Distributed Computing, Peer-to-Peer and GRIDS Final Report

Martin Borek, Miguel Gordo

June 8, 2016

1 Introduction

This document presents the analysis of the final project for Distributed Computing, Peer-to-Peer and GRIDS course. The document is divided in four sections, following the structure presented in the project specification. In each section we will present the solution and statistics relevant to the task at hand.

2 Task 1 - News Flood

The effect of changing the TTL can be clearly seen in Figure 1, where increasing the TTL is directly correlated with a higher percentage of news received.

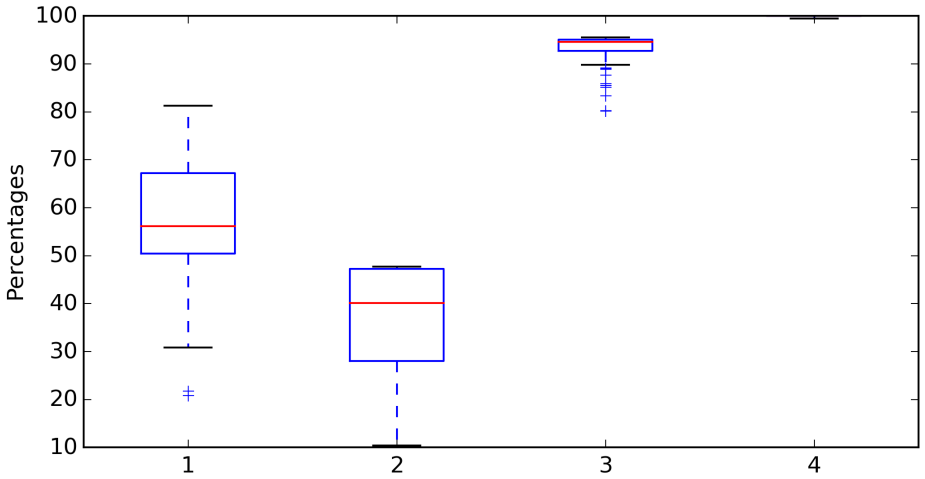

Figure 1: Boxplot for different TTLs. 1 represents TTL 5, 2 represents TTL 2, 3 TTL 20 and 4 represents TTL 20 with leader (version for task 3.1). Y axis represents percentage of news received

For each the node we can see which percentage of news they received, for example, with TTL 2 in Figure 2.

