Performance Analysis of SyncML Server System Using Stochastic Petri Nets

Byung Yun Lee, Gil Haeng Lee, and Hoon Choi

Synchronization Markup Language (SyncML) is a specification of a common data synchronization framework for synchronizing data on networked devices. SyncML is designed for use between mobile devices that are intermittently connected to a network and network services that are continuously available on the network. We have designed and developed a data synchronization system based on the SyncML protocol and evaluated the throughput of the system using the stochastic Petri nets package (SPNP) and analyzed the relationship between the arrival rate and the system resources. Using this model, we evaluate various performance measures in different situations, and we estimate the relationship between the arrival rate and the system resources. From the results, we can estimate the optimal amount of resources due to the arrival rate before deploying the developed system.

Keywords: SyncML, PIMS, SPN.

Manuscript received Oct. 23, 2003; revised Feb. 19, 2004.

Byung Yun Lee (phone: +82 42 860 5191, email: bylee@etri.re.kr) and Gil Haeng Lee (email: ghlee@etri.re.kr) are with Broadband Convergence Network Research Division, ETRI, Daejeon, Korea.

Hoon Choi (email: hc@cnu.ac.kr) is with Chungnam National University, Daejeon, Korea.

I. Introduction

Synchronization is the process of making data consistent with data that is distributed via various devices such as PDAs, notebooks, and desktop computers. Device manufacturers have developed their own synchronization mechanisms, and as a result there is a lack of interoperability between devices and application services. To solve these problems, many device manufacturers such as Nokia, Ericsson, IBM, and so on have prepared the Synchronization Markup Language (SyncML) protocol specification [1].

In this paper, we propose an enhanced architecture for the SyncML server system which has the capability of synchronizing personal information management service data. To improve the performance of the data synchronization server, we have designed and implemented a software frame, the elements of which are a sync adapter, sync agent, sync engine, session manager, and an open database interface based on its own processing functions. In this study, we present the enhanced architecture of our system based on the concept of a software frame. To design the system in a frame-based manner provides more portability and scalability. The sync agent has the role of processing the synchronization commands, which are independent of specific applications. The sync engine has the role of processing the actual synchronization in the system. The session manager maintains the session information between a client and server. The open database interface has the role of processing and adapting all kinds of database systems.

To evaluate our system, we model our system using stochastic Petri nets and evaluate the model using the stochastic Petri nets package (SPNP) 6.0 [2], [3], and we analyze various performance factors including throughput, buffer length,

system utilization, and response time. Lastly, we present conclusions regarding our research.

II. SyncML Server Architecture

The SyncML server in this study consists of eight frames, as shown in Fig. 1. The server application provides an interface for a user or the application administrator to modify application data at the server. The server adapter passes messages from client devices to the SyncML toolkit via the communication adapter [4]. The server adapter also passes parsed data from the toolkit to the sync agent. The SyncML toolkit encodes and decodes SyncML messages.

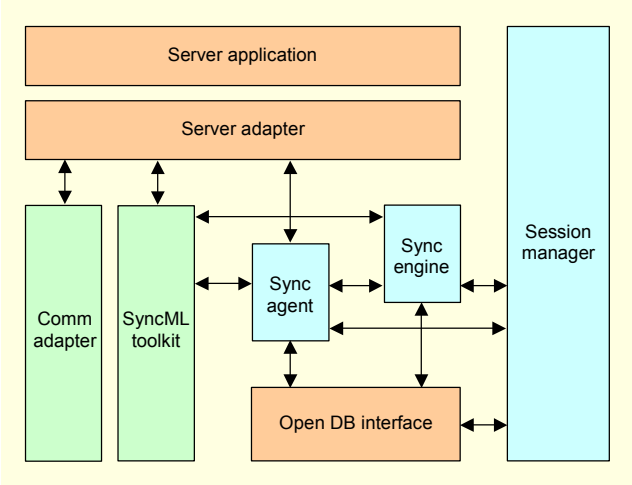


Fig. 1. Our SyncML server framework.

