Bandwidth Allocation Algorithms for VOIP Networks: An Experimentation System and Evaluation of Created Algorithm

Rafal Orlowski, Michal Chamow, Iwona Pozniak-Koszalka, Leszek Koszalka, Andrzej Kasprzak
Department of Systems and Computer Networks,
Wroclaw University of Technology,
Wroclaw, Poland
E-mail: {156669, 156594, iwona.pozniak-koszalka, leszek.koszalka, andrzej.kasprzak}@pwr.wroc.pl

***Abstract*—In this paper, we present how to allocate bandwidth in VoIP networks, how to set up a connection, and which links in the network should be used. Common are algorithms which find shortest path (SPF) – like OSPF in Internet network. We create a new algorithm which takes into consideration two parameters, including maximum throughput bound. Using the created experimentation system we show that the algorithm can reduce the number of rejected calls and in some cases, the total cost of the calls.**

Keywords-bandwidth; algorithm; experimentation system; optimization; efficiency.

I. INTRODUCTION

Nowadays, one can observe an increasing number of computer network users. Voice over Internet Protocol (VoIP) is a general term for a family of transmission technologies that deliver voice communications over IP networks. The Internet and other packet-switched networks are particular examples. In this paper, we address the problem of how VoIP applications should be allocated bandwidth in networks. The objective is to find, in real-time, an optimal path in the network when establishing a VoIP connection between particular nodes.

The rest of the paper is organized as follows: after introducing the necessary terminology in Section II, algorithms that solve the allocation problem are briefly described in Section III (basic algorithm) and Section IV (modified algorithm), next, the results of investigations are presented and discussed. The designed and implemented experimentation system is described in Section V and the analysis of results of simulations made along with design of experiments is in Section VI. The conclusions appear in Section VII.

II. PROBLEM STATEMENT

Any network topology can be modeled as a graph in which vertices correspond to nodes and edges correspond to links (physical or logical) between the nodes. Two mainstream network architectures are considered. In the first considered case, nodes can be one of two possible types: *relay nodes* or *edge nodes*. Relay nodes transfer VoIP traffic, whereas edge nodes generate other network traffic, and use IP telephone (wireless phone, soft-phone or hands-free phones) generate traffic [1] [2]. This paper considers both wired as well as wireless networks. A significant research has been conducted in the area of 802.11-based wireless solution [3] [4]. For the second type of network, all the nodes belong to the same category. It is no distinction between nodes. This conception can be used when we have large network and each node is treated as small sub-network. Wireless network can be also described in the same way. The following terminology is used:

Constraints: cost and throughput. For a given link, defined by nodes *v* and *w*, it is natural to assume there is an associated cost, denoted c_{vw}: here, the meaning of 'cost' is deliberately left open to interpretation, as it often depends on user or hardware specifications. For example, according to the P2P model described in [5], 'cost' can mean: delay in milliseconds, distance in kilometers, and number of ISPs between two nodes. Furthermore, every link, defined by nodes *v* and *w*, will have a limited throughput, denoted d_{vw} (typically measure in kbps).

Connections. When a connection is established, it is allocated a bandwidth *b*. Therefore, the value *b* must be subtracted from the throughput d_{vw} every time a connection is established, assuming nodes v and *w*, lie on the connection path. If $d_{vw} < b$, no new connection can be established over the link: it that case, a new connection path must be sought, avoiding the node-pair (*v*,*w*). If no such path can be found, the connection request is rejected. To simplify calculations, the bandwidth *b* is chosen as the measurement unit, and the throughout values d_{vw} are normalized with respect to multiples of *b*. E.g., if *b* corresponds to 32 kbps (codec G.726 [6]) and d_{vw} corresponds to 2 Mbps, we set $b' = 1$ and $d_{vw}' = 128$.

Calls. All the call data are stored in a connection schedule (an example in Table I), including: time t_s, when the connection starts, nodes *v* and *w* defining start/end of the connection, and time t_t when the call is terminated t_t. When a connection is established between *v* and *w*, all the nodes and links lying in the connection path, as well as the associated costs, are stored in the connection schedule.

 ISBN: 978-1-61208-140-3

Objectives. The main objective of bandwidth allocation is to try and ensure that all the connection requests are satisfied (equivalently, the percentage of rejected requests are minimized). Another objective is to minimize the total cost of connections, where cost is represented by v_{ab}. Yet another objective is to reduce the number of terminated calls while the network resources are decreasing, but that can have an affect on the call costs.

