Internet Engineering Task Force
Internet-Draft
Intended status: Standards Track
Expires: April 21, 2016

H. Bellur
C. Wendt, Ed.
Comcast

October 19, 2015

Distributed Registry Protocol
draft-wendt-modern-drip-00

Abstract

This document describes a protocol for allowing a distributed set of nodes to synchronize a set of information in real-time with minimal amount of delay. This is useful for registry types of information like identity and telephone numbers with associated routing and ownership information and could be extended to support other distributed real-time information updates as well.

Table of Contents

1. Introduction

This document describes the Distributed Registry Protocol (DRiP). DRiP defines a set of peer protocols for how an arbitrary number of nodes arranged in a distributed mesh architecture can be used to synchronize data in real-time across a network.

1.1. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Initiator Node:
A node that initiates data propagation.

Receiver Node:
A node that forwards the propagated key-value data.

2. DRiP Overview

DRiP uses a mix of a gossip protocol with update counters for distribution of key-value data with the addition of a voting system to avoid race conditions on writing of key-value data.

3. Distributed MESH Architecture

The DRiP architecture is based on a peer-to-peer communication model where a given node associated with a data store is not necessarily aware of the total number of nodes in the entire network. Minimally, every node should reachable by at least one multi-node path from every other node. Each node in the DRiP network maintains a list of peer nodes from which it receives and transmits updates. Information is propagated by forwarding to it's peer nodes until the information received by a node has already been received.

```
    ___           ___                         ___           ___
   |DB |_________|DB |                       |DB |_________|DB |
   |___|         |___|                       |___|         |___|
     |   Data      |                           |   Data      |
     |   Store     |                           |   Store     |
    _|_  Cluster  _|_                         _|_  Cluster  _|_
   |DB |_________|DB |                       |DB |_________|DB |
   |___|         |___|                       |___|         |___|
                     \                       /
                      \                     /
                      _\___     DRIP     _ /__
                     |Node |------------|Node |
                     |  A  |    HTTPS   |  C  |
                     |_____|            |_____|
                         \H           H/
                         D\T         T/D
                          R\T       P/R
                           I\P     P/I
                            P\S   S/P
                            __\__ /   DRIP     _____
                           |Node |------------|Node |
                           |  B  |   HTTPS    |  D  |
                           |_____|            |_____|
                          /                   /
       ___           ___ /                 __/_          ___
      |DB |_________|DB |                 |DB |_________|DB |
      |___|         |___|                 |___|         |___|
        |   Data      |                     |   Data      |
        |   Store     |                     |   Store     |
       _|_  Cluster  _|_                   _|_  Cluster  _|_
      |DB |_________|DB |                 |DB |_________|DB |
      |___|         |___|                 |___|         |___|

                       Distributed Mesh Architecture
```

4. DRiP procedures

4.1. Distributed Registry Rules

All nodes in the distributed mesh MUST agree upon a specific key-value data model. The choice of data store is implementation specific.

All nodes MUST be configured with at least one peer node before propagation.

A node MUST ignore any updates or commands it receives from other nodes that are not configured as peer nodes.

All nodes MUST send a periodic heartbeat or keep-alive message via HTTPS to the respective peer nodes. If a heartbeat is not received the peer node is removed from the list of active peer nodes.

4.2. Node State

The peer node should maintain a state that defines whether it is active, inactive, or synchronizing key-value data with a peer node.

The node should proactively tell it's peer nodes its state by sending the following POST messages. The GET query is available for nodes to query the state of peer nodes.

4.2.1. API - POST /node/:nodeid/active

Request:

POST /node/:nodeid/active

Description:

TBD

Example:

TBD

4.2.2. API - POST /node/:nodeid/inactive

Request:

GET /state

Description:

A node should query the state of its peer node before it initiates a sync operation. This request responds with either "active" or "sync" or no response, if in "inactive" state.

