

Improving Accuracy of a Network Model Basing on the Case Study of a Distributed System with a Mobile Application and an API

Wojciech Rząsa¹, Marcin Jamro², and Dariusz Rzonca¹(✉)

¹ Department of Computer and Control Engineering,
Rzeszow University of Technology,
al. Powstancow Warszawy 12, 35-959 Rzeszow, Poland
{wrzasa, drzonca}@kia.prz.edu.pl

² TITUTO Sp. z o.o. [Ltd.], Zelwerowicza 52G Street, 35-601 Rzeszow, Poland
marcin@tituto.com
<http://kia.prz.edu.pl>
<http://tituto.com>

Abstract. Nowadays, many IT products are created as distributed solutions that consist of many parts, such as mobile applications, web-based back-ends, as well as APIs that connect various parts of the system. It is a crucial task to apply a suitable architecture to provide users of mobile applications with satisfactory operation, especially when the Internet connection is necessary to get or send some data. The simulation of network architecture and configuration using a high-level model of the system described with dedicated Domain-Specific Language (DSL), enabled by the Timed Colored Petri Nets (TCPNs) formalism is a beneficial approach that could be applied in real case studies. The already proven research method has been applied to one of the scenarios regarding the system offered by TITUTO Sp. z o.o. [Ltd.] company (Rzeszow, Poland). The first obtained results were not sufficiently precise for detailed analysis of the system. Thus, the case study was used to improve the simulation method in order to more accurately model data transmissions over the network. After modifications were implemented in the simulation tool, significantly better results have been received, as discussed in the paper.

Keywords: Simulation · Petri Nets · Performance · Distributed system · API · Mobile application

1 Introduction

While preparing a distributed solution, it is an important task to choose a suitable architecture that provides sufficient performance and correctly handles scenarios with diverse number of users who get or send some data via the Internet. For this reason, introducing research methods into the process of choosing an architecture is a beneficial approach. There are several approaches that could be used for this purpose. One of possible solutions is modeling network to check

whether the given system architecture and configuration could properly handle the operation by a specified number of users.

As mentioned in the title of this paper, it is based on the real case study. It is related to one of the products developed by TITUTO Sp. z o.o. [Ltd.]¹ (Rzeszow, Poland). This company offers a set of own IT products dedicated to hotels and tour operators. One of solutions, which is analyzed in this paper, is the mobile trip assistant TOURISER². It uses smartphones and tablets of trip participants to present a set of information useful during the organized tour. Of course, the product consists of many parts forming together the distributed system:

- the mobile applications running on smartphones and tablets with Android, iOS, and Windows operating systems
- web applications, available via web browser, to manage data presented in the mobile applications, as well as to perform other tasks
- Application Programming Interface (API) to connect various system parts.

The mobile applications are equipped with a set of features, such as presentation of a trip program. Of course, they should operate also in the off-line mode, so it is crucial to provide users with a possibility to download all necessary data, including textual content, as well as a set of photos and audio recordings. Such a process is further referred as update. It can be performed by many people simultaneously, especially when the update process is forced by the tour guide who informs the group of trip participants to launch the application to get the latest data for the current trip. What is more, the update process could be performed from various devices with diverse performance, as well as using the broad range of speed of the Internet connection, even with a limited availability.

To provide users with satisfactory update times, ensure that the initial architecture of the system is sufficient, as well as prepare the product in an expandable way, the company has decided to cooperate with researchers from Department of Computer and Control Engineering from Rzeszow University of Technology. At the beginning, the authors of this paper used the already available tools and research methods. Unfortunately, while performing experiments, it has been necessary to introduce modifications in order to improve accuracy of a network model to present results more proper for the real-world scenario.

In this paper, the detailed analysis of results, together with a set of graphs, is presented. In the second chapter, the initial approach is described. Then, the doubts regarding received results are discussed. The fourth chapter shows the problem of parallel data transfers, together with improving the previously mentioned model, and estimating the improvement. In the following chapter, the corrected simulator is applied to the case study with a detailed comparison and explanation of the received results.

¹ <http://tituto.com>.

² <http://touriser.com>.

