FastPay
High-Performance Byzantine Fault Tolerant Settlement
RTGS
A simplified view
RTGS
A simplified view

Very centralized
Low capacity (expensive)
RTGS
A simplified view

**TPS:** 500 tx/s

**Latency:** minutes
RTGS
A simplified view

TPS: 80,000 tx/s  TPS: 500 tx/s
Latency: seconds  Latency: minutes
RTGS
A simplified view

Promises of payment  Settlement
RTGS
A simplified view

Fast settlement
RTGS
A simplified view

BFT resilience
High capacity (cheap)
Byzantine Fault Tolerance

> $\frac{2}{3}$
In summary

**What we want**
- Low latency
- BFT reliance
- Fast finality
- High capacity

**Current industry**
- Low latency (not settled)
- Centralized
- Slow finality
- High capacity (not settled)
Make it practical for retail payment at physical points of sale

This requires extremely low latency
Difference with blockchains

**Blockchains**

Byzantine Consensus

**FastPay**

Byzantine Consistent Broadcast
FastPay
How does it work?

sender

recipient
FastPay
How does it work?

1. transfer order

sender

recipient
FastPay
How does it work?

1. transfer order

sender

2. verify

recipient
FastPay
How does it work?

1. transfer order
2. verify
3. signed transfer order

sender

recipient
FastPay
How does it work?

1. transfer order
2. verify
3. signed transfer order
4. confirmation order
5. confirmation order
6. confirmation order

sender
recipient
FastPay
How does it work?

1. transfer order

2. verify

3. signed transfer order

4. confirmation order

5. confirmation order

6. confirmation order

7. update

sender

recipient
FastPay
Increasing capacity
Byzantine Consistent Broadcast

Validity
No duplication
Integrity
Consistency
FastPay
Authorities' state

 Authorities

• Authority name and keys
• Committee information
• Accounts information
• Last primary tx index

 Each account

• Verification key
• Balance
• Sequence number
• Last transfer order
• List of certificates and synchronization orders
FastPay
Clien's state

• Their account's address
• Their secret key
• Committee information
• Last sequence number
• Last signed transfer order
FastPay
Interface it with a primary infrastructure

Smart Contract's state

- The committee information
- Total funds in the contract
- Last primary tx index
- "Redeem log"
FastPay
Authorities' state

 Authorities

- Authority name and keys
- The committee information
- Committee information
- Accounts information
- Last primary tx index

 Each account

- Verification key
- Balance
- Sequence number
- Last transfer order
- List of certificates and synchronization orders
FastPay
From primary infrastructure to FastPay

1. funding transaction
FastPay
From primary infrastructure to FastPay

1. funding transaction

sender

2. synchronization order

smart contract
FastPay
From primary infrastructure to FastPay

1. funding transaction
2. synchronization order
3. verify & update
FastPay
Interface it with a primary infrastructure

Smart Contract's state

- The committee information
- Total funds in the contract
- Last primary tx index
- "Redeem log"
From the primary infrastructure to FastPay

1. transfer order
FastPay
From the primary infrastructure to FastPay

1. transfer order

sender

2. verify

smart contract
FastPay
From the primary infrastructure to FastPay

1. transfer order
2. verify
3. signed transfer order
FastPay
From the primary infrastructure to FastPay

1. transfer order
2. verify
3. signed transfer order
4. confirmation order

sender

smart contract
FastPay
From the primary infrastructure to FastPay

1. transfer order
2. verify
3. signed transfer order
4. confirmation order
5. update

sender

smart contract
FastPay
From the primary infrastructure to FastPay

1. transfer order
2. verify
3. signed transfer order
4. confirmation order
5. update
6. redeem transaction
7. verify & update
FastPay
Implementation

- Written in Rust
- Networking: Tokio & UDP
- Cryptography: ed25519-dalek

https://github.com/novifinancial/fastpay
FastPay
Throughput Evaluation
Figure 5: Variation of the throughput of transfer orders with the number of shards, for various levels of concurrency (in-flight parameter). The measurements are run under a total load of 1M transactions.

