A bandwidth-aware algorithm for solving topology mismatch problem in peer-to-peer networks utilizing the combination of learning automata and X-BOT algorithm

Maryam Haji Ghorbani Dolabi
Dep. Engineering and Information Technology
Islamic Azad University
Qazvin, Iran
Maryam_h.ghorbani@yahoo.com

Mohammad Reza Meybodi
Dep. Engineering and Information Technology
AmirKabir University of Technology
Tehran, Iran
mmeybodi@aut.ac.ir

Ali Mohammad Saghiri
Dep. Engineering and Information Technology
AmirKabir University of Technology
Tehran, Iran
A_m_saghiri@aut.ac.ir

***Abstract*—Peer-to-peer networks construct an overlay network above underlying networks. These networks are classified as structured and unstructured networks. Unstructured overlay networks are characterized by a relaxed topology where neighboring associations are random in nature and oblivious to the properties of the underlying network which leads to topology mismatch. This usually results in the overlay networks owing several sub-optimal links leading to latency, redundant traffic, wasting bandwidth and finally degrading the performance of the network. So designing an appropriate algorithm for topology adapting can considerably affect the efficiency as much as performance. In this paper, an adaptive and smart algorithm for solving topology mismatch will be proposed by considering the bandwidth of each peer and combining learning automata and X-BOT algorithm. In proposed algorithm, neighboring association constitutes based on bandwidth and by utilizing the information provided by learning automata. The information about peer's bandwidth is kept in its local table used periodically for making decision about appropriate links. Simulations has demonstrated that the proposed algorithm would improve the efficiency through decreasing end to end latency, traffic and appropriate use of bandwidth.**

Keywords— Peer-to-peer networks; unstructured overlay networks; topology adaptatio; X-BOT algorit; learning automata.

I. INTRODUCTION

Peer-to-peer networks are application-level networks built on top of underlying networks. The properties and nature of the overlay network topology have a high impact on the performance of p2p services and applications executed on top of them. Overlay networks usually belong to one of two classes, according to the mechanisms employed to build and maintain the overlay topology. These classes are named structured and unstructured [1]. Structured overlay networks impose constraints to the neighboring relationships that may be established among peers. These neighboring relations are required to follow a global coordination strategy. As a result, the topology is completely under control and follows specific algorithms. In contrast, unstructured overlay networks are characterized by a much more relaxed topology having peers establishing neighboring relations among them in an uncoordinated and random fashion which form a complete random graph. Due to this random nature, overlay topology is oblivious to the properties and performance metrics of underlying networks resulting in a topology mismatch between the overlay and the underlying network [2].

Topology mismatch leads to increasing average end-to-end delay and traffic as well as wasting resources such as bandwidth which degrade the performance of overlay networks. Due to the large scale of these networks and high churn (individual peers join, leave and fail concurrently) causing widespread and continuous changes in the topology, we need a self-adaptive and smart algorithm so as to accelerate sending and receiving of information and prevent the waste of network resources [3]. We determine to optimize the overlay topology intellectually, with respect to the churn in overlay as well as changes that occur in the underlying networks, so that the end to end delay and cost of transferring information will be reduced.

So far, several topology management algorithms like Gia [2], Narada [2], GoCast [4], T-Man [5] and X-BOT[6] have been proposed to improve the performance of p2p networks. Gia operates on top of an unstructured overlay network which is biased as the system evolves to give preference to neighboring associations where one of the peers is a high capacity node to improve the performance of one-hop replication of resource indexes. The biased topology is then leveraged to efficiently route queries primarily to high capacity nodes. Narada includes self-organizing protocols to construct and maintain overlay networks. The approach is based on applying a utility function periodically to some peers. The output of the utility function is used to locally decide concerning joining/disjoining peers to overlay. Since Narada is targeted

