Universal Payment Channels

Jehan Tremback, Zack Hess
jehan.tremback@gmail.com,
zack.bitcoin@gmail.com

November 2015
v0.5

Abstract

This paper concerns a payment network called Universal Payment Channels, or UPC. UPC can handle transfers of any type of conventional or crypto currency, as well as physical or virtual goods, as long as these goods can be considered owned without physical possession and kept in escrow.

UPC consists of a series of "channels" between network participants. A channel is a private ledger arrangement between any two parties and a blockchain, a bank, or some other kind of ledger. It allows the parties to exchange payment by sending updated ledger balances to one another (not to the bank or blockchain). At any time, each of the participants can be confident that they will be able to retrieve all the money that they are owed.

Individual UPC payments are made without disturbing the third party bank or blockchain backing the channel. This means that there can be an unlimited amount of payments that do not put any strain on bank servers, or add anything to the blockchain. The only time that the bank or the blockchain is involved is when a network participant wants to take money out of the channel, or put money into it.

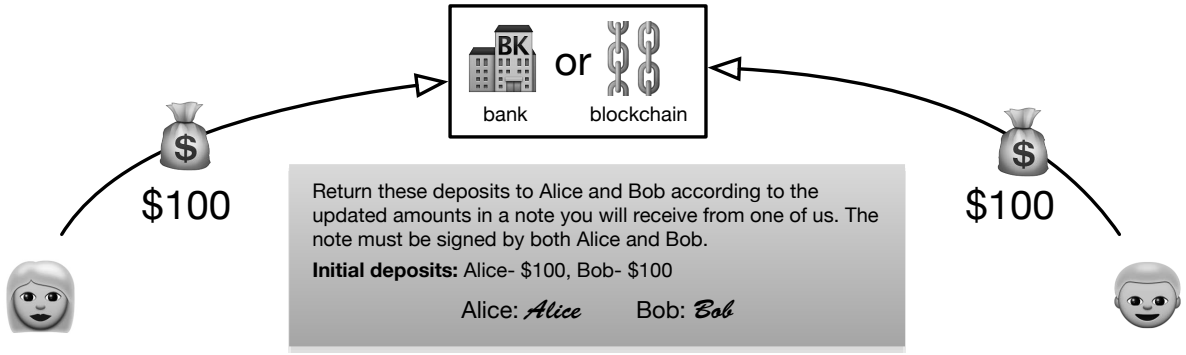
UPC channels can process "smart conditions", which are Turing-complete pieces of code evaluated by the bank or blockchain before some amount of money is released. One such smart condition is the "hashlock", which allows payments to be passed across several channels without any need to trust the intermediary nodes. This type of trustless transfer enables a global network of channels allowing for instant, anonymous payments in any currency, as well as automatic exchange across currencies.

1 Simplified Explanation

This is a simplified explanation of the protocol in this paper, using diagrams to get the idea across. Some details of the protocol's mechanics are glossed over and simplified to make it easier to follow. If you want the full specification, skip to the next section.

In a payment channel, two parties deposit money with a third entity that both trust. If the channel is to transfer conventional currency, a bank or payment processor plays the role of trusted third party and holds onto the money. Both

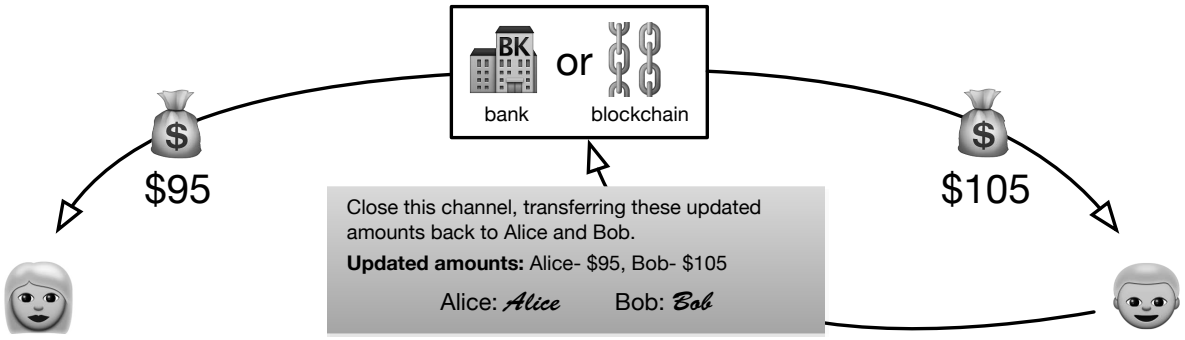
parties must trust the integrity of that bank or payment processor. If the channel holds cryptocurrency, a contract on a blockchain locks the funds from both parties. Both parties must then trust the integrity of that blockchain.



