Service-Controlled Networking

Koji ZETTSU

Recent advances of IoT technologies has witnessed the dynamic changes of network from conventional data transportation to a collaboration filed of information services provided by information objects, devices and humans. We first propose a novel concept named Service-Controlled Networking (SCN), a network middleware which is capable to dynamically process data fusion at in-network level over a programmable network like Software Defined Networking. In this paper, we introduce our recent research effort and elaborate how we try to overcome these challenges with a large scale testbed environment. We then verify the usefulness by extensive experiment evaluation. Last, we discuss the future direction of SCN.

1 Introduction

As exemplified by the term IoT (Internet of Things), nearly everything, let alone information devices, and every person in modern society is connected to networks for exchanging information. Networks do not represent simple data transmission paths for data transfer, but assume the role of a platform in which various smart services gather, analyze and distribute information. As a result, the need for new network technologies that allow smooth and efficient connection among the various "information services," quickly responding to the requests for information from applications, is growing. These technologies should serve as a vehicle to connect a variety of objects—information devices, things and people—for transmitting, gathering and processing information, with focused consideration given to effective utilization of limited network resources. In smart cities, for example, the network system at normal times provide smart services based on the information gathered from various sensor networks (environment, traffic, etc.). In a time of emergency such as a disaster, the system is expected to be used in the service of collective gathering of damage information for comprehensive analysis, and distributing essential information (evacuation, rescue, etc.) to the general public[1]. Up to the present, those network technologies (typically VPN) capable of separating communication traffic for each application have been used for construction of application specific networks. However, these are not necessarily suitable for on-demand configuration of the networks that have to change information service paths adapting to specific circumstances. The reason for this is the fact that existing technologies need up-front arbitrations for determining network configuration. In recent years, many attempts have been made actively to develop a platform that operates utilizing the Cloud and Software-Defined Networking (SDN)[2][3] in a coordinated fashion. These efforts, however, have focused mainly on technologies to virtualize networks and computational resources, and resource reallocation required to respond correctly to the demands from specific applications is still manually controlled by the administrator. However, configuration changes through manual operation have become increasingly difficult along with the advent of IoT and the explosive increase of information services. New technologies are needed to interpret the linkage demand among the applications correctly for modifying and tuning the network configuration automatically.

To address these demands, we have proposed the concept of Service-Controlled Networking (SCN) and implemented it in some middleware applications of our own development[4]–[7]. SCN represents a technology that enables such network controls as node discovery, path generation, data processing and QoS setting, in dynamic response to demand from applications for new information service linkage. SCN is placed as one of the vehicles to realize the concept propounded in data awareness and service awareness of future networks (ITU-T Y3001[8], an international standard for new generation networks). In this report, the author outlines aspects of SCN including the basic design, implementation on a wide area network testbed, development of application systems, and future perspectives.

2 SCN: basic concept

SCN is an approach to mapping the flow of information that each application exchanges with various information services on a network. Conventionally, networks have been configured manually. With the advent of programmable networks, typically SDN, dynamic configuration of virtual networks by means of software has become a reality. In line with this trend, SCN tries to decide the network configuration most appropriate to secure smooth flow of information among the applications, and automatically configure programmable networks dynamically so that information flow-driven virtual networks are constructed. To attain this goal, SCN consist of the purpose-designed elements shown in Fig. 1. **The Declarative Service Networking (DSN)** is composed of a rule language—used to describe declaratively the information flows among information services—and its interpreter. As shown in the example in Fig. 1, the flow for collecting and processing data from various information services is designed, and, based on this, the application is described using DSN. **The Network Control Protocol Stack (NCPS)** searches the nodes that perform an information service and creates paths connecting them, with the goal of construction and execution of virtual networks that allow the application to run the DSN-described information flow. An instance of NCPS is provided for each protocol because the way in which a network is constructed depends on the network protocol.

