ASD Project Phase 1

Emanuele Vivoli*[1], Joao Miguel Costa†[2] and Pedro Guilherme Silva‡[2]

[1]Department of Informatics Engineering, University of Florence, IT
[2]Department of Informatics Engineering, NOVA FCT, PT

October 2019

Contents

*57284 - emanuele.vivoli@stud.unifi.it
†50171 - jmp.costa@campus.fct.unl.pt
‡50390 - pg.silva@campus.fct.unl.pt

i Introduction

This the the first phase report for the project "publish/subscribe system based on Topics" assigned in the class "Algorithm and Distributed Systems" 2019/ 2020 [Leitao, 2019a].

A distributed system is based on the idea of decoupling (of space and time) where lots of different processes cooperate without sharing memory (everyone has its own memory not accessible from others) or timers (they are not synchronized one each other and they don't need to be online at the same time). In this publish system the decoupling of both memory and timer is obtained between the message generators and the message consumers.

This first phase has the goal of decoupling memory, where each processes has the own data structure to memorize the message that flows in broadcast on the network and we need to build the first prototype of the system and manage the recovery phase (when one node is temporary disconnected and lost some messages). One important thing: for this phase the system is still time-driven (all connections are based on synchronized protocols like TCP).

The project will use the Babel framework and its functionality is described in the documentation [Leitao et al., 2019].

ii Publish/Subscribe System

In topic-based publish/subscribe systems, each node can do three actions: **subscribe** and **unsubscribe** to a topic and **publish** a message with one or more topics. Messages that flow in the network are delivered only from processes that have been subscribed to that or those topics. A process can unsubscribe a topic to inform the system that it is not more interested in receiving message about that topic. For this phase each node will store and manage its own subscriptions system that will work as a filter for that specific node. In fact each message flows in broadcast to every node, and the node deliver the message only if it is subscribed to that topic.

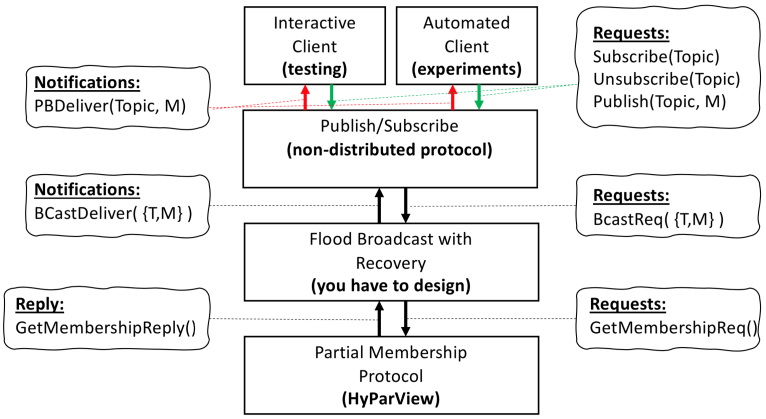

Figure 1: System architecture and description of inter layer messages

1 Architecture

The architecture is composed by 4 layers that provide API function to the upper-layers in the system. The Figure 1 [Leitao, 2019b] shows an overview of the architecture.

The client (Interactive or Automated) defines a message with the correspondent topic, with the intent to send that message to every node in the network. The message goes to the Publish/Subscribe layer that in this case just forward the message to the layer below. Than the Broadcast layer get the message and ask to the HyParView layer to give its neighbours in order to send them the message [Leitao et al., 2007]. At this point the HyParView answer with the active view and the Broadcast layer send a message to every host in the list. When an host receive the message it forward the message to the upper layer (Publish/Subscribe layer) and, in case the host is subscribed to that specific topic, it will forward the message to the Client layer that will show the new message.

This is a simple explanation of the workflow. In the following Sections we will describe the layer in the opposite direction of the message flow: from the bottom layer to the upper layer.

2 HyParView

HyParView [Leitao et al., 2007] has the role of building and maintaining the structure of the network by managing the active and passive views for each node in the *unstructured overlay network* [Leitao, 2019a].

2.1 Messages flow

It is composed by two different type of messages: **Intra-Layer** and **Inter-Layer**.

