Implementing Replace-By-Fee (BIP 125) in LDK Node

Rita Anene

You broadcast a transaction.

Fees spike.

Six hours pass.

It's still unconfirmed.

The fix exists. It's called Replace-By-Fee.

- BIP 125 lets you replace an unconfirmed transaction with one paying a higher fee
- LDK Node had no support for this
- PR #628 — Enhance onchain transaction management

First, what BIP 125 actually says

- A transaction signals RBF by setting nSequence < 0xFFFFFFFE on any input
- This is a signal, not a guarantee — nodes can ignore it
- LDK Node signals RBF on all onchain transactions by default
- So every outbound tx is already bumpable — the question is how

The replacement rules

Fee-rate rule	Absolute fee rule	Bandwidth rule
The replacement must pay a higher fee rate than the original	The replacement's absolute fee must also be higher, not just the rate	The replacement must pay relay fees for every tx it evicts, not just itself

Who does what

LDK Node owns

- Payment store lookup
- Guards: Onchain / Outbound / Unconfirmed
- Fee floor (old_fee_rate + RELAY_FEE)
- 1.5x sanity cap retry logic
- Broadcast
- Store update with new txid

`build_fee_bump(txid)`

BDK owns

- Replacement tx assembly
- BIP125 fee-rate floor enforcement
- FeeRateTooLow error
- sign() + extract_tx()

Architecture

LDK Node owns the policy. BDK owns the transaction.

How an outbound tx moves through LDK Node

Two foundational decisions

The store

Existing PaymentStore tracks current state. We needed a watch list, payments not yet final, carrying conflict history.

Bumpable tx detection

Three conditions must all be true:

- Onchain
- Outbound
- Unconfirmed

The FeeRate decision

A hybrid approach was implemented

Auto-retry

- Hidden complexity
- Risk of surprising fee escalation

Surface to user

- Explicit
- Caller in control
- Pushes complexity up

Thank you

Any questions?

PR #628 · github.com/lightningdevkit/ldk-node/pull/628

GitHub: Camillarhi