Figures 11 and 12 (see Appendix C) show the variation of the throughput of transfer and confirmation orders with the number of shards, for various total system loads—namely the total number of transactions in the test; they show that the throughput is not affected by the system load. The tests were performed with 4 authorities, and the client concurrency in-flight parameter set to 1,000. These figures illustrate that FastPay can process about 160,000 transactions per second even under a total load of 1.5M transactions, and that the total load does not significantly affect performance. These supplementary figures 5 and 6 that illustrate the concurrent transaction rate (in-flight parameter) also does not influence performance significantly (except when it is too low by under-utilizing the system).

Readers may be surprised those measurements are key. The key measurement work by Han et al. [23] compares a number of permissioned systems under a high load, and shows that for all of Hyperledger Fabric (v0.6 with PBFT) [26], Hyperledger Fabric (v1.0 with BFT-Smart) [27], Ripple [15] and R3 Corda v3.2 [39] the successful requests per second drops to zero as the transaction rate increases to more than a few thousands transactions per second (notably for Corda only a few hundred). An important exception is Tendermint [10], that maintains a processed transaction rate of about 4,000 to 6,000 transactions per second at a high concurrency rate. Those findings were confirmed for Hyperledger Fabric that reportedly starts saturating at a rate of 10,000 transactions per second [34]. Our results demonstrate that FastPay continues to be very performant even under the influence of extremely high rates of concurrent transactions (in-flight parameter) and overall workload (total number of transactions processed), as expected. This is apparently not the norm.

Influence of the number of authorities

As discussed in Section 4, we expect that increasing the number of authorities only impacts the throughput of confirmation orders (that need to transfer and check transfer certificates signed by 2f+1 authorities), and not the throughput of transfer orders. Figure 7 confirms that the the throughput of confirmation orders decreases as the number of authorities increases. FastPay can still process about 80,000 transactions per second with 20 authorities (for 75 shards). The measurements are taken with an in-flight concurrency parameter set to 1,000, and under a load of 1M total transactions. We note that for higher number of authorities, using an aggregate signature scheme (e.g. BLS [8]) would be preferable since it would result in constant time verification and near-constant size certificates. However, due to the use of batch verification of signatures, the break even point may be after 100 authorities in terms of verification time.

7.3 Latency

We measure the variation of the client-perceived latency with the number of authorities. We deploy several FastPay multi-shard authorities on Amazon Web Services (all in Stockholm, eu-north-1 zone), each on a m5d.8xlarge instance. This class of instance guarantees 10Gbit network capacity, on a 3.1 GHz, Intel Xeon Platinum 8175 with 32 cores, and 128 GB memory. The operating system is Linux Ubuntu server 16.04. Each instance is configured to run 15 shards. The client is run on an Apple laptop (MacBook Pro) with a 2.9 GHz Intel Core i9 (6 physical and 12 logical cores), and 32 GB 2400 MHz DDR4.
FastPay
High concurrency

Figure 5: Variation of the throughput of transfer orders with the number of shards, for various levels of concurrency (in-flight parameter). The measurements are run under a total load of 1M transactions.

Robustness and performance under total system load (Figures 11 and 12, see Appendix C) show the variation of the throughput of transfer and confirmation orders with the number of shards, for various total system loads—namely the total number of transactions in the test; they show that the throughput is not affected by the system load. The tests were performed with 4 authorities, and the client concurrency in-flight parameter set to 1,000. These figures illustrate that FastPay can process about 160,000 transactions per second even under a total load of 1.5M transactions, and that the total load does not significantly affect performance. These supplementary figures (5 and 6) that illustrate the concurrent transaction rate (in-flight parameter) also does not influence performance significantly (except when it is too low by under-utilizing the system).

