Performance Evaluation of the Message Queue Protocols to Transfer Binary JSON in a Distributed CNC System

Maxim Ya. Afanasev, Yuri V. Fedosov, Anastasiya A. Krylova, Sergey A. Shorokhov
School of Computer Technologies and Control, Faculty of Control Systems and Robotics,
Instrumentation Technologies Department, ITMO University
49 Kronverksky Pr., St. Petersburg, 197101, Russia
Email: amax@niuitmo.ru, yf01@yandex.ru, {ananasn94, stratumxspb}@gmail.com

***Abstract*—This paper presents a performance study of the message queue protocols that are used for binary encoded JSON data transmission in a distributed network, which is the foundation of the CNC system. The distributed architecture of the control system makes it possible to simplify the integration of automated and robotic equipment into the cyber-physical production system (CPPS) by creating an open program interface for the interaction between the components of a distributed CNC system. For effective interaction, a unified protocol is needed that combines the high performance of TCP sockets with the possibility of many-to-many routing between endpoints. A message queue was selected as the protocol; the main format of the transmitted data is JSON. A comparative analysis of the bandwidth of TCP and UNIX sockets was performed; the average bandwidth of various network connections used in the distributed CNC system was determined. The maximum compression ratio for various JSON packers was calculated. The dependence of the compression ratio and the packing/unpacking time on the size of the transmitted data is obtained. The four message queue protocols ware compared. Based on the analysis, the most productive bundle of the data transmission pattern and the package/unpacking algorithm was determined. A technique for determining the total bandwidth of the message queue and network overhead for various network scenarios has been developed.**

I. Introduction

Considering the development trends of modern production it should be noted that new organization forms of interconnection between technological equipment components are gradually phasing out the older ones. Increasingly, enterprises create unified equipment control centres and introduce novel methods of production process control. The boundaries between physical components (such as machine tools, industrial robotics, etc.) and the virtual model are getting smoother. Consequently, a new term CPPS (Cyber-Physical Production System) has appeared [1].

CPPS is an information technology concept, which involves the integration of computing resources with physical processes. Obviously, a computer network acts as a go-between in such systems. The network is the CPPS component that needs to meet the strictest requirements because workability of the whole system depends on the network reliability and fault-tolerance. Currently, flexible decentralized productions slowly replace centralized networks integrated with different automation tools. In other words, interoperability is phasing out integration.

However, machine tools with CNC systems and industrial robotics are still the main components of automated production. Such devices are quite complex and its integration via SCADA just complicates production control process. A monolithic architecture of the system for machine tools and robotics control leads to the monolithic SCADA. Given the above does not match the main CPPS principle—simplification of the production process, creating a dynamic distributed environment and carrying out appropriate actions autonomously. Thus, we can conclude that a novel distributed CNC equipment system architecture should be developed in order to create a highly effective CPPS.

The base of such architecture is the open interface that allows hardware components stay autonomous but at the same time able to communicate with other components if necessary. Obviously, it is impossible to create such interface without any unification of the data transferring protocol. This protocol should work above all protocols and provide the ability to transparently exchange information between operation system processes as well as between heterogeneous network nodes that can act as a software service running under the operating system and a physical controller [2].

All these requirements are met by message queues. A message queue is an asynchronous communication protocol so the receiver and sender do not communicate directly and use a message queue instead. Generally, message queues have a message size constraint. Message queues improve system fault-tolerance as messages stay in the queue and can be handled even if the transmitter node fails.

Message queues guarantee message delivery, at least until one node is active and can handle it, and do not violate message order. In academia, a queue becomes both a handy instrument to implement complex asynchronous message-driven systems and it is a wide-ranging topic in computer science research. In other words, the authors consider a queue of two sides.

