Can SCA 4.1 Replace STRS in Space Applications?

Ran Cheng[1], Li Zhou[1*], Qi Tang[1], Dongtang Ma[1], Haitao Zhao[1], Shan Wang[1], Jibo Wei[1]
[1]College of Electronic Science and Engineering, National University of Defense Technology, China
[*]Email: zhouli2035@nudt.edu.cn

***Abstract*—Software Defined Radio (SDR) has gained a great deal of attentions in both civil and military applications. As the most important SDR standard, Software Communications Architecture (SCA) has been accepted and extensively adopted in the development of radio systems since it was first released. A new standard, Space Telecommunications Radio System (STRS), was developed by National Aeronautics and Space Administration (NASA) to align with the mission requirements due to the drawbacks of SCA 2.2.2. In this paper, we explore whether SCA 4.1, the latest release version of SCA, can meet the requirements of space application by making a comparison between SCA 4.1 and STRS in static memory occupation, inter-components communication delay, waveform deployment delay and waveform switching delay. We have tested our realizations of full SCA 4.1, lightweight SCA 4.1 and STRS in our testbed extensively, where full SCA 4.1 is considered as a benchmark. The experiment results show that although lightweight SCA 4.1 has relative larger demand on static memory occupation, longer waveform loading and switching delay, its inter-components communication could be as efficient as STRS.**

Keywords—Software Defined Radio, Software Communications Architecture, Space Telecommunications Radio System, efficiency

I. INTRODUCTION

The concept of Software Defined Radio (SDR) is adopted in implementing communication systems in the space environment with three main goals, reducing the development cycle-time, reducing the development cost, and increasing the communication flexibility among space radios and ground stations. However, implementing SDR in space environment comes with some challenges that need special concerns. Firstly, the space communication system is a strict size, weight and power (SWaP) constrained system, which limits its total capability. Secondly, the chips for space use is developing much slower than chips for commercial use. For example, the central processing unit (CPU) and memory chips for space use are quite expensive and have much lower clock speeds, smaller cache and memory size due to the strict reliability requirements in the space environment. Finally, there are urgent demands for the reuse of both the platform and software. The platform reuse allows different applications to be deployed on the same platform, enabling the innovation of the system. The software reuse reduces the cycle-time of reprogramming and speeds up the date-to-market [1][2].

Software has been contributed to the development of communication systems for decades, which brings a lot of advantages compared with hardware-based or firmware-based system development. Among the advantages, modularity, lower development costs and shorter development time are the best benefits and highlights. Based on the software-based development paradigm used in SDR-based system, some hardware problems could be turned into software problems [3].

Quite a few SDR standards have been proposed since the concept of SDR was coined, e.g., Software Communications Architecture (SCA) [4], GNU Radio [5], Abstraction Layer and Operating Environment (ALOE) [6] and Space Telecommunications Radio System (STRS) [7] etc. SCA is the most widely adopted standard in SDR area and has been deployed in thousands of radios worldwide [8], while STRS is dedicated for space SDR development and has been deployed on the space communications and navigation (SCAN) testbed of NASA [9].This paper is designed to verify if appropriate realization of the latest version of SCA can catch up with the efficiency indexes of STRS[11]. Thus, we first give a brief introduction of highlights of SCA 2.2.2, STRS and SCA 4.1 in a chronological sequence.

A. SCA 2.2.2

The final specification of SCA 2.2.2 was proposed in May, 2006. It describes the core framework which provides the infrastructure to allow software components to plug-and-play and also describes set of rules and application program interfaces (APIs) that must be used by developers of SCA-compliant software components. Fig. 1 presents the SCA software execution model. As depicted in the figure, the hardware is separated from the software through the hardware abstraction layer (HAL), making the operation environment (OE) independent of the hardware.

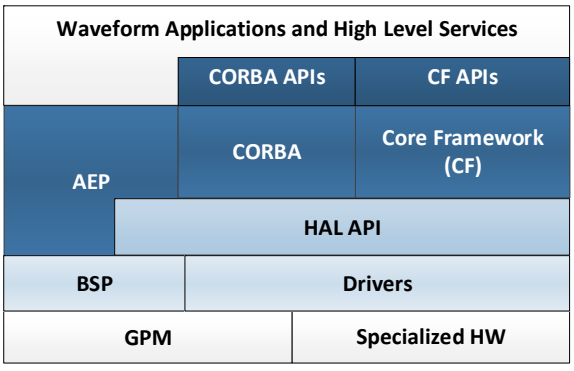

Fig. 1. SCA 2.2.2 software execution model [11].

