LN as a Directed Graph

Single-Funded Channel Topology

Thaddeus Dryja <rx@awsomnet.org>

DG717

2016-04-11

LN recap

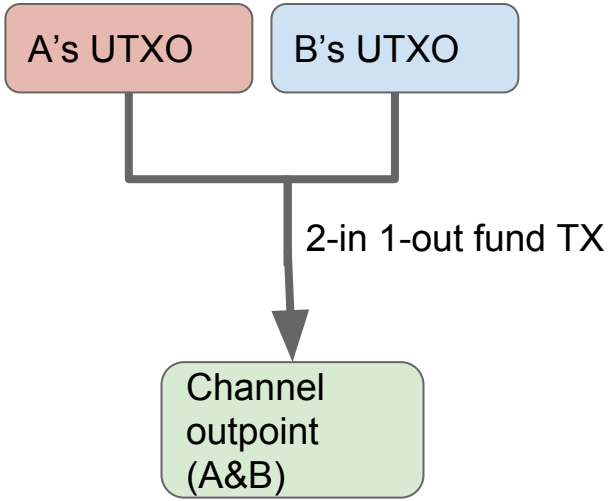
- Make channel between 2 nodes
- Update channel state
- Close co-operatively via interactive multisig
- Close non-cooperatively with stored signatures and timeout

LN recap

- Multi-hop via trustless HTLC
- Channels are edges of a graph
- Traverse graph via HTLCs
- Off-chain payments:
  - In non-adversarial conditions, big speed and throughput increases
  - In adversarial conditions, falls back to similar security as underlying network

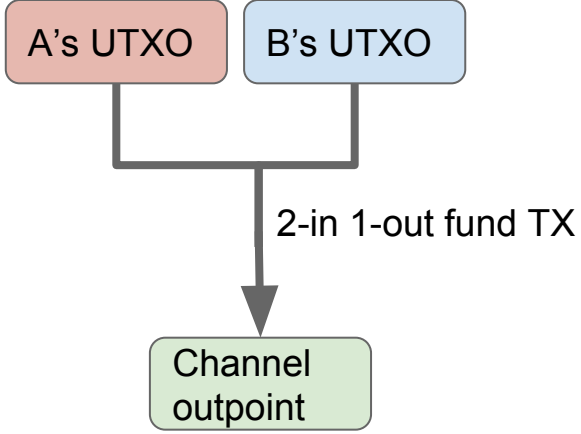
How to build the network

- A and B want to make a channel
  - But wait… do they? Why?
  - Do they know each other?
- Dual-funded channels are useful but tricky



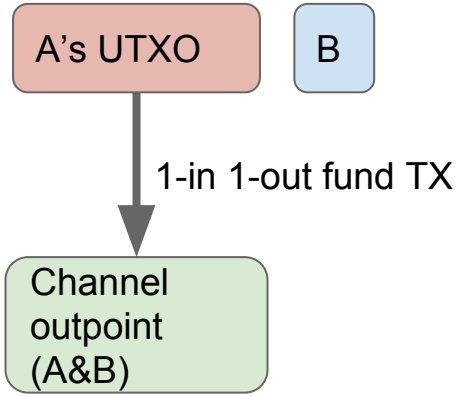
Dual-funded channels

- Some trust involved
- You're signing your UTXOs away!
- Do they really want a channel? Or do they just want to waste your time (and your money's time?)
- Timing, identity, fun stuff



Single-funded channels

- Simple: A asks B for a pubkey, then tells B about the channel.
- B has nothing at risk, never signs their UTXOs.
- Couple payment and channel creation. While A funds completely, initial state allocates money to B.



Channel exhaustion

- Exhausted channels must be avoided in general
- Makes attacks free:
  - state 5 I have 0 coins, in state 4 I had >0 coins.
  - Broadcast state 4! I have nothing to lose!
- Don’t accept state transitions that would exhaust the other side
- Don’t try to exhaust from your side