at small and medium scale systems, it operates using full membership information, therefore scalability is poor. GoCast includes mechanisms to bias the overlay topology maintaining symmetric partial views supported by TCP connections. It builds an overlay networks maintain both close and distant neighbors while balancing node degree in such a way that the degree of nodes converge to a pre-established value. The protocol relies on complex mechanisms. T-MAN is a generic topology management scheme for overlay networks which can evolve a given overlay topology to a desired target topology. this is achieved by having neighbors periodically exchanging their partial views. Both nodes update their vies by merging these views and selecting the best nodes. this scheme does not aim at protecting relevant properties of the original overlay and stability of in-degree of nodes during optimization. X-BOT explores a bias approach to manage the topology. According to a given criteria the topology will be optimized. The protocol operates iteratively in decentralized fashion utilizing partial views and evolves the topology to more efficient configurations while protecting the connectivity and node in-degree during the convergence process. Among the algorithms proposed, X-BOT could better match the topology of unstructured overlay and underlying network, but so far no algorithm based on X-BOT has been provided with the knowledge of bandwidth[7].

In this paper, a new version of X-BOT algorithm is presented in which, by using learning automata, the topology of p2p overlay network has been made intelligent and adapted to the underlying topology according to bandwidth metric. Meanwhile it manages to easily reconfigure itself in face of churn and preserves its random nature. By utilizing learning automata in the network, every peer will be informed about the situation of its peer and also make decision about neighboring relationships based on received feedback.

This paper is organized as follow: Section II describes X-BOT algorithm. In section III we have an overview about learning automata as the fundamental strategy in proposed algorithm. Our algorithm will be explained in section IV. In section V we validate and evaluate the benefits that can be extracted from the contributions proposed in this paper. In section X, we come up with conclusion and result summary.

II. X-BOT ALGORITHM

X-BOT is a protocol that exploits the bias approach for managing the topology of unstructured overlay networks at the overlay layer. Each peer, i. e., peeri periodically performs an optimization round every second. X-BOT employs a class of oracles (local agent) to provide information about the cost among peers for them (more details about oracles will be discussed in sub-section IV). Moreover, peers in X-BOT own two partial views: active view and passive view. Partial views define neighboring association among peers [8]. Active view is a symmetric view is used to support communication among participants and disseminate messages. Active views are sorted descending by link cost. Each peer also maintains a larger passive view k times larger than the active view that rely on local knowledge, i.e., a node makes optimization decisions based on local information regarding the distance to its own peers. A typical optimization round involves 4 peers of the system, and each round is composed of 4 steps, one for each peer that participates in the optimization(X-BOT pseudo-code is shown in appendix I).

Step 1 starts at $peer_i$ with the random selection of a set of peers from the $peer_i$'s passive view (CandidPeers) for executing optimization round. It also selects the highest cost peer (OldPeer) from its active view which is replacing during optimization process. To check if a target peer is a suitable candidate, $peer_i$ iterates over its active view, consulting the oracle to compare the cost of its CandidPeer with the cost of OldPeer. When a suitable CandidPeer is found, $peer_i$ sends an optimization message to CandidPeer, stating its interest to exchanging them in its active view. This step ends with the reception of an OptimizationReply message from CandidPeer or with the suspicion of failure of CandidPeer. If CandidPeer accepts the exchange, then $peer_i$ will add CandidPeer to the active view. If OldPeer is still in its active view, peeri will send a Disconnect message to OldPeer.

Step 2 is initiated at $peer_i$ with the reception of an Optimization message. If peeri does not have a full active view it immediately replies by sending an OptimizationReply message accepting the exchange, otherwise sends REPLACE message to a random peer in its active view.

Step 3 begins with the reception at $peer_i$ of a Replace message. This message explicitly requests $peer_i$ to exchange CandidPeer with OldPeer in its active view. If the active view of peeri is not full, it immediately replies by sending a ReplaceReply message accepting the exchange. Otherwise, $peer_i$ sends Switch message to OldPeer, if there is a gain in the exchange, and replies with SwitchReply message. Naturally, these exchanges will be verified by consulting their oracles.

Step 4 is executed by $peer_i$ upon the reception of a Switch message. After checking all constraints, $peer_i$ sends DisconnectWait and updates its active view.

Furthermore, in an optimization round, if the time of optimization has been over at each step, $peer_i$ starts from step1.

III. LEARNING AUTOMATA

Learning automata (LA) is a machine that can execute a finite set of actions [9,10]. Each action has a certain probability (unknown for the automaton) and is evaluated by the environment. The results of evaluation are sent to automaton as a positive or negative signal as shown in Fig. 1. Each action gets reward by this signal. The aim is to learn to choose the optimal action (i.e. the action with the highest probability of being rewarded) through repeated interaction automata works in feedback connection with a random environment.

