Applied Mathematics & Information Sciences
An International Journal

http://dx.doi.org/10.12785/amis/092L31

A Load Balancing Scheme by Chord Algorithm for DDSB Service

Gunjae Yoon[1], *Hoon Choi*[1,*], *Soohyung Lee*[2] *and Wontae Kim*[2]

[1] Department of Computer Science and Engineering, Chungnam National University, 99 Daehak-ro, Yuseong-Gu, Daejeon, 305-764, Republic of Korea
[2] CPS Research Team, Embedded S/W Research Department, Electronics and Telecommunications Research Institute, 218 Gajung-ro, Yuseong-Gu, Daejeon, 305-700, Republic of Korea

Received: 12 Jun. 2014, Revised: 14 Aug. 2014, Accepted: 17 Aug. 2014
Published online: 1 Apr. 2015

Abstract: A DDS Bridge (DDSB) allows embedded devices with limited computing resources to participate in DDS communication using a non-DDS protocol. The DDSB incurs heavy overhead, because it connects between DDSB nodes and non-DDS nodes in the full-mesh topology. In this paper, we propose to apply the addressing algorithm of Chord for reducing connection overhead of DDSB. The proposed C-DDSB improves the performance of the DDSB, because it avoids the full-mesh topology through distributed assignment of non-DDS nodes.

Keywords: Cyber Physical System, Data Distribution Service, Data Distribution Service Bridge, Chord

1 Introduction

Cyber-Physical System (CPS) is a system of collaborating computational elements controlling physical entities. Computational elements collect data from physical entities by a communication protocol and then process the data to control the physical entities[1]. Traditional examples of CPS include factory automation and networked combat systems. Automated agriculture, Intelligent Transportation System (ITS), and many IT convergent industries have recently attained major advances by adopting the CPS concept.

A CPS requires communication middleware to support frequent, real-time transmission of data/control information between physical entities and cyber entities. The Data Distribution Service (DDS) middleware of Object Management Group (OMG) is suitable for this requirement [2].

DDS is the Application Programming Interface (API) of communication middleware that creates network domains dynamically and serves a 1:1, 1:N or N:N real-time data distribution. DDS provides publish/subscribe communication capability with various Quality of Service (QoS) levels [3]. DDS nodes may freely join or leave the network domain. Thus, if DDS is used for the CPS, physical devices in a CPS system form a network domain to exchange collected data and control command even if they change their location dynamically. DDS is composed of two layers of protocols, Data Centric Publish Subscribe (DCPS) and Real-Time Publish Subscribe (RTPS). DCPS provides a communication interface to DDS applications and performs QoS control [4]. RTPS dynamically discovers the information of DomainParticipants, DomainEntities, and Topics in addition to performing reliable or best effort communication [5].

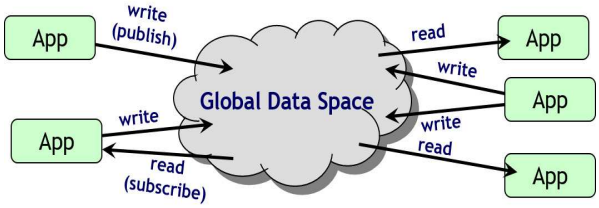

Fig. 1: DDS communication concept

Although the DDS has many advantages, it consumes a large amount of memory and processor load because of

* Corresponding author e-mail: hc@cnu.ac.kr

DCPS and RTPS protocols as well as all the information for constructing a domain network and for supporting diverse levels of QoS. As such, it is difficult to port the DDS on a small device that has limited resources. Physical entities of CPS are composed of desktop-level computing devices as well as small sensor nodes [6]. Not all of the devices have enough computing resources to run DDS. Therefore, in order to connect physical devices without a DDS to devices with a DDS, a gateway system between them acting as a DDS proxy is required.