Example:

TBD

4.2.3. API - GET /state

Request:

POST /node/:nodeid/active

Description:

TBD

Example:

TBD

4.3. Custom HTTP header fields

Custom HTTP header fields will be used to carry node specific information.

Field Name	Description
DRiP-Node-ID	Each node in the mesh MUST have a unique identifier. An Initiator node MUST set its own node ID as the field value. A Receiver Node MUST NOT change the DRiP-Node-ID field value as it forward the HTTPS request to its peer nodes.

Example:

```
DRiP-Node-ID: xyz
```

Field Name	Description
DRiP-Node-Counter	Every node maintains a count of the number of times it initiates key-value data propagation. This counter MUST be an unsigned type, typically, a 64 bit integer. The Initiator node MUST set this count as the field value. A Receiver Node MUST NOT change the DRiP-Node-Counter field value as it forward the HTTPS request to its peer nodes.

Example:

```
DRiP-Node-Counter: 123
```

Field Name	Description
DRiP-Node-Counter-reset	A node can reset the count (to zero) of the number of times it initiates key-value data propagation. If the counter value is reset, prior to initiating data propagation, then this field value MUST be set to true. Otherwise, it MUST be set to false, at all times. A typical use case to reset the counter value is when the counter (of unsigned type) value wraps around. The Initiator node MUST set this field value to either true or false. A Receiver Node MUST NOT change the DRiP-Node-Counter-reset field value as it forward the HTTPS request to its peer nodes.

Example:

```
      DRiP-Node-Counter-reset: false
```

Field Name	Description
DRiP-Transaction-Type	The Initiator node MUST set this field value to be either "update" or "sync". A Receiver Node MUST NOT change the DRiP-Transaction-Type field value as it forward the HTTPS request to its peer nodes.

Example:

```
      DRiP-Transaction-Type: update
```

Field Name	Description
DRiP-Sync-Complete	For sync transaction type, the Initiator node MUST set this field value to be true, if synchronization is complete. Otherwise, this field value MUST be set to false.

Example:

```
      DRiP-Sync-Complete: false
```

4.4. Key-Value Data Propagation Rules

A node propagates key-value data to all its peer nodes except the the node from which it received data. For example, in Figure 1, when node B receives key-value data from node A, it will propagate the data received to nodes C and D but not back to node A.

For each transaction type (Update or Sync), the following set of actions MUST take place when a node receives a HTTPS request with propagated key-value data:

- o If DRiP-Node-ID field value (in the HTTP header) contains Initiator node ID that has never been seen, both DRiP-Node-ID and DRiP-Node-Counter field values MUST be stored for future reference and the key-value data is propagated to all peer nodes.

- o If DRiP-Node-ID field value (in the HTTP header) matches with a stored node ID and DRiP-Node-Counter-reset field value is false.

  * The received key-value data MUST be propagated to the peer nodes if DRiP-Node-Counter field value is greater than the saved counter value. The DRiP-Node-Counter field value MUST be saved as the new counter for the stored node ID.

  * If DRiP-Node-Counter field value is less than or equal to saved counter value, then the key-value data has already been received and MUST NOT be propagated to peer nodes. This ensures that propagation stops when all nodes have received the key-value data from the Initiator node.

- o If DRiP-Node-ID field value matches with a stored node ID and DRiP-Node-Counter-reset field value is true:

  * The received key-value data MUST be propagated to the peer nodes. The DRiP-Node-Counter field value MUST be saved as the new counter for the stored node ID.

4.5. Key-Value Data Update

When an Initiator node has new data it wants to propagate to the distributed mesh, it initiates an Update. The Update consists of a two-phase commit (2PC) procedure in order to guarantee there are no race conditions for updating the same key's data, as well as for any error conditions in the distributed mesh that would cause the update to not complete for all nodes in the network.

