

Evaluation of the Neo P2P Blockchain Network Protocol Efficiency

Anna Shaleva and Vladimir Korkhov(✉)

Saint Petersburg State University, St. Petersburg, Russia
st050450@student.spbu.ru, v.korkhov@spbu.ru

Abstract. P2P networks along with the other distributed systems need to satisfy the strict requirement of efficiency in terms of the information propagation process. Fast message transmission affects the whole network functionality which includes data processing, attacks robustness, topology changes, etc. In this paper we consider the most wide-spread gossip-based network protocols, explore and compare several approaches to improve the information dissemination model. We describe the Neo blockchain network protocol which extends gossip algorithms for message propagation over the P2P network. Based on the imitation modelling results, we build a network protocol model of the Neo blockchain and evaluate the Neo network protocol efficiency in terms of reactive message processing in several scenarios, propose protocol enhancements and evaluate resulting performance gain.

Keywords: Gossip protocol · P2P network · Blockchain

1 Introduction

Distributed peer-to-peer systems offer a trustful, scalable and powerful alternative to the traditional client-server architecture. This model assumes that end-users act both like a client and server, sharing resources in a peer style. The P2P approach is devoid the central server failures and performance bottleneck as far as the load balancing issues. However, building efficient, scalable and cheap P2P application level protocol still represents a non-trivial task. At the current moment, the most effective and reliable solution to design P2P protocols for information dissemination in a large dynamic systems is epidemic algorithms. These algorithms are based on the idea of contagious disease spreading and used in a variety set of purposes from distributed systems monitoring to the usage in blockchain broadcasting protocols.

One of the crucial factors for P2P distributed networks functioning which affects the network efficiency is the delay during network messages propagation [7]. This includes message processing delays as far as delays during message broadcasting. Delays have a direct impact on the frame non-delivery probability [5] which is the key performance measure in the distributed P2P networks.

© Springer Nature Switzerland AG 2021
O. Gervasi et al. (Eds.): ICCSA 2021, LNCS 12952, pp. 56–71, 2021.
https://doi.org/10.1007/978-3-030-86973-1_5

Another vital issue is a problem of flooding network with network-wide broadcast messages [5] or the problem of a high traffic load. A plenty of synchronisation-related information in the network results in a low bandwidth protocol utilization efficiency which may slow down the data processing. Consider the case of blockhain networks, where producing a large number of duplicated messages increases block-acceptance time which directly affects transaction processing delays.

Motivation. Epidemic algorithms are widely explored for the case of trustful environments. However assuming that peers may behave arbitrary makes a large part of well-studied algorithms inapplicable which is the case of blockchain networks. Consider the case of blockchain systems, while consensus protocols efficiency and reliability receives a lot of attention, the performance and scalability of broadcast network protocols remain poorly explored. However, the efficiency, fairness and cost of the network protocol directly affects not only the key metrics described above, but also the consensus algorithm operation and even the functionality of the whole blockchain network. In this article we study the main features of the gossip broadcast algorithm which affect the efficiency of the network protocol, and try to improve them.

Paper Outline. Section 2 contains a brief introduction to the gossip-family algorithms classification and several known theoretical results for algorithms complexity and performance related to the subject of study. In Sect. 3 we describe the network protocol used in the Neo blockchain and propose an assumption on the ways it may be improved. Section 4 summarizes several papers related to the push-gossip imitation algorithm modelling. Based on these works we propose an enhanced model of the Neo network protocol which reflects the aspects needed to analyze the protocol efficiency. Section 5 contains the modelling results for both original and enhanced protocols and points out the original protocol weaknesses as far as the enhanced protocol advantages. Based on the modelling information we evaluate the enhanced Neo network protocol efficiency for scenario of blocks propagation over the network. The analysis includes comparing delays during specified message propagation and evaluation of the network bandwidth utilisation used to keep all the participants in sync. We use the data collected to propose several fields for further research in Sect. 6.

2 Related Work

Gossip-based algorithms are well-known and widely explored field of study. One of the most recent works that summarizes known theoretical results for different gossip-based protocols is described in [1]. The study investigates a knowledge-based approach to the gossip problem in a multi-agent system and results in a reach epistemic gossip protocols theoretical basis. Need to mention that gossip algorithms are highly reliable in a probabilistic sense. Paper [7] briefly summarizes known mathematical evaluations of basic gossiping models which includes

probability of atomic infection, proportion of infected process and latency of infection evaluation.

A wide set of approaches for gossip algorithms optimisation can be analyzed. Our study is inspired by the ideas presented in [3–5]. Paper [5] proposes a novel protocol for real-time N-to-N group communication achieving a considerably lower traffic load with higher performance gain in networks with smaller delays. Paper [3] describes simulation model for push-gossip protocol efficiency evaluation. Based on the modelling results the optimal number of peers the current peer should forward the message to (also known as $fan - out$) is calculated as a function of the number of nodes in the network. Paper [4] proposes a robust and fast protocol for weakly-consistent knowledge of process group membership information at all participating processes.