The sync agent implements the SyncML protocols. Therefore, its function is application independent and can be commonly used for all kinds of SyncML applications. The sync engine implements application-dependent parts such as service policies, as well as resolution rules in case of data conflicts. A brief scenario of the SyncML server is as follows.

i) The client wants to synchronize the data to the server.
ii) The client builds the SyncML message based on the data from the database. The client sends a set of commands which calls a function that sends a message to the sever using the available transport binding.
iii) The sync agent in the server analyzes the received message and finds changed records.
iv) The sync agent finds the available device which must change and requests the sync engine to modify the records of all devices because it is necessary to create consistency between different devices of same user.
v) The sync engine resolves the data and requests an open DB interface (ODBI) to update the data in the server.
vi) The sync engine makes and sends the status message to the client.

III. Performance Comparison

To evaluate the performance of our server, we measured the time to process one synchronization session in each type of server. The implementation and test environment was Microsoft Visual Studio C++ on Windows platforms, and the measurement was carried out with the parameters presented in Table 1.

As shown in Table 2, we measured the times from the SyncML demo architecture (SDA), SyncML Server 1.0 (session information in database), and Server 1.1 (session information in memory) under the same conditions. The SDA is the model architecture that was presented by the SyncML Initiative. The time we measured is the processing delay by the server only; it does not include the message transmission delay on the network. Test messages include mixed commands including add, replace, and delete commands.

Figure 2 depicts the results from the performance measurement. As we anticipated, SyncML Server 1.1 performs best. SyncML Server 1.1 is 44% faster than the SDA and 31% faster than SyncML Server 1.0. This is because the framework-based design and session-management mechanism in our system shows a better performance than that of existing servers based on the SDA. The Session Manager in SyncML Server 1.1 manages the session information of each device, and thus it has a faster processing time than the other mechanisms assessed here.

Table 1. Parameters for the performance test.

Parameter	Value
Number of users	10 Users
Number of devices	More than 2 devices for each person
Number of the synchronization.	500 times

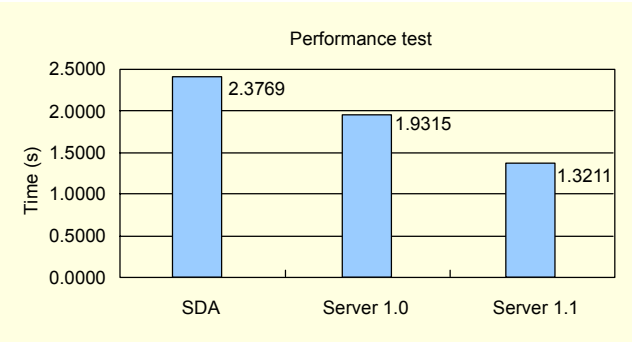


Fig. 2. A performance comparison.

Table 2. Comparison with existing server.

	Existing server (SDA2)	Our server
Architecture	Does not manage session (DLL structure)	Session is managed by session manager frame in a memory
Database connection	Dedicates DB (Lotus domino)	Provides open DB connectivity using open DB interface frame
Portability	Single structure without a frame	Divides by frame based on the processing functions, so enhances portability and scalability.
Processing method	Processes a set of commands without managing session, so it can process only one service	Processes command with managing session, so it can support more than one service.

IV. Petri Net Model

This section presents a Petri net model for the performance analysis of the proposed SyncML server system. Figure 3 illustrates the SyncML server system implemented for data synchronization. For a steady state evaluation, we use the SPNP which is a versatile modeling tool for the solution of stochastic Petri net (SPN) models [2], [5]. The SPN models are described in the input language for SPNP, C-based SPN language (CSPL), which is an extension of the C programming language with additional constructs that facilitate easy description of SPN models. SPNP can be employed to solve non-Markovian SPNs and is used to obtain the analytic numeric solution of Markovian model discrete-event simulations [6].

V. Performance Evaluation of SyncML Server

To evaluate the performance of the implemented server, we use SPNP 6.0 which is based on a stochastic Petri net model [7], [8]. In this model, we define the arrival rate (λ), service processing rate of the sync adapter (μ), memory size (pM), and number of concurrent DB connections (pDB). For each case, we change the value of λ and analyze the average number of waiting jobs in the input buffer, the probability that the input buffer is empty, and the bottleneck frame. We also evaluate system response time [9].

