A New Service Platform Unifying Network Control and Its Prototyping

†Masanori Yamazaki, †Yuya Inoue, †Yusuke Hirota, †Kazuhiko Kinoshita, ††Hideki Tode, †Koso Murakami
†Department of Information Networking, Osaka University, JAPAN
††Department of Computer Science and Intelligent Systems,
Osaka Prefecture University, JAPAN
E-mail:†{yamazaki.masanori, inoue.yuya, hirota.yusuke, kazuhiko, murakami}@ist.osaka-u.ac.jp
††tode@cs.osakafu-u.ac.jp

Abstract—**Recently, advanced network services have been provided by assortment of distributed components at low cost. However, network elements are not considered in such a framework. In this paper, we propose a new service platform which unifies componentized network control function. This platform aims to provide advanced services flexibly according to the available network and service resources. To achieve this, we propose a management method for the componentized service/network control function by using distributed databases, and a service generation method based on information from these databases. Finally, the performance and feasibility of the platform are evaluated by simulation experiments and prototyping.**

I. Introduction

In conventional networks, stereotypical services are provided via fixed routes, since service control function and network control function work independently. In addition, QoS guarantees for these services are still based on best effort manner, and their stable guarantees have not been achieved. Therefore, reconstruction of network control infrastructure for stable QoS guarantees is strongly desired. Hence, some projects concerning the NWGN (New Generation Network) architecture which takes the place of the IP network are proceeding today [1]-[3].

Some distributed object technologies such as Web service working on a common platform are spotlighted. With these technologies, distributed Web service makes it possible for users belonging different platform to use the distributed objects via the Internet. Moreover, service providers can provide a new service using exiting services as components. In addition, by reusing the existing components, they can reduce the time and the cost to create a new service.

In addition, NGSON (Next Generation Service Overlay Network) [4]-[6] has been proposed as a framework to integrate objects and services provided at different architecture by using overlay networks constructed on the IP network. NGSON aims to bridge the service layer and network layer to address the accommodation of highly integrated services. In order to achieve this, NGSON provides some functions for service control layer such as “service routing” and “service composition” and for network control layer such as “QoS control” and “content delivery”.

In network services using distributed objects which are componentized services, overload or failure at a particular service component may cause significant degradation of service qualities. Therefore, we have proposed a service platform which has fault tolerance and can provide stable service qualities [7][8]. In this platform, distributed components are controlled autonomously by using information which is exchanged on overlay networks. These technologies for service control can provide advanced services, but these technologies do not include network control function for QoS guarantees. To the contrary, there are many existing researches about QoS guarantees on IP networks.

In a conventional network, traffic bandwidth-guaranteed systems such as IntServ, DiffServ using RSVP (Resource Reservation Protocol) [9][10] and so on have been proposed. However, IntServ needs to keep reservation status for each traffic at relay nodes, so it has a scalability problem. On the other hand, in DiffServ flow is classified into some groups and guaranteed QoS for each group, so it is difficult to guarantee QoS of each flow strictly. In recent years, the network virtualization approach have been proposed due to spread of cloud computing [11], but it cannot overlook wide area network. These existing studies for QoS guarantees have not been put into practical use widely yet due to scalability issue.

In this paper, we propose a new platform which can provide advanced services flexibly according to the available network and service resources.

II. Service Platform Unifying Network Control

A. Service model

First of all, Fig. 1 shows service model of the proposed service platform unified with network control.

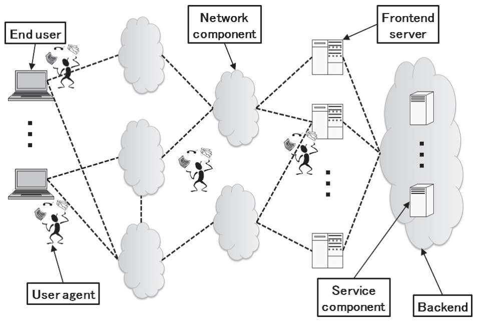

Fig. 1. Service model

We call a componentized service control function a "service component" and a componentized network control function a "network component", respectively. A service component exists on backend, and provides a service for an end user's request through a frontend server. Each frontend server registers services which combine one or more service components for providing to end users.

