Distributed Name Rewriting
(DINRG Feb 17, 2018, San Diego)

<christian.tschudin@unibas.ch> University of Basel
Overview

What is the common abstraction behind “distributed Internet infrastructures”?

Hypothesis: **Distributed Name Rewriting**

This talk revisits (in 15 min): DNS, BTC, ARP, DHT, NFN, Tangle, GitHub …

Other possible names (instead of DNR) from this morning:
- *distributed secure mappings* (*bind identities to keys*), *potentially trustless*
- *NFaaS* (*securely map inputs to computation results*)
DNS vs BTC

DNS:

\[
\text{lookup(in)} \rightarrow \text{out}
\]

e.g. ‘in’ is hierarchical domain name, ‘out’ is IP number

BTC:

\[
\text{lookup(in)} \rightarrow \text{out}
\]

‘in’ is random (account or tx) number, ‘out’ is an account balance, a transaction, a smart contract, an insurance policy etc
DNS vs BTC (contd)

At the end of the day, DNS and BTC both are:

• small to midsize databases (BTC is a ledger, after all)
• global (distributed or replicated)
• simply query interface, maps one name to another

There are differences, of course …
# DNS vs BTC - somehow different

<table>
<thead>
<tr>
<th></th>
<th>DNS</th>
<th>BTC</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>versioning</strong></td>
<td>no</td>
<td>yes (IOTA has snapshots and forgets)</td>
</tr>
<tr>
<td>(history)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>consistency</strong></td>
<td>eventually cons. (dependent on caching params)</td>
<td>strong cons. IFF you are on the winning branch</td>
</tr>
<tr>
<td><strong>input names</strong></td>
<td>unique because pre-coordinated</td>
<td>unique because random</td>
</tr>
<tr>
<td>(how to prevent conflicts)</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>decentralized storage</strong></td>
<td>yes (iterative/recursive remote query)</td>
<td>no (full replica)</td>
</tr>
</tbody>
</table>

one very big difference, though:
DNS vs BTC - really different

b/c of the UPDATE method

DNS: “pre-established agreement on delegation”
   updates only possible in delegated subtree, are independent and can be done in parallel

BTC: trustless process
   Byzantine Agreement Protocol for global, synchronous consensus

Is “distributed name rewriting” still a good common abstraction? I think yes.
More Name-Rewriting Infrastructures

Seen so far: DNS, BlockChain (BTC), Tangle (IOTA)

**ARP** - dynamic mapping

**Forwarding** - routing table with next-hop lookup

**DHT** - an index, beside DNS the other “exemplary lookup” infrastructure

**PKI** - secured_lookup(some_public_key) → signing_key

**cloud computation** - lookup( fct(in) ) → result
Web pages are computation results: lookup results are cacheable, see memcachd

**NFN** (Named-Function-Networking) — resolve( symbolic_expr ) → result

scalable! immutable inputs, *confluence of resolution strategies avoids need for consensus finding*

Again: “update” is probably the strongest differentiator
Communication is Computation is Distributed Name Rewriting is Communication is ...

Notation used: \( A(\text{something}) \) means: "something is on host A" config[..] represents global state

