Fair Gossiping

Fabian Kaelin
Communication Systems
Bachelor, 6th Semester
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Fabian.Kaelin@epfl.ch

Supervised by:
Maxime Monod
Distributed Programming Laboratory (LPD)
School of Computer & Communication Sciences
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Maxime.Monod@epfl.ch

***Abstract*—The goal of this semester project was to implement and test a fair gossip protocol. The proposed approach is a push-based algorithm with a mechanism that adapts the workload according to the benefit received. At the same time the algorithm should guarantee reliability and stability. In order to be able to efficiently test the protocol, the Peersim framework was used for the implementation.**

```
 1: Initialization:
 2:    delivered ← {}
 3:    events ← {}
 4: upon TIMER($t$ time units) at process $p_i$ do
 5:    Neighbors ← SELECTPARTICIPANTS($F$)
 6:    gossip-msg.events ← SELECTEVENTS($N$ in events)
 7:    for all $p \in$ Neighbors do
 8:       P2PSEND($p$, gossip-msg)
 9:    end for
10: end upon
11: upon RECEIVE(gossip-msg) do
12:    for all $e \in$ gossip-msg.events do
13:       if $e \notin$ delivered then
14:          events ← events ∪ {$e$}
15:          if ISINTERESTED($e$) then
16:             DELIVER($e$)
17:             delivered ← delivered ∪ {$e$}
18:          end if
19:       end if
20:    end for
21: end upon
```

Fig. 1. A basic push gossip-dissemination algorithm

I. Introduction

Up to now, fairness was not a main concern when designing gossip protocols. The algorithm presented in Figure 1 represents a classic push-based gossip algorithm at first glance. But there is one notable difference, namely lines 15 to 17. The algorithm disposes over a method that determines if a received message is interesting to the peer or not. The basic idea which makes the algorithm proposed in this project *fairer* is the following: after a predefined number of rounds, what we call a phase hereafter, the peer decides according to his benefit whether it wants to decrease, or has to increase, its own workload. If one peer profits more from the system, e.g. receives more messages of interest than another peer, the former peer's share of the total workload should also be bigger. The main challenge is that the total workload should stay constant, otherwise the reliability of the network cannot be guaranteed. In other words, if one peer decreases its workload, other peers have to compensate the 'missing' workload. In the next section, the algorithm will be explained in more detail.

II. Fair Gossip

Now we present a generic version of a *fair gossip* algorithm for selective event dissemination based on a push-based and cycle-driven gossip algorithm as shown in Figure 1. Our fair gossip algorithm (Fig. 2, p.3) is composed of three procedures:

- An *application procedure* which is triggered at the beginning of every round.
- An *adaptation procedure* which is triggered at the beginning of every adaption phase.
- A *receive procedure* which is triggered whenever the peer receives a gossip message.

Basically, the application procedure is the one that gossips information and, most importantly, changes the workload of the peer. The adaptation procedure determines the benefit of the peer in the last phase and decides how the peer should change its workload. The receive procedure handles the incoming message and delivers the event if deemed to be interesting to the peer.

A. Definitions

Definition 2.1 (Fanout): Each peer disseminates its messages to a number of peers, hereafter we refer to this value as the **fanout**. In the proposed fair algorithm, this fanout is variable.

Definition 2.2 (Phase): A **phase** is a predefined number of rounds. After each phase an adaption phase is started, and from the end of the adpaption phase until the end of the phase, the algorithm can be considered as *fair*.

Definition 2.3 (Adaption Phase): The length of the **adaption phase** is composed of the defined R rounds plus R' rounds needed to compensate all $Redistributions$ that couldn't be applied in round R.

Definition 2.4 (Distribution): When a peer has changed its own fanout F during an adaption phase, other peers need to compensate this change by adapting their own fanout F. This operation is called **Distribution**.

Definition 2.5 (Redistribution): Every peer has to try to apply the $Distributions$ it has received from other peers during the round before. If a peer is not able to do this, e.g. exceeds its fanout limits while trying to apply this value, the peer has to pass this compensation on to other peers. This operation is called **Redistribution**.

Definition 2.6 (Compensation): The **Compensation** value of peer is composed from the $Distribution$ value and the $Redistribution$ value. It is the final compensation which will be passed on to other peers.

B. Problem Statement or Challenges

Challenge 1: To guarantee the reliability and stability of the network, the **global fanout** needs to be **constant** outside of an adaption phase ($\sum_i F_{p_i} = F_{global} = F_{global}^{optimal} = \sum_i F_{init}$).

Challenge 2: By allowing single peers to change their fanout during an apaption phase, we accept small flucuations of the $global fanout$ ($\sum_i F_{p_i} \simeq F_{global}^{optimal}$) that could have an effect on the reliability and stability of the network. **Minimizing the average change of fanout** in one round minimizes those fluctuations, thus increases the reliability, and at the same time minimizes the number of $Redistributions$, thus increases the stability.

Challenge 3: At the end of an adaption phase the network can be considered as *fair* and it should stay *fair* for as long as possible. Therefore, the **length of the total adaption phase** $R_{total} = R + R'$ **should be minimized**.

C. Generic Algorithm

The algorithm proposed in Figure 2 is a generic version of a push-based and fair gossip protocol. All decisions a peer makes are local. The main challenges resulting from fact that the peers only send information but never request any is deciding on how to calculate the changes in fanout for the different peers and how to bring other peers to compensate those changes.

1) **application procedure**: At the beginning of every round the peer is forced to try to apply the compensations it has received from other peers. This is mandatory because receiving a compensation means that another peer has changed its fanout and this change needs to be compensated to keep the global fanout F_{global} constant. If an adaption phase is in progress, then the peer tries to change its own fanout according to the benefit. Finally the peer calculates the compensation it has to pass on to the other peers and sends this compensation, embedded into a normal $gossip.message$.
2) **adaptation procedure**: At the beginning of a new adaption phase, all peers calculate the benefit from the last phase. Then, based on the benefit and the fanout, they determine by how much they have to decrease or increase their fanout. Finally, they distribute this value over the length of the adaption phase to make the adaption phase as *smooth* as possible.
3) **receive procedure**: All the $gossip.messages$ received contain a compensation value and the actual content of the messages, called events. The peer extracts the compensation value which it will then try to apply in the next application procedure. Finally the peer verifies if the events that the message contained are interesting or not. If this is the case, it delivers the event (e.g. to another application) and increments the variable that counts the number of interesting events received.

Some explanations to the generic algorithm in Figure 2:

- Line 9,16,17 : A peer first tries to compensate the β it has accumulated during the last round. If this fails, e.g. β is not zero after the applying it, there are two scenarios. Ouside an adaption phase, the remaining β has to be redistributed. During an adpation phase, there is a second chance to compensate the β, that is when the application of α_r enables the peer to further compensate its β (line 18).
- Line 20-24 : During an adaption phase, when a peer changes its fanout F, the question is at what point to affect the new fanout F. There are two possiblities, either before or after sending the $gossip-msg$. In order to minimize the average compensation value ($compensate/F$), we maximize the number of peers to send the message to. So in case the new fanout is smaller than the old fanout, we affect the fanout after the message has been sent and vice versa.
- Line 18 : The final compensation value $compensate = -\alpha_r + \beta$ is the sum of what the peer needs to distribute after applying an α_r to his fanout F, and what the peer needs to redistribute, e.g. the rest of the β that couldn't by applied. The different signs result from the fact that α_r needs to be distributed and β needs to redistributed. Let us say a peer has $\alpha_r = -3$ and $\beta = 2$ at line 18. This peer needs to distribute a value of $-\alpha_r = 3$ (because it changed its fanout F by -3) and to redistribute a value of $\beta = 2$ (because it couldn't change its fanout F by 2) among the other peers. In this case we can conclude that $compensate = 5$, the new fanout is $F = F_{max}$ and that $\beta = 5 + (F_{max} - F_{old})$ at the beginning of the application procedure.
- Line 12,19 : As soon as β has been added to the compensation value, it has been redistributed and needs to be reset $\beta = 0$.
- Line 39 : All the gossip messages that a peer receives in between to rounds contain a gossip-msg.β. This value is accumulated in the variable β = β + gossip-msg.β, which will be applied or redistributed during the next application procedure.

```
Initialization:
    delivered = events ← {}
    $d^{P-1} = d^P = 0$
    gossip-msg.adaptF $= \alpha_r = 0$
    $F = F_{init}$
    $\beta = 0$
upon TIMER($t$ time units) at process $p_i$ do
    $F_{old} = F$
    $\beta$ = APPLYBETA($\beta$)
    if $currentRound > R$ then
        $compensate = \beta$
        $\beta = 0$
        SENDMSG($F$, gossip-msg, $compensate$)
    else
        $\alpha_r = \alpha[currentRound]$
        $\alpha_r$ = APPLYALPHA($\alpha_r$)
        $\beta$ = APPLYBETA($\beta$)
        $compensate = -\alpha_r + \beta$
        $\beta = 0$
        if $F < F_{old}$ then
            SENDMSG($F_{old}$, gossip-msg, $compensate$)
        else
            SENDMSG($F$, gossip-msg, $compensate$)
        end if
        $currentRound = currentRound + 1$
    end if
end upon
upon TIMER($T$ rounds) do
    currentRound = 1
    if $d^P > 0$ then
        $\Delta B$ = ($d^P$-$d^{P-1}$)/$d^{P+1}$
        $\alpha^{P+1}$ = TOTALALPHA($\Delta B, F$)
        $d^{P-1}$ = $d^P$
        $d^P$ = 0
        $\alpha^{P+1}$[] = DISTRIBUTIONOFALPHA($\alpha^{P+1}$)
    end if
end upon
upon RECEIVE(gossip-msg) do
    $\beta$ = $\beta$ + gossip-msg.$\beta$
    for all $e \in$ gossip-msg.events do
        if $e \notin$ delivered then
            events ← events ∪ $\{e\}$
            if ISINTERESTED($e$) then
                DELIVER($e$)
                delivered ← delivered ∪ $\{e\}$
                $d^P$ = $d^P$ + 1
            end if
        end if
    end for
end upon
```