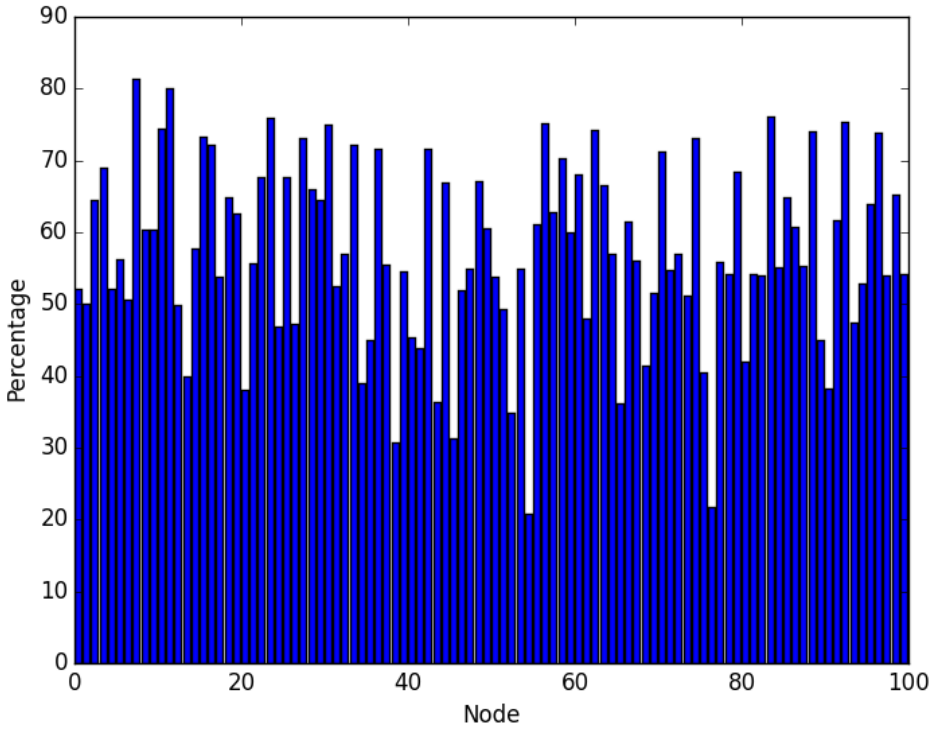

Figure 2: Percentage of news received per node for TTL 2

For each of the news we can also observe which nodes received it in Figure 3:

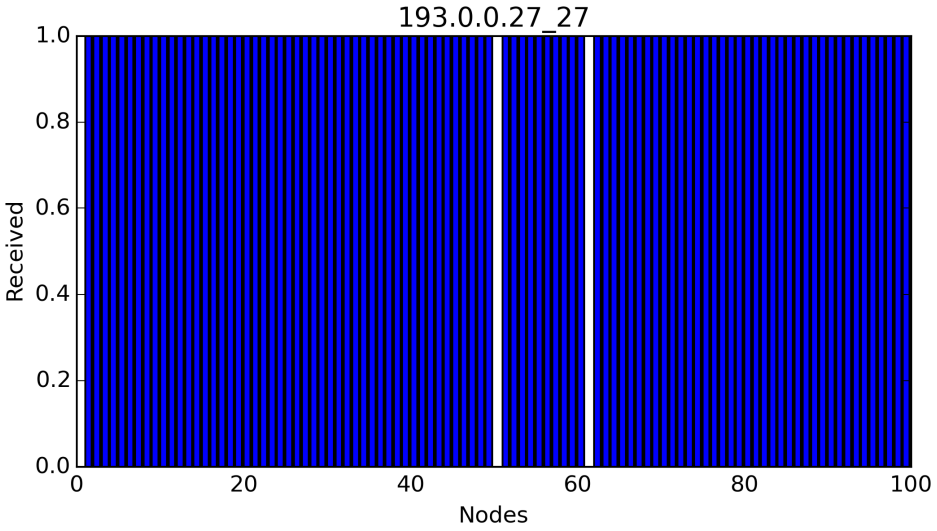

Figure 3: This Figure shows which nodes got the 27th message of node 27 with flooding

From the results we can clearly see that increasing the TTL drastically improves the news coverage in the network. This is shown in individual node coverage, and news propagation.
Each node generates a message every 12 seconds, starting after one minute of the simulation starting. Each node receives around 4900 news messages, and each node replicates a message blindly when it receives it for the first time. This means that the total number of messages is exponential. In particular, if each message is forwarded to 10 random Croupier neighbors, TTL is greater than 10 and we have 100 nodes, the number of messages generated will be $10 * 100$ since messages, once seen, are not forwarded any longer.
A final image shows total percentage of known news per node:

