JDCS
디지털콘텐츠학회논문지
Journal of Digital Contents Society
Vol. 19, No. 2, pp. 407-413, Feb. 2018

사물인터넷 컴퓨팅 환경에서 QoS를 고려한 데이터 전송 구조

이 윤 석
한국외국어대학교 컴퓨터전자시스템공학부

QoS-aware Data Delivery Infrastructure for IoT Computing Environments

Yunseok Rhee
Division of Computer & Electronic Systems Eng., Hankuk University of Foreign Studies, Yongin 17035, Korea

[요 약]

최근 사물인터넷(IoT) 기술의 발전과 함께 수많은 센서와 소형 구동장치들로 구성된 새로운 컴퓨팅 환경이 도래했다. 본 논문은 이와 같은 IoT 기반 컴퓨팅 환경에서 데이터 제공자들과 소비자들 사이에 센싱 데이터를 쉽게 공유하고 접근하도록 지원하는 공유 플랫폼으로서, 확장성있는 데이터 전송 기반구조를 제안한다. 확장성과 효율성을 제공하기 위해, 이 논문은 특히 소비자들 간의 서로 다른 QoS 요구사항을 활용하여 전송 대역폭을 효과적으로 활용한 전송 경로를 구성하는 방법을 제시한다. 전송경로 구성과 재구성 과정이 제안하는 구조의 가장 큰 오버헤드로 판단되므로, 본 논문에서는 그 비용을 산정하는 기본적인 실험을 수행하였는데, 결과는 제안된 구조의 우수한 확장성에 비해 오버헤드는 비교적 적은 것으로 확인되었다.

[Abstract]

In this paper, we present a scalable data delivery infrastructure for such IoT computing environment where we need a common platform where data providers share their diverse sensing data and applications can easily access and receive such data from providers. For efficient data delivery, this paper proposes a new delivery path management technique that take advantage of diverse consumer QoS when building bandwidth-efficient delivery paths. We perform primitive experiments on the path construction and reconstruction which may be major overhead of the scalable infrastructure. The results show that the proposed infrastructure achieves a high level of scalability, and demonstrates that the management overhead is not significant.

색인어 : 데이터 전송 구조, 사물인터넷, 센싱 응용, Quality of Service (QoS)

Key word : Data delivery infrastructure, Internet of Things (IoT), Sensing application, Quality of Service (QoS)

http://dx.doi.org/10.9728/dcs.2018.19.2.407

Received 19 February 2018; **Revised** 26 February 2018
Accepted 27 February 2018

***Corresponding Author;** Yunseok Rhee

Tel: +82-10-3429-5768
E-mail: rheeys@hufs.ac.kr

Ⅰ. Introduction

The rapid advance of Internet of Things (IoT) technologies has opened up a new computing environment with a myriad of sensors and tiny devices. It enables numerous sensing real-time applications to grow in scale, complexity, and diversity. In addition to static sensor networks, recent IoT technologies tend to enrich such applications in a more dynamic and instant fashion [1-4]. In particular, mobile sensing nodes such as mobile phones and vehicles with sensors are promoting the practical use and the deployment of such dynamic sensor networks [5,6].

Also, recent fog networking enables lots of computing devices to interact at edge nodes of a network, and results in fully distributed data and application processing [7]. These network-centric computing environments require a QoS-aware data delivery infrastructure that efficiently shares limited network resources while meeting the service requirements of each application [8]. Without these infrastructures, building an infrastructure for each application is not easy nor inefficient.

These real-time detection applications must continually monitor the current state of a particular entity or situation in a particular region so that application users can provide context-aware services in a timely manner. The applications have requirements for a new type of delivery service. Depending on the application users' business or personal purposes, users want to observe data of interest in different QoS levels of detail, and the applications need real-time delivery in the sense that they receive data within the quality of service to perform agile actions.

Ⅱ. Network Architecture

We present a scalable data delivery infrastructure, in short SDI, for continuous sensing applications in IoT environments. As in Fig. 1, the infrastructure relies on a collection of nodes that are strategically located. Data providers and consumers connect to SDI through nearby SDI nodes. For each data provider, SDI forms an efficient delivery path along with the data from the supplier to all interested consumers. Data consumers receive data delivery service through SDI according to the following procedure based on the publish/subscribe system [9-11].