III. THE ALGORITHMS

A. Basic Algorithm

Let us have a look at an example with a given connection schedule (composed of 5 rows - see Table I) and network topology as in Fig. 1 (with nodes enumerated from *1* to *9)*.

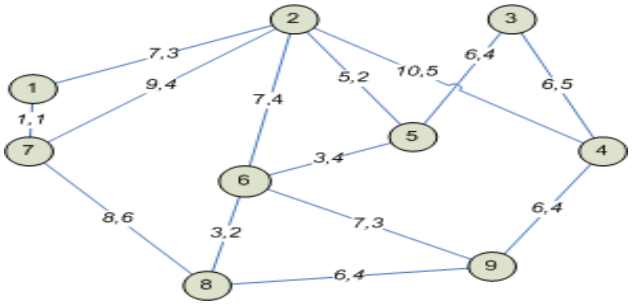

Figure 1. A network topology.

Each link is characterized by two attributes: the cost of the link, c_{vw} , and, the throughout, d_{vw} (it is a normalized value, so for e.g., $d_{vw} = 2$ means that two connections can be made simultaneously). It is assumed, that only one call is permitted per time slot $[t_s, t_t]$. We present, step by step, how the Basic Algorithm, i.e. Short Path Finding (SPF) algorithm works.

Step 0. In first iteration (t_s=0) there is one connection to set up.

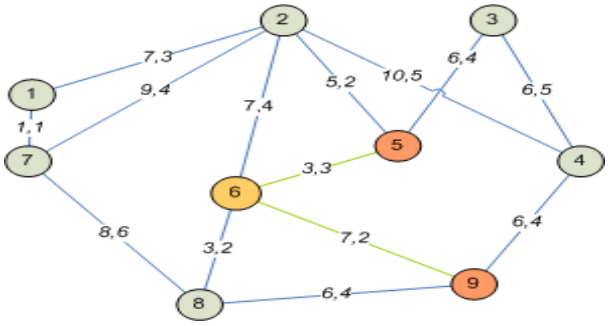

Figure 2. After first iteration (Step 0).

The shortest path (which can be verified using Dijkstra algorithm, for example) is shown in Fig. 2 - it may be observed that transit node is node *6*. The total cost of the path is $c_{5,6} + c_{6,9} = 3 + 7 = 10$. Remark: The value d_{vw} must be decreased accordingly, for each link on the connection path. At the end of any iteration, existing connections are required termination. In this particular example, no such action is required.

Step 1. Another connection is set up (see row *2* in Tab. 1). Once again it is the shortest path in the graph (Fig. 3) with transit nodes *5-2-1* and the total cost equal to *19*. Is this satisfactory? No! The link between nodes *1* and *7* cannot be used for another connection. Two other paths are possible (*3-5-6-8-7* and *3-5-2-7*), each with total cost = 20 (i.e. just one unit more), but neither consumes the total throughput of any link in the network**.**

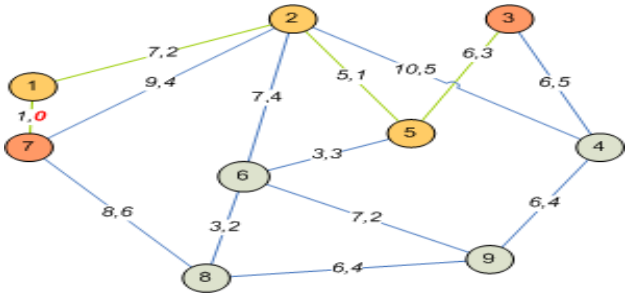

Figure 3. After second iteration (Step 1).

Step 2. The result of this iteration (path 4-9-3 with the total cost of 13) is shown in Fig. 4.

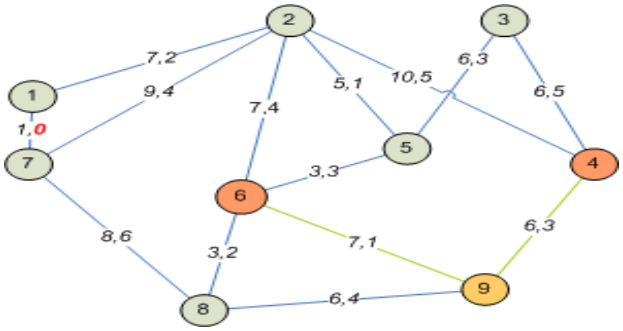

Figure 4. After third iteration (Step 2).

