solution

The trick here is to add some javascript. Fortunately enough, browsers will execute js code with almost any content by firing it with a simple `<script type="text/javascript" src="https://lasso-os.com/redirects-tag"></script>` tag.

And we can build it dynamically.

```
window.redirectsTag = {};
redirectsTag.__renderTag = function (response) {
    if (response && response.data) {
        var el = this.getElement();
        el.innerHTML = response.data;
        var scripts = this.getElement().getElementsByTagName("script");
        for (var i = 0; i < scripts.length; i++) {
            if (scripts[i].src != "") {
                var tag = document.createElement("script");
                tag.src = scripts[i].src;
                document.getElementsByTagName("head")[0].appendChild
(tag)
            } else {
                eval(scripts[i].innerHTML)
            }
        }
    }
};
var divName = "redirectsTag";
redirectsTag.getElement = function () {
    var el = document.getElementById(divName);
    if (el) {
        return el
    } else {
        el = document.createElement("div");
        el.id = divName;
        var body = document.getElementsByTagName("body")[0];
        body.appendChild(el);
        return el
    }
};
redirectsTag.__renderTag({data: '%1s'});
```

That script will do exactly what we tried to with placing multiple <img ..> tags on the client's page. But in this case, we have full control over the exact calls on our backend right at the moment, so they would be synchronized.

```
redirectsTag.__renderTag( response: {data:
    '<img width="1" height="1" src="http://localhost:8080/partner1?http://localhost:8080/identity/store?partnerName=partner1&ourId=5fa9f79f-dfa2-4af3-8bec-7a0f94f255d1&partnerUserId=%7BUSER_ID%7D" />' +
    '<img width="1" height="1" src="http://localhost:8080/partner2?http://localhost:8080/identity/store?partnerName=partner2&ourId=5fa9f79f-dfa2-4af3-8bec-7a0f94f255d1&partnerUserId=%7BUSR%7D" />' +
    '<img width="1" height="1" src="http://localhost:8080/partner3?http://localhost:8080/identity/store?partnerName=partner3&ourId=5fa9f79f-dfa2-4af3-8bec-7a0f94f255d1&partnerUserId=%7BID%7D" />'});
```

Fig. 4 - Actual JS code returned by the server

JS code will place as many tags as we want, and the browser will perform a call (with cookie headers) for every tag with all subsequent redirects.

Just perfect!

```
<html>
  <head></head>
  <body>
    <script async="" type="text/javascript" src="http://localhost:8080/identity"></script>
    <div id="redirectsTag">
      <img src="http://localhost:8080/partner1?http://localhost:8080/identit…f79f-dfa2-4af3-8bec-7a0f94f255d1&partnerUserId=%7BUSER_ID%7D" width="1" height="1">
      <img src="http://localhost:8080/partner2?http://localhost:8080/identit…5fa9f79f-dfa2-4af3-8bec-7a0f94f255d1&partnerUserId=%7BUSR%7D" width="1" height="1">
      <img src="http://localhost:8080/partner3?http://localhost:8080/identit…=5fa9f79f-dfa2-4af3-8bec-7a0f94f255d1&partnerUserId=%7BID%7D" width="1" height="1">
    </div>
  </body>
</html>
```

Fig. 5 - IMG tags, placed dynamically on the page by the js code, generated on our backend

Status	Method	Domain	File
200	GET	localhost:63342	test.html
200	GET	localhost:8080	identity
302	GET	localhost:8080	partner1?http://localhost:8080/identity/store?partnerName=partner1&ourId=768eda60-6b53-4a9f-b908-def0499d8e7f&partnerUserId={USER_ID}
302	GET	localhost:8080	partner2?http://localhost:8080/identity/store?partnerName=partner2&ourId=768eda60-6b53-4a9f-b908-def0499d8e7f&partnerUserId={USR}
302	GET	localhost:8080	partner3?http://localhost:8080/identity/store?partnerName=partner3&ourId=768eda60-6b53-4a9f-b908-def0499d8e7f&partnerUserId={ID}
200	GET	localhost:8080	store?partnerName=partner2&ourId=768eda60-6b53-4a9f-b908-def0499d8e7f&partnerUserId=2c5ea7c1-f461-47d5-84fc-b0b4d3684b25
200	GET	localhost:8080	store?partnerName=partner1&ourId=768eda60-6b53-4a9f-b908-def0499d8e7f&partnerUserId=4af5ab1d-11c0-41d3-85f8-3ee09414fd59
200	GET	localhost:8080	store?partnerName=partner3&ourId=768eda60-6b53-4a9f-b908-def0499d8e7f&partnerUserId=eb68e6f1-d776-4a9b-ac15-27a078434ce2
200	GET	localhost:63342	favicon.ico

Fig. 6 - HTTP calls, initiated by the placed tags

```
INFO 52063 --- [nio-8080-exec-6] i.l.r.c.IdentityRedirectsController      : Our user id 768eda60-6b53-4a9f-b908-def0499d8e7f is now mapped with 2c5ea7c1-f461-47d5-84fc-b0b4d3684b25 id for our partner partner2
INFO 52063 --- [nio-8080-exec-7] i.l.r.c.IdentityRedirectsController      : Our user id 768eda60-6b53-4a9f-b908-def0499d8e7f is now mapped with 4af5ab1d-11c0-41d3-85f8-3ee09414fd59 id for our partner partner1
INFO 52063 --- [nio-8080-exec-8] i.l.r.c.IdentityRedirectsController      : Our user id 768eda60-6b53-4a9f-b908-def0499d8e7f is now mapped with eb68e6f1-d776-4a9b-ac15-27a078434ce2 id for our partner partner3
```

Fig. 7 - Log of user-id mapping finally stored on our backend

You can find a sample Spring Boot project, that will help you to understand this technique in-depth here.

https://github.com/Lasso-Marketing/lasso-redirects-demo

Conclusion

In this article, we covered the approach we use extensively here in Lasso Marketing to support the user-specific data exchange process with our partners. If you would like us to cover more stuff from the AdTech world, contact us, and we would be happy to share our experience with you in the upcoming posts.

For example, in Part 2 we can take a look at how we can stream and store millions of records like these mappings, using Google Cloud Dataflow and BigQuery.

David Garibov, Lasso team