First, the service description for the sensor network is obtained through the SDI portal, which helps consumers to search for interesting IoT networks with relevant information such as supplier ID, location, data type, and so on. According to the service description, each consumer creates a query specifying his/her requirement. The query is then forwarded to the SDI

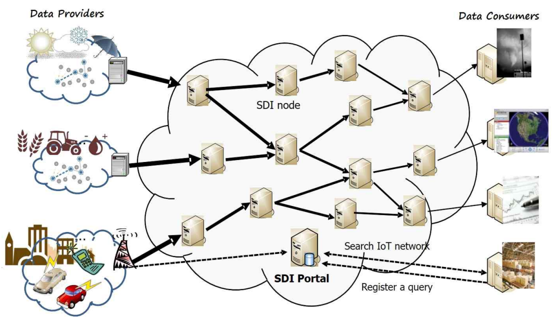


그림 1. SDI 서비스 개략
Fig. 1. SDI service overview

portal for registration. Upon receipt of the query, the SDI portal designates the closed SDI node to the consumer as a proxy. When SDI establishes an end-to-end delivery path from a provider to a designated proxy node, a consumer receives the data of interest through the assigned SDI nodes within an tolerable delay. A similar process allows data providers to publish data to consumers through SDI. Each provider that owns a sensor network registers network specifications which contains network ID, data schema, data rate, and network description, etc.

As shown in Fig. 2, SDI nodes are hierarchically organized for scalable network management, and nodes are clustered based on network proximity. With an appropriately sized cluster unit, the SDI network is managed in a two-level hierarchy of these clusters. The hierarchy can be extended to more levels if necessary, but for simplicity, we consider only two levels in this work. In each cluster, one of the SDI nodes is elected as a local head. With all the local heads, a second-level cluster is formed and one of them is elected as a global head. Each local or global head node must be located near the center of its cluster for efficient cluster management; the node with the least sum of network latencies to other members within its cluster is chosen as a head node. The head node manages the size of the cluster (i.e., the number of nodes in the cluster), and splits a large cluster or merges small clusters so as to keep the size of each cluster within a certain range. The head node maintains information about each member node such as available bandwidth, latencies to other member nodes from the node, etc. The information is used not only for cluster management, but also for efficient delivery path management.

A data delivery path is established based on the cluster-based hierarchy mentioned above. That is, a node is responsible to deliver data to all member nodes in its cluster, and also creates a tree rooted at the node. We refer to the tree formed in each local cluster as a local tree and its root node as LPX (Local ProXy), respectively. In a local cluster, the first member node, i.e., the root of a delivery tree becomes an LPX of the cluster. Similarly,

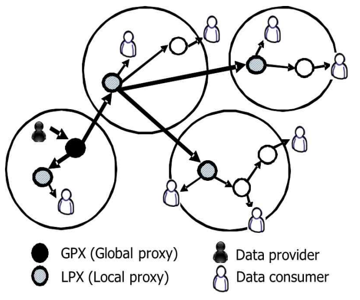


그림 2. SDI의 계층적 구조
Fig. 2. Hierarchical Structure of SDI

the tree that connects the local trees is referred to as a global tree, and its root node as GPX (Global ProXy), respectively. The SDI node closest to a provider is chosen as a GPX for the provider, which sends data to consumers through the GPX. Since SDI consists of one or more delivery paths, An SDI node can act differently for each delivery path it serves. GPX and LPX jointly manage the delivery path for scalable path management. We hereafter refer to both GPX and LPX simply as manager nodes, since the operations performed at a GPX or an LPX are almost same.

Each manager node is responsible for a small portion of the overall delivery path. Based on the node information from its local head, it constructs delivery paths connecting all the other members in the cluster. For efficient path construction, the number of members per manager needs to be less than a hundred. We expect that the administrative overhead imposed on each manager node is not significant for the SDI network.

Ⅲ. Delivery Path Management

3-1 Delivery Path Model

For simplicity, we focus on the construction and maintenance of a single delivery tree. Given a delivery tree for a provider, each member, m_i, declares its own QoS requirement, and aggregates the requirements of all consumers connected to itself. As an example, in this paper, we consider two QoS parameters of data volume (or bandwidth) and delay for a member m_i, and denote as B_i and d_i, respectively.

An aggregated QoS requirement at a member is defined as the accumulation of delivery requirements imposed on all nodes over the path from the root to the member m_i. As a result, the aggregated requirement at m_i should meet itself as well as all its descendents as in Figure 3. We denote the aggregated requirements of data and delay at m_i as B_i^a and d_i^a, respectively.