Consider the case of untrustful environment and, more specifically, blockchain networks, the performance and scalability of broadcast network protocols remain poorly explored. The inspiring study of the Hyperledger Fabric gossip layer is presented in [2], taking into account several significant features of messaging in an untrustful environment. However, the field of study still needs to be investigated, and one of the most significant reasons for that is presented below.

Aggregation Problem. Most of the gossip optimisation algorithms (e.g. ones presented in [2] and [3]) require local peer access to the global information about the network. This is a separate problem which is known as the task of aggregation [8] and refers to a set of functions providing access to such distributed system components as network size, average load and uptime, network map, etc. There is a set of existing approaches for gossip-based information aggregation over distributed P2P networks which can be divided into two groups. The first one uses epoch-based solution to calculate a global estimation from the initial set of predefined local values [8]. Network members are exchanging with the aggregated information by iterating over aggregation epochs; once the estimation converges system-wide, a new epoch is started with fresh initial values which makes the algorithm robust to the dynamic estimations aging. The other approach uses continuous epoch-less data aggregation [10] and considers the local measurements of the nodes in every round. This approach does not require initial measurements and performs aggregation and inter-node exchange rounds continuously.

However, these approaches rely on the information veracity and are not able to cope with Byzantine failures with nodes behaving arbitrarily [8]. Thus, some extra assumptions and constraints are needed to apply these algorithms in untrustful distributed systems. One of the ways to handle this problem is malicious nodes detecting [9] and excluding their data from the aggregation service results. However, this issue is outside of the paper scope, but should be taken into account while applying mentioned algorithms in untrustful distributed systems.

3 Neo Network Protocol

Neo is an open-source community driven blockchain platform [12]. It was founded by Da Hongfei and Erik Zhang in 2014 under the name Antshares and then has grown into a first-class smart contract platform. The Neo blockchain is based on the dBFT consensus algorithm providing one block finality and broadcast network protocol based on the gossip model. In this section we describe the current implementation of the Neo blockchain epidemic network protocol. We also summarize the impact of the gossip protocol to a consensus protocol functionality and the whole network operation.

Gossip protocol is used in Neo for a set of purposes: building and maintaining local peer view on a network structure, consensus-related message exchange, new peers discovering, network metadata exchange (i.e. current chain height and etc.) and, finally, blocks, transactions and other payloads dissemination. In this study we mostly focus on the blocks propagation from the consensus service to all peers because the network protocol mechanism is almost the same for all payload messages.

3.1 Gossip Protocol Description

Consider the case of blocks propagation, the beginning of block dissemination process is the moment of block's appliance by the consensus service. Consensus nodes then receive, validate and store the block built by a consensus service and forward it to a set of connected peers. There are three main phases in the Neo gossip algorithm which define the rules of blocks propagation over network: *push*, *pull* and *recovery*. Below we describe these phases with focus on their strengths and weaknesses.

Push. Neo uses infect-and-die push model [7] in which peer tries to contaminate others for only one gossip round and then stops. Speaking more technically, the new block is firstly being put into the queue, validated and stored in a local database. And only after the new block is persisted to a local storage iterating over all peers with sending them the new block digest occurs. Current protocol implementation does not have $fan - out$ setting, instead peer either forward messages to all connected peers or does not forward at all. The number of peers to keep an open connection with can be configured by min_{peers} and max_{peers} settings. Thus, these settings directly affect not only the network connectivity level, but also the traffic load which can definitely be improved.

It should also be mentioned that Neo uses "Inv-GetData" model for all payload-related messages including blocks. It assumes that push phase has three steps: firstly, the payload hash (the short unique payload digest) is sent to a connected peer with the help of Inventory command which contains the hash (or hashes). Then if the peer doesn't yet have the payload with the digest specified it responds with the GetData command containing the requested hashes. Finally, the peer that initiates dissemination forwards the payload itself. Such

push model allows to avoid unacceptable communication overhead while passing heavy payloads, e.g. blocks, allowing to ensure that large payload messages will be passed only $n + o(n)$ times.

Pull. The pull phase is used by peers to receive the blocks they didn't receive during the push phase. At each time interval t_{ping} a peer sends Ping message containing current local chain's height to all its connected peers. The Pong response contains a connected peer's height, so if some of the peers are higher than the pull phase initiator, then the new blocks will be requested by the GetBlockByIndex command. A default t_{ping} interval is set to be 30 s. If the peer receives a new block via the pull request, it will be handled as described in the push phase (the Inventory command containing the block hash will be set to all connected peers after the block is successful persisted to a local storage).

Recovery. Peers use the recovery phase to restore the chain after a long out-of-sync period or for the first time they join the network. During this phase peers exchange meta information including the network magic and current chain's height and request batches of missing blocks by GetBlockByIndex command from those peers which have the latest chain state. This mechanism is running at the very beginning of the connection establishment and is similar to the pull phase. Consider the real stable network functioning, this phase does not affect blocks propagation latency, thus won't be taken into account.