There are more than one backend servers at backend, and some service components are running on each backend server. It is necessary to keep the stable service quality by managing and controlling these many service components. Therefore, we have proposed a management method for network services based on distributed service components [7][8]. By applying the method, the network services achieve to have fault tolerance and to keep the stable service quality.

The platform componentizes network control included within the scale that can guarantee QoS. We call a componentized network control function "network component". A network component controls some network elements included in particular area like AS (Autonomous System). It is assumed that the network component can guarantee QoS inside the area. Each network component manages some network resources, and guarantees QoS which requested from end users while requested period, and then charges prescribed amount of money to users. But when sufficient link capacity to guarantee requested QoS is not remained, CAC (Call Admission Control) is invoked, which does not accept requests more. End users and frontend servers communicate with each other by using one or more network components.

In this platform, an end user describes his/her request as a form of agent, and then sends out it to the platform. We call this agent as a "user agent".

There exists each user's user agents and each user agent is prescribed both service elements such as the contents ID, service time and so on, and network elements such as bandwidth, communication delay and so on at once.

A user agent sent from an end user generates clones of itself if necessary, and searches for the combination of a frontend server providing the contents meeting the user's request and network components between the frontend server and the end user. After that, our platform provides this combination as a service. Here, the sum of fee for the contents provider and the network provider is charged to the end user who has requested a service.

By the grace of this platform, end users can flexibly customize an advanced service, and receive it with stable QoS. In the following subsections, elemental technologies for the service component and network component are described.

B. Service component management

It is assumed that one or more services combining the distributed service components are provided. Information about the services which each frontend server provides is managed by using databases. We call these databases "service maps". By searching a nearby service map, each user agent can find the service which meets user's demand with faster processing time than by referring to frontend servers one after another.

C. Network component management

It is assumed that a network component uses different network resources in itself depending on which network components connect with. For this reason, the QoS such as bandwidth, communication delay, and so on which can be provided are different for each combination of network components. This information is managed by using distributed databases. We call these databases "network maps".

Each network map manages the information of some close network components, and is connected with each other reflecting the physical network distance.

From a network map, connections which include neighbor network components and frontend servers, QoS which can be guaranteed and charges to use are informed. In addition, the number of access availability similar to service map is managed.

III. Service generation method

In this section, we propose a method to generate a service provided to a user by using the information from service map and network map. Our platform generates a service by searching for the service provided at a frontend server and network components which meet users demand and combining them. The proposed method aims to generate services which meet user's demand considering the effective use of the server resources of the frontend server and the network resources of network components.

Path selection for QoS routing is formulated as a MCP (Multi-Constrained Path) problem. If all constraints can be ignored, Dijkstra's algorithm finds the optimal path in deterministic polynomial time, and it is scalable. However, the MCP problem is NP-complete [12]. Therefore, it is impossible to generate a service within real time, but many approximate or heuristic algorithms have been proposed. Some of them address an optimal version of MCP problem, which is known as the MCOP (Multi-Constrained Optimal-Path) problem. Anyway, in the environment that the position of the frontend server is not fixed, only those algorithms cannot generate services which meet user's requirements.

In the proposed method, multiple candidates of the frontend server are selected from service map, and then for the each candidate a combination of network components is searched from network map, and finally optimal one is selected to provide to an end user. At the phase of searching for the combination of network components, Enhanced Fallback+ [13], which is one of the solution of the MCOP problem is applied.

Specific procedures are as follows.