The Declarative Networking technology[9] provides the basis for SCN. Declarative Networking has its origin in the declarative rules used in graph-structured databases and the recursive query language Datalog[10]. Network Datalog (NDlog[11]) is an extension of Datalog to include functions to describe network data flow. Declarative rules and recursive query language—suited for use to express the relationship and correlation constraints between the original and derived data—are being actively investigated in the field of deductive databases. In NDlog, the network is abstracted to provide a field for data exchange, in which the calculation rules—transition rules form one node state (data collection) to another—are described using the network protocol. An example of a protocol-based shortest path description in NDlog is shown in Fig. 2. This example comprises 4 rules, sp1 to sp4. path (Src, Dest, Cost Path) indicates the existence of a path, Cost, connecting from the node Src to the node Dest. The rules sp1 and sp2 express path generating rules: sp1 uses existing inter-node link (Src, Dest, Cost) as the path as-is, and sp2 creates a new path by combining links adjacent to a path.

Sp3 indicates the rule to identify the minimum cost required to route Src to Dest, and sp4 indicates the rule to connect paths from Src to Dest at the minimum cost. The lower part of Fig. 2 shows the way in which the shortest path from each node is calculated through recursive application of these rules. The figure shows a path tuple for each node that represents a series of paths connecting the node to another. The figure "Initially" shows the initial state, in which the links to neighboring nodes are shown.

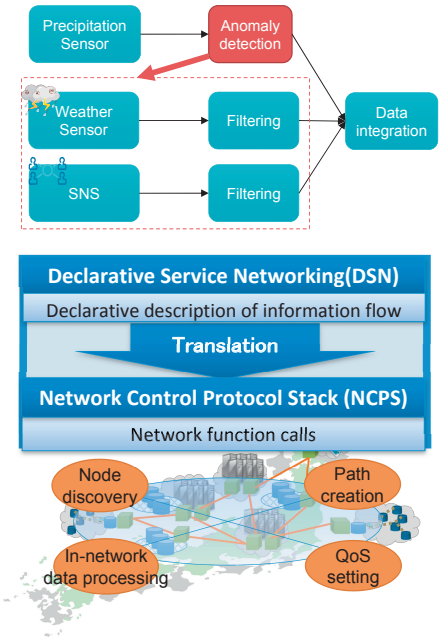

Fig. 1 Basic concept of SCN

```
sp1-sd pathDst(@Dest,Src,Path,Cost) :- magicSrc(@Src),
       link(@Src,Dest,Cost), Path=f_init(Src,Dst).
sp2-sd pathDst(@Dst,Src,Path,Cost) :-
       pathDst(@Nxt,Src,Path1,Cost1),link(@Nxt,Dest,Cost2),
       Cost=Cost1+Cost2, Path=f_concatPath(Path1,Dest).
sp3-sd spCost(@Dest,Src,min<Cost>) :- magicDst(@Dest),
       pathDst(@Dest,Src,Path,Cost).
sp4-sd shortestPath(@Dest,Src,Path,Cost) :-
       spCost(@Dest,Src,Cost), pathDst(@Dest,Src,Path,Cost).
Query shortestPath(@Src,Dest,Path,Cost).
```

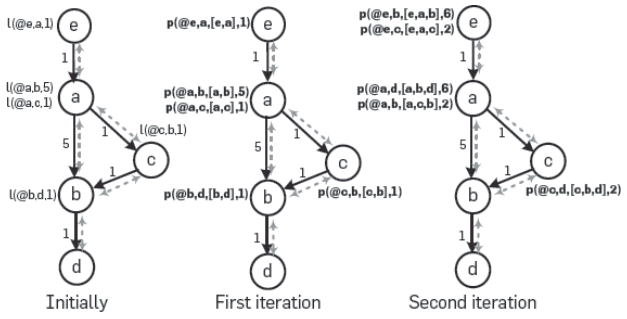

Fig. 2 Examples of shortest path description using Declarative Networking (NDlog) (from Reference [9])

Then, the first iteration (application of the rule sp1) generates a path tuple corresponding to one hop. In the second iteration, application of the rule sp2 creates a path tuple corresponding to two hops. Recursive application of these steps generates the set of path information, and then, subsequent application of rules, sp3 and sp4, calculates the shortest path (shortestPath) form one node to another. Overlog[12] is an extension language of NDlog for additional capability of describing overlay networks: it can create virtual nodes and logical links over a physical network. Extended capabilities of Overlog over NDlog include such functions as message delivery, receipt acknowledgement (ACK), failure detection, and timeout, enabling Overlog to find applications in P2P[13] and content delivery networks.