Due to the stringent portability and interoperability requirements, SCA radios generally contain much more software than previous generations of SDRs. To be more

specific, SCA radios use some software modules to process the input and output signals. Most SCA radios contain several millions lines of source code. As a result, SCA radios generally take longer time to boot, use more memory and has lower communication speed [8].

In order to support the ability of multi-platform communication, common object request broker architecture (CORBA) standard is adopted. It brings great benefits to support portability and interoperability. However, its high demands on memory consumption and CPU occupation make it an obstacle of adopting SCA in space applications.

B. STRS

The most recent specification of STRS, NASA-STD-4009, was approved in May, 2014. In order to benefit from the SDR technology in space applications, NASA proposed STRS at a historic moment [12]. Fig. 2 presents the STRS software execution model. As can be seen from the figure, the waveform applications and high level services are abstracted from the hardware by a common, consistent operating environment which includes a set of published APIs, including POSIX APIs, STRS APIs and HAL APIs [12].

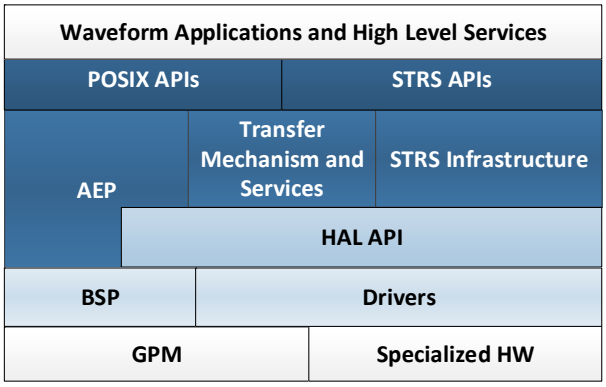

Fig. 2. STRS software execution model [12].

STRS is designed to be able to evolve for each generation of space communication. The standard provides many flexibility to meet a variety of requirements of users and make it possible to reuse components in previous versions, which reduces the cost and the risk of deploying SDRs for space missions.

C. SCA 4.1

The Joint Program Executive Office (JPEO) for the Joint Tactical Radio System (JTRS) put forward SCA 4.0 in 2009. With multiple recommendations incorporated, it is replaced by the latest version, SCA 4.1, which was announced in August, 2015. Thanks to the great contributions from JTRS, JTNC, WInn Wireless and many other companies, the SCA keeps advancing, making it potentially applicable to resource-limited scenarios.

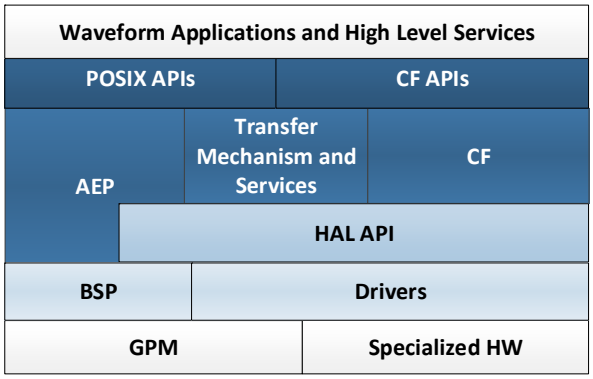

Fig. 3. SCA 4.1 software execution model [4].

The highlights of evolution from SCA 4.1 to SCA 2.2.2 can be summarized as follows.

Adopt "push model" behavior. The "push model" behavior reduces the total number of calls, which lowers the overall complexity and enhances the system efficiency.

Remove dependence on CORBA. SCA 4.1 has removed the CORBA requirements and replace it by a reconfigurable transfer mechanism with common APIs, so that the architecture can gain more flexibility and efficiency.

Add static registration behavior. SCA 4.1 provides both static registration and dynamic registration. The introduce of static registration can enhance the system performance and reduce overhead.

Provide Units of Functionality (UOF) and SCA Profiles. UOFs are able to omit unnecessary interfaces and modules, so that the object size can be reduced and the system performance can be improved.