An environment is a triple $E = \{\alpha, \beta, c\}$, where $\alpha \equiv \{\alpha_1, \alpha_2, ..., \alpha_s\}$, $\beta \equiv \{\beta_1, \beta_2, ..., \beta_m\}$, $c \equiv \{c_1, c_2, ..., c_s\}$ are an action set with s actions, an environment response set and the penalty probabilities set c containing s probabilities, respectively. Each element c_i of c corresponds to one input of action α_i. Learning automata generates actions α (n). These actions are evaluated by environment. The environment response (β) is sent to learning automata to generate the next generation of actions α (n+1). There are three types of environment based on its response. Whenever β= {β1, β2} is a two members set, environment type is P. In this environment β1=1 is the punishment and β2=0 is the reward. A further generalization of the environment, known as Q-model, allows finite output set with more than two elements that take values in the range [0, 1]. A further step in this direction is the S-model whose responses can take continuous values over the unit range [0, 1]. Learning automata can be classified into two main families: Fixed Structure Learning Automata and Variable Structure Learning Automata (VSLA). We consider a VLSA in our algorithm and it operates as follows:

A VSLA is a quintuple <α, β, p, T> where α and β are the same parameters as the environment's, and $p \equiv \{p_1, p_2, ..., p_s\}$ is the action probability sat, each being the probability of performing every action in the current internal automaton state. The function $p(n+1) = T[\alpha(n), \beta(n), p(n)]$ is the reinforcement algorithm which modifies the action probability vector p with respect to the performed action and received response at the next generation. This automaton operates as follows. Based on the action probability set p, automaton randomly selects an action α_i, and performs it on the environment.

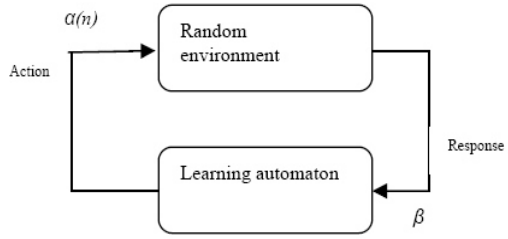

Fig. 1. Interaction between environment and learning automata.

After receiving the environment's reinforcement signal, automaton updates its action probability set based on the following reinforcement scheme; the equation (1) for favorable response, and (2) for unfavorable one. When an action gets reward, its probability (p_i) increases and the probability of other actions decreases.

If β=0

$$p_i(n+1) = p_i(n) + a[1 - p_i(n)]$$
$$p_j(n+1) = (1-a)\,p_j(n) \qquad \forall j, i \quad j \neq i \tag{1}$$

If β=1

$$p_i(n+1) = (1-b)\,p_i(n)$$
$$p_j(n+1) = (b/r-1) - (1-b)\,p_j(n) \qquad \forall j, i \quad j \neq i \tag{2}$$

When a and b are rewarded and penalty parameters. When a=b, automaton is called "L_{RP}". If b=0 and 0<b<<a<1, the automaton is called "L_{RI} "and "L_{Rep}", respectively.

IV. The Proposed Algorithm

In this section, a new approach to solving topology mismatch in p2p overlay networks, using learning automata is proposed. To this end, first, an algorithm should be chosen. In this paper, X-BOT algorithm is used to be optimized according to bandwidth by learning automata (LA-BOT).

In the proposed algorithm, each peer considers as a learning automata. Moreover, its action is selecting one of the neighbors according to action probability vector. Updating the action probability vector is performed by using the information provided by oracles and executing reward/penalty function over the vector. Similar to X-BOT, our algorithm (LA-BOT) consists of four steps, as follows:

Step 1 aims to select a set of peers from its passive view to initiate optimization (as X-BOT).

Step 2 starts after selecting peers with appropriate bandwidth for exchanging links. The information provided by oracle is used to choose candid peers (as X-BOT).

Step 3 is responsible for selecting the peer which has the capability of increasing bandwidth according to the changes happened in topology after two previous steps. According to the feedback from environment, LA executes reward/penalty function on probability vectors of candid peers. Updating the probability vectors, increase the probability of choosing a high bandwidth peer.