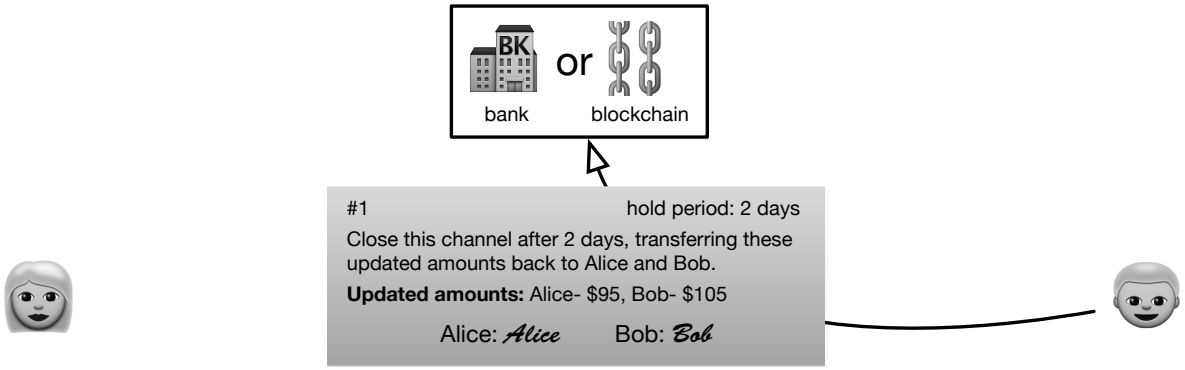
The bank or the blockchain will transfer the locked funds back to the channel participants upon receiving a message signed by both.

Upon receiving this message, the bank or blockchain also updates the amounts to be transfered back. If Alice and Bob both deposited $100 to open the channel, and close it with balances of $95 and $105, Alice has effectively given Bob $5. So, to pay Bob, Alice signs a message updating her balance to $95 and Bob's balance to $105.

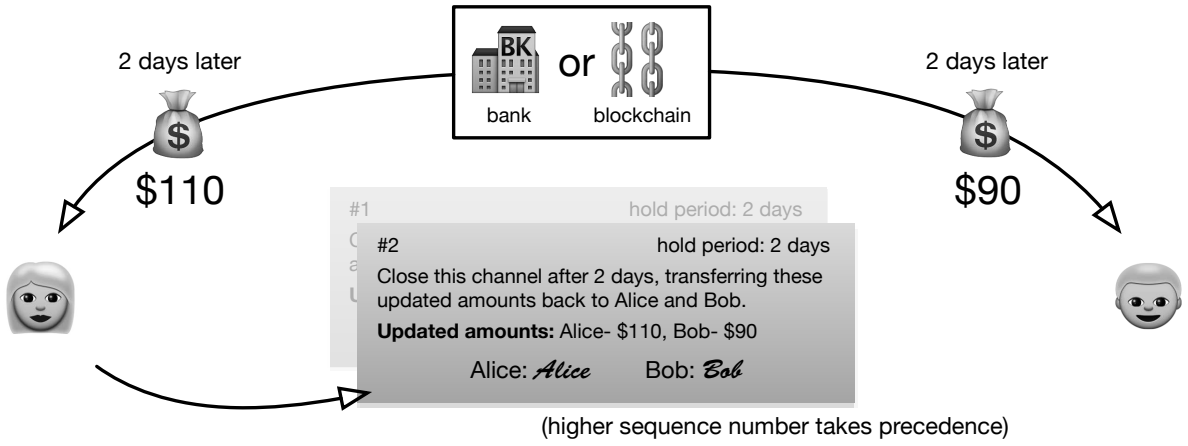
Alice sends this message only to Bob, without contacting the bank or the blockchain that the channel is open with. If Bob wants to get his money out, he simply posts the last signed message to the bank or the blockchain.