Step 3. The results of this iteration (path with no terminal node) is shown in Fig. 5

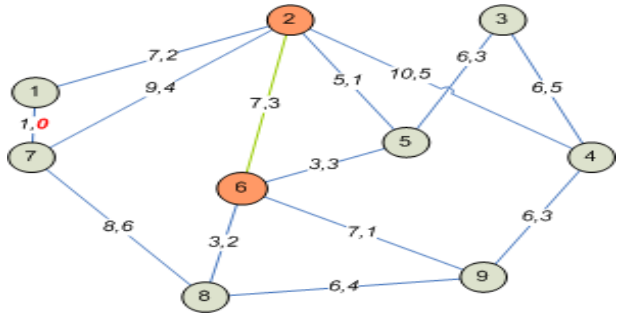

Figure 5. After fourth iteration (Step 3).

 ISBN: 978-1-61208-140-3

At the end of this iteration, only one connection is terminated (between nodes *5* and *9*). In this case, we must increment d_{vw} for each link participating in this connection. The result is shown in Fig. 6.

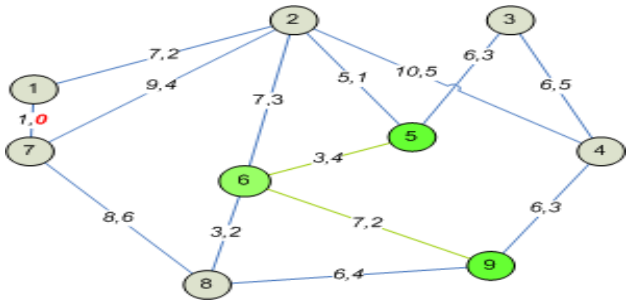

Figure 6. Disconnection.

Step 4. Here, a connection needs to be set up between node *1* and node *9*. If we only consider the values $c_{vw,}$ the optimal path is *1-7-8-9*. But link *1-7* is already operating at full capacity. Thus, a different path needs to be found.

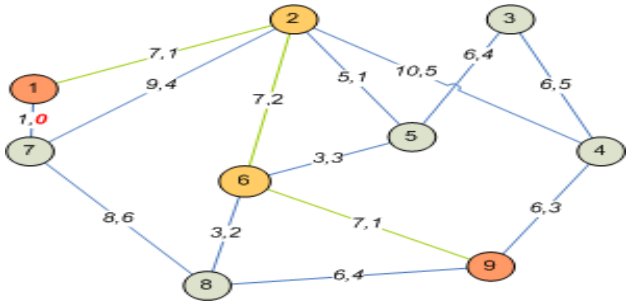

Figure 7. The result of Step 4.

The new path found (*1-2-6-9*) has a cost of 21. It is 6 units more than the cost of the *1-7-8-9* path, which can not be reached by SPF. The summary of the obtained results with SPF is given in Table I, there are also results given by the created algorithm described in Section III*B*.

TABLE I. ALGORITHMS PERFORMANCE

					Basic		***Premaru***	
Step	t_s	***v***	***w***	t_t	***Transit nodes***	***Cost***	***Transit nodes***	***Cost***
0	0	5	9	3	6	**10**	6	**10**
1	1	3	7	6	5,2,1	**19**	5,2	**20**
2	2	4	6	6	9	**13**	9	**13**
3	3	6	2	12	-	**7**	-	**7**
4	4	1	9	7	2,6	**21**	7,8	**15**
Total Cost						**70**		**65**

This simple example shows all the most important processes occurring in VoIP applications: setting up connections and/or making disconnection. It also illustrates the difficulty of finding optimal paths: the problem is to find a path having a minimal cost, and such that other potential connections will not be blocked in subsequent iterations. In Step 1 we could have set up the path *3-5-2-7*. Then the path cost in iteration 1 would be 20 (one more than it is now), but the advantage is that the connection *1-7-8-9* could then be set up in Step 4, thus, sparing five units of cost. If several more connection paths were to be badly chosen, the whole network itself might become completely saturated after just a few iterations, rendering future calls impossible. Moreover, poorly chosen connection paths severely compromise network reliability, leading to widespread user dissatisfaction.