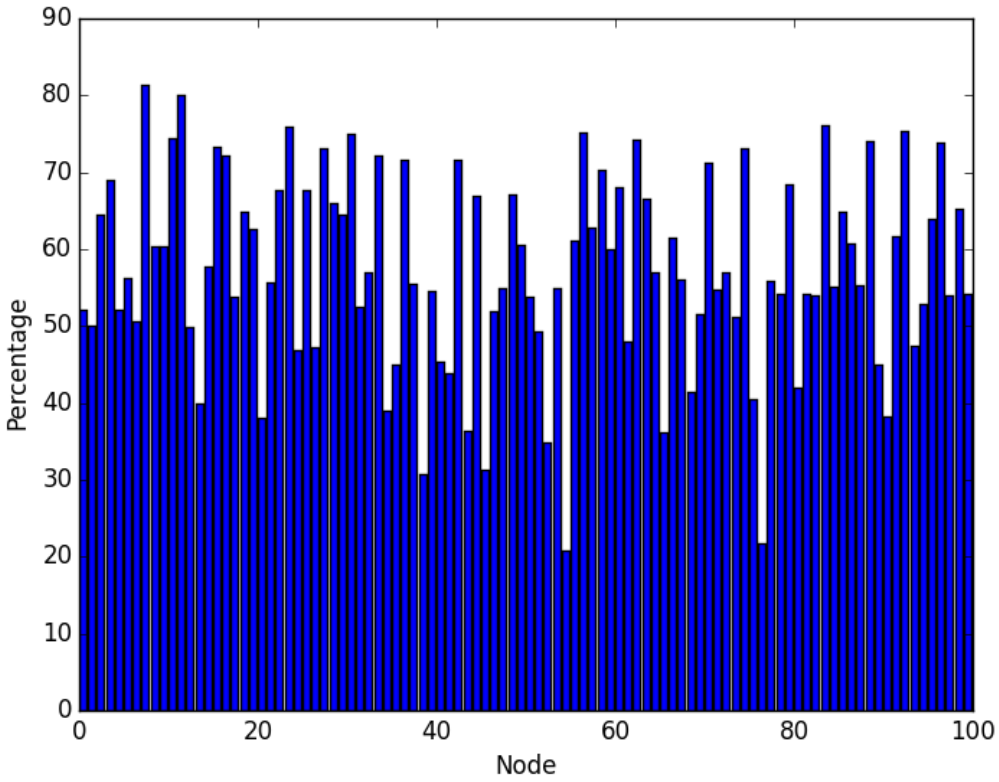

Figure 4: Percentage of news known by each peer with news flooding and TTL 2

3 Task 2 - Leader Selection

As for statistics, our algorithm shows the first leader gets elected 26 seconds into the simulation. Since each round of Gradient Updates is two seconds long, this means the algorithm takes 13 rounds to converge. The total number of messages required are 10 *AmILeader* messages, to inform the gradient neighbors of the decision of the proposed leader, plus 10 *AmILeader* responses, and then flooding to inform all the network of the decision. That is 10*2 messages that get flooded over the network. That is, in total, 20 messages per peer: 20*100. In total: 20*100+10+10=220 messages.
The leader selection algorithm consists of nodes comparing their value with their Gradient neighbours. If a node sees its value to be the highest for a number of Gradient cycles (5) and their Gradient neighbours do not change for this period, the leader initiates the Leader selection process; it asks its neighbours *AmILeader*. These neighbours respond positively in *AmILeader*-

Response if none of their neighbours has a value higher than the initiating note. If the initiating node receives agreement from all its neighbours, it claims itself a leader and starts the dissemination.

4 Task 3.1 - Leader Dissemination

For the leader dissemination we have used a simple push approach. To be able to use push with Gradient, all nodes notify their Gradient neighbours who save their addresses. This way a reverse Gradient topology is created and used for Leader and News dissemination. Each node remembers 10 freshest reverse neighbours (nodes that notified it most recently). Each leader dissemination message includes an ID. This ID is increased with each update and is used for recognizing if the update has already been processed by a node or if it should be forwarded to its reverse neighbours. Otherwise messages would be replicated eternally. This is necessary for nodes joining later to get the information about the current leader because we are using the push model.

The statistics show that nodes receive the update at the same time it is pushed from the leader, therefore they all learn it in one round (due to kompics adding no delays in message sending). The first leader gets elected in 26 seconds (13 rounds), which is the maximum time they spend without a Leader Update. Afterwards, leader pushes are sent every 30 seconds. The average time between updates is 30014 ms, and the maximum time between updates is 31073 ms

5 Task 3.2 - News Dissemination

News are disseminated the same way as the leader update. Push model is used. When a node wants to send a message, it sends it to the leader. The leader pushes it to its reverse Gradient neigbours, those forward it to their reverse Gradient neighbours and so on. Since a reverse Gradient topology is used, once a leader receives a message, it forwards it to its reverse neighbours. These are nodes that are close the center of Gradient. These forward it to their reverse neighbours and the message is gradually disseminated from the center to the edge of the Gradient topology.

