How Wallets Can Handle Real Transaction Fees

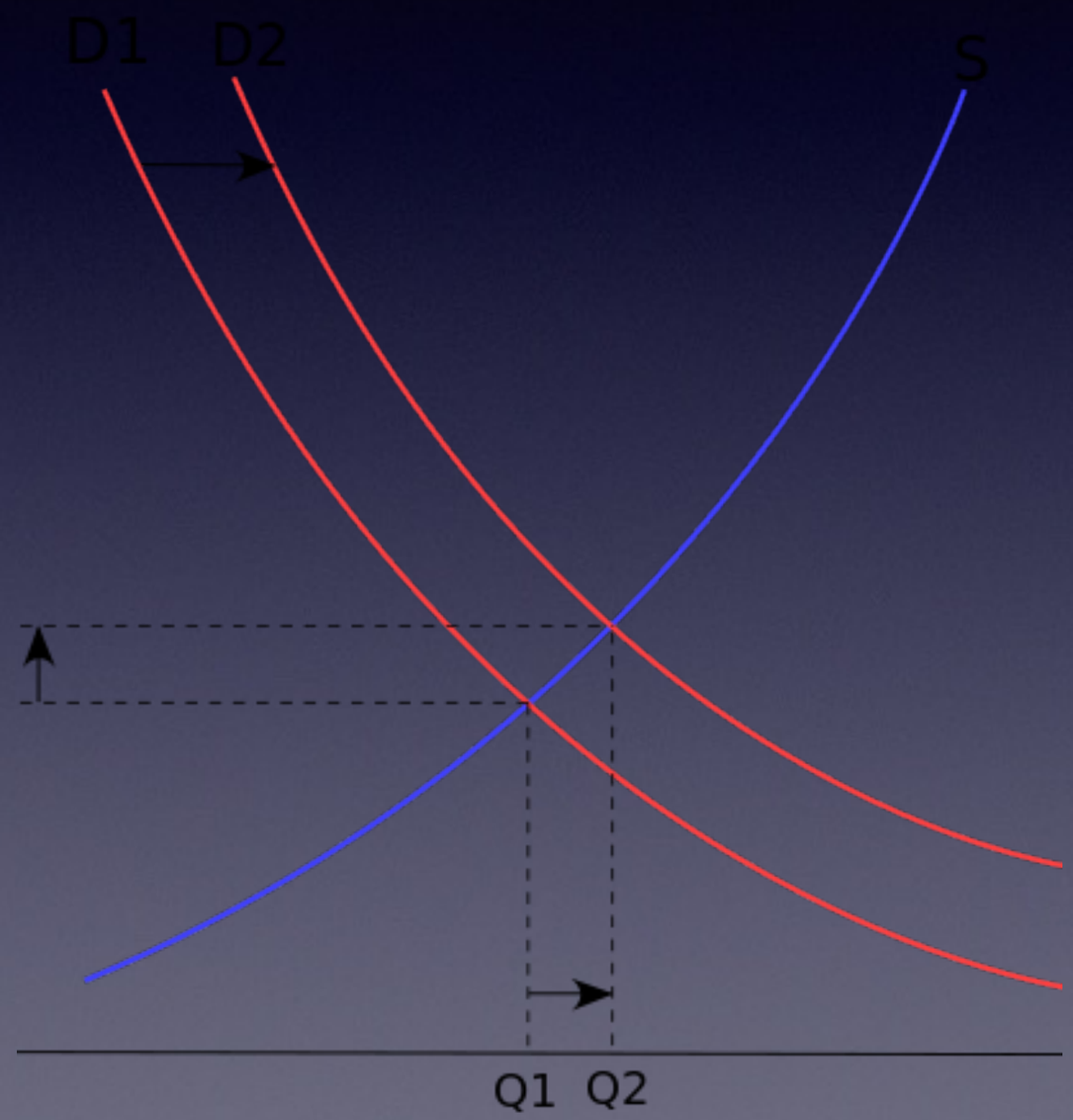
Thoughts by Bram Cohen

Ground rules

- We're talking about consumer wallets
- No microchannels, but they should come later
- Replace by fee in effect
- Aggressive replace by fee in effect

What should transaction fees be?

- Supply and demand
- But supply is noisy, demand is noisy
- Demand has day/night and weekly cycle
- There should be a patience tradeoff - if you're willing to wait longer your fees will on average be lower.



What should wallet UX be?

- Needs to specify max fee and how long until giving up
- Needs to have state of 'failed send'
- Currently no max height in transactions! Needs protocol extension!

Information which can be used

- Past transaction fees
- Current transactions in mempool

- Past transaction fees which the local client has paid
- How long the current payment attempt has been going on

Problems with possible inputs

- Past fees can result in fees getting stuck at a high amount, which peers continue based on tradition
- Past fees Fees can get trivially pumped by any miner
- For mempool SPV clients have to trust on full nodes, creating a trivial attack and incentive to do it
- Most conservative to stick with all locally available info

Using just local info

- Pick a starting point which is de minimis for your first transaction or 1/2 (or less, configurable) your last fee paid if you've sent coin before
- B = max number of blocks from start before giving up, S = starting fee, M = max fee
- For each new block at height H from the start, post a new transaction with fee e^(lg(S) + (lg(M) - lg(S)) * H/B)
- To avoid artifacts when multiple wallets use the same magic numbers, do this before the first block: pick V in [0, 1], let S = e^(lg(S) + (lg(M) - lg(S)) * (V/(V+B)))

Handling utxo combining

- Matters surprisingly little. Extra size will happen eventually for every extra coin in wallet regardless of whether it's merged sooner or later
- Transactions to combine wallet utxos during times when fees are low might be a good idea
- Extension could help a bit with size: Using Schnorr, allow a single signature with multiple inputs
- A 'better' extension would allow inputs with the same key to share a signature, **if used properly** only pushes up merge reveal a little bit
- Another extension would allow any public key which has ever been revealed in the block chain to date to not have to be revealed again. This is a bad idea!