Rollercoaster: a Reproduction Study

Naqib Zarin
Delft University of Technology
Delft, The Netherlands
n.zarin@student.tudelft.nl

1 INTRODUCTION

In this work, some of the results from Huygenroth *et al.* [1] are reproduced and evaluated. More precisely, the group-multicast scheme *Rollercoaster* for mix models is tested by means of simulations with the aim of reproducing and evaluating Figure 5 and Figure 6 from the original paper. Furthermore, we extend their research by investigating how various parameters influence the performance if not set fixed. This paper is chosen because it relates to content of the lecture on anonymity approaches (week 2). In this part of the lecture, mix models are discussed.

2 ORIGINAL STUDY

This section elaborates on the original study conducted by Hugenboth *et al.* The topic is introduced and motivated. Additionally, the main contributions are discussed.

Topic and motivation

Anonymity is nowadays an important requirement for Internet applications in various domains. Those applications rely on the network layer to transfer messages, which can reveal sensitive metadata such as the sender and the receiver of a message. Therefore, scholars have focused on how to provide strong metadata privacy in communication networks.

This metadata privacy can be provided by so-called mix networks. In a nutshell, mix networks hide the relationship between sender and receiver of a message by encrypting the message in multiple layers and send it in multiple hops. An adversary will have a hard time correlating incoming and outgoing messages because the incoming messages in a hop are mixed with other messages. This metadata privacy protection generally comes at the cost of network performance as messages are delayed for reordering. This calls for solutions that have lower latency whilst upholding the strong metadata privacy guarantees.

One recent solution to protect metadata privacy in a relatively efficient way is called Loopix[2]. Loopix allows one-to-one communication in a novel mix network architecture. Hugenroth *et al.* uses Loopix as the underlying mix network, and extend it by providing an efficient way to allow one-to-many and group communication. This group-multicast scheme is called *Rollercoaster* and can be used for collaborative applications where group messaging takes place (e.g., WhatsApp and Google Drive). One key difference between Loopix and *Rollercoaster* lies in how messages are disseminated. Loopix uses a unicast approach, whereas *Rollercoaster* leverages a group multicast protocol.

Main contributions

Hugenboth *et al.* make multiple contribution to human knowledge. First, they provide an overview of the current state of research on group multicasting. This is useful as it allows other academics to know what already has been researched and what the limitations of these studies are. Second, they show that distribution trees can also be used in mix networks to realize efficient message dissemination. This connection of the two research areas might invite experts from both areas to collaborate and propose even better algorithms in mix network. Third, the authors fill the research gap on metadata privacy in the context of group communication by providing a concrete and reproducible scheme that achieves low latency while preserving metadata privacy. They show that their approach yields better results in terms of performance than naive alternatives and account for realistic scenarios such as nodes not being reachable in the network. This is important as there are a lot of group collaboration application that require strong metadata privacy. Finally, Hugenboth *et al.* provide an open-source network simulator for *Rollercoaster*. This is important as it allows both academia and industry to reproduce their work. It also enables them to test potential improvements of the *Rollercoaster* scheme.

3 ASSIGNMENT A: REPRODUCTION STUDY

In this section, the reproduction study is explained in more detail.

Scope

Figure 5 and Figure 6 of the original paper will be reproduced in a slightly different simulation setting. This part of their work is chosen for the reproduction study because it demonstrates the key contribution of the paper: the efficiency of their novel multicast scheme for mix network. This paper would not make any sense without the proof in Figure 5 and Figure 6 that it actually achieves better performance. It

accounts for two scenarios: i) all users of a group are online (Chapter 6.2), and ii) some users of the group are offline (Chapter 6.3). Other scenarios where users are part of multiple groups (Chapter 6.4) and further improvements of the multicast algorithm (Chapter 6.5) are considered out of the scope.

Approach

The authors explain that the real network performance of Loopix has extensively been tested in the Loopix paper. This lead to the decision to use a deterministic simulator instead. The original papers contains a reference to the Github[1] repository where the open-source simulator can be found. In this work, the same simulator is used to reproduce Figure 5 and Figure 6. The documentations of the simulator were very clear and helped speed up this process. Since the aim of the present work is to test whether similar results can be produced, alterations are not be made to the network composition and node behavior described in Chapter 6.