A DDS Bridge (DDSB) [7] is a gateway mechanism for connecting small devices with limited resources to the DDS domain. In [7], the authors suggested a protocol and management technique for connecting small devices to a DDSB node. However, the management of possible, replicated connections was not defined. Consequently, if there are multiple DDSB nodes around a non-DDS device, all the DDSB nodes may try to connect with the same non-DDS device, resulting in replicated DDS proxy nodes. It is a waste of computing resources of DDSB nodes, and the replicated connections also cause an excessive amount of network traffic in the domain. This problem should be resolved.

The Chord [8] defines how to maintain/manage address space by allocating data into a specific node by using the node name and the data name in the network. The Chord evenly allocates/distributes data to nodes. This study has applied the Chord algorithm to allocate a non-DDS device to a single DDSB node. The connections between non-DDSB devices and DDSBs are efficiently managed, and replicated network traffic is avoided. To the knowledge of these authors, a similar approach has not been reported.

2 Related work

2.1 DDSB

A DDSB connects non-DDS nodes with DDS nodes and creates DDS communication entities. In order to perform these functions, a DDSB is composed of an Abstract Layer, Entity Manager, and Protocol Manager. The Abstract Layer communicates with nodes that do not use the DDS and consists of the Sender and Receiver. The Sender is used to transmit data to non-DDS nodes, and the Receiver receives data on a non-DDS protocol, such as the TCP/IP. The Protocol Manager relays the received data between DDS and non-DDS nodes. The manager creates DDS entities that will be connected with non-DDS nodes, and it maintains connection information between DDS entities and non-DDS nodes.

2.2 Chord

The Chord was proposed for peer-to-peer applications in a decentralized and unstructured network. It specifies how

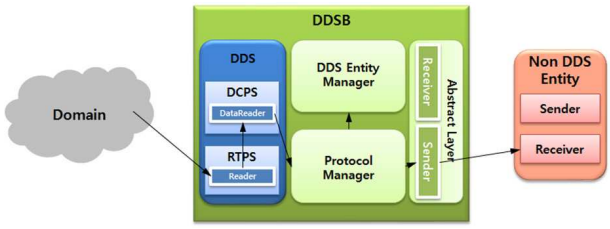

Fig. 2: DDSB communication concept

new nodes join the system and how to recover from a failure of existing nodes. It also defines how to associate the key, i.e., data with a node responsible for them. The Chord assigns each node and key an m-bit identifier obtained using the consistent hashing [9] function. It structures a circle address space [10]. The key k is assigned to the first node whose identifier is equal to or follows (the identifier of) k in the identifier space. This node is called the successor node of key k. Each node maintains additional routing information in a finger table. This additional routing information is concerning the minimum locations to look up correct successors, not all nodes. Each node maintains a routing table with at most m entries at the m-bits address space.

Table 1: Attributes in the finger table at the Chord

Notation	Definition
finger[k]	$(n+2^{k-1}) mod 2^m, 1 <= k <= m$
Successor	finger[1] node
Predecessor	the previous node on the identifier circle

The finger[k] in the finger table is the information used to look up the key k. Each node maintains successor information and predecessor information for structuring the address space. The successor and predecessor are modified when the node joins or leaves the address space.

Figure 3 shows an address space composed of 6 bits. The address space has 10 nodes and stores 5 keys of data. The successor of identifier 10 is node 14. Hence, key 10 would be located at node 14. Similarly, keys 24 and 30 would be located at node 32, key 38 at node 38, and key 54 at node 56.

Figure 4 shows the finger table entries for node 1. The first finger of node 1 points to node 8, as node 8 is the first node that succeeds $(1+2^0) mod 2^6 = 2$. Similarly, the last finger of node 1 points to node 38, as node 38 is the first node that succeeds $(1+2^5) mod 2^6 = 38$. These fingers are used to find the location of data. As an example, consider the address space in Figure 3, and suppose node 1 wants to find the successor of key 54. Since the largest finger of node 1 that precedes 54 is node 38, node 1 will ask node 38 to resolve the query. In turn, node 38 will determine the largest finger in its finger table that precedes 54, i.e., node