Story: We want to replicate an item, send a unicast datagram from A to B via X

\[
\text{config}\[ A(\text{srcA}, \text{nameB}, \text{item}), B(\text{srcB}) \]}
\[
\rightarrow \text{name rewriting due to DNS: map nameB to B's IP address}
\]
\[
\text{config}[ A(\text{srcA}, \text{nameB}, \text{dstB}, \text{item}), B(\text{srcB}) ]
\rightarrow \text{name rewriting due to route table lookup: map dstB to gwX}
\]
\[
\rightarrow \text{name rewriting due to ARP: map gwX IP name to eth name}
\]
\[
\text{config}[ A(\text{srcA}, \text{nameB}, \text{item}), \text{lan1(ethX,srcA,dstB,item)}, B(\text{srcB}) ]
\rightarrow \text{delivery at gateway X}
\]
\[
\text{config}[ A(\text{srcA}, \text{nameB}, \text{item}), X(\text{pkt(srcA,dstB,item)}), B(\text{srcB}) ]
\rightarrow \text{name rewriting due to route table lookup: map dstB to dstB}
\]
\[
\rightarrow \text{name rewriting due to ARP: map dstB IP name to eth name}
\]
\[
\text{config}[ A(\text{srcA}, \text{nameB}, \text{item}), \text{lan2(ethB,srcA,dstB,item)}, B(\text{srcB}) ]
\rightarrow \text{delivery at B}
\]
\[
\text{config}[ A(\text{srcA}, \text{nameB}, \text{item}), B(\text{srcB}, \text{pkt(srcA,dstB,item)}) ]
\rightarrow \text{delivery at application level:}
\]
\[
\text{config}[ A(\text{srcA}, \text{nameB}, \text{item}), B(\text{srcB}, \text{item}) ] \quad \# \text{voila: the item was replicated through DNR}
\]
Name-ReWriting Service, the API

Name-Rewriting as an Abstract Data Type (ADT), basically a key-value store

class NaRW: # a name rewriting service, its interface
    def get():     # also called "lookup", "resolve", "compute"
    def put():     # also called "update", "define", "undefine"
    def items():  # also called "walk", "listdir"

DNS, BTC, etc are then subclasses, type refinements, interface implementors.

Goal of this “ADT talk” is to abstract away from the implementation details, define the ADT by its properties, not the implementation
Name-ReWriting Service, the API

Name-Rewriting as an Abstract Data Type (ADT), basically a key-value store.

class NaRW: # a name rewriting service, its interface

    def get(): # also called "lookup", "resolve", "compute"
    def put(): # also called "update", "define", "undefine"
    def items(): # also called "walk", "listdir"

DNS, BTC, etc are then subclasses, type refinements, interface implementors.

Goal of this "ADT talk" is to abstract away from the implementation details, define the ADT by its properties, not the implementation.
Name-ReWriting Service, the API

Name-Rewriting as an Abstract Data Type (ADT), basically a key-value store.

class NaRW: # a name rewriting service, its interface
def get(): # also called "lookup", "resolve", "compute"
def put(): # also called "update", "define", "undefine"
def items(): # also called "walk", "listdir"

DNS, BTC, etc are then subclasses, type refinements, interface implementors.

Goal of this "ADT talk" is to abstract away from the implementation details, define the ADT by its properties, not the implementation.

A potential DIN result: Name-ReWriting-as-a-Service spec
Implementing NaRW with two sub-services

Hypothesis: DIN will revisit these two services over and over

a) **Persistent storage** to store a new item (**lookup(id) → data**)
   take some CRUD database (**create,read,update,delete**), potentially append-only

b) **head- (or “tip”) service** -- points to the most recent versions of an item

The rest is chaining items to other items via hash pointers (**= items’ intrinsic names**)

Intuition:

- GitHub, BlockChain (fuses a and b), IOTA’s tangle has multiple tips

- DNS has/is only head-service, ICN offers only storage …
The sweet spot for scalability and trustlessness?

NaRW API (put, get, items)

“Conflict-Free Replicated Data Types” (CRDT): deterministic eventual consistency without consensus, hence scalable

--- medium guarantees ---

persistent storage service

head- ("tip") service

--- strong guarantees ---

DNS scales but:
- no history, trust-based,
- no auto-conflict resolution

BlockChain à la BTC:
- trustless, history, does not scale

Tangle-style, w/o consensus
The sweet spot for scalability and trustlessness?

--- medium guarantees ---

NaRW API (put, get, items)

e.g. scalable key-value store with “observed removal semantics”,
or a voting DIN, etc

--- strong guarantees ---

"Conflict-Free Replicated Data Types" (CRDT):
deterministic eventual consistency without consensus, hence scalable

DNS scales but:
no history, trust-based, no auto-conflict resolution

BlockChain à la BTC:
trustless, history, does not scale

tangle-style, w/o consensus

persistent storage service
head- ("tip") service
Questions