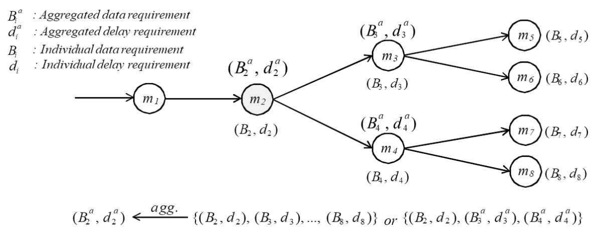

B_i^a : Aggregated data requirement
d_i^a : Aggregated delay requirement
B_i : Individual data requirement
d_i : Individual delay requirement
m_1
m_2
m_3
m_4
m_5
m_6
m_7
m_8
(B_2^a, d_2^a)
(B_2, d_2)
(B_3^a, d_3^a)
(B_3, d_3)
(B_4^a, d_4^a)
(B_4, d_4)
(B_5, d_5)
(B_6, d_6)
(B_7, d_7)
(B_8, d_8)
$(B_2^a, d_2^a) \xleftarrow{agg.} \{(B_2, d_2), (B_3, d_3), ..., (B_8, d_8)\}$ or $\{(B_2, d_2), (B_3^a, d_3^a), (B_4^a, d_4^a)\}$

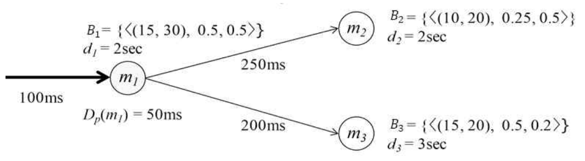
그림 3. 총량화된 QoS 요구사항
Fig. 3. Aggregated QoS requirement

It can be recursively computed from the individual delivery requirement of m_i and the aggregated ones of m_i's all descendants.

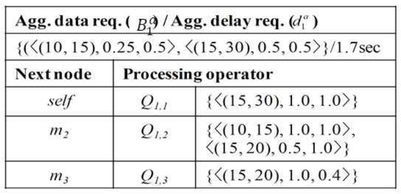
3-2 Selective Forwarding

Each node is designed to maintain data and forwarding requirements on each node with a data structure called a Selective Forwarding Table (SFT) to efficiently deliver data to the descendants of the node. The forwarding information in an SFT contains a list of next members and processing operators that must be applied before passing the data to the next member. We denote an operator applied at member m_i while forwarding incoming data to the member m_j by $Q_{i,j}$. The number of processing elements, k_j, depends on the range of data values required by its descendants. Similar to the aggregated requirements, a processing operator also specifies the range of data values of interest to each subsequent node as well as spatial and temporal sampling rates to be applied to incoming data streams prior to delivery. Given an operator { <d_{min}, d_{max}>, res_{time}, res_{space}} for the next node m_j, the data is forwarded to m_j at corresponding spatial and temporal sampling rates of res_{time} and res_{space} only when a incoming data meets the range (d_{min}, d_{max}). The operator is applied to remove unnecessary data for m_j. Thus, the next member m_j receives data as specified in its aggregated data requirement B_j^a.

In the example shown in Figure 4(b), the aggregated data requirement B_1^a at m_1 is {<(10, 15), 0.25, 0.5>, <(15, 30), 0.5, 0.5>} and the aggregated delay requirement d_1^a is 1.7sec, which means that m_1 requires data between 10 and 15 with spatial and temporal resolutions of 0.25 and 0.5, respectively, and those between 15 and 30 with spatial and temporal resolutions of 0.5 and 0.5, respectively, within 1.7sec. The figure also shows an SFT construction example at m_1 which satisfies the delivery path shown in Figure 4(a). In the table, an operator $Q_{1,2}$, {<(10, 15), 1.0, 1.0>, <(15, 20), 0.5, 1.0>} is applied at m_1 for the member m_2. Thus, it filters data within (10, 15) or (15, 20), and then forwards the data with the associated spatial and temporal


$B_1 = \{\langle(15, 30), 0.5, 0.5\rangle\}$
d_1 = 2sec
100ms
$D_p(m_1)$ = 50ms
m_1
250ms
200ms
m_2
$B_2 = \{\langle(10, 20), 0.25, 0.5\rangle\}$
d_2 = 2sec
m_3
$B_3 = \{\langle(15, 20), 0.5, 0.2\rangle\}$
d_3 = 3sec