Channel exhaustion exception

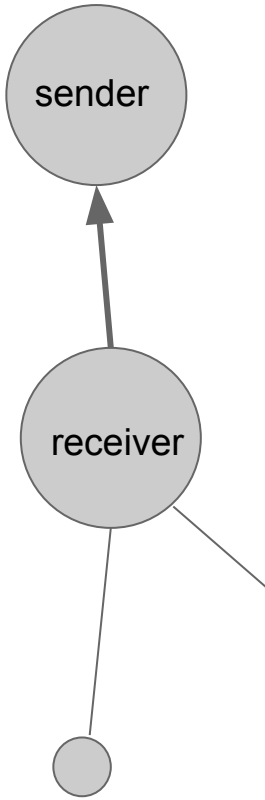
- At state 0, channel exhaustion is OK.
- There is no previous non-exhausted state to broadcast, so there's no attack
- Helps flexibility of single-funded channels:
  - A can make a "full push channel" to B. Open a 1 coin channel and send the whole coin over. (A is exhausted)
  - A can open a "zero push channel" to B. Open a 1 coin channel and send nothing. (B is exhausted)

Simplest possible UI

- pay(dest, amt)
- Can payment be made with existing channels?
- If yes, do that! (whole point of LN) Done.
- If no, make a full push channel to dest. Done.
- Works!

Directed -> Undirected

- Channels can start out exhausted, but won’t be for long
- Anyone trying to send you funds can use those channels
- The arrows in the directed graph may not point the direction you think
- Can make cycles



TX efficiency

- Over-funding channels can help
- Lots of channels between the same 2 nodes is kindof ugly.
- But, channels are super cheap to make (only 12 bytes more than p2wpkh)
- Also cheap to close (2of2 multisig, but vsize only ~25 bytes more.

TX efficiency

- All the complex transactions (preimages, HTLCs, timeouts) will basically never happen.
- Oxygen mask TXs

- Open channel without payment only for new use cases

The real problem

- People have been talking (arguing?) about scalability for a while
- Propagation, capacity, centralization, block size, segwit, XT, hard forks, soft forks, salad forks
- Everyone’s been ignoring the REAL problem with bitcoin’s scalability...

Downward scalability

- Bitcoin only has 8 decimal places (“satoshis”)
- 1 USD is 250,000 satoshis
- A micro-payment is 10^{-6} $USD_{(\mu SD)}$ = **<u>0.25</u>** satoshi

<u>Bitcoin does not support micro-payments.</u>

(unless the price of 1 bitcoin falls below $100)

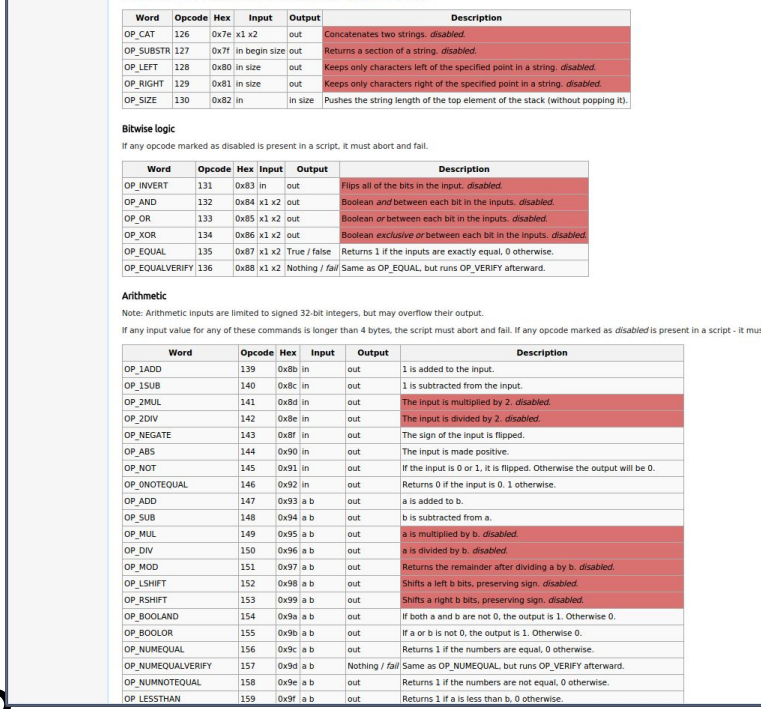
Sub-satoshi

- Sure you could TRUST your channel counterparty. But where's the fun in that?
- How to subdivide without trust?
- Probabilistic payments
  - Rivest's "Peppercoin" (early 2000s)
  - Based on inequality of LSBs of an RSA sig
- Previous ideas of this in bitcoin

(Rafael Pass and abhi shelat 2015)

