UNDERSTANDING THE LIGHTNING NETWORK, PART 2: CREATING THE NETWORK

AARON VAN WIRDUM • JUN 7, 2016

The Lightning Network is probably the most highly anticipated technological innovation to be deployed on top of Bitcoin. The payment layer, first proposed by Joseph Poon and Tadge Dryja about a year ago, promises to support a virtually unlimited number of off-chain transactions among users, at nearly no cost – while leveraging the security offered by Bitcoin.

At least three companies – Poon and Dryja's Lightning, Blockstream and Blockchain – are currently working on implementations of the technology. But few outside this small technological frontline fully grasp how the "future of micropayments" is set to boost Bitcoin's capabilities.

In this three-part series, *Bitcoin Magazine* lays out the basic building blocks of the Lightning Network, and shows how they fit together to realize this upcoming protocol layer.

The first part of this series covered basic building blocks, and explained how these are used to establish bidirectional payment channels. This second part explains how bidirectional payment channels are turned into a network.

The Network

In the previous article, Alice and Bob established a bidirectional payment channel. Now, Alice wants to pay one bitcoin to a third person, Carol.

To do so, Alice and Carol could open up a payment channel between them. But they don't actually need to. As it turns out, Bob and Carol already have a mutual channel, so Alice can simply pay Carol through Bob.

Specifically, Alice can pay Bob one bitcoin, and Bob can pay Carol one bitcoin.

However, Alice doesn't really trust Bob – or Carol for that matter. She's afraid that if she pays Bob, Bob will never actually pay Carol. Or perhaps Bob *will* pay Carol, but Carol will claim she never received the money, and Alice wouldn't know whom to blame.

Alice, therefore, wants to ensure that she only pays Bob one bitcoin, *if* he also pays Carol one bitcoin. This is accomplished (in part) with a simple cryptographic trick.

When Alice wants to send Carol a bitcoin, she tells Carol to create a value (a random string of numbers) and send her the hash. Alice also tells Carol to exchange the original value with Bob for a bitcoin.

Alice, meanwhile, takes the hash from Carol, turns to Bob, and tells Bob she will give him a bitcoin if he provides her the corresponding value (which only Carol has).

So, Bob turns to Carol, and gives Carol one bitcoin in return for the value.

Then, Bob turns back to Alice with the value. Alice knows Bob must have gotten the value from Carol in exchange for a bitcoin, and therefore concludes Carol got her bitcoin. So Alice can confidently give Bob a bitcoin.

Everybody is happy.

Well. .. *almost* everybody is happy.

In this “naive” scenario, middleman Bob still has to trust Alice and Carol. Bob has to trust Carol to really give him the value after he sent her a bitcoin, and Bob has to trust Alice to really give him a bitcoin once he presents her the value.

The bitcoin-for-value trades must therefore be absolutely guaranteed along the network. More specifically: *if* Bob gives a bitcoin to Carol, he *must* be guaranteed to get a bitcoin back from Alice.

That's where Hash Time-Locked Contracts (HTLCs) come in.

Hash Time-Locked Contracts

So Alice and Bob want to exchange a bitcoin for the value through an HTLC. (And Bob and Carol also want to a bitcoin exchange for that same value - but never mind that for now.)

To do so, rather than sending Bob a bitcoin straight up, Alice sends a bitcoin to a new (and, again: funky) multisig address. The bitcoins locked up on this address can be unlocked in two different ways.

The first option is for Bob to include his signature *and* the value.

The second option is for Alice to include her own signature. However, this option has a CLTV-timelock on it: Alice can sign and broadcast the transaction only after – say – two weeks have gone by.

This means that Bob has two weeks to create a subsequent transaction in which he includes his signature and the value, and broadcast it to send the bitcoin from the funky multisig address to himself. As such, this trade is guaranteed. Bob can *only* claim Alice's bitcoin if he provides the value: broadcasting it over the Bitcoin network makes it publicly visible for Alice to see.

And if Bob doesn't provide the value in time, there is a "time-out alternative" for Alice to get her bitcoin back. Simple.

Back to the network, as that's really why this HTLC setup is needed.

As mentioned, not only Alice and Bob, but also Bob and Carol established an HTLC. So, *if* Carol claims her bitcoin from Bob, Bob *will* get the value in return; it will be visible on the blockchain.

Therefore, *if* that happens, Bob is guaranteed to get a bitcoin from Alice as well. Bob can take the value that Carol made publicly visible on the blockchain, include it in his HTLC with Alice, and claim a bitcoin for himself, too. The two channels are effectively linked.

As a final detail, it *is* important that Bob gets the value from Carol *before* Alice can reclaim her bitcoin from Bob. If Bob gets the value from Carol only *after* Alice already reclaimed hers back, Bob is stuck in the middle after all. The time-out in Bob and Carol's HTLC must therefore expire *before* the time-out in Alice and Bob's HTLC expires. (For example after exactly ten days, instead of two weeks. This is also why HTLCs need CheckLockTimeVerify (CLTV)--and not CheckSequenceVerify (CSV).)

Lastly, there's one more problem to solve: for the Lightning Network to be useful, all this must be accomplished off-chain. How this is done, is covered in the third and final article of this series.