SCN attempts to extend the Declarative Network technology described above, so that it can configure overlay networks dynamically in tune with the flow of data exchange between information services. Figure 3 shows the schematic overview of an SCN-based network configuration. In SCN, the flow of information that comes and goes via logical links among the information services running on the nodes is described using DSN language, which is an extension of Overlog. When the DSN descriptions are put into a run, information services are mapped onto the node by NCPS, followed by setting paths between the nodes for data transfer among the information services. Then, the data processing specified to the information flow are executed on one of the nodes along the route (in-network data processing). During this process, reconfiguration of paths and in-network processing is continuously an option to avoid congestion and delay of data transfer. It can be triggered in response to the ever-changing situations in the network environment—through careful monitoring of traffic and node loading.

These capabilities provided by SCN enable application developers to ensure network resources elastically for continuous and stable operation of the application irrespective of the changes that may occur in the information flow between the information services. These characteristics free the developer from worries about acquiring network resources preemptively in preparation for the arrival of peak loading. They also provide beneficial effect to the network providers: dynamic and automatic reallocation of resources in response to real-world situations enables providing space for a larger number of applications within the scope of limited network resources, leading to a higher level of utilization ratio than in the conventional networks that require contract-based allocation of network resources.

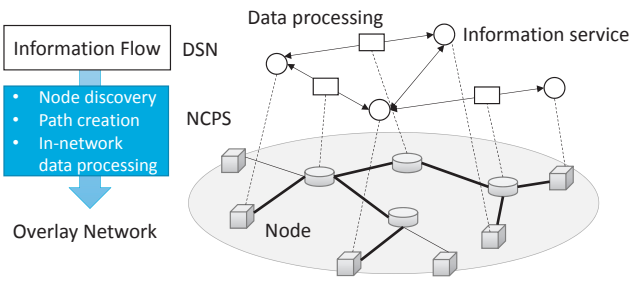

Fig. 3 Dynamic configuration of a network using SCN

3 Related research

As one of the technologies that can provide software control of networks, SDN has received attention in recent years. SDN allows programs to configure network path settings using APIs. For this purpose, various domain-specific programming languages have been proposed aiming at upgrading efficiency in network control. These programming languages, such as Frenetic[14], NetCore[15], NICE[16], and Nettle[17] aim at developing network functions—routing, topology discovery, etc.—on OpenFlow, a typical implementation of SDN. One of the main objectives of these languages is to support the development of programs to modify path settings (flow table) in tune with OpenFlow's switch status, wherein focus is placed on the functions to run simulations and hunt bugs using the programs that run within a closed OpenFlow network. On the other hand, Procera[18] is designed to describe the rules for flow control by responding to the events taking place in other nodes than OpenFlow switches: e.g. user authentication, time and day, utilization ratio of the band, and server load. This mechanism enables the language to control the network in compliance with the application's administration policies. As described above, there are many methods to control SDN at various levels. Among them, the SCN approach is characterized by its ability to describe the flows of information for each application using the DSN language (a Declarative Networking-based language) and automatically configure the network that runs those flows of information on an SDN implementation (e.g. OpenFlow). It is also capable of automatic reconfiguration during the application's run-time if any congestion or delay takes place in the network. With such capabilities, SCN can be considered to provide strengthened coordination between applications and network controls compared to

conventional SDN control technologies.

