One benefit of the current Merkle tree design for collations is that you can prove that a transaction was included in a Merkle tree even before the light client execution process relays the information that the transaction was executed. If the execution process is occasionally delayed behind transaction execution, as might be the case if short-range forks are frequent and/or cross-shard transactions can happen quickly, then the fact that a transaction was included can by itself tell users a lot of useful information about the state. For example, consider the following story:

- Alice has 50 ETH at block N.
- There is no transaction spending Alice's coins after block N, except for transaction T in block N+5.
- Transaction T sends 50 ETH to Bob.

Here, Bob can immediately know that he has 50 ETH, even if the execution process has not yet progressed beyond block N.

However, the gas rules in the current proposal work in a way that unfortunately nullifies this advantage. Here is how the current proposed gas rules work:

- A collation has some MAX_SIZE, eg. 1048576 bytes, and there is some exchange rate of bytes to gas, eg. 40.
- If a collation uses D bytes, then when the execution process starts executing the collation, it starts executing with (1048576 - D) * 40 gas. If the gas runs out, then all subsequent transactions in the collation are not executed.

This is problematic because a Merkle proof of inclusion no longer proves that the transaction will actually be executed, preventing Bob from knowing that he has 50 ETH (unless the transaction is the very first one in the entire collation).

I propose two possible remedies:

1. Every transaction has a guaranteed allotment of gas, which is proportional to 40 times the number of bytes in the transaction. It is considered acceptable behavior to pad a transaction with zero bytes to increase its gas allotment if needed.
2. Every transaction has a guaranteed allotment of gas, which is proportional to $k < 40$ (eg. 10) times the number of bytes in the transaction. Transaction senders are welcome to ask for more gas, but this gas also comes from this pool of (1048576 - D) * 40 gas, and so availability of this gas is not guaranteed until execution time or unless the client downloads and checks the entire collation up until the transaction.

(1) has the main weakness that it is inefficient. The inefficiency is less than it seems, because hashing many zero bytes many times over to create the Merkle tree is a fairly memoizable operation, and even in the current proposal there are expected to be zero bytes at the end of a collation, but some overhead still remains.

(2) has the weakness that the ability to Merkle-prove that fact that a transaction will be executed only applies to transactions spending low amounts of gas, but it does avoid the efficiency losses of (1).