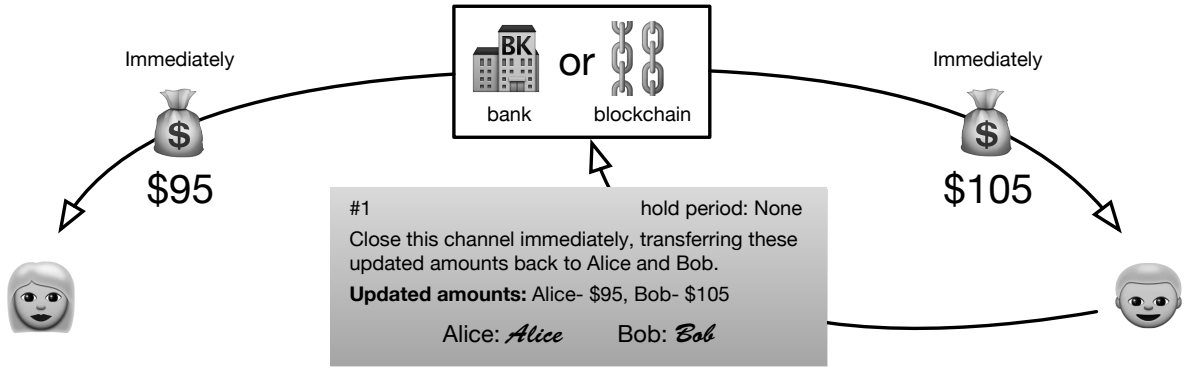
There's one issue though- someone could cheat. Let's say that Bob makes a payment to Alice and the balances are updated to Alice- \$50 and Bob- \$150. Then Alice makes a payment to Bob, reversing the balances to Alice- \$150, Bob- \$50. Bob could take the old message where he has \$150 and post it, cheating Alice out of \$100.



How to prevent this? We need some way for the bank or the blockchain to find out whether a message represents the account balances that Alice and Bob most recently agreed on. If Alice and Bob put a sequence number on each message and increment it every message, either of them can prove if one message is more recent than another. If the bank or blockchain then waits a certain length of time (or "hold period") before transferring the money back, it gives either party a chance to prove that the other is cheating.



What if Alice and Bob don't want to wait to get their money out? They can simply sign a message with a hold time of 0. This means that the bank or blockchain will immediately transfer the money to their accounts. The only situation in which the hold time will actually be a factor is a situation where one of the parties wants to close the channel and the other is unresponsive or uncommunicative.



1.1 Multihop payments

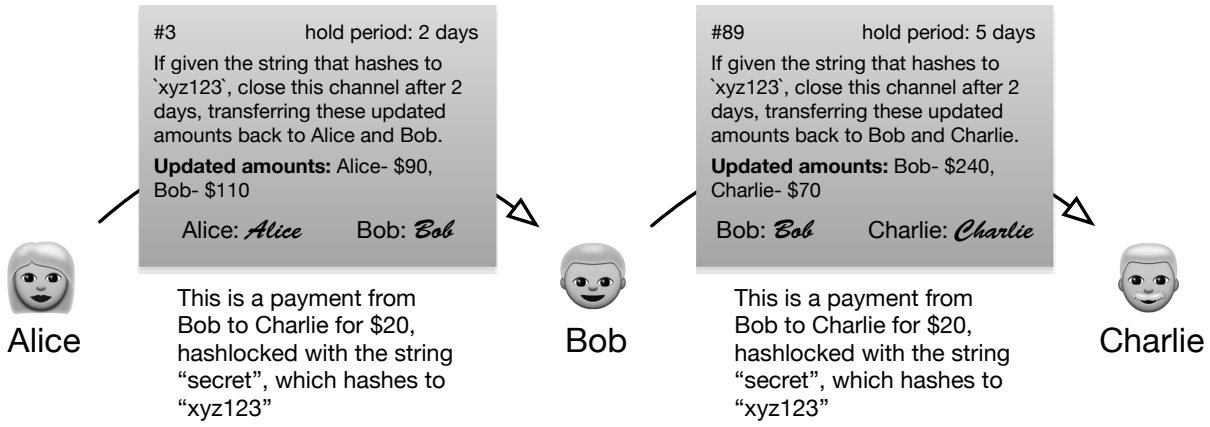
Let's say that Alice wants to send Charlie a payment, but she does not have a channel open with him. Opening a channel usually involves some cost and delay. What if both Alice and Charlie have channels open with Bob? Alice could send Bob a payment, who would then send Charlie a payment. But now Alice needs to trust Bob.