Set-up simulator. The set-up of the simulator for this reproduction study is as follows. First, Docker is installed on the computer which runs the simulation. The computer has 8 cores and runs on Linux. After a successful download, the *Rollercoast* repository is cloned from Github and a Docker image is created. Please note that this happens within a Python environment (e.g., CPython or PyPy). The Docker image is created by the running a predefined script with the command *./scripts/docker_01_build.sh*.

Create simulations. The next step is to create and run the simulations. In order to start Jupyter Notebook within the Docker container, port 8888 needs to be available to localhost. This is can be done by running a predefined script with the command *./scripts/docker_03_jupyter_notebook.sh*. The console will give a link to access Jupyter Notebook in the browser. In order to create simulations, the map *create_simulations* needs to be entered in the Jupyter Notebook. The original paper ran a simulation with the simulation timespan of 24h. In this work, a shorter simulation timespan is chosen due to lack of resources. This is done by changing the *SIM_TIME_MS* to 1 * 3600 * 1000 in the second cell of the notebook. Next, new configuration files need to be created. This can be done by clicking on *Restart and run all* button. Finally, the simulation will start by running the script *docker_04_parallelrunner.sh*. The present simulation took up to 4 hours with the above-mentioned computer. Note that this set-up handles all the possible scenarios and produces results for all the figures in the original paper. An attempt was made to only run scenarios related to Figure 5 and Figure 6, but this took more code investigation time than simply than running it for 4 hours.

Create visualizations. In order create insights, the simulator contains scripts to create visualizations. This first requires the simulation results to be converted to Numpy Array format for efficiency purposes and can be done by running the script *docker_05_process_results.sh*. In order to create the same figures as Figure 5 and Figure 6 in the original paper, a new Jupyter Notebook needs to be started first. This can be done by running the script *docker_06_jupyter_notebook.sh*. Next, the map *create_all_graphs* needs to be entered in Jupyter in order to execute the visualization code (click *Restart and run all*). Once all the above-mentioned steps are followed correctly, visualizations of the simulation results can be found in the folder *output*.

Results and evaluation

As mentioned before, this reproduction study aims to reproduce Figure 5 and Figure 6 in the original paper. In this section, both reproduced figures are shown and discussed.

Scenario where all nodes are reachable. Figure 5 in the original paper shows the simulation results for *Rollercoaster* and Loopix for the scenario where all the nodes are reachable (online). The reproduced results can be found in Figure 1. Figure 1 should be interpreted as follows. The message latency is measured on the application layer as the difference in time (in seconds) between the moment a message enters the payload queue of the sender, and the moment the message is delivered to the recipients. The blue boxes describe the performance for the naive solution based on the sequential unicast technique in Loopix. The green boxes are results for the *Rollercoaster* multicast technique with different values for the parameters k and p. The parameter k stands for how many children each parent node has in the distribution tree. This means that each node sends at most k messages to next layer of the tree. The parameter p refers to the maximum number of messages inside a novel message package format called *MultiShpinx* (Chapter 5.4). Since this is not part of the standard *Rollercoaster* algorithm, this reproduction work solely focuses on the results for $p=1$. Therefore, only the light green boxes are considered.

The following key observations can be made:

(1) As the group sizes grow, the latencies for the sequential unicast approach grows sub-linearly with the size of the group.
(2) The *Rollercoaster* approach results in latencies that are much lower than the sequential unicast approach.
(3) With the Rollercoaster approach, the latency does not grow sub-linearly with the group sizes.

[1] https://github.com/lambdapioneer/rollercoaster

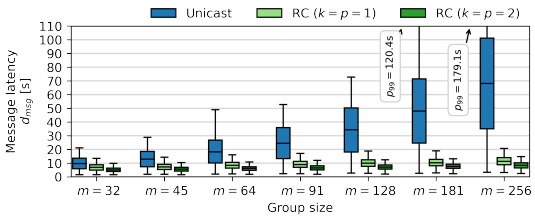

Figure 1: Reproduced results for the scenario where all the nodes in the network all reachable (corresponding with Figure 5 of the original paper).

