XEP-0362: Raft over XMPP

Peter Membrey
mailto:peter@membrey.hk
xmpp:peter@membrey.hk

2017-09-11
Version 0.2

Status	Type	Short Name
Deferred	Standards Track	NOT_YET_ASSIGNED

This specification provides a means for transporting messages from the Raft consensus algorithm over XMPP.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction **1**
- 1.1 Why <message/> and not <iq/> . . . 1
- 1.2 Example Usecase . . . 2

2 Requirements **3**

3 Glossary **3**

4 Protocol **3**
- 4.1 RequestVote . . . 4
- 4.2 AppendEntries . . . 4

5 Security Considerations **5**
- 5.1 Checking cluster membership . . . 5

6 IANA Considerations **5**

7 XMPP Registrar Considerations **6**
- 7.1 Protocol Namespaces . . . 6

8 XML Schema **6**

1 Introduction

This XEP does not attempt to implement Raft. Rather, based on the message exchanges defined in the Raft paper, it defines a complementary set of <message/> stanza elements to allow Raft messages to be transported cleanly over XMPP. The messages for Node to Node communication are well documented in the Raft paper and are reproduced here with minor additions to leverage the benefits of XMPP. However Client to Node communication is only hinted at and, as it does not use Raft natively it is outside the scope of this XEP.
Raft is a consensus algorithm that is easy to understand and implement. When you want to keep a distributed system consistent, you will need some form of consensus algorithm. Raft was developed at Stanford University as an alternative to the incumbent consensus algorithm PAXOS. PAXOS is claimed to be very complex, hard to teach and even harder to implement. All of these are discussed in a paper here and additional information (including a graphical simulation of a cluster) can be found on the project's website: `https://raftconsensus.github.io/`.
Raft defines a set of core messages needed to implement the protocol. However, it does not specify the transport layer for this protocol. Most implementations have chosen to use vanilla TCP due to its simplicity. However if you want to run a cluster over the Internet, you are likely going to want more than what vanilla TCP provides. This is where XMPP would really shine as a transport layer for Raft. XMPP offers:

1. Encrypted transport (TLS)
2. Authenticated Endpoints
3. Ability to use JIDs to identify cluster nodes
4. Re-use XMPP if the application is already XMPP enabled

These are all things that would traditionally have to be re-engineered each time somebody wanted to use Raft across the public Internet. By supporting Raft in XMPP, developers looking to use Raft would have a transport layer that's as easy to use and understand as the Raft protocol itself. As Raft does not offer its own transport protocol and has deliberately left that to the developer, there is no conflict in standardizing an XMPP based transport layer.

1.1 Why <message/> and not <iq/>

The Raft algorithm can be categorized as a request-response protocol. Normally this would make it a prime candidate for using <iq/> stanzas to handle the communication. However because Raft is designed to cope with message loss, it intrinsically supports automatic recovery. There is no need for the transport layer to report errors as even if the transport layer provided them (such as an <iq/> 'error' response), the Raft implementation cannot use it.
This has a number of benefits. First, it makes Raft adaptable to lossy transport layers where packets can (and do) get lost. Raft is able to automatically recover in this scenario because the next message the Leader sends will allow a Follower to detect that it has missed a message

and ask for it to be sent again. The Leader has no way to deal with an error condition caused by sending a message to a Follower.
Second, when it comes to implementing Raft over XMPP, using <message/> instead of <iq/> greatly simplifies the implementation. As <iq/> stanzas require a reply, the implementation would need to handle detecting and reporting errors conditions back to the sender. This could mean adding arbitrary timers to try to determine if a Follower has 'timed out'. This adds complexity and uncertainty to the system, and given that Raft itself cannot make use of this information, using <iq/> does not add any value to the Raft over XMPP protocol.

1.2 Example Usecase

Making databases or datastores available over the Internet has been the norm for many years. Databases containing PGP keys, certificates or other information can be found hosted by many different organizations. The problem with these systems is that as they become more critical to users, the impact of a server failing increases dramatically. For example, a server that provides a spam database that clients can verify email against, must be operational for those clients to be able to filter spam. Having a single server in this scenario is not acceptable; to provide redundancy there must be multiple servers.
The problem then becomes how to keep the servers in sync. Raft partly solves this problem by providing the means to ensure the cluster maintains consensus i.e. maintains a consistent view of the data. As mentioned previously, it does not however provide a means for the nodes in the cluster to actually communicate with each other or clients.
This is where being able to use Raft over XMPP would be highly beneficial. As it stands now, developers must implement their own transport and security, which although certainly possible, is not ideal. First, most developers are not security experts and a wealth of knowledge and experience is needed to properly design a secure system. Second, even with the required expertise, it takes a considerable amount of time and effort to actually implement and test any new implementation. Third, this extra work takes developers' focus away from the problem that they were trying to solve in the first place.
So how could Raft over XMPP be applied in this instance? First, XMPP has an excellent history when it comes to security. Considerable time and effort was spent ensuring that XMPP was secure when XMPP Core was being standardized. By using XMPP, this hard work and battle tested approach can be leveraged by Raft. This means developers do not need to concern themselves with securing Raft messages, rather they now only need to concern themselves with using XMPP appropriately.
Raft over XMPP further simplifies things by allowing developers to think at a higher level of abstraction. Nodes in the cluster can be communicated with simply by knowing their JID. It would not be necessary to know a node's IP address (which could change) or what TCP port the node is running on. In addition developers would not need to worry about which node connects to which, managing multiple TCP sockets and how to multiplex data across them.
Lastly, integrating a Raft implementation with Raft over XMPP, would be relatively straight forward as Raft over XMPP defines and uses the same names as those provided by the Raft paper with few additions. This means that it could be much easier to get a Raft implementation

up and running using Raft over XMPP than it would be to do so even with pure vanilla sockets.