1.1.1 Smart conditions and hashlocks

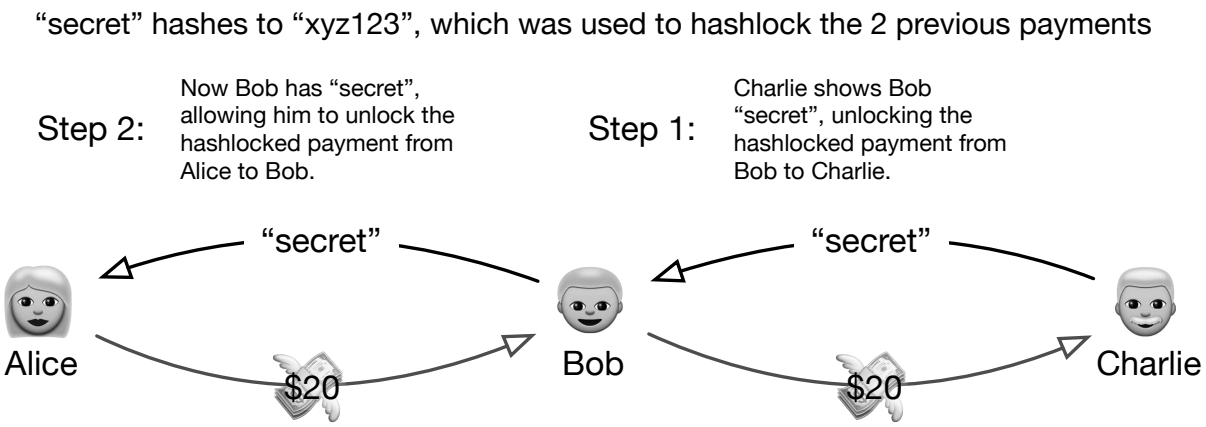
We have to make sure that Bob can't steal the money. UPC allows us to make payments with pieces of code called "smart conditions". The bank or blockchain evaluates the smart condition, to find out whether it should transfer some money. We can make a type of smart condition called a hashlock which allows us to trustlessly route payments through one or more intermediary nodes. A hashlock basically says: "transfer this amount of money if you are given the string that hashes to this hash".

1.1.2 Trustless multihop payments with hashlocks

To route a payment through Bob, Alice sends Charlie a secret, and the amount of the payment. Alice then sends a payment to Bob, hashlocked with the secret she sent Charlie. Bob sends his own payment message to Charlie, hashlocked with the same secret, and containing a payment of the same amount.



To unlock the payment from Bob, Charlie reveals the secret to him, which allows Bob to unlock the payment from Alice.



This all happens as fast as packets can be forwarded, and doesn't store anything on the blockchain or bank servers.

2 Full Specification

Two parties create and sign an **Opening Transaction** to put money from both parties in escrow with an institution such as a bank, or lock up coins on a blockchain. At some point in the future the money will be transferred back to the parties. This transfer will occur when the bank or the blockchain receives an **Update Transaction** signed by both parties. This **Update Transaction** has a **Sequence Number**, and specifies two additional actions to take before transferring the funds back to the parties:

1. Adjust the amounts that both parties have in escrow, lowering one amount and raising the other by the same amount. This effectively transfers funds from one party to the other.
2. Wait for a certain **Hold Period** before releasing the funds to back to the parties. If someone gives you another **Update Transaction** signed by both parties, and this transaction has a higher **Sequence Number**, throw the current **Update Transaction** out and use the new one, restarting the **Hold Period**.

Alice and Bob exchange **Update Transactions** back and forth, changing the amount of funds to be transferred to make payments to one another. Each time they make a new **Update Transaction**, they increment the **Sequence Number**. To accept a payment, both of them sign it.

- If Bob disappears, Alice can post the last signed **Update Transaction** and collect the money owed her without Bob's involvement, after waiting for the hold period to end.
- If Bob tries to cheat by posting an old **Update Transaction** where he has more money in his side of the channel, Alice can post the latest **Update**

Transaction, which will override the old one. As long as Alice checks whether Bob has posted an old **Update Transaction** at least once every **Hold Period**, she can post the latest **Update Transaction** and stop him from cheating.

2.1 Opening Transaction

A channel is opened with an **Opening Transaction**. The **Opening Transaction** serves to identify the channel and the parties, and places the money in escrow. This could be sent to a bank, or supplied to a smart contract on a blockchain.