978-1-5386-0837-1/17/$31.00 ©2017 IEEE

The first one copes with an inner queue optimization. There are many papers on improved queuing schedulers that were found by a queue algorithm modification [3], [4]. For example, K. S. Kahlon modified a weighted fair queuing algorithm for the WiMAX network that contains both a real and a non-real-time traffic [5]. Also, queues are used in energy-awake scheduling, so Matthew Andrews and Lisa Zhang offered a rate-adaptive version of the Weighted Fair Queuing scheduling algorithm and proved its energy consumption [6]. In order to gain a high performance, researchers started developing novel hybrid genetic queuing algorithms for scheduling [5], [7].

The other one solves some application domain tasks by queue employment. For example, J. Acosta-Cano uses the message queuing system MSMQ and web service-oriented solutions for coupling a shop floor software with a set of production equipment [8]. In order to create an enterprise messaging system, Gursev Singh Kalra used ActiveMQ along with the Java Messaging Service [9]. Researchers from Kielce University of Technology implemented a wireless network for mobile robot applications using MQTT protocol [10]. Also, ZeroMQ framework usage is described in several papers [11]–[13]. Message queues have become pretty ubiquitous and used in many spheres including manufacturing [14], [15].

This article gives the comparative analysis and performance evaluation of different message queue protocols for binary JSON data transmission. Currently, this protocol is the industry standard for both data storing and big data transmission. Also, it is widely employed for the creation of distributed networks of the Internet of Things (IoT) as well as web services. The format is textual, which simplifies the work with it. At the same time, there is a sufficient number of software tools that make it possible to implement binary serialization of JSON in order to reduce the message size and increase the message queue, which in turn increases the responsiveness of the distributed CNC system. The rest of the article will be organized as follows. In Section II, specific motivations for using a message queue in the distributed CNC system will be discussed. The basic network scenarios will be presented in Section III. A description of the research methodology will be presented in Section IV. The results are displayed in Section V along with the analysis. The main findings will be summarized and main conclusions will be drawn in Section VI.

II. Motivation

The distributed CNC system that is part of the CPPS is a challenge for networks and computers. It is assumed that the protocol based on the message queue will become the main protocol of interaction within a large production environment. Accordingly, there are numerous scenarios for using this protocol as well as limitations imposed on software and hardware.

Obviously, the distributed CNC system will be implemented on the basis of the existing corporate computer network, consisting of a wide variety of computers, programmable logic controllers, low-level controllers, servers, etc. As a medium for data transmission, a high-speed wire connection (based on copper or optical cables) or wireless can be used. This is especially important in production because technological equipment can be located in sufficiently large workshops where it is economically inexpedient to use only a wired connection.

It should not be forgotten that the corporate network of an enterprise could be geographically distributed. Today, the industry is increasingly using cloud technologies for such things as performing complex calculations related to the preparation of control programs for CNC equipment or the creation of virtual models of equipment.

All this leads to the necessity of economical use of network resources to achieve the maximum performance of a distributed CNC system. The purpose of this work is to estimate the network overhead associated with the packaging and unpacking of binary JSON data as well as transfer it through the message queue. A comparison will be made between the bandwidth of the message queue and "raw" TCP sockets both for transmission between the operating system processes and for transmission over various network connections. As a result, the fastest variant of data transmission in the distributed CNC system under consideration will be determined [16]–[20].

III. Network Scenarios

The possible scenarios of network interaction between components of a distributed CNC system were considered.

A. Operation System Processes Interaction

It is proposed that the distributed CNC system components can be installed on a personal computer (PC) or a general-purpose server. The possibilities of inter-process communication (IPC) for POSIX compliant operating systems were considered. This scenario is the most common in the implementation of high-level CNC logic. For example, the G-code control program-parsing module interacts with trajectory optimization and moves the planning module in this way. It is obvious that both of these modules do not solve tasks in real time, and as a rule they can be installed on ordinary PCs.