Step 4 updates the active views of exchanged peers (likewise X-BOT).

Before addressing LA-BOT algorithm, we present an overview on the fundamental concepts used in our algorithm.

A. Oracle

The class of oracles, settled in each peer, employed by LA-BOT relies on local information. Oracles are components that export a gerlinkcost($peer_p$) interface, which returns the link bandwidth between the invoking peer and the given target node p in the system. Each peer makes optimization decisions based on local information regarding its bandwidth.

B. Probability automata table

Each peer maintains a list containing probability vector to its neighbors. It updates based on feedbacks receiving from environment.

C. Message design.

According to the algorithm procedure, different messages exchange among peers in specific phase. Table (1) indicates the framework of these messages. In all the messages peer's identifier for communicating, is specified by unique information, in LA-BOT we use {IP:Port} for distinguishing the target peer.

TABLE I. MAIN FRAMWORK OF MESSAGES AT OVERLAY LAYER

Message name	The fields of message
Optimization	MessageType, CandidPeer, OldPeer, peeri, TimeStamp
OptimizationReply	MessageType, Answer, OldPeer, DisconnectPeer, CandidPeer, TimeStamp
Replace	MessageType, OldPeer, peeri, CandidPeer, TimeStamp
ReplaceReply	MessageType, Answer, OldPeer, peeri, CandidPeer, DisconnectPeer, TimeStamp
Switch	MessageType, peeri, CandidPeer, DisconnectPeer, TimeStamp
SwitchReply	MessageType, Answer, peeri, CandidPeer, OldPeer, TimeStamp
DisconnectWait	MessageType, OldPeer, peeri, TimeStamp
Disconnect	MessageType, OldPeer, peeri, TimeStamp

Considering the complete similarity in step one, two and four to X-BOT, we explain the third step for LA-BOT algorithm in which learning automata activates and perform smart selecting based on bandwidth.

Step three of LA-BOT initiates by peeri through receiving REPLAE message and terminates by sending ReplaceReply. REPLACE message explicitly asks $peer_i$ to exchange CandidPeer with OldPeer according to information provided by local oracles. If the active view of $peer_i$ is not full, it immediately adds the CandidPeer to its active view and sends ReplaceReply to CandidPeer. Otherwise, it enters the learning phase. In this phase, learning automata is awaken in peeri and decides about appropriate action. Following the chosen action, the neighbor will be evaluated by learning automata as well as computing reward. Consequently, learning automata updates its probability vectors. As the final process, learning automata compares the bandwidth of CandidPeer and OldPeer consulting the oracles. If the bandwidth is improved, it will award related probability vector else penalty will be granted. Then, the reward function updates probability vector table. Considering updated probability vector, $peer_i$ determines about the replacement and sends ReplaceReply to related peer, as the step three of LA-BOT pseudo-code is illustrated in Fig. 2(For detailed study about X-BOT pseudo-code, see appendix I).

```
//phase 3: //learning phase
upon Receive(REPLACE, OldPeer, peer_i, CandidPeer) do
    If= isBetter(peer, OldPeer)
//# reflects better peer according to the cost retrieved from oracles.
        then
            Send(REPLACEREPLY, CandidPeer, false, peer_i, OldPeer,
                myself);
        else
            LA.choose an action;
            Set selected_neighbor using action selected by LA
            Gather information of delays from candidate peers
            If Bandwidth and latency after switching improved
              (oldcost –newcost>0)
                then
                    β ← 1;
                else
                    β ← 0;
            EndIf;
            Update LA using β;
    EndIf;
End;
upon Receive(SWITCHREPLY, answer, peer_i, CandidPeer, OldPeer)
  do
        If answer then
            ActiveView ← ActiveView \ {CandidPeer};
            ActiveView ← ActiveView ∪ {OldPeer};
        EndIf;
        Send(REPLACEREPLY, answer, peer_i, myself);
End;
```

Fig. 2. LA-BOT algorithm in pseud-code, step three.