(a) forwarding example



Agg. data req. (B_1^a) / Agg. delay req. (d_1^a)		
{(⟨(10, 15), 0.25, 0.5⟩, ⟨(15, 30), 0.5, 0.5⟩}/1.7sec		
Next node	**Processing operator**	
self	$Q_{1,1}$	{⟨(15, 30), 1.0, 1.0⟩}
m_2	$Q_{1,2}$	{⟨(10, 15), 1.0, 1.0⟩, ⟨(15, 20), 0.5, 1.0⟩}
m_3	$Q_{1,3}$	{⟨(15, 20), 1.0, 0.4⟩}

(b) selective forwarding table

그림 **4.** 선택적 전달 구조 예
Fig. 4. Selective Forwarding Structure Example

sampling rates. The SFT also includes a processing operator $Q_{1,1}$, {<(15, 30), 1.0, 1.0>} for itself, in this example, operation on the data from m_1 to m_1. The operator exists to obtain data that match m_1's own delivery requirement. Then, the data is selectively delivered to all consumers connected to m_1 based on their delivery requirements.

3-3 Path Construction

In this section, we describe delivery path construction which may be conducted in an urgency- and similarity-based manner. Note that the task determines not only parent-child relationships between nodes but also the appropriate processing to be taken at each node prior to delivery of incoming data to subsequent nodes. We assume that initially only the root member is in a delivery tree, and each new member is then added to an existing tree.

A simple way to add a new member is to select a parent node from existing member nodes without changing the existing tree structure. It is simple, but the method is not effective in practice. First, the insertion of a new member can increase the bandwidth requirements of the ancestor members, and potentially exceed the available amount. Also, if the new node has a tight delay requirement, the situation becomes more complicated. Members must be added upward to the tree near to the root due to the stringent requirements. However, available bandwidth often becomes scarce near the root.

Consider a member trying to join a delivery tree. It first contacts the manager node that is responsible for managing the tree and the join requests (i.e., the root member in the delivery path). Upon request, the manager node selects the best parent member for the new member. Note that our member joining process does not simply specify an adequate parent member to the new member; rather it figures out a valid tree structure that can accommodate all of the existing members as well as the new member while satisfying their delivery requirements. The join process attempts a tree update plan to include a new member, and the plan is executed to actually update the delivery tree.

The join process is performed in two phases; (1) normal join and (2) aggressive join. The normal join phase conducts to add the new member m_k while maintaining the current tree structure. If a new member cannot be accommodated, it begins an aggressive join phase where partial rearrangement is performed to accommodate the member. Thus, the aggressive join may cause some members to be detached and rejoined later. The join process is therefore a recurring iterative process for the new member and the detached members. Unless it can find an adequate parent for either the new member m_k or the other members to rejoin, the join process consequently fails. We then determine that the current network resources are insufficient to accommodate the new member.

Given a new member m_k, the normal join phase finds a parent member that can deliver data to m_k while satisfying the delivery requirement. For the purpsose, it first finds parent candidates out of all existing members and then selects one that provides the most efficient delivery path in terms of bandwidth consumption. To determine whether an existing member m_p can become the parent candidate of m_k or not, we check m_p with a few conditions under the assumption that m_p becomes the parent of m_k.

3-4 Path Reconstruction

If no candidate parent meets the previous three conditions on bandwidth and delay constraints, an aggressive join phase will begin. The aggressive join means that the new member selects its parent m_p while leaving out some existing children with lower priorities in terms of urgency (see Figure 5). This phase is performed in two steps as follows. First, even if some conditions are violated, the new member is added to the tree by selecting a parent. Then, the violations would be eliminated by partially reorganizing the tree structure. Prior to the rearrangement, a plan must be established to describe the sequence of detachment and re-joining of members. This plan should accommodate new members as well as existing members after relocation. If an adequate plan is not found, the join process fails. We also determine that the current network resources are insufficient to include the new member.

In the case of aggressive join phase, the three conditions are incrementally alleviated one by one to select a candidate parent. Such relaxations allow us to take into an account the members not considered in the normal join phase. As before, we let m_k be the new member and m_p be the candidate parent member being tested.