Opening Transaction:

- **Party 1:** Public key or other signature verification information for one of the participants.
- **Party 2:** Public key or other signature verification information for the other participant.
- **Amount 1:** The amount of money that **Party 1** has placed in the channel
- **Amount 2:** The amount of money that **Party 2** has placed in the channel

Signature 1: **Party 1**'s signature on **Opening Transaction**

Signature 2: **Party 2**'s signature on **Opening Transaction**

2.2 Update Transaction

Update Transactions are sent back and forth between **Party 1** and **Party 2** and serve to transfer the money. Only the last of these **Update Transactions** should be posted to the bank or blockchain. When one of the parties posts an **Update Transaction**, it always results in the closure of the channel (whether the bank or the blockchain honors the **Update Transaction**, or an **Update Transaction** with a higher **Sequence Number** invalidates it).

Update Transaction:

Sequence Number: This number is incremented with each new **Update Transaction**.

Net Transfer Amount: The amount of money to transfer from **Party 1** to **Party 2** (can be negative).

Hold Period: An amount of time (or number of blocks) to wait before closing the channel and transferring funds, after one of the parties posts this **Update Transaction**.

Signature 1: Party 1's signature on **Update Transaction**.

Signature 2: Party 2's signature on **Update Transaction**.

2.2.1 Making payments

To make payments to one another, Alice and Bob pass signed **Update Transactions** back and forth.

If Alice wants to pay Bob, she adjusts the **Net Transfer Amount**, signs the **Update Transaction**, then passes it to Bob. None of this involves the **Update Transaction** being shown to anyone else, and can happen instantly. Alice and Bob can do this as many times as they want.

To actually claim the funds, either party posts the latest **Update Transaction** to the bank or the blockchain. After **Hold Period** is over, the channel closes: the **Net Transfer Amount** is subtracted from **Amount 1** and added to **Amount 2**, and the amounts are transferred back to the accounts of the participants.

This means that if Alice disappears or becomes uncooperative, Bob can still get his money out by posting the last valid **Update Transaction** he has and waiting for the **Hold Period** to end.

2.2.2 Stopping cheaters

If one of the parties posts an **Update Transaction** with a higher **Sequence Number** before the **Hold Period** ends, it overrides the older **Update Transaction**.

If Bob tries to cheat by publishing an old **Update Transaction** where he has more money than he does currently, Alice can simply publish the newer **Update Transaction**, which will have a higher **Sequence Number**.

2.3 Smart Conditions

Update Transactions can have a list of **Smart Conditions**. **Smart Conditions** are pieces of Turing-complete code that are evaluated by the bank or blockchain during the **Hold Period**. **Smart Conditions** are supplied with a piece of data, which is referred to as a **Fulfillment**. The **Smart Condition**

evaluates the **Fulfillment** and returns a **Conditional Multiplier**, which is a number between 1 and 0. They have an associated **Conditional Transfer Amount**, which is multiplied by the **Conditional Multiplier** and added to the channel's **Net Transfer Amount**.

Update Transaction:

Sequence Number: This number is incremented with each new **Update Transaction**.

Net Transfer Amount: The amount of money to transfer from **Party 1** to **Party 2** (can be negative).

Hold Period: An amount of time (or number of blocks) to wait before closing the channel and transferring funds, after an **Update Transaction** has been posted.

Conditions:

1:

Function(argument): Takes an argument and returns a number between 1 and 0.

Conditional Transfer Amount: Multiply this by the number returned by the **Function** and add it to the channel's **Net Transfer Amount**.

2: ...

Pieces of data called **Fulfillments** can be posted during the **Hold Period**. These act to "fulfill" the "conditions". **Fulfillments** only need to be signed by one of the channel participants.

Fulfillment:

Condition: Which condition does this fulfill?

Argument: Data with which to evaluate the **Smart Condition**.

When one of the parties posts a **Fulfillment**, the bank or blockchain supplies it to the corresponding **Smart Condition**. The **Conditional Transfer Amount** is multiplied by the number returned by the **Smart Condition**, and added to the channel's **Net Transfer Amount**. The **Smart Condition** is removed from the list.

2.3.1 Gas

Notice that one channel participant could send the other a **Smart Condition** that resulted in an infinite loop or other excessive use of resources. Blockchain-based smart contract systems like Ethereum[9] or Tendermint[10] use a concept

of "gas" where each step of code execution costs a small amount. Execution aborts if there is insufficient gas. Such a gas scheme could be specified here, but we believe that it is an implementation detail, and outside of the scope of this specification.

Whatever the gas scheme used, nodes should be required to pay gas upon posting a **Fulfillment**. This way, incentives are aligned so that the party who would like a certain condition evaluated pays for it.

