timestamping.ixi

Samuel Rufinatscha

March 2019

1 Abstract

A directed, acyclic graph (DAG) is a graph with only a partial order structure. It's possible for two vertices u,v to be incomparable, in other words one is not a descendent of the other one in the graph. This makes it difficult to establish the correct order of its vertices. In the Tangle, to deal with this effect, we enforce every transaction to declare a timestamp: the time when it was attached. However, even though all the transactions contain timestamps, we can not be sure if they are correct (e.g. nodes could lie about timestamps; or declare wrong timestamps because of wrong clocks, ...). Either way, we expect most transactions to declare their timestamp as accurate and precise as they can. By inspecting other transactions relative to a transaction, we expect to be able to determine the time interval within the transaction was attached. Serguei Popov described two algorithms [1] to find this confidence interval.

2 Algorithms

A brief summary about the described algorithms is given below. For a better understanding, following terms should be clarified:

- P(v) — the past of v; all transactions which are referenced directly or indirectly by v
- F(v) — the future of v; all transactions which reference directly or indirectly v
- Ind(v) — all transactions which are neither in P(v) nor F(v); by definition, v is part of Ind(v)

Besides that, every transaction contains three timestamps T-, T and T+; where T stands for the actual time when a transaction was attached. Note that T- (lowerbound) and T+ (upperbound) are necessery since a nodes' clock could be slightly imprecise.

2.1 Quantile procedure

The first described procedure makes use of a p-quantile calculation and two multi-sets. The special characteristic of a multi-set compared to an usual set is that the elements of a multi-set can occur several times. Following two multi-sets are declared:

D-	all lowerbound timestamps of Ind(v)
D+	all upperbound timestamps of Ind(v)

Besides that, p will be fixed between [0, 0.5]. It is currently not known which p fits best, but a value of 0.2 or 0.3 can be considered as acceptable. The calculation of the confidence interval [av, bv] would be as follows:

av = p-quantile(D-)
bv = (1-p)-quantile(D+)

2.2 Random walk procedure

The second procedure makes use of the random walk with an entry point very deep in the tangle. For this, a very large alpha value should be considered (perhaps infinity). Moreover, G represents the path of the random walk. The calculation of the confidence interval [av, bv] would be as follows:

av = the largest lowerbound timestamp which occurs in P(v) AND G
bv = the lowest upperbound timestamp which occurs in F(v) AND G

3 Implementation

Boths algorithms described in the mentionend paper by Serguei have been implemented [3]. It seems like that the random walk procedure is more reliable.

3.1 Tangle.java

As the name suggests, this class represents the Tangle. The internal transaction mappings are remarkable as they allow fast calculation of a transactions' future set. This can be achieved by mapping all the transactions with all its direct approvers. Since tip transactions don't have any mappings, it's trivial to find tips.

3.2 Interval.java

This class describes an interval which specifies an upper and lower bound. It will be used by getTimestampInterval() to return the calculated confidence interval.

3.3 TimestampingCalculation.java

This class represents an abstract timestamping calculation. It provides an link to the transaction that is to be inspected. Besides that it provides methods to add additional transaction to help find a more accurate confidence interval. Once the interval is found by the timestamping algorithm, it can be accessed by **getResult()**.

3.4 RandomWalkTimestampingCalculation.java

Implements the abstract timestamping calculation and extends it by the additional parameter **entryPoint**, which is required for the random walk.

3.5 QuantileTimestampingCalculation.java

Implements the abstract timestamping calculation and extends it by the additional parameter **beta**, which is required for the quantile calculation.

3.6 AbstractTimestampingProcedure.java

The architecture of timestamping.ixi was designed to support the implementation of different algorithms. AbstractTimestampingProcedure.java serves as template for every timestamping-algorithm. Based on the timestamping.ixi specification written by Paul Handy [2], following methods have been implemented:

Starts the timestamping calculation for a transaction and returns an identifier. Since certain algorithms require additional parameters (e.g. p-value for the quantile-procedure), the parameter 'Object... args' was introduced.
abstract String beginTimestampCalculation(String txToInspect, Object... args)

Adds additional transactions from other ontologies to the timestamping calculation to help find a more accurate confidence interval.
void addTimestampHelper(String identifier, String... referringTx)

Computes the actual confidence interval for a transaction
abstract Interval getTimestampInterval(String identifier, Tangle tangle)

Returns the confidence level of the calculation
abstract double getTimestampConfidence(String identifier)

3.6.1 Additions

To simplify the interaction with the Tangle, following methods have been included:

Returns all transactions which are referenced directly or indirectly by a transaction. Makes use of a recursive breadth-first search (BFS) by traversing trunk and branch of each transaction in the path.
static Set<String> getPast(String transactionHash, Tangle tangle)

Returns all transaction which reference directly or indirectly a transaction. Makes use of a BFS by traversing all direct approvers of direct approvers.
static Set<String> getFuture(String transactionHash, Tangle tangle)

Returns all transactions which are neither in P(v) nor F(v)
static Set<String> getIndependent(Set<String> past, Set<String> future, Tangle tangle)

Returns all direct approves/descendants of a transaction
static String[] getApproves(String transactionHash, Tangle tangle)

Returns all direct approvers/successors of a transaction
static Set<String> getApprovers(String txToInspect, Tangle tangle)

Returns all timestamps of a specific type
static List<Long> getTimestamps(Set<String> set, TimestampType timestampType, Tangle tangle)

3.7 RandomWalkProcedure.java

This class inherits and implements all methods from the AbstractTimestampingProcedure class. Besides that, it includes also procedure specific methods that are required for the confidence interval calculation:

Calculates the ratings of all transactions from a certain entry-point. These ratings will be processed and handled automatically by the random walk. This method returns a key-value mapping; with key representing a transaction and

the value its calculated rating.
Map<String, Integer> calculateRatings(String entry, Tangle tangle)

Walks towards the tips from a certain entry-point. Returns all transaction from the path.
LinkedHashSet<String> getPath(String entry, Tangle tangle)

3.8 QuantileProcedure.java

This class inherits and implements all methods from the AbstractTimestampingProcedure class. Besides that, it includes also procedure specific methods that are required for the confidence interval calculation:

Calculates the percentile of a multi-set.
long percentile(List<Long> list, double percentile)

3.9 TangleGenerator.java

For testing purposes, a Tangle generator was implemented. It follows a simple tip selection algorithm.

Create a new Tangle with a specific number of transactions
static Tangle createTangle(int size)

Continues a specific Tangle with a specific number of transactions
static void continueTangle(Tangle tangle, int size)

Returns two tip transaction of the Tangle.
static void getTransactionsToApprove(Tangle tangle)

4 Leap-seconds

Since we cannot expect IoT devices to be maintained with any leap-seconds announcements, we can not guarantee time to be exact on the second during those rare leap-second occasions [4]. Besides that, timestamps derived by a predictable rate seems much better suited to be handled by machines.

5 References

[1] On the timestamps in the tangle, by Serguei Popov
https://assets.ctfassets.net/r1dr6vzfxhev/4XgiKaTkUgEyW8O8qGg6wm/32f3a7c28022e35e4d5d0e858c0
On_the_timestamps_in_the_tangle_-_20182502.pdf

[2] Specification of timestamping.ixi, by Paul Handy
https://github.com/iotaledger/omega-docs/blob/master/ixi/timestamping/
Spec.md

[3] timestamping.ixi github repository
https://github.com/iotaledger/timestamping.ixi

[4] discussion about the problematic nature of leap seconds
https://discordapp.com/channels/397872799483428865/539546052714561538/
551796882725404682