In this requirement relaxation process, a candidate parent is selected by iteratively detaching some of its children and checking the previous three conditions after the detachment. Suppose m_p detaches its children with aggregated delay requirements greater than m_k (see also Figure 5). The changed bandwidth consumption of m_p after the detachment is denoted by W_p'', and the changed aggregated delay requirement is denoted by d''^{a}_{p}, respectively. The difference between W_p and W_p'' is also denoted by Δ_p'. At this time, m_p can be a candidate parent if it meets these three conditions after the detachment.

i) Parent bandwidth condition: $\Delta_p' \leq \Phi(m_p)$

ii) Delay condition: $d''^{a}_{p} \geq D(m_{a_n}, m_p)$

iii) Ancestor bandwidth condition:

$\Delta_{a_0}' \leq \Phi(m_{a_0}), \Delta_{a_1}' \leq \Phi(m_{a_1}), ..., \Delta_{a_n}' \leq \Phi(m_{a_n}),$

If several parent candidates are found, a candidate with the best bandwidth saving is selected. We may not find a proper candidate for the new member m_k, despite the first relaxation. Then, in the further relaxation steps, we attempt to alleviate delay condition and ancestor's bandwidth condition, each one from the above requirement conditions, respectively, to find an adequate position while detaching ancestors' children.

After selecting a parent for the new member, we establish a rearrangement plan to construct a valid delivery tree which accommodates the new member. The plan is performed in two steps. First, we look up the members with bandwidth violations, and select the members to detach. To remove the bandwidth violation of a member, we select its child members with the longest delay requirement for detachment. The selection can be repeated until the bandwidth violation is removed.

Ⅳ. Experiments

4-1 Experimental Setup

We simulated an virtual network of SDI nodes over the network topology modelled by DS2 [12], which specifies static round-trip propagation delays between all pairs of network nodes. The default number of nodes is set to 4,000 and the default node bandwidth to 5Mbps. These parameters are chosen to demonstrate the performance of SDI in a large-scale environment where delivery paths consist of many nodes with limited bandwidth resources. In this experiment, we limit the number of clusters to 100.

For query workloads, we consider two different scenarios which have similar characteristics, but different preferences on selecting data providers. In the first, we assume that data consumer is a real world navigation application where application users want to observe sensing data from all siets worldwide. In the second, we assume that data consumer is a mesoscale weather forecast application where users are primarily interested in local weather information. In the real world navigation applications, data providers are selected randomly. On the other hand, in the mesoscale weather forecasting applications, the provides are selected based on the proximity to data consumers. Based on a Zipf distribution, adjacent data providers are selected with high probability, whereas distant ones are selected with low probability.

4-2 Path Management Overhead

In this paper, we show the overhead of our path construction process. To do this, we measure the average number of join operations that are executed to include a new member. Recall that our scheme performs path rearrangement to accommodate the members that are not included only by the normal join. The rearrangement is performed by detaching and rejoining some members, and thus additional join operations are executed. To clearly show the cost of join operations, a node is allowed to have at most one query for each data provider. In this experiment setting, a new query registration corresponds to a new member join.

Figure 5 compares the path management overheads with the normal join only scheme and with normal join and aggressive join scheme, respectively. It is obvious that the scheme using aggressive joins is more scalable and efficient, but the overhead will be quite significant. In the experiments, however, our path construction scheme with both the normal and the aggressive join phases shows a slightly higher number of join operations, ranging from 1.02 to 1.19, compared to the scheme of normal join only. This result shows that the overhead by the aggressive join phase is not significant.

To better understand the effects of aggressive joins, we need to take a closer look at the results in Figure 6. As described in Section 3.2, in the aggressive join phase, the three conditions of the normal join phase are mitigated one by one. To show the overhead of each join type, we measure the average number of actually performed join operations for each join operation. To add a new member with a normal join, the join operation needs to be executed only once for the new member itself. However, to add the member with an aggressive join, more than one join operation is required because the aggressive join may generate a few rejoin operations. In the case of the relaxation, about 3.5 join operations are required, generating 2.5 rejoin operations on the average. The rejoined members are the ones that have longer delay requirements than the newly joined member.