2.3.2 Using Smart Conditions

Smart Conditions can be used to implement complex logic over channels to give them enhanced capabilities. Here is how Alice and Bob would implement a "hashlock" **Smart Condition**. Specifically, Alice wants to guarantee that she will transfer 32 coins to Bob if he can supply a string (referred to as a **Payment Secret**) that hashes to "59A1904325CCB". Alice is **Party 1** and Bob is **Party 2**.

Alice to Bob

Update Transaction:

Sequence Number: 12

Net Transfer Amount: -34

Hold Period: 8

Conditions:

1:

Conditional Transfer Amount: 32

Function(secret):

```
if hashFunction(secret) equals
"59A1904325CCB", return 1;
else return 0
```

Signature 1: Alice's signature on **Update Transaction**

2.3.3 Closing the channel

If Bob wants to close the channel at this point, he posts the **Update Transaction**, along with his and Alice's signatures. To fulfill the **Smart Condition** and have the **Conditional Transfer Amount** added to the channel's **Net Transfer Amount**, Bob must post a **Fulfillment** that causes the **Smart Condition** to return 1 during the **Hold Period**. Note that the **Fulfillment** only needs to be signed by the party posting it.

*Along with the above **Update Transaction**, Bob posts*

Fulfillment:

Condition: 1

Argument: "theSecret"

Signature: Bob's signature on **Fulfillment**

2.3.4 Fulfilling the condition without closing the channel

Of course, most of the time Bob doesn't want to close the channel right away. Bob can now prove that he could unlock the money if he wanted, so Alice might as well adjust the channel's **Net Transfer Amount** as specified by the **Smart Condition**. Bob sends the **Payment Secret** to Alice, who adjusts the channel's **Net Transfer Amount**, increments the **Sequence Number**, removes condition 1, and signs a new **Update Transaction**.

Bob to Alice

Fulfillment:

Condition: 1

Argument: "theSecret"

Signature: Bob's signature on **Fulfillment**

Both sign

Update Transaction:

Sequence Number: 13

Net Transfer Amount: -2

Hold Period: 8

Signature 1: Alice's signature

Signature 2: Bob's signature

2.3.5 Canceling the condition

Similarly, Bob can inform Alice that he will never be able to provide the secret. In this case there is no reason for them to keep passing a condition that will never be fulfilled back and forth. Bob simply makes a new **Update Transaction**, without the **Smart Condition**.

Both sign

Update Transaction:

Sequence Number: 13

Net Transfer Amount: -34

Hold Period: 8

Signature 1: Alice's signature on **Update Transaction**

Signature 2: Bob's signature on **Update Transaction**

2.4 Multihop payments

Hashlock conditions make it possible to trustlessly route payments across multiple hops. Let's say that Alice would like to transfer some funds to Charlie, but she does not have a channel open with him. If she has a channel with Bob, and Bob has a channel with Charlie, the funds can be transferred.

First, Alice sends a **Payment Secret** to Charlie:

Alice to Charlie

Payment Secret: "theSecret"

Then, Alice sends a hashlocked payment to Bob:

Alice to Bob

Update Transaction:

Sequence Number: 13

Net Transfer Amount: -2

Hold Period: 8

Conditions:

1:

Conditional Transfer Amount: -101

Function(secret):

```
if hashFunction(secret) equals
"73B88F8C24EAA", return 1;
else return 0
```

Signature 1: Alice's signature on **Update Transaction**

Notice that Alice has sent Bob 101 coins instead of 100, as Bob charges her

a 1% fee for routing payments.

Now, Bob sends the payment along to Charlie (Bob is **Party 1** and Charlie is **Party 2** in their channel):

Bob to Charlie

Update Transaction:

Sequence Number: 42

Net Transfer Amount: 56

Hold Period: 10

Conditions:

1:

Conditional Transfer Amount: 100

Function(secret):

```
if hashFunction(secret) is equal to
"73B88F8C24EAA", return 1;
else return 0
```

Signature 1: Bob's signature on **Update Transaction**

To claim the payment, Charlie can post this **Update Transaction**, along with the **Payment Secret** that Alice sent him.