The two phases are called the "voting" phase and the "commit" phase.

```
                               _________
      -----------------------<|         |
     |                        | Waiting |
     |                        | For     |
     |   ---------------------| Events  |
     |  | (Update,            |_________|
     |  |  Start Timer)
     |  |                            --------------------------------
     |  |                           | Received Update From Peer Node |
     |  |                           |                                |
     |  |             ______________|_  If key matches an            |
     |  |            |                |  in-progress update          |
     |   ----------->|                |  vote "no".                  |
     |               | Waiting For    |  Otherwise, vote "yes".      |
     |               | Response From  |                              |
     |               | Peer Nodes     |<-----------------------------
     |               |                |
     |           ----|                |----
     |  Timer   |    |________________|    |
     |  Expired |                          | Received Votes
     |          |                          | From All Peer
     |          |                          | Nodes
     |          |      _______________     |
     |          |    |                |    |
     |          |    |                |    |
     |           --->|                |<---
     |               |  Validating    |
     | (If all Votes |  Votes         |
     |  are "YES",   |                |
     | propagate     |                |
     | commit)       |                |
      ---------------|________________|
```

                           Update State Diagram

4.5.1. Voting Phase

The voting phase is the phase where all nodes are queried to "vote" whether they are aware of any potential conflict that would cause the transaction not to complete.

The Initiator node MUST set a timeout period to get response from its peer nodes.

The peer nodes known to the initiator node will continue propagate the information to their peer nodes and so on. However, these peer nodes beyond the initiator node will no longer need to keep track of the time interval for responses. A node will stop continuing to

propagate information when it determines it has received the same information again. This can be determined by keeping track of the counter and originating node id.

If all peer nodes vote "yes", then the second phase or commit phase in the local node is initiated. If any node in the distributed mesh votes "no" or if the timeout period expires and all peer nodes have not responded, then the commit of the information MUST NOT be completed. No action is taken for responses received after the timeout period.

Note: The voting procedure is intentionally split into two separate full HTTP transactions for reliability.

```
   ___           ___      ___           ___      ___           ___
  |DB |_________|DB |    |DB |_________|DB |    |DB |_________|DB |
  |___|         |___|    |___|         |___|    |___|         |___|
    |   Data      |        |   Data      |        |   Data      |
    |   Store     |        |   Store     |        |   Store     |
   _|_  Cluster  _|_      _|_  Cluster  _|_      _|_  Cluster  _|_
  |DB |_________|DB |    |DB |_________|DB |    |DB |_________|DB |
  |___|         |___|    |___|         |___|    |___|         |___|
      \                      \                      |
       \                      \                     |
       _\___    Vote(HTTPS)  _\___    Vote(HTTPS)  |____
      |Node |  <----------  |Node |  ---------->  |Node |
      |  B  |---------------|  A  |---------------|  C  |
      |_____|  ---------->  |_____|  <----------  |_____|
                 Yes/No                 Yes/No
```

Voting Phase

4.5.1.1. API - POST /voting

Request:

POST /voting

Description:

A post from either Initiator node or subsequent peer nodes to request a vote of "yes" or "no" whether the key-value data could be committed without error or conflict.

Example (using cURL)

Request

```
$ curl -i -H "Content-Type: application/json" -H "DRiP-Node-ID:
  nodeA" -H "DRiP-Node-Counter: 1234" -H
  "DRiP-Node-Counter-reset: false" -X POST -d '{<key-value
  data>}' https://nodebregistry.com/voting
```

Response

```
HTTP/1.1 200 OK
```

4.5.1.2. POST /votingphase/node/:nodeid/response/:response

Request:

POST /voting/peernode/:nodeid/response/:response

Description:

A POST from peer node back to node with response of vote.