In terms of node knowledge, all nodes know all messages according to our scripts:

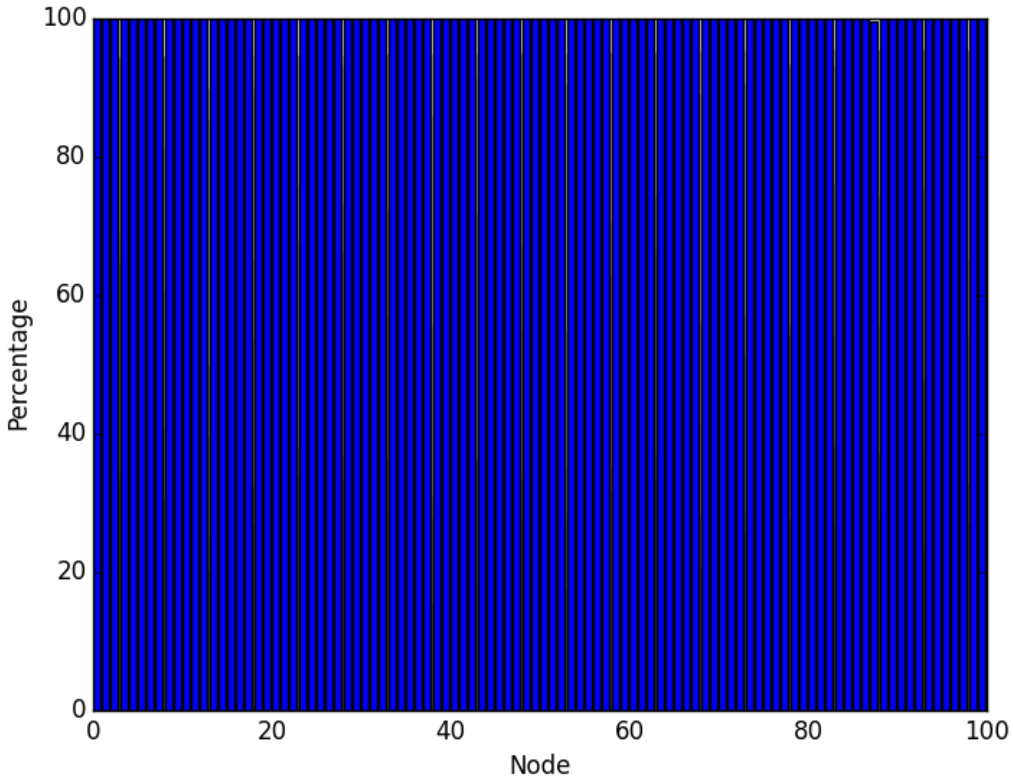

Figure 5: Percentages of news known to each node with leader, and TTL 20

In terms of message dissemination on a particular basis, all messages get seen by all nodes.

6 Task 4.1 - Leader Failure

Since we are using a push model, the new leader gets selected automatically. Nodes have a timer to detect if the leader disconnects to stop sending it messages. Once a new leader is selected, it pushes the Leader update to all nodes. From logs, it is apparent that the new leader gets correctly selected and all nodes get to know this.
The time for a new leader to start the selection process is 2 minutes. Then it takes another 10 second for the new leader to be selected and all nodes being informed.

7 Task 4.2 - News

Since we are using a push model, we focused on simulation new nodes joining the system. When a node joins a system, it waits for an information (Push) about a leader. Once it knows a leader, it asks it for all messages the leader knows. Subsequently, the leader sends to the node all those messages. From attached log it is visible that all new joined nodes receive all messages that were sent earlier.