Overall, SCA 4.1 has introduced many lightweight designs which enhance the flexibility and efficiency of the architecture. With these new features, it is supposed that SCA 4.1 has the potential to be deployed on space SDRs. Therefore, whether STRS is the only option to be deployed on the radio device in space environment and whether SCA 4.1 can be utilized in space environment still need to be explored.

To make these doubts clear, this paper makes a comparison between SCA 4.1 and STRS by different metrics, including static memory occupation, inter-components communication delay, waveform deployment delay and waveform switching delay.

The reminder of this paper is organized as follows. Section II presents our core framework designs for SCA 4.1 and STRS. Section III introduces our testbed implementation. Section IV presents the experiment results and section V concludes the paper.

II. CORE FRAMEWORK DESIGN

Fig. 4 and Fig. 5 illustrate the layered framework that we designed according to the standard of SCA 4.1 and STRS. In STRS, core framework is also known as STRS infrastructure.

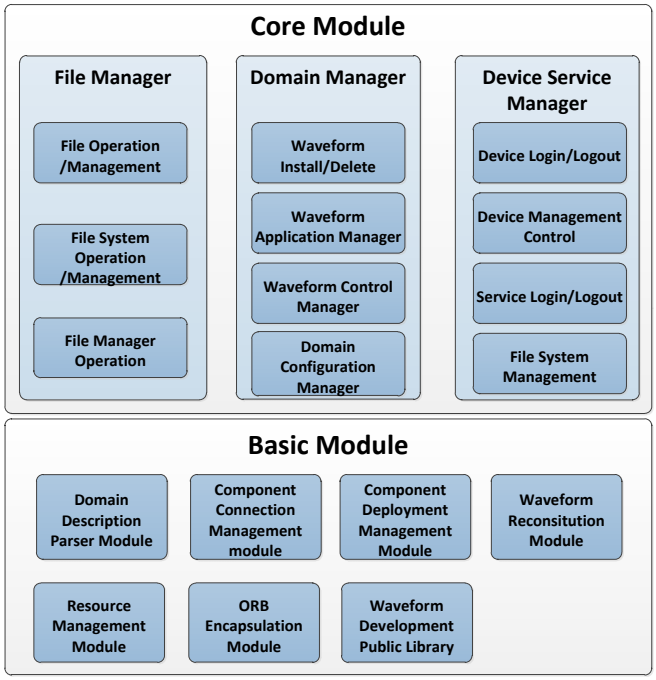

Fig. 4. SCA 4.1 layered framework.

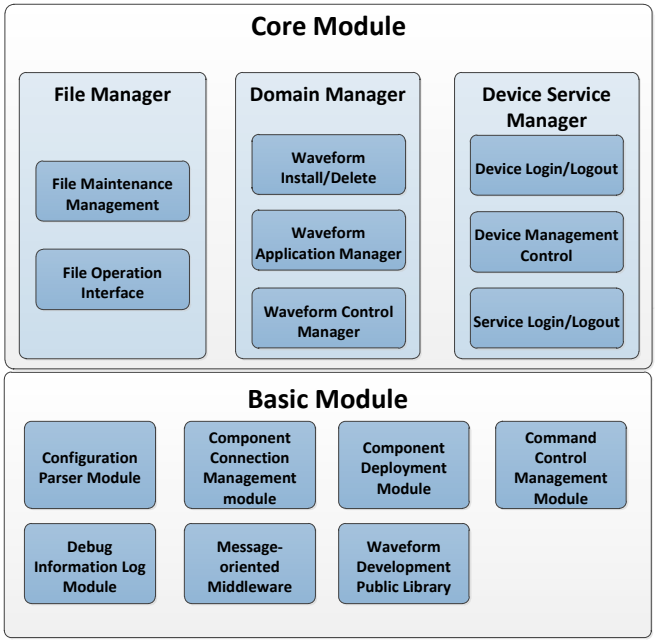

Fig. 5. STRS layered framework.

As can be seen from Fig. 4 and Fig. 5, STRS does not include the Domain Configuration Manager, File System Management and ORB encapsulation module. It is obvious that STRS has much less requirements than SCA, so that STRS owns a more lightweight core framework structure.

Fig. 6 and Fig. 7 presents the core framework interfaces of SCA 4.1 and STRS respectively. STRS core framework is written in C language and does not have the complex inheriting relationship as SCA, but only has some common infrastructure interfaces, which simplifies the realization of the core framework.