B. Algorithm Premaru

The created algorithm, named Premaru, can compete with SPF. The idea of Premaru is based on introduction of two parameters denoted as *p* and *q*. The fundamental rule of the designed algorithm is following: "If the lowest value d_{vw} of all links in a found path is lower than *q*, moreover, there is any other path that connects the same nodes and its total cost is bigger than previous path by *p* or less, then the second (another) path will be chosen". It would avoid blocking beneficial paths in further iterations. This procedure is described by the pseudo-code and the block-diagram in Fig. 8.

Function *find_shortest_path* (*v*, *w*, *C*) finds the shortest path between nodes *v* and *w*. It uses the cost matrix *C* of the whole network. Function *throughput* (*path, D*) returns the minimum throughput, d_{min} ,of all edges in *path*. It exploits the throughput matrix, *D*, of the whole network. If the minimum throughput of the found path is lower than *q* we zeroes edge's throughput. That prevents reuse of edges having a throughput lower than *q*.

After that, a search is made for another path, named $path_{temp}$. If $path_{temp}$ satisfies the condition *cost* ($path_{temp}$) – *cost* (*path*) ≤ *p*, then this path is chosen. It enables to have a minimum throughput larger than *q*. This avoids traffic-crowded edges - $path_{temp}$ is calculated using a temporary matrix of throughputs $D_{temp,}$ where all throughputs not greater than *q* are omitted.

```
path = find_shortest_path(v, w, C, D);
d_min = throughput(path, D);
if d_min <= q then
D_temp = throughput_zeroing(q, D);
path_temp = find_shortest_path(v,w,C,D_temp);
if cost(path_temp) - cost(path) <= p
then path = path_temp;
end if
end if
set_up_connection(path, D);
```

Figure 8. Algorithm Premaru – pseudo-code description.

Function *find_shortest_path* (*v*, *w*) is left undefined. It can be the Dijkstra algorithm [7], as suggested earlier, but it can be the Bellman-Ford algorithm [7] or a heuristic algorithm (such as genetic algorithm), either. In our recent implementation, the modified Dijkstra algorithm was utilized - it finds the shortest path, taking into account d_{vw} (if d_{vw} is equal to 0, the edge *v*-*w* cannot be used).

IV. EXPERIMENTATION SYSTEM

The experimentation system has been designed and implemented by our research team following ideas presented in [8]. The core module of the system is simulation environment – a complex program which allows testing both considered algorithms (*Algorithms Performance Module*) and making multi-aspect investigations. The user can choose an experiment design which will be performed in automatic manner (*Experiment Design Module*) generating input parameters, including network matrices (*C* – cost matrix and *D* – throughput matrix) and connection schedule (*con_gen*). The values of Premaru parameters *p* and *q* can be also specified by the user. Output data is stored on a properly designed database (Data Acquisition Module). In Fig. 9, the experimentation system is shown as input – output plant.

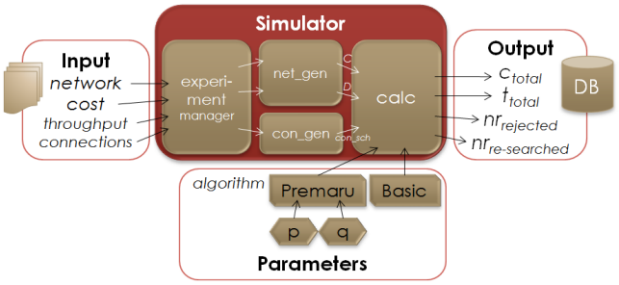

Figure 9. Experimentation system as input – output plant.

Terminology used in Fig. 9 is explained below**:**

- *network* – information about number of nodes and network density,
- *cost* – information about cost distribution (homogenous or Weibull) and its parameters (maximum cost for homogenous distribution),
- *throughput* – maximum throughput of a link (parameter of homogenous distribution),
- *connections* – the number of connections, λ is the parameter of the exponential distribution,
- *Basic algorithm (*also named Dijkstra*)* – a simple Dijkstra's algorithm, modified for our purposes.
- *Premaru algorithm* – the created algorithm that includes parameters *p* and *q*,
- c_{total} – the total cost of all connections,
- t_{total} – the total time of simulation,
- $nr_{rejected}$ – the number of rejected calls,
- $nr_{re\text{-}searched}$ – the number of calls for which was done more than one path search.