1. Measurement for the Parameters

To decide the parameter values of the Petri net model, we measure the real processing time per command in each defined frame (Table 3), and we reflect these values in the proposed Petri net model (Table 4). This measurement was performed in Windows 2000 (CPU: Pentium 1.4 GHz, Main memory: 128 MB). We defined the processing time for each frame as follows.

2. Processing Rate for the Defined Model

From the measurement of the processing time, we can obtain the rate of the exponential distribution rate, taking the reciprocal number of the mean processing time.

3. Performance Analysis

A. Average System Throughput

Throughput is the term used for the amount of commands that the system can process in a unit time [10], [11]. Figure 4 presents the system throughput of our system.

As shown in Fig. 4, with an increase of λ, the average system throughput of the modeled system increases gradually until λ is 1.5. When λ is above 1.5, the throughput of the system decreases until λ is 2.0, which is maintained as a constant value. This means that, in a given circumstance, the average system throughput of the modeled system is best when the arrival rate of the commands is near 1.5. These decreases in capacity are caused by a preemption of the system resources. As a result, we can estimate the best throughput of our server system before deploying the services.

B. Average Number of Waiting Jobs in the Input Buffer

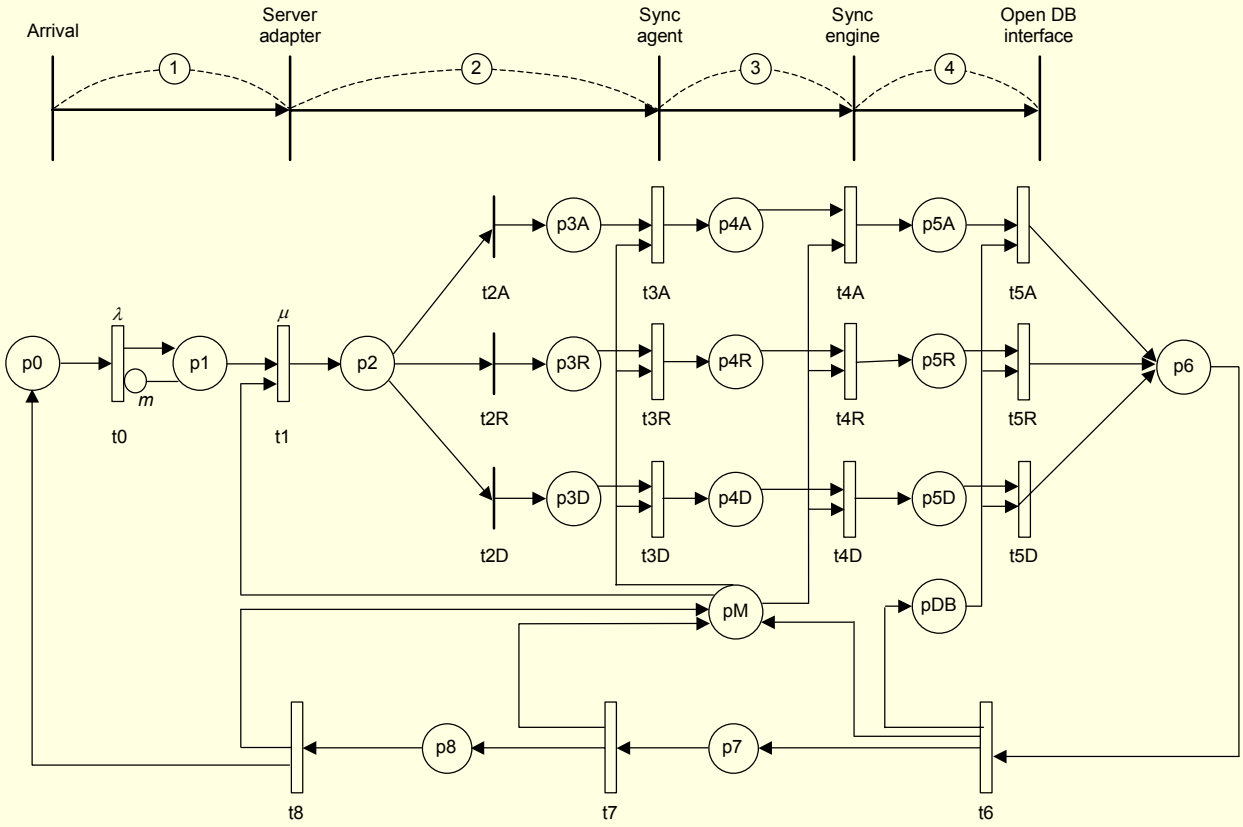
The request messages from the client wait for processing in the server input buffer. If many jobs arrive in the input buffer, those beyond the number of p1 are discarded.

