Research Article

International Journal of Advanced Computer Research, Vol 9(40)
ISSN (Print): 2249-7277 ISSN (Online): 2277-7970
http://dx.doi.org/10.19101/IJACR.COM16004

Supporting synchronous and asynchronous communications in event-based communication framework for client-server applications

Mingyu Lim*
Professor, Department of Smart ICT Convergence, Konkuk University, Korea

Received: 29-June-2018; Revised: 22-July-2018; Accepted: 9-November-2018

Abstract
This paper proposes a communication framework (CM) that supports both of synchronous and asynchronous communication between a client and a server. Original CM is an event-based asynchronous communication framework and provided applications with communication services only in the asynchronous manner. The extended CM provides synchronous communication mechanism using a blocking socket channel and a non-blocking socket channel. By explicitly providing blocking socket channels to applications, CM allows a client to organize its own synchronous communication protocol with a server. With the non-blocking socket channel, CM can change the original asynchronous communication services to synchronous services using the synchronization technique between the main thread and the processing thread. Because applications can use both the asynchronous and synchronous communications, they can apply communication services to broader contexts. For performance analysis, the proponent compared the asynchronous and synchronous methods with the qualitative analysis and the quantitative experiment. The qualitative analysis verifies that developers can design an application logic more intuitively with the synchronous communication. The quantitative experiment shows that the server-response delay of the synchronous communication is shorter than that of the asynchronous case.

Keywords
Synchronous/asynchronous communication, Event-based communication framework, Blocking/non-blocking socket channel, Client-server system.

1.Introduction
Developers of distributed applications can implement communication functionalities from the scratch, but it is more efficient for them to use services provided by communication middleware or framework. Although previous work [1, 2] of the proposed scheme, existing general communication middleware's [3−7], middleware's for online social networks [8−12], and other recent middleware's [13−15] provide generic or application-specific communication services, they do not support both the synchronous and asynchronous communications. Communication between a client and a server that existing approaches use is not determined by application requirements, but by their internal ways of communication channel management or kinds of services. Thus, if an application selects a middleware or a service, it also uses a fixed communication method between the client and the server.

For example, because previous versions of a communication framework (CM) on an event-based asynchronous communication method were developed, they support all the services only with the asynchronous communication. However, all kinds of services can be provided not only with the asynchronous, but also with the synchronous communication. If a middleware or framework allows an application to choose the communication method according to its context and requirement, then it could provide applications with more flexible services.

In this paper, an extended version of CM in order to support both the synchronous and asynchronous communications is proposed. CM provides an application with various communication services both in the synchronous and asynchronous manners. The communication methods of CM are developed based on the multi-threaded and event-based asynchronous communication architecture. To realize both the synchronous and asynchronous

*Author for correspondence

communications, CM internally manages blocking and non-blocking socket channels. With these two types of channels, CM supports the synchronous communication in two aspects. On one hand, by providing a blocking socket channel to an application as a service, CM allows a client and a server to organize their own synchronous communication protocol. On the other hand, CM synchronizes its main thread and processing thread so that it can change the original asynchronous communication services to the synchronous ones. An application can use CM services both with the synchronous and asynchronous communications. The two communication methods with a qualitative analysis and a quantitative experiment are compared. From the qualitative analysis, it is verified that application development with the synchronous services is more intuitive and efficient than that with the asynchronous services. In the quantitative experiment, it is confirmed that the synchronous services show shorter server-response delay than the asynchronous services.

2.Communication framework (CM)

CM is a communication framework that helps developers build distributed applications by providing various communication services. Using application programming interfaces (APIs) and configuration files of CM, application developers can easily implement various communication functionalities with which a CM node (aka. an application that uses CM services) can interact with other nodes.

Nodes that use the previous CM communicate with each other in the event-based asynchronous manner. Comparing to the synchronous communication, the asynchronous communication does not make a client wait for the reply to a request from a server. The client can perform other tasks until it receives the reply. Instead, the client should register a callback function so that it can receive the reply event anytime. Whenever CM receives an event, it calls the registered callback function, and the client can asynchronously deal with the reply event after it sends the request to the server.

To support the event-based asynchronous communication, CM is organized with four threads: main, processing, sending, and receiving threads. When an application starts, its main thread also starts CM. The CM main thread then starts the processing, sending, and receiving threads. The main thread has a role of handling a local service request from the application. The processing thread handles a CM event that is delivered by the receiving thread. The received CM event from a remote CM node is processed internally by CM itself, and if necessary, externally by the application. To process an event in the application, it should register an event handler to CM. The event handler contains an event-handling method and is executed by the processing thread. After the processing thread handles a received event first, it calls the event-handling method if the event needs to be handled by the application.

The main task of the sending thread is to send a byte message through the socket channel. When the main or processing thread needs to send a CM event to a remote node, it composes the event, converts it to a byte message, and puts the message to a sending queue through which the message is delivered to the sending thread. On the contrary, the receiving thread receives a byte message and puts it to a receiving queue through which the message is delivered to the processing thread.

3.Synchronous communication support

CM originally supports the event-based asynchronous communication between the server and client. Furthermore, CM also provides the synchronous communication according to application requirements or service features. The synchronous communication can be supported with the blocking and non-blocking socket channels. The server and client can use a separate blocking socket channel for the synchronous communication if they need to send a byte message synchronously without a CM event. If the client needs to call CM services synchronously, CM supports such a synchronous communication with the default non-blocking socket channel.

3.1Synchronous communication with blocking socket channel

Among CM services is a communication channel management. By adding a new blocking socket channel, the client can realize the synchronous communication with the server. For example, if a client needs to send raw sensor data to the server, it cannot use the event transmission service of CM because the raw sensor data do not follow the CM event format. The event transmission service can be used only if the raw data is wrapped in a CM event. However, this method becomes inefficient, if the amount and the number of raw data is quite large or if the data need to be sent frequently because the transmission and processing delay of such an event is increased. Alternatively, the client can use a separate

blocking socket channel to send the sensor data without any CM event. Firstly, the client and the server create a new blocking socket channel and prepare the data transmission. The preparation for the data transmission is done by exchanging some relevant CM events between the two nodes. For example, if a user requests to send data using the blocking socket channel, the client CM send the server CM an event that requests for the server to start receiving the data. If the preparation for the data transmission completes, the client starts the data transmission through the blocking socket channel, and then the server receives the data through its blocking channel. Details of the synchronous communication with the blocking socket channel are shown in *Figure 1*.

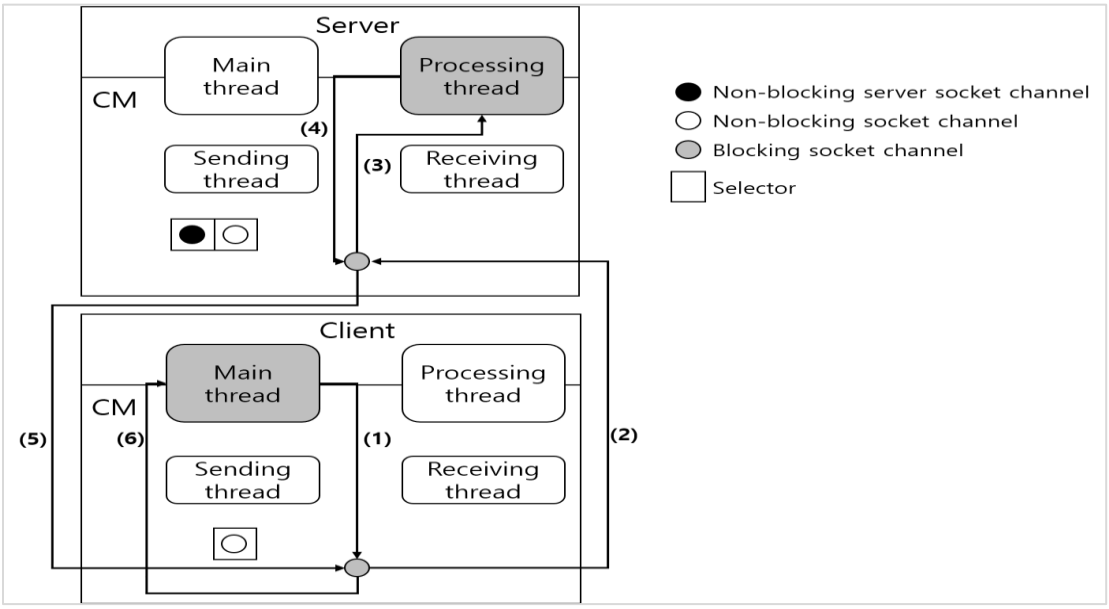

Figure 1 Synchronous communication via blocking socket channel

(1) The client main thread sends a byte message through the blocking socket channel. (2)−(3) The server then receives the data through its blocking channel. The server processing thread is suspended and cannot do other tasks until it receives the data. (4)−(5) When the server receives all data, it can send a response or analysis result message to the client. In this case, the client main thread waits for the reply message by calling the receiving function of the blocking channel. (6) When the client receives the server reply message, it closes the blocking channel and executes the next task. The server also closes the blocking channel and executes its next task after it sends the reply message. With the blocking socket channel, the server and client synchronously communicate by calling the receiving function that suspends their execution until they receive a message from the opponent. If the client or server should execute other tasks concurrently while it uses the blocking socket channel, it requires an additional dedicated thread that is in full charge of message transmission through the blocking channel.

3.2Synchronous communication with non-blocking socket channel

The second way of synchronous communication between the client and server is that CM enables communication services to be called synchronously. In the previous version of CM, the client asynchronously receives a response to its request because the main and the processing threads are separate. That is, after the client main thread sends a request event, the processing threads independently from the main thread receives a reply. However, there may be a situation where the client should receive a response from the server synchronously. For example, the client might have to request a sequence of CM services such as creating a socket channel and sending an event through the channel. To this end, the client can use the new socket channel only after it receives the reply event, ensuring that the server has added the client's socket channel information. In the existing asynchronous channel addition service, the client event handler should separately receive the reply event because it does not suspend the execution after it calls the CM API

function. If CM provides the channel addition service synchronously, the client can directly receive the server reply event as the return value of the synchronous API function. The client execution is suspended until the API function returns. Thus, the client can immediately use the channel when it receives the return value of the synchronous API function. Such a way of synchronous communication is similar to the remote procedure call (RPC). *Figure 2* shows the detailed procedure of synchronous communication between the client and server using the original non-blocking socket channel of CM.

When a user requests a CM communication service through the client, (1) the main thread of the client CM makes the request CM event, converts it to the byte message, and delivers the message to the sending thread through the sending queue. (2) The main thread acquires the lock of a synchronization object and becomes suspended by calling the wait function of the synchronization object so that it can wait for the reply event. (3)−(4) The sending thread sends the message to the server CM using the default non-blocking socket channel. (5) The receiving thread of the server CM receives the request message while it monitors the selector object. (6) The received message is delivered to the processing thread through the receiving queue. The processing thread converts the message to the corresponding CM event and processes the request event. (7) The processing thread makes a reply event, converts it to a byte message, and delivers the message to the sending thread through the sending queue. (8)−(9) The sending thread sends the reply message to the client through the non-blocking socket channel. (10) The receiving thread of the client CM receives the reply message while it monitors the selector object. (11) The receiving thread delivers the message to the processing thread through the receiving queue. The processing thread converts the message to the CM event and gets the request result from the converted event. (12) The processing thread makes the main thread wake up from the suspended state by calling the notify function of the synchronization object and delivers request result in the main thread. The main thread then resumes the execution and returns the called API function with the result value to the application. The client application gets the return value and can execute the next task.

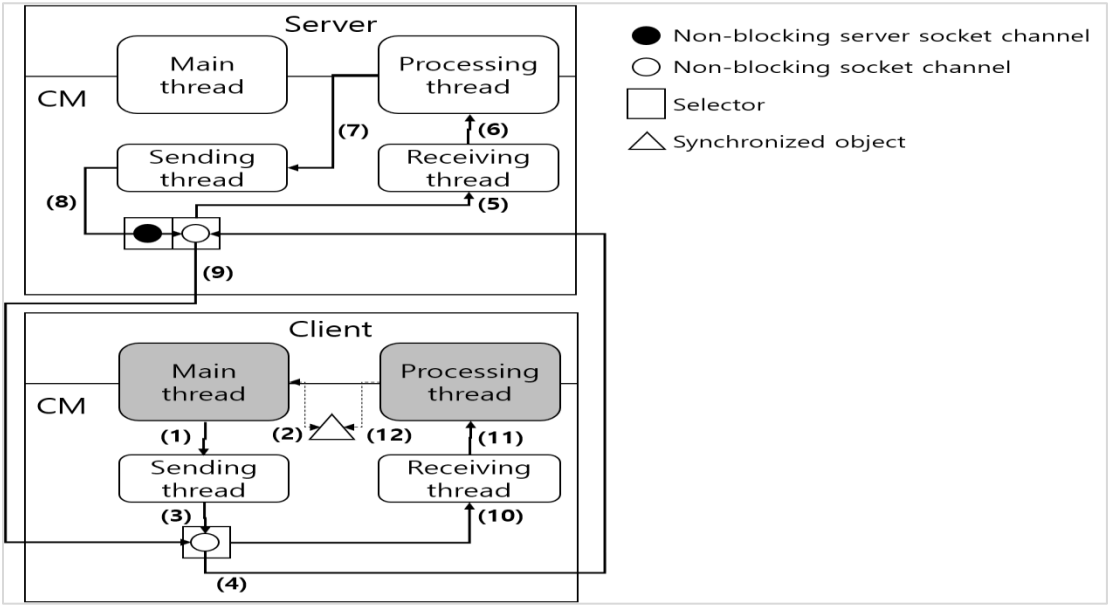

Figure 2 Synchronous communication via non-blocking socket channel

In the synchronous communication using the non-blocking socket channel, the main thread is suspended until it receives the result of its service request from being synchronized with the processing thread.

On the other hand, the server does not need to suspend any thread because it deals with the request in the same way of the asynchronous communication.

4.Performance analysis

In this section, the proponent presents a comparison about the performance of the synchronous and asynchronous communication mechanisms in terms of both qualitative and quantitative analyses. In the qualitative analysis, development efficiency of using CM APIs between the two mechanisms is compared. In the quantitative analysis, the API return delay and the server response delay between the mechanisms are measured.

4.1Qualitative performance analysis

Every communication service API of CM supports both synchronous and asynchronous requests of the client. *Table 1* describes the part of synchronous and asynchronous versions of CM services for comparison. loginCM() and syncLoginCM() functions request user authentication. requestSessionInfo() and syncRequestSessionInfo() functions request available session information that the server manages. joinSession() and syncJoinSession() functions request to join a session in the server. addBlockSocketChannel() and syncAddBlockSocketChannel() functions request to add a blocking socket channel to the server. removeBlockSocketChannel() and syncRemoveBlockSocketChannel() functions request to remove a blocking socket channel from the server.

Table 1 Synchronous/asynchronous API of client CM

Synchronous API	Asynchronous API
CMSessionEvent syncLoginCM(String strUserName, String strPassword)	boolean loginCM(String strUserName, String strPassword)
CMSessionEvent syncRequestSessionInfo()	boolean requestSessionInfo()
CMSessionEvent syncJoinSession(String sname)	boolean joinSession(String sname)
SocketChannel syncAddBlockSocketChannel(int nChKey, String strServer)	boolean addBlockSocketChannel(int nChKey, String strServer)
boolean syncRemoveBlockSocketChannel(int nChKey, String strServer)	boolean removeBlockSocketChannel(int nChKey, String strServer)

Each synchronous and asynchronous APIs have the same parameters, but different return types. The return value of all the asynchronous APIs is the boolean type. The boolean return type of the asynchronous API means, whether the client's request is successfully sent to the server or not. The true return value implies the successful transmission of the request. The API function returns false if the client CM meets an error during the process of the request. However, the asynchronous API functions cannot return the reply from the server. Instead, the client should asynchronously get the request result from the server using the event handler that it registers to CM.

Unlike the asynchronous API, the synchronous API directly returns the server reply to a service request. Return value types can also be different according to service types as follows. syncLoginCM(), syncRequestSessionInfo(), and syncJoinSession() functions return a server response event that contains a request result and additional information given by the server. The return value of syncAddBlockSocketChannel() function is the reference to a newly added blocking socket channel object. This function returns the socket channel only if the client CM successfully has created a new socket channel and if the servers CM also successfully have added the corresponding socket channel information. This function returns a null value if the request fails. The client can use this return type of synchronous API to directly get the service object after the client CM receives a server reply event. syncRemoveBlockSocketChannel() function has the boolean return type like the synchronous API, but return values have different meanings. If the return value is true, it implies that both the client and server successfully have removed the mutual blocking socket channels. The false value means that either the client or server causes an error during the channel removal process. This kind of return type of the synchronous API can be used when the client needs to confirm the completion of its request.

Using the synchronous API rather than the asynchronous one, a developer can build an application more intuitively and efficiently. Especially, if the application requires to request a sequence of relevant communication services, it can clearly benefit from the series of synchronous requests. For example, a client should send a request for current available session information to a server so that it can participate in the CM network. To request session information, the client firstly should log into the server. To this end, the client can simply

call the synchronous APIs such as syncLoginCM(), syncRequestSessionInfo(), and syncJoinSession() functions sequentially, because the client can directly get the server response to the previous request if the client gets an error return value of a previous request, it can stop participating in the CM network. *Figure 3* depicts a flow chart of the three synchronous API calls.

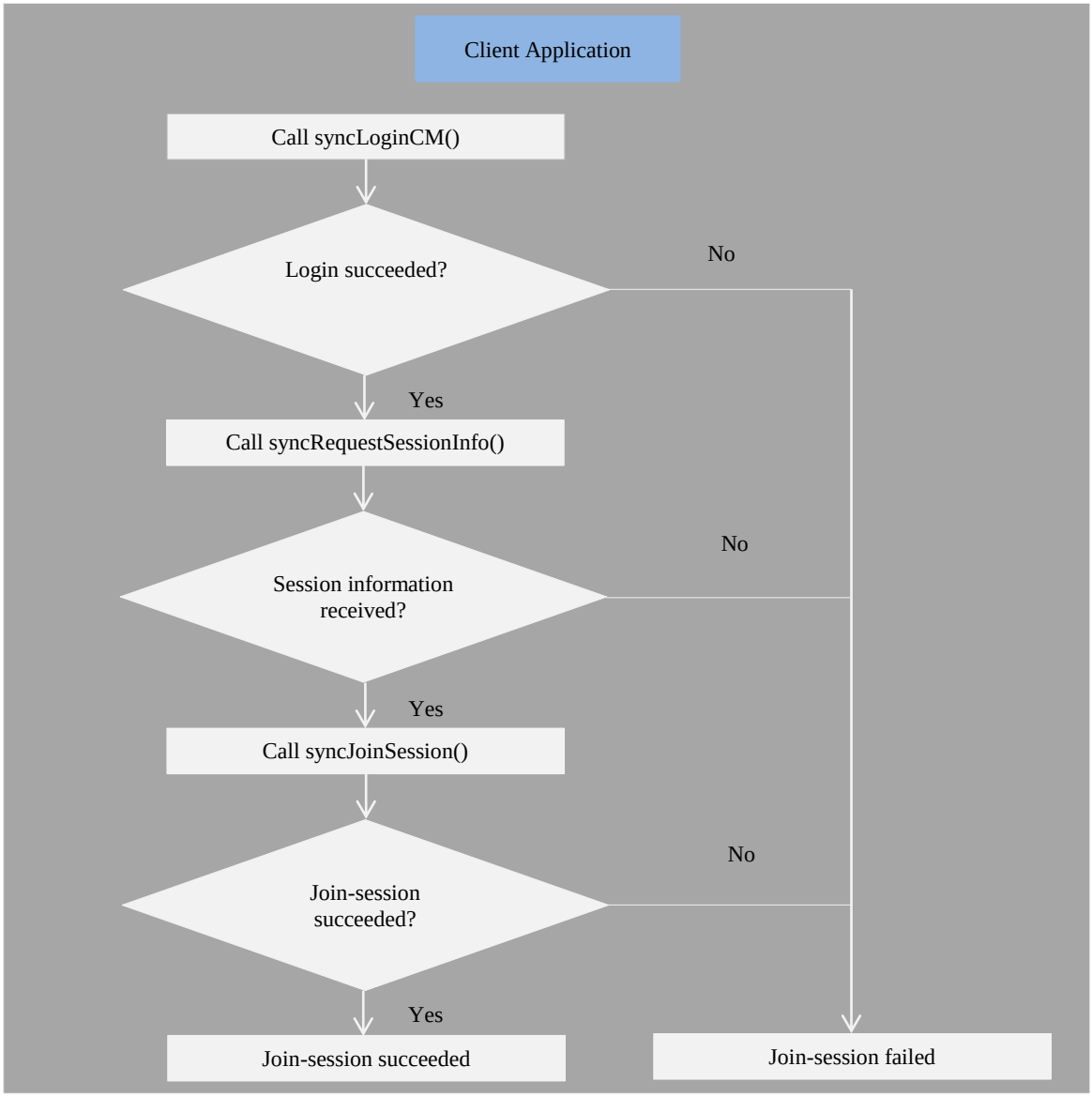

Figure 3 Flowchart of sequential calls of synchronous APIs

Alternatively, the client also can follow the join-session procedure using the asynchronous APIs. However, the asynchronous approach makes it inefficient because the service request and the reception of the server reply are separate from the client application and the client event handler. After the client sends the login request by calling loginCM() function, it should explicitly mark that the next task is to request available session information when the client receives the reply from the server. For such a mark, the client can assign a flag variable that is set to a specific value and that is shared with the event handler. The client event handler then needs to catch the response event to the login request among all the received CM events. When the event handler receives the login reply event, it checks whether the flag is set or not. If the flag variable is set, then the event handler requests the next service

by calling requestSessionInfo() function. The event handler also sets another flag variable to mark that the next request after it receives the session information is to join a session. If the event handler receives the available session information and the relevant flag is set, it requests to join a selected session by calling joinSession(). *Figure 4* depicts the aforementioned procedure of three sequential requests using the asynchronous APIs.

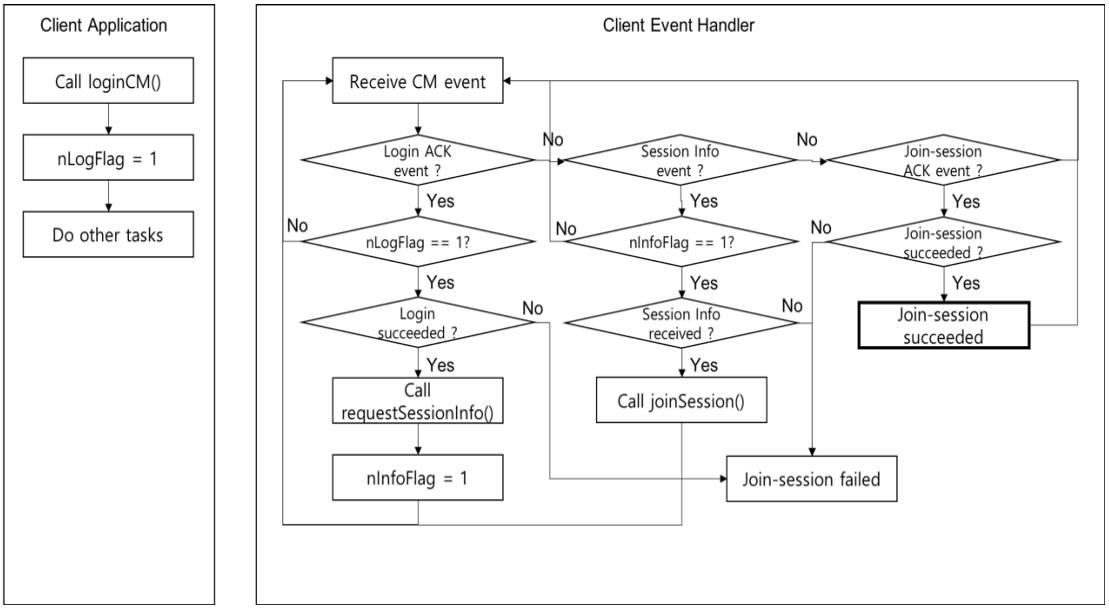

Figure 4 Flowchart of sequential calls of asynchronous APIs

4.2 Quantitative performance evaluation

Using the CM synchronous services, the client can send a request and receive its reply by calling the synchronous function and by getting the return value from the function. This gives more intuitive and efficient development method than the asynchronous services do. However, a problem of the synchronous API is that the client becomes suspended until it receives a reply from the server. In this section, the response delay of the synchronous and asynchronous communication when the client requests various CM communication services is compared. The server response delay is defined as an elapsed time from the moment when the client calls an API function to the moment when it receives the response event from the server.

For the experiment, the proponent developed a sample CM client and a CM server application using Eclipse integrated development environment (IDE) and Java. A computer (Windows 10, i3 central processing unit (CPU), 8 gigabytes (GB) memory) that executes the server is connected to a wired local area network (LAN) with 1 gigabit-per-second (Gbps) bandwidth, and a computer (MacOS, i5 CPU, 16GB memory) that executes the client is connected to the server through a wireless LAN. The average server response times of the synchronous and asynchronous APIs for the login, request-session-information, join-session, add-blocking-socket-channel, and the remove-blocking-socket-channel services are then measured.

Figure 5 shows the measurement result of the server response time of the synchronous and asynchronous CM APIs. Unlike the other functions, the add-blocking-socket-channel function has a higher response time as shown in *Figure 6*, because this function requires a relatively long processing task in the server CM that should create and add a new communication channel. Comparing the synchronous and asynchronous functions, it is verified that the server response delay of the synchronous functions is shorter than that of the asynchronous ones. In the synchronous case, the processing thread of the client CM receives the reply event and delivers it to the synchronized main thread. The client main thread then delivers the return value of the function from the client CM to the client application. In the asynchronous case, after the processing thread receives the reply event, it delivers the event to the client event handler by calling the registered callback function. That is, given the assumption that event transmission delay is the same between the

synchronous and asynchronous APIs, the function return task of the main thread after it wakes up from the suspended state costs less than the callback function task of the processing thread.

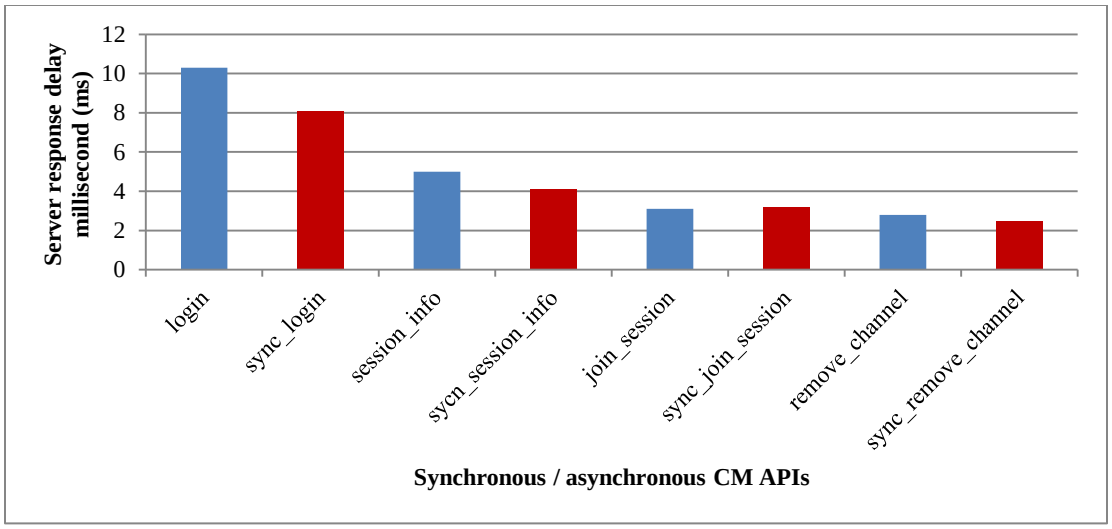

Figure 5 Server-response delay of synchronous and asynchronous APIs

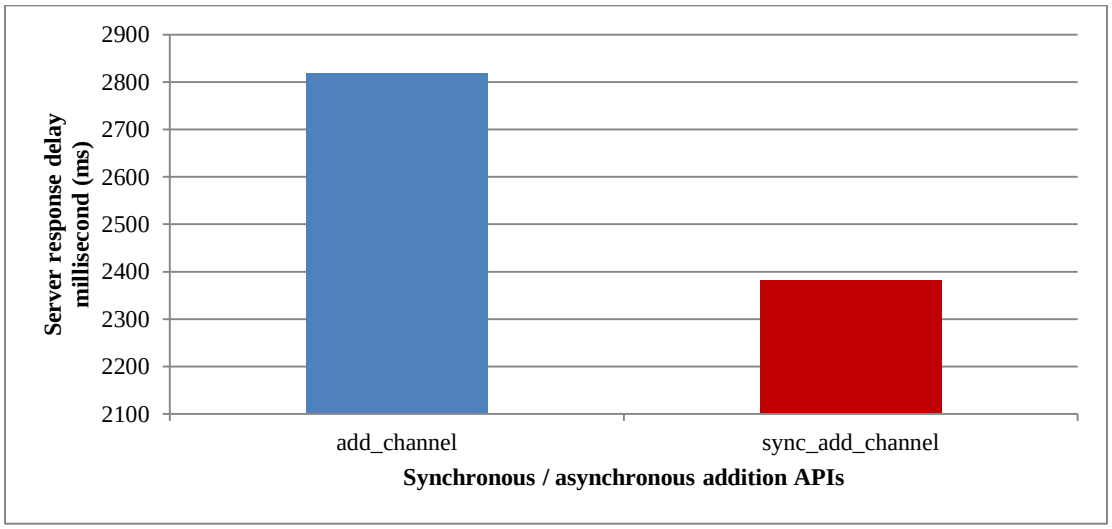

Figure 6 Server-response delay of channel addition APIs

In summary, the qualitative and quantitative comparison reveals that the synchronous communication gives more useful and intuitive development APIs with less server response delay than the asynchronous communication.