B. Interaction Hardware and Software Modules in a Heterogeneous Computer Network

This is the most common interaction scenario in a distributed CNC network. It is assumed the possibility of transparent communication between low-level controllers, computers and general-purpose servers. Each of the participants of this interaction can act both as a client and as a server. It can be implemented as a two-point or a broadcast connection. For example, a low-level programmed logical controller that collects data from sensors and broadcasts them to the network can act as a data source.

In this case, all nodes interested in obtaining these data will act as the receivers: motion controllers, servers for collecting statistical data on the production process, interface modules of the operator, etc. Another example is a video camera that broadcasts a video signal from the processing area. This signal can be displayed on the physical monitor or on the user

interfaces widget, transmitted to machine vision system for analysis, etc. On the other hand, specialized control signals are transmitted directly between two predetermined nodes.

C. Interaction with the User (Operator)

The users interface is one of the most important parts of every CNC system. The use of the decentralized approach for the design of such systems provides enormous advantages over the classical monolithic systems. In the classic systems, the users interface is inseparable from the CNC controller. In early CNC systems, user interaction was mostly done by using physical control components such as buttons, switches, character indicators, and so on. But in modern systems the interface becomes virtual, that is, it is displayed on the CNC system screen. The number of physical control components is reduced, and only components that provide operational safety (for example, an emergency stop button) or ease of management (for example, a pendant hand-wheel) remain.

It is obvious that the use of a virtual interface that is tightly bound to the controller is inconvenient and inexpedient. Such interfaces have low flexibility and it is hard to adapt for users needs as well as almost impossible to organize remote management. Therefore, to create the operators interface for a distributed CNC system, the use of web technologies is assumed.

In this case, the main control terminal will be a web browser installed either on a personal computer or on a mobile device. At the same time some of the computing tasks will be transferred directly to the browser, which requires the ability to receive data from the distributed network directly through it. This creates a separate network scenario because the client web applications use their own stack of technologies that impose certain restrictions on the operation of the distributed CNC system.

IV. Methodology

The measurements described in this article were carried out by transferring data between two processes connected via TCP/IP. This allowed us to create a stable test environment, which makes it possible to exclude overhead costs associated with the preparation of data. First, a comparison of the methods of inter-process communication was carried out. The performance of the TCP/IP protocol was analysed and compared to UNIX sockets and shared memory. Then load testing of the channels for various network scenarios was performed.

Furthermore, the performance analysis of various binary JSON data packaging/unpacking tools was performed. After that, the bandwidth analysis for several message queue libraries was performed on the basis of which the most productive variant was selected. At the end, the final calculation of the maximum bandwidth for all the network scenarios under consideration was performed.

A. Test Set-ups

Four test set-ups were used, representing different architectures and performance components of a distributed CNC system:

1) *General PC* is a low-performance computer used in a distributed CNC system to solve a wide range of tasks. Specifications: Intel Core i5 CPU M 520 @ 2.40 GHz, 4 GiB RAM. Operating system: Ubuntu 16.10, GNU/Linux 4.8.0-41 x86-64.
2) *Laptop* is a portable computer that can be used by the operator of a distributed CNC system as a terminal, or for remote control. Specifications: MacBook Pro, Intel Core i5 CPU @ 2.60 GHz, 8 GiB RAM. Operating system: OS X 10.10.5, Darwin 14.5.0.
3) *Server* is a high-performance computer used for the most resource-intensive tasks of the distributed CNC system. Specifications: 2 x Intel Xeon E5620 CPU @ 2.40 GHz, 32 GiB RAM.
4) *Programmable logic controller (PLC)* is a embedded system-on-chip used to implement low-level algorithms in real-time mode. Specifications: Amlogic S905 Quad Core Cortex-A53 @ 1.5 GHz 64bit ARMv8 CPU with Mali-450 GPU, 2 GiB RAM. Operating system: Ubuntu 16.04 LTS.
5) *Virtual private server (VPS)* is a cloud-based virtual machine used to optimize the processing power of a distributed CNC system by transferring part of the software components to the dedicated servers. Specifications: Intel Xeon CPU E5645 @ 2.40 GHz, 512 MiB RAM. Operating system: Ubuntu 16.04.2 LTS, GNU/Linux 2.6.32-042stab120.18 x86-64.

