Some key numbers in determining the viability of stateless ethereum are (1) how big the witnesses will be, and (2) what witness size the network can support. (1) must be less than (2). To that end there are a couple different proposals for making witnesses as small as possible, one of them is code merkelization. It would take a while to get perfect numbers on the impact of merkelization so I tried to come up with a quick estimation of the benefits.

I wrote some logic to record which bytes of contract bytecode are accessed during transaction execution, and then ran it over a sampling of recent blocks. I scanned every 50th block, starting with 9375962 and ending with 9775962 (a total of 8001 blocks).

This post is kind of a grab-bag, I don't have any conclusions but wanted to put this data somewhere public so we can refer to it later. Here are some results which seemed useful:

Code merkelization could have a large impact:

We all assumed this but it's nice to put some numbers behind that assumption:

The above histogram bins blocks by the total size of all the contract bytecodes which were executed in that block. The largest block by this metric is 9697612, which executes 1.7MB of contract bytecode. Code merkelization would likely have a massive impact here, as that block only ever executes 220044 (12%) of those bytes. 220044 is near the high end:

This histogram bins blocks by the number of contract bytes which are actually accessed during execution. I should note that actually witnessing these bytes requires proving them! These numbers provide an upper bound on the potential savings, no accumulator will ever be able to reduce the code witness size of this block by more than 88%.

What does the average block look like?

There are a few blocks which use a high proportion of the contract bytes but those are tiny blocks which don't touch many blocks to begin with. Most blocks execute ~15% of the bytes of the contracts they execute.

This is great news! No block that I scanned accessed more than 251403 bytes. That's not insignificant, but it almost certainly fits into whatever our witness size budget turns out to be. So, we should be able to witness contract bytecode without massively increasing gas costs for existing transactions.

For the average transaction, gas costs will not greatly increase:

Vitalik has proposed charging transactions 3 gas per witnessed byte. This would lead to a theoretical maximum block size of under 3.2MB (assuming a 10M block gas limit) which seems too large but let's stick with that number for now. If transactions were charged 3 gas for each byte of bytecode which they accessed, here's how gas prices would increase across the set of transactions this scan found:

For most transactions, the impact is minimal. Some transactions would see their gas cost increase by ~30%. Of course, this is an underestimate. For one, I'm not counting the witness bytes required to prove the accessed contract bytes. For another, state accesses also must be witnessed.

We need to special-case extcodesize

Again, I think everyone already assumed this but it's nice to be able to back those assumptions up with data. The naive implementation of extcodesize

just counts the number of bytes in the target contract, and would require witnessing the entire bytecode.

I recorded most calls to extcodesize

[1] to estimate how much of an impact this would have have on witness size. For block 9697612 this would witness 1.4MB of data, just to prove extcodesize

calls! 52 of the blocks I scanned (<1%) would use over 1MB of data for extcodesize

, and 1012 of them (~13%) would use 500KB of data.

extcodesize

appears to be widely used. Unless we want to break a large proportion of the current transactions we'll need to provide some way of succinctly proving the codesize when we rebuild the account trie. This could be an explicit field in the account structure. If we merkelize using fixed-size chunks a proof of the right-most chunk would be enough.

[1] I'm only recording countracts which had extcodesize calls on them, and which were also executed. Contracts which had extcodesize

called on them but which never executed any bytes are not counted, so the true numbers are even higher than reported here.

Some additional complications:

- Not all JUMPDEST

bytes are valid jumpdests, some of them are part of PUSHDATA

and cannot be jumped to. At the beginning of execution geth scans through the entire contract and computes the set of valid JUMPDEST

s. Trinity does something which touches fewer bytes, but in order to merkelize bytecode both of them will have to come up with a new strategy. I forget who told me this strategy, I think it was Piper? But if we merkelize bytecode by chunking it up, each chunk could start with an extra byte, the offset of the first valid JUMPDEST

in that chunk.

- I'm not counting the initcode of CREATE

calls, since the only ways to get an executable initcode are already witnessed.

- In these sums I'm including bytes from contracts which were created in the same block. This makes the numbers larger than they should be, those contract bytes won't need to be witnessed, but I think correctly handling this edge case would take more work to fix than it improves the results.