Fig. 2. *fair-gossip* algorithm, generic

D. Modulating the distribution: APPLYALPHA(α_r), Figure 3

An α_r represents the change of fanout a peer **wants** to execute. The peer is not forced the actually perform those changes. But as soon as a peer applied a certain α_r to its own Fanout F , other peers **have** to compensate this α_r in order to keep the global fanout F_{global} constant. For those peers, this α_r has become a β.
So in case the application of a specific α_r would exceed the F_{min} or F_{max} limits, it is a question of algorithm design what to do with the $\Delta\alpha_r$ that couldn't be applied. One way would be to just discard $\Delta\alpha_r$. Another way would be to redistribute this $\Delta\alpha_r$ over the remaining $\alpha^{P+1}[]$ of this adaption phase. The protocol would be fairer because a peer's effort to reach its prospected fanout $F + \alpha^{P+1}$ is more consequent. Imagine a case where in one round a peer has to crop its α_r, but in the next round, the application of β would have allowed the resulting $\Delta\alpha_r$ from before : the discarded $\Delta\alpha_r$ was 'lost'.

```
function APPLYALPHA($\alpha_r$)
    $F_{temp} = F + \alpha_r$
    if $F_{temp} \notin [F_{min}..F_{max}]$ then
        if $F_{temp} > F_{max}$ then
            $\alpha_r = F_{max} - F$
            $\Delta\alpha_r = F_{temp} - F_{max}$
            $F = F_{max}$
        end if
        if $F_{temp} < F_{min}$ then
            $\alpha_r = F_{min} - F$
            $\Delta\alpha_r = F_{min} - F_{temp}$
            $F = F_{min}$
        end if
    else
        $F = F_{temp}$
    end if
    {Process $\Delta\alpha_r$ , e.g. discard, redistribute over remaining
    $\alpha^{P+1}$[]}
return $\alpha_r$
```

Fig. 3. *fair-gossip* algorithm, application of $alpha_r$

E. How to apply the distribution: APPLYBETA(β) , Figure 4

The β is the accumulated value all gossip-msg.β received in between two rounds. The peer has to either apply β or pass it on the other peers. If $F_{temp} = F+\beta \notin [F_{min}..F_{max}]$, the new value of β is the $\Delta\beta = F_{temp}-F_{max}$ or $\Delta\beta = F_{temp}-F_{min}$, because the rest still needs to applied. The rest of the β that couldn't be applied has to be redistributed.

F. SENDMSG(f, gossip-msg, c) , Figure 5

The compensation value c is passed on to f neighbors by distributing c over the gossip-mesg.β for each neighbor. At this time, no other method has been proposed.

G. TOTALALPHA($\Delta B, F$)

The return value α^{P+1} is the fanout increase a peer wants to execute during the next adaption phase. α^{P+1} is calculated as a function of the fanout F and the benefit ΔB, the realtive change of interesting messages. The prospected fanout after the adaption process is $F = F + \alpha^{P+1}$.

```
function applyBeta( $\beta$)
    $F_{temp} = F + \beta$
    if $F_{temp} \notin [F_{min}..F_{max}]$ then
        if $F_{temp} > F_{max}$ then
            $\beta = F_{temp} - F_{max}$
            $F = F_{max}$
        end if
        if $F_{temp} < F_{min}$ then
            $\beta = temp - F_{min}$
            $F = F_{min}$
        end if
    else
        $F = F_{temp}$
        $\beta = 0$
    end if
return $\beta$
```

Fig. 4. *fair-gossip* algorithm, application of β

```
function sendMsg($f$, gossip-msg, $c$)
    Neighbors ← selectPeer($f$)
    gossip-msg.events ← selectEvents($N$ in events)
    for all $p \in$ Neighbors do
        if $c = 0$ then
            gossip-msg.$\beta = 0$ for all $f$ neighbors
        else if $|c| > f$ then
            gossip-msg.$\beta = c/(f-1)$ for $(f-1)$ neighbors
            gossip-msg.$\beta = c \bmod (f-1)$ for the last neighbor
        else if $|c| \leq f$ then
            gossip-msg.$\beta = c/|c|$ for $|c|$ neighbors
            gossip-msg.$\beta = 0$ for the $f - c$ other neighbors
        end if
        p2pSend($p$, gossip-msg)
    end for
return
```

Fig. 5. *fair-gossip* algorithm, send gossip messages

1) Cumulative method, Figure 6: The prospected fanout is $F_{prospected} = F_{old} + \lceil \Delta B \cdot F_{init} \rceil$. There is a memory effect because F_{old} is used to calculate $F_{prospected}$. Because of the strong influence of the last adaption phase on the current one, this is called the cumulative adaption method.

2) Relative method, Figure 7: The prospected fanout is $F_{prospected} = F_{init} + \lceil \Delta B \cdot F_{init} \rceil$. The peer calculates its $F_{prospected}$ without considering F_{old}. But to determine α^{P+1}, it still takes into consideration F_{old} by setting $\alpha^{P+1} = F_{prospected} - F_{old}$. Because of the softer influence of the last adaption phase on the current one, this is called the relative adaption method.

3) Memoryless method, Figure 8: At the beginning of each adaption phase, the fanout F of every peer is reset to the initial fanout F_{init}. This means the network has no memory about the benefit of a peer. Imagine two peers, one with $F = F_{min}$ and the other with $F = F_{max}$ at the end of the last adaption phase. At the beginning of the next adaption phase they both have the same benefit ΔB, so therefore they both have the exact same chance to reach their prospected fanout $F_{prospected} = \Delta B \cdot F_{init}$.

```
function totalAlpha($\Delta B, F$)
    if $\Delta B \geq 0$ then
        $\alpha^{P+1} = \lceil \Delta B \cdot F_{init} \rceil$
    else
        $\alpha^{P+1} = \lfloor \Delta B \cdot F_{init} \rfloor$
    end if
return $\alpha^{P+1}$
```

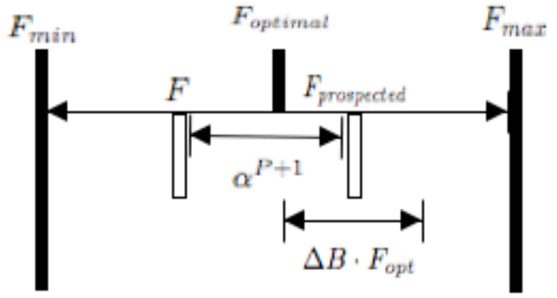

Fig. 6. *fair-gossip* algorithm, α^{P+1} cumulative

```
function totalAlpha($\Delta B, F$)
    if $\Delta B \geq 0$ then
        $\alpha^{P+1} = \lceil (1 + \Delta B) \cdot F_{init} \rceil - F$
    else
        $\alpha^{P+1} = \lfloor (1 + \Delta B) \cdot F_{init} \rfloor - F$
    end if
return $\alpha^{P+1}$
```

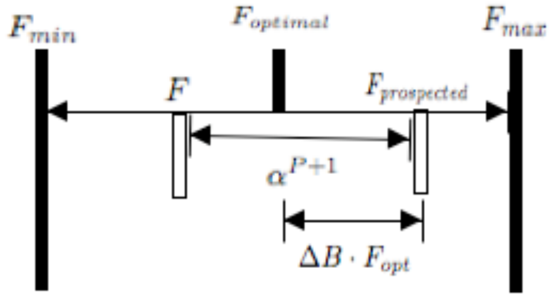

Fig. 7. *fair-gossip* algorithm, α^{P+1} relative

H. distributionOfAlpha(α^{P+1})

When a peer calculated its α^{P+1} at the beginning of an adaption phase, it has to distribute this value over an array of the length of the adaption phase. The adaption is done in little steps to avoid too brisk fluctuations of the global fanout, e.g. if a lot of peers would decrease their fanout in one round by a significant value, the network would loose its reliable.

```
function TOTALALPHA($\Delta B, F$)
    $F = F_{init}$
    if $\Delta B \geq 0$ then
        $\alpha^{P+1} = \lceil \Delta B \cdot F_{init} \rceil$
    else
        $\alpha^{P+1} = \lfloor \Delta B \cdot F_{init} \rfloor$
    end if
return $\alpha^{P+1}$
```