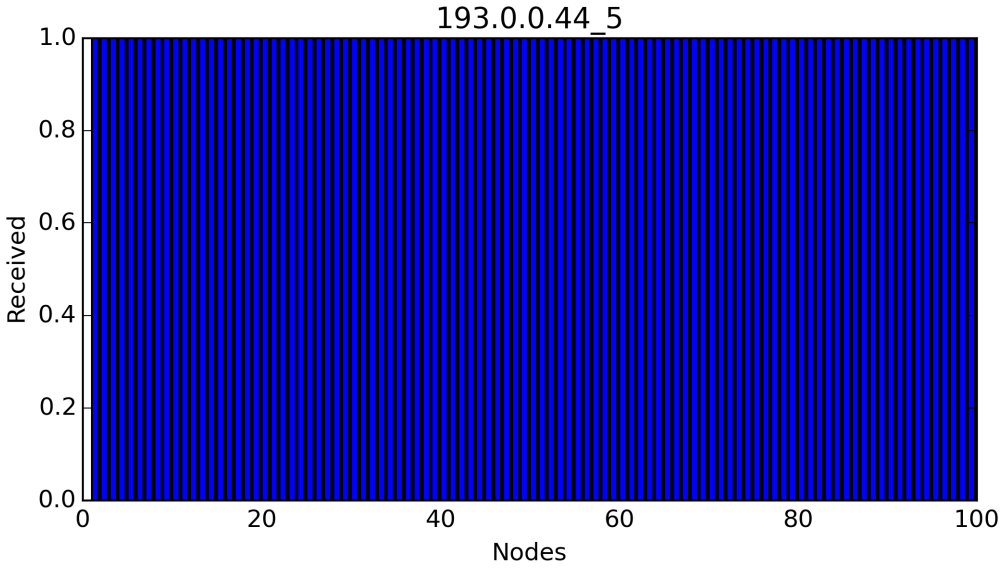

Figure 6: News dissemination for node 44, message 5, with leader and pull mechanism for task 4.2

We also present percentage of news known by each node at the end of the simulation, which is for all nodes 100%

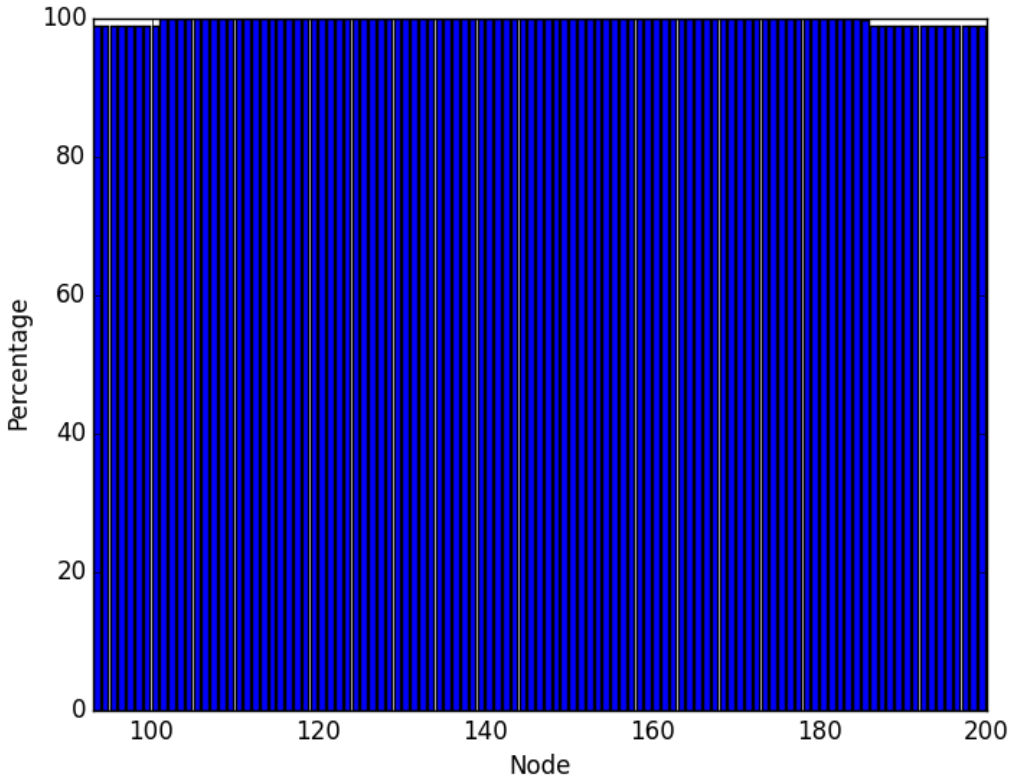

Figure 7: Percentage of known messages for nodes that are alive at the end of the simulation for Task 4.2

A Simulation

To run simulations, use source codes submitted with the report. Four scenarios are available (News flooding, Reliable dissemination, Leader failure and Nodes joining later).