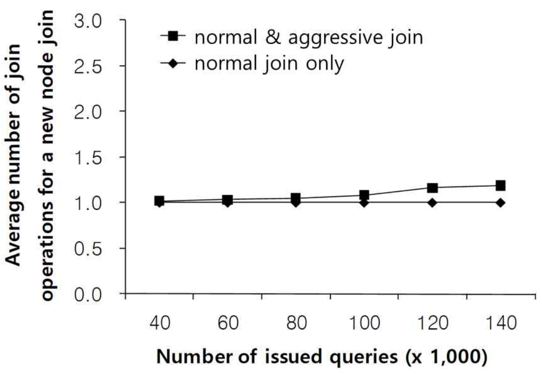


그림 **5.** 경로 관리 오버헤드
Fig. 5. Path Management Overhead

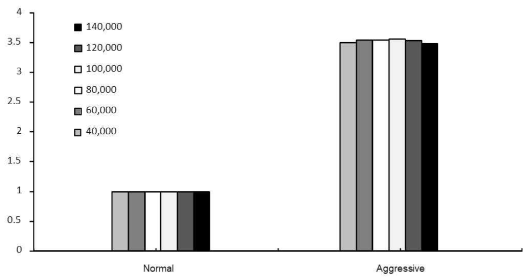


그림 **6.** 평균 Join 연산 횟수
Fig. 6. Average Number of Join Operations

Ⅴ. Conclusions

We have presented a scalable data delivery infrastructure named SDI, for IoT computing environment where we need a network platform for data providers to share their diverse data with lots of consumers and for applications to easily access and receive sensor data from the providers. For efficient data delivery, this paper proposes a new delivery path management scheme that carefully exploits the delivery QoS requirements of different consumers in constructing bandwidth-efficient delivery paths. We perform primitive experiments on the path construction and reconstruction which may be major overhead of the scalable infrastructure. The results show that SDI achieves a high level of scalability, and demonstrates that the overhead incurred by the aggressive join phase is not significant.

Acknowledgements

This work was supported by Hankuk University of Foreign Studies Research Fund of 2017.

참고문헌

[1] J. Lin, W. Yu, N. Zhang, X. Yang, H. Zhang and W. Zhao, "A Survey on Internet of Things: Architecture, Enabling Technologies, Security and Privacy, and Applications," *IEEE Internet of Things Journal*, vol. 4, no. 5, pp. 1125-1142, Oct. 2017.
[2] Eleonora Borgia, "The internet of things vision: Key features, applications and open issues," *Computer Communications*, 54:1–31, 2014.
[3] Jeong-Rae Cho, Hye-Suk Kim, Doo-Keol Chae, and Suk-Ja Lim, "Smart CCTV Security Service in IoT Environment," *Journal of Digital Contents Society*, vol. 18, no. 6, pp. 1135-1142, 2017.
[4] Seong-Pyo Hong, "Design and Implementation of amount of contained water, earth and sand Monitoring System based on IoT," *Journal of Digital Contents Society*, vol. 18, no. 4, pp. 787-793, 2017.
[5] B. B. P. Rao, P. Saluia, N. Sharma, A. Mittal and S. V. Sharma, "Cloud computing for Internet of Things & sensing based applications," in *Proceedings of Sixth International Conference on Sensing Technology (ICST)*, Kolkata, pp. 374-380, 2012.
[6] Dongyu Wang, Dixon Lo, Janak Bhimani, Kazunori Sugiura, "AnyControl -- IoT Based Home Appliances Monitoring and Controlling", in *Proceedings of the IEEE Annual Computer Software and Applications Conference (COMPSAC)*, Taichung, vol. 3, pp. 487-492, 2015.
[7] M. Chiang and T. Zhang, "Fog and IoT: An Overview of Research Opportunities," *IEEE Internet of Things Journal*, vol. 3, no. 6, pp. 854-864, Dec. 2016.
[8] G. Daneels et al., "Real-Time data dissemination and analytics platform for challenging IoT environments," in *Proceedings of Global Information Infrastructure and Networking Symposium (GIIS)*, St. Pierre, pp. 23-30, 2017.
[9] Sasu Tarkoma, *Publish/subscribe systems: design and principles*, John Wiley & Sons, 2012.
[10] D. Sarkar, N. Rakesh and K. K. Mishra, "Content delivery networks: Insights and recent advancement," in *Proceedings of Fourth International Conference on Parallel, Distributed*

and Grid Computing (PDGC), Waknaghat, pp. 1-5, 2016.

[11] H. Yin, X. Liu, G. Min, C. Lin, "Content Delivery Networks: a Bridge between Emerging Applications and Future IP Networks," *IEEE Network*, vol. 24, no. 4, pp. 52-56, 2010.

[12] B. Zhang, T. S. E. Ng, A. Nandi, R. Riedi, P. Druschel, and G. Wang, "Measurement based analysis, modeling, and synthesis of the Internet delay space," *IEEE/ACM Transactions on Networking*, vol. 18, no. 1, pp. 229-242, 2010.

이윤석 (Yunseok Rhee)

1988년 : 서울대학교 계산통계학(학사)
1995년 : KAIST 정보통신공학 (석사)
1999년 : KAIST 전산학 (박사)

1988년～1993년: 시스템공학연구소 연구원
1999년 : IBM Watson 연구소 방문연구원
1999년～현재: 한국외국어대학교 컴퓨터공학부 교수
※관심분야 : 분산병렬시스템, 운영체제, 인터넷 서비스