5.Conclusion

In this paper, the asynchronous communication mechanism for an event-based asynchronous framework was proposed. CM provides an application with more flexible communication services as it supports every communication service between a client and a server synchronously as well as asynchronously. To support the synchronous communication, CM provides synchronous communication method using a blocking and non-blocking socket channel. With synchronous and asynchronous communication services of CM, a client can select not only a communication service, but also a communication way about how it interacts with a server according to current execution contexts or communication requirements. To compare the feature of the synchronous and asynchronous communications, the proponent conducted the qualitative performance analysis as well as the quantitative experiments. From the performance

analysis, it is verified that the synchronous service makes the application development more intuitive and efficient, and that it also causes less server response delay than the asynchronous service.

For future research, the proponent plans to apply the synchronous communication support to the event transmission service of the CM. Although CM already provides various one-to-one and one-to-many event transmission services, they are only the asynchronous service. If such transmission services are provided synchronously as well, a client can send any event and wait until it receives the reply event. In the synchronous one-to-many transmission, it is a challenging issue to decide how many reply events the client should wait as the number of nodes increases.

Acknowledgment

This research was supported by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education (NRF-2018R1D1A1A09082580).

Conflicts of interest

The author has no conflicts of interest to declare.

References

[1] Lim M, Kevelham B, Nijdam N, Magnenat-Thalmann N. Rapid development of distributed applications using high-level communication support. Journal of Network and Computer Applications. 2011; 34(1):172-82.

[2] Lim M. CMSNS: a communication middleware for social networking and networked multimedia systems. Multimedia Tools and Applications. 2017; 76(17):18119-35.

[3] Tortonesi M, Stefanelli C, Suri N, Arguedas M, Breedy MR. MOCKETS: a novel message-oriented communications middleware for the wireless internet. In WINSYS 2006 (pp. 258-67).

[4] Morgan G, Lu F, Storey K. Interest management middleware for networked games. In symposium on interactive 3D graphics and games 2005 (pp. 57-64). ACM.

[5] Pakkala D, Pakkonen P, Sihvonen M. A generic communication middleware architecture for distributed application and service messaging. In joint international conference on autonomic and autonomous systems and networking and services 2005 (p. 22). IEEE.

[6] Henning M. A new approach to object-oriented middleware. IEEE Internet Computing. 2004; 8(1):66-75.

[7] Carvalho M, Suri N, Arguedas M. A mobile agent-based communications middleware for data streaming in the battlefield. In MILCOM 2005 (pp. 794-800). IEEE.

[8] Brooker D, Carey T, Warren I. Middleware for social networking on mobile devices. In Australian software engineering conference 2010 (pp. 202-11). IEEE.

[9] Mokhtar SB, McNamara L, Capra L. A middleware service for pervasive social networking. In proceedings of the international workshop on middleware for pervasive mobile and embedded computing 2009 (pp. 1-6). ACM.

[10] Pietiläinen AK, Oliver E, LeBrun J, Varghese G, Diot C. MobiClique: middleware for mobile social networking. In proceedings of the workshop on online social networks 2009 (pp. 49-54). ACM.

[11] Gupta A, Kalra A, Boston D, Borcea C. MobiSoC: a middleware for mobile social computing applications. Mobile Networks and Applications. 2009; 14(1):35-52.

[12] Borcea C, Gupta A, Kalra A, Jones Q, Iftode L. The MobiSoC middleware for mobile social computing: challenges, design, and early experiences. In proceedings of the international conference on mobile wireless middleware, operating systems, and applications 2008.

[13] Garcia-Valls M, Bellavista P, Gokhale A. Reliable software technologies and communication middleware: a perspective and evolution directions for cyber-physical systems, mobility, and cloud computing. Future Generation Computer Systems. 2017; 71:171-6.

[14] Majumdar D, Zhang L, Bhaduri P, Chakraborty S. Reconfigurable communication middleware for flex ray-based distributed embedded systems. In international conference on embedded and real-time computing systems and applications 2015 (pp. 159-66). IEEE.

[15] Luo X, Wu W, Bosilca G, Patinyasakdikul T, Wang L, Dongarra J. ADAPT: an event-based adaptive collective communication framework. In proceedings of the international symposium on high-performance parallel and distributed computing 2018 (pp. 118-30). ACM.

Mingyu Lim is a Professor at the Department of Smart ICT Convergence, Konkuk University, Korea from the year 2017. He was an Assistant and Associate Professor at the Department of Internet & Multimedia Engineering, Konkuk Univeristy, Korea from 2009 to 2016. His current research activities are focused on Efficient Communication Middleware, Event Transmissions, and Content Distribution in Networked and Ubiquitous Computing Systems.
Email: mlim@konkuk.ac.kr