2 Initial Approach to Analysis

Different approaches are described in the literature as suitable for analyzing distributed applications and network impact on performance. It is possible to simulate computer networks behavior in so-called “network simulator” (e.g. widely used family of ns-2 and ns-3 [1] open-source simulation tools). Another approach involves creating models of network components in one of the formalisms dedicated to analyze and solve concurrency problems. Among these formalisms various classes of Petri nets seem to be the most popular, due to their ability to model activities of concurrent and distributed systems. For example Fuzzy Petri Nets may be used as a formal representation of a concurrent control algorithm [2]. Queueing Petri Nets can be used as a performance prediction tool during the software engineering process of a distributed component-based system [3] or for response time analysis of distributed web systems [4]. Timed Coloured Petri Nets are also used for simulation and performance analysis of distributed internet systems [5]. Another Petri net class, RTCP-nets are used for modeling and analysis of embedded and real-time systems [6]. Analysis of such Petri net models is possible in strictly formal way [7–9], but also the model can be simulated to observe behavior of the modeled system [10], and perform statistical performance analysis. Both ways provide reliable results and can be used to verify some properties and validate the requirements [11].

In the case considered in this paper, the analysis of application was based on the method that allows to obtain precise results, as described in [12–14]. The method assumes that a high-level model of application should be created. The model is then a basis for simulation. It is described with a Domain Specific Language (DSL) designed especially for this purpose. Reliability of simulation is ensured by the formalism of Timed Colored Petri Nets (TCPN) [15]. The concept of connecting a high-level model with TCPN is described in [16] and also applied in [17].

2.1 DSL Concepts

The DSL allows presenting model of application using concepts of programs that describe actions performed by processes running on nodes. Each pair of nodes can be connected through one or more net segments that allow remote communication. In the model used in this research, there were two programs, namely describing (1) an API server and (2) mobile clients. There was one instance of the API server program running in each simulation. The number of processes that run client programs depended on configuration of specific simulation. Each process in the simulation run on a separate node.

Network communication between clients and the server was possible through network segments that modeled server and clients Internet links. Network route between a client and the server always led through two network segments that represented (1) the client’s Internet link and (2) the servers Internet link. Thus, it was possible to model limited bandwidth (1) between the server and the Internet

(shared between all clients) and (2) that of each client Internet link. The network links enabled full-duplex communication.

2.2 Processes

The server process served two types of requests, as follows:

1. `:config` that returned content of the update configuration file. The configuration described all files that should be downloaded during the update.
2. `:update` with specific file name as its parameter. It returned a given file to the client.

Each file included in the update configuration had its specific size and, therefore, required a given network transmission time.

A client process, after it was started, sent the `:config` request to the server in order to obtain information about files that should be updated. Thereafter, it downloaded subsequent files from the received list by sending the `:update` requests for each of them. Downloading of each file was accompanied with timeout. If a timeout occurred, it was considered that the whole update process for this client failed. There was a startup delay between the moment the client program started and the moment it begun its update. It was set to a random value from a specified range. Consequently, all clients did not start their updates at the same moment in time. It is consistent with the real world behavior, even if application update is triggered by a person suggesting this update to a group of people.

2.3 Gathering Statistics

The model was used to gather performance-related statistics about the application. The most important factor was the update time for clients. This was measured for each client as time difference between the moment the client started update and the moment the last file from the update was received. Excessive number of simultaneous updates could result in the number of clients not being able to complete the process and timeout due to the overloaded server network. Therefore, the number of timeouts during the update processes was the second parameter measured during simulations.

The simulations were carried out for three types of clients, characterized by the following bandwidth of Internet link:

```
:slow – 512 kbps
:normal – 2 Mbps
:fast – 20 Mbps
```

In accordance with parameters of an exemplary scenario of the analyzed application, each client performed an update process consisting of 308 files of diverse sizes. Total size of each update was almost 70 MB. Network bandwidth for the

Table 1. Average update time for a client in different configuration of simulation.

Number of clients of each type	Average update time [s]
$20 \times \text{:slow}$	1145.91 s
$20 \times \text{:normal}$	293.56 s
$40 \times \text{:fast}$	195.18 s
$40 \times \text{:fast}, 20 \times \text{:normal}, 20 \times \text{:slow}$	546.33 s
$80 \times \text{:fast}, 20 \times \text{:normal}, 20 \times \text{:slow}$	675.23 s
$20 \times \text{:fast}, 20 \times \text{:normal}, 20 \times \text{:slow}$	551.17 s

API server was set to 100 Mbps. Simulations were performed for different numbers of clients of each type, as shown in Table 1, which presents also an average update time for a client.