1) A user agent accesses to nearby service map and gets the m number of frontend server IDs in the ascending order of the value $1/A_{FE_\alpha}$. Note that these frontend servers need to meet user's requirements. Here, let A_{FE_α} denote the number of available accesses of the frontend server α.
2) The user agent also accesses to nearby network map and gets the frontend server IDs within n hops from the end user who has sent the user agent.

3) The common frontend servers chosen in Step 1. and Step 2. are selected as candidates.
4) For each candidate, a combination of network components which has the smallest sum of the value of $1/A_{NW_i}$ is searched based on information from the network map according to Enhanced Fallback+ algorithm. Note that the combination finally needs to meet user's requirements. Here, let A_{NW_i} denotes the number of available accesses of network component i.
5) From the candidates in Step 4., what the value of W of Eq. 1 is smallest is selected and generated as the service for the end user who sends the user agent.

$$W = \frac{1}{A_{FE_\alpha}} \times c + \sum_i \frac{1}{A_{NW_i}} \quad (1)$$

Here, c is a constant which is the weight for the reciprocal of the availability of a frontend server.

This method aims to achieve the effective use of the entire resources of the platform by minimizing the weighted sum of the reciprocal number of access availability shown by Eq. 1.

IV. Prototype implementation

A. Implementation environment

The prototype system was implemented with eight machines. Four machines are for the network component, two for the frontend server, one for network map, and one for end user. Service map is included one of frontend servers. A platform program runs on each machine, and a user agent is exchanged on the program. OS of all machines is Debian 6.0.3 32bit and the version of kernel is 2.6.32-5-686. CPU and RAM are machine-dependent. User/Frontend Server machines have Intel Core2 Duo 3.06GHz CPU and 4GB RAM. Network MAP/Network Component machines have Intel Core2 Duo 2.93GHz CPU and 2GB RAM. Network of platform is constructed by 10/100Mbps Ethernet.

Figure 2 shows the network structure. To explain clearly, machines for end user, network components, and frontend servers are referred by using their IDs. ID of the machines for end user is u_1, for network components is n_1, n_2, n_3, n_4 and for frontend servers is f_1, f_2, respectively. There are two kinds of network links in the environment. One is to manage the platform, and the other is to provide service. In the figure, the solid lines indicate management network links and the dotted lines are service providing links. All machines have their IP address belonging to the same sub-network 172.16.1.0/24 for management network. They are connected with switching hubs. In addition, the machines of end user and frontend servers have another one IP address which belongs to the same sub-network 10.0.0.0/24 for service providing network. DNS server and NTP server for all computers in the network are running on the machine of network map, and all commands for managements and experiments are issued by this machine.

B. Implementation for proposed platform

Our software for this prototype was written in Ruby 1.8.7 p-302. The platform is listening to the request from an end user with WEBrick [15] as a web server. Since the web server accepts the request only from localhost, an

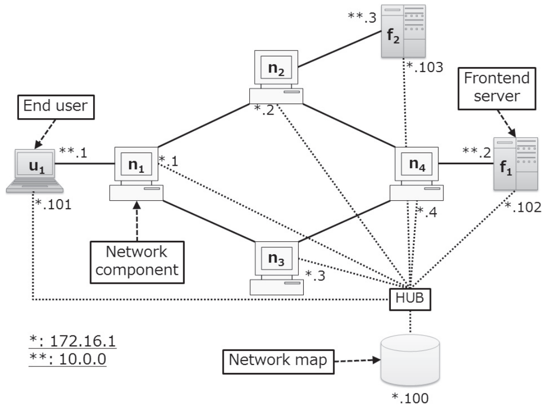

Fig. 2. Network structure

end user has to make HTTP GET request to URI like `http://localhost/service?contents_id=1&service_quality=20&bandwidth=30`. After the "?", users requirements are described with the form of key-value. When the request is accepted, a user agent moves within platform according to the service generation method. The user agent on the platform is implemented as an exchanging serialized Ruby object via TCP socket.

