Exhibit 109

Transaction Cost

To protect the XRP Ledger from being disrupted by spam and denial-of-service attacks, each transaction must destroy a small amount of XRP. This *transaction cost* is designed to increase along with the load on the network, making it very expensive to deliberately or inadvertently overload the network.

Every transaction must specify how much XRP to destroy to pay the transaction cost.

Current Transaction Cost

The current minimum transaction cost required by the network for a standard transaction is **0.00001 XRP** (10 drops). It sometimes increases due to higher than usual load.

You can also query `rippled` for the current transaction cost.

Special Transaction Costs

Some transactions have different transaction costs:

Transaction	Cost Before Load Scaling
Reference Transaction (Most transactions)	10 drops
Key Reset Transaction	0
Multi-signed Transaction	10 drops × (1 + Number of Signatures Provided)
EscrowFinish Transaction with Fulfillment	10 drops × (33 + (Fulfillment size in bytes ÷ 16))
AccountDelete Transaction	2,000,000 drops

Beneficiaries of the Transaction Cost

The transaction cost is not paid to any party: the XRP is irrevocably destroyed.

Load Cost and Open Ledger Cost

When the FeeEscalation amendment is enabled, there are two thresholds for the transaction cost:

- If the transaction cost does not meet a `rippled` server's load-based transaction cost threshold, the server ignores the transaction completely. (This logic is essentially unchanged with or without the amendment.)
- If the transaction cost does not meet a `rippled` server's open ledger cost threshold, the server queues the transaction for a later ledger.

This divides transactions into roughly three categories:

- Transactions that specify a transaction cost so low that they get rejected by the load-based transaction cost.

- Transactions that specify a transaction cost high enough to be included in the current open ledger.
- Transactions in between, which get queued for a later ledger version.

Local Load Cost

Each `rippled` server maintains a cost threshold based on its current load. If you submit a transaction with a `Fee` value that is lower than current load-based transaction cost of the `rippled` server, that server neither applies nor relays the transaction. (**Note:** If you submit a transaction through an admin connection, the server applies and relays the transaction as long as the transaction meets the un-scaled minimum transaction cost.) A transaction is very unlikely to survive the consensus process unless its `Fee` value meets the requirements of a majority of servers.

Open Ledger Cost

The `rippled` server has a second mechanism for enforcing the transaction cost, called the *open ledger cost*. A transaction can only be included in the open ledger if it meets the open ledger cost requirement in XRP. Transactions that do not meet the open ledger cost may be queued for a following ledger instead.

For each new ledger version, the server picks a soft limit on the number of transactions to be included in the open ledger, based on the number of transactions in the previous ledger. The open ledger cost is equal to the minimum un-scaled transaction cost until the number of transactions in the open ledger is equal to the soft limit. After that, the open ledger cost increases exponentially for each transaction included in the open ledger. For the next ledger, the server increases the soft limit if the current ledger contained more transactions than the soft limit, and decreases the soft limit if the consensus process takes more than 5 seconds.

The open ledger cost requirement is proportional to the normal cost of the transaction, not the absolute transaction cost. Transaction types that have a higher-than-normal requirement, such as multi-signed transactions must pay more to meet the open ledger cost than transactions which have minimum transaction cost requirements.

See also: Fee Escalation explanation in `rippled` repository ↗.

Queued Transactions

When `rippled` receives a transaction that meets the server's local load cost but not the open ledger cost, the server estimates whether the transaction is "likely to be included" in a later ledger. If so, the server adds the transaction to the transaction queue and relays the transaction to other members of the network. Otherwise, the server discards the transaction. The server tries to minimize the amount of network load caused by transactions that would not pay a transaction cost, since the transaction cost only applies when a transaction is included in a validated ledger.

For more information on queued transactions, see Transaction Queue.

Reference Transaction Cost

The "Reference Transaction" is the cheapest (non-free) transaction, in terms of the necessary transaction cost before load scaling. Most transactions have the same cost as the reference transaction. Some transactions, such as multi-signed transactions, require a multiple of this cost instead. When the open ledger cost escalates, the requirement is proportional to the basic cost of the transaction.

Fee Levels

Fee levels represent the proportional difference between the minimum cost and the actual cost of a transaction. The Open Ledger Cost is measured in fee levels instead of absolute cost. See the following table for a comparison:

Transaction	Minimum cost in drops	Minimum cost in Fee levels	Double cost in drops	Double cost in fee levels
Reference transaction (most transactions)	10	256	20	512
Multi-signed transaction with 4 signatures	50	256	100	512
Key reset transaction	0	(Effectively infinite)	N/A	(Effectively infinite)
EscrowFinish transaction with 32-byte preimage.	350	256	700	512

Querying the Transaction Cost

The `rippled` APIs have two ways to query the local load-based transaction cost: the `server_info` command (intended for humans) and the `server_state` command (intended for machines).