For these configurations, no timeouts were observed and all update processes finished successfully. The results show that for the considered configuration of resources the application should behave correctly. The update times for :slow clients were significant, but such a situation was caused by their own slow Internet links and could not be changed. The update times for other groups of clients were on the limit of acceptance. This was a result of insufficient bandwidth of the Internet link of the API server. However, the value used in the experiments was deliberately low to verify behavior of the application in unfavorable conditions. The real application was supposed to work on a server with 1 Gbps Internet link. It seemed that the application was not threatened by transmission timeouts, since the first indicators of too slow Internet link of the API server were significant client update times. The update times would become unacceptably long before insufficient bandwidth of server link could cause transmission timeouts.

3 Assessing Simulator Accuracy

3.1 Doubts Concerning Results

The analysis of application provided satisfactory results. However, unlike in case of previous research [12–14], additional consideration was required for this specific case. The modeled and simulated application performance proved to significantly depend on the network transmission efficiency. However, the simulator used a basic network data transmission model. Data sent between two processes in a single message were treated as a whole and transmission time over a network was modeled as a delay computed for the whole data package. If a route between two nodes consisted of more than a single network segment, a data package was transmitted over another segment only after it had been completely transmitted over the previous one.

This approach allowed the very efficient simulation with accuracy sufficient for applications where data processing had important impact on performance.

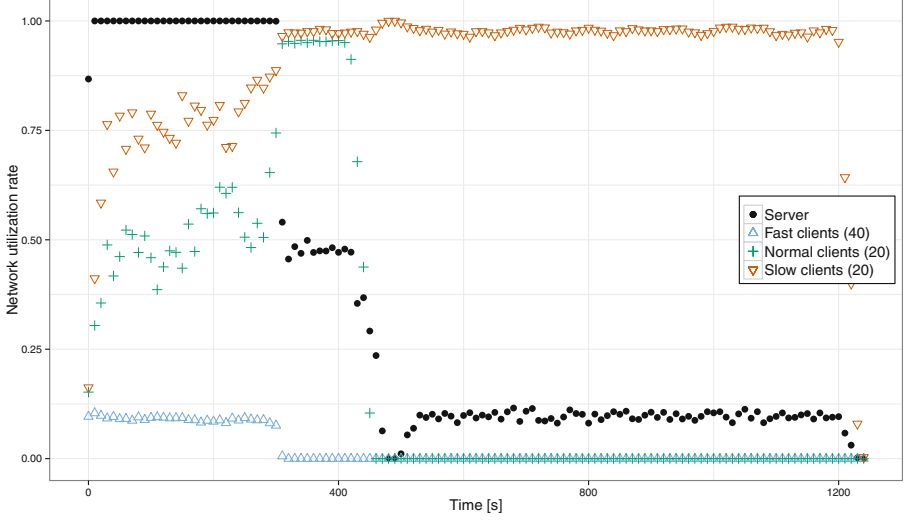


Fig. 1. Network utilization rate for multiple clients of each type.

However, applying this simplified model to the application presented in this paper required consideration of possible inaccuracies. Therefore, more detailed results of simulation concerning network usage were analyzed.

Figure 1 presents mean rate of network utilization for individual types of Internet links for clients and for the API server. At the beginning, the server link was fully loaded since all the clients performed their update processes. After the :fast clients finished their update processes, the server's network utilization decreased, since the :normal and :slow clients' Internet links did not allow them to fill the servers network bandwidth. This was an expected behavior. However, the rate of network utilization of :normal and :slow clients while the :fast ones still performed updates raised justified doubts. The results of additional simulations performed for single clients of each type, presented in the Fig. 2, had the same characteristics: slower clients did not fill bandwidth of their network links despite the fact that their share of the server's network bandwidth should be sufficient for this.

3.2 Comparison with a Real-World Experiment

The results of the basic simulations with individual clients of each type could be verified by comparison with the real-world experiments. This would be hardly feasible for the case in which a larger number of clients is considered.

The experiments were performed on three virtual machines running Linux. Three clients (fast, normal, and slow) were running on the virtual machines and downloading data from the host. The operating system's traffic control capabilities configured with the Linux tc program were used to limit bandwidth available for the transfers. The server's bandwidth was limited with Token Bucket Filter