C. Implementation for network map/service map

For the implementation of network map and service map, Apache CouchDB 1.1.1 [16] was applied. CouchDB is a document-oriented database. CouchDB also can be accessed from any environments that allow HTTP requests, and the result of the request is returned as JSON [17] formatted data. In the prototype, CouchDB is installed on two machines, which are network map machine and one of the frontend servers. Any hosts which belong to the sub-network 172.16.1.0/24 can access to the databases.

On network map, documents for each network component are registered, and each document has fields which include the information of availability and QoS for every couple of neighbor network components. Similarly, on service map, documents for each frontend server are registered, and each document has fields which include the information of availability and service qualities for every content.

D. Implementation for network component

When providing a service, path setting of inter/intra network component is realized by OpenFlow technology [18][19]. OpenFlow is an open standard to provide software defined network. OpenFlow could serve as a useful campus component in the proposed large-scale testbeds like GENI.

By using OpenFlow, path setting for each flow can be achieved according to a user agent. However, since it is not realistic to prepare many OpenFlow switches for the prototype, virtualized OpenFlow switches are emulated in the implementation of network component. A machine in which emulates some virtual switch is regarded as one network component. To build such a virtual environment using OpenFlow, trema [20] is utilized. Trema is the OpenFlow programming framework.

In addition, network component management information is stored on SQLite3 [21] database. It is file-based relational database implementation for easy use. It does not need user authentication or other annoying duties but offers advantages of relational database enough. To use it flexibly with Ruby, Sequel [22] is used as Object Relation Mapper. Each network component periodically updates network map using information stored on the SQLite3 database.

E. Evaluation results for prototype

The time from making the request to starting the service was measured with dividing into several processes. The processes are as follows.

1) Agent sending time: the time from an end user making the request to the platform starting to search for a service
2) Service map accessing time: the time that the user agent is accessing to a service map and is processing Step 1.
3) Network map accessing time: the time that the user agent is accessing to network map and is processing Step 2., 3. and 4.
4) Path setup time: the sum of the time that the user agent is sending reservation requests to network components that is determined to use and the time to complete the reservation actually
5) User waiting time: the time to start the service for the end user

Table I shows the result of 100 independent requests. Each request was made every 22 seconds from u_1 in Fig.2, and the service time of these requests was 10 seconds. Additionally, all network maps and frontend servers were updated every 10 seconds. Thus, each request never conflicts, services are provided through the same path . In this situation, the path is $f_1 \rightarrow n_4 \rightarrow n_3 \rightarrow n_1 \rightarrow u_1$.

TABLE I
RESULT OF EXPERIMENT

	average (ms)	standard deviation
(1) Agent sending time	3.94	5.65
(2) Service map accessing time	166.64	5.83
(3) Network map accessing time	192.90	14.54
(4) Path setup time	50.49	16.06
(5) User waiting time	406.26	22.31

The difference of service map accessing time and network map accessing time can be considered as the time calculating Enhanced Fallback+. The computation time of Enhanced Fallback+ depends on the topology scale but [13] refers that it is possible to scale up enough. In addition, the path setup time depends on the number of network component hops. The number of the hop can be controlled by the parameter n.

Next, we also raised 100 independent requests every 1 second from u_1, and the service time of these requests was 10 seconds. In this situation, each request must conflict. Soon after the start of the experiment, the service is provided thorough the same path as the previous experiment, $f_1 \rightarrow n_4 \rightarrow n_3 \rightarrow n_1 \rightarrow u_1$. But, since the access availability of n_3 in connecting to n_1 and n_4 was set at 20 and network maps and service maps were not updated, another path would be constructed after the 21st request. In the experiment, services are provided the path $f_2 \rightarrow n_2 \rightarrow n_1 \rightarrow u_1$ after the 21st request.