Readers may be surprised those measurements are key. The key measurement work by Han et al. [23] compares a number of permissioned systems under a high load, and shows that for all of Hyperledger Fabric (v0.6 with PBFT) [26], Hyperledger Fabric (v1.0 with BFT-Smart) [27], Ripple [15] and R3 Corda v3.2 [39] the successful requests per second drops to zero as the transaction rate increases to more than a few thousands transactions per second (notably for Corda only a few hundred). An important exception is Tendermint [10], that maintains a processed transaction rate of about 4,000 to 6,000 transactions per second at a high concurrency rate. Those findings were confirmed for Hyperledger Fabric that reportedly starts saturating at a rate of 10,000 transactions per second [34]. Our results demonstrate that FastPay continues to be very performant even under the influence of extremely high rates of concurrent transactions (in-flight parameter) and overall work load (total number of transactions processed), as expected. This is apparently not the norm.

Influence of the number of authorities. As discussed in Section 4, we expect that increasing the number of authorities only impacts the throughput of confirmation orders (that need to transfer and check transfer certificates signed by 2f+1 authorities), and not the throughput of transfer orders. Figure 7 confirms that the throughput of confirmation orders decreases as the number of authorities increases. FastPay can still process about 80,000 transactions per second with 20 authorities (for 75 shards). The measurements are taken with an in-flight concurrency parameter set to 1,000, and under a load of 1M total transactions. We note that for higher number of authorities, using an aggregate signature scheme (e.g. BLS [8]) would be preferable since it would result in constant time verification and near-constant size certificates. However, due to the use of batch verification of signatures, the break even point may be after 100 authorities in terms of verification time.

7.3 Latency
We measure the variation of the client-perceived latency with the number of authorities. We deploy several FastPay multi-shard authorities on Amazon Web Services (all in Stockholm, eu-north-1 zone), each on a m5d.8xlarge instance. This class of instance guarantees 10Gbit network capacity, on a 3.1 GHz, Intel Xeon Platinum 8175 with 32 cores, and 128 GB memory. The operating system is Linux Ubuntu server 16.04. Each instance is configured to run 15 shards. The client is run on an Apple laptop (MacBook Pro) with a 2.9 GHz Intel Core i9 (6 physical and 12 logical cores), and 32 GB 2400 MHz DDR4.
Figure 10: Authority algorithms for cross-shard updates and (Primary) synchronization orders.

Figure 11: Variation of the throughput of transfer orders with the number of shards, for various loads. The in-flight parameter is set to 1,000.

Figure 12: Variation of the throughput of confirmation orders with the number of shards, for various loads. The certificates are issued by 4 authorities, and the in-flight parameter is set to 1,000.

FastPay
Robustness
FastPay
Robustness

Figure 10: Authority algorithms for cross-shard updates and (Primary) synchronization orders.

Figure 11: Variation of the throughput of transfer orders with the number of shards, for various loads. The in-flight parameter is set to 1,000.

Figure 12: Variation of the throughput of confirmation orders with the number of shards, for various loads. The certificates are issued by 4 authorities, and the in-flight parameter is set to 1,000.
Figure 7: Variation of the throughput of confirmation orders with the number of authorities, for various number of shards. The in-flight parameter is set to 1,000 and the system load is of 1M transactions.

RAM; and connected to a reliable WIFI network. We run experiments with the client in two different locations; (i) in the U.K. (geographically close to the authorities, same continent), (ii) in the U.S. West Coast (geographically far from the authorities, different continent). Each measurement is the average of 300 runs, and the error bars represent one standard deviation; all experiments use our UDP implementation.

We observe that the client-authority WAN latency is low for both transfer and confirmation orders; the latency is under 200ms when the client is in the U.S. West Coast, and about 50ms when the client is in the U.K. Figure 8 illustrates the latency between a client creating and sending a transfer order to all authorities, and receiving sufficient signatures to form a transfer certificate (in our experiment we wait for all authorities to reply to measure the worse case where f authorities are Byzantine). The latency is virtually constant as we increase the number of authorities, due to the client emitting orders asynchronously to all authorities and waiting for responses in parallel.