As shown in Fig. 5, with an increase of λ, the average number of waiting jobs in the buffer increases gradually until λ is 1.5. When λ is above 1.5, the average number of waiting jobs maintains a constant value. This means that the average number of waiting jobs is not in excess of the number of input buffers in a server; it increases gradually, and when it comes to reach the threshold of the system, it then converges on the predefined value.

C. The Probability the Input Buffer is Empty

If the job arrival rate (λ) is too low or the system processing rate is too high, the system input buffer is almost empty, and therefore it is profitable to reduce the input buffer size or degrade the hardware of the system. Figure 6 presents the probability that the input buffer is empty.

As shown in Fig 6, with an increase of λ, the probability that the input buffer is empty decreases gradually. This means that if the job arrival rate exceeds the processing capability of the system, the probability converges on 0.



Place p0: Initial number of commands to process
Place p1: Number of input buffers in the server to hold the SyncML commands
Place p2: Branch place for command type
Place p3A, p3R, p3D: Place for sync agent processing (add, replace, and delete commands, respectively)
Place p4A, p4R, p4D: Place for sync engine processing (add, replace, and delete commands, respectively)
Place p5A, p5R, p5D: Place for ODBI processing (add, replace, and delete commands, respectively)
Place p6: Place after ODBI processing
Place p7: Place after sync engine processing
Place p8: Place after sync agent processing
Place pM: Number of available resources in the server
Place pDB: Number of concurrent DB connections
Transition t0: Transition for processing the arriving jobs, having the number of inhibitor arcs, which represent the available input buffer.
Transition t1: Place after sync adapter processing; the rate of the sync adapter is μ
Transition t2A, t2R, t2D: Transition for processing the add, replace, and delete commands; each taking place with a reward functions probability defined by prob2A, prob2R, and prob2D, respectively.
Transition t3A, t3R, t3D: Transition for sync agent processing (add, replace, and delete commands)
Transition t4A, t4R, t4D: Transition for sync engine processing (add, replace, and delete commands)
Transition t5A, t5R, t5D: Transition for ODBI processing (add, replace, delete command)
Transition t6: Transition after ODBI processing
Transition t7: Transition after sync engine processing
Transition t8: Transition after sync agent processing
m: multiplicity of the inhibitor arc which disables the transition when the number of Petri net tokens in the place is greater than or equal to m
λ: arrival rate of the synchronization commands from the client devices
μ: service rate of sync adapter

Fig. 3. Petri net model for SyncML server.

Table 3. Average processing time per command (sec/command).

	Tadapter	Tsa	Tse	Todbi
Add	0.17	0.22	0.612	0.12
Replace	0.17	0.1732	0.51	0.14
Delete	0.126	0.1071	0.245	0.1

Tadapter: Average processing time in a sync adapter frame
Tsa: Average processing time in a sync agent frame
Tse: Average processing time in a sync engine frame
Todbi: Average processing time in an open DB interface frame

Table 4. Number of commands per second (commands/sec).

	Tadapter	Tsa	Tse	Todbi
Add	5.88	4.55	1.63	8.33
Replace	5.88	5.77	1.96	7.14
Delete	7.94	9.34	4.08	10

Tadapter: Average processing time in a sync adapter frame
Tsa: Average processing time in a sync agent frame
Tse: Average processing time in a sync engine frame
Todbi: Average processing time in an open DB interface frame