2.1.1 Intra-Layer messages

The messages that flow only between HyParView layer of different processes are:

- JOIN, message that is sent when a new-node join the network (it must has a hook: "ip-port" of a node to connect with).
- FORWARD-JOIN, message that allows the flooding of the JOIN information through the network until it reach the maximum number of steps and some node must connect to the new-node.
- NEIGHBOUR request, message that is sent to a node to notify that the sender want to add the receiver to its active view.

- NEIGHBOUR reply, message sent to the sender of the NEIGHBOUR request, that notify if the request is accepted or refused. In case of accepted both will add one each other the node to their active view.
- DISCONNECT, message sent to notify that the sender has removed the receiver from its active view. The receiver can remove the sender from its active view, but without notify.
- SHUFFLE request, message that is composed from some information about active and passive view, used to refresh the passive view of the node.
- SHUFFLE reply, message that is composed from some information about active and passive view, sent to the node who made the SHUFFLE request. After both the node will update the passive view based on the algorithm.

2.1.2 Inter-Layer messages

As shown in Figure 1 the HyParView layer provide to the Broadcast layer two functionalities that allow to "request" and "reply" a list of neighbours which indicates the nodes, in the network, which one specific node is linked with. These two functionalities work with two type of messages (respectively ProtocolRequest and ProtocolReply) defined as:

- PEERNEIGHBOURS request, it's a message that is administrate by the HyParView and permit to create a GetPeerNeighboursReply and answer with that to the broadcast layer.
- PEERNEIGHBOURS reply, it's a message that is sent to the broadcast layer; it contains the active view of the node.

3 Broadcast

The Broadcast layer is responsible for sending to the peers nodes the messages that come from the Publish/Subscribe layer.

3.1 Messages flow

The messages, also in this case, can be defined as **Intra** and **Inter Layer**.

3.1.1 Intra-Layer messages

The messages that flows in Broadcast layer of different processes are:

- BCAST message, it's the message that contains the payload and the topic attribute. It is message the client wants everyone to receive.
- CACHEDIDS request, it's a message sent to all the neighbours with a periodic timer; it contains a list of all the (stored) message ids that the node have delivered. It is used for the recovery policy.
- CACHEDIDS reply, it's a message generate as answer to a CACHEDIDS request, it contains the set of message ids received with CACHEDIDS request but subtract to this set the message id that the node has already delivered (and stored).

3.1.2 Inter-Layer messages

As shown in Figure 1 the Broadcast layer provide to the Publish/Subscribe layer two functionalities that allow to "send" and "deliver" a message. We can see also that the broadcast layer can use two functionality provided by the HyParView layer such "request" and "reply" the list of neighbours which the node is linked with. These two functionalities work with different type of messages:

- BCAST request, received from the upper layer when the host want to send in broadcast a brand new message.

- PEERNEIGHBOURS request, generated from the broadcast to the HyParView layer asking the list of neighbours for sending the broadcast message to.
- PEERNEIGHBOURS reply, it's a message that is sent from the HyParView layer; it contains the active view of the node.
- BROADCASTDELIVER, it's a message generate for the upper layer when the Broadcast receive a message for the first time, and it decide to deliver it. It is sent to the Publish/Subscribe layer and will be its role to accept or delete it.

3.2 Properties

Broadcast with Recovery [Leitao, 2019a]

BwR1 (Validity): If a correct process i broadcasts message m, then i eventually delivers the message.

BwR2 (No Duplication): No message is delivered more than once.

BwR3](No Creation): If a correct process j delivers a message m, then m was broadcast to j by some process i.

BwR4 (Agreement): If a message m is delivered by some correct process i, them m is eventually delivered by every correct process j.

3.2.1 BwR1 (Validity)

The validity property is guarantee from the usage of TCP connection in Best Effort Broadcast, to spread messages between nodes.

3.2.2 BwR2 (No Duplication)

To avoid the delivered of duplication messages we use a list variable called *delivered* that store all the IDs of the messages already delivered. With that list we are able to check if the message has been already delivered or not and than we proceed delivering the message or ignore the message without any action.