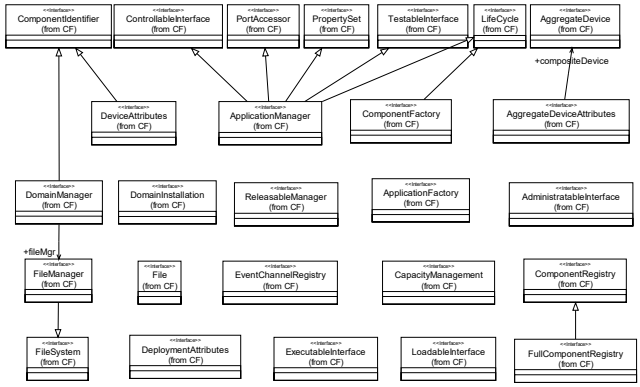

Fig. 6. SCA 4.1 core framework interfaces.

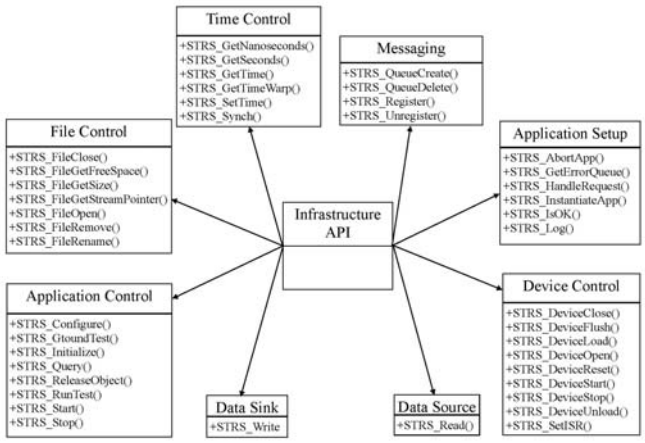

Fig. 7. STRS core framework interfaces.

The purpose of this paper is to compare the efficiency of SCA 4.1 and STRS. The comparison is carried out with respect to full SCA 4.1, lightweight SCA 4.1 and STRS. Lightweight SCA 4.1 is a profile with the lightest selection of interfaces, modules, which is considered as a potential replacement of STRS. Full SCA 4.1 is an realization with full CORBA Profile transfer mechanism and we consider it as a benchmark in our experiments [4][12].

Full SCA 4.1 still adopt the concept of agency, making users easier to access services provided by the component in the system. The agency mechanism is depicted in Fig. 8. The Remote Method Invocation (RMI) system establishes three abstract layers, which contain socket communication, the serialization and deserialization of variables and results, etc. The stub and skeleton combine to form the RMI frame protocol. The remote reference layer is adopted to find the communication object. The transport layer provides the interconnection of client and server based on the TCP/IP protocol.

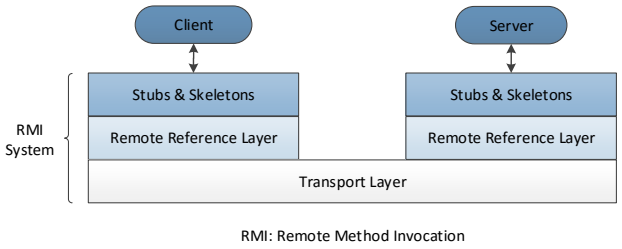

Fig. 8. Agency mechanism.

On the contrary, lightweight SCA 4.1 and STRS do not utilize the mechanism of agency, and launch all GPP components in a process address space. In this way, we could gain higher efficiency without the burden of serialization, deserialization and various protocols, etc.

III. TESTBED INPLEMENTATION

The experiments are carried out on our own testbed, which is shown in Fig. 9. The hardware ZLSDR-1000 is a high performance general SDR platform, which is designed for rapid prototype of waveforms. The main chip inside is ZYNQ 7030 which includes a dual-core of ARM Cortex-A9 (clock speed 667MHz) and a FPGA of Kintex-7 (logic cells 125K, DSP Slices 400). The DDR memory size is 1GB. This platform contains the Linux 3.17 operating system, which is convenient for users to develop applications in a Ubuntu system. All experiments are carried out on this testbed.

Fig. 9. General SDR platform (ZLSDR-1000).