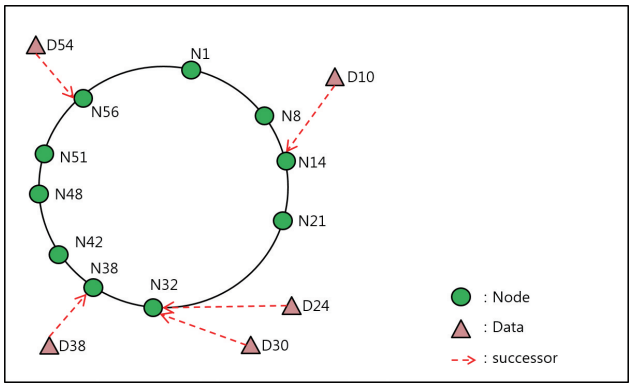

Fig. 3: An address space

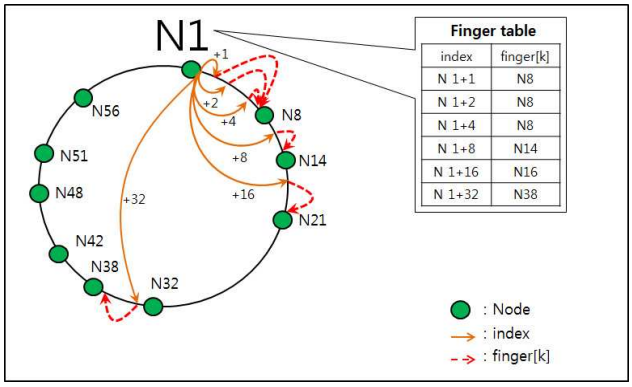

Fig. 4: The finger table entries for node 1

56. Finally, node 56 will find that its own successor, node 56, succeeds key 54, and thus will return node 56 to node 1.

3 DDSB assignment by the Chord algorithm

3.1 Data Assignment

The Chord creates a key using the hash from the data name and the node name. The Chord assigns data to the successor node that has the smallest key in the node key for values greater than or equal to the data key value.

We propose using a method of mapping the node of the Chord to a DDSB node, the data of the Chord to a non-DDS node, and the node/data name of the Chord to a Global Unique Identifier (GUID) of the DDS. The GUID can be applied to the data/node name of the Chord, because GUID is used for identifying objects that are participating in a domain in the DDS. Following the approach of assigning an address space at the Chord, the C-DDSB distributes locations of DDSB nodes and non-DDS nodes, and the C-DDSB creates agency objects in the DDSB nodes. The previous DDSB creates agency objects of non-DDS nodes for all DDSB nodes. However, C-DDSB distributes and assigns DDSB nodes and non-DDS nodes. This approach can reduce resources and distribute the load, because it creates 1:1 connections.

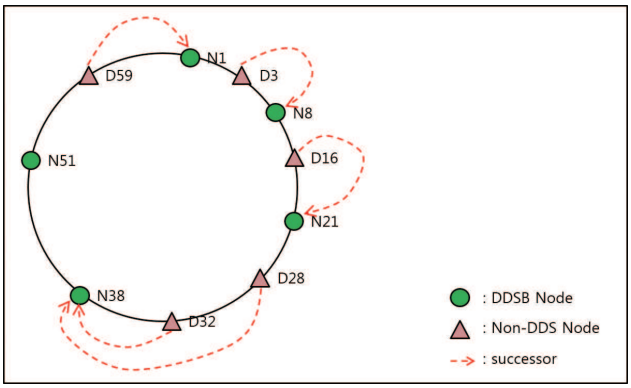

Fig. 5: The Connections of a successor

Figure 5 shows the connection between DDSB nodes and non-DDS nodes of the C-DDSB. In the network, there are DDSB nodes (N) for which the key is 1, 8, 21, 38, and 51 and non-DDS nodes (D) for which the key is 3, 16, 28, and 32. Following the assignment of data at the Chord, the successor of identifier 3 is N8. Thus, non-DDS node D3 is connected with N8 and creates a DDS agency object, and D3 is not related with N1, N21, N38, and N51. Similarly, non-DDS nodes D28 and D32 would be located at DDSB node N38, D16 at N28, and D59 at N1.

3.2 Stabilization

The Chord manages the key of data/nodes using a successor and a predecessor in the finger table, known as stabilization. The successor and the predecessor are used similarly with the front/rear pointer at a doubly linked list. When a new node joins an address space, the data which has the new node as the successor is transferred to the new node. The successor of the predecessor at the new node and the predecessor of the existed successor node are then changed to the new node. This is similar to the insertion process of a doubly linked list. Meanwhile, the leaving process is similar to the deletion of a doubly linked list. The joining of new nodes and the leaving of existing nodes are announced to existing nodes in the address space. Disconnection due to failures of existing nodes is recovered by the periodic execution of the stabilization process.

Figure 6 shows the process of changing an address space due to joining a new node. In process (1), following

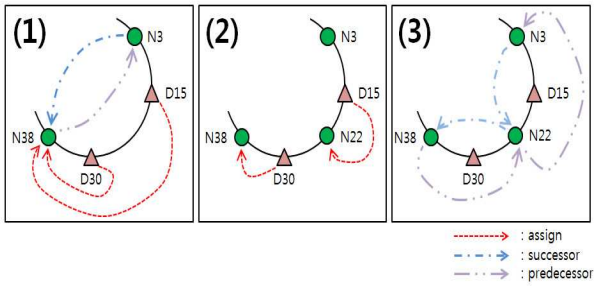

Fig. 6: Joining a node

the method of the address space definition at the Chord, the non-DDS nodes D15 and D30 are connected with DDSB node N38. The predecessor of N38 is set up as N3, and the successor of N3 is set up as N38.

Process (2) shows the changing of the non-DDS nodes when DDSB node N22 joins the address space. Because the successor of D15 is N22, the connection of D15 is changed from N38 to N22, which is newly joined. Process (3) shows the changing of the predecessor and the successor in the finger table when DDSB node N22 joins the address space. The predecessor of N22 is set up as N3, which is the successor of N38, and the predecessor of N38 is changed to N22 from N3. The successor of N3 is then changed to N22.

3.3 Finger Table

The Chord maintains a finger to find nodes that contain data when accessing data corresponding to a particular key, and it retains the predecessor and successor to maintain the address space. However, the DDS does not have to perform additional data position detection processing of data access because it maintains separate location information and the topic of nodes. The proposed strategy uses a DDS to access data after defining an address space. Because of this, it is possible to reduce the required resources to save the finger by maintaining successor and predecessor information for stabilization and definition of the address space.

Table 2: Attributes in the finger table at C-DDSB

Notation	Definition
Successor	finger[1] node
Predecessor	the previous node on the identifier circle

4 Communication protocol

4.1 pre-processing

A C-DDSB node basically works as a DDS node. Therefore, a C-DDSB node needs to join a DDS domain before making a connection with non-DDS nodes.

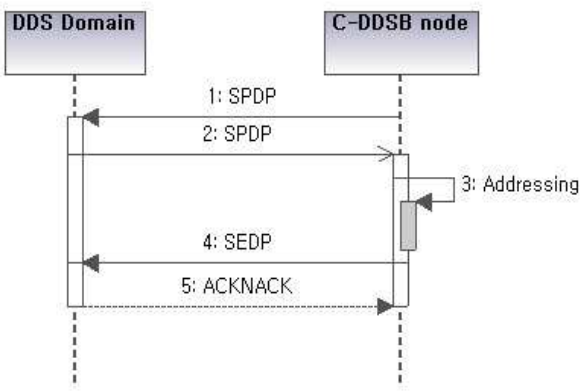

Fig. 7: Pre-processing sequence diagram

Figure 7 shows pre-processing sequences at a C-DDSB. At first, Simple Participant Discovery Protocol (SPDP) messages are exchanged through a multicast transmission. The SPDP specifies how the DomainParticipants discover each other in the DDS domain network. Once two DomainParticipants have discovered each other, they exchange information about the Endpoints they contain using a Simple Endpoint Discovery Protocol (SEDP) message. The endpoints represent the objects executing data writing/reading in the DDS.

For achieving the addressing rule of the Chord, the addressing process (step 3 in Figure 7) is inserted between the SPDP and the SEDP. In this process, the C-DDSB node configures the address space and the finger table using the information of discovered remote C-DDSB DomainParticipants. When new C-DDSB DomainParticipants join the DDS domain network, they send SPDP messages and the existing C-DDSB nodes reconfigure the address space and the finger table. Through these configuration and reconfiguration processes, DDSB nodes maintain the necessary routing information.

4.2 C-DDSB connection

A previous study on DDSB specified protocol sequences and message frames [7]. We specify additional sequences

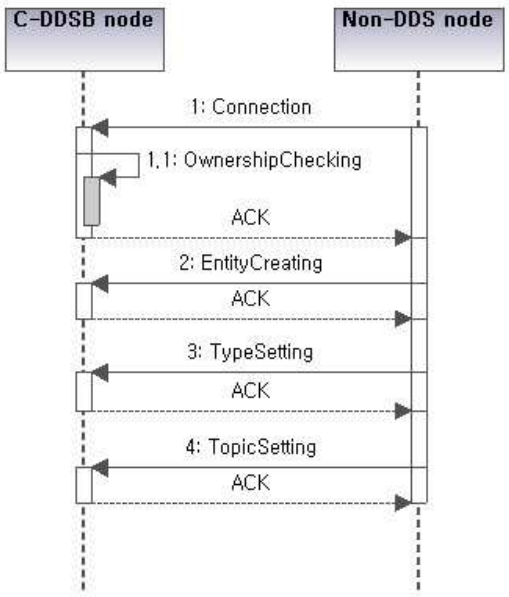

Fig. 8: DDSB connection sequence diagram

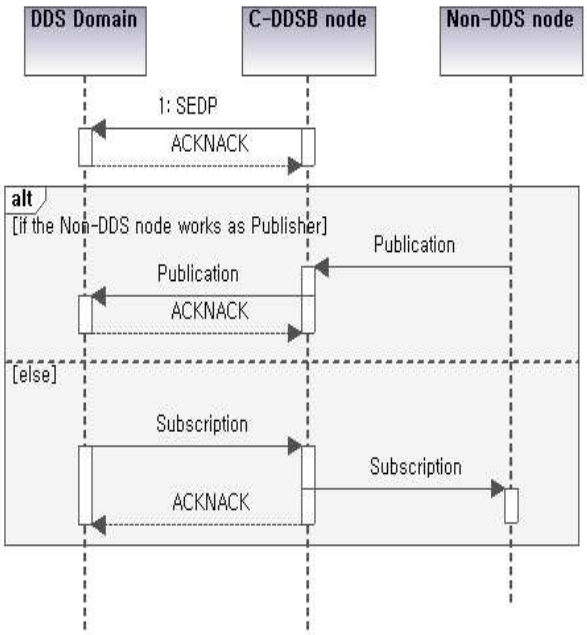

Fig. 9: Data communication sequence diagram

in order to connect C-DDSB nodes with non-DDS nodes. When a non-DDS node joins network, it sends a connection message through a multicast transmission. The receiving node checks the ownership of the non-DDS node using the successor and predecessor information. Just one C-DDSB node sends an ACK message to this non-DDS node through a unicast transmission. After this connecting sequence, the non-DDS node requests creation of a DDS entity to the C-DDSB node. For the setting of the created DDS entity, the non-DDS node sends its Topic information. In the DDS, the Topic is composed of a topic name and a data structure type. Therefore, the non-DDS node sends the information type and the topic name. All these connections and creating and setting sequences are executed through reliable communication.

4.3 Data communication

After the C-DDSB connection stage, the C-DDSB node creates an agent DDS entity and announces the creation of this DDS entity to the DDS domain using the SEDP. When this SEDP process is finished, the corresponding non-DDS node works as one of two types of the DDS entity, either a publisher or a subscriber.

The non-DDS node sends or receives data to or from a C-DDSB node using only the best-effort approach. Then, the related C-DDSB node relays the topic data between the DDS domain and the non-DDS node. When it communicates with the DDS domain, the topic data are sent either in a reliable or best-effort approach.

5 Test results

To measure the performance of C-DDSB, we compared the DDSB applications with the C-DDSB applications with respect to memory usage and the amount of messages for the connection setup. Additionally, we checked the amount of generated messages during the run-time.

Table 3: Memory usage with 1 DDSB node

Number of Non-DDS nodes	100	200	300	400	500
DDSB	2,166	3,676	5,186	6,696	8,206
C-DDSB	2,279	3,789	5,299	6,809	8,319

Table 4: traffic with 1 DDSB node

Number of Non-DDS nodes	100	200	300	400	500
DDSB	607	1,207	1,807	2,407	3,007
C-DDSB	607	1,207	1,807	2,407	3,007
Non-DDS(DDSB)	4	4	4	4	4
Non-DDS(C-DDSB)	4	4	4	4	4

Table 3 and Table 4 show the memory usage and the amount of message traffic with respect to the number of

non-DDS nodes when the network has one DDSB node. With results in Table 3, we checked that the memory usage of the C-DDSB application increases similarly with the DDSB application when the number of non-DDS nodes is increased. With results in Table 4, we confirmed that the amount of message traffic of the C-DDSB application is also similar to that of the DDSB application when the number of non-DDS nodes is increased. Through the results in Table 3 and Table 4, we verified that the performance of the C-DDSB application is similar with that of the DDSB application when the network has one DDSB node.

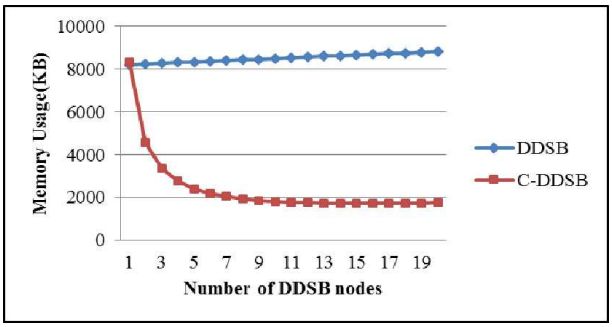

Fig. 10: Memory usage with 500 non-DDS nodes

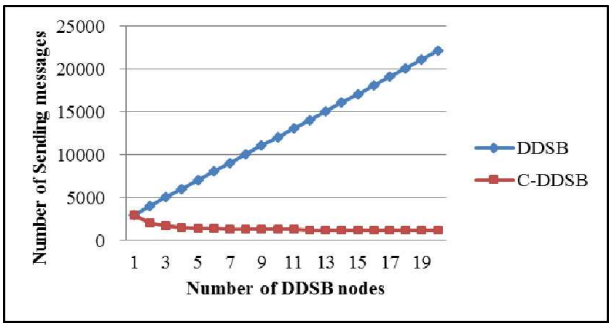

Fig. 11: Traffic with 500 non-DDS nodes

Figure 10, Figure 11 and Figure 12 show the memory usage and the amount of traffic due to the change of DDSB nodes when the network has 500 non-DDS nodes. Shown in Figure 10, we found that the memory usage of the C-DDSB application is much less than that of the DDSB application as the number of DDSB nodes is increased.

In Figure 11 and Figure 12, we see that the amount of change in traffic of the C-DDSB application is smaller than that for the DDSB application when the number of DDSB nodes is increased. Through the results shown in Figure 10, Figure 11 and Figure 12, we confirmed that the

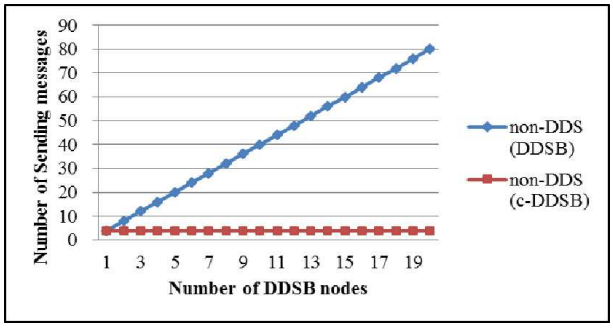

Fig. 12: Traffic changes of Non-DDS nodes

memory usage and the amount of message traffic of the C-DDSB application are superior to the case of the DDSB application when the network has numerous non-DDS nodes, and this gap becomes greater when the number of DDSB nodes increases.

In the previous DDSB study, the non-DDS nodes announce participation to the DDSB nodes in the network. All the DDSB nodes that receive these participating messages create DDS translation objects and connect with these non-DDS nodes. When the network has M DDSB nodes and N non-DDS nodes, each DDSB node makes connections to all N non-DDS nodes, and each non-DDS node is connected to all M DDSB nodes. Therefore, the network has M*N connections. At the C-DDSB, the DDSB nodes select a responsible DDSB node through the addressing algorithm of the Chord after the DDSB nodes recognize the non-DDS nodes. Then, only the responsible DDSB node makes a connection to a specific non-DDS node. As a result, the network has the same number of connections as non-DDS nodes even if the network has numerous DDSB nodes and non-DDS nodes. This leads to the results seen in Figure 11 and Figure 12.

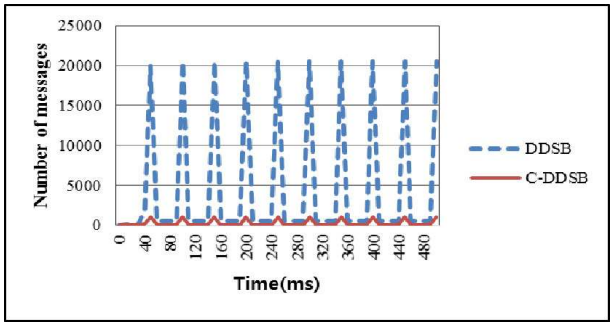

Fig. 13: Amount of generated messages during run-time

Figure 13 shows the number of generated messages when the network has 20 DDSB or C-DDSB nodes and 500 non-DDS nodes. In this experiment, every non-DDS

node works as a Publisher and generates messages at regular intervals. Shown in Figure 13, we found that the number of run-time generated messages of the C-DDSB application was smaller than the DDSB application during the entire test time, and this difference is even bigger at regular intervals, showing peak patterns. This feature is due to the heartbeating of DDS entities, rather than data publishing. When a network has N non-DDS nodes and M DDSB or C-DDSB nodes, each DDSB node has N agent DDS entities. However, if they are C-DDSB nodes, each node has N/M agent DDS entities, because they make 1-to-1 connections and distribute assignments of the agent DDS entity. Therefore, when all non-DDS nodes publish a message, DDSB generates N*M DDS topic data, whereas the C-DDSB generates only N DDS topic data into the domain network. The number of generated messages increases linearly. Each DDS entities must send the heartbeat message to all DDS nodes at regular intervals for announcing their liveliness. The domain network, created by the DDSB, generates N*M*(M-1) heartbeating messages, but the C-DDSB domain network generates (N/M)*M*(M-1) heartbeating messages. From the calculated results, the heartbeating messages of the DDSB are exponentially increased by the number of DDSB nodes. The C-DDSB solves this exponential traffic increase problem of heartbeating.

6 Conclusion

In this paper, we applied a strategy of generating an address space at the Chord to the DDSB. We resolved the problems of simultaneously connecting a small embedded device in a CPS System to more than one DDSB. As a result, network traffic is reduced, and the load balancing of the DDSB is possible. The C-DDSB achieved load balancing via the distribution of embedded devices and DDSB nodes, and the C-DDSB performs error recovery and a dynamic join/leave process through a stabilization strategy. With application of the Chord, we reduced the amount of information for maintaining an address space through reduction of a finger table. The application occurs to solve the exponential traffic problem in heartbeating.

The size of GUID is fixed at 96 bits, because the address space of the DDS is not flexible. With consideration of the characteristics of GUID, it is necessary to research prevention/avoidance strategies for collisions generated by the hash at the Chord. Hand-off problems also occur when the DDS entities are moved due to the join/leave process of connected embedded devices. We plan to study ways of reducing the hand-off problem.

Acknowledgement

This research was financially supported by the Ministry of Education (MOE) and National Research Foundation of Korea (NRF) through the Human Resource Training Project for Regional Innovation (No. 2013H1B8A2032180).

This work was partially supported by the Dual Use Technology Program through Civil Military Technology Cooperation Center funded by The Ministry of Trade, Industry & Energy and Defense Acquisition Program Administration.

References

[1] Edward A. Lee. ”Cyber physical systems: Design challenges.” Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on. IEEE, 2008.

[2] Gerardo Pardo-Castellote, Bert Farabaugh, and Rick Warren. ”An introduction to dds and data-centric communications.” Real-Time Innovations. OpenURL, 2005.

[3] Angelo Corsaro, and Douglas C. Schmidt. ”The Data Distribution ServiceThe Communication Middleware Fabric for Scalable and Extensible Systems-of-Systems.” System of Systems, InTech.(March 2, 2012). DOI 10 (2012): 30322.

[4] Jung-Dal Jeon, Hoon Choi, Chum-Su Kim, Design and implementation of DCPS protocol, The Korea Institute of Military Science and Technology, pp.758-761, June 2011.

[5] Gun-Jae Yoon, Hoon Choi, Chum-Su Kim, Design and implementation of the RTPS protocol, The Korea Institute of Military Science and Technology, pp.762-765, June 2011.

[6] Fang-Jing Wu, Yu-Fen Kao, and Yu-Chee Tseng. ”From wireless sensor networks towards cyber physical systems.” Pervasive and Mobile Computing 7.4, 2011, 397-413.

[7] Jung-Dal Jeon, et al. DDS Bridge for Embedded System. WORLDCOMP 12, July, 2012.

[8] Ion Stoica, et al. Chord: A scalable peer-to-peer lookup service for internet applications. ACM SIGCOMM Computer Communication Review. **31**, 2001.

[9] David Karger and et al., Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web, The 29th Annual ACM Symposium on the Theory of Computing, pp. 654-663. 1997.

[10] Yuta Shimano, and Fumiaki Sato. Dynamic Reconfiguration of Chord Ring Based on Physical Network and Finger Table Information. Network-Based Information Systems (NBiS), 2012 15th International Conference on IEEE, 2012.

Gunjae Yoon received his B.S. degree in Computer Engineering from Chungnam National University, Korea, in 2011. He joined the Mobile Distributed Computing Laboratory in Chungnam National University. He is currently working toward a Ph. D. degree in Computer Engineering at Chungnam National University. His research area includes the network protocol, distributed computing and the communication middleware.

Hoon Choi He is a professor of the Department of Computer Science and Engineering, the Chungnam National University (CNU), Korea. He received a MS and a PhD in computer science from Duke University in 1990 and 1993, respectively. His research area includes the system software for mobile, distributed computing and the communication middleware.

Soohyung Lee received BS, MS degrees from Hanyang University, Korea in 1991 and 1993 respectively, and PhD degree from Chungnam National University in 2012. In August 1993, he joined the network design laboratory of DACOM corporation. Since Oct. 2000, he has been a Principal Member of Engineering Staff in Cyber-Physical Systems (CPS) research team, Electronics and Telecommunications Research Institute (ETRI), Korea. His research interests include IT converging system,distributed communication, network security.

Wontae Kim received BS, MS and Ph.D from Hanyang University, Korea in 1994, 1996 and 2000 respectively. He established a ventrue company, Rostic Technologies, Inc. in 2001. He joined ETRI in 2005 and now he is the team manager of CPS(Cyber-Physical Systems) research team. He is a president of CPS project group of TTA(Telecommunication Technology Association) from 2011.