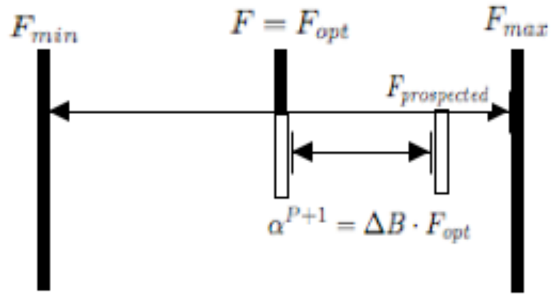

Fig. 8. *fair-gossip* algorithm, α^{P+1} memoryless

1) Modulo method, Figure 9: This simple distribution method is designed for an adaption phase of R_{init} rounds by dividing the α^{P+1} in $R_{init} - 1$ parts, and the rest - e.g. the modulo - is stored in α^{P+1}[R].

```
function DISTRIBUTIONOFALPHA($\alpha^{P+1}$)
    if $|\alpha^{P+1}| > R_{init}$ then
        $R = R_{init}$
        $\alpha^{P+1}$[] = $newArray[R]$
        for all $i = 1 < R$ do
            $\alpha^{P+1}[i] = \alpha^{P+1}/(R-1)$
            $i = i + 1$
        end for
        $\alpha^{P+1}[R] = \alpha^{P+1} \bmod (R-1)$
    else
        $R = 1$
        $\alpha^{P+1}$[] = $newArray[R]$
        $\alpha^{P+1}[R] = \alpha^{P+1}$
    end if
return $\alpha^{P+1}$[]
```

Fig. 9. *fair-gossip* algorithm, α^{P+1}[] modulo

2) Decreasing method, Figure 10: Given R_{init}, the idea of the decreasing distribution is to adapt as much as possible of α^{P+1} at the beginning of the adaption phase in order to give the network more time to get adjusted. By letting the peers adapt a big percentage of their α^{P+1} in the first round, it is ensured that there won't be any big fluctuations towards the end of the adaption phase, e.g. the average α_r will decrease as well.

3) Constant method, Figure 11: A somewhat different approach is the constant distribution. Contrary to the methods before, the idea is not to limit the number of rounds of the adpation phase R. The goal is to keep the average α_r constantly low, therefore keep the number of necessary redistributions to a minimum. On the other hand there is no guarantee about the length adaption process, as all peers can have a different R.

```
function DISTRIBUTIONOFALPHA($\alpha^{P+1}$)
    $R = R_{init}$
    $\alpha^{P+1}$[] = $newArray[R]$
    $\alpha^{P+1}[R] = \alpha^{P+1}$
    for all $i = 1 < R$ do
        $\alpha^{P+1}[i] = \lfloor \alpha^{P+1} \cdot p_i \rfloor$
        $\alpha^{P+1}[R] = \alpha^{P+1}[R] - \alpha^{P+1}[i]$
        $i = i + 1$
    end for                              $\{p_i > p_{i+1}\ ,\ \sum_{i=1}^{R-1} p_i < 1\}$
return $\alpha^{P+1}$[]
```

Fig. 10. *fair-gossip* algorithm, α^{P+1}[] decreasing

```
function DISTRIBUTIONOFALPHA($\alpha^{P+1}$)
    $R = |\alpha^{P+1}|$
    $\alpha^{P+1}$[] = $newArray[R]$
    for all $i = 1 < R$ do
        $\alpha^{P+1}[i] = \alpha^{P+1}/|\alpha^{P+1}|$
        $i = i + 1$
    end for
return $\alpha^{P+1}$[]
```

Fig. 11. *fair-gossip* algorithm, α^{P+1}[] constant

III. IMPLEMENTATION

The goal of this project was to implement a push-based and fair gossip algorithm. Since the idea of fair gossiping is still in its early stages, the framework for the implementation of the protocol needed to be high level and fast. Peersim offers just that.

A. Peersim : The Simulator

The Peersim framework can be used to simulate P2P protocols. It is written entirely in Java. It disposes of a cycle-driven engine and is able to simulate up to one million peers. There is no transport layer or concurrency handling, all peers are instances of a class $Node$ and they can communicate with each other by just accessing the other peer's instance. On top of the nodes are the protocol instances. In some sequential order every node executes its protocols, during which those can perform calculations or call methods of other peers protocols for example. Additionally, there are the control protocols which run at the beginning or the end of each cycle. By using a control instance you can access the network, e.g. compute statistics, add or remove peers or create events.

B. Fair Gossip Implemenation

The fair gossip protocol presented in this project is a combination of two protocols, the network layer protocol and the fair gossip protocol itself. There are several controls used for the simulation, e.g. a timer which manages the rounds and the length of the adaption phase, an observer which computes statistics about the current state of the network and a control to insert messages into the network. Finally there are some utilities that simplify the handling of the big amount of data collected.

1) Simplifying assumptions: At this point, the fair gossip protocol is designed to test and experiment with the adaption phase. Therefore, the following simplifying assumption has been made. The benefit ΔB is not calculated by sending messages and evaluating their interest (even though there is a functionality to send messages over the network), but rather by just choosing a random benefit based on predefined distribution. So far, it has been tested with a uniform, a constant, a bernoulli and a gaussian distribution.

2) Protocols:

*a) **RandomProtocol**:* This protocol simulates a simplified network layer. It bypasses the graph-based functionality of peersim but instead offers a fast way to create a random number neighbors for a peer. All nodes can have a different fanout F. We suppose that the neighbors are selected randomly among the population of the whole network and the list of neighbors is updated every round.

*b) **FairGossip**:* This is the main protocol which simulates the actual adaption phase. It is this protocol which determines the prospected fanout, distributes the α^{P+1}, tries to compensate β and sends the messages over the network.

3) Controls:

*a) **Timer**:* The timer manages the rounds of the phase, e.g. increments the $currentRound$ after each simulation cycle and resets it after the length of one phase.

*b) **Observer**:* After each simulation the observer computes statistics about the current state of the network and stores them in a file. The collected data includes the average, variance, maximum, minimum and the sum of the fanout F, the benefit ΔB, the distribution α_r, the redistribution β, etc. of each peer.

*c) **ValueDumper**:* Peersim offers this control to dump one specific value into a file. This is used to compute the distribution of the peers among the fanout F by dumping the fanout F of each peer and in every round.

4) Utilities:

*a) **GNUPlot**:* With the significant amount of data collected during the simulation, the work necessary to display the results becomes painful. To simplify this task, the observer uses an instance of $GNUPlot$ to generate a plot file.

5) Additional functionality: It exists the possibility to **introduce messages** into the network by using the control $InitializeMessages$. The observer collects relevant data about the dissemination of the messages and plots statistics like the number of infected peers, the number of newly infected peers, the number of messages sent and the number of duplicated messages per round. In the future, this mechanism could be used to simulate the calculation of the benefit based on a real ISINTERESTED(e) method.

In order to test under which configuration there is an **atomic broadcast**, the python script $fdplot.py$ executes a number of experiments with different random seeds, fanouts and 'infect for x rounds' values. The resulting datafile can be transformed in a 3d plot showing the percentages of an atomic broadcast for each configuration.

One of the most important criterion to evaluate the fair gossip protocol is the distribution of how many peers have a certain fanout, the **'Fanout/Peers' plot**. The control $ValueDumper$ stores the fanout F of each peer in every round. In a next step, a new data file is computed by counting the number of peers who have the same fanout F in one round. Therefore there are as many plots as there have been cycles during the experiment which show the evolution of how the peers are distributed over the fanout.

IV. EVALUATION

During the implementation of such a high level algorithm as proposed, there are a lot of design choices that have an influence on the performance. In order to evaluate different strategies and test cases, we need to define criteria which can be used to judge the performance.

A. Evaluation Metrics

1) **'Fanout/Peers' plot** : For every round of our adaption process, we can count how many peers have the same fanout. So on the x-axis of the resulting plot is the fanout range $[F_{min}, F_{max}]$ and on the y-axis the number of peers with a certain fanout. In the end there are $R + R'$ graphs that indicate the evolution of the distribution of the fanout over the peers.
2) **Global fanout** F_{global} : Small fluctuations of the global fanout $\sum_i F_{p_i} = F_{global} \simeq F_{global}^{optimal} = \sum_i F_{init}$ cannot be avoided.
   - There are no problems with the reliability if $\sum_i F_{p_i} > F_{global}^{optimal}$.
   - But if the global fanout drops $\sum_i F_{p_i} < F_{global}^{optimal}$, we loose reliability.