Moreover, we raised 100 independent requests every 0.1 second from u_1, and the service time of these requests was 60 seconds. In this situation, other requests were generated during the process of another request. In the experiment, services were similarly provided through the path $f_1 \rightarrow n_4 \rightarrow n_3 \rightarrow n_1 \rightarrow u_1$ at first, and the path was switched to $f_2 \rightarrow n_2 \rightarrow n_1 \rightarrow u_1$ after the 21st request. Although more than four requests would be processed at the same time, each request was processed as well as table I.

As the result of these experiments, we confirmed that the proposed platform worked as expected. In addition, the proposed platform can start the service in a practical time, and can be considered to be scalable.

V. CONCLUSIONS

In this paper, we implemented a prototype of our proposed platform which unifies componentized service control and network control, and evaluated its performance in real environment.

In future work, we will evaluate our proposed platform in more complex situation, and implement the architecture which componentizes services in the backend.

ACKNOWLEDGMENT

This research is partly supported by the National Institute of Information and Communications Technology (NICT).

REFERENCES

[1] The Global Environment for Network Innovations (GENI) , http://www.geni.net/.
[2] Seventh Framework Programme (FP7) , http://cordis.europa.eu/fp7/.
[3] "AKARI" Architecture Design Project for New Generation Network, http://akari-project.nict.go.jp/eng/index2.htm.
[4] IEEE NGSON WG, http://grouper.ieee.org/groups/ngson/
[5] IEEE P1903, "Draft White Paper for Next Generation Service Overlay Network," NGSON WG, May 2008.
[6] S. Lee, and S. Kang, "NGSON: Features, State of the Art, and Realization," IEEE Commun. Mag., Vol. 50, issue 1, pp. 54–61, Jan. 2012.
[7] W. Miyazaki, et. al., "An Efficient Failure Recovery Scheme for Next Generation Network Services based on Distributed Components," IEEE APNOMS 2008, Technical Session 2, Oct. 2008.
[8] M. Yamazaki, et. al., "Sustainable QoS Provisioning Platform for Network Services based on Flexible Combination of Distributed Components" 12th IFIP/IEEE International Symposium on Integrated Management (IM 2011), May 2011.
[9] M. Karsten, "Collected experience from implementing RSVP, " IEEE/ACM Trans. Networking, Vol. 14, No. 4, pp. 767–778, Aug. 2006.
[10] A. Ponnappan, et. al., "A policy based QoS management system for the IntServ/DiffServ based Internet, " 3rd International Workshop on Policies for Distributed Systems and Networks, pp. 159–168, 2002.
[11] X. Cheng, et. al., "Virtual network embedding through topology-aware node ranking," ACM SIGCOMM Computer Communication Review, vol. 41, pp. 38–47, Apr. 2011.
[12] J. M. Jaffe, "Algorithms for Finding Paths with Multiple Constraints," Networks, vol. 14, pp. 95–116, 1984.
[13] K. Kinoshita, et. al., "Enhanced Fallback+: An Efficient Multiconstraint Path Selection Algorithm for QoS Routing," IEICE Transactions on Communications, Vol. E87-B, No.9, pp. 2708–2718, Sep. 2004.
[14] D. J. Watts, and S. H. Strogatz, "Collective Dynamics of 'Small-World' Networks," Nature 393, pp. 440–442, June 1998.
[15] WEBrick, http://www.ruby-lang.org/ja/man/html/WEBrick_HTTPServer.html.
[16] The Apache CouchDB Project, http://couchdb.apache.org/.
[17] JSON, http://www.json.org/.
[18] Open Networking Foundation, https://www.opennetworking.org/index.php.
[19] N. McKeown, et. al., "OpenFlow: Enabling Innovation in Campus Networks," ACM SIGCOMM Computer Communication Review, Vol. 38, issue 2, Apr. 2008 .
[20] Trema Full-Stack OpenFlow Framework for Ruby/C, http://trema.github.com/trema/.
[21] SQLite3, http://www.sqlite.org/.
[22] Sequel: The Database Toolkit for Ruby, http://sequel.rubyforge.org/.