There are several aspects that may affect the performance, e.g., packet size, total amount of packets and number of components, etc. Four metrics are adopted to investigate the performance between STRS, lightweight SCA 4.1 and full SCA 4.1, namely, static memory occupation, inter-components communication delay, waveform deployment delay and waveform switching delay. The delay is captured by adding timestamps at the input or output port of a component and the inter-components communication delay is presented in Fig. 10.

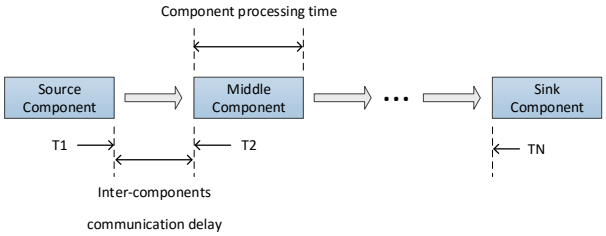

Fig. 10. Inter-components communication delay.

As shown in Fig. 10, the timestamp T1 is recorded before the source component sends a packet and the timestamp T2 is recorded after the middle component receives a packet. The inter-components communication time of a single link (T) can be computed as [T=(T2-T1)]. The experiments only record two timestamps (T1 and TN). The timestamp TN is recorded after the sink component receives the last packet. The difference of TN and T1 (TN-T1) approximately represents the amount time of all links in inter-components communication with the component processing time has been curtailed extremely to approach zero.

The waveform deployment delay and waveform switching delay are displayed in Fig. 11. As can be seen from Fig. 11, the waveform deployment delay is comprised of the delays of uploading and initializing the waveform, and the waveform switching delay includes of delays of waveform stop to waveform launch.

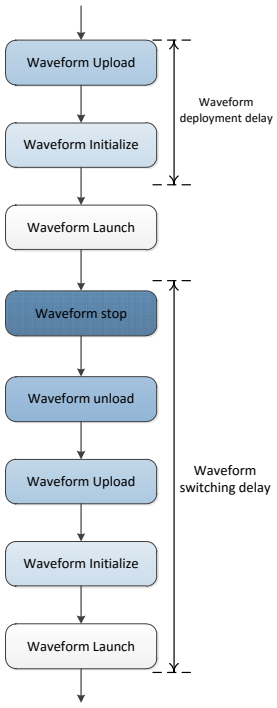

Fig. 11. Waveform deployment delay and waveform switching delay.

The efficiency of inter-components communication is a critical element for evaluating a SDR system and is influenced

by several aspects, including packet size, total amount of packets and number of components. The number of links of inter-components, which also affects the delay, is not considered as a metric as our test waveform is a cascaded flow waveform with no processing codes in each middle components [13].

IV. EXPERIMENT RESULTS AND ANALYSIS

To compare the performance between Lightweight SCA 4.1 and STRS, we carry out extensive experiment and discuss the results in this section. The results of full SCA 4.1 are presented as benchmarks. The experiment parameters are listed in Table I.

TABLE I. EXPERIMENT PARAMETERS

Parameter	Value
Packet size (bytes)	128, 256, 512, 1024, 2048, 4096
Amount of packets (×10^6)	1, 2, …, 10
Number of components	3, 4, …, 11

In the experiment, continuous packets are pushed from the source component to the sink component. The purpose is not only to evaluate the delay performance but also to conduct pressure test. The test is to evaluate the long-term performance of the system with different core frameworks and transfer mechanisms.

A. *Experiment Results*

Table II shows the static memory occupation of STRS, lightweight SCA 4.1 and full SCA 4.1. As shown in Table II, the memory occupation of STRS is much smaller than that of lightweight SCA 4.1 and full SCA 4.1, which is more suitable for memory limited radio systems.

TABLE II. STATIC MEMORY OCCUPATION COMPARISON

Implementation	Static memory occupation (MB)
STRS	4.77
Lightweight SCA 4.1	27.82
Full SCA 4.1	77.20

Table III illustrates the communication delay of each link between components in microsecond with different packet sizes. The number of component is 11. As shown in the table, the communication delay of STRS remains almost the same for different packet sizes. The communication delay of lightweight SCA increases with the packet size when it is small, and finally tends to saturate when the package size raises to 2048 bytes. The communication delay of full SCA 4.1 keeps increasing with the packet size. The delay of STRS and lightweight SCA is close, while that of full SCA 4.1 is over ten times larger than the other two frameworks. It should be noted that, in the experiment one packet is passed through 10 links from the transmitter to the receiver since the component number equals to 11.

TABLE III. INTER-COMPONENTS COMMUNICATION DELAY COMPARISON WITH DIFFERENT PACKET SIZES

Package size (bytes)	STRS (us)	Lightweight SCA (us)	Full SCA 4.1 (us)
128	17.30	17.20	189.00
256	17.30	18.40	189.00
512	17.30	19.00	196.00
1024	17.30	20.80	200.00
2048	17.30	21.00	216.00
4096	17.50	21.00	240.00

Table IV presents the total time consumption with different amount of packets. As shown in the table, that of STRS, lightweight SCA and full SCA 4.1 all increase linearly with the amount of data package in almost the same rate.

TABLE IV. TOTAL TIME COMSUPTION COMPARISON WITH DIFFERENT AMOUNT OF PACKETS

Amount of packets	STRS (s)	Lightweight SCA 4.1 (s)	Full SCA 4.1 (s)
100,000	17.50	21.00	240.00
200,000	40.10	42.20	476.00
300,000	66.70	63.60	715.00
400,000	81.60	85.00	946.00
500,000	101.90	105.40	1190.00
600,000	119.20	125.40	1441.00
700,000	143.10	147.80	1669.00
800,000	164.90	162.20	1905.00
900,000	184.10	191.00	2142.00
1,000,000	201.70	207.40	2399.00

Table V shows the communication delay of each link between components in microsecond. As shown in the table, that of STRS, lightweight SCA and full SCA 4.1 increase with the components number.

TABLE V. INTER-COMPONENTS COMMUNICATION TIME COMPARISON IN THE COMPONENT NUMBER

Number of components	STRS (us)	Lightweight SCA 4.1 (us)	Full SCA 4.1 (us)
3	15.50	23.00	175.00
4	16.00	19.67	186.67
5	16.75	20.50	197.50
6	16.00	19.80	208.00
7	16.50	21.33	211.67
8	16.43	20.14	218.57
9	17.00	20.75	227.50
10	17.22	20.22	233.33
11	17.50	21.00	240.00

Table VI shows the waveform deployment delay of each SDR architecture in millisecond. As shown in the table, the

deployment delay increases with the component number for all of the three SDR architecture used in the experiment.

TABLE VI. WAVEFORM DEPLOYMENT DELAY COMPARISON IN THE COMPONENT NUMBER

Number of components	STRS (ms)	Lightweight SCA 4.1 (ms)	Full SCA 4.1 (ms)
3	12	46.7	843.3
4	13.3	51	1078
5	14.7	55	1318
6	16	58.7	1588
7	17.7	62	1841.8
8	18.7	65.3	2130
9	20	69.7	2385
10	20.5	73	2641.6
11	21	79.6	2903.7

Table VII shows the waveform switching delay of each standard in millisecond. Through the observation, the deployment delay increases with the increasing of the amount of components for all of the three standards.

TABLE VII. WAVEFORM SWITCHING DELAY COMPARISON IN THE COMPONENT NUMBER

Number of components	STRS (ms)	Lightweight SCA 4.1 (ms)	Full SCA 4.1 (ms)
3	13	51.6	880.6
4	14.6	56	1124
5	15.7	60	1376.3
6	17.7	64.4	1661.6
7	19.4	67.7	1930.4
8	20.4	71.3	2234
9	21.6	76.7	2502
10	22.2	79	2769.6
11	23	85.9	3035.1

B. *Analysis*

According to the experiment results, lightweight SCA 4.1 and STRS have very close transfer efficiency. Their transfer delay is almost 1/10 of full SCA 4.1.

As shown in Fig. 12, each component in full SCA 4.1 creates a new process and keeps "alive" by circling the accepting code inside. Components accept and send packet once available from middleware, as shown in Fig. 11. While Components in STRS and lightweight SCA 4.1 fetch and push packets by transfer mechanism. The creation and communication between processes devote to the lower efficiency in full SCA 4.1.