Charlie posts

Update Transaction:

Sequence Number: 42

Net Transfer Amount: 56

Hold Period: 10

Conditions:

1:

Conditional Transfer Amount: 100

Function(secret):

```
if hashFunction(secret) is equal to
"73B88F8C24EAA", return 1;
else return 0
```

Signature 1: Bob's signature on **Update Transaction**

Signature 2: Charlie's signature on **Update Transaction**

Charlie also posts

Fulfillment:

Condition: 1

Argument: "theSecret"

Signature: Charlie's signature on **Fulfillment**

Once Charlie has posted the **Update Transaction**, Bob can see the **Payment Secret** and unlock his hashlocked funds from Alice. In this way, a network of nodes are able to exchange payment trustlessly with one another. While Alice and Bob both need to have channels open with the same blockchain or bank, and Bob and Charlie need to have channels open with the same blockchain or bank, Alice and Charlie do not. As long as the banks involved hold money in escrow and honor **Update Transactions** for their customers, and the blockchains involved handle the smart contract logic to do the same, a payment network can be created that spans banks and blockchains.

2.4.1 Multihop payments across currencies

In the multihop payment example above, Alice and Bob's channel does not necessarily need to use the same bank or blockchain as Bob and Charlie's channel. The channels don't even need to hold the same store of value.

If Alice wants to send Charlie some euros, and Charlie and Bob have a euro channel open, it can be done. Alice needs to know how many dollars she needs to send Bob to have him send Charlie the right number of euros (Bob calculates this from his exchange rate and fee). Alice sends the hashlocked dollars to Bob, and Bob sends hashlocked euros to Charlie. If everything goes smoothly, Charlie reveals the **Payment Secret** to Bob as usual.

This can also be used to connect two parties transacting in the same currency, across hops of another currency. Let's say that Alice wants to send dollars to Doris, and she can reach Doris through Bart and Conrad, who have a channel open on the dogecoin blockchain. Alice can send Bart hashlocked dollars, while Bart sends Conrad hashlocked dogecoins. Conrad then sends Doris hashlocked dollars. Doris can reveal the hashlock secret allowing Conrad and Bart to unlock their payments as usual.

This technique could be very powerful for providing payment connectivity between separate groups of people using non-crypto currency channels, as it will probably be a lot quicker to open a channel on a blockchain vs with a bank. Enterprising individuals can identify parts of the network lacking connectivity and supply it, earning transaction fees for their efforts.

Related work

Payment channels are based on the idea of commercial credit. Commercial credit allows two parties to consolidate a large number of smaller payments into

one larger payment which is made periodically. Clearinghouses extend this instrument by placing funds in escrow so that two parties can exchange a large number of transactions without having to trust each other. Payment channels use cryptography and game theory to allow payments to be consolidated trustlessly without a third party clearinghouse.

Some of the first formalization of the role of a clearinghouse into the concept of a payment channel occurred in the Bitcoin community with Mike Hearn's work on micropayment channels[2][3], Alex Aakselrod's work on chained micropayment channels[4] and C. J. Plooy's system Amiko Pay[5]. Further innovation appeared with Poon and Dryja's Lightning Network[7] and Decker and Wattenhofer's Duplex Channels[6]. All of these protocols are designed for Bitcoin, whose limited scripting capabilities demand complicated specifications. Interledger[1] is the only protocol we know of specifying a multihop payment channel system generalized across stores of value. It is a bit more complex than the protocol in this paper, and requires a ledger (a bank or blockchain) to be contacted for every transaction.

Zackary Hess's work in Flying Fox[11] deserves special mention, because his channel specification forms the basis of the one in this paper.

None of the above work includes Turing-complete **Smart Conditions** ("smart contracts" are a similar concept).

Acknowledgements

Zackary Hess, for coming up with much of the definition of the **Basic Channel** as part of Flying Fox[11], a channel-based cryptocurrency and prediction market. Jae Kwon, for years of advice and guidance on the theory, implementation, and philosophy of cryptocurrency. Anke Tremback, for editing the first complete draft of this paper. Alice Townes, for editing and feedback from a legal and finance perspective.

Glossary

Conditional Multiplier A number between 1 and 0 returned by **Smart Conditions** when supplied with a **Fulfillment**. The **Conditional Transfer Amount** is multiplied by the **Conditional Multiplier** and added to the **Net Transfer Amount**. 9, 15, 16

Conditional Transfer Amount An amount included in **Smart Conditions** which is multiplied by the **Conditional Multiplier** and added to the channel's **Net Transfer Amount** when a **Smart Condition** is evaluated. 9, 10, 15, 16

Fulfillment A piece of data used as input to a **Smart Condition**. This can be posted at any time during the **Hold Period**, and only needs to be signed by one of the parties. 8–11, 14–16