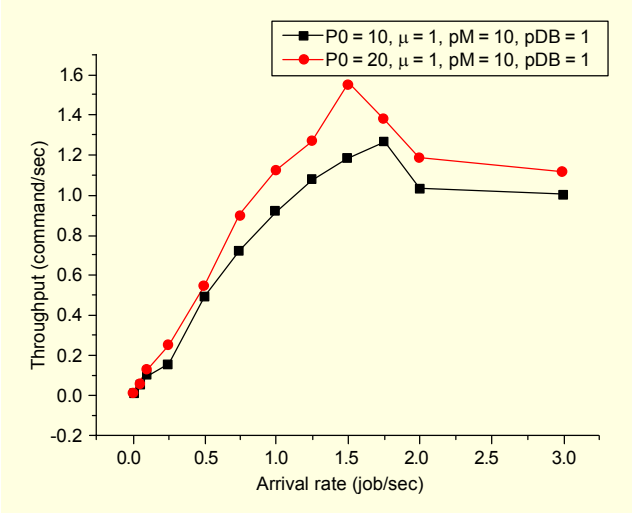


Fig. 4. System throughput (p0 = 10 and p0=20).

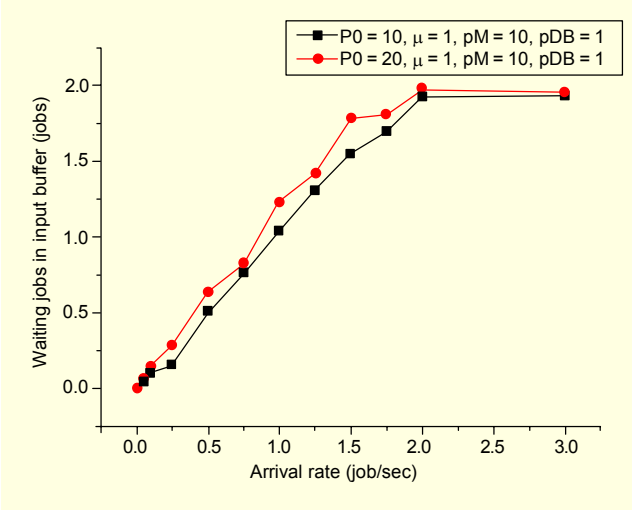


Fig. 5. Average number of waiting job.

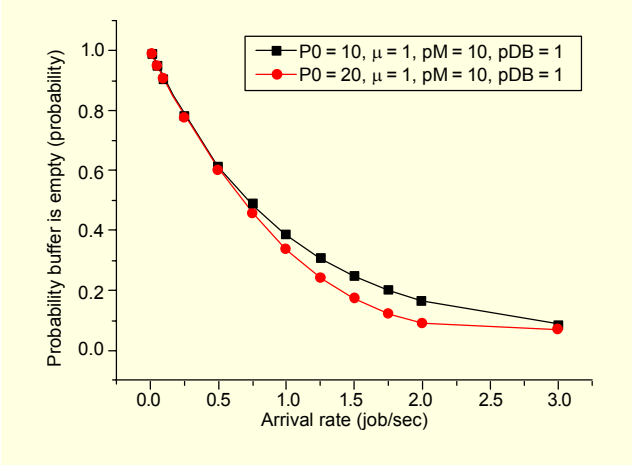


Fig. 6. The probability that the input buffer is empty.

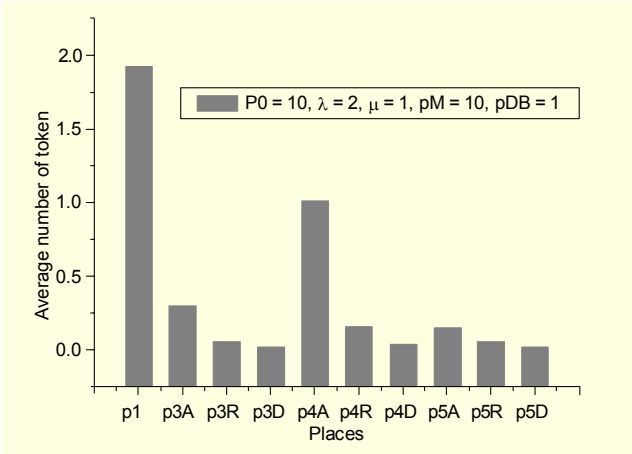


Fig. 7. Average number of tokens in each place.

D. Bottleneck Frame

Our developed system consists of frames, and while processing the various synchronization commands, a bottleneck may occur in a specific frame. This may be influenced by the type of services and synchronization commands.

