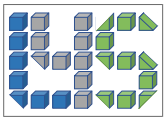


All the current, major, publicized, DLT/Blockchain implementations can demonstrate the ability to maintain a distributed ledger securely. They can record balances of value and their secure transfer between network participants. This is both useful and a hindrance. It is useful because it helps with understanding and experimenting with this new technology. It is a hindrance because it creates the illusion that these solutions work when they are not commercially viable. Their lack of viability is due to the unsolved problems of being slow and having limited capacity throughput. This is because the fundamental designs are flawed.

If a lawn mower engine was installed in a Ferrari's chassis, it could still drive and move around. From its outer appearance and the theoretical dynamics of the chassis, it would have the potential of a supercar but it would not be one. When considering this analogy to the current DLT/Blockchain implementations it is important to critically analyze the engines that support them from a performance standpoint.

Why are they not producing the low latency and high capacity that is required by the financial services and other industries? Is there an answer or solution to this problem?
The following is a fundamental production engineering review of the current DLT processes.

If:

P = the available time for production and

T_L = the Total Production Cycle Time (or latency) to produce a product and

T_C = a process cycle time

C, which is the Output Capacity throughput of the process within P, then

$$C = \frac{P}{T_L}$$

For a single process product then

$T_L = T_C$.

For a multiple process product, say one that involves six, sequential and independent steps and a final assembly, the Process Cycle Times for each process (T_C) could be represented by T_1, T_2, T_3 T_4, T_5, T_6 and a final assembly T_F, therefore:

$$C = \frac{P}{T_L} = \frac{\mathrm{P}}{(T_1+T_2+T_3+T_4+T_5+T_6+T_F)}$$

If the multiple processes are completed by one resource then, T_L, the total Production Cycle Time, defines the time to make a product, which in this example is the process latency as well. However, the individual process cycles time for each T_C are obviously much shorter than the Production Time (T_L). One of the ways to increase the Capacity is to increase the number of resources.

There are two ways this can be achieved.

a) A resource can be assigned to each process or
b) more resources can perform all the processes.

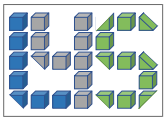
Option (a)

If a resource was assigned to each process, then the total resources would be seven and the capacity is defined by the slowest (bottleneck) process cycle time Max(T_C).

$$C = \frac{P}{Max(T_c)}$$

So, while the Product Cycle time is now Max(T_C) the product's Total Production Time or latency is still the same.

$T_L = (T_1 + T_2 + T_3 + T_4 + T_5 + T_6 + T_F)$



Note each process could be individually improved but there would always be one that would be the slowest i.e. $Max(T_c)$. The other processors can't produce at a better rate than the bottleneck process without creating backlogs before the bottleneck process or have delays waiting for the hand-off after the bottleneck.

Option (b)

If we created seven versions of multiple step process it would have the same latency with any differences being attributed to the performance variation of each of the processors. It would be the equivalent of:

$(T_1 + T_2 + T_3 + T_4 + T_5 + T_6 + T_F) \times 7.$

The capacity would be slightly improved as the Capacity would now be driven by the average process cycle time rather than the bottleneck, however unless there is a large difference between $Max(T_c)$ and $Avg(T_c)$, this difference will be negligible.

$$C = \frac{P}{Avg(T_c)}$$

There may not be much difference in overall process but option (a) is usually higher quality with process expertise but less flexible. Option (a) is also more effective when the process cycles are very similar. Option (b) can have quality consistency issues between the different processors but can give more flexibility for process adaptation.

The best way to improve latency, if it is logistically possible, is to run as many of the processes in parallel. In this case, the process would consist of:

$$T_L = \text{Max} \begin{pmatrix} T_1 \\ T_2 \\ T_3 \\ T_4 \\ T_5 \\ T_6 \end{pmatrix} + T_F$$

Now the Product Cycle Time and the Total Production Time or latency are both

$= Max(T_C) + T_F$

For further efficiencies, the process can use independently created pre-assembled parts for the process. The only process left is final assembly. Therefore:

$T_L = T_F$

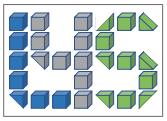
and

$$C = \frac{P}{T_F}$$

The above equation does not take into account the cost or time for delivery of the pre-assembled parts.

A variation on both the pre-assembly and parallel processing ideas is the model for high frequency trading and algorithmic execution. The pre-assembled processes are the algorithms combined with the portfolio and market data. The only processes left are order routing and risk management, which are run in parallel for efficiency.