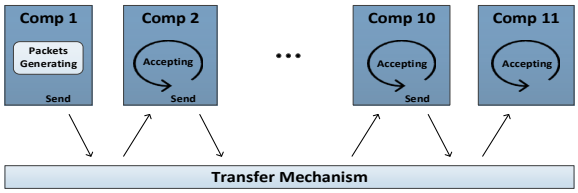

Fig. 12. Circling mission of SCA component

Another difference for full SCA 4.1 is the adoption of object request broker (ORB) mode, which brings more manipulations such as serialization and deserialization, which involuntary cause the decrease of efficiency.

The standards of STRS and SCA both have their advantages and disadvantages. The merits of STRS are its high efficiency and the fact that it can operate on the lightweight platform which is resource-limited. In the earlier versions of VxWorks, e.g., 5.4 and 5.5, that use the sharing address space for all tasks, the efficiency of the inter-task communication is quite high. STRS also has disadvantages in term of portability and interoperability. The portability and interoperability of STRS is lower than other SDR architectures, making it less attractive in implementing ground-based communication systems. Meanwhile, the implementation of STRS does not provide the agency mechanism, placing more difficulty for the application developer. The merits of the SCA are increasing flexibility by adopting self-defined lightweight middleware and reducing the expenditure with thread communication. The drawbacks of the SCA are high waveform deployment delay, high waveform switching delay and large static memory occupation.

V. CONCLUSIONS

This paper evaluate the performance of three SDR implementation, i.e., STRS, lightweight SCA and full SCA 4.1 with four typical metrics. The experiment results demonstrate that the lightweight SCA 4.1 and STRS have similar performance on inter-components transfer efficiency. The experiment results also show that the efficiency of full SCA 4.1 CORBA is much lower than the other two SDR architectures in all aspects. According to the experiment results, it is worth considering adopting lightweight SCA 4.1 in the space platform owing to its high efficiency. However, when use this architecture, one should pay special attention to the waveform deployment, switching delay and static memory occupation. Possible future work include carrying out experiments in platform with limited computing capability and memory size. Besides, the experiments can also be extended to evaluate the power consumption of different architectures, which is also a challenge in space environment.

ACKNOWLEDGEMENT

This research was supported in part by the National Natural Science Foundation of China (Grant No. 61601482).

REFERENCES

[1] Adrat M, Ascheid G. Special Issue on Recent Innovations in Wireless Software-Defined Radio Systems[J]. signal processing systems, 2015, 78(3): 239-241.

[2] Cheng Y, Xu K, Hu Y F, et al. Technology demonstrator of a novel software defined radio-based aeronautical communications system[J]. Iet Science Measurement & Technology, 2015, 8(6): 370-379.

[3] Baranda J, Henarejos P, Gavrincea C G. An SDR implementation of a visible light communication system based on the IEEE 802.15.7 standard[C] International Conference on Telecommunications. IEEE, 2013:1-5.

[4] "Software Communications Architecture (SCA) Specification," Joint Tactical Networking Center (JTNC), Version 4.1, August 2015.

[5] Grahamcumming J. The GNU Make Book[J]. 2015.

[6] Gomez I, Marojevic V, Bracke J, et al. Performance and overhead analysis of the ALOE middleware for SDR[J]. 2010, 49(1):1134-1139.

[7] Reinhart R C, Johnson S, Kacpura T J, et al. Open Architecture Standard for NASA's Software-Defined Space Telecommunications Radio Systems[J]. Proceedings the IEEE, 2007, 95(10): 1986-1993.

[8] Adrat M, Bernier S P, Buchin B, et al. A Technical Review of SCA Based Software Defined Radios: Vision, Reality and Current Status[J]. Journal of Signal Processing Systems, 2016: 1-11.

[9] Reinhart R C, Johnson S K. NASA's SDR Standard: Space Telecommunications Radio System[J]. 2007.

[10] Putthapipat P. Lightweight Middleware for Software Defined Radio (SDR) Inter-Components Communication[J]. 2013.

[11] "Software Communications Architecture (SCA) Specification," JTRS Standards, Joint Program Executive Office (JPEO) Joint Tactical Radio System (JTRS), Version 2.2.2, May 2006.

[12] "Space Telecommunications Radio System (STRS) Architecture Standard, NASA-STD-4009", NASA Technical Standard, June 2014.

[13] Putthapipat P, Andrian J, Liu C, et al. Studies on inter-component communication latency based on variation number of components and packet size in SDR-SCA waveform application[J]. International Journal of Computational Science and Engineering, 2016, 12(1): 65-72.