   The global $\alpha_r^{global} = \sum_i \alpha_r^i$ is directly linked to F_{global}. Minimizing α_r^{global} in one round signifies minimizing the fluctuations of the global fanout F_{global} and the number of redistributions. The graphs display the average fanout $F_{average}$ and average $\alpha_r^{average}$ rather than the global fanout F_{global} or α_r^{global}.
3) **Deviation of the prospected fanout** $F_{prospected}$: The difference between the fanout F after an adaption phase and the prospected fanout $F_{prospected}$ is the $Deviation = |F - F_{prospected}|$. It determines how close the peers came to reaching the fanouts that they wanted.
4) **Number of redistributions**: An elevated number of redistributions in one round towards the end of the adaption phase is likely to enlarge R'. In general, we would like to keep the number of redistributions to a minimum.

5) **Length of the adaption phase** : We want the protocol to be $fair$, reliable and stable for as long as possible. Therefore, the length of the total adaption phase $R_{total} = R + R'$ needs to be minimized.

B. Test cases

The next step is to compose an algorithm, from the different options proposed in the generic form, which satisfies all evaluation metrics. The challenge is to find a good trade-off between the length of the adaption phase on one side, and α_r^{global}, F_{global} and the number of redistributions on the other side.

All tests simulate a network of 100'000 peers, each of which starts with fanout $F_{init} = F_{optimal} = 11$. The boundaries for the fanout are set to $F_{min} = 1$ and $F_{max} = 21$. All of those evaluation metrics should be satisfied when applied to different scenarios. The following scenarios have been considered.

1) **Constant ΔB Distribution**: The benefit of all nodes is the same. As a consequence, all peers should have to same fanout at the end of the adaption phase as at the beginning $F_{new} = F_{old}$ because all compensations cancel each other out. That all peers loose interest at the same time is an extreme event but useful to test the robustness of the protocol.
2) **Uniform ΔB Distribution**: The benefit of each node is uniformly distributed over the interval [-range,range]. Thus the expectation value for ΔB is zero. We expect the peers to be distributed uniformly among the fanout interval $[(1 - range) \cdot F_{init}, (1 + range) \cdot F_{init}]$
3) **Bernoulli ΔB Distribution**: There are two possible values a and b for ΔB and each node's benefit corresponds to one of those two values with the same probability. The result we expect in the 'Fanout/Peers' plot are two peaks that are situated at $(1 + a) \cdot F_{init}$ and $(1 + b) \cdot F_{init}$.
4) **Gaussian$(0, \sigma)$ ΔB Distribution**: As with all the other distributions, we expect the peers to be normally distributed over the fanout range with a mean equals to F_{init} and a standard deviation similar to σ in appearance.

C. Cumulative vs. Relative vs. Memoryless

To evaluate the different results between the cumulative, the relative and the memoryless way to calculate the new α^{P+1} at the beginning of an adaption phase, we consider the following scenario. There are **three adaption phases** in serie, at the beginning of each the peers generate a new ΔB using the same distribtion. For the ΔB distribution, we apply a Bernoulli variable with the two values [-0.3,0.3] and then we compare the resulting 'Fanout/Peers' plots for the constant method to distribute α^{P+1} among $\alpha_r[]$. In Figure 12 (p. 8) you can see the results of the simulations. The graph is expected to yield two peaks, one for the peers who want to increase their fanout by 30% and those who want to decrease it by 30%.We can clearly see that the **cumulative method is not suited** for its purpose because after only three rounds the 'Fanout/Peers' plot is already distorted. In fact, half of the peers that could decrease their fanout by 30% in the first adaption phase want to increase their fanout by 30% in the second phase and vice versa. The memoryless approach logically produces three times the same graph since the fanout is reset to F_{init} at the beginning of each adpation phase. Is there a way to increase fairness by taking into account the previous adaption phase? E.g. imagine two peers A and B with $F_{old}^A = a$ and $F_{old}^B = b$, $a < b$. Assume both peers had the same positive benefit during the last round, thus calculate the same $F_{prospected} = F_{init} + \lceil \Delta B \cdot F_{init} \rceil$. Using the relative method, this results in $\alpha_A^{P+1} = F_{prospected} - a < F_{prospected} - b = \alpha_B^{P+1}$. With α_A^{P+1} and α_B^{P+1} we now have a mean to distinguish two peers who have the same $F_{prospected}$. How do we define fairness? Which of the peers has more right to reach $F_{prospected}$? For example, let us say that a peer with a small benefit ΔB (thus running 'steadily') has more right to reach its $F_{prospected}$ than one with a big benefit. Applied to the previous example this would mean that B has more right to reach $F_{prospected}$. To increase the possibility that peer B reaches $F_{prospected}$, we could define that a peer decides in function of α^{P+1} whether to apply a compensation or whether to redistribute it directly. The smaller α^{P+1}, the smaller the percentage to accept a compensation. Still, there exists the prerequisite that if nothing is done with α_A^{P+1} and α_B^{P+1}, both peers A and B should have the same chance to reach $F_{prospected}$. For the graph in Figure 12 (p. 8) this means, since currently all α^{P+1} are treated the same way, that the 'Fanout/Peers' distribution should stay the same for all three adaption phases. This is the case, therefore the **relative strategy is valid** and can be extended to implement a *fairer* protocol.

D. General observations for 'Fanout/Peers' distribution

In Figure 13 on page 9 we can see results of one adaption phase for each proposed scenario for the distribution of ΔB (from top to bottom: constant[-0.6], bernoulli[-0.6,0.6], uniform[-0.6,0.6], gaussian(0,0.09)) applied to each of the proposed strategies to distribute α^{P+1} over $\alpha_r[]$ (from left to right: constant, modulo, decreasing). The distributions of ΔB are extensive, meaning almost all peers deliver a very different number of interesting events in each phase. In reality we expect the ΔB to relatively small (e.g. a normal distribution with mean zero and a small variance). In this case the ΔB distributions serve to determine the reaction of the network to extreme events. We observe, by looking at each line seperately, that the graphs look alike and more or less how they are supposed to look like. It is worth noticing that the decreasing strategy has the tendency to accumulate peers at the boundaries F_{min} and F_{max}.

E. Fanout Global F_{global}

In this section we assume that we can work with the relative method to calculate the α^{P+1} of the next adaption round. We want to evaluate the effects of the different methods to distribute α^{P+1} over the R adaption rounds. First, let us have a look at the event when all peers want to decrease their fanout by 60% (constant[-0.6]). In Figure 14 (p. 10) we can see the evolution of $F_{average}$. It is clear that the F_{global} will

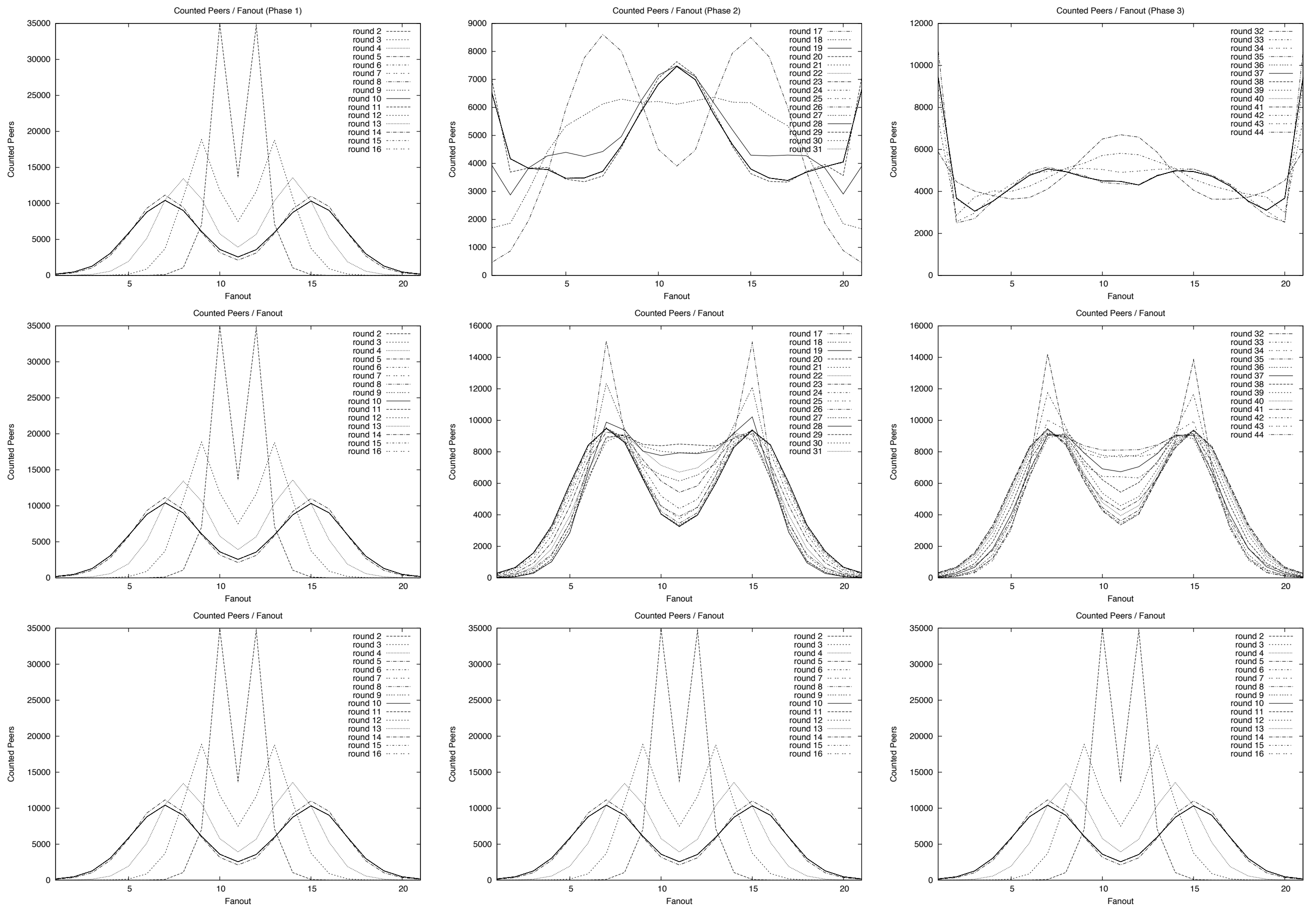
Fig. 12. Comparison between cumulative(top), relative(middle) and memoryless(bottom) strategy for calculation of α^{P+1} over three adaption phases using a Bernoulli[-0.3,0.3] distribution for ΔB

drop below the $F_{global}^{optimal}$ because all the peers try to decrease their fanout. In the graph on the left side, using the constant method, the fanout drops to 10.5 at most. When using the modulo or the decreasing method the fanout drops to 9.5 and 9 respectively, which can have an effect on the reliability of the network. So the constant method has an advantage over the two other methods regarding the global fanout F_{global}. But looking at Figure 15 (p. 11), the disadvantage of the constant method becomes apparent. Because all the peers change their fanout by just 1 at the time, the length of the adaption phase is variable and long. For a gaussian(0,0.36) ΔB distribution it can take up to 18 rounds for the F_{global} to adjust. To be able to compare the modulo and decreasing method, we can see in Figure 16 (p. 12) that the α_r^{global} is decreasing with every round in the decreasing method, whereas with the modulo method this value can fluctuate, depending on the modulo for the last change $\alpha_r[R]$. In the graph displayed, you can see that for the modulo method, $\alpha_r[1]^{average}$ is smaller than $\alpha_r[2]^{average}$. For the decreasing method, $\alpha_r[1]^{average}$ is the largest of all.

F. Redistributions

Even though the redistributions can be embedded in normal messages and thus don't necessarily mean that extra messages need to be transported over the network, we still want to minimize the number of redistributions necessary. More exactly, minimizing the number of redistributions in one round is more important than minimizing the overall number redistributions because we want to maximize the probability of transporting all redistributions over normal messages. Looking at Figure 17 (p. 13), we immediately see that the number of redistributions is generally lower when using the constant method (1st row), but they are distributed over more rounds. This is logical because $\alpha_r = 1$ for all peers, and the network stays more or less balanced during the adaption phase, e.g. the chance that a peer receives a compensation that it cannot apply is minimal. The modulo method (2nd row) also yields a smaller number of redistributions compared to the decreasing method (3rd row). This is due to the fact that α^{P+1} is uniformly distributed over $\alpha_r^{1...(R-1)}[]$ when using the modulo method. The decreasing method assigns 50% of the α^{P+1} to $\alpha_r[1]$. The network is not balanced at all after the first adaption round (see IV-E), thus the probability that a peer receives a compensation that it is not able to apply is bigger, and this explains why the number of redistributions is more elevated.

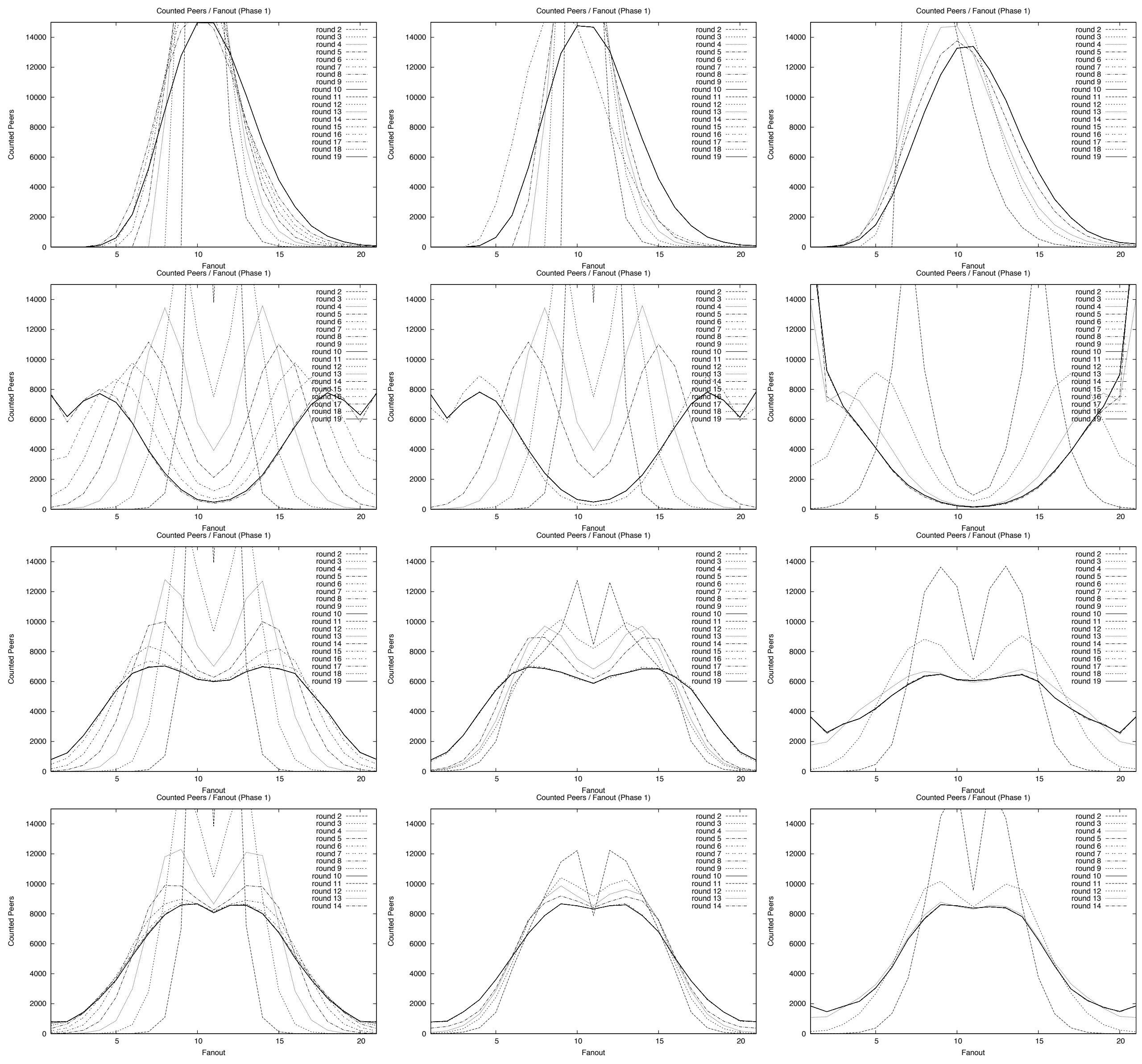

Fig. 13. 'Fanout/Peers' plot for a first adaption phase. Top to bottom : constant[-0.6], bernoulli[-0.6,0.6], uniform[-0.6,0.6], gaussian(0,0.09). Left to right : constant, modulo, decreasing strategy to distribute α^{P+1} over $\alpha_r[]$

G. Deviation of $F_{prospected}$

The deviation gives an indication about how close a peer came to reach its prospected fanout $F_{prospected}$. Now it is important to understand that we cannot simply say, the smaller the deviation the better. In table I for example, if all peers would like to decrease their fanout by 80 percent (constant[-0.8]), theoretically they should all have the same fanout at the end of the adaption phase as at the beginning, $F_{new} = F_{old}$. For the deviation, this means that it should be excatly the difference $deviation = F - F_{prospected} = -\alpha^{P+1}$ (indeed we have in this example : $\alpha^{P+1} = \lfloor \Delta B \cdot F \rfloor = \lfloor -0.8 \cdot 11 \rfloor = \lfloor -8.8 \rfloor = -9$). Therefore the deviation can only be used as a metric when comparing values that originated from the same ΔB distribution. In table I we can see that the average as well as the variance deviation don't differ among the same ΔB distribution. Therefore, to profit from this metric, several adaption phases have to be taken into consideration (see VI-A). Also, it would be interesting to track the deviation of a specific peer over several adaption phases (see VI-B).

TABLE I
DEVIATION OF $F_{prospected}$

(avg,var)	∼Constant	∼Bern	∼Unif	∼Gauss
constant	(9 , 9)	(1.6 , 2.5)	(1.6 , 1.65)	(1.9 , 3)
modulo	(9 , 9)	(1.5 , 2.4)	(1.6 , 1.65)	(1.9 , 3)
decreasing	(9 , 9)	(1.5 , 2.4)	(1.8 , 1.82)	(1.9 , 3)

H. Summarization

At this early stage, it is hard to say which combination of the proposed methods works best for the fair gossip protocol.

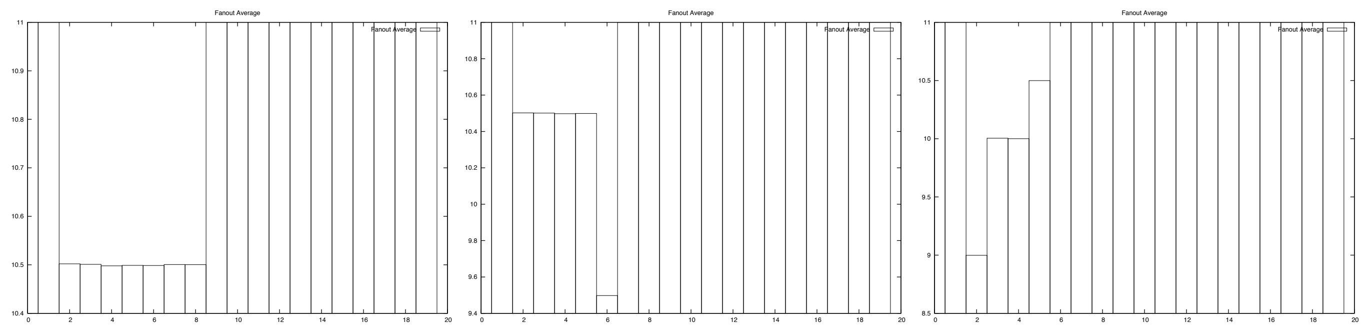

Fig. 14. Extreme Event : every peer has a benefit of $\Delta B = -0.6$. Left to right : constant, modulo, decreasing strategy to distribute α^{P+1} over $\alpha_r[]$

Each of the methods has its advantages and disadvantages as described during the evaluation. In the table II, those observations are displayed in a more compact way.

TABLE II
OVERVIEW OF EVALUATION RESULTS (LEFT TO RIGHT: DIFFERENT METHODS TO DISTRIBUTE α^{P+1} OVER $\alpha_r[]$)

	constant	modulo	decreasing
Adaption length $R + R'$	↓	→	↑
F_{global}	↑	↓	↓
Redistributions	↑	→	↓
Stability	↑	→	↑
Fairness	↑	↑	→

To see why the modulo method has disadvantage concerning adaption length and stability, have a look at Figure 14 (p. 10). It is possible that due to an unfortunate combination of adaption length and α^{P+1}, the modulo $\alpha_r^{average}[R]$ is much larger than than the other $\alpha_r^{average}[1...R-1]$. This means that the R' is likely to be longer and brings the network out of balance just before the end of the adaption phase. The decreasing method has a slight disadvantage concerning the fairness because we saw in Figure 13 (p. 9) that it has the tendency to accumulate peers at the boundaries F_{min} and F_{max} when it shouldn't.

V. USER MANUAL

The purpose of this section is to give instructions on how to setup a development workstation. The requirements are a Java SDK, a Python SDK and of course the Peersim framework.

A. Starting Simulations

All simulations are run using peersim. Nothing was changed in the source of peersim itself, so you could work with the binaries only (peersim-1.0.1.jar at the time of writing), but having access to the source is still useful to understand how the FairGossip algorithm works. The class *peersim.Simulator* is used to start a simulation by passing it the config file as an argument. The config file for the FairGossip algorithm is located in *config/fairgossip.txt*. To start a simulation using a terminal :

1) go to the source folder FairGossip/src and compile all classes in the folders fairGossip/controls , fairGossip/protocols and fairGossip/utils using *javac fairGossip/XXX/*.java* (or type *make* in the base folder FairGossip to compile all classes, peersim and FairGossip)
2) go to the folder base folder FairGossip and make sure there are all libraries (djep-1.0.0.jar, jep-2.3.0.jar). Start the simulation by typing *java -classpath jep-2.3.0.jar:djep-1.0.0.jar:src peersim.Simulator config/fairgossip.txt*.

Of course it is much more convenient to setup a project in an IDE as for example Eclipse.

B. Scripts

Two python scripts were used, one for the 'Fanout/Peers' plot which serves as a metric for the evaluation and the other one to evaluate the dissemination of messages concerning atomic broadcasts.

- **peer_fanout.py**: This script collects the fanout data about each peer in each round and then generates a data file for each round which maps the fanout and the number of peers with that specific fanout. This is done by iterating over the fanout dump file of each round and counting how many peers have the same fanout. Notice that in order for the dump file to be generated, the control *fanoutdumper* has to be activated in the config file for the simulation (*config/fairgossip.txt*). After the script has generated the data file it prepares a gnuplot file in the folder *reports/peers_fanout.plt* which can be plotted by executing *gnuplot peers_fanout.plt*. Last but not least the script has to know the length of an adaption phase and how many adaption phases were simulated. These two variables have to be mapped from the config file of the simulation. To executed the script, type *python2.5 scripts/python/peer_fanout.py* in the FairGossip folder.
- **fdplot.py**: To use this script, the flag *RDINFECT* has to be set to true in the control *FairGossipObserver*. The script runs a number of experiments and creates a data and a plot file to represent the results in a 3d plot. By varying the random seed and iterating over different fanout and 'infect for x rounds' values the script tries to determine under which condition the we have an atomic broadcast (thus an increased reliability). This feature could prove to be useful to test the reliability of the FairGossip algorithm, up to know it has only been used