Different from conventional terminal ID (e.g. IP address) based technologies, Information-Centric Networking (ICN[19]) (or Content-Centric Networking[20]) uses the IDs of information contents (typically the content name) as the key to control networks. In ICN, the content ID and its network address are registered, and ICN resolves the name for data transfer in response to a "push"/"get" of the content ID from the application. This approach enables the applications to exchange content IDs without the need to take heed of the network configuration, as well as the network administrators to rearrange network resources more elastically. SCN has a mechanism, as with other methods, to separate information flow from the network configuration by means of DSN and NCPS, which enables dynamic discovery of correspondence between information services and nodes. In addition, SCN steps up its control capability into distributed processing of contents, which involves dynamic path settings among information services and intra-network data processing based on the information flow described in DSN.

4 SCN: Design and implementation

SCN has been implemented and introduced as middleware that runs on the nodes of the network targeted for dynamic configuration. Coordinated operation of these actually executes SCN functions. The information services targeted for coordination through SCN are those provided by existing systems and Web services. They are wrapped using a common interface before registering to SCN with service metadata. The data exchanged among these services is also transformed into the common format (AMON de facto standard[21]) by the wrapper. The next section describes the process of actualizing SCN functions in the execution environment.

4.1 Declarative Service Networking

DSN descriptions are implemented based on Bloom[22][23], which in turn is an implementation of the Overlog language. Bloom is a domain specific language embedded in the code of the general-purpose programming language Ruby, for the objective of describing data flows using production rules. For use in DSN, Bloom is further extended to allow construction of information flows that links services more efficiently (functions shown in Table 1 are implemented).

An example of descriptions in DSN (an excerpt) is shown in Fig. 4. This program monitors rainfall data, and if higher than specified precipitation (deluging rain) is detected, it gathers additional sensing data from the surrounding area indicative of the extent of the damage caused by the rain—e.g. meteorological, traffic data, and social media reporting. Major rules composing the program are as follows:

① *state* section declares the information services used in DSN. *@xxxxx* represents the service name, and the discover function performs service discovery.

② *scratch_xxxxx* represents an output from or input to the service, and *channel_xxxxx* represents the channel used for inter-service data transfer. "<~" designates a data transmission from the right member to the left. For example, "*scratch_evwh <~channel_panda*" indicates a data transfer to *scratch_evwh* (I/O of *@evwh* service) through *channel_panda* (data transfer channel connecting *@panda* service and *@evwh* service). The data sent out from *scratch_panda* is filtered when it passes through *channel_panda* by the filter function with filtering conditions given as an argument.

③ *event_xxxxx* represents an event that occurs when the transferred data satisfies certain conditions, and the trigger function generates an event if it detects

Table 1 DSN functions

Name	Synopsis	Example
discover	Service discovery	@jma_rainfall: discover (category=sensor, type=rain)
filter	Data filtering	channel_datastore <~ filter (scratch_panda, average_rainfall > 25.0)
cull_time, cull_space	Data thinning at specified time/position intervals	channel_datastore <~ cull_time (scratch_panda, 1, 2, time (time, "2015 /01 /01 T00:00:00", "2015/03/20 T23:59:59", 30, "second"))
aggregate	Aggregation of multiple data into one	channel_name1 <~ aggregate (scratch_panda, aggregate_data, time (time, "2015/01/01 T00:00:00 ", "2015/03/20 T23:59:59", 15, "second"), space (latitude, longitude, -12.34, -5.67, 34.56, 78.9, 0.1, 0.3))
trigger	Event detection	event_heavyrain <+ trigger (channel_datastore, 30, count > 130, average_rainfall > 25.0)