3.2 Impact on the Network Operation

The network protocol described above has a large impact on the whole network operation process which can be described in two ways. Firstly, the protocol directly affects network bandwidth consumption level which is one of the key metrics for network efficiency evaluation. Heavy payloads and a lot of sync-related messages result in a high traffic load which creates load-balancing issues and contention. Moreover, a plenty of large payloads in the network results in a higher broadcast protocol operation cost.

Another vital aspect concerns the byzantine fault-tolerant consensus protocol. The dBFT consensus protocol Neo is operating on requires all consensus nodes to have the same set of transactions to accept a new block. Thus, one of the most priority task of the network protocol is to keep all the consensus nodes in sync. However, the consequence of a high network traffic load is a slow transaction dissemination process which leads to delays and timeouts in the consensus service, i.e. slower blocks producing.

While a lot of attention is payed to the consensus protocol performance improvements, the performance and scalability of the network protocol remain poorly explored. We have showed that the broadcast protocol directly affects overall blockchain network operation, so it's important to keep an eye on the efficiency and cost of the gossip protocol used for peers communication.

4 Enhanced Protocol

One of the main advantages of the Neo broadcast model is that all phases are decoupled from each other and can be modified independently. It also allows to completely resynchronize pull, push and recovery so that cheaper push operations can be run more frequently than expensive pulls.

From the other hand, the Neo push model has several significant drawbacks. Inability to tune f_{out} without affecting the network connectivity level results in an unwanted overhead of the network utilisation level. Infect-and-die algorithm has relatively low probability of delivering block to all peers, which also results in a high bandwidth consumption.

Considering the network aggregation problem and assuming the presence of malicious nodes in the network, the ways of protocol optimisation are desired to improve information dissemination process with the minimal membership maintenance and without an assumption on the underlying network structure.

4.1 Proposed Improvements

Based on the Neo protocol drawbacks described above along with the most effective ways to optimise the gossip protocol from related works, we propose several protocol enhancements described in the next subsections.

Introducing f_{out} Setting: Current Neo protocol scheme assumes that peer either transmits received blocks to all peers it's connected to or doesn't transmit blocks at all. Although maximum and minimum peers number can be configured for Neo node, it still lacks the f_{out} setting, i.e. the number of randomly selected peers the current peer should forward the message to. Missing f_{out} setting leads to network flooding with the sync-related messages for large max_{peers} values and a lack of connectivity resulting in a slower block processing for small max_{peers} values. At the same time, proposed f_{out} setting can have considerably lower value than the node's maximum connected peers constraint, which allows to vary network traffic load irrespective to the nodes connectivity rate. As shown in the [2], given a relatively small f_{out} it is possible to provide the desired level of frame non-delivery probability for a fixed number of gossip rounds. This improvement allows to decrease the traffic utilisation level keeping probabilistic reliable frame non-delivery rate at the low level.

Randomisation of the Source Gossiper: According to the Neo consensus protocol, blocks produced by consensus service are firstly passed to nodes picked from the standby committee list which has quite a limited capacity. These leader nodes are those who start the dissemination, thus their network utilisation is up to max_{peers} times higher then that of regular peers. To smooth this difference out, we'll set $f_{out}^{leader} = 1$. We expect this improvement to distribute the initial dissemination load over all regular peers.

Using Infect-Forever Model for the Push-Phase: Neo protocol currently uses infect-and-die model for the push-phase of the gossip algorithm. We propose to replace it with the infect-forever model [6] modified to use a stop condition. It assumes that peers forward blocks whenever they receive them until the stop condition is satisfied. To implement the stop condition, the counter r initialized to 0 is attached to each new block. For the first time peer receives this block, it increments the counter and continues block transmission with the incremented counter in a standard way. When the peer receives the same block next time, it just forwards the block to f_{out} peers selected randomly from a set of the connected peers. Dissemination of the block stops when r reaches preconfigured time-to-live TTL value which is the same for all peers in the network. This improvement is highly dependant on the f_{out} parameter and in combination with f_{out} is expected to balance load between peers initiating gossip rounds and the others.

To summarize, the proposed improvements are presented in Table 1.

4.2 Choosing the Parameters

Given a fixed probability of imperfect dissemination p_e, i.e. the probability that block does not reach all peers during the push phase, depends on n, f_{out} and TTL, these parameters can be calculated very precisely. According to estimations for p_e, n, f_{out} and TTL presented in the [2], for our experiments with $n = 100$ peers and target $p_e = 10^{-6}$ we set up a configuration with $f_{out} = \lfloor \ln(n) \rfloor = 4$ and $TTL = 9$.

Table 1. Proposed protocol improvements.

Improvement	Explanation	Effect
Introducing f_{out} setting	Transmitting messages to a predefined number of peers instead of sending it to all the connected peers	Ability to control network bandwidth consumption and decrease the number of messages in the network
Randomisation of the source gossiper	Restrict $f_{out}^{leader} = 1$ to randomise peer initiating gossip round	Alleviation of the leader peers traffic burden
Using infect-forever model for the push-phase	Forward blocks whenever they are received until the stop condition is met	Load balancing over all peers in the network

5 Experiments and Evaluation

We evaluate proposed NeoGo gossip module improvements and compare the results with the original version of module implementation. Inspired by the [2] we are mostly focusing our evaluation on the three key metrics described above: block non-delivery probability, blocks propagation latency and traffic load (also called the bandwidth utilisation) and how these metrics correlate with each other.

5.1 Setups

As many other Neo projects, NeoGo currently is on its way to Neo 3.0 - the next evolution of the blockchain with new architecture, native contracts, and an improved economic model. While developing is continuing, it's important to reveal places to be improved and optimized, thus we use latest intermediate Neo3 release of NeoGo v0.91.0 for experiments and evaluation.

We base our experiment environment on the NeoBench [11] - an open-source benchmark tool for Neo nodes stress testing. We set up a network of 101 peers where one peer is an RPC node accepting transactions from the client, four peers are consensus nodes building and approving blocks, and the rest of peers are regular nodes storing blocks and disseminating network messages. All nodes are deployed on a cluster equipped with 2 x Intel® Xeon® CPU E5-2690 v4 at 2.60 GHz and 256 GB of RAM and running inside Docker containers sharing the same proportion of CPU cycles and grouped into the network. Benchmark tool with an RPC client concurrently sending pre-generated transactions to the RPC node and gathering statistics is running inside a separate Docker container.

As a default node setup for each experiment we use blocks dump moving all NEO and almost all GAS tokens from default multisignature account to a simple one. We use the following network configuration settings for the experiments: each block contains 50 transactions at max, the interval which should pass between two successive blocks equals 2 s. For each experiment we generate and concurrently push to the network 50,000 transactions, each of them transfers one NEO from the simple account back to the same account. As a result, a full block containing 50 transactions is created about every 2 s which results in 1,000 blocks of 4320 byte per block.

5.2 Original Protocol Evaluation

To compare proposed protocol improvements with the current implementation, we firstly have tested the original NeoGo protocol with the $max_{peers} = 10$ for each node. This value is considered to be optimal for the experimental network of 100 peers because too large values lead to the network flooding with synchronisation-related messages and traffic wasting; too small values are impractical for real networks due to the lack of connectivity between peers.

Figure 1 shows the latency at the peer level (LPL) by measuring the time peer takes to receive each block starting from the beginning of its dissemination. The start time is defined by the moment when block is received by one of the consensus nodes from the consensus service. Thin lines correspond to peers; colored markers highlight the slowest, the fastest and median peers, i.e. peers received all the blocks with highest/lowest/median overall latency respectively. Figure 2 illustrates the latency at the block level (LBL) by showing how much time block takes to reach each peer starting from the beginning of block's dissemination defined by the same value as the one calculated for LPL. Each thin lines corresponds to a separate block; the slowest, fastest and median blocks are highlighted with colored markers, where the block delivered to all the peers with

the highest latency considered to be slowest and vice versa. Both figures are probability plots and shown in a logarithmic scale based on logistic distribution for OY axis for better tails latency focusing.

From Fig. 1 we can say that about 94% of the blocks were delivered to peers for less than 2 s after the beginning. However, things get more interesting at the upper part of the plot where some peers take up to 160 s to receive the rest 6% of the blocks. Likely, these are the most remote from the consensus nodes peers which use pull-phase of the gossip algorithm to fetch missing blocks. The similar summary can be drawn from the block-level latency analysis of the Fig. 2. Up to 65% of all peers have insignificant latency during blocks receiving, but a set of the slowest blocks takes quite a long time (up to 160 s) to reach the rest 35% of all peers. That is the first field for the future research: try to make the block dissemination process more uniform and accelerate transmission of the backward blocks.

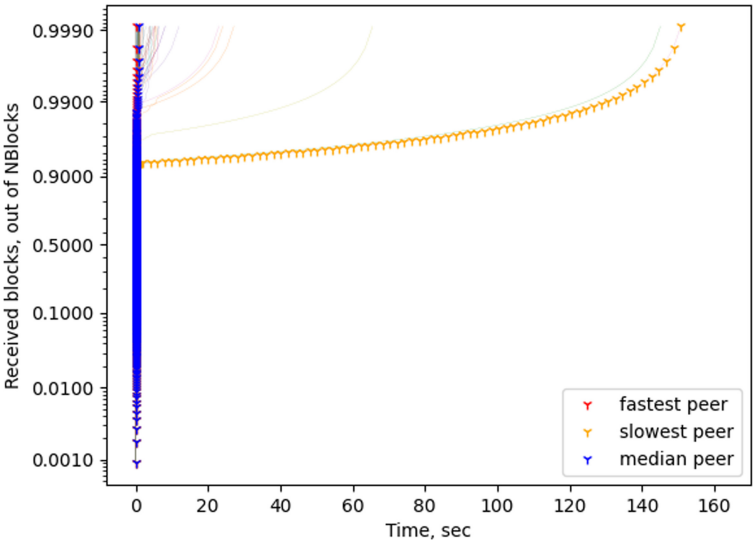

Fig. 1. Latency at the peer level using the original gossip module of NeoGo.

Figure 3 illustrates the network traffic load not only throughout the whole experiment, but also a few moments before and after it. That's done to compare the network utilisation in the ideal state with the overloaded one. Two lines correspond with the average consensus and regular peers bandwidth consumption measured by the size of sent messages per second. For readability purposes the bandwidth consumption was gathered only for blocks-related messages with an interval of 10 s. The benchmark service has started pushing transactions to the network since $\approx$ 400 s from the start and ends after $\approx$ 2400 s from the start. Thus, the figure shows significantly low block-related traffic load $\approx$ 0.7 KB/s for

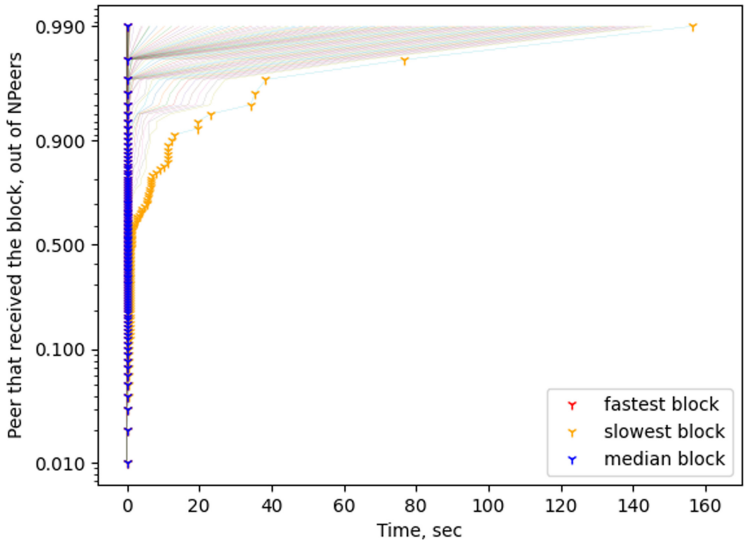

Fig. 2. Latency at the block level using the original gossip module of NeoGo.

regular peers and ≈ 2.7 KB/s for consensus peers when the network is in an ideal state during empty blocks processing. During the experiment average consensus peer bandwidth consumption oscillates from 15 to 18 KB/s which is about 6 times higher than the network load of the regular peer. It occurs because once the consensus node receives a new block from the consensus module, it transmits the block to all connected peers while regular nodes follow infect-and-die dissemination scheme. Such a considerable difference illustrates an importance of gossip-based protocols bandwidth consumption optimisation, which is the other field for optimisation-related research. It includes decreasing the number of messages in the network and balancing the network bandwidth load.

5.3 Proposed Changes Efficiency Evaluation

We tested the performance of the improved protocol and compared it to the original results. Each of the proposed improvements was tested separately as far as the combination of the infect-forever model for the push phase and introducing f_{out} parameter (as mentioned in Sect. 4.2, these settings are highly connected).

Firstly, the protocol with $f_{out}^{leader} = 1$ improvement was evaluated. Figures 4, 5 and 6 respectively show the latency at the peer level, latency at the block level and bandwidth consumption for the enhanced protocol. While the original protocol takes up to 160 s to deliver last 6% of blocks to all peers, the proposed improvement shows twice times better result for LPL metric (Fig. 4) and takes ≈ 74 s to disseminate 3% of the slowest blocks to all peers while the rest 97% of blocks are delivered for less than two seconds. As for the bandwidth utilisation, significant improvement was reached for the consensus peers traffic load

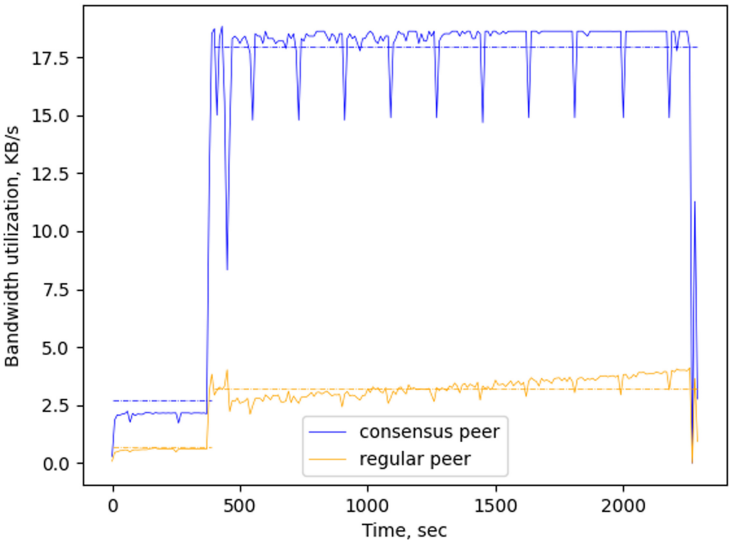

Fig. 3. Bandwidth utilisation using the original gossip module of NeoGo.

shown in Fig. 6 which is decreased by more than 80% and is even lower than the regular peers bandwidth consumption. At the same time, network utilisation of the regular peers remains almost the same as with the original protocol and oscillates around 2.7 KB/s value. We expect this improvement to become more significant with larger block size, as blocks-related traffic predominates over other sync-related messages.

We evaluated the performance gain of introducing f_{out} parameter and infect-forever model for the push phase applied separately. However, the results of these experiments are quite controversial. Applying the $f_{out} = 4$ setting allows to decrease both consensus and regular peers average bandwidth utilisation to $\approx$ 16.3 and 2.3 KB/s respectively which is about 9% and 28% lower than the reference values. At the same time, maximum latency at both peer and block level was decreased by less then 30 s from 162 to 138 s while median latency remains almost the same. Experiment with infect-forever model for the push phase with $TTL = 9$ didn't show the substantial LPL and LBL efficiency gain. Moreover, without proper f_{out} setting this improvement significantly increases average network traffic utilisation, so that f_{out} limitation expected to compensate this effect.

Finally, we tested the enhanced protocol with combination of $f_{out} = 4$ and $TTL = 9$ settings. Figures 7 and 8 are LPL and LBL plots respectively. The enhanced protocol allows to reach significant median latency decrease from 1.13 to 0.47 s for peer-level latency and from 0.19 to 0.15 s for block-level latency which is even better than source gossiper randomisation shows. At the same time, average network traffic utilisation was decreased down to 11.3 and 2.7 KB/s for consensus and regular peers respectively, which is effectively $\approx$ 36% and 15%

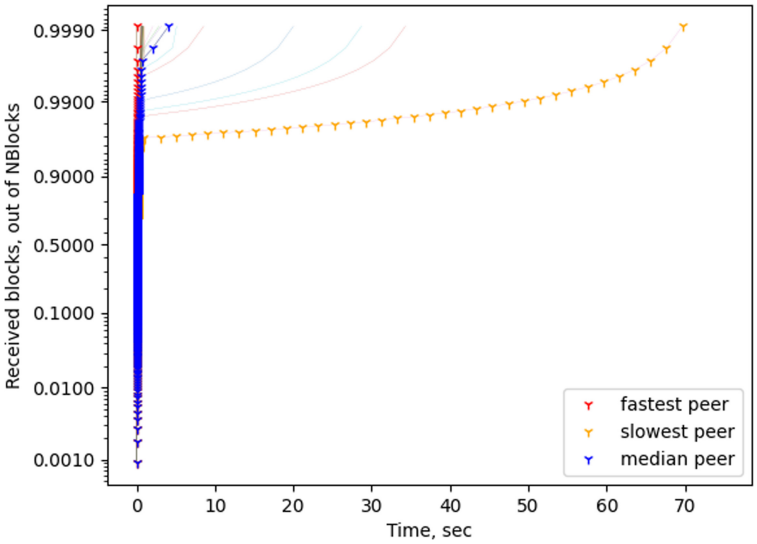

Fig. 4. Latency at the peer level using NeoGo module with the source gossiper randomisation.

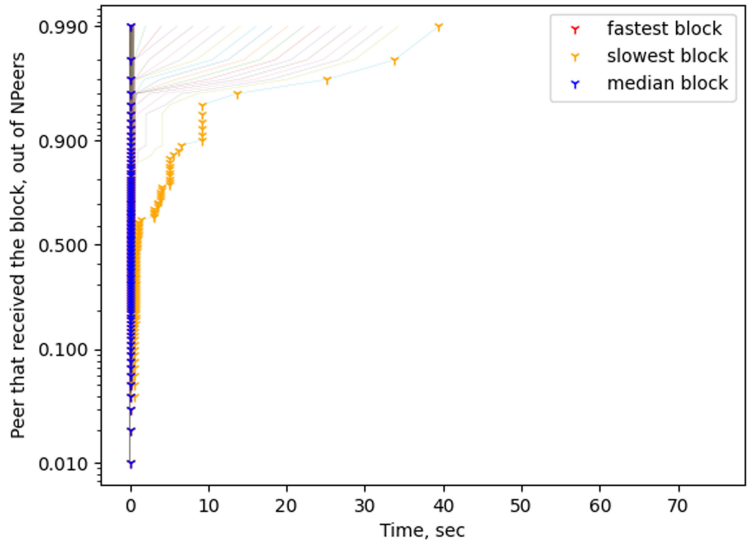

Fig. 5. Latency at the block level using NeoGo module with the source gossiper randomisation.

lower than the reference values. So the combination of the infect-forever model for the push phase and introducing f_{out} parameter results in a more uniform network traffic distribution over the set of regular peers and noticeably lower bandwidth utilisation by the consensus peers (Fig. 9).

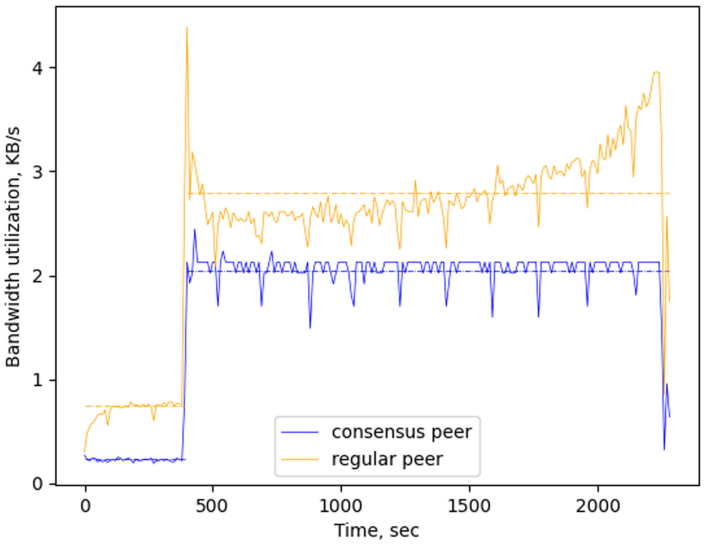

Fig. 6. Bandwidth utilisation using NeoGo module with the source gossiper randomisation.

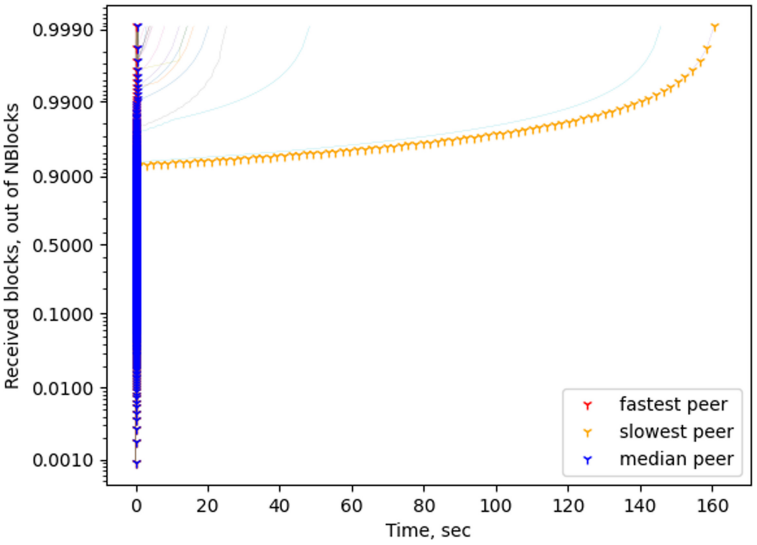

Fig. 7. Latency at the peer level using NeoGo module with $f_{out} = 4$ and $TTL = 9$.

Although the efficiency gain obtained from the last experiment is not such substantial as the one got from the initial gossiper randomisation enhancement, it still perfectly balances network traffic load between regular and consensus peers and reduces median block-level and peer-level latency. Thus, set of protocol

enhancements considered in the work, should be combined together to achieve better efficiency improvement in both terms of message delivery latency and bandwidth utilisation balancing.

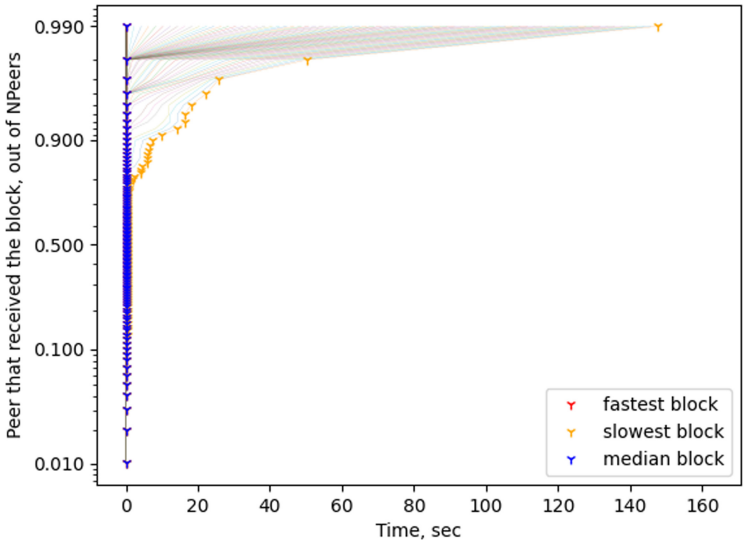

Fig. 8. Latency at the block level using NeoGo module with $f_{out} = 4$ and $TTL = 9$.