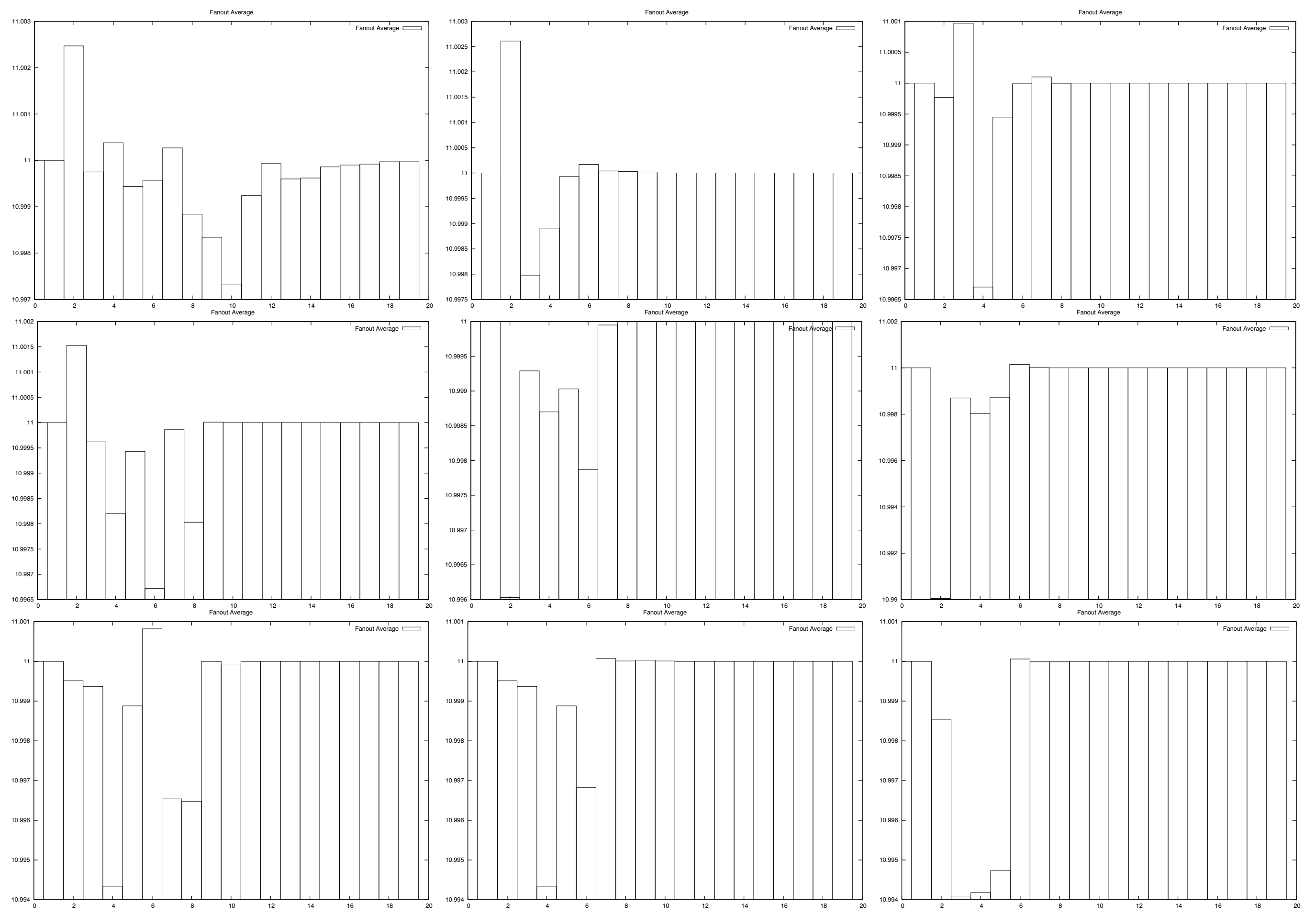

Fig. 15. The evolution of $F_{average}$ over one adaption phase. From top to bottom : Gaussian, Uniform, Bernoulli. From left to right : Constant, Modulo, Decreasing

to verify the correctness of the implementation. Notice that the script generates its own configuration. It can be executed by typing *python2.5 scripts/python/fdplot.py* in the FairGossip folder. Since this script was used at a very early stage of the project it might need some updates in order to work properly. See Figure 18 for an example of the plot.

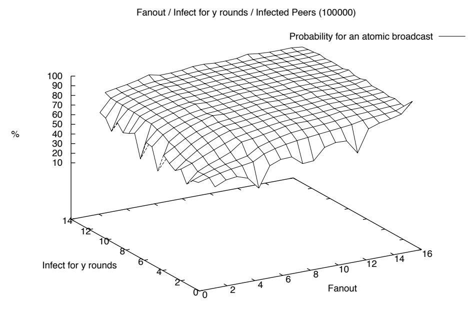

Fig. 18. Probability for an atomic broadcast

- **PlotandMoveToReport.sh**: A little executable script that collects all the simulation results and moves them to the folder *reports/report* (the system output if you redirect it to the file *reports/log/syso_output.txt*, all graphs from the plot file created by the FairGossipObserver and all graphs from the 'Fanout/Peers' script)). The script is located in *reports/PlotandMoveToReport.sh*, execute by typing *./PlotandMoveToReport.sh* (after making it executable by *chmod u+x PlotandMoveToReport.sh*). Notice that the reports are moved rather than copied to avoid too much confusion.

VI. Future work

As always and as with almost everything, there are a lot of things than can be worked on. Here an overview over the different directions.

A. Next steps

- The deviation of $F_{prospected}$ in regard to F at the end of an adaption phase should be measured over several adaption phases. This should give us more information about the two different approaches to calculate α^{P+1}, the relative (fig. 7, p. 4) and the memoryless approach

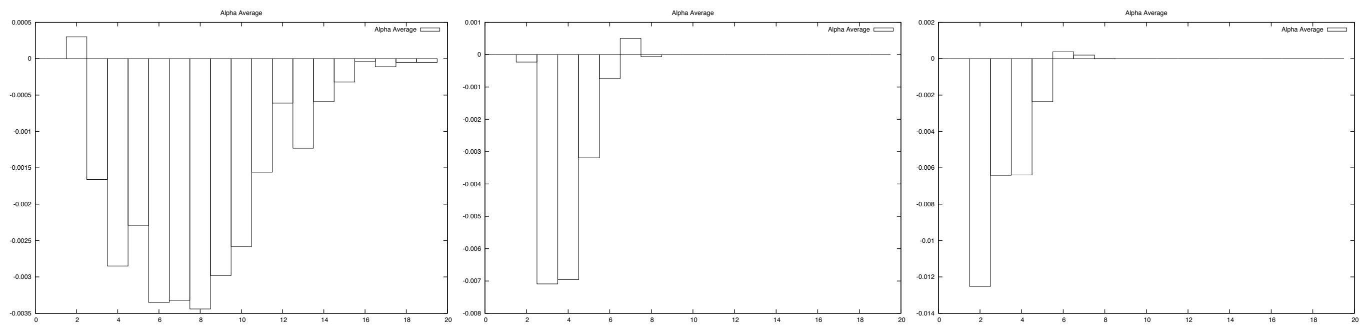

Fig. 16. The $\alpha_r^{average}$ for a gaussian(0,0.35) ΔB distribution. Left to right : constant, modulo, decreasing strategy to distribute α^{P+1} over $\alpha_r[]$

(fig. 8, p. 5), e.g. we want the deviation to be the same in case all peers are given the same priority (see IV-C).

- At this point, the benefit ΔB is generated according a random variable. Implementing the concept of a message would enable us to calculate a real benefit ΔB. The model would be closer to reality and the effects of the different strategies on the stability and reliability could be examined more closely.
- As described in IV-E, there could be problems with the reliability of the network in case $F_{global} < F_{global}^{optimal}$. The effects could be tested using the script *fd_plot.py* (see V-B), which simulates a series of experiments to determine the probability of an atomic broadcast.
- The effect of the length of the adaption phase on the result is difficult to determine because the effects of numerous proposed strategies are too significant to draw conclusions about the length of the adaption phase. The amounts of simulations and analysis would exceed the scope of this project, especially when considering that the impact probably won't be huge.

B. Additional evaluation

- Refine the ability to measure the deviation of $F_{prospected}$, e.g. implement an observer who tracks peers along the adaption process. We could then study the differences between the deviation from peers with small a α^{P+1} and the ones with big a α^{P+1} (see IV-C).
- The lower and upper limits for the fanout F , F_{min} and F_{max}, surely have an effect on the performance of the algorithm. At this point it is difficult to actually determine the consequences of changing these bounds. As an example, see Figure 19 on page 15. All graphs are as expected, notably maybe the logically much larger number of redistribution with narrow range.

C. Ghost Fanout

In the FairGossip algorithm as proposed, all peers change their fanout F during an adaption phase. Another approach would be the perfom the adaption process on a *Ghost Fanout* F_{ghost}, an artificial fanout that is affected only after F_{global} is stable again. The challenge with this approach is that there is no mean to detect when this ghost fanout is stable again. A solution would be to fix a adapation length where the probability of a stable system is close to 1.

D. Network size

All the theory works fine when considering a network that constantly has the same number of nodes. But in reality peers join and quit the network frequently. A peer doesn't know the size of the network, so how can we guarantee reliability without knowing what the global fanout F_{global} should be? One solution would be to estimate the size of the network, which is difficult to implement. Another would be to fix the optimal fanout $F_{optimal}$ higher than what it has to be in order to have a margin for a growing network.

E. Self Punishing

The fair algorithm proposed is only fair as long as all participating peers have clear conscience. Unsocial peers could easily corrupt the network by just decreasing their own fanout and not accepting any compensations that would increase it. There is no mechanism that could prevent or even detect such behavior in a push-based gossip algorithm. What makes it even worse with the fair algorithm proposed, a peer could decrease its fanout and others would even bear his workload.

F. Contagion period

At this point the nodes only change their fanouts. We can also imagine that the fanout stays constant and it is the contagion period that is being modulated by the node. This means that a peer disseminates a message not only one time but multiple times ('infect for x rounds'). The mechanism that guarantees stability would need to be adapted but could work on the same principles. One could also imagine a hybrid version where a peer changes both its fanout and its contagion period.