The high frequency trading and algorithmic execution processes are shown below:



Pre-Process	Process	Post Process
Algorithms	Routed Orders	
⏩	⏩	Order Status*
Portfolio & Mkt Data	Risk Mgmt	

*Filled or Not Filled

This is an optimally designed process for high capacity and low latency.

Production Analysis of Bitcoin

If the Bitcoin Transactions are assumed to be 0.5kb in size on average.

Then, as the Bitcoin blocks are targeted at 1Mb in size, there are usually in the range of 2000 transactions per block.

The Bitcoin network controls the time to produce a new block to approximately 10 minutes.

Therefore, the maximum capacity per second for Bitcoin is

$$= \frac{2000\ Transactions}{(\ 10\ minutes\ x\ 60\ seconds\ \ )}$$

$$= \frac{2000\ Tx's}{(\ 600\ s\ \ )}$$

= 3.33 Transactions per second.

The above means that as well as taking 10 minutes to produce any new blocks – i.e. the minimum latency to validate a transaction, if there are more than 3.33 transactions per second being created in the network there will be an increasing backlog of transactions to put into blocks which will never clear unless the rate of creation of new transactions falls below the maximum capacity of the network for a sustained period. Note as the transactions are agreed away from the network, and are created through an Active-Passive role between the Payee and the Payor, the transaction process does not individually affect the speed of the ledger validation.

The only way to increase the capacity and reduce the latency of the Bitcoin Network is to either shorten the time to create a block or to increase the block size. The problem is that either of these actions will make forks in the blockchain more likely, so while transactions may put into blocks quicker, the resolution of a dominant block may take longer.

There is no viable way for the Bitcoin model to realistically handle the high payments volumes of a credit card processor or the total volume and high capacities required to support major exchanges.

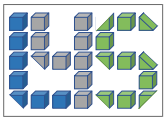
Production Analysis of Consensus Driven DLT Solutions

Although the major implementations of DLT solutions vary in their ledger recording protocols and mechanisms, they all follow a similar process due to their all using some form of consensus algorithm to reach agreement on the ledger.

Transactions between two parties need an instigator and an associate. Through a Smart Contract, one party creates the transaction and the second party agrees to the transaction. The transaction is then validated by other nodes or independent Oracles or Auditors. The transactions are then assembled into candidate blocks to be added to the one-dimensional blockchain.

Seven generic steps which are similar across the solutions are: Instigation, Affirmation, Validation, Block Proposal, Consensus Voting, Consensus Block Sharing, Ledger Validation and Update.

Note for each process there are different parties:



Process	Party	Process Time
Instigation:	Party 1	T_i
Affirmation:	Party 2	T_a
Validation	Oracle/Auditor	T_{va}
Block Proposal	Notary	T_p
Consensus Voting	All Nodes	T_{vo}
Block Sharing	All Nodes	T_{bl}
Ledger Update	Individual Nodes	T_u

Effectively the time to produce a validated transaction would be:

$$T_L = (T_i + T_a + T_{va} + T_p + T_{vo} + T_{bl} + T_u)$$

These are all sequential, dependent steps across multiple different parties and therefore cannot be run in parallel for an optimal solution. The only way is to increase the capacity throughput is to reduce the time for each process which all have finite limits. It worth noting that for any process to generate a 1000 results per second, the process time must be 0.001 seconds or $1/1000^{th}$ second.

Also, this simple model does not include any latency between transmission and receipt at each step across the network between nodes, which adds to the time involved, where dependency is present.

Similar to Bitcoin, the capacity of the block proposal and voting process must be greater than the capacity at which the transactions are generated. Otherwise there is a risk of a backlog similar to the scenario that was described in the Bitcoin analysis. It is also important to realize that as the participants and number of transactions increase the number of transactions to be validated and votes to be cast and tallied increase by a factor of both numbers. Therefore, the latency of the processes becomes greater with increased participation and the maximum network process capacity remains unchanged, so the risk of a backlog being created increases.

Realistically, if a network is generating 1,000's of transactions per second they would be generated by a large number of participants of which at least 67% would have to vote on all the transactions to gain consensus at the same rate of transactions per second. While it is easy to generate a lot of transactions, it is impossible for a group to vote and create consensus at an equivalent rate.

The current, but ultimately, doomed solutions are a hybrid of multiple processors and parallel processing. Transactions can be generated and validated by smaller populations of so-called Lightning Networks. They can then be constructed into a separate distributed ledger or side-chain by other smaller populations or "Shards" for quicker consensus. The Shards then have to be added to the larger main ledger at a later time.