V. SIMULATION EXPERIMENTS

All the simulation results presented in this paper were produced using OverSim[8] in OMNET++ simulation environment [11]. We conducted extensive experimental evaluations of LA-BOT against Gia, T-MAN and X-BOT via two simulation sets. In the experiments reported here, we use the following scenarios, which allow us to assess the benefits of LA-BOT. First, we perform an evaluation in different conditions between X-BOT and LA-BOT by using latency oracle, and then we extend our experiments to the three above mentioned algorithm according to bandwidth and churn as well.

Our scenario is composed of 500 peers and every 1 second, 100 byte data are sent to random destination. In all experiments we conducted, the time of simulation was set to 100, 200, 300, 400, 500 seconds. To ensure a fair comparison, we consider 500 seconds for network warm up.

A. First experiment

The aim is to investigate LA-BOT and X-BOT considering the number of peers do not participate [6] in optimization. This number is fundamental as preventing partitioned overlay. Fig. 3, depicts that existing only one unbiased peer provides the best condition for improving the topology. The result in Fig. 4, has shown the relative

advantage of LA-BOT when considering delay average to numbers of unbiased peers.

B. Second experiment

In this set of experiments we are mainly concerned with evaluating the effect of different algorithms in topology adaptation. Interestingly, LA-BOT has the much better capability to adapt the overlay network to the underlying network as shown in Fig.5. This is due to the fact that operation of LA-BOT has implicitly optimized the topology according to well-defined information provided by probability vector tables along with local oracles. Fig.6, depicts end-to-end latency among the algorithms and noticeable performance of proposed algorithm.

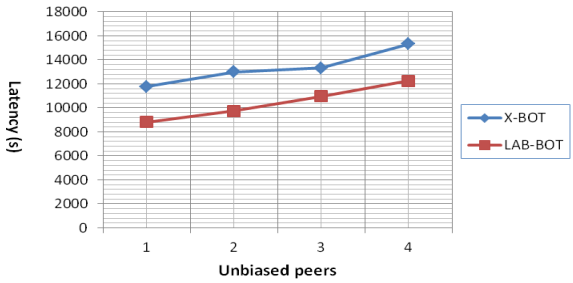

Fig. 3. Latency/Unbiased peers.

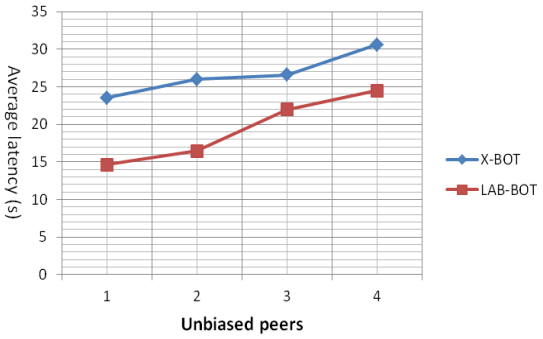

Fig. 4. Average latency/Unbiased peers.

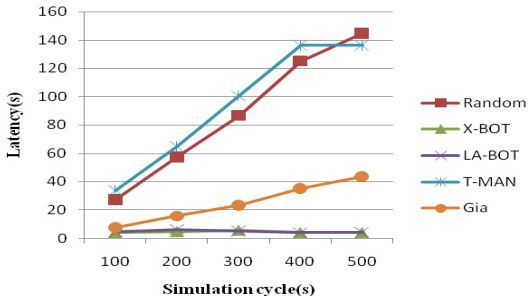

Fig. 5. Comparing average latency.

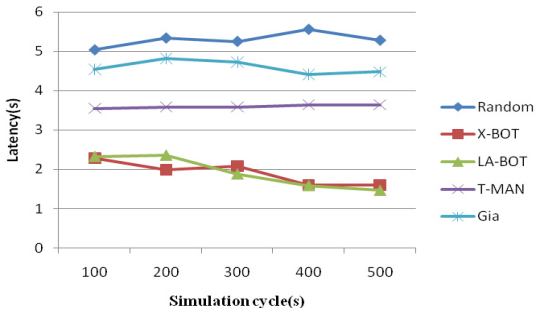

Fig. 6. End-to-end latency among different algorithms.