3.2.3 BwR3 (No Creation)

The No Creation property is obtained from the TCP protocol, that guarantees the integrity of the message without the possibility of compromising the message payload.

3.2.4 BwR4 (Agreement)

This property is guaranteed from the recovery phase of cache share procedure. It allows every nodes to periodically share the ids of the recently cached messages with these neighbours. Once the $node_j$ received a share message msg_{share} from $node_i$, it will confront the ids from msg_{share} with his cached ids and send a list with the missing ids back. For every message ids the $node_j$ received, it will send to $node_i$ a broadcast message.

3.3 Pseudocode

Interface :
  Requests :
    BCastRequest(m)
  Indications :
    BCastDeliver(m)

upon init() do
    $delivered \leftarrow \{\}$
    $pending \leftarrow \{\}$
    $cache \leftarrow \{\}$

upon event $BCastRequest(m)$ **do**
    $mid \leftarrow$ generateUniqueID();
    peerList $\leftarrow$ getPeer(funout, null);
    for each $p \in$ peerList **do**
        trigger Send(BCastMessage, p, mid, m, $timestamp$, $myself$);
    end for
    trigger $DELIVER(m)$;
    delivered $\leftarrow$ delivered $\cup\{mid\}$
    cached $\leftarrow$ cached $\cup\{(mid, m)\}$

upon event $ReceiveBCast$(BCastMessage, mid, m, $timestamp$, $sender$) **do**
    if $mid \in$ delivered **then**
        peerList $\leftarrow$ getPeer(funout, null);
        for each $p \in$ peerList **do**
            trigger Send(BCastMessage, p, mid, m, $timestamp$, $myself$);
        end for
        trigger $DELIVER(m)$;
        $delivered \leftarrow delivered \cup \{mid\}$
        $cached \leftarrow cached \cup \{(mid, m)\}$
    end if

upon timer $shareCache$ **do**
    peerList $\leftarrow$ getPeer(funout, null);
    for each $p \in$ peerList **do**
        trigger Send(CachedIdsMessage, p);
    end for

upon timer $refreshCache$ **do**
    for each $(mid, m) \in cached$ **do**
        $ttl \leftarrow getMessageTimestamp(mid)$
        $ttl \leftarrow ttl - 1$
        if $ttl == 0$ **then**
            $cached \leftarrow cached/\{mid, m\}$
        end if
    end for

upon event $ReceiveCachedIdRequest$($cachedIds$, $sender$) **do**

$missingMsgs \leftarrow \{\}$
for each $mid \notin cachedIdsReceived$ **do**
$missingMsg \leftarrow missingMsg \cup \{mid\}$
end for
if $\#missingMsgs > 0$ **then**
trigger Send(CachedIdsReply, $sender$, $missingMsgs$);
end if

upon event $ReceiveCachedIdReply(\ missingMsgs, sender)$ **do**
for each $mid \in missingMsgs$ **do**
$m \leftarrow getMessageFromCache(mid)$
trigger Send(BCAstMessage, $sender$, $msgId$, m, $myself$);
end for

3.4 Recovery

To ensure the reliability even in cases where some node might fail the BroadCast layer was designed with a recovery mechanism inside. The approach is very similar to a lazy push strategy, the motivation was to reduce the size of the messages that flow in the network, preventing big messages to circulate in the network if there is no necessity for it. The process consists of periodic interaction between adjacent node and is divided in three phases:

3.4.1 Send Cache Share Request

In this initial step every node must send a copy of ids in cache to every node in the active. This action is made periodically by a timer that can be configured by the user in a configuration file. The TTL of messages in cache can also be parametrised.

3.4.2 Receive Cache Share Request and send reply

Upon receiving a cache share message the node must confront this cached messages id's with the ones in the message. By doing this to every id received the node can create an list of all missing messages and send the list back to the node who send the share cache message.

3.4.3 Receive Cache Share Reply and send BCastMessage

Upon receiving a list with the missing message the node must send the message with the given id to sender of the message. Note that by sending this message as a BCastMessages it forces the receiver to broadcast the message thous making the recovery process a little bit fast.