Probabilistic payment

- How to pay someone a satoshi with an agreed upon probability?
- No trusted 3rd party
- Where to get randomness?
- Limited opcodes (fun opcodes all disabled)



Word	Opcode	Hex	Input	Output	Description
OP_CAT	126	0x7e	x1 x2	out	Concatenates two strings. *disabled.*
OP_SUBSTR	127	0x7f	in begin size	out	Returns a section of a string. *disabled.*
OP_LEFT	128	0x80	in size	out	Keeps only characters left of the specified point in a string. *disabled.*
OP_RIGHT	129	0x81	in size	out	Keeps only characters right of the specified point in a string. *disabled.*
OP_SIZE	130	0x82	in	in size	Pushes the string length of the top element of the stack (without popping it).

Bitwise logic

If any opcode marked as disabled is present in a script, it must abort and fail.

Word	Opcode	Hex	Input	Output	Description
OP_INVERT	131	0x83	in	out	Flips all of the bits in the input. *disabled.*
OP_AND	132	0x84	x1 x2	out	Boolean *and* between each bit in the inputs. *disabled.*
OP_OR	133	0x85	x1 x2	out	Boolean *or* between each bit in the inputs. *disabled.*
OP_XOR	134	0x86	x1 x2	out	Boolean *exclusive or* between each bit in the inputs. *disabled.*
OP_EQUAL	135	0x87	x1 x2	True / false	Returns 1 if the inputs are exactly equal, 0 otherwise.
OP_EQUALVERIFY	136	0x88	x1 x2	Nothing / *fail*	Same as OP_EQUAL, but runs OP_VERIFY afterward.

Arithmetic

Note: Arithmetic inputs are limited to signed 32-bit integers, but may overflow their output.

If any input value for any of these commands is longer than 4 bytes, the script must abort and fail. If any opcode marked as *disabled* is present in a script - it mus

Word	Opcode	Hex	Input	Output	Description
OP_1ADD	139	0x8b	in	out	1 is added to the input.
OP_1SUB	140	0x8c	in	out	1 is subtracted from the input.
OP_2MUL	141	0x8d	in	out	The input is multiplied by 2. *disabled.*
OP_2DIV	142	0x8e	in	out	The input is divided by 2. *disabled.*
OP_NEGATE	143	0x8f	in	out	The sign of the input is flipped.
OP_ABS	144	0x90	in	out	The input is made positive.
OP_NOT	145	0x91	in	out	If the input is 0 or 1, it is flipped. Otherwise the output will be 0.
OP_0NOTEQUAL	146	0x92	in	out	Returns 0 if the input is 0. 1 otherwise.
OP_ADD	147	0x93	a b	out	a is added to b.
OP_SUB	148	0x94	a b	out	b is subtracted from a.
OP_MUL	149	0x95	a b	out	a is multiplied by b. *disabled.*
OP_DIV	150	0x96	a b	out	a is divided by b. *disabled.*
OP_MOD	151	0x97	a b	out	Returns the remainder after dividing a by b. *disabled.*
OP_LSHIFT	152	0x98	a b	out	Shifts a left b bits, preserving sign. *disabled.*
OP_RSHIFT	153	0x99	a b	out	Shifts a right b bits, preserving sign. *disabled.*
OP_BOOLAND	154	0x9a	a b	out	If both a and b are not 0, the output is 1. Otherwise 0.
OP_BOOLOR	155	0x9b	a b	out	If a or b is not 0, the output is 1. Otherwise 0.
OP_NUMEQUAL	156	0x9c	a b	out	Returns 1 if the numbers are equal, 0 otherwise.
OP_NUMEQUALVERIFY	157	0x9d	a b	Nothing / *fail*	Same as OP_NUMEQUAL, but runs OP_VERIFY afterward.
OP_NUMNOTEQUAL	158	0x9e	a b	out	Returns 1 if the numbers are not equal, 0 otherwise.
OP_LESSTHAN	159	0x9f	a b	out	Returns 1 if a is less than b, 0 otherwise.