G. Push-Pull algorithm

A logical advancement would be to extend the existing push-algorithm to be a push-pull-algorithm. The advantages are numerous, just to name a few :

- A peer could send redistribution only to peers who can compensate it

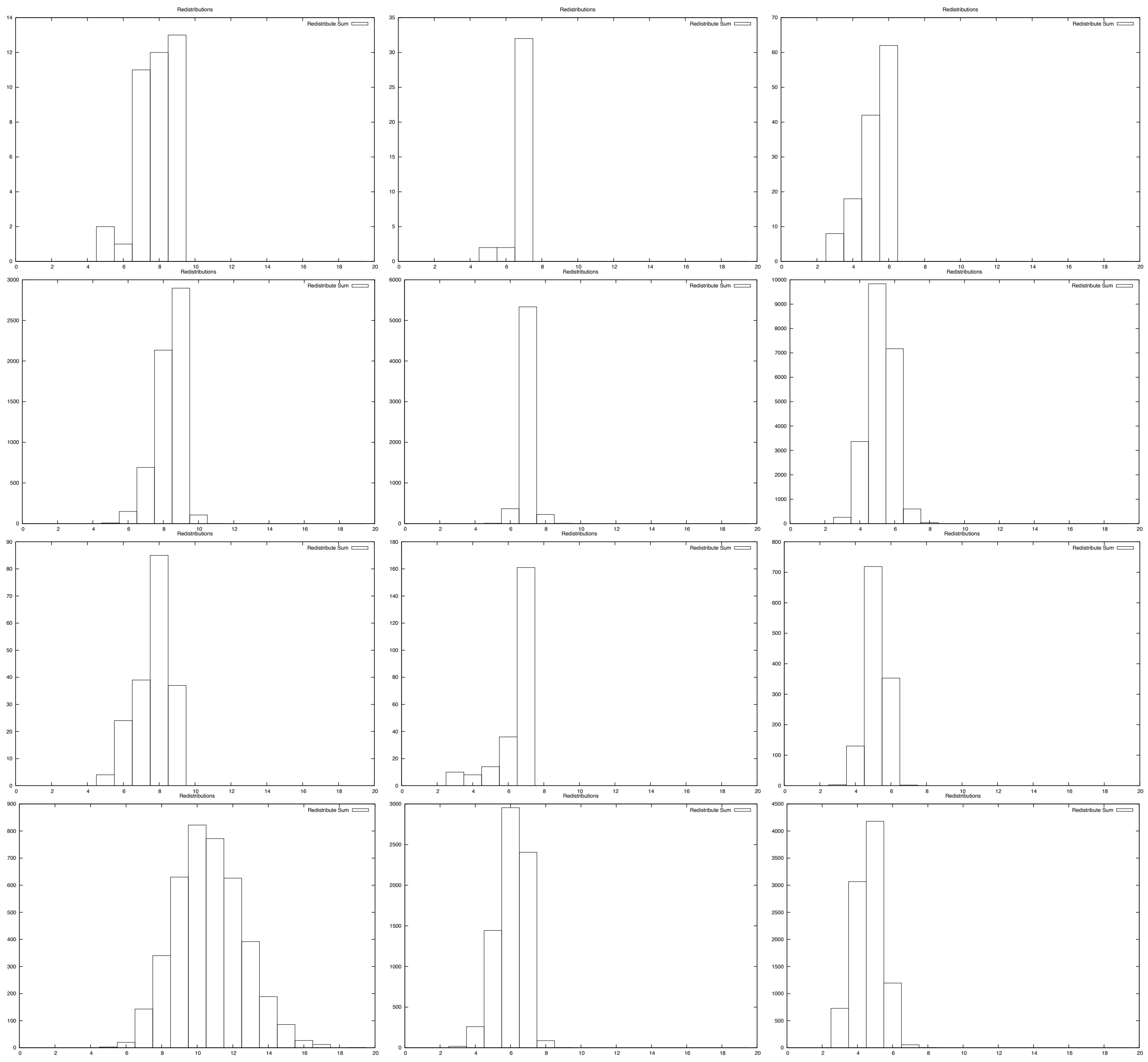

Fig. 17. The number of redistributions during the first adaption phase. Top to bottom : constant[-0.6], bernoulli[-0.6,0.6], uniform[-0.6,0.6], gaussian[0.6]. Left to right : constant, modulo, decreasing

- Reliability issues could be resolved by saying that a peer can only lower its fanout after the change was already compensated
- etc.

Unfortunately and logically, there are also disadvantages that make life more complicated:

- Added complexity
- Concurrency problems
- Added network traffic
- etc.

On the other hand, when one attempts to deploy the proposed FairGossip algorithm onto a real distributed system (see VI-H), the communication between peers needs to be reliable anyway, e.g. a peer doesn't know what happens with a compensation that it distributes, but it still needs to be certain that the compensation has been received on the other side. As a consequence, some sort of push-pull algorithm is already needed when deploying the push-based FairGossip algorithm, therefore it would make sense to extend the algorithm when performing that step.

H. Deploy onto a distributed system

Peersim is a good way to conceptually develop the idea for the proposed fair protocol. But when the algorithm has reached a certain stage, it makes sense to deploy it onto a real distributed system to test its behavior. Only then we can test things like the reliability, the stability, the additional workload compared to a standard gossip dissemination, the clustering effects etc.

I. Fairness generalisation

The only parameter which is used to evaluate the benefit of a peer is the the number of interesting events received in one phase. One can easily imagine to use another parameter to calculate the benefit. The FairGossip algorithm is not bound to the number of events delivered but instead could be generalized for any other measurable and comparable parameter. Consider for instance the optimization of resource problem where the load of each peer should be proportional to their capacities. We can assume that here the benefit (the capacity) reflects the average load (e.g.available bandwidth) of a peer during one phase, and that the adaptivity mechanism we developed can be used to lower the load on loaded peers and increase it on unloaded ones.

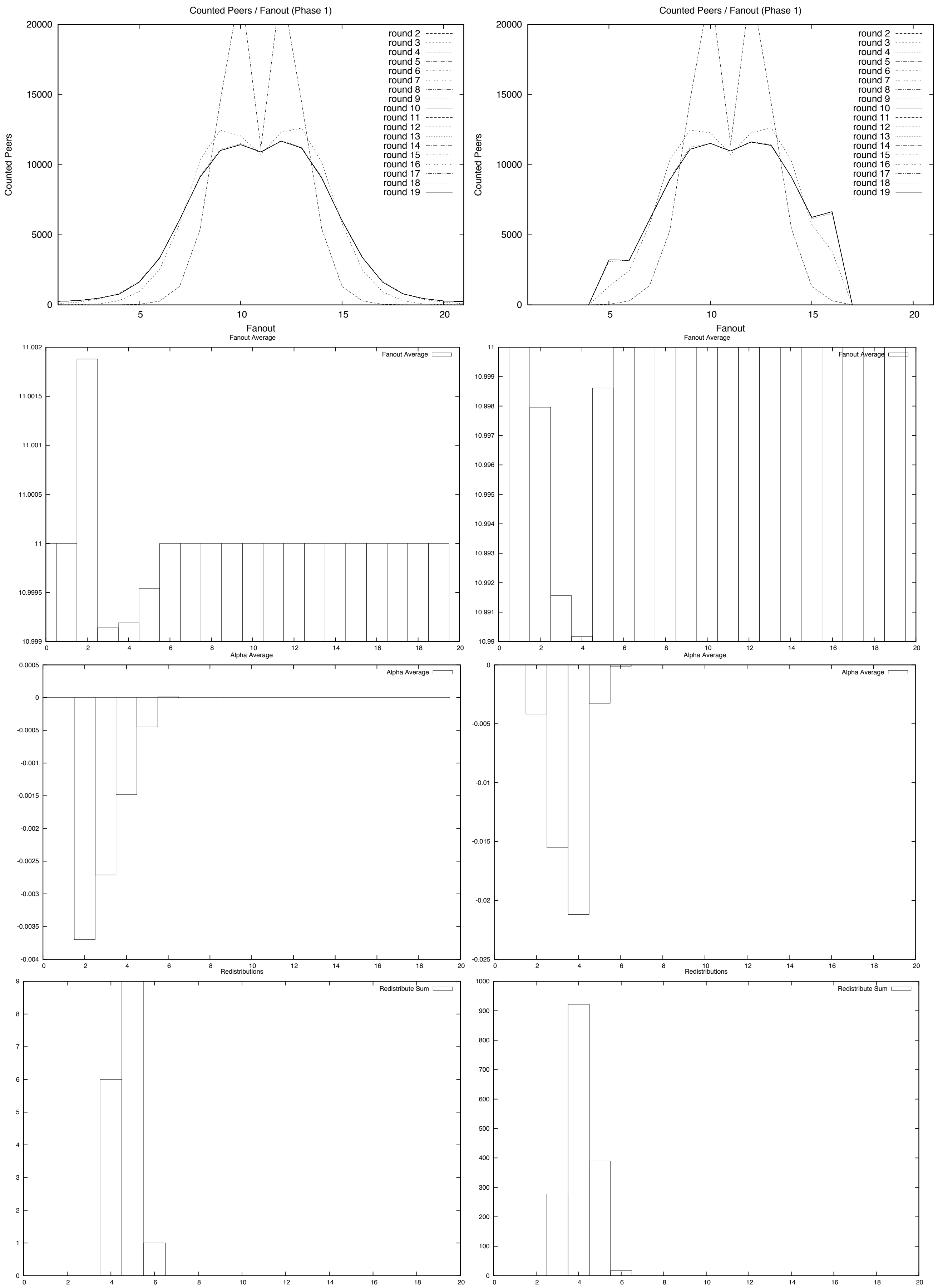

Fig. 19. Comparison between a fanout range of [1,21] (left) and [5.16] for a ∼Gaussian(0,0.04).Top to Bottom : 'Fanout/Peers' plot, $F_{average}$, $\alpha_r^{average}$, redistributions.