Building and verifying a quasi-certification entity over Distributed Hash Tables

Join work with X. Bonnaire, R. Cortes & O. Marin
UPMC (France), UFSM (Chile), NYUS (China)

Fabrice.Kordon@lip6.fr
Certification, why?

Motivations

- Digital filling for tax purpose
  - Certify that somebody did it before a given deadline
- Certified emails
  - Use emails for legal purposes
- Online game refereeing
- e-voting, etc.
Certification, why?

Motivations

- Digital filling for tax purpose
  - Certify that somebody did it before a given deadline
- Certified emails
  - Use emails for legal purposes
- Online game refereeing
- e-voting, etc.

Existing Solutions

- Centralized
  - Public Key Infrastructures (traditional PKI)
  - Scaling problem/prone to faults/implementation (atomic multicast)
- Decentralized
  - Certification on top of Distributed Hash Tables (DHT)
  - Rapidly brings Byzantine consensus (for a 100.0% guarantee)
Certification, why?

Motivations

- Digital filling for tax purpose
  - Certify that somebody did it before a given deadline
- Certified emails
  - Use emails for legal purposes
- Online game refereeing
- e-voting, etc.

Existing Solutions

Centralized

- Public Key Infrastructures (traditional PKI)
- Scaling problem/prone to faults/implementation (atomic multicast)

Decentralized

- Certification on top of Distributed Hash Tables (DHT)
- Rapidly brings Byzantine consensus (for a 100.0% guarantee)

Objective

(quasi)-certify that a given action has been performed at a certain time

Distributed context (DHT)
3.4. Building Trust in Peer to Peer Systems

Building trust is a major concern in Peer to Peer systems \cite{ABE01,YU04}. The self organization of the nodes as well as the non-existing control of which node can enter or not into the system makes the presence of malicious nodes inevitable. In this context, the self organization and maintenance of the nodes, that appear initially one of the strongest features of a P2P system, becomes one of the most vulnerable ones because of the presence of malicious nodes.

A malicious node is an odd data node that does not follow the normal rules of the system. It can be a byzantine node, or a node that does not always behave maliciously (alternative nodes). The behaviour of malicious nodes can have various consequences on the Peer to Peer system. Malicious nodes can:

1. Reject queries and not respond to `Get()` or `Put()` operations. The transaction will then fail.
2. Make the routing algorithm fail, denying the network alive operation (this is a typical behaviour in Distributed Hash Tables).
3. Provide false information to the nodes (data integrity problems).
4. Prevent the nodes from correctly achieving their maintenance tasks.

Using traditional solutions to build trust based on a set of servers providing certification or any kind of control over the nodes that are members of the system is impossible.
3.4. Building Trust in Peer to Peer Systems

Building trust is a major concern in Peer to Peer systems [ABE 01][YU 04]. The self organization of the nodes as well as the non-existing control of which node can enter or not into the system makes the presence of malicious nodes inevitable. In this context, the self organization and maintenance of the nodes, that appear to be initially one of the strongest features of a P2P system, becomes one of the most vulnerable ones because of the presence of malicious nodes.

A malicious node is one that does not follow the normal rules so that the system. It can be a byzantine node, or a node that does not always behave maliciously (alternative nodes). The behaviour of malicious nodes can have various consequences on the Peer to Peer system. Malicious nodes can:

- Reject queries and may not respond to Get() or Put() operations. The transaction will then fail.
- Make other output algorithms fail, denying war damage (this is a typical behaviour in Distributed Hash Tables).
- Provide false information to the other nodes (data integrity problems).
- Prevent the other nodes from correctly achieving their maintenance tasks.

Using traditional solutions to build trust based on a set of servers providing certification or any kind of control over the nodes that are members of the system is impossible.
DHT in a nutshell

Retrieve data (key + value)

- put (v,k)
- get(k) → v
DHT in a nutshell

- Retrieve data (key + value)
  - put \((v,k)\)
  - get\((k) \rightarrow v\)

Diagram:
- Root node
- Leafset
- \(L\) close nodes (+ root)
DHT in a nutshell

- Retrieve data (key + value)
  - put (v,k)
  - get(k) → v

- Leafset
  - L close nodes (+ root)

- Root node

- Classical values for L
  - 8, 16, 32 (best)
A → an actor performing a service
Quasi-certification — entities

- **A** → an actor performing a service
- **S** → leafset `hash(service)` offering the service
Quasi-certification — entities