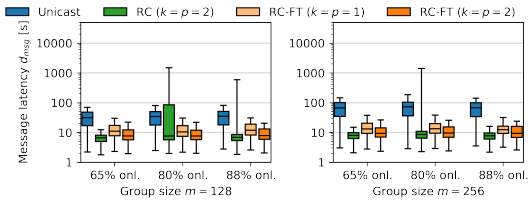

Figure 2: Reproduced results for the scenario where only some of the nodes in the network are reachable (corresponding with Figure 6 of the original paper).

The first observation can be explained as follows. In the sequential unicast approach, each message in the queue is sent individually to the recipients. This means that last message in the queue will have to wait for n-2, where n denotes the total number of nodes in the network, messages to be sent before it can leave the queue. Therefore, the message delay grow sub-linearly with the size of the groups.

The second observation can be explained as follows. After a sender sends a message to the first recipient in the first simulation round, this recipient helps with the message dissemination by forwarding it to another node in the next round. This means that in the second simulation round, four nodes will have received the message instead of two. Since it is assumed that every node can be reached in the network, the set of nodes that receives the message double in every round. This explains why the *Rollercoaster* outperforms the naive sequential unicast approach for all different group sizes.

This also explains our third observation since it implies that the total number of rounds the dissemination process needs is logarithmic in the group size. Hence it is not sub-linear.

Scenario where some nodes are reachable. Figure 6 in the original paper shows the simulation results for *Rollercoaster* and Loopix for the scenario where only some nodes are reachable. The reproduced results can be found in Figure 2. Figure 2 should be interpreted as follows. Each boxplot shows three groups of scenarios with different ratios between online and offline time. The blue bar shows the naive sequential unicast algorithm. The green bar shows *Rollercoaster* version without fault tolerance. The light orange bar shows the results of the *Rollercoaster* version with fault tolerance. The dark orange boxplot refers to the improved version of *Rollercoaster* (Chapter 5.4) and is considered out of scope. Also note that the y-axis (latency) in this figure is logarithmic scale in contrast to the y-axis in Figure 1.

The following key observations can be made:

(1) The standard *Rollercoaster* approach often performs worse than the naive sequential unicast approach for all online/offline ratios.
(2) The standard *Rollercoaster* approach has a latency distribution with wide bounds. This means that very large latencies occur more often.
(3) The fault tolerance version of *Rollercoaster* performs better that the naive unicast approach and the standard *Rollercoaster* without fault tolerance for all online/offline ratios.

The first observation can be explained as follows. The hierarchical nature of the distribution tree implies that online children nodes cannot receive messages from their offline parent node. With sequential unicasting, the sender will continue sending messages from its queue, even if some recipients are offline. This is why sequential unicast will often outperform the *Rollercoaster* approach.

The second observation can be explained as follows. The latency heavily relies on which nodes in the distribution tree are unreachable. Generally speaking, if a node is located high is the distribution tree, it means that this node is the root node of a large subtree. If a root node becomes unreachable, the entire subtree cannot receive the message until this node returns online. On the contrary, if a node is located in the lower levels of the distribution tree, it contributes less to the overall latency because its subtree is has a smaller size.

The third observation can be explained as follows. The *Rollercoaster* version with fault tolerance uses acknowledgment messages and time-outs to detect unreachable nodes. If the sender does not get an acknowledge message back from another node before the time-out, it knows that this node is unreachable. The sender then re-sends the failed message to a another nodes. This is repeated until sender node has successfully sent the message to a new reachable node. Furthermore, it gives this new node special instructions that it should act as the unreachable node the sender tried to send the message. This way, children nodes from the unreachable node do not have to wait for the unreachable node to return online in order to receive the message. This approach outperforms the sequential unicast approach as reachable nodes

can still forward the messages in each round and therefore speed up the message dissemination process.