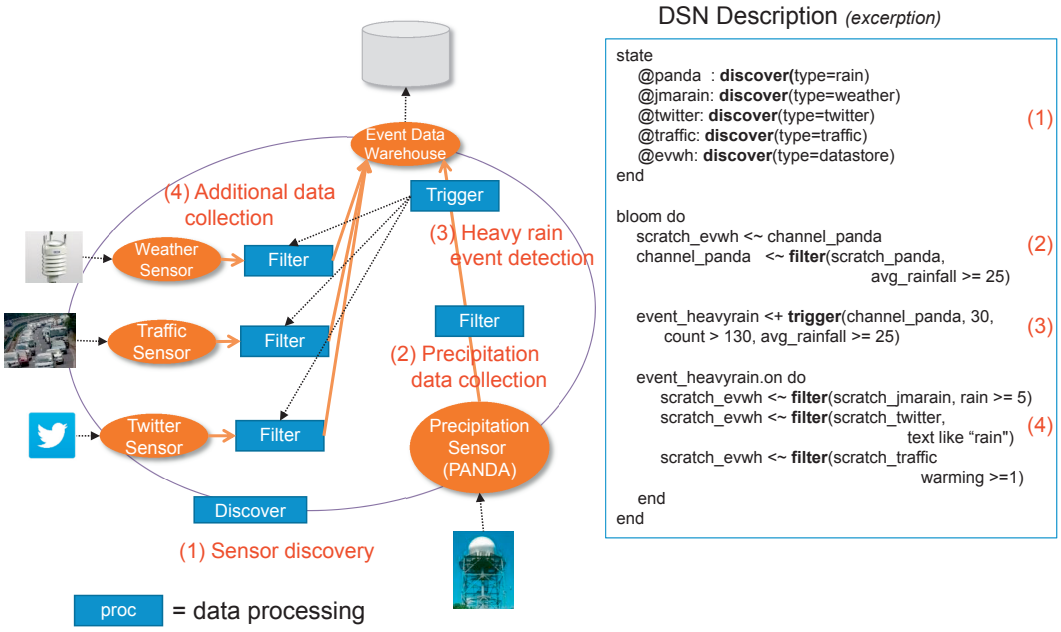

Fig. 4 An example of information flow descriptions using DSN

the condition indicated by its argument.

④ *event_xxxxxon do* section is run only when the corresponding event takes place.

The DSN description interpreter performs the following tasks: (1) execution of the rules specified by the DSN descriptions, (2) data transmission/reception between the services, and (3) invokes NCPS functions.

The DSN interpreter is implemented based on the distributed data flow execution engine of the Bloom language (Bud), and the data transmission/reception between the rule execution and a service, described above, is carried out by the Bud. The DSN functions (extension of Bloom) are implemented as Ruby subroutines within which Bloom code is embedded, and are used to invoke NCPS functions via external libraries. For example, the *discover* function uses the argument to create a query for service search, which invokes the node discovery function provided by NCPS. Other data processing functions also invoke, in similar fashion, NCPSs corresponding to intra-network data processing functions.

4.2 Network Control Protocol Stack

To achieve DSN-based dynamic configuration of overlay networks, NCPS has implemented three functions: node discovery, path creation and intra-network data processing. In the node discovery function, NCPS searches services that meet specified conditions, followed by identification of execution nodes. Namely, this function consists of two processes: service search and node lookup. The service search performs search by query in reference to the given metadata that describes service attributes—e.g. type and name of the service, input/output data format, and others. For example, if the given query contains "*type=rain*," a search is made to locate a service that provides precipitation data.

The node lookup function registers a combination of service ID and node ID (normally the IP address) to NCPS, and creates a hash table using the service ID as the key. In node discovery, the service search is performed to find service metadata that fits the given query, followed by a node lookup process to locate the node using the service ID obtained from the metadata. By combining two dissimilar processes as described above, a higher level of flexibility is attained in service search: applications can issue various queries, and node lookups can be accelerated even in the case of frequent reallocation of services to the nodes.

In addition, by adopting distributed hash-based node lookup[24], enhanced scalability can be realized as the network is scaled up.