If the FeeEscalation amendment is enabled, you can use the fee method to check the open ledger cost.

server_info

The server_info method reports the unscaled minimum XRP cost, as of the previous ledger, as `validated_ledger.base_fee_xrp`, in the form of decimal XRP. The actual cost necessary to relay a transaction is scaled by multiplying that `base_fee_xrp` value by the `load_factor` parameter in the same response, which represents the server's current load level. In other words:

Current Transaction Cost in XRP = `base_fee_xrp` × `load_factor`

server_state

The server_state method returns a direct representation of `rippled`'s internal load calculations. In this case, the effective load rate is the ratio of the current `load_factor` to the `load_base`. The `validated_ledger.base_fee` parameter reports the minimum transaction cost in drops of XRP. This design enables `rippled` to calculate the transaction cost using only integer math, while still allowing a reasonable amount of fine-tuning for server load. The actual calculation of the transaction cost is as follows:

Current Transaction Cost in Drops = (`base_fee` × `load_factor`) ÷ `load_base`

Specifying the Transaction Cost

Every signed transaction must include the transaction cost in the Fee field. Like all fields of a signed transaction, this field cannot be changed without invalidating the signature.

As a rule, the XRP Ledger executes transactions *exactly* as they are signed. (To do anything else would be difficult to coordinate across a decentralized consensus network, at the least.) As a consequence of this, every transaction

destroys the exact amount of XRP specified by the `Fee` field, even if the specified amount is much more than the current minimum transaction cost for any part of the network. The transaction cost can even destroy XRP that would otherwise be set aside for an account's reserve requirement.

Before signing a transaction, we recommend looking up the current load-based transaction cost. If the transaction cost is high due to load scaling, you may want to wait for it to decrease. If you do not plan on submitting the transaction immediately, we recommend specifying a slightly higher transaction cost to account for future load-based fluctuations in the transaction cost.

Automatically Specifying the Transaction Cost

The `Fee` field is one of the things that can be auto-filled when creating a transaction. In this case, the auto-filling software provides a suitable `Fee` value based on the current load in the peer-to-peer network. However, there are several drawbacks and limitations to automatically filling in the transaction cost in this manner:

- If the network's transaction cost goes up between auto-filling and submitting the transaction, the transaction may not be confirmed.
    - To prevent a transaction from getting stuck in a state of being neither definitively confirmed or rejected, be sure to provide a `LastLedgerSequence` parameter so it eventually expires. Alternatively, you can try to cancel a stuck transaction by reusing the same `Sequence` number. See reliable transaction submission for best practices.
- You have to be careful that the automatically provided value isn't too high. You don't want to burn a large fee to send a small transaction.
    - If you are using `rippled`, you can also use the `fee_mult_max` and `fee_div_max` parameters of the sign method to set a limit to the load scaling you are willing to sign.
    - Some client libraries (like xrpl.js ↗ and xrpl-py ↗) have configurable maximum `Fee` values, and raise an error instead of signing a transaction whose `Fee` value is higher than the maximum.
- You cannot auto-fill from an offline machine nor when multi-signing.

Transaction Costs and Failed Transactions

Since the purpose of the transaction cost is to protect the XRP Ledger peer-to-peer network from excessive load, it should apply to any transaction that gets distributed to the network, regardless of whether or not that transaction succeeds. However, to affect the shared global ledger, a transaction must be included in a validated ledger. Thus, `rippled` servers try to include failed transactions in ledgers, with `tec` status codes ("tec" stands for "Transaction Engine - Claimed fee only").

The transaction cost is only debited from the sender's XRP balance when the transaction actually becomes included in a validated ledger. This is true whether the transaction is considered successful or fails with a `tec` code.

If a transaction's failure is final, the `rippled` server does not relay it to the network. The transaction does not get included in a validated ledger, so it cannot have any effect on anyone's XRP balance.

Insufficient XRP

When a `rippled` server initially evaluates a transaction, it rejects the transaction with the error code `terINSUF_FEE_B` if the sending account does not have a high enough XRP balance to pay the XRP transaction cost.

Since this is a `ter` (Retry) code, the `rippled` server retries the transaction without relaying it to the network, until the transaction's outcome is final.

When a transaction has already been distributed to the network, but the account does not have enough XRP to pay the transaction cost, the result code `tecINSUFF_FEE` occurs instead. In this case, the account pays all the XRP it can, ending with 0 XRP. This can occur because `rippled` decides whether to relay the transaction to the network based on its in-progress ledger, but transactions may be dropped or reordered when building the consensus ledger.

Key Reset Transaction

As a special case, an account can send a SetRegularKey transaction with a transaction cost of `0`, as long as the account's `lsfPasswordSpent` flag is disabled. This transaction must be signed by the account's *master key pair*. Sending this transaction enables the `lsfPasswordSpent` flag.

This feature is designed to allow you to recover an account if the regular key is compromised, without worrying about whether the compromised account has any XRP available. This way, you can regain control of the account before you send more XRP to it.

The `lsfPasswordSpent` flag starts out disabled. It gets enabled when you send a SetRegularKey transaction signed by the master key pair. It gets disabled again when the account receives a Payment of XRP.

When the FeeEscalation amendment is enabled, `rippled` prioritizes key reset transactions above other transactions even though the nominal transaction cost of a key reset transaction is zero.

Changing the Transaction Cost

The XRP Ledger has a mechanism for changing the minimum transaction cost to account for long-term changes in the value of XRP. Any changes have to be approved by the consensus process. See Fee Voting for more information.

See Also

- **Concepts:**
  - Reserves
  - Fee Voting
  - Transaction Queue
- **Tutorials:**
  - Reliable Transaction Submission
- **References:**
  - fee method
  - server_info method

- FeeSettings object
- SetFee pseudo-transaction