Comparison of results

In this section, a comparison is made between the original results and the reproduced results. As mentioned before, the simulator in this work ran for a shorter period of time and consequently missed some node behavior and message traffic scenarios. This has impact on the results and leads to interesting insights. Since the simulator is deterministic, a 24 hour simulation would produce exactly the same results as in the paper. The six key observations from Figure 1 and Figure 2 in section 3 are compared against Figure 5 and Figure 6 respectively from the original paper.

Scenario where all nodes are reachable. The first observation from Figure 1 aligns with the observation that can be made from Figure 5 in the original paper. The relation between latency and size of the group for sequential unicast approach does not depend on how many rounds the simulation runs. Also note that the 50th percentile of both figures for all scenarios are almost identical. Additionally, the 99th percentile also does not differ much for group sizes up to 128. This indicates that 1 hour simulation for this part is sufficient in future work.

The second observation from Figure 1 also holds for Figure 5 in the original paper. This is straightforward.

The third observation from Figure 1 can also be made for Figure 5. This mathematical property does not rely much on the number of simulations as long as it exceeds a certain minimum threshold.

Scenario where some nodes are reachable. The first observation from Figure 2 does not hold for Figure 6 in the original paper. The standard Rollercoaster approach without fault tolerance seems to perform better than the sequential unicast algorithm. This can be explained as follows. Many scheduled events in the simulation model are not executed because of the decreased runtime. Some of the 'missed' events include events where nodes return online and participate in the message forwarding algorithm. This means that some nodes might may have remained offline during the entire simulation. The overall delay will increase once they expose themselves and are considered as part of the network.

The second observation from Figure 2 does not hold for Figure 6 in the original paper. The same explanation can be given for this mismatch as for the first observation.

The third observation from Figure 2 partly holds for Figure 6 in the original paper. The fault-tolerance version performs better than the naive sequential unicast approach, but worse than the standard *Rollercoaster*. The later is also in line with the previous two observations.

4 ASSIGNMENT B: EXTENTED STUDY

In this section, the extension study is explained in more detail.

Extension

In the previous sections, Figures 5 and 6 of the original paper were reproduced and discussed in more depth. In the following sections, we extend the research by varying parameters k and p. The aim of this extented study is to gain more insight in the influence of these two parameters on the performance of *Rollercoaster*.

Preliminaries. Parameter k stands for the branching factor in each level. More precisely, all parent nodes have k children in the distribution tree. and in each level the total number of users that have received the messages is increased by a factor of $k + 1$. This explains why the total number of levels in the distributed tree is bounded by $\log_{k+1} |U|$, where U is the total number of users that participate in a group communication. Furthermore, Hugenroth *et al.* argue that the queuing delay at each hop in the distribution tree is

$$\frac{k-1}{\lambda pay}$$

where λpay is the rate parameter for sending a payload message as a Poisson process.

Parameter p stands for the multiplication factor in the mix nodes. By allowing some mix nodes, called multiplication nodes and present in layer 2 of the mix network, to multiply one input message into p output messages, the total number of messages sent in the network can be decreased. It works as follows. A number of p messages is grouped into one message using the MultiSphinx package format. As a result, every outgoing message is p times the usual size. When the multiplication nodes receive this message, they mix the p separate messages independently and pad them before forwarding it. This strategy enhances the performance as sending fewer but p times larger messages allows for lower power consumption on mobile devices.

The authors have kept parameters k and p equal. They do not motivate this decision and only mention that it is for simplicity purposes. They have conducted experiments for k=p=1 and k=p=2. The present study aims to gain more insights in how the two parameters influence the performance *separately*. That is, we vary one of the parameters k and p and keep the other fixed. Note that due to the scope of this assignment the parameters' influence on anonymity aspects is disregarded.

Extension 1: Varying parameter k. It is worth investigating how the pure *Rollercoaster* algorithm performs without its extension *p-restricted multicast*. Some applications might prefer *Rollercoaster* without the multiplication factor. For example,

the size of a social media chat group (e.g. Whatsapp) is often small. Large p values might mirror the naive multicast approach where the multiplication nodes learn the group size (see Chapter 4 in the original paper). Therefore, we keep p constant at $p=1$ because this means that messages are not multiplied in mix nodes.

