Abstract

The BioCluster protocol is a session based protocol that uses TCP for data transfer, and can be bootstrapped using UDP multicast. Nodes participating in the BioCluster use this protocol to connect to each other and exchange configuration and data. Each node maintains a complete set of the database in order to allow foreign entities (e.g. the Asterisk PBX) to use the data stored.

Usage

The network uses a hashing function in order to assign to each node and to data unit a position on a circular keyspace, in a way similar to the Chord distributed hash table. Each node is considered responsible and authoritative to the part of the keyspace that is "lower" than its own node identification hash value up to the next "lower" node, and uses nodes "higher" in the keyspace to back up information for redundancy. Data in the network database is stored on keys, where a key is a URI that identifies a data object uniquely, and each key is "stored" in the circular keyspace at the "location" of its hash value according to the network's hash function.

In order to locate a data object on the network, a node need to formulate a URI that describes the requested data, either exactly or a URI that describes a larger group that is known to contain the requested data. The node would then compute the hash value of the URI and consult its own known peer list to locate a peer whose node identifier is most immediately higher then the key's computed hash value. That peer is known as the most authoritative node and can be queried for data pertaining to the specified URI.

The peer network uses strong authentication to prevent "unauthorised" nodes from participating in the network but does not use encryption. Strong authentication is achieved by each node having a pre-configured "network secret" that is used to sign authentication tokens passed in order to construct any peer-to-peer connection.

The process of logging in to the network - also known as a 4-way handshake - is as follows:

- A node that wishes to connect to another peer will create a new TCP connection to the peer's known address and port, and will send a ''Discover'' message asking for permission to access the network. See below for if the node does not know of any peer in the network.
- The peer shall then decide on an arbitrarily selected (and unique as possible) "nonce" value and send back an ''Offer'' request with that nonce value.
- The initiating node, upon receiving the ''Offer'' will encrypt the nonce using the network's pre-configured shared secret, and then use the network's hash function to retrieve a digest of the encrypted nonce. It will then send back a ''Request'' message with the computed hash value, asking to be authenticated for access.
- The peer shall examine the value received in the ''Request'' message and compare it to a value computed the same way as above from the nonce value it sent in the ''Offer'' message. If the values match, then the connection is authenticated and the peer will send back an ''Ack'' message authorising the new node to continue communications.

In case a new node has no knowledge of an existing peer (or have tried and exhausted all known peer addresses), a new node may boot strap itself by sending the initial ''Discover'' request over multicast UDP to a known port. Each functioning peer on the local network must listen on the known port for tor such UDP message, and respond with an ''Offer'' message sent using unicast UDP to the source address of the ''Discover'' message. The second half of the 4-way handshake will be performed over TCP as normal after the new node opens a new TCP connection to a peer that sent such an ''Offer''.

For the reliable functioning of the network, the following items must be known in advance by each node participating in the network, and thus should be pre-configured:

- The network's shared secret - shall be an arbitrary bit stream and should be stored locally as securely as possible.
- The network's hashing function. Currently SHA-1 is used.
- The network's known UDP port. Currently 4521 is being used.

Dependent Nodes

In addition to a full fledged peer, the management network recognises a type of degenerate node that has limited capabilities. Such a limited use node can be used by a foreign entity to temporarily connect to the network, retrieve or update information, and disconnect. A dependent node is therefore not a full member in the peered network and while it has a node identifier, it shall not be included in the distributed hash table and can never be an authoritative node. Moreover, a dependent node cannot communicate directly with the network and while it uses the same protocol (including the authenticated 4-way handshake) it may only connect to and communicate with a single peer, dubbed the "supporting node".

A dependent node may issue queries and receive data responses, and may also issue ''Authoritative Update''s to notify of requested changes in the database. It MUST not propagate updates using ''Update'' messages, nor accept connections from other peers.