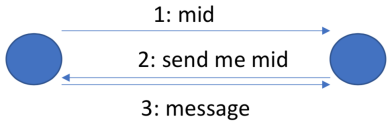

Figure 2: Recovery Mechanism [Leitao, 2019a]

4 Publish/Subscribe

The Publish/Subscribe module is used as a simple filter layer between the client (that want to receive only the messages that belongs to topic it is interested to) and the Broadcast layer that receive and forward to

the upper layer every message it delivered (it receive every message broadcasted, but deliver each message just once).

4.1 Messages flow

In this context the type of messages that can flow are only of type **Inter Layer**, because this layer doesn't allow any node to send message (of any type) directly to other node without passing through the lower layers.

4.1.1 Inter-Layer messages

This messages go from Client to Broadcast and vice-versa. The type of messages are:

- SUBSCRIBE request, it is a message that comes from the client when one wants to subscribe to a topic because of some interest.
- UNSUBSCRIBE request, it is a message that comes from the client when one is not longer interested in a topic, the action is to unsubscribe to that topic.
- PUBLISH request, it is a message that contains the payload, the topic and has to be sent to every process in the network.
- BCAST request, it is a message that is sent from the Publish/Subscribe layer to the Broadcast layer and ask for sending the new message to every node in the network.
- BROADCASTDELIVER, it is a message that comes from the Broadcast layer and that notify the Publish/Subscribe layer that a new message has been delivered for the first time.
- SUBSCRIBEMESSAGE notification, it is the message that is forwarded to the client in order to be shown as message of interest because it belongs to a topic the client is subscribed to.

4.2 Properties

The Publish Subscribe layer does not support any kind of distribution in this initial phase of the project, working simply as a filter of all messages that are delivered by the BroadCast layer. The messages are filtered according to the topics the client is subscribed to, delivering to the client only the messages whose topic is in the client subscription set.

The user can subscribe or unsubscribe a specific topic that is uniquely identify by a string. There is also the possibility of sending a message (array of bytes) in a specific topic that will be delivered to all process and then submitted to a filter process in the Pub/Sub layer.

5 Client

This layer is the responsible to create the message to publish, to subscribe or unsubscribe to a topic and show to the interface the message that it will receive from the Publish/Subscribe layer (that are, by construction, message the user is interested to receive).

5.1 Messages flow

5.1.1 Inter-Layer messages

The type of message that a Client layer (automated or manual) can manage are:

- SUBSCRIBE request, it is a message that contains the topic the user is interested to, and it goes to the next layer.
- UNSUBSCRIBE request, it is a message that with the topic the client is not longer interested in, it goes to the lower layer.

- PUBLISH request, it is a message that contains the payload, the topic and has to be sent to every process in the network.
- SUBSCRIBEMESSAGE notification, it is the message that is forwarded from the Publish/Subscribe layer in order to be shown to the client because it belongs to a topic the client is subscribed to.

5.1.2 Manual Client Commands

In this phase two different clients were created, one automated one design for tests and collection of statistics and a manual one one for debug and demonstration purposes.

The Manual client offers the following commands:

- sub <topic> - subscribe to a given topic
- unsub <topic> - unsubscribe to a given topic
- pub <topic> <message> - send message to a given topic

6 Experimental Results

In order to evaluate the correctness of all the developed protocols we conducted some tests using the developed Automated Client, which runs random commands and can be configured to publish a specific number of messages. The experiments were mainly aimed at measuring the following stats: reliability without failures; total number of received broadcast messages by all the nodes; convergence time (time it takes all the messages to reach every node); total number of HyParView messages sent by all the nodes; reliability with failures.

6.1 Experimental Setting

The experiments were conducted using the same protocol configurations. In the Broadcast Protocol both the Cache Refresh Rate and the Cache Share Rate were set to 5 seconds and the Message TTL was set to 7 (messages stay in the cache for 7 refreshes). In HyParView we set the active view size to 5 and the passive view size to 30. Active Random Walk Length was set to 6 and the Passive Random Walk Length was set to 3. For the shuffle we used a Shuffle Rate of 5 seconds and a Shuffle TTL of 5. Each shuffle message had at most 3 elements from the active view and at most 4 elements from the passive view.