Figure 9 illustrates the latency to submit a confirmation order, and wait for all authorities to respond with a success message. It shows latency is virtually constant when increasing the number of authorities. This indicates that the latency is largely dominated by the network (and not by the verification of certificates). However, since even for 10 authorities a FastPay message fits within a network MTU, the variation is very small. Due to our choice of using UDP as a transport there is no connection initiation delay (as for TCP), but we may observe packet loss under very high congestion conditions. Authority commands are idempotent to allow clients to re-transmit to overcome loss without sacrificing safety.

Performance under failures. Research literature suggests permissioned blockchains based on (often leader-based) consensus suffer an enormous performance drop when some authorities fail [30]. We measure the effect of authority failure in FastPay and show that latency is not affected when f or fewer authorities are unavailable.

<table>
<thead>
<tr>
<th>Latency (ms ± std)</th>
<th>0</th>
<th>43 ± 2</th>
<th>1</th>
<th>41 ± 3</th>
<th>2</th>
<th>44 ± 4</th>
<th>3</th>
<th>47 ± 2</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
<td>43</td>
<td>±2</td>
<td>1</td>
<td>±3</td>
<td>2</td>
<td>±4</td>
</tr>
</tbody>
</table>

Table 2: Crash-failure Latency.

We run our baseline experimental setup (10 authorities distributed over 10 different AWS instances), when a different number of authorities are not available for f = 0...3. We measure the latency experienced by a client on the same continent (Europe), sending a transfer order until it forms a valid transfer certificate. Table 2 summarizes the mean latency and standard deviation for different f. There is no statistically significant difference in latency, no matter how many tolerable failures FastPay experiences (up to f ≈ 3 for 10 authorities). We also experimented with killing FastPay.
We observe that the client-authority WAN latency is low for both transfer and confirmation orders; the latency is under 200ms when the client is in the U.S. West Coast, and about 50ms when the client is in the U.K. Figure 8 illustrates the latency between a client creating and sending a transfer order to all authorities, and receiving sufficient signatures to form a transfer certificate (in our experiment we wait for all authorities to reply to measure the worse case where \( f \) authorities are Byzantine). The latency is virtually constant as we increase the number of authorities, due to the client emitting orders asynchronously to all authorities and waiting for responses in parallel.

Figure 9 illustrates the latency to submit a confirmation order, and wait for all authorities to respond with a success message. It shows latency is virtually constant when increasing the number of authorities. This indicates that the latency is largely dominated by the network (and not by the verification of certificates). However, since even for 10 authorities a FastPay message fits within a network MTU, the variation is very small. Due to our choice of using UDP as a transport there is no connection initiation delay (as for TCP), but we may observe packet loss under very high congestion conditions. Authority commands are idempotent to allow clients to re-transmit to overcome loss without sacrificing safety.

Performance under failures. Research literature suggests permissioned blockchains based on (often leader-based) consensus suffer an enormous performance drop when some authorities fail [30]. We measure the effect of authority failure in FastPay and show that latency is not affected when \( f \) or fewer authorities are unavailable.
Figure 7: Variation of the throughput of confirmation orders with the number of authorities, for various number of shards. The in-flight parameter is set to 1,000 and the system load is of 1M transactions.

We observe that the client-authority WAN latency is low for both transfer and confirmation orders; the latency is under 200ms when the client is in the U.S. West Coast, and about 50ms when the client is in the U.K. Figure 8 illustrates the latency between a client creating and sending a transfer order to all authorities, and receiving sufficient signatures to form a transfer certificate (in our experiment we wait for all authorities to reply to measure the worse case where $f$ authorities are Byzantine). The latency is virtually constant as we increase the number of authorities, due to the client emitting orders asynchronously to all authorities and waiting for responses in parallel.

Figure 9 illustrates the latency to submit a confirmation order, and wait for all authorities to respond with a success message. It shows latency is virtually constant when increasing the number of authorities. This indicates that the latency is largely dominated by the network (and not by the verification of certificates). However, since even for 10 authorities a FastPay message fits within a network MTU, the variation is very small. Due to our choice of using UDP as a transport there is no connection initiation delay (as for TCP), but we may observe packet loss under very high congestion conditions. Authority commands are idempotent to allow clients to re-transmit to overcome loss without sacrificing safety.