In this work, we create simulations for $k=4$, $k=8$ and $k=16$. Note that this series is determined by the quadratic function. In theory, it should decrease the amount of levels in the distribution tree and consequently decrease the overall delay for larger k values according to formula 6 in the original paper. One goal of the present study is to test whether this is truly the case and whether it is different for online and offline scenarios. Therefore, the first research question is as follows: *Does significantly increasing parameter k decrease the latency in different group sizes in both online and offline scenarios?*

Extension 2: Varying parameter p. It is also worth investigating how the multiplication nodes influence the performance of *Rollercoaster*. Hugenroth *et al.* have only discussed how p influences the anonymity in the system. This study aims to fill this research gap.

Simulations are created for $p=3$, $p=4$ and $p=5$ while keeping k constant at $k=4$. This value for k is chosen because it allows to compare the produced results with the results from extension 1. Note that we have values for p that are lower than, equal to and larger than k. Since multiplication nodes await p messages, the *p-restricted* strategy could decrease the overall latency. However, other nodes than multiplication nodes might gain performance. This is because p wrapped message have the same position in the sending queue and therefore are less affected by the sending delay parameter. Nodes that were originally scheduled later will be sent out earlier. In order to get more insight in this aspect of the *p-restricted* strategy, the following research question will be answered: *Does increasing parameter p decrease the latency in different group sizes in both online and offline scenarios?*

Approach

Figures 5 and 6 of the original paper need to be created for different p and k values in order to answer the two research questions. Therefore, the same decisions have been made as in section 3 regarding other parameters, group sizes, online-offline ratio's, delaying rates etc. They are kept the same as in chapter 6.1 of the original study. However, some changes have been made to the code to allow changing parameters p and k. Because the code modification process requires a lot of explanation, the decision has been made to create a new repository with only the necessary code. Now, in total 122 independent simulations are created instead of 306.

Set-up simulator. The computer specifications are the same as in section 3. First, the new repository[2] has to be cloned from Github. Also, Docker needs to be installed. Then a Docker image can be created by running the command $./scripts/docker\_01\_build.sh$. In this section, we only explain how to run the simulations for research question 1 as the procedure is exactly the same for research question 2.

Create simulations. In order to create simulations port 8888 needs to be available to locate host for Jupyter Notebook (JN). This can be one by running the script $./scripts/docker\_03\_jupyter\_notebook.sh$. Console will output a link to access the JN in the browser. After accessing JN, six different notebook files can be seen. Two files are for creating simulations and visualizations of the original study, two for research question 1 (varying k) and two for research question 2 (varying p).

The map $create\_simulations - vary - k$ needs to entered in JN. Note that there is less code than in the original map $create\_simulations$. Code that reproduced other figures than Figure 5 and 6 have been deleted. This means than less simulations were created, bringing the runtime back from 4 hours to 1 hours. In order to created the configuration pickle files, all kernels should be restarted and run. Note that older simulation files might still be present in the input or pickles folder as they will not be overwritten. They should be deleted. Finally, the simulation can be started by running the script *docker_04_parallelrunner.sh*. This scripts used the pickles files as input and creates simulations.

Create visualizations. When the simulation finished, the produced results need to be pre-processed before visualizations can be generated. Therefore, the output files are converted to Numpy Array format by running the script *docker_05_process_results.sh*. Then, a new JP needs to be started by running the script $./scripts/docker\_06\_jupyter\_notebook.sh$. In the new JN, the $create\_allgraphs - vary - k$ folder needs to be entered in order to execute the visualization code. Again, the kernel should be restarted and ran. Once all the steps are followed correctly, visualizations of the simulation results can be found in the folder *output*.

Results and evaluation

As mentioned before, this extension study also aims to produce Figure 5 and Figure 6 in the original paper, but for different k and p values. In this section, both produced figures are shown and discussed.