The path creation function discovers the shortest path routing the nodes on which services are executed. In this route discovery process, the network is represented as a weighted directional graph, in which the weight is determined based on traffic and node loading. Even after a path is created, monitoring on the traffic amount between the service nodes continues: if the traffic amount fails to fulfill the throughput conditions placed on the DSN channel, a

trigger is generated inside NCPS for path recalculation. Because this approach requires path setting and traffic statistics acquisition for each overlay network, the process has to be performed for each protocol running on the target network. In the case of OpenFlow, the path information is written to the flow table of OpenFlow controller, and the flow IDs enable unified management of OpenFlow switch connections under its control. The OpenFlow controller monitors and tallies the OpenFlow switch traffics for each flow ID for calculating statistics using the *Stats Request/Replay* function (standard equipment of OpenFlow).

To achieve efficient data exchange abiding by the information flow specified by DSN while using the path created as described above, the network has provisions for executing DSN data processing functions (filter, cull, aggregate, trigger) on the nodes along the route inside the network. NCPS monitors each node's load information, and determines a node on the route on which the processing should be carried out.

5 Application of SCN

To exemplify the effect of SCN on network control, an experiment was carried out to compare data exchange performance in two cases: one with SCN, and one without it. Several applications, each known to exhibit different throughput for different services, were used in the experiment. They were run in sequence with time for observing how each application throughput changes. An experimental OpenFlow network was constructed on the network testbed StarBED[25], on which 6 OpenFlow switches were connected in full mesh configuration using 10 Mbps lines. All of the switches were placed under control of an OpenFlow controller.

Two nodes (service nodes), on which the information service runs, were connected to each OpenFlow switch. Both OpenFlow switch and OpenFlow controller were software implementations: the former executes Open vSwitch[26] and the latter NOX[27] software on a server running Linux OS (Ubuntu). Specifications for each node are shown in Table 2.

Figure 5 shows the results of the experiment. The vertical axis represents throughput, and the horizontal axis indicates elapse of time. Each polygonal line corresponds to throughput changes of each application: the longer the horizontal segments, the better the stability of data transmission/reception under the required throughput conditions. The system with SCN in operation, the applications—each executed in sequence—maintained stable performance even though traffic amount changes from one throughput level to another. In contrast, the system without SCN showed heavy fluctuations in throughput starting from a certain point in time, indicating that the application fell into an unstable state. The results can be explained by the SCN's working wherein it switches the path dynamically to avoid network congestion, and to maintain each application's throughput among the service nodes (i.e. end-to-end throughput), clearly indicating the beneficial

Table 2 Configuration of experiment node

	CPU	Memory	OS
OpenFlow controller	Xeon 2.668 GHz × 4	2,048 Mbytes	Ubuntu 11.10
OpenFlow switch	Xeon 2.668 GHz × 1	8,192 Mbytes	
Service node	Xeon 2.668 GHz × 1	8,192 Mbytes	

Fig. 5 Results of throughput comparison experiment

effect for sustaining stability in many application systems than the system without SCN running on the network under best-effort control.

To take advantage of the effects SCN can provide—i.e. dynamic configuration of networks enables stable and sustainable execution of information flow—a sensing data collection/analysis platform was developed on the sensor network testbed JOSE (Japan-wide Orchestrated Smart/Sensor Environment)[28]. The objective of the platform is to gather heterogeneous sensing data collectively from various information sources (sensors, social media, etc.) to cope with events and emergencies that occur in an unexpected fashion.