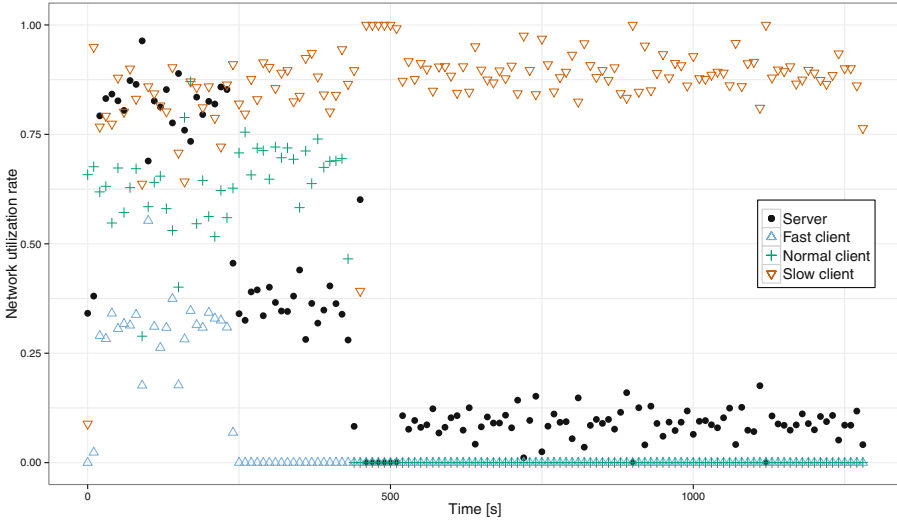


Fig. 2. Network utilization rate for single clients of each type.

(TBF) attached to outgoing network interface. Incoming traffic for each client was shaped with a TBF attached to Intermediate Functional Block devices – virtual network interfaces enabling control of ingress network traffic in Linux. Consequently, each stream of downloaded data was passing two network segments with different bandwidth: one on server’s network interface (shared by all clients) and the other on the clients network interface. The bandwidth of the server network was set to 5 Mbps and bandwidths for the clients were configured as follows:

- **slow** – 0.5 Mbps
- **normal** – 2 Mbps
- **fast** – 8 Mbps

Consequently, slower clients were not able to fill the bandwidth of server’s network, and the fast client had a faster network link than the server. This is consistent with the configuration of the original simulations with multiple clients if the total bandwidth available for each type of clients is considered.

As observed during the experiment, average bandwidths used by the clients in two-minute download processes were as follows:

- **slow** – 0.47 Mbps
- **normal** – 1.9 Mbps
- **fast** – 2.2 Mbps

The results clearly show that in the analyzed configuration the slower clients almost fully utilized bandwidths of their own network links. This was a significant difference in comparison with the simulation results.

4 Improving Precision of the Network Model

4.1 The Problem of Parallel Data Transfers

A more precise estimation of network transmission time could be achieved by combining existing simulator with a precise model of Transmission Control Protocol (TCP). It is even feasible to include crucial parts of the real TCP implementation into the simulator, as shown in [18]. However, such a solution requires significant effort not only to modify existing simulator, but also – what is even more important – to obtain reliable information about details of TCP versions and configurations of particular servers and clients. The second problem seems to be especially difficult for real-world business applications. For this reason, another – less precise, but more practical – solution was chosen.

As already mentioned, the network model in the simulator assumed that a data package (e.g., a file) was transmitted as a whole, generating a proper delay for delivery of the data package and occupying a network segment for the whole transmission time. Consequently, the next data transfer was delayed until the previous one was finished. Thus, in the simulations described above, data packages designated for slower clients were frequently delayed on the server link by other transfers, especially the ones sent to the faster clients. Thus, the slower client's network links were idle for some periods of time, waiting for data from the faster link. Therefore, although the server link was significantly faster than the links of normal and slow clients, the crude network model prevented the transfers from filling bandwidths of the slower networks.

This situation does not occur in the real-world scenarios, because transmitted data are divided into segments. Thus, even large data transfers do not delay the others until they are completed. First, the data transfers over one network overlap. Second, segments already transmitted over one network are passed to another one and a single data package (e.g. file) can be concurrently transmitted over more than one of subsequent networks on a route. Thus, faster network links, even if significantly loaded, usually can transfer sufficient number of data segments to fill bandwidth of the slower ones ensuring as efficient data transmission as possible.

4.2 Improving the Simplified Model

In order to mitigate the problem without including the complete TCP model in the simulator, an intermediate solution was devised. In order to ensure a lower grained and more precise simulation of parallel transmissions, the transmitted data packages (e.g., files) were divided into a number of fragments. Each fragment was transmitted separately and the simulator reported data reception only after all fragments were received. This solution also ensured that a single data package could be transmitted through more than one network on a route. Thus, with a larger number of smaller data fragments to transfer, the faster network links should have been able to fill bandwidth of the slower ones even if the faster links were busy.

The solution with data fragmentation was implemented in the simulator as an additional option. The maximum number of fragments for a data package can be configured for each simulation. The actual number of fragments for a specific package is selected according to (1), where M is the maximum number of fragments configured for simulation and s is size of the whole data package in bytes. The value of 1500 is derived from the Maximum Transmission Unit (MTU) for the most popular Ethernet protocol. Consequently, the fragments are not smaller than 1500 B and with value of M simulator may be configured to use more or less fine grained model.

$$\min(M, \lfloor s/1500 \rfloor) \quad (1)$$

4.3 Estimation of Achieved Improvement

The results provided by the new version of the simulator were again compared with the results of the experiment described in Sect. 3.2. The maximum number of fragments was set to different values in order to find a setting that ensured the sufficient precision. As presented in Fig. 3, setting the maximum number of fragments to 5, significantly improved accuracy of results. Figure 4 shows additional improvement for the maximum number of fragments set to 10, but in this case the difference is not significant. Similarly, increasing the maximum number of fragments to 30 did not cause meaningful improvement (Fig. 5).

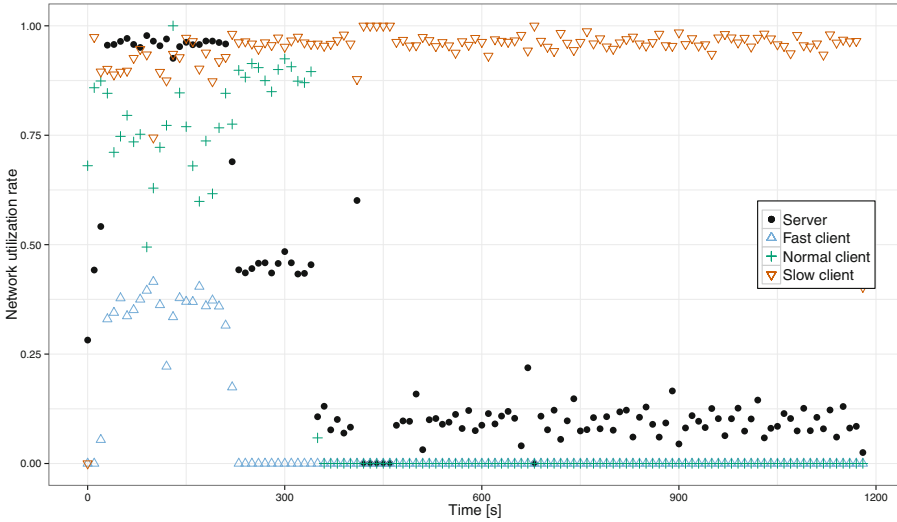


Fig. 3. Results of simulations for the maximum number of data fragments set to 5.

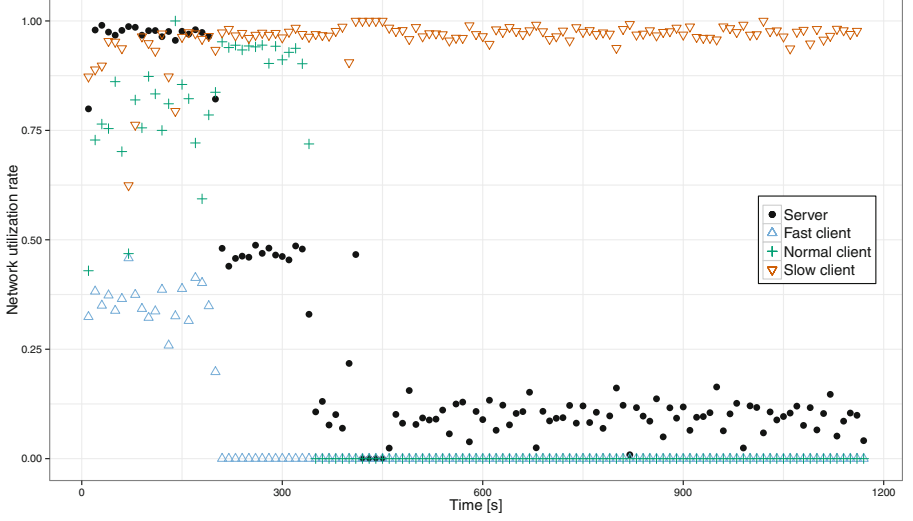


Fig. 4. Results of simulations for the maximum number of data fragments set to 10.

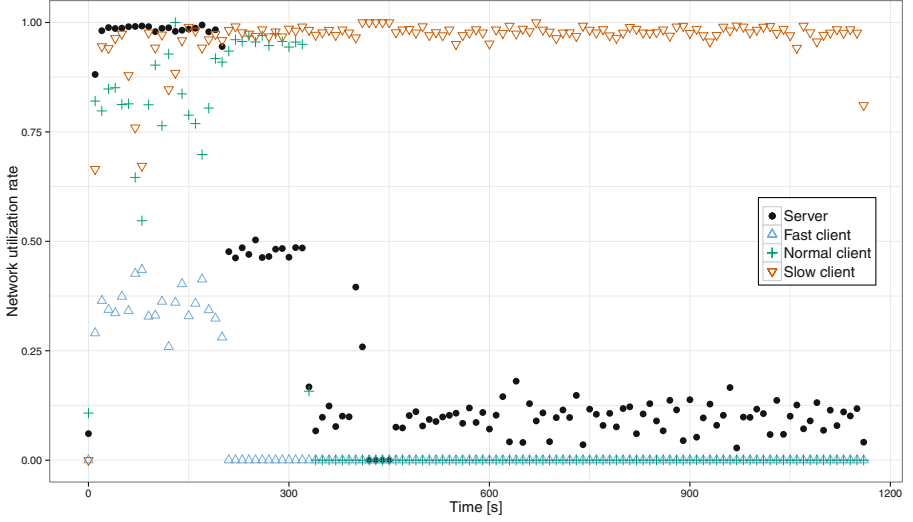


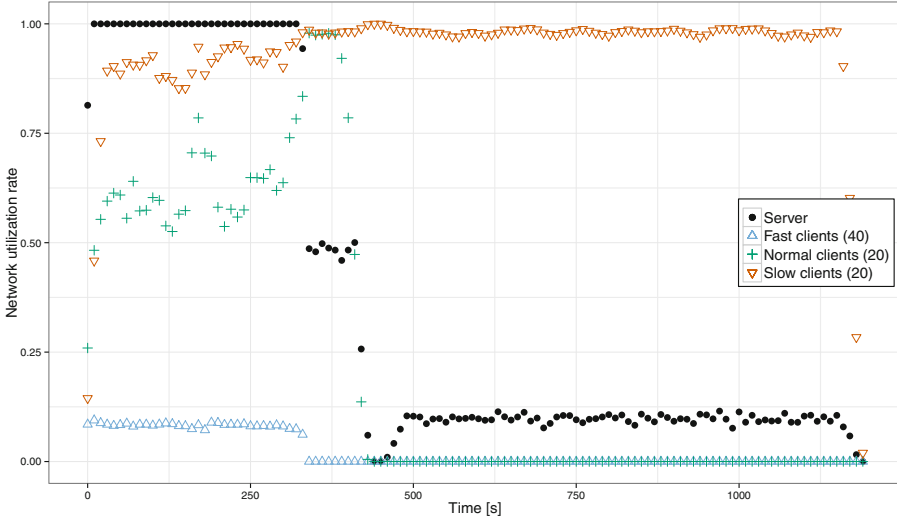
Fig. 5. Results of simulations for the maximum number of data fragments set to 30.

5 Applying Improved Simulator to the Case Study

The improved simulator was used again to analyze the real case-study application in order to check how the more precise simulation affected results. The simulations were carried out for two values of the maximum number of data fragments set to 5 and 10. Average update times are summarized in Table 2. Graphs of the network usage are presented in Figs. 6 and 7.

Table 2. Average update time for a client for different values of the maximum number of data fragments.

Number of clients of each type	Average update time [s]		
	No fragmentation	5 fragments	10 fragments
40 × :fast, 20 × :normal, 20 × :slow	546.33 s	542.26	541.05
80 × :fast, 20 × :normal, 20 × :slow	675.23 s	683.10	679.02
20 × :fast, 20 × :normal, 20 × :slow	551.17 s	541.28	540.74

**Fig. 6.** Results of simulations for the maximum number of data fragments set to 5.

Comparison of results from Table 2 does not show significant differences in update times for different configurations of simulator. This is due to the fact, that in average, shorter update time for :slow and :normal clients is compensated by longer update times for :fast clients. Therefore, analysis of the less aggregated results of simulation presented in Figs. 6 and 7 more clearly shows differences in obtained results.

To illustrate these differences between the initial result of simulation (without fragmentation, Fig. 1) and the final one (maximum 10 fragments, Fig. 7), such scenarios are directly compared in Fig. 8. For clarity of the graph, only results for slow clients are shown.

It is clearly seen that the network utilization rate is higher when fragmentation is considered, especially for the first part, when fast clients were still downloading the update. Thus, the obtained simulation results more closely correspond to the experiment results described in Sect. 3.2.

For the considered application, the aggregate metrics, i.e., the average client update time and the number of client timeouts were the required results. These

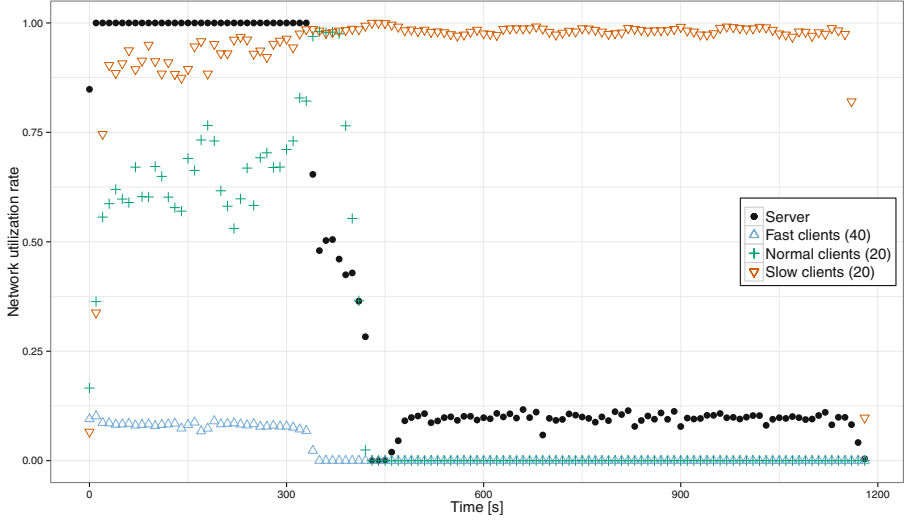


Fig. 7. Results of simulations for the maximum number of data fragments set to 10.

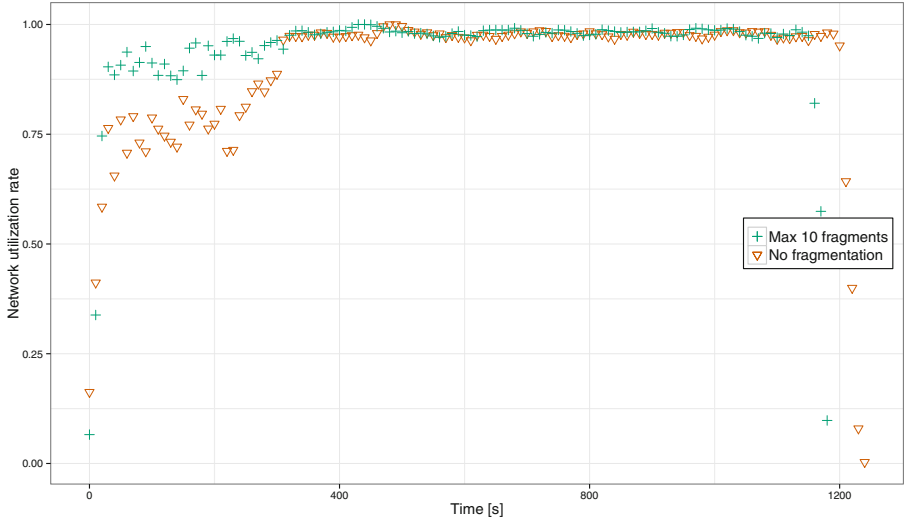


Fig. 8. Comparison of simulation results for slow clients depending on fragmentation.

values were not significantly affected by the basic model of network data transmission, despite of the fact that the application is network-bounded. However, simulations are frequently used also to observe behavior of a system in a more detailed way. In such the case, where not only aggregated results are important and when performance of an application depends mainly on network transmission time, the more precise network model may be an important improvement for the simulator.

6 Summary

Based on the experiments described in the paper, using simulation enabled by the formalism of Timed Colored Petri Nets (TCPNs) to analyze a high-level model described with a Domain-Specific Language (DSL) allows to obtain reliable results regarding network-related performance of distributed systems. The aim of the case study presented in this paper was to examine performance of one of the systems developed and offered by TITUTO Sp. z o.o. [Ltd.] company (Rzeszow, Poland).

The initial version of the simulation method did not provide satisfactory results while more detailed analysis of the real-world scenario was performed. For this reason, the research approach has been adjusted to properly handle transmission of files over the network, by adding the possibility of dividing them into fragments. In the paper, a few experiments are described, together with a detailed analysis of results. At the end, the already checked and described tool has been enhanced with additional features.