Varying k. The results for different k values can be found in Figure 3 and Figure 4. The interpretation of these figures is discussed in detail in section 3 of the Reproduction study.

[2] https://github.com/nzarin/rollercoaster-tudelft

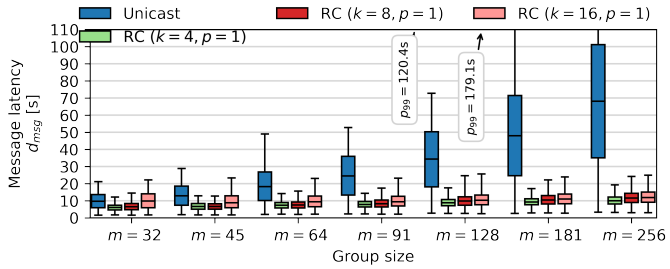

Figure 3: Results for different k values for the scenario where all the nodes in the network all reachable (corresponding with Figure 5 of the original paper).

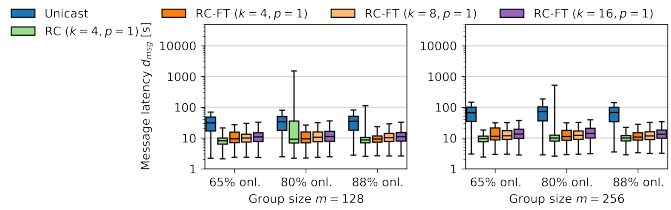

Figure 4: Results for different k values for the scenario where some the nodes in the network are reachable (corresponding with Figure 6 of the original paper).

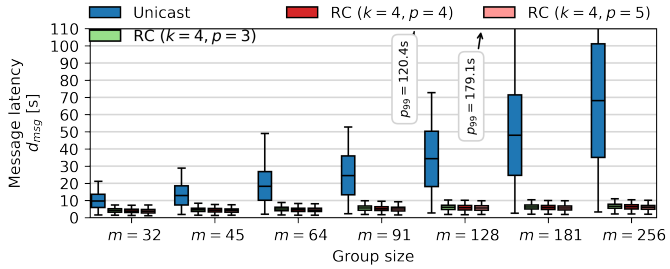

Figure 5: Results for different p values for the scenario where all the nodes in the network all reachable (corresponding with Figure 5 of the original paper).

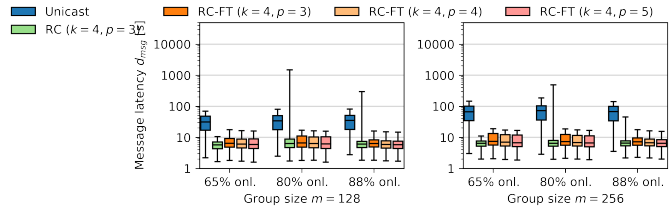

Figure 6: Results for different p values the scenario where some the nodes in the network are reachable (corresponding with Figure 6 of the original paper).

The following key observations can be made with respect to the first research question :

(1) The latency increases for larger k values for larger group sizes when all nodes are online.
(2) The latency increases for larger k values in the same group sizes when all nodes are online.
(3) The latency slightly increases for larger k values when some portion of the nodes is online.
(4) The latency slightly increases for larger k values if a larger portion of the nodes is online.

The first observation can be explained as follows. Larger k values means that nodes forward the messages to a larger group in the network. This cuts down the number of levels in the distribution tree when all nodes are online and consequently decreases latency. However, messages are queued in incoming queues and outgoing queues. Apparently, outgoing queues outweight the impact on the sublevel delay than having less subtrees. This means that in general, the message delay d_{msg} influences the latency more than $\log_{k+1} |U|$ in formula 6 of the original paper. Furthermore, it follows from the formula that if $|U|$ increase, then the delay does too.

The second observation has a similar explanation as the first observation. Even if $|U|$ is kept the same, higher k values increase the delay.

The third observation is interesting as it implies that the naive multicast approach in the original paper would not have decreased the delay in the current set-up. Having more children in the distribution tree increases the portion of leaf nodes and this would increase the chance that the offline node is a leaf node. This should result in lower latency for higher k values. But this appears not to be the case. One possible explanation is that the queue delay factor is still dominant. However, this observation requires a more in-depth study in the future.