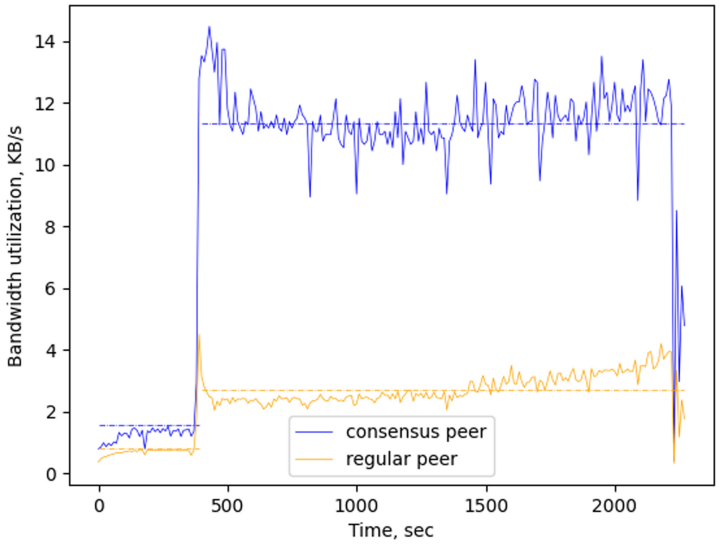

Fig. 9. Bandwidth utilisation using NeoGo module with $f_{out} = 4$ and $TTL = 9$.

6 Conclusion and Future Work

In this work, we enhanced the Neo gossip layer. Decreasing its latency allowed to maintain a reliable basis for consensus protocol functioning. The enhanced protocol also shows significant network bandwidth consumption improvement which results in avoiding load-balancing issues and decreasing the cost of broadcast protocol operation. We propose the following directions for the further research.

Current work considers the case of an "ideal" network state without network packet losses, data corruption and message transmission delays caused by imperfect inter-node connection. This may imply better target metric results compared to the case of network deployed on the production environment. Although the experiments set up allow to estimate the expected efficiency gain, we still need to evaluate the proposed protocol enhancements in a real physically-distributed network. This can be reached either by network packets corruption and message losses modelling with the help of network simulation instruments or by evaluation of the target metrics in the production environment.

Another vital assumption the paper is based on is a known predefined size of the network. Although the f_{out} and TTL estimations vary slowly with number of peers and can be stored in a lookup table, each peer still needs a reliable way to estimate the number of nodes in the network. That's the case of the aggregation problem mentioned in Sect. 2, when considering the untrustful environment factor, a peer can not rely on the information received from its neighbours, thus standard aggregation algorithms are not applicable.

Finally, we did not consider malicious nodes trying, for instance, to inhibit the dissemination process by dropping blocks received from the other peers. The enhancements robustness to such behavior still needs to be evaluated, although given relatively low probability of imperfect dissemination, it shouldn't be a problem for the proposed algorithms to cope with adversarial peers. So the impact analysis and possible countermeasures are to be investigated.

References

1. Apt, K., Grossi, D., Hoek, W.: When are two gossips the same? Types of communication in epistemic gossip protocols (2018). http://arxiv.org/abs/1807.05283
2. Berendea, N., Mercier, H., Onica, E., Riviere, E.: Fair and efficient gossip in hyperledger fabric. In: 2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS), Singapore, Singapore, pp. 190–200 (2020). https://doi.org/10.1109/ICDCS47774.2020.00027
3. Vanin, A., Bogatyrev, V.: Push-gossip protocol efficiency with network topology propagation. In: Proceedings of the 10th Majorov International Conference on Software Engineering and Computer Systems (MICSECS-2018), CEUR Workshop Proceedings, vol. 2344 (2019)
4. Das, A., Das, A., Gupta, I., Motivala, A.: SWIM: scalable weakly-consistent infection-style process group membership protocol. In: Proceedings of 2002 International Conference on Dependable Systems and Networks (DSN 2002), pp. 303–312 (2002)

5. Luk, V.W.H., Wong, A.K.S., Lea, C.T. et al.: RRG: redundancy reduced gossip protocol for real-time N-to-N dynamic group communication. J Internet Serv. Appl. **4**, 14 (2013). https://doi.org/10.1186/1869-0238-4-14
6. Koldehofe, B.: Simple gossiping with balls and bins. Studia Informatica Universalis. **3**, 43–60 (2004)
7. Eugster, P.T., Guerraoui, R., Kermarrec, A., Massoulie, L.: Epidemic information dissemination in distributed systems. Computer **37**(5), 60–67 (2004). https://doi.org/10.1109/MC.2004.1297243
8. Jelasity, M., Montresor, A., Babaoglu, 0.: Gossip-based aggregation in large dynamic networks. ACM Trans. Comput. Syst. **23**(3), 219–252 (2005). https://doi.org/10.1145/1082469.1082470
9. Jelasity, M., Montresor, A., Babaoglu, O.: Detection and removal of malicious peers in gossip-based protocols. S.O.S., Bertinoro, Italy, InFuDiCo II (2004)
10. Rapp V., Graffi K.: Continuous gossip-based aggregation through dynamic information aging. In: 22nd International Conference on Computer Communication and Networks (ICCCN), pp. 1–7 (2013). https://doi.org/10.1109/ICCCN.2013.6614118
11. NeoBench. https://github.com/nspcc-dev/neo-bench
12. Neo blockchain platform. https://neo.org/