2 Requirements

The author has designed Raft over XMPP with the following requirements in mind:

1. The protocol needs to support all messages as defined in the Raft paper
2. The protocol ought to leverage the benefits of using XMPP as the transport layer
3. Client to Node interaction is out of scope for this XEP

3 Glossary

Raft A distributed consensus algorithm designed at Stanford University to be simple and easy to implement. It aims to replace PAXOS as the conensus algorithm of choice for real world use and teaching. The Raft website is https://raftconsensus.github.io/

Log Replication Log replication is how Raft exchanges commands with the rest of the cluster.

Follower The default state of a cluster member. It receives and applies updates from the Leader

Candidate When a Follower has not seen a heartbeat from the Leader for a period of time, it will assume the leader has failed and will look to become the Leader itself.

Leader The Leader of the cluster is responsible for making all changes to the log and sending them to the other members of the cluster

VoteRequest A VoteRequest message is sent by a Candidate in order to solicit votes to become the Leader of a cluster

AppendEntries An AppendEntries message is sent by a Leader to other nodes in the cluster when it has updates that it needs to replicate.

4 Protocol

This XEP defines a transport layer for Raft and not an actual implementation. That is, it does not seek to implement the Raft consensus algorithm within XMPP, but instead to simply define the means for Raft messages to be transported over XMPP. To facilitate this, both the message name used in the Raft spec (shown in camel case) and the corresponding element name are mentioned together where appropriate.
Node to Node communication is the back-bone of a Raft cluster. In operation, only the Leader or a Candidate will send messages. In all other cases, nodes will only reply to messages

received. The two messages are AppendEntries and RequestVote.

4.1 RequestVote

When a Follower has not received a heartbeat from the Leader for a given period of time, it will determine that the Leader has failed and will seek to replace it. To do this it needs the support of the majority of nodes in the cluster. It can solicit support from other nodes by declaring itself a Candidate and sending a 'request-vote' (RequestVote) message to all nodes in the cluster:

Listing 1: Duncan is soliciting votes to become leader of the cluster

```
<message from="duncan@inverness.lit/castle" to="macbeth@cawdor.lit/
    castle">
      <request-vote xmlns="urn:xmpp:raft" term="1" last-log-term="1"
          last-log-index="1" cluster="scotland"/>
    </message>
```

A node will respond with a 'vote' (RequestVoteResponse) message:

Listing 2: Macbeth votes for Duncan to become the next leader

```
<message from="macbeth@cawdor.lit/castle" to="duncan@inverness.lit/
    castle">
      <vote xmlns="urn:xmpp:raft" term="1" vote-granted="true" cluster
          ="scotland"/>
    </message>
```

A node can either vote for a given Candidate (vote-granted="true") or against a Candidate (vote-granted="false").
If a node does not receive a reply, no special handling is required.

4.2 AppendEntries

The 'append' (AppendEntries) message is used by the Leader to tell Followers that they should append a new entry (or entries) to their logs. It contains additional information to allow a Follower to determine which log entries have been executed and committed on the Leader and also if it has dropped any messages. These features are implemented in Raft directly.

Listing 3: Duncan sends an append message to his followers

```
<message from="duncan@inverness.lit/castle" to="macbeth@cawdor.lit/
    castle">
    <append xmlns="urn:xmpp:raft" term="1" prev-log-index="1" leader-
        commit="1" cluster="scotland">
      <entry xmlns="urn:xmpp:raft" encoded="false">
```

```
        SET X = 1
      </entry>
      <entry xmlns="urn:xmpp:raft" encoded="true">
        U0VUIFggPSAx
      </entry>
    </append>
  </message>
```

The AppendEntries message is described as a simple array in the Raft paper and this has been expanded on in XMPP to take advantage of structured XML. In addition, Raft is designed to be able to replicate any form of command and this could be binary data rather than textual data. To accommodate this, an attribute has been added to the 'append-entries' element to allow a sender to flag when the receiver needs to decode the Entry before passing it to the Raft implementation. The data is encoded using base64.
When followers receive this message, they send a single 'append-response' (AppendEntries-Response) in reply as follows:

Listing 4: Macbeth sends an append-response message to Duncan

```
<message from="macbeth@cawdor.lit/castle" to="duncan@inverness.lit/
    castle">
    <append-response xmlns="urn:xmpp:raft" term="1" success="true"
        cluster="scotland"/>
  </message>
```

As before, if a message is missed in either direction, the transport layer does not need to take action.

5 Security Considerations

5.1 Checking cluster membership

It is not the responsibility of the transport layer to determine whether a node is a member of a cluster or not before delivering messages to the Raft implementation. The Raft implementation should ignore messages that it receives from nodes that aren't part of the cluster.

6 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA).

7 XMPP Registrar Considerations

7.1 Protocol Namespaces

The XMPP Registrar [1] includes 'urn:xmpp:raft' in its registry of protocol namespaces.

8 XML Schema

REQUIRED for protocol specifications.

[1] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.