Performance under failures. Research literature suggests permissioned blockchains based on (often leader-based) consensus suffer an enormous performance drop when some authorities fail [30]. We measure the effect of authority failure in FastPay and show that latency is not affected when $f$ or fewer authorities are unavailable.

<table>
<thead>
<tr>
<th>$f$</th>
<th>Latency (ms) ± std</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>43 ± 2</td>
</tr>
<tr>
<td>1</td>
<td>41 ± 3</td>
</tr>
<tr>
<td>2</td>
<td>44 ± 4</td>
</tr>
<tr>
<td>3</td>
<td>47 ± 2</td>
</tr>
</tbody>
</table>

Table 2: Crash-failure Latency.

We run our baseline experimental setup (10 authorities distributed over 10 different AWS instances), when a different number of authorities are not available for $f = 0, ..., 3$. We measure the latency experienced by a client on the same continent (Europe), sending a transfer order until it forms a valid transfer certificate. Table 2 summarizes the mean latency and standard deviation for different $f$. There is no statistically significant difference in latency, no matter how many tolerable failures FastPay experiences (up to $f \leq 3$ for 10 authorities). We also experimented with killing FastPay.
Worst-case efficiency

**Blockchains**

Bad leader can slow down the protocol

**FastPay**

No leader, nothing changes
FastPay
The cost of simplicity

- Less than 4,000 LOC
- Over 1,500 Git commits
- Took 2.5 months to 3 engineers
FastPay
Deployment costs

• AWS m5d.8xlarge instance
• ~ 5 USD / hour
FastPay
Further works

- Checkpointing?
- Change the authorities?
- Privacy?
Conclusion

FastPay

• Based on Byzantine Consistent Broadcast
• Simple design, low latency, high capacity, very robust

• **Code**: https://github.com/novifinancial/fastpay
Protocol Details
From FastPay to FastPay
FastPay
Protocol details

1. Transfer order

- Sender address
- Recipient address
- Amount
- Sequence number
- Sender's signature
FastPay
Protocol details

2. verify

- The sender's signature
- No previous tx is pending
- The amount is positive
- Sequence number is as expected
- Balance is sufficient
3. signed transfer order

- Each authority signed the transfer order received in step 1.
FastPay
Protocol details

4. 5. 6. confirmation order

- Collect enough signed transfer orders from step 2.
FastPay
Protocol details

7. update

- Check there are enough signatures
- Decrease the senders' balance
- Increase the sequence number
- Set the pending order to None
- Increase the recipient's balance
Protocol Details

From FastPay to primary infrastructure
FastPay
From FastPay to primary infrastructure

1. transfer order
   • Sender address
   • Recipient address
   • Amount
   • Sequence number
   • Sender's signature
FastPay
From FastPay to primary infrastructure

2. verify

- The sender's signature
- No previous tx is pending
- The amount is positive
- Sequence number is as expected
- Balance is sufficient
3. signed transfer order

- Each authority signed the transfer order received in step 1.
4. 6. confirmation order

- Collect enough signed transfer orders from step 2.
FastPay
From FastPay to primary infrastructure

5. update

- Check there are enough signatures
- Decrease the senders' balance
- Increase the sequence number
- Set the pending order to None
From FastPay to primary infrastructure

7. verify & update
- Check sequence number is not on the redeem log
- Update the redeem log
- Transfer the amount to recipient
Protocol Details
From primary infrastructure to FastPay
FastPay
From primary infrastructure to FastPay

1. funding transaction
   - FastPay recipient
   - All fields required by the primary infrastructure (and the amount)
FastPay
From primary infrastructure to FastPay

2. synchronization order

- Read the transaction on the primary infrastructure (once it is sequenced)
FastPay
From primary infrastructure to FastPay

3. update & verify
- Check last primary tx index
- Increment last primary tx index
- Create a FastPay account for the recipient (if needed)
- Increase recipient's balance