Wire Protocol

The wire protocol is a binary protocol where each message contains a mandatory header:

Bit Size	Value	Description
16	Version Number	Used to notify the receiving node of the version of the version of the protocol
16	Op Code	The type of message being carried by the message
32	Transaction ID	Transaction identification for this message, where all messages that carry the same transaction id are to be considered the same message
HASH-SIZE	Node Identifier	A sequence of bits that represents this node in a unique and uniform way
HASH-SIZE	Seq	A sequence of bits used for cryptographic challenge-response (includes sequencing). for every packet which is sent as a reply to other packets (except for OFFER): $Seq = HASH($site-key . $prev->Seq)

Operation: Ack

Acknowledgement of the previous message with the same transaction ID.

No further data is to be expected.

Operation: Nack

Rejection of the previous message with the same transaction ID.

No further data is to be expected.

Operation: Discover

Discover new nodes in the network and start a login protocol (4-way handshake) with them. If the sending node already knows an IP address of an exiting peer, it can start a TCP connection to the previously known port and address and perform the login 4-way handshake entirely over TCP. In the case of a new node that has no previous or preconfigured knowledge of an existing node (or when such knowledge was deemed invalid), a node initiates a network-local bootstrapping process by sending a ''Discover'' message using UDP multicast on a known port. Any node that listens on the known UDP port must reply with a unicast UDP ''Offer'' message, and the "discovering" node may then continue the second part of the 4-way handshake over TCP directly with each offering node.

No further data is to be expected.

Operation: Offer

Offer a challenge to a new peer and allow it to complete the network handshake, proving that it knows the network's secret key. When responding to a UDP ''Discover'' message, ''Offer'' will be send using unicast UDP, and the sending peer has to maintain the sent "nonce" value used across the forthcoming switch from UDP to TCP. If the ''Discover'' message was sent using TCP then the ''Offer'' response shall be sent back on the same connection.

The ''Offer'' message contains these additional fields:

Bit Size	Value	Description
HASH-SIZE	Nonce value	An arbtirarilly selected value the size of the network's hash function result

Operation: Request

Request rights to access the network. The sending node authenticates that it knows the network's secret key and hash function by transforming the nonce valued received through the previous ''Offer'' message in a way that cannot be performed without this knowledge and cannot be replayed. If the first part of the network handshake (''Discover'' and ''Offer'') was done over UDP then the ''Request'' message will be sent by opening a new TCP connection to the node sending the ''Offer'' message, to the source address and port of the ''Offer'' message, and the ''Request'' message will be sent on as the first message on that connection. If the first part of the handshake was already done over TCP then ''Request'' will be sent also on the same TCP connection as previous messages.

A node sending a ''Request'' message shall only do so after receiving an ''Offer'' message and will set the ''Request'' authentication value by using the network's preconfigured shared secret to encrypt the nonce value submitted in said ''Offer'' message, then inputting it into the network's hash function and using the result as the authentication value.

A node receiving a ''Request'' message should examine the remembered nonce value that was sent, perform the same computation on that nonce value and compare the result to the authentication value in the ''Request''. If they match, then the receiving node MUST mark the TCP connection as valid and send back an ''Ack'' message, else the receiving node MAY send back a ''Nack'' message and SHOULD close the connection - such a connection is not valid and any other messages received on it should be disregarded.

The "Request" message contains data units that are considered "self" data of the sending node. These contain information about the requesting node which otherwise can't be found out by other peers. It this way the "Request" packet structure is similar to the "Update" packet, and a peer receiving the "Request" packet from a node should treat the data in it as an authoritative update.

The ''Request'' message contains these additional fields:

Bit Size	Value	Description
HASH-SIZE	Authentication value	The result of computing the authentication value from a nonce value
32	Data Size	The expected size (in bytes) of the elements composing the data section for the packet
32	Key Length	Signed 32 bit integer which represents the size in bytes of the key field
Key Length * 8	Payload key	A URI that is used to uniquely identify the data being carried
32	Data Length	Signed 32 bit integer which represents the size in bytes of the data field
Data Length * 8	Payload data	Data object associated with the above key

Query And Data Operations

For all other elements, an additional field is to be expected:

Bit Size	Value	Description
32	Data Size	The expected size (in bytes) of the elements composing the data section for the packet

Operation Query:

Query the receiving node for some information. The data section would contain data payloads of the following format, with total size exactly the same as `Data Size`:

Bit Size	Value	Description
32	Key Length	Size in bytes of the key field
Key Length * 8	Key	A URI that is used to uniquely identify the data being carried

A node receiving a ''Query'' message should respond with a ''Data Response'' message with a payload that answers the query to the best ability of the receiving node with regard to the authority of the given node for the specified keys.