Example (using cURL)

Request

```
$ curl -i -X POST http://nodearegistry.com/node/nodeA/response/yes
```

Response

```
HTTP/1.1 200 OK
```

4.5.2. Commit Phase

The Initiator node, that originated the gossip, upon receiving a successful aggregated "yes" vote from all the peer nodes should start the commit phase. This node MUST commit the data to its data store. Subsequently, this information is propagated to all the nodes so that each node in the mesh will commit the same information in their respective data stores.

```
         ___           ___                         ___           ___
        |DB |_________|DB |                       |DB |_________|DB |
        |___|         |___|                       |___|         |___|
          |   Data      |                           |   Data      |
          |   Store     |                           |   Store     |
         _|_  Cluster  _|_                         _|_  Cluster  _|_
        |DB |_________|DB |                       |DB |_________|DB |
        |___|         |___|                       |___|         |___|
                          \                       /
                           \COMMIT               /COMMIT
                           _\___     COMMIT   _ /__
                          |Node |------------|Node |
                          |  A  |    HTTPS   |  C  |
                          |_____|            |_____|
                              \H           H/
                               \T         T/
                          COMMIT\T       P/COMMIT
                                 \P     P/
                                  \S   S/
                                 __\__ /  COMMIT    _____
                                |Node |------------|Node |
                                |  B  |   HTTPS    |  D  |
                                |_____|            |_____|
                               /                   /
                              /COMMIT             /COMMIT
            ___           ___/                  _/__          ___
           |DB |_________|DB |                 |DB |_________|DB |
           |___|         |___|                 |___|         |___|
             |   Data      |                     |   Data      |
             |   Store     |                     |   Store     |
            _|_  Cluster  _|_                   _|_  Cluster  _|_
           |DB |_________|DB |                 |DB |_________|DB |
           |___|         |___|                 |___|         |___|
```

Commit Phase

4.5.2.1. API - POST /commit

Request:

POST /commit

Description:

A commit message is sent from Initiator or subsequent peer nodes to signal the Receiver node to commit the data to its data store.

Example (using cURL)

Request

```
$ curl -i -H "Content-Type: application/json" -H "DRiP-Node-ID:
   nodeA" -H "DRiP-Node-Counter: 1234" -H
   "DRiP-Node-Counter-reset: false" -X POST -d
   '<key-value data>' https://nodebregistry.com/commit
```

Response

```
HTTP/1.1 200 OK
```

4.6. Node Sync Operation

A node, either newly added to the distributed mesh or put back into service after being inactive, will get the state of a peer node to determine if it is in "active" state. If so, the node can immediately initiate a Sync transaction. The peer node MUST start propagating a comprehensive and complete set of key-value data from its data store.

The two phase commit does NOT apply here as the contents of the initiating node's data store is either outdated or empty. During this phase (HTTPS requests received will have DRiP-Sync-Complete field value set to false), this node SHOULD NOT become an Initiator node to provision data. While this transaction is going on, this node MUST vote "yes" to all real-time updates. The commits corresponding to the Updates should also be completed and reflected in the data store.

4.6.1. API - PUT /sync/node/:nodeid

Request:

PUT /sync/node/:nodeid

Description:

API call for initiating a full registry synchronization from node to peer-node.

Example (using cURL)

Request

```
$ TBD
```

Response

HTTP/1.1 200 OK

4.7. Heartbeat

TBD - definition of heartbeat

TBD - determination of the need to sync after heartbeat fails

4.8. Key-Value Data Update Entitlement Verification

When a node owner would like to create or modify particular key-value data, generally in the context of a registry, there MAY be a verification procedure that key-value data write or modification can be performed. This could include validating whether key-value data is entitled to be written, modified or subsequently propagated based on application policy. For example, identity or telephone number ownership or porting. The exact mechanics of this are out of scope of this document and are generally application specific.

5. Security Considerations

5.1. HTTPS

All nodes MUST perform HTTP transactions using TLS as defined in [RFC7230].

5.2. Authentication

Secure authentication of node to node communication is beyond the scope of this document, however best practices in terms of protecting the node API interface should be followed.

6. References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC7230] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

Authors' Addresses

Harsha Bellur
Comcast
One Comcast Center
Philadelphia, PA 19103
USA

Email: Harsha_Bellur@cable.comcast.com

Chris Wendt (editor)
Comcast
One Comcast Center
Philadelphia, PA 19103
USA

Email: chris-ietf@chriswendt.net