References

1. Riley, G.F., Henderson, T.R.: The ns-3 network simulator. In: Wehrle, K., Güneş, M., Gross, J. (eds.) *Modeling and Tools for Network Simulation*, pp. 15–34. Springer, Heidelberg (2010)
2. Gniewek, L.: Sequential control algorithm in the form of fuzzy interpreted Petri net. *IEEE Trans. Syst. Man Cybern. Syst.* **43**(2), 451–459 (2013)
3. Kounev, S.: Performance modeling and evaluation of distributed component-based systems using queueing Petri nets. *IEEE Trans. Softw. Eng.* **32**(7), 486–502 (2006)
4. Rak, T.: Response time analysis of distributed web systems using QPNs. *Math. Probl. Eng.* **2015**, 1–10 (2015). doi:[10.1155/2015/490835](https://doi.org/10.1155/2015/490835). Article ID 490835
5. Rak, T., Samolej, S.: Simulation and performance analysis of distributed internet systems using TCPNs. *Informatica Int. J. Comput. Inform.* **33**(4), 405–415 (2009)
6. Szpyrka, M., Szmuc, T.: Integrated approach to modelling and analysis using RTCP-nets. In: Sacha, K. (ed.) *Software Engineering Techniques: Design for Quality*. IIFIP, vol. 227, pp. 115–120. Springer, Boston (2006). doi:[10.1007/978-0-387-39388-9_11](https://doi.org/10.1007/978-0-387-39388-9_11)
7. Gniewek, L.: Coverability graph of fuzzy interpreted Petri net. *IEEE Trans. Syst. Man Cybern. Syst.* **44**(9), 1272–1277 (2014)
8. Murata, T.: Petri nets: properties, analysis and applications. *Proc. IEEE* **77**(4), 541–580 (1989)
9. Szpyrka, M.: Analysis of RTCP-nets with reachability graphs. *Fundamenta Informaticae* **74**(2), 375–390 (2006)
10. Wells, L., Christensen, S., Kristensen, L.M., Mortensen, K.H.: Simulation based performance analysis of web servers. In: *Proceedings 9th International Workshop on Petri Nets and Performance Models*, pp. 59–68 (2001)
11. Girault, C., Valk, R.: *Petri Nets for Systems Engineering. A Guide to Modelling Verification, and Applications*. Springer, Heidelberg (2003)
12. Rzaśa, W.: Simulation-based analysis of a platform as a service infrastructure performance from a user perspective. In: Gaj, P., Kwiecień, A., Stera, P. (eds.) *CN 2015. CCIS*, vol. 522, pp. 182–192. Springer, Cham (2015). doi:[10.1007/978-3-319-19419-6_17](https://doi.org/10.1007/978-3-319-19419-6_17)

13. Rzaša, W., Rzonca, D.: Event-driven approach to modeling and performance estimation of a distributed control system. In: Gaj, P., Kwiecień, A., Stera, P. (eds.) CN 2016. CCIS, vol. 608, pp. 168–179. Springer, Cham (2016). doi:[10.1007/978-3-319-39207-3_15](https://doi.org/10.1007/978-3-319-39207-3_15)
14. Rzaša, W.: Predicting performance in a PaaS environment: a case study for a web application. *Comput. Sci. [S.l.]* **18**(1), 21–39 (2017). <http://dx.doi.org/10.7494/csci.2017.18.1.21>
15. Jensen, K., Kristensen, L.: *Coloured Petri Nets. Modeling and Validation of Concurrent Systems*. Springer, Heidelberg (2009)
16. Rzaša, W.: Timed colored Petri net based estimation of efficiency of the grid applications. Ph.D. thesis, AGH University of Science and Technology, Kraków, Poland (2011)
17. Jamro, M., Rzonca, D., Rzaša, W.: Testing communication tasks in distributed control systems with SysML and Timed Colored Petri Nets model. *Comput. Ind.* **71**, 77–87 (2015). <http://dx.doi.org/10.1016/j.compind.2015.03.007>
18. Rzaša, W.: Combining timed colored Petri nets and real TCP implementation to reliably simulate distributed applications. In: Kwiecień, A., Gaj, P., Stera, P. (eds.) CN 2009. CCIS, vol. 39, pp. 79–86. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-02671-3_9](https://doi.org/10.1007/978-3-642-02671-3_9)

Computer Networks

24th International Conference, CN 2017, Łądek Zdrój,
Poland, June 20–23, 2017, Proceedings

Gaj, P.; Kwiecień, A.; Sawicki, M. (Eds.)

2017, XVIII, 460 p. 212 illus., Softcover

ISBN: 978-3-319-59766-9