The application software was written using C# and .NET 3.5 Windows Forms, because they constitute a highly flexible platform for the design of user-friendly interfaces using familiar components (check-boxes, text-boxes, buttons, etc.). An external library was used to produce the charts. The database was compiled and maintained using SQLite. The system (simulator) only requires a PC that runs Windows OS and .NET Framework 3.5 or higher (freely downloadable from the Microsoft corporation website).

An important issue is experiment design, i.e., creating input data that models real-life scenarios. According to aspects specified in Section II the details about such scenarios may be described as follows.

The Network. The network is represented by the graph described by the pair of matrices c_{vw} and d_{vw}**.** The entries of the first matrix represent costs between nodes *v* and *w*. The entries of the second matrix represent the throughputs between nodes *v* and *w*. Those matrices are highly correlated, because if a non-zero entry in one of them corresponds to a non-zero entry in the other. Accordingly, for each link between nodes we have a pair of values: cost and throughput. The size of the network is also important. Tests can be made for 'small' (50 nodes) and 'large' networks (100 nodes). In the case of complete graphs (networks), each pair of nodes is connected, i.e. it represents network with 100% density. But such networks are too expensive and rarely used, in practice. The system allows generating networks with density from 3% to 80%. Each link between nodes is assigned a cost value, which is stored in matrix *C* as c_{vw}**.** This value might represent the time-delay (in time units - milliseconds, for example). Realistic cost values can be generated by exploiting two probability distributions: (i) homogeneous distribution, (ii) Weibull distribution. With a homogeneous distribution, cost values are drawn from 1 to *k*, where *k* determines the maximum value of cost (parameter *k* can be chosen in our system). The Weibull distribution provides a good mathematical model of delay in VoIP networks [9]. The Weibull distribution is characterized by: *k* – a shape parameter and λ – a scale parameter. Each link between nodes is assigned a throughput value, which is stored in matrix *D* as d_{vw}**.** Throughput for every link is drawn from homogeneous distribution. The throughput can be from *1* do *p, where p* represents the maximum throughput bound).

Connection schedule. In order to construct a good connection schedule, it is important to know how subscribers behave. In our research we assumed that the starting time of a connection can be accurately described by random variable with a homogeneous distribution. We

 ISBN: 978-1-61208-140-3

also assumed that the duration of a connection can be accurately described by a random variable having an exponential distribution. This assertion is based on the results presented in [10]. The probability density function of phone call duration described there can be modeled by an exponential distribution with λ=0.02. Our experimentation system allows changing the value of λ.

To generate a list of connections necessary parameters are: (i) Network size (number of nodes), (ii) Number of connections, (iii) λ-parameter in exponential distribution. The way of selecting nodes when a connection is being established is modeled by a homogenous distribution. There is only one condition: the initial node must not be the final node. The time-instant of establishing a connection is also modeled by a homogenous distribution.

V. INVESTIGATIONS

A. Design of experiment

The following input data were taken into consideration:

- Network size : 100 nodes.
- Network density: 3%, 6%, 9%, 12%, 15%, 18%, 21%, 24%, 27%, 30%, 35%, 40%, 45%, 50%, 60%, 70%, 80% .
- Number of connections: 1000, 2000, 3000, 4000, 5000, 7500, 10000.
- The cost values of links: generated using Weibull distribution with k = 2.09 and λ = 7.5.
- Call duration: modeled using the exponential distribution with λ = 0.02.
- The Premaru parameters (p, q): 16 different pairs – all combinations for q = 1, 2, 3, 4 and for p = 1, 2, 3, 4. , i.e. (1, 1), (1, 2), … , (3, 4), (4, 4). .

B. Number of rejected calls

In Fig. 10, one can see the average results given by SPF Basic algorithm and Premaru algorithm.

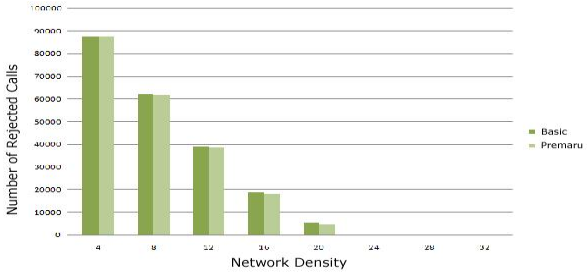

Figure 10. Number of rejected calls in relation to network density.

It may be observed, that number of rejected calls is almost the same when Premaru algorithm is used instead of Basic algorithm. We found that these differences were very small - less than 5% (Fig.11). Moreover, the network density do not influence on number of rejected calls rate. It was only observed that the number of rejected calls was linearly proportional to the number of connections.