To analyze the bottleneck frame of the server, we evaluate the proposed Petri net model for the following cases (p0 = 10, $\lambda = 2$, $\mu = 1$, pM = 10, pDB = 1) and count the average number of tokens in each place. As shown in Fig. 7, places p1 and p4A have many tokens to process the synchronization. This is because all the synchronization requests from the client must arrive and wait in the input buffer p1, and t4A takes the longest time to complete its operation. In this SPNP model, we define the reward function probability to reflect the distribution of synchronization commands. This reward function depends on the application of the synchronization; in this model, we assume that reward functions of the add commands constitute 70 percent of the total synchronization commands.

E. System Response Time

System response time is the processing time for requests from the client to arrive at the server, which processes the synchronization commands, and the response messages to return to the client. It is very important to forecast the response time of the designed system. We have measured the real processing time of the developed system, and we reflected the various factors of the system to the defined Petri net model. In doing so, we can change the various factors that affect the system performance and estimate the real performance of the system for many different cases such as the job arrival rate, processing rate of the input buffer, and system resources defined in the Petri net model. To measure the response time of the system, we redefine the Petri net model with respect to the

add command and measure the response time of the add command. We assume that there is a single token in p0. This is done in order to simplify the redefined model and minimize the states of the system.

λ is the arrive rate per second (job/sec). Thus if λ is 0.01, then 1 job arrives at the system in 100 seconds. Therefore, the inverse number of the arrival rate is the inter-arrival time between continuous jobs. We define the Ra as the response time of the add command and Pa as the probability that a token exists at p0. Then,

$$\mathrm{Pa} = \frac{1/\lambda}{1/\lambda + \mathrm{Ra}}. \quad (1)$$

From (1)

$$\mathrm{Ra} = \frac{\frac{1}{\mathrm{Pa}} - 1}{\lambda}. \quad (2)$$

As defined in (2), the mean response time of add command Ra is determined by the job arrival rate, λ, and the probability, Pa. Using pr_std_average() in the SPNP package, we can obtain the probability of a token existing in p0A. The probability of the token in p0A (p0 = 5, μ = 1, pM = 15, pDB = 1) is given in Table 5. We can also obtain the average response time using (2).

As shown in Table 5, using the configuration of a windows 2000-based personal system, we estimated that system response time Ra is about 1 second.

4. Performance Estimation

In the proposed model, if pM and job processing rate μ are fixed, then the throughput of the system increases steadily until the system exhausts the resources; at which point, the throughput shows a slight decrease. As a result, we can estimate the optimal throughput for various situations.

Table 5. Average response time (p0 = 5, μ = 1, pM = 15).

λ	Prob[p0A]	Expected response time (s)
0.5	6.7961754652e-01	0.9428316120
0.75	5.8555629185e-01	0.9437036516
1.0	5.0300074997e-01	0.9880686063
1.25	4.3581453893e-01	1.0356432118
1.5	3.8212781385e-01	1.0779502978
1.75	3.3906039896e-01	1.1139011608
2.0	3.0410485167e-01	1.1441697567
3.0	2.3834923843e-01	1.1834948327
4.0	1.7383294657e-01	1.2405323349
5.0	1.3303014366e-01	1.3034186575

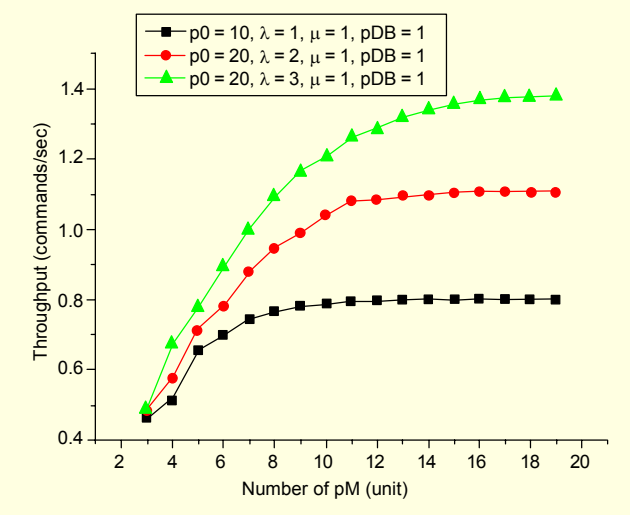


Fig. 8. Estimated throughput in various situations.