A → an actor performing a service

S → leafset hash(service) offering the service
Quasi-certification — entities

\(A\) → an actor performing a service

\(S\) → leafset \texttt{hash(service)} offering the service

\(C\) → certification authority leafset \texttt{hash(A/service)}

1 - request init

2 - transaction

End ack

3 - transaction ack
Quasi-certification — entities

A → an actor performing a service

S → leafset hash(service) offering the service

C → certification authority leafset hash(A/service)

1 - request init

2 - transaction

S

3 - transaction ack

4 - certificate generation

A

End ack

C

Certificate

Log Entry
Quasi-certification — protocol structure

<table>
<thead>
<tr>
<th>A</th>
<th>S</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Quasi-certification — protocol structure

- A requests leaf set
- receive leaf set
- A requests cert. service
- ack cert. service
Quasi-certification — protocol structure

A requests cert. service
receive leaf set
A requests leaf set
ack cert. service
the service
Quasi-certification — protocol structure

A requests cert. service

receive leaf set

A requests leaf set

ack cert. service

the service

nodes request leaf set

nodes receive leaf set

nodes ack transaction
Quasi-certification — protocol structure

1: A & S secure exchanges

2: exchanges to perform the service

3: S get trustset from C

4: C elaborates the side certificate

A requests cert. service

A requests leaf set

receive leaf set

ack cert. service

the service

nodes request leaf set

nodes receive leaf set

nodes ack transaction
Quasi-certification — protocol structure

1: A & S secure exchanges

2: exchanges to perform the service

3: S get trustset from C

4: C elaborates the side certificate

Majority ⇒ L/2+1 answers
Quasi-certification — protocol structure

1: A & S secure exchanges
2: exchanges to perform the service
3: S get trustset from C
4: C elaborates the side certificate

Diversity routing
To serve the leafset

Majority
⇒ L/2+1 answers
The verification process

Proof (by any method?)

Proven to be undecidable [FLP 85]
The verification process

Proof (by any method?)

- Proven to be undecidable [FLP 85]

So what?

- Being pragmatic
- Going for «quasi»
The verification process

Proof (by any method?)

- Proven to be undecidable [FLP 85]

So what?

- Being pragmatic
- Going for «quasi»

Two steps

1. Modeling the protocol in a perfect world (no error)
   - Use of Petri nets
2. Probabilistic analysis to evaluate the failure rate
   - Use of a classical fault model, building a formula + numeric evaluation
Hypotheses

- $H_1$: perfect world
- $H_2$: service reduced to 1 interaction
- $H_3$: L+1 answer requested instead of L/2+1

- Symmetric net with Bags?
Types and variables

type tsid is 0..L;
type tsidxtsid is <tsid, tsid>;

var i in tsid;
Types and variables

type tsid is 0..L;
type tsidxtsid is <tsid, tsid>;

var i in tsid;
Modeling the protocol (step 1)

Types and variables

```
type tsid is 0..L;
type tsidxtsid is <tsid, tsid>;
var i in tsid;
```
Types and variables

```plaintext
type tsid is 0..L;
type tsidxtsid is <tsid, tsid>;
var i in tsid;
```
Modeling the protocol (step 1)

Types and variables

```plaintext
type tsid is 0..L;
type tsidxtsid is <tsid, tsid>;
var i in tsid;
```
Modeling the protocol (step 1)

Types and variables

```plaintext
type tsid is 0..L;
type tsidxtsid is <tsid, tsid>;
var i in tsid;
```

Diagram of the protocol steps and variables.
Modeling the protocol (step 1)

Types and variables

type tsid is 0..L;
type tsidxtsid is <tsid, tsid>;
var i in tsid;
\[ F_{ok} = |S_{stopOK}| > \frac{L}{2} \land |C_{stopOK}| > \frac{L}{2} \]

\[ F_{ok} = |S_{stopOK}| = L + 1 \land |C_{stopOK}| = L + 1 \]
Modeling the protocol (step 1)

\[
F_{\text{abort}} = |S_{\text{stopAbort}}| > \frac{L}{2} \lor |C_{\text{stopAbort}}| > \frac{L}{2}
\]

\[
F_{\text{ok}} = |S_{\text{stopOK}}| = L + 1 \land |C_{\text{stopOK}}| = L + 1
\]

\[
F_{\text{abort}} = |S_{\text{stopAbort}}| > 0 \lor |C_{\text{stopAbort}}| > 0
\]
\[ F_{ok} = |S_{stopOK}| = L + 1 \land |C_{stopOK}| = L + 1 \]
\[ F_{abort} = |S_{stopAbort}| > 0 \lor |C_{stopAbort}| > 0 \]
\[ AF(F_{ok} \lor F_{abort}) \]
Verifying the perfect world