6.2 Reliability without failures

The first set of experiments were aimed at testing the protocols on a environment free from failures to check the correctness of the protocols. In all the following three tests the measured Reliability was 100% - all the nodes in the network received all the messages.

6.2.1 Total Broadcast messages received

This test was conducted in a network of 20 nodes with each node generating 10,25,50 or 75 publish messages. Figure 3 shows the relation between the total number of broadcast messages received by all nodes and the number of messages generated by each node.

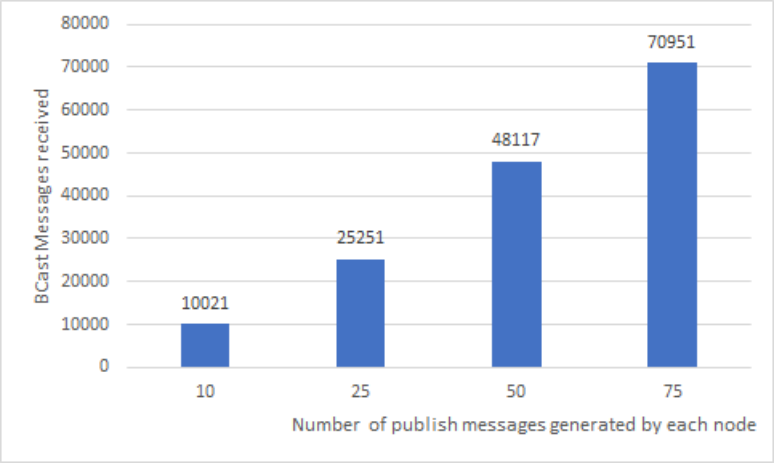

Figure 3: Total number of Broadcast messages received by all nodes

6.2.2 Convergence time

The goal of this test was to measure how much time it takes for all the messages to reach every node. It was conducted in a network of 20 nodes with each node generating 10, 25, 50 or 75 publish messages.Figure 4 shows the relation between the convergence time and the number of messages generated by each node.

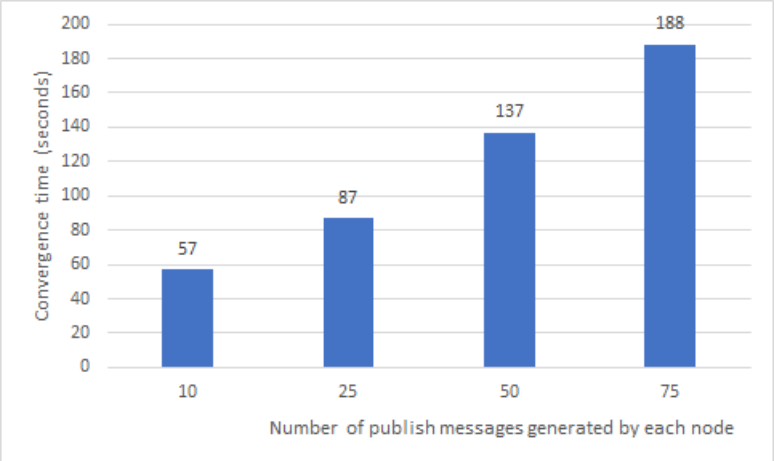

Figure 4: Convergence time

6.2.3 Total HyParView messages sent

The goal of this test was to show the relation between the number of nodes in the network and the total number of HyParView messages sent by all node. It was conducted in networks of 10/20/30 nodes with each node generating 20 publish messages. Figure 5 shows the result graph of this test.