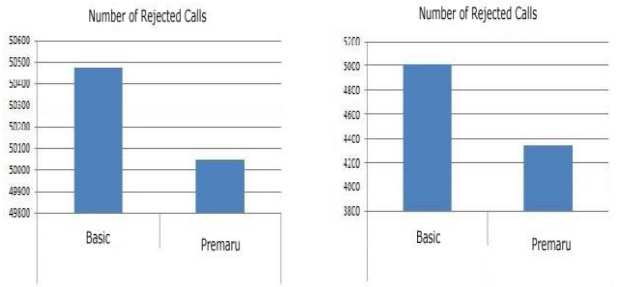

Figure 11. Number of rejected calls for network density 10% (on left) and 20% (on right).

C. The total cost

The average total cost given by Basic algorithm and Premaru algorithm in relation to the number of calls for various network densities is shown in Fig.12.

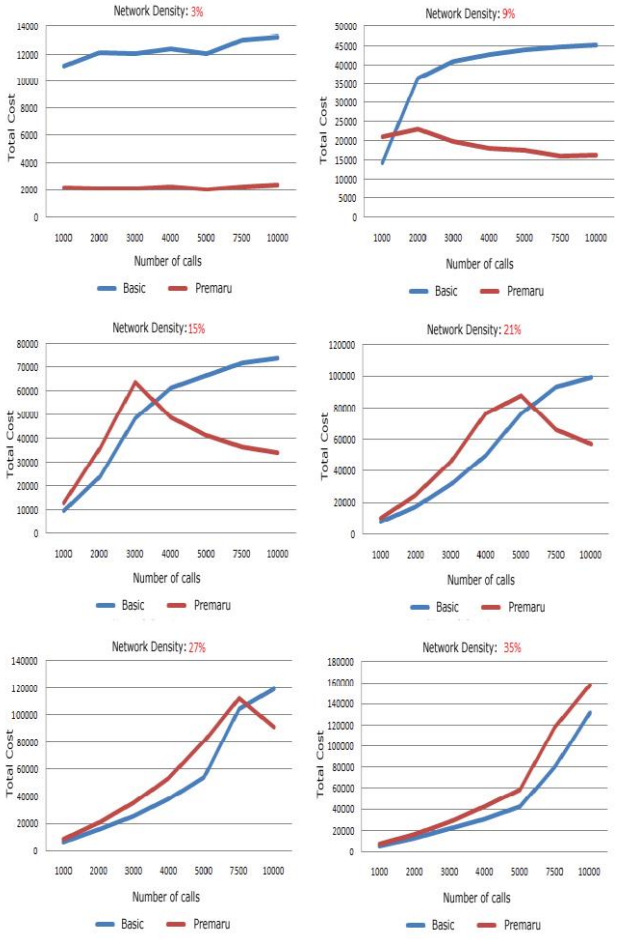

Figure 12. The total cost in relation to the number of connections for different network densities.

It may be observed, that Premaru algorithm is better (producing lower costs) in two situations: (i) for relatively small network densities almost independently on the number of connections, and (ii) in some cases of network with greater densities if the number of connections is relatively big.

D. *The influence of (p, q) parameters*

The influence of *q* parameter is shown in Fig. 13. By increasing the *q* parameter the total cost is reduced but this property was found as evident for networks with low densities. It may be observed in Fig. 13 that for $q = 1$, the total cost is lower than for $q = 4$.