C. Third experiment

We evaluate bandwidth metric through focusing on smart topology adaptation by using learning automata. Experiments were conducted by executing LA-BOT for peers with random and primary 1 to 100 mbps bandwidth and observing the overlay and its final configuration. Fig. 7, plots the evolution of the overlay bandwidth, Fig. 8, reports on the end-to-end bandwidth. LA-BOT shows an optimized and high bandwidth which is favorable for live streaming application.

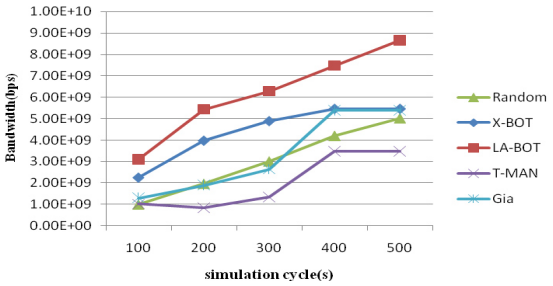

Fig. 7. Total bandwidth.

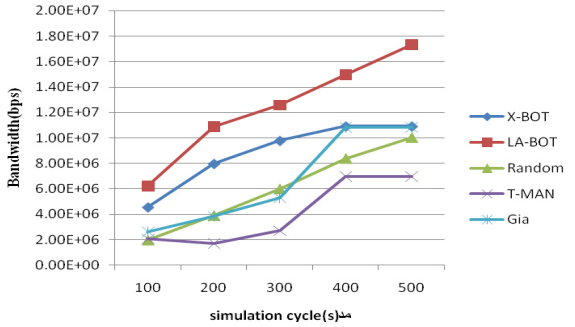

Fig. 8. Average bandwidth.

VI. Conclusions

The paper has proposed, developed, and evaluated a novel and adaptive bandwidth-aware algorithm for solving topology mismatch in peer to peer unstructured overlay network and underlying network utilizing learning automata and X-BOT algorithm (LA-BOT). The challenge addressed

in the paper was to improve the overlay topology by smart selecting high bandwidth peers. In each optimization cycle, LA-BOT optimizes the structure of overlay in a distributed and aware way that decreases the latency in overlay as well as increases smart selecting of high bandwidth peers. The results of simulations presented throughout the paper have demonstrated that LA-BOT algorithm performs more efficiently comparing with the previous algorithms. Moreover, LA-BOT adapts the topology of overlay network and underlying network intelligently.

As a result, LA-BOT is able to support efficient and smart solution when equipped with learning automata and appropriate oracles.

References

[1] R. Steinmetz and k. Wehrle, Pee-to-peer Systems and Application, Springer, pp. 55-73, 2011.

[2] X. Shen, H. J. Buford and M. Akon, Handbook of peer to peer networking: a definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. Springer Science and Business Media, pp: 327-365, 2010.

[3] K. Kwong and R. Kwok, Pee-to-peer computing (application, architecture, protocols and challenges). Chapman & Hall/CRC, pp. 55-73, 2011.

[4] C. Tang and C. Ward, "GoCast: Gossip-enhanced overlay multicast for fast and dependable group communication", Proc. Int. Conf. on Dependable Systems and Networks, pp. 140 – 149, 2005.

[5] M. Montresor. Jelasity and O. Babaoglu, "T-Man: gossip-based fast overlay topology construction", Journal computer networks: The international journal of computer and telecommunications networking, Vol.13, No.53, pp.2321-2339, 2009.

[6] J. Leitão, J. P. Marques, J. Pereira and L. Rodrigues, "X-BOT: a protocol for resilient optimization of unstructured overlays", IEEE Trans. Parallel and Distributed Systems, Vol. 23, No. 11, pp. 12-23, 2012.

[7] B. Lalitha and CH. Subba Rao, "Bandwidth aided topology awareness", In the journal of advanced in electronic and electric engineering. ISSN: 2231-1297. Vol. 3. No. 7. pp.811-816, 2013.

[8] J. Leitão and L. Rodrigues, "HyparView: a membership protocol for reliable gossip-based broadcast", Proc. Int. Conf. on Dependable Systems and Networks, pp. 419-429, 2007.

[9] C. J. C. H. Watkins and P. Dayan, *"Machine learning",* Springer Science and Business Media, vol. 8, pp. 279-292, 1992.

[10] K. Najim and A.S. Poznyak, "Machine learning: theory and applications", Elsevier Science ltd, 2end edition, 2014.