JOSE is a testbed to actively utilize observation data gathered from a multitude of sensors deployed in wide area, using distributed computer systems located in many sites each connected with high-speed networks. The testbed consists of a large-scale network (customizable), server facilities and wireless sensor facilities. SCN can operate on JOSE to automatically configure application-specific sensor data gathering networks. Figure 6 shows how a specific application (that shown in Fig. 4) actually operates on the testbed and gathers information. The figure shows two visualized 3D renditions—geometrical space (plane) and time (vertical) axis—of the various data collected by the application: the left with SCN implemented, and the right without it. In the center of both figures, precipitation data when very heavy rainfall (≧25mm/hr) was observed is shown. Along with it, additional data (meteorological, traffic and social media) are displayed on the same frame. The small window on the lower left of Fig. 6 presents a visualized rendition of SCN operational status: it enables real-time monitoring of the network being dynamically and automatically reconfigured by NCPS in response to the events specified by the *trigger* function of DSN—e.g. heavy rain event, delay in data transfer, and others. Comparison of sensor data gathering performance in these two cases—one with SCN and the other without it—indicates that the system without SCN implementation (right figure) tends to collect all the data from around the country independent of the precipitation sensor readings. The system using SCN (left figure), on the other hand, starts intensive data gathering from the affected area when it acknowledges the occurrence of heavy precipitation (signal from DSN functions such as trigger). Thanks to SCN, no susceptible delay in real-time data gathering was observed because SCN ensures elastic reallocation of network resources at the time of the event for maintaining throughput. In contrast, the system without SCN is prone to suffer from network congestion because it gathers data around the country at all times. In addition, once congestion takes place, the system will take longer to recover the original throughput level (see Fig. 5) because it is a best-effort network, often resulting in a large delay in data gathering.

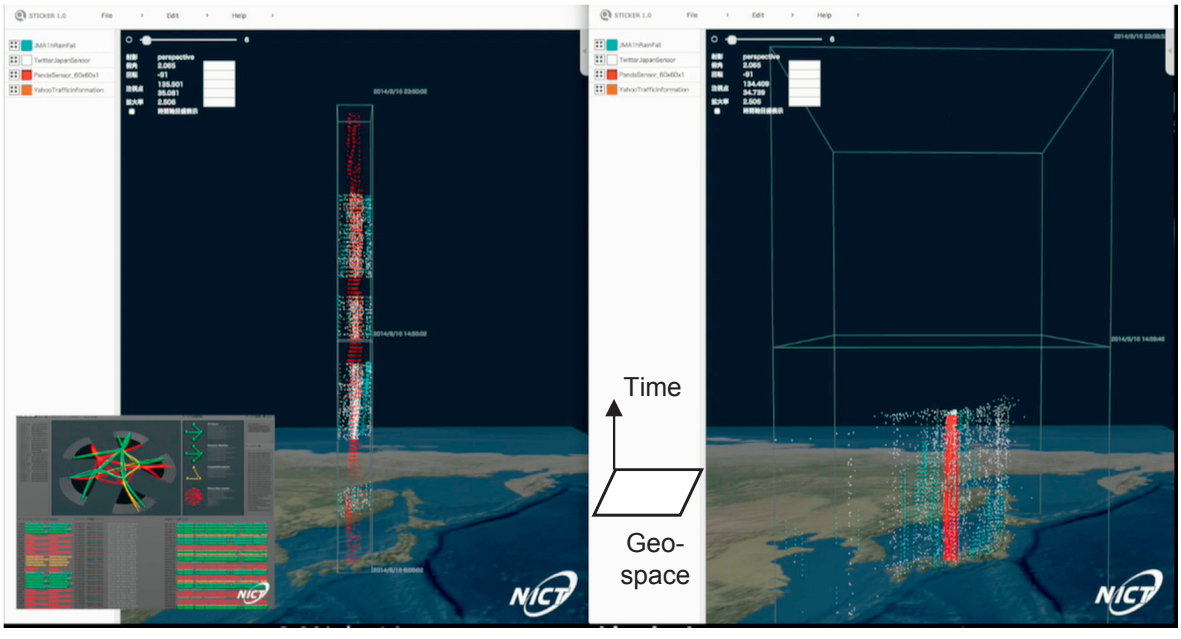

Fig. 6 Performance examples of applications run on the sensing data collection/analysis platform

6 Future perspective

With the development of IoT, strong growth of such applications is expected that provide various types of network links among the information services provided by various agents including objects, people and information devices. In line with this trend, certain types of technologies seem to move to the forefront: typically those, including SCN, dynamically configure networks in response to the changes in information flows among the services. To cