@AlexeyAkhunov mentioned this in the last call and I thought I would expand on it:

Currently, geth performs validation on incoming transactions before it propagates those transactions on to other peers. This is done to prevent malicious nodes from spamming the network with transactions which will never be accepted.

A naive implementation of a stateless client would reproduce that validation, but it requires knowing the current state of the sender:

- the transaction's nonce

must be greater than or equal to the sender's nonce

- the sender must have a high enough balance to pay for the transaction

If the client does not already know the current state of the account it must fetch the state over the network. This has a couple ill effects. It slows down transaction propagation and it allows malicious actors to perform DOS attacks. A transaction is relatively small. You could perform traffic amplification if fetching the state required to verify the transaction used up more bandwidth than the transaction itself. This is especially important because it's cheap to generate random accounts and sign transactions from them. It's easy for you to create the invalid transaction, but expensive for the remote peer to discover that the transaction is invalid.

There are a couple approaches Stateless Ethereum might take:

1. Don't verify transactions, just propagate anything which has the correct format and a valid signature. Do this on the theory that it's already possible to DOS nodes and flush their mempools by first creating a thousand accounts and then sending a bunch of transactions from each. The miners who eventually receive those transactions can easily check for validity.

2. Do the naive thing, when a transaction arrives fetch any state needed to validate it before forwarding it along to peers. If a peer sends you a transaction which turns out to have been invalid (the sender doesn't exist) kick that peer. Only verify one transaction at a time from each peer, to limit the amount of work they can force you to do.

3. Require transactions to include merkle proof of the sender's account (it doesn't have to be part of the signed data!). The proof can be out of date, a stateless client which has been receiving witnesses can easily update the proof and check that the transaction is still valid before forwarding the transaction along. They could also not

update the proof, anyone with the header chain can still check that the transaction used to be valid, and that's probably enough to keep transaction floods as expensive as they are now.

1. Any of the above, but also require transactions to include a low-difficulty proof of work. This doesn't prevent DOS attacks but at least makes them more expensive.

2. Something fancy. If we go the incentivized state retrieval route then there might already be a way of paying other peers. Transactions could include some kind of payment for forwarding them, or the sending peer could be slashed if the transaction turned out to be invalid.

I prefer (3), but I wonder what other people think.