About the complexity of the state space

- Roughly $10^L$ states
- 1 state = 13 int + 17 multistep $\Rightarrow$ memory problem to check for $L=32$
Verifying the perfect world

About the complexity of the state space

- Roughly $10^L$ states
- 1 state = 13 int + 17 multistep $\Rightarrow$ memory problem to check for $L=32$

GreatSPN (use of symmetries)

- $L=24$ fails after 11h45mn of CPU (costly canonisation function)
  - Implementation limit = size of a hash table
Verifying the perfect world

About the complexity of the state space

- Roughly $10^L$ states
- 1 state = 13 int + 17 multistep ⇒ memory problem to check for $L=32$

GreatSPN (use of symmetries)

- $L=24$ fails after 11h45mn of CPU (costly canonisation function)
  - Implementation limit = size of a hash table

PNXDD (decision diagrams + variable ordering)

- Unfolding to P/T nets
- $L=10$ fails after 3h20mn (memory overflow > 16GB)
About the complexity of the state space

- Roughly $10^L$ states
- 1 state = 13 int + 17 multistep ⇒ memory problem to check for $L=32$

GreatSPN (use of symmetries)

- $L=24$ fails after 11h45mn of CPU (costly canonisation function)
  - Implementation limit = size of a hash table

PNXDD (decision diagrams + variable ordering)

- Unfolding to P/T nets
- $L=10$ fails after 3h20mn (memory overflow > 16GB)

ITS-Tools (hierarchical decision diagrams)

- Completed for $L=32$ in less than one minute
  - Handling of symmetries in the system by means of a dedicated encoding
Verifying the perfect world

CosyVerif

http://cosyverif.org
Probabilistic analysis (step 2)

Classical approach of the domain

- Based on $p$, probability of node failure
- Hypotheses required
- Diversity routing to avoid coalitions
Probabilistic analysis (step 2)

Classical approach of the domain

- Based on $p$, probability of node failure
- Hypotheses required
- Diversity routing to avoid coalitions

Origin of problems

- Source 1 $\rightarrow$ failure of the protocol
  - No answer to A + no ack to A
  - Interactions between A, S (leafset)
- Source 2 $\rightarrow$ inappropriate certificate
  - Lost of a certificate (inconsistency)
  - Interactions between S (leafset) and C (leafset)
Probabilistic analysis (step 2)

Classical approach of the domain

Based on $p$, probability of node failure
- Hypotheses required
- Diversity routing to avoid coalitions

Origin of problems

Source 1 → failure of the protocol
- No answer to A + no ack to A
- Interactions between A, S (leafset)

Source 2 → inappropriate certificate
- Lost of a certificate (inconsistency)
- Interactions between S (leafset) and C (leafset)

Numerical applications

$p = 0.3$ («untrusted»)
$P = 0.05$ («trusted»)
Formulas and experimental values

Protocol failure

More that $L/2$ nodes are malicious

The formula:

\[
\sum_{i=1}^{L+1} \binom{L+1}{i} p^i (1 - p)^{L+1-i} - \sum_{i=1}^{L/2} \binom{L+1}{i} p^i (1 - p)^{L+1-i}
\]

at most $L+1$ malicious nodes

at most $L/2$ malicious nodes
Protocol failure

More that \( L/2 \) nodes are malicious

The formula:

\[
\sum_{i=1}^{L+1} \binom{L+1}{i} p^i (1 - p)^{L+1-i} - \sum_{i=1}^{\frac{L}{2}} \binom{L+1}{i} p^i (1 - p)^{L+1-i}
\]

at most \( L+1 \) malicious nodes

Table 3 shows the probability of failure between DHT nodes. In this case, the probability of failure for each node of the trustset to retrieve the logs, or if more than \( L/2 \) nodes are malicious and make it impossible to obtain identical answers upon downloading the logs.

Table 4 gives the results for each probability for a certificate logs cannot be repaired if the node cannot contact the TrustedRing.

In the case when the malicious nodes represent a Certificate Generation and then assesses its message complexity. Therefore, the cost of the transaction between a node is leaving the trustset.