B. Libraries and Frameworks

The performance evaluation required analysing the development tools (software libraries) for packing/unpacking the binary JSON data and the message queue implementation, and selecting the ones most appropriate to solve the given problem. The following requirements for libraries and frameworks for packaging/unpacking binary JSON data were formulated:

- Open source license
- Support for programming languages C, Python, Go and JavaScript (the main programming languages used to implement the distributed CNC system)
- A small amount of the software library
- High-quality documentation

As a result of the performed analysis, the following libraries were selected for testing: MessagePack, BSON (Binary JSON), UBJSON (Universal Binary JSON), CBOR (Concise Binary Object Representation).

The requirements for message queue libraries are similar to the requirements for the packaging/unpacking libraries; additionally an open API is required to add new transport protocols. As a result of the analysis, the following libraries were selected for testing: ZeroMQ, NanoMsg, RabbitMQ, and ActiveMQ. The first two can work without a dedicated message broker. For others, the use of a special message broker is mandatory.

V. Results

A. Comparing the Bandwidth of TCP and UNIX Sockets

To compare the bandwidth of TCP and UNIX sockets, the following methods were used:

1) socat utility, transfer from memory to disk, shared memory is used.

```
$ sudo dd if=/dev/urandom of=/dev/shm/data.dump
    bs=1M count=512
$ sudo socat -u -b32768 UNIX-LISTEN:/tmp/unix.sock
    ./data.dump &
$ sudo socat -u -b32768 "SYSTEM:dd
    if=/dev/shm/data.dump bs=1M count=512"
    UNIX:/tmp/unix.sock
$ sudo rm -rf /dev/shm/data.dump
```

2) `socat` utility, transfer from memory to memory, shared memory is used.

```
$ sudo dd if=/dev/urandom of=/dev/shm/data.dump
    bs=1M count=512
$ sudo socat -u -b32768 UNIX-LISTEN:/tmp/unix.sock
    /dev/shm/data.dump.out &
$ sudo socat -u -b32768 "SYSTEM:dd
    if=/dev/shm/data.dump bs=1M count=512"
    UNIX:/tmp/unix.sock
```

3) `socat` utility, transfer from memory to the null device, shared memory is used.

```
$ sudo dd if=/dev/urandom of=/dev/shm/data.dump
    bs=1M count=512
$ sudo socat -u -b32768 UNIX-LISTEN:/tmp/unix.sock
    /dev/null &
$ sudo socat -u -b32768 "SYSTEM:dd
    if=/dev/shm/data.dump bs=1M count=1024"
    UNIX:/tmp/unix.sock
```

4) `socat` utility, transfer from `/dev/zero` (special file that provides as many null characters as are read from it) to the null device.

```
$ sudo socat -u -b32768 UNIX-LISTEN:/tmp/unix.sock
    /dev/null &
$ sudo socat -u -b32768 "SYSTEM:dd if=/dev/zero
    bs=1M count=512" UNIX:/tmp/unix.sock
```

5) `nc` utility, data transfer through a common file.

```
(SEND) $ sudo pv /dev/zero | nc -U /tmp/socket
(RECV) $ sudo nc -lU /tmp/socket > /dev/null
```

6) `iperf` utility, data transfer through a TCP socket, TCP window size is 2.5 MB.

```
(SEND) $ iperf -s; (RECV) $ iperf -c localhost
```

The results of the comparative analysis are presented in Table I. It can be seen that the best way for inter-process communication in POSIX-compatible operating systems is to use TCP sockets.

B. Determining the Average Bandwidth of the Network Connections

To determine the average bandwidth of the network connections, the following methods were used:

1) HTTP server, 10 GB of data is transferred.

```
(SEND) $ dd if=/dev/urandom of=test bs=1024K
    count=10000
(SEND) $ sudo python -m SimpleHTTPServer 80
(RECV) $ wget <server-ip>/test
```

TABLE I
Inter-process Communication Bandwidth Comparison

Test Set-up	Bandwidth, Gbit/s					
	Meth. 1	Meth. 2	Meth. 3	Meth. 4	Meth. 5	Meth. 6
General PC	0.69	5.76	7.46	12.03	6.33	22.3
Laptop	1.79	4.47	5.98	10.18	3.65	23.2
Server	2.41	2.76	2.75	3.23	3.13	31.6
PLC	1.98	1.96	2.38	2.86	2.04	4.61

2) `iperf` utility, TCP window size is 84.3 KB, 10 GB of data is transferred.

```
(SEND) $ ipref -s; (RECV) $ ipref -c <send-ip>
```

3) `netperf` utility, 10 GB of data is transferred.

```
$ netperf -H <any-ip-in-network> -4 -T TCP_STREAM
```

4) `nuttcp` utility, 10 GB of data is transferred.

```
(SEND) $ nuttcp -S
(RECV) $ nuttcp -vvv -i1 <send-ip>
```

5) `nttcp` utility, 10 GB of data is transferred.

```
(SEND) $ nttcp -i; (RECV) $ nttcp -t -T <send-ip>
```

The results of the comparative analysis are presented in Table II. Obviously, the use of the HTTP protocol leads to the emergence of additional overhead, so this method is excluded from the calculation of the average bandwidth.

TABLE II
Bandwidth of the Network Connections

Conn.	Bandwidth, Mbit/s					
	Meth. 1	Meth. 2	Meth. 3	Meth. 4	Meth. 5	Avg.
Wi-Fi*	**5.25**	5.95	8.00	6.86	5.47	6.57
Cable*	**559.00**	941.00	907.77	941.27	940.72	932.69
Cloud**	**12.44**	13.70	13.67	13.60	14.42	13.85

*connection between *General PC* and *Server* in the local network

**connection between *Server* and *VPS* via the Internet

C. Performance Evaluation of the JSON Packers

Fig. 1 shows the dependence of the compression ratio on the JSON file size. The JSON files consist of an ordered list of elements. Each element consists of a field name and a value. Field names are strings. Values include numbers, strings, Booleans, or null objects. All tests are written in Python. The figure shows that for all libraries the dependence of the compression ratio on the JSON file size is practically not observed.

Then the dependence of the packing/unpacking rate on the size of the JSON file was obtained. The results are shown in Fig. 2 and 3. As one can see, for all the libraries under consideration this dependence is linear. The CBOR library demonstrated the best time for both packaging and unpacking. The next step was to test the performance of the CBOR libraries, which were written in different programming languages. There were 1000 packaging/unpacking cycles; the size of the original JSON file was 15 MB. The test results are shown in Table III.

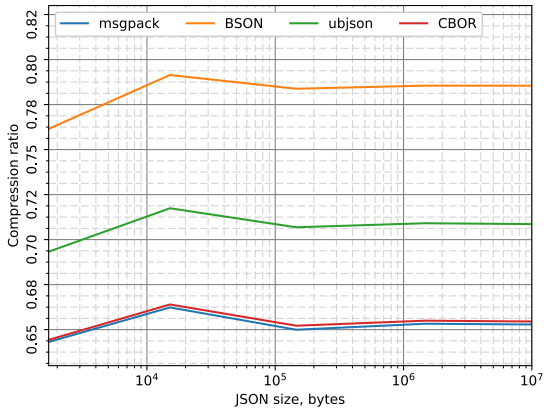

Fig. 1. Compression ratio as a function of the JSON file size.

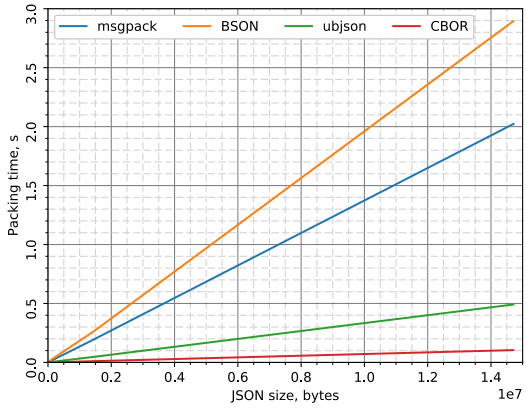

Fig. 2. Packing time as a function of the JSON file size.

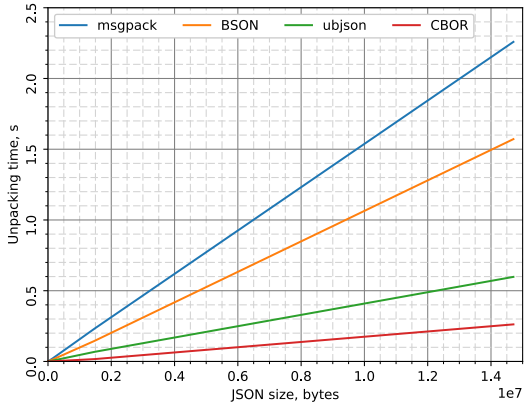

Fig. 3. Unpacking time as a function of the JSON file size.

D. Performance Evaluation of the message queue protocols

Fig. 4 shows the dependence of the bandwidth on the message size. For each method, one million messages were transmitted, and inter-process communication is used. Based on the results of the analysis, we can conclude that the most rational message size ranges from 1 KB to 100 KB. The NanoMsg and ZeroMQ libraries demonstrated the best performance. NanoMsg was selected as the main message transfer protocol

TABLE III
BANDWIDTH OF THE CBOR LIBRARIES

Library	Bandwidth, Mbit/s	
	Packing	Unpacking
Python 3.5.2 CBOR	112.73	19.13
Python 3.5.2 CBOR2	7.59	4.30
C libcbor (GCC 6.2.0 20161005)	244.13	273.07
JavaScript cbor (Google Chrome 56.0.2924.87)	76.19	100.21

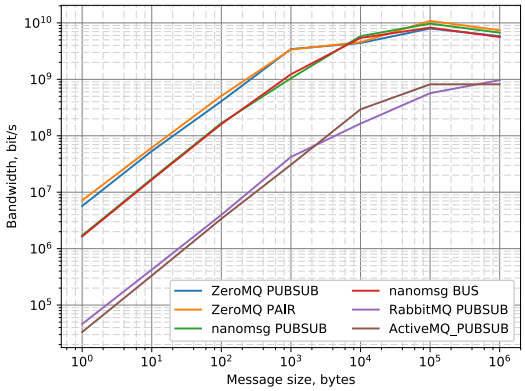

Fig. 4. Message queue protocols bandwidth as a function of the message size.

in the distributed CNC system under consideration. The main reasons for this solution are better compatibility with POSIX and a more flexible API.

Table IV shows the results of measuring the bandwidth of the NanoMsg protocol for various network connections. All tests were written in Python where the message size is 1 KB and one million messages were sent in each test.

TABLE IV
BANDWIDTH OF THE NANOMSG PROTOCOL

Library	Bandwidth, Mbit/s	
	Nanomsg PUBSUB	Nanomsg BUS
Wi-Fi	5.93	7.34
Cable	880.74	880.38
Cloud	15.18	14.98

E. Overall Evaluation of the Performance

To give an overall evaluation of the performance of the message queue protocol, we obtained the formula for the average bandwidth (1).

$$\mathcal{B}_a = \frac{\mathcal{M}}{t} = \frac{\mathcal{M}}{t_p + t_t + t_u} = \frac{\mathcal{M}}{\mathcal{M}/\mathcal{S}_p + \gamma\mathcal{M}/\mathcal{B}_t + \mathcal{M}/\mathcal{S}_u} = \\ = \frac{\mathcal{M}}{\frac{\mathcal{M}(\mathcal{B}_t\mathcal{S}_u) + \gamma\mathcal{M}(\mathcal{S}_p\mathcal{S}_u) + \mathcal{M}(\mathcal{S}_p\mathcal{B}_t)}{\mathcal{S}_p\mathcal{B}_t\mathcal{S}_u}} = \frac{\mathcal{S}_p\mathcal{B}_t\mathcal{S}_u}{\mathcal{B}_t\mathcal{S}_u + \gamma\mathcal{S}_p\mathcal{S}_u + \mathcal{S}_p\mathcal{B}_t}, \quad (1)$$

where $\mathcal{M}$ is a message size, t is total point-to-point time, t_p is packing time, t_t is transfer time, t_u is unpacking time, $\mathcal{S}_p$ is a JSON data packing speed, $\mathcal{S}_u$ is a JSON data unpacking speed, $\mathcal{B}_t$ is a message queue bandwidth, γ is a JSON compression ratio.

Frequently, the Python libraries demonstrate a performance that is approximately equal to the performance of C libraries. This is due to the fact that these libraries are just wrappers of the C libraries. In this case, the use of the Python libraries is preferable because of the simpler syntax of the Python programming language. However, the Python libraries under consideration did not show sufficient performance, so it is obvious that a C library was chosen to implement the server part. When implementing the client part, the JS library was used, since this is the only option for implementing programs in the browser.

Table V shows the results of calculations. The calculations show that average bandwidth of the described link channel is in the range of 6 % (the worst case, packaging/unpacking is required at both endpoints) to 145 % (the best case, packing/unpacking is not required at all, both endpoints use binary JSON data, which implies the ability to compress messages due to the fact that the binary JSON format takes up less space compared to the text JSON).

TABLE V
COMMUNICATION BANDWIDTH, CABLE CONNECTION, $\gamma = 0.65$

$\mathcal{S}_p \setminus \mathcal{S}_u$	JS library	C library	No unpacking
JS library	n/a^*	**57.06**	70.94
		(**6.22** %)	(7.60 %)
C library	67.50	117.70	203.18
	(7.24 %)	(12.62 %)	(21.78 %)
No packing	93.31	227.27	**1354.98**
	(10.00 %)	(24.37 %)	(**145.28** %)

*connection between client is not implemented in the system under consideration

VI. CONCLUSION

In this paper, we examine the performance of the transmission of JSON data through the message queue, which is the foundation of a distributed network of a CNC system. It was shown that the best time of packing/unpacking and the compression ratio of the JSON data were demonstrated by the CBOR library.

It was determined that the compression ratio does not depend on the volume of the original JSON file. Based on the analysis of advantages and disadvantages, the best message queue protocol is NanoMsg. Based on the results of the analysis, it was also concluded that the most suitable message size is from 1 KB to 100 KB.

The calculations have been made of the overall performance of the bundle of the NanoMsg protocol and the CBOR packer. Calculations showed that the bandwidth of this data channel is in the range of 6 % to 145 % of the average network bandwidth when using the raw TCP sockets, which can be considered a good result for this type of software.The source code of the tests is available at `github.com/futoke/message-queue-protocols`.

REFERENCES

[1] B. Vogel-Heuser, C. Diedrich, D. Pantförder, and P. Göhner, "Coupling heterogeneous production systems by a multi-agent based cyber-physical production system," in *2014 12th IEEE International Conference on Industrial Informatics (INDIN)*, 2014, pp. 713–719.

[2] M. Y. Afanasev and A. A. Gribovskiy, "Kontseptsiya adaptivnoy platformy tekhnologicheskogo oborudovaniya [An adaptive platform of technological equipment concept, in Russian]," *Isvestiya vusov. Priborostroenye*, vol. 58, no. 4, pp. 268–272, 2015.

[3] P. Valente, "Reducing the execution time of fair-queueing packet schedulers," *Computer Communications*, vol. 47, pp. 16–33, 2014.

[4] L. Rizzo and P. Valente, "On service guarantees of fair-queueing schedulers in real systems," *Computer Communications*, vol. 67, pp. 34–44, 2015.

[5] Akashdeep, K. Kahlon, and M. Kaushal, "Analysis of a queue length aware and latency guaranteed fuzzy-based adaptive resource allocator for WiMAX networks," *Optik—International Journal for Light and Electron Optics*, vol. 127, no. 1, pp. 357–367, 2016.

[6] M. Andrews and L. Zhang, "Rate-adaptive weighted fair queueing for energy-aware scheduling," *Information Processing Letters*, vol. 114, no. 5, pp. 247–251, 2014.

[7] S. Rashidi and S. Sharifian, "A hybrid heuristic queue based algorithm for task assignment in mobile cloud," *Future Generation Computer Systems*, vol. 68, pp. 331–345, 2017.

[8] J. Acosta-Cano and F. Sastrónáguena, "Loose coupling based reference scheme for shop floor-control system/production-equipment integration," *Journal of Applied Research and Technology*, vol. 11, no. 3, pp. 447–469, 2013.

[9] G. S. Kalra, "Threat analysis of an enterprise messaging system," *Network Security*, vol. 2014, no. 12, pp. 7–13, 2014.

[10] R. Kazala, A. Taneva, M. Petrov, and S. Penkov, "Wireless network for mobile robot applications," *IFAC-PapersOnLine*, vol. 48, no. 24, pp. 231–236, 2015.

[11] K. Kirsanov, "Software architecture of control system for heterogeneous group of mobile robots," *Procedia Engineering*, vol. 100, pp. 278–282, 2015.

[12] B. Goertzel, D. Hanson, and G. Yu, "A software architecture for generally intelligent humanoid robotics," *Procedia Computer Science*, vol. 41, pp. 158–163, 2014.

[13] V. Andreev, K. Kirsanov, P. Pletenev, Y. Poduraev, V. Pryanichnikov, and E. Prysev, "Technology supervisory control for mechatronic devices via the internet," *Procedia Engineering*, vol. 100, pp. 33–40, 2015.

[14] D. Stock, M. Stöhr, U. Rauschecker, and T. Bauernhansl, "Cloud-based platform to facilitate access to manufacturing it," *Procedia CIRP*, vol. 25, pp. 320–328, 2014.

[15] C. Morariu, O. Morariu, T. Borangiu, and S. Raileanu, "Manufacturing service bus integration model for implementing highly flexible and scalable manufacturing systems," *IFAC Proceedings Volumes*, vol. 45, no. 6, pp. 1850–1855, 2012.

[16] X. Huang, "Enhancing STEP-NC compliant CNC controller for distributed and reconfigurable environment in production line," in *2010 International Conference on Computer, Mechatronics, Control and Electronic Engineering*, vol. 2, 2010, pp. 106–109.

[17] V. Lesi, Z. Jakovljevic, and M. Pajic, "Towards plug-n-play numerical control for reconfigurable manufacturing systems," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–8.

[18] H. A. ElMaraghy, "Flexible and reconfigurable manufacturing systems paradigms," *International Journal of Flexible Manufacturing Systems*, vol. 17, no. 4, pp. 261–276, 2005.

[19] K. Latif and Y. Yusof, "New method for the development of sustainable STEP-compliant open CNC system," *Procedia CIRP*, vol. 40, pp. 230–235, 2016.

[20] H. Hong, D. Yu, X. Zhang, and L. Chen, "Research on the data hungry problem in CNC system based on the architecture of real-time multitask," in *2011 3rd International Conference on Computer Research and Development*, vol. 2, 2011, pp. 103–108.