As shown in Fig. 8, with an increase of pM, the throughput of the system can be obtained in various circumstances. In each case, we can estimate the best throughput of the system before deploying the server system.

VI. Conclusions

Estimating the throughput of a system is important to analyze and optimize the implemented system. The SPNP package used in this study is based on the Markov model, which represents the behavior of the system as a Petri net model, and transforms it into an analytic Markov model. Through this package, we can measure the various performance-related characteristics of the modeled system. We employ analytic modeling for the developed system in order to focus on the real factors that influence the system.

In this paper, we model the system exactly and adapt the various factors that are acquired from the developed system. We evaluate the processing time of synchronization commands in each frame of the system and define the reward function, adapting it to the Petri net model.

Using this model, we can evaluate various performance measures in different situations, and we can estimate the relationship between the arrival rate and the system resources.

As a result, we can estimate the optimal amount of resources due to the arrival rate before deploying the developed system.

References

[1] SyncML Initiative, SyncML Representation Protocol version 1.1.1, Oct. 29, 2002.

[2] Kishor S. Trivedi., SPNP Package 6.0 Manual, Sept. 1999, http://www.ee.duke.edu/~chirel/index.html, Oct. 2002.
[3] Jang Hyun Baek and Byung Han Ryu, "Modeling and Analysis of Distance-Based Registration with Implicit Registration," *ETRI J.*, vol.25, no.6, Dec. 2003, pp.527-530.
[4] SyncML Initiative, SyncML HTTP Binding version 1.1.1, Oct. 29, 2002.
[5] Dong-Hyun Heo, Sang-Wook Chung, and Gil-Haeng Lee, "The Effects of Management Traffic on the Local Call Processing Performance of ATM Switches Using Queue Network Models and Jackson's Theorem," *ETRI J.*, vol.25, no.1, Feb. 2003, pp.34-40.
[6] Tae-Sung Kim, "Performance Analysis of a Discrete-Time Two-Phase Queueing System," *ETRI J.*, vol.25, no.4, Aug. 2003, pp.238-246.
[7] L. Ojala, E. Parviainen, Penttinen, H. Beaver, T. Tynjala, "Modeling Feynman's Quantum Computer Using Stochastic High Level Petri Nets Systems," *IEEE Int'l Conf. on Systems, Man, and Cybernetics*, vol. 4, Mar. 2001, pp. 2735-2741.
[8] Chuang Lin, Mingwei Xu, and Marinescu, "A Petri Net Approach to Stability Analysis of Buffer Priority Scheduling Policies in Manufacturing Systems," *IEEE Int'l Conf. on Systems, Man, and Cybernetics*, vol. 3, Mar. 2001, pp. 1811-1816.
[9] Lopez-Benitez, "Petri Net Based Performance-Evaluation of Distributed Homogeneous Task Systems," *IEEE Trans. on Reliability*, vol. 49, Dec. 2000, pp. 188-198.
[10] M. Ajmone Marsan, M. Gribaudo, "On Petri Net-Based Modeling Paradigms for the Performance Analysis of Wireless Internet Accesses," *9th Int'l Workshop on Petri Nets and Performance Models*, Oct. 2001, pp. 19-28.
[11] Young Ha Hwang, "A Performance Analysis of TMN Systems Using Models of Networks of Queues, Jackson's Theorem and Simulation", *ETRI J.*, vol.24, no. 5, Oct. 2002, pp. 381 -390.

Byung Yun Lee is currently a Senior Member of BcN Research Division at Electronics and Telecommunication Research Institute (ETRI), Korea. He received the PhD degree in computer engineering from Chungnam National University, Korea, in 2003. Since joining ETRI in 1992, his work has focused on mobile communications, network technology, and network management.

Gil Haeng Lee is a Principal Member of Engineering Staff and Team Leader at ETRI, Korea. He received the MS and PhD degrees in computer science from KAIST in 1986 and 1996. His research interests are in load balancing and distributed processing, network management, speech recognition, and real time DBMS.

Hoon Choi is currently Associate Professor, Dept. of Computer Engineering at Chungnam National University, Korea. He received the PhD degree in computer science from Duke University, Durham, NC, USA, in 1993. His work has focused on mobile communications, distributed systems, fault-tolerant systems, and performance modeling using stochastic Petri nets.