Hashlock Condition A **Smart Condition** that hashes its argument, usually a **Payment Secret** and compares the hash to a pre-specified string. If

the hash and the pre-specified string match, the **Smart Condition** returns 1, and the bank or blockchain adds the corresponding **Conditional Transfer Amount** to the channel's **Net Transfer Amount**. If they do not match, the **Smart Condition** returns 0, and nothing is added to the **Net Transfer Amount**. 16

Hold Period A time period included in the **Update Transaction**. The bank or blockchain must wait this amount of time before transferring any money when closing the channel. This provides a chance for one of the parties to counteract a cheating attempt where the other party posts an old **Update Transaction**. If one of the parties posts a newer **Update Transaction** with a higher **Sequence Number** before the **Hold Period** is over, it will override the older **Update Transaction**. 6–10, 15, 16

Net Transfer Amount An amount included in **Update Transactions** which specifies how much money to transfer from **Party 1** to **Party 2** when the channel closes. If it is negative, funds are transferred in the other direction. 8–11, 15, 16

Opening Transaction A message signed by both parties to create a channel. One of the parties posts this to the bank or blockchain. **Opening Transactions** serve to identify the parties and place funds in escrow. 6, 7, 16

Payment Secret A secret shared between the source and destination of a multihop payment. The source hashes the **Payment Secret** and gives the hash to the intermediary nodes. Intermediary nodes use it to create **Hashlock Conditions** between all the intermediary nodes involved in the multihop payment. The destination reveals the **Payment Secret** to the last intermediary node in order to claim the payment. The last intermediary node reveals it to the second-to-last and so on back to the source. 10–16

Sequence Number An integer included the **Update Transaction** which must be incremented with each new **Update Transaction**. The bank or the blockchain uses the **Sequence Number** to ascertain the ordering of **Update Transactions**. An **Update Transaction** with a higher **Sequence Number** will always override one with a lower **Sequence Number**. 6–8, 11, 16

Smart Condition A piece of Turing-complete code included in the **Update Transaction**. The **Smart Condition** is evaluated by the bank or blockchain during the **Hold Period**. It returns a **Conditional Multiplier** when supplied with a **Fulfillment**. The **Smart Condition** has an associated **Conditional Transfer Amount**, which is multiplied by the **Conditional Multiplier** and added to the channel's **Net Transfer Amount** when the **Smart Condition** is evaluated. 8–11, 15, 16

Update Transaction A message signed by both parties, updating the state of a channel. One of the parties posts this to the bank or blockchain to close

the channel. However, before this happens an infinite number of **Update Transactions** can be exchanged between the two parties. 6–14, 16, 17

References

[1] *A Protocol for Interledger Payments*
Stephan Thomas, Evan Schwartz
`https://interledger.org/interledger.pdf`
2015

[2] *Micropayment Channel*
Bitcoin Wiki Contributors
`https://bitcoin.org/en/developer-guide#micropayment-channel`
2014

[3] *[ANNOUNCE] Micro-payment channels implementation now in bitcoinj*
Mike Hearn
`https://bitcointalk.org/index.php?topic=244656.0`
2013

[4] *Decentralized networks for instant, off-chain payments*
Alex Akselrod
`https://en.bitcoin.it/wiki/User:Aakselrod/Draft`
2013

[5] *Amiko Pay*
C. J. Plooy
`http://cornwarecjp.github.io/amiko-pay/doc/amiko_draft_2.pdf`
2013

[6] *A Fast and Scalable Payment Network with Bitcoin Duplex Micropayment Channels*
Christian Decker, Roger Wattenhofer
`http://www.tik.ee.ethz.ch/file/716b955c130e6c703fac336ea17b1670/duplex-micropayment-channels.pdf`
2015

[7] *The Bitcoin Lightning Network: Scalable Off-Chain Instant Payments*
Joseph Poon, Thaddeus Dryja
`https://lightning.network/lightning-network-paper.pdf`
2015

[8] *Ad-hoc On-Demand Distance Vector Routing*
Charles E. Perkins, Elizabeth M. Royer
`https://www.cs.cornell.edu/people/egs/615/aodv.pdf`

[9] *A Next-Generation Smart Contract and Decentralized Application Platform*
Vitalik Buterin
`https://github.com/ethereum/wiki/wiki/White-Paper`
2014

[10] *Tendermint: Consensus without Mining*
Jae Kwon
http://tendermint.com/docs/tendermint.pdf
2015

[11] *Flying Fox*
Zackary Hess
https://github.com/BumblebeeBat/FlyingFox
2015