A new node enters the trustset.

A node is leaving the trustset.

Maintenance of the certificate logs fails if the node cannot encounter the DHT - \( p = 0.3 \) \( \begin{array}{c|c|c} \hline L & DHT - p = 0.3 & CORPS - p = 0.05 \\ \hline 8 & 0.188 & 6.64 \times 10^{-5} \\ 16 & 0.079 & 6.57 \times 10^{-8} \\ 32 & 0.016 & 8.24 \times 10^{-14} \\ \hline \end{array} \end{array} \)
Protocol failure

- More than $L/2$ nodes are malicious

The formula:

\[
\sum_{i=1}^{L+1} \binom{L+1}{i} p^i (1 - p)^{L+1-i} - \sum_{i=1}^{L/2} \binom{L+1}{i} p^i (1 - p)^{L+1-i}
\]

Inappropriate certificate generation

Two parts

- Existence of more $L/2$ malicious nodes
- Unable to retrieve at least $L/2 + 1$ identical answers

The formula (combines problems between $S$ and $C$)

\[
1 - \left(1 - P_{\geq \frac{L}{2}}\right)^2
\]
Protocol failure

More that L/2 nodes are malicious

The formula:

\[
\sum_{i=1}^{L+1} \binom{L+1}{i} p^i (1 - p)^{L+1-i} - \sum_{i=1}^{\frac{L}{2}} \binom{L+1}{i} p^i (1 - p)^{L+1-i}
\]

Inappropriate certificate generation

Two parts

- Existence of more L/2 malicious nodes
- Unable to retrieve at least L/2 + 1 identical answers

The formula (combines problems between S and C)

\[
1 - (1 - P_{\geq \frac{L}{2}})^2
\]

\[
1 - \left( 1 - \sum_{i=1}^{L+1} \binom{L+1}{i} p^i (1 - p)^{L+1-i} + \sum_{i=1}^{\frac{L}{2}} \binom{L+1}{i} p^i (1 - p)^{L+1-i} \right)^2
\]
Protocol failure

More than L/2 nodes are malicious

The formula:

\[ \sum_{i=1}^{L+1} \binom{L+1}{i} p^i (1 - p)^{L+1-i} - \sum_{i=1}^{L/2} \binom{L+1}{i} p^i (1 - p)^{L+1-i} \]

Inappropriate certificate generation

Two parts

- Existence of more L/2 malicious nodes
- Unable to retrieve at least L/2 + 1 identical answers

The formula (combines problems between S and C)

\[ 1 - (1 - P_{> \frac{L}{2}})^2 \]

\[ 1 - \left(1 - \sum_{i=1}^{L+1} \binom{L+1}{i} p^i (1 - p)^{L+1-i} + \sum_{i=1}^{L/2} \binom{L+1}{i} p^i (1 - p)^{L+1-i}\right)^2 \]
Conclusion

Quasi-certification entity *(elaboration + verification)*

- Low probability of failure + Good message complexity (not discussed)
Conclusion

**Quasi-certification entity (elaboration + verification)**

- Low probability of failure + Good message complexity (not discussed)
- Application to digital tax filling in France
  - 36.5 M revenues declarations (in 2012)
  - a wrong tax certificate every 5132 years
  - lost of a tax certificate every 997 years
Conclusion

**Quasi-certification entity** (elaboration + verification)

- Low probability of failure + Good message complexity (not discussed)

**Application to digital tax filling in France**

- 36.5 M revenues declarations (in 2012)
- A wrong tax certificate every 5132 years
- Lost of a tax certificate every 997 years

**3 years of work (details recently fixed)**

- Work partially published in TrustCom’2013
- Without formal proof (there was yet glitches details with hypotheses)
Conclusion

Quasi-certification entity (elaboration + verification)
- Low probability of failure + Good message complexity (not discussed)
- Application to digital tax filling in France
  - 36.5 M revenues declarations (in 2012)
  - a wrong tax certificate every 5132 years
  - lost of a tax certificate every 997 years

3 years of work (details recently fixed)
- Work partially published in TrustCom’2013
- Without formal proof (there was yet glitches details with hypotheses)

Realistic problem with applicability to e-government
- Probably numerous applications in the future
- The model is now a metric for the Model Checking Contest
- Potential applicability for Symmetric Nets with Bags
  - Excellent playground for this type of problems