Probabilistic payment

- Op code to use: OP_SIZE
- Basic idea: 2-party envy-free cake cutting (divide and choose)
- Divider picks a pre-image length

(pre-image bytes are from /dev/urandom or can be from some hash tree or whatever)

- Divider sends hash
- Chooser picks pre-image length based on hash
- Choose the wrong length, get the Satoshi

Probabilistic payment

- Making it iterative needs a few more steps
- Done in a “limbo” channel (can’t be closed un-cooperatively for a number of blocks)
- Dave divides, Carol chooses
- Dave is paying Carol 0.5 satoshi

(0.000000005 BTC) (5 nanoBTC)

Probabilistic payment

- Carol makes random Y1, Y2, (normal length) hashes both and sends the hashes to Dave
- Dave makes X (L bytes long), hashes, and makes 2 output scripts. (3 paths to spend each) Puts script in txout, txin is channel outpoint, Signs both, sends txs to Carol

Script 1 / Tx 1

SigCarol && 10 blocks ||

SigDave && Y1 ||

SigDave && len(X) == 20

Script 2 / Tx 2

SigCarol && 10 blocks ||

SigDave && Y2 ||

SigDave && len(X) == 21

Carol’s choices

- Carol now has 2 half signed TXs. She can:

1. Nothing
2. Sign & broadcast both
3. Sign & broadcast TX1
4. Sign & broadcast TX2
5. Choose TX1, sign and send sig, Y1 to Dave
6. Choose TX2, sign and send sig, Y2 to Dave

1: Nothing

- Channel is in limbo for 20 blocks or so. Dave is like “what the heck Carol” and closes after that

2: Sign & broadcast both

- This is really the same as signing and broadcasting only one
- She’s just letting the miners choose which TX happens instead of picking herself
- Once one gets into a block, Dave proceeds as if that one is chosen

3/4: Sign & broadcast a chosen TX

- If she chose the WRONG length, she wins after 10 blocks
- If she chose the RIGHT length, Dave sweeps immediately

Script 1 / Tx 1

SigCarol && 10 blocks ||

SigDave && Y1 ||

SigDave && len(X) == 20

Script 2 / Tx 2

SigCarol && 10 blocks ||

SigDave && Y2 ||

SigDave && len(X)==21

5/6: Sign & send sig and other Y preimage

- Choose 20, sign TX1 and send Y2 to Dave
- Claims TX1, Revokes claim on TX2
- Carol can no longer sign and broadcast TX2 as she loses even if len(X)==21
- Dave can broadcast TX1 and only TX1. He reveals X and they say GG

Script 1 / Tx 1

SigCarol && 10 blocks ||

SigDave && Y1 ||

SigDave && len(X) == 20

Script 2 / Tx 2

SigCarol && 10 blocks ||

SigDave && Y2 ||

SigDave && len(X)==21

Probabilistic payment: iterate

- After Carol sends Y1, sig2, Dave needs to reveal X. They both know who gets the satoshi
- The cooperate and update the channel state based on where the satoshi goes. If they don’t, either can close at the updated state
- Note that for sub-satoshi, these TXs actually have 2 outputs -

TX1(script1:49, script2:51) TX2(script2:49, script1:51)

Probabilistic payment: timing

- When Carol gets the 2 TXs from Dave, she considers that a payment
- Carol can unilaterally close with the probability of an incremented satoshi
- Deliver goods, then respond with length choice

Probabilistic payment: probabilities

- Just described 0.5 chance; for 0.333 chance, there are 3 scripts, for lengths 20, 21, 22
- Can have 0.25, 0.1... Can’t really do 0.001 though (preimages get too long to manage)
- But does get us at least one OOM scalability

Probabilistic payment: probabilities

- Relies on collision resistance of hash function; if Dave can do ~2^{80} work and find a collision with different lengths, he can cheat
- OK, just use hash256 instead of hash160…
- But ~2^{80} work costs way more than a satoshi
- Can actually do 12/13 byte pre-images! 2^{96} work for Carol to break (pre-image attack), and would be 2^{48} for Dave to collide… but there are no colliding 12 byte pre-images

Summary

- Single funded channels are easier
- Exhausted channels are bad, but exhausted-on-open is OK
- Network can grow with single funded, TXs are cheap
- The real scalability problem has been ignored… until now
- Pre-Image Length Probabilistic Payments (PILPP?) can scale Bitcoin into the true micro-payment range
- Trustless nano-payments (femto, atto?) possible with chains of PILPPs
- Avoid divisibility forks (hard / soft / spork)
- May have low demand with current price of Bitcoin, but we

Questions

- There probably are some
- But they’re not written on this slide

- Because causality

Thanks to: DG717 for hosting, Denise & Michael for setting this up, sponsors for pizza, and everyone for coming!