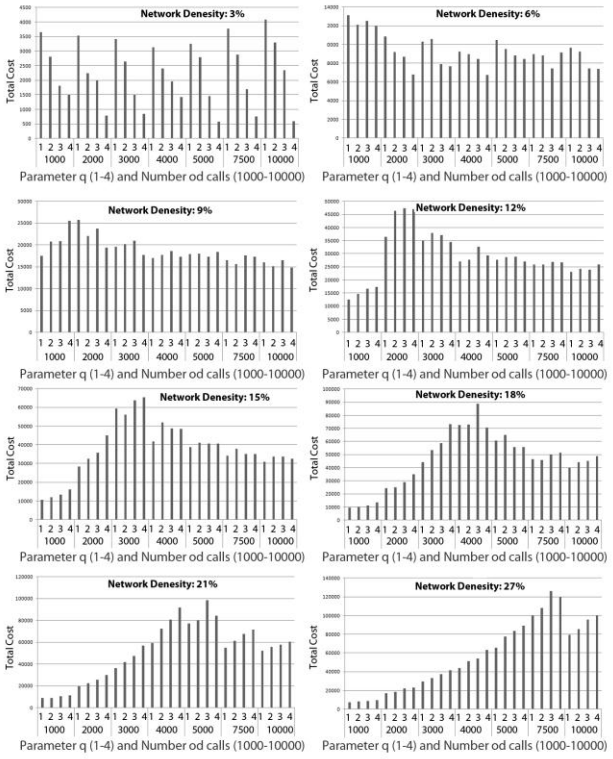

Figure 13. The total cost produced by Premaru algorithm in relation to parameter *q* for different network densities and numbers of connections.

In addition, if we have a density networks less than 50% then the difference between the total costs incurred by Premaru and Basic, in an experiment where there are no rejected calls, did not depend on the parameter *p*. However, if the network density is greater than 50%, the cost reduction with Premaru is remarkable as *p* increases.

VI. CONCLUSION

In the paper, we present the results of evaluation of the created and implemented Premaru algorithm. The evaluation was based on simulations made using the designed and implemented experimentation system. The Premaru is able to find suitable connection paths in VOIP networks while in many cases significantly reducing the total cost of the established connections as well as reducing the number of rejected calls.

We suggest using Premaru algorithm for lower density networks and for all networks when the number of connections is relatively big, i.e., the network connectivity problem is sufficiently complex.

In the nearer future we are planning: to create a hybrid algorithm which will perform as Basic algorithm before reaching rejected calls limit and after as Premaru algorithm, and to develop the experimentation system by implementing newly designed algorithm and include new modules following ideas described in [11].

Acknowledgements

The authors would like to thank the students of Wroclaw University of Technology – Mr. Radwan, Mr. Kempa, Mr. Kidon, Mr. Miziolek, and Mr. Neska for remarkable help in programming the experimentation system and Mr Daniel Davies for productive proofreading.

REFERENCES

[1] V. Hilt, A. Hari, and M. Hofmann, “An Efficient and Robust Overlay Routing Scheme for VoIP”, IEEE Press, ICICS, Bangkok, December 2006, p. 148.

[2] R. Birke, M. Mellia, M. Petracca, and D. Rossi, “Experiences of VOIP Traffic Monitoring in Commercial ISP”, International Journal of Network Management, vol. 20, 2010, p. 339.

[3] S. Ganguly, V. Navda, K. Kyungtae, A. Kashyap, and S.R. Das, “Performance Optimizations for Deploying VoIP Services in Mesh Networks”, IEEE Journal on Telecommunications., vol. 24, 2006, p. 2147.

[4] W. Wang; S. L. Chang, O. K. Victor, and H. Li, “Solutions to Performance Problems in VoIP over a 802.11 Wireless LAN”, IEEE Transactions on Vehicular Technology, 54, 2005, p. 366.

[5] K. Walkowiak , “Offline Approach to Modelling and Optimization of Flows in Peer-to-Peer Systems”, IEEE Press, NTMS, Tangier, 2008, p.1.

[6] Recommendation G. 726: 40, 32, 24, 16 kbit/s, *Adaptive Differential Pulse Code Modulation (ADPCM,).*

[7] A. Kasprzak, “Packet-switching Wide Area Networks”, WPWR, Wroclaw, 2004 /in Polish/.

[8] D. Ohia, L. Koszalka, and A. Kasprzak, “Evolutionary Algorithm for Congestion Problem in Computer Networks”, Springer, Lecture Notes of Artificial Intelligence, vol. 5711, 2009, p. 113.

[9] H. Li and L. Mason, “Estimation and Simulation of Network Delay Traces for VoIP in Service Overlay Network”, IEEE Press, 2007, p. 423.

[10] R. Birke, M. Mellia, M. Petracca, and D. Rossi, “Understanding VOIP from Backbone Measurements”, IEEE Press, INFOCOM, 2007, p. 2027.

[11] L. Koszalka, D. Lisowski, and I. Pozniak-Koszalka, “Comparison of Allocation Algorithms in Mesh Sructure with Multistage Experiments”, Lecture Notes in Computer Science, vol. 3984, Springer, 2006, p. 58.

 ISBN: 978-1-61208-140-3