The fourth observation is interesting too. If the portion of offline users increases it means that the overall delay is increased because the root node only detects this after the ACK deadline is expired. Thus, a node in the next level picks up the work, meaning that the required levels increase. If k increases, the detection of an offline node should be sooner because the distribution tree is more flat. This limits the latency. So, it can expected that the delay decreases if the offline ratio is smaller and the k value is larger. This contradicts our observation. One possible explanation is the queue delay factor is still dominant. This observation also requires in depth analysis in future work.

In summary, the delay does not decrease for larger k values. The queue delay factor outweights the tree level factor from a performance perspective. Therefore, the answer to our first research question is negative.

Varying p. The results for different p values can be found in Figure 5 and Figure 6. The interpretation of the figure is the same as in the previous section.

The following key observations can be made with respect to the second research question:

(1) The latency decreases for higher p values in the same group sizes when all nodes are online.
(2) The latency slightly increases for higher p values in larger groups sizes when all nodes are online.
(3) The latency slightly decreases for higher p values when some portion of the nodes is online.
(4) The latency slightly decreases for higher p values when a larger portion of the nodes is online.

The first observation can be explained as follows. Since messages are always scheduled earlier if p increases, they encounter less often the sending delays. The previous sections have showed that the delay rate plays a large role in the latency of *Rollercoaster*. Therefore, it is not surprising that the latency decreases when more messages are combined together and sent at once. Since the multiplier nodes are in a single layer, their influence on the performance is limited.

The second observation is interesting, but not surprising. If the size of the network increases, then the distribution tree becomes deeper and more messages should be sent. Therefore, the delay of the last node that has received the message increases. However, we see that this additional delay is very limited. The results are very similar and often remain under the target of 10s. It would be interesting to find out at what point the group sizes significantly increase the delay.

The third observation can be explained as follows. If not all users are online, higher p values still decrease the queue sizes and thereby decrease the latency. However, it could be the case that one node has to wait longer to receive the message if its parent is offline. Therefore, the delay distribution should be more spread than in the scenario when all nodes are online. This can be verified by comparing Figure 5 and 6. But the ACK mechanism limits the delay. Therefore, the trend is the same when p is increased when all nodes are the online and when some nodes are online.

The fourth observation is related to the third observation. The larger the portion of online nodes is, the shorter a forwarding node has to wait for enough messages to be wrapped. Also, it is worth noticing that the chances that messages have to wait is larger when more nodes are offline for larger p values because they have to wait for more messages. This cannot be verified easily in the figures and requires future study.

In summary, the delay decreases for larger p values. Less queuing delays outweights the waiting for enough messages from a performance perspective. Therefore, the answer to our second research question is positive.

Conclusion

This study has assessed the influence of parameters k and p on the performance of *Rollercoaster* independently.

The simulations show that the incoming and outgoing queue delays heavily determine the performance of the network. Making the distribution tree more flat (fewer levels) does not reduce the delay in the network. Increasing k is not a good strategy from a performance perspective. Therefore, the answer to our first research question is negative.

The parameter p has a positive influence on the performance of the network. This is because it decreases the queuing delays that were limiting the influence of increasing parameter k. As a result, the answer to the second research question is positive.

REFERENCES

[1] Daniel Hugenroth, Martin Kleppmann, and Alastair R. Beresford. 2021. Rollercoaster: An Efficient Group-Multicast Scheme for Mix Networks. In *30th USENIX Security Symposium (USENIX Security 21)*. USENIX Association, 3433–3450. https://www.usenix.org/conference/usenixsecurity21/presentation/hugenroth

[2] Ania M. Piotrowska, Jamie Hayes, Tariq Elahi, Sebastian Meiser, and George Danezis. 2017. The Loopix Anonymity System. In *26th USENIX Security Symposium (USENIX Security 17)*. USENIX Association, Vancouver, BC, 1199–1216. https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/piotrowska