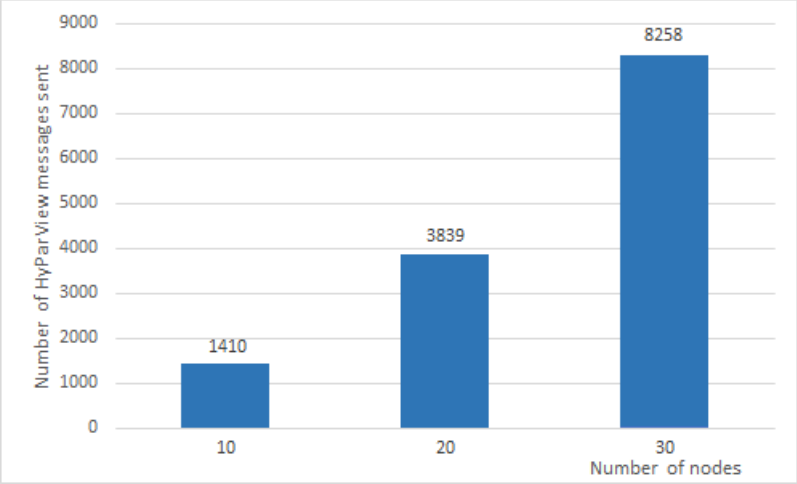

Figure 5: Total number of HyParView messages sent by all nodes

6.3 Reliability with failures

The last set of experiments were aimed at testing the impact of failures in the system behaviour. The main goals were to measure the reliability of the protocols with failures and to check the correctness of the Recovery process (Broadcast).

The tests were conducted in a network of 20 nodes, each node generating 20 publish messages, and inducing failures of 10/25/50/75/90% (2/5/10/15/18 nodes failing). In each test we made all nodes join the network, generate some number of messages and then we induced some failure percentage. We then measured the reliability to check if all correct active nodes received all the intended messages. In all the experiments we obtained a Reliability of 100%, as shown in figure 6.

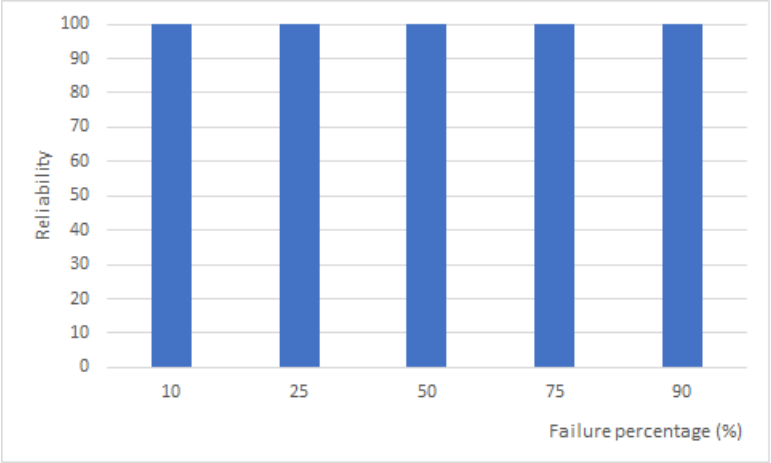

Figure 6: Reliability with failures

6.4 Discussion

The results of the experiments showed good overall measures of the developed solution. One of the main goals of these tests was to check the level of reliability of the solution in environments with failures and without failures, and we obtained good results - every node received all the published messages by other nodes.

The tests were conducted in small networks (mainly constituted of 20 nodes) and with small published messages due to hardware restrictions. However, we think that the reliability of our solution in networks with more nodes and bigger messages would be equally good.

7 Conclusion

The goals for this first phase of the project were achieved: design and implementation of a simple publish subscribe system protocol; design and implementation of a Broadcast protocol with Recovery mechanisms; implementation of the Hyparview protocol. The results obtained from the experimental tests showed a good measure of correctness of the implemented system.

References

[Leitao, 2019a] Leitao, J. (2019a). Algorithms and distributed systems. *MIEI - Integrated Master in Computer Science and Informatics, University Lectures.*

[Leitao, 2019b] Leitao, J. (2019b). Algoritmos e sistemas distribuidos - project phase 1. *NOVA Laboratory for Computer Science and Informatics (NOVA LINCS) and Departamento de Informatica (NOVA FCT).*

[Leitao et al., 2019] Leitao, J., Fouto, P., and Costa, P. A. (2019). Babel: Internal framework developed at nova lincs. *http://asc.di.fct.unl.pt/ jleitao/babel/docs/* , October - 2019.

[Leitao et al., 2007] Leitao, J., Pereira, J., and Rodriguess, L. (June 2007). Hyparview: A membership protocol for reliable gossip-based broadcast. *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, pages 419–429.