If N_L represents the number of Lightning Networks operating in the network and N_S represents the number of Shards for consensus, then there are two parallel processes running in conjunction with each other.

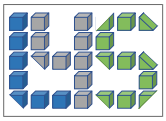
The processing time for each process would be:

$$T_L = (T_i + T_a + T_{va}) \text{ for the Lightning networks}$$

and

$$T_L = (T_p + T_{vo} + T_{bl} + T_u) \text{ for the Consensus Shards.}$$

While this has shortened the individual process time, they are still relatively large and involve multiple validations for the Lightning Networks and multiple voting and tallying for the Sharded consensus.



Within these smaller population groups, the capacity is increased and the number of transactions processed are reduced both by a factor of N_L for the Lightning Networks and factor of N_S for the shards. However, unless process times are reduced overall to $1/1000^{th}$ second, these smaller populations cannot reach a capacity of 1000's of transactions per second. Remembering that, whatever capacity is reached, it only applies to the members within the Lightning Networks or Shards. Once a party in one group transacts with a party in another group, the process slows down to the capacity at which the greater network can validate all the Lightning Network and Shard outputs to the larger network blockchain ledger.

This creates competing variables. One reduces the number of counterparties to increase speed, which, by definition, increases the probability of the transacting party's counterparty being outside smaller group and therefore subject to a slower process.

Mathematically, Network Transaction cycle time decreases as an inverse to the number of lightning networks and consensus shards:

$$T_L = f\{\frac{1}{(N_L)}, \frac{1}{(N_S)}\}$$

Whereas the probability of transactions ($p_{o/n}$) outside the lightning networks or consensus shards increases

$$p_{o/n} = f\{N_L, N_S\}$$

Regardless of these short cuts, assuming each network participant creates the same number of transactions. The total number of transactions (Tr_{TOTAL}) created in a network requiring validation, voting and consensus tallying is the product of the number of participants (P_N) and the transactions they produce (Tr_P).

Or:

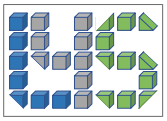
$$Tr_{TOTAL} = P_N \times Tr_P$$

While this demonstrates, on an average basis, the linear relationship between the participants and number of transactions broadcast, with the transaction validation, voting, vote tallying, block creation, block broadcast and ledger validation, the data transfer, computational demands and data storage requirements within the network expand exponentially with the adoption by more participants, so transaction processing and ledger validation must be as optimal as possible otherwise the network's limits can be quickly reached. A seven step, sequential, dependent process involving multiple parties is not optimal.

The above proves why current DLT implementations do work but will always be slow, unable to handle high volumes and still have scalability issues with their continuously expanding active memory requirements. The analysis also does not include any reconciliation processes and controls to ensure all data from the Lightning Networks and Consensus Shards are transferred the main distributed ledger accurately. These deferred control processes would further reduce the productivity of the network, while also creating vulnerabilities to attack.

An Optimal Design Proposal

Similar to the high frequency trading or algorithmic execution, an optimal model for a distributed ledger has to have pre-assembled transactions, which can create self-validating entries to the shared ledger. Then, the only capacity constraint is the broadcast and receipt of the transactions across the network, which is only limited by the capacity of the network routing hardware rather than the ledger process. If the transactions are self-



validating, the any other node can receive the transactions, and update the ledger without having to refer to any of the other nodes. See the representation below.

Pre-Process		**Process**		**Post Process**
Self-Validating Tx	⏩	Tx Broadcast Tx Receipt	⏩	Tx/Ledger Validation & Ledger Update

Conclusion

The current DLT/Blockchain implementations do resemble a Ferrari with a lawn mower engine. No amount of tuning or adaptation is going to make that vehicle a supercar. DLT/Blockchain solutions operate on three fundamental layers. The Core layer, the Protocol layer and the Application (or "App") layer. Most have locked down their Core and Protocol layers and are now focusing predominantly on functionality at the App layer, even though the Core and Protocol layers can't produce the performance requirements required.

Similar to the Ferrari-Lawn Mower Engine analogy where the solution is to replace the engine of the vehicle, the current DLT/Blockchain implementations need to fundamentally redesign their Core and Protocol layers and create a solution that doesn't rely on consensus algorithms but, unlike Bitcoin's non-consensus algorithm solution, it must have the capacity throughput and low latency capabilities to meet the financial services' and other industries' needs. L4S has designed a very different distributed ledger core and protocol layer to make the above optimal model and its performance potential possible. If you are interested, then please contact me.