[11] I. Baumgart and B. Heep, Oversim Community Site, [online], Available: http://www.oversim.org/wiki

Appendix I: X-BOT pseudo-code

Algorithm maintain()
Begin
Every ΔT **do**
//**phase 1**: Finding appropriate candidate from PassiveView.
If isFull(ActiveView) **then**
$CandidateSet \leftarrow RandomSample(PassiveView, PSL)$;
For $peer_i = UN$ and i <size of(ActiveView) **do**
$i = i + 1$;
$OldPeer \leftarrow ActiveView[peeri]$;
While CandidateSet≠{} **do**
$CandidPeer \leftarrow RemoveFirst(CandidateSet)$;
If isBetter(OldPeer, CandidPeer) **then**
Send(OPTIMIZATION, CandidPeer, OldPeer, myself);
EndIf;
EndFor;
EndIf;
upon Receive(OPTIMIZATIONREPLY, answer, OldPeer, DisconnectPeer, CandidPeer) **do**
If ANSWER **then**
If $OldPeer \in ActiveView$ **do**
If $DisconnectPeer = Null$ **then**
Send(DISCONNECTWAIT, OldPeer, myself);
Else
Send(DISCONNECT, OldPeer, myself);
EndIf;
$ActiveView \leftarrow ActiveView \setminus \{OldPeer\}$;
EndIf;
$PassiveView \leftarrow PassiveView \setminus \{CandidPeer\}$;
$ActiveView \leftarrow ActiveView \cup \{CandidPeer\}$;
EndIf;
End;
//**phase2:** Exchange the response of replaceable peers.
upon Receive(OPTIMIZATION, OldPeer, peer) **do**
If$\not\models isFull(ActiveView)$**then**
$ActiveView \leftarrow ActiveView \cup \{peer\}$;
Send(OPTIMIZATIONREPLY, True, OldPeer, null, myself);
else
$DisconnectPeer \leftarrow ActiveView[UNOPT]$;
Send(REPLACE, DisconnectPeer,OldPeer, peer, myself);
EndIf;
End;
upon Receive(REPLACEREPLY, answer, $peer_i$, OldPeer, DisconnectPeer) **do**
If answer **then**
$ActiveView \leftarrow ActiveView \setminus \{DisconnectPeer\}$;
$ActiveView \leftarrow ActiveView \cup \{peer_i\}$;
EndIf;
Send(OPTIMIZATIONREPLY, answer, OldPeer, DisconnectPeer, myself);
End;
//**phase3:** Comparing the cost of replaceable peers.
upon Receive(REPLACE, OldPeer, $peer_i$, CandidPeer) **do**
If $\not\models isBetter(peer, OldPeer)$ **then**
//# reflects better peer according to the cost retrieved from oracles.
Send(REPLACEREPLY, CandidPeer, false, $peer_i$, OldPeer, myself);
else
Send(SWITCH, OldPeer, $peer_i$, CandidPeer, myself);
EndIf;
End;
upon Receive(SWITCHREPLY, answer, $peer_i$, CandidPeer, OldPeer) **do**
If answer **then**
$ActiveView \leftarrow ActiveView \setminus \{CandidPeer\}$;
$ActiveView \leftarrow ActiveView \cup \{OldPeer\}$;
EndIf;
Send(REPLACEREPLY, answer, $peer_i$, myself);
End;
Phase 4: // Verifying and ensuring the symmetry of ActiveViews.
upon Receive(SWITCH, $peer_i$, CandidPeer, DisconnectPeer) **do**
If $i \in ActiveView$ or received(DISCONNECTWAIT from $peer_i$) **then**
send(DISCONNECTWAIT, $peer_i$, myself);
$ActiveView \leftarrow ActiveView \setminus \{peer_i\}$;
$ActiveView \leftarrow ActiveView \cup \{DisconnectPeer\}$;
EndIf;
Send(SWITCHREPLY, answer, DisconnectPeer, $peer_i$, CandidPeer, myself);
End;
//Function: isBetter.
Begin
isBetter(OldPeer, peer)
return Oracle.getCost(OldPeer) > Oracle.getCost(peer);
End;
End of procedure;