Operation Data Response:

A response to a query received, that contains the sending node's response to the receiving node for some information queried by the previous query with the same transaction ID. The data section would contain data payloads of the following format, with total size exactly the same as `Data Size`:

Bit Size	Value	Description
32	Key Length	Signed 32 bit integer which represents the size in bytes of the key field
Key Length * 8	Payload key	A URI that is used to uniquely identify the data being carried
32	Data Length	Signed 32 bit integer which represents the size in bytes of the data field
Data Length * 8	Payload data	Data object associated with the above key

The following semantics apply to the data carried by the data response:

- If the queried URI is contained in at least one of the data payloads in the

response, then the response will be deemed to contain any and all information available to the sending node for that request, persuant to the authority of the sending node for that key.
- If the queried URI is contained in exactly one of the data payloads in the response, and the data object size for that payload is a negative number (i.e., has the most significant bit enabled), then the response will be deemed to contain the information that the sending node knows to the extent of its authority on that key, that the key is unbound (never existed, or have existed and deleted).
- If the queried URI is not contained in any of the data payloads in the response, then the response will be deemed to represent the unauthority of the sending node to have any information available on that key.

The special case where the response contains multiple data payloads with the same key, where one or more has a negative data size is undefined and implementations encountering this case SHOULD disregard all information regarding the specific key and consider the response not authoritative for that key.

Operation: Update

Request a node to update its database with new or modified data associated with a key, or request a node to delete data associated with a key. The format of the message is the same as the ''Data Response'' message, with somewhat reverse semantics regarding payloads with a negative data length - if a payload in an update message has a negative data length, then the update is instructing the receiving node that the specified key was disassociated from the database and it should remove any data associated with it. A node that receives an ''Update'' message should perform the request and send back an ''Ack'' if the operation was successful, or ''Nack'' if it wasn't.

When a node receives an ''Update'' message, it needs to check if it has already seen in the past an update with the specified transaction identifier, and if not it should forward the update (after sending the ''Ack'') to the node with the next higher node identifier, in order to make sure the entire network is updated.

Operation: Authoritative Update

Request the node authoritative for a given key to transactionally update the network database with the given information. This message has exactly the same format as ''Update'' and ''Data Response'' but carries different semantics and expected behaviour: A node should use ''Authoritative Update'' when it receives a valid (i.e. authenticated) request to update information which is new to the network and not pushed from a peer node - for example, from a dependant node that is trying to update the network database with new information. In such a case, a node that needs to update the network database will start the following process:

- A node receiving an ''Authoritative Update'' should check if it believes itself to be the most authoritative node for that key. If it is not the case:
  - It must forward the message to the node it believes to be the most authoritative for that key and wait for an ''Ack''.
  - Once an ''Ack'' or ''Nack'' was received, it will forward the ''Ack'' or ''Nack'' to the originating node.
- If a receiving node confirms that it is indeed the most authoritative node for the specified key, then:
  - It MUST update its own database
  - Select the next ''n'' less authoritative nodes and send them a normal ''Update'' message and wait for an ''Ack'' from each one (n being the network's redundancy constant).
  - Once enough nodes have acknowledged the update, the authoritative node will send back an ''Ack'' to the originating node notifying it of the completion of the transaction.
  - If not enough nodes have acknowledged the update, then the

authoritative node MUST revert the update to its own view of the database and send back a ''Nack'' to the originating node.
- The authoritative node should then send a normal ''Update'' message to the next higher value node as per the ''Update'' procedure if it did sent this node an ''Update'' previously.
- If the node that has the new information considers itself the most authoritative node for the update, then it should behave as the most authoritative node as above.

For both ''Update'' and ''Authoritative Update'', if a message is not acknowledged in a reasonable time, then the sending node MUST deem the receiving node as invalid, remove it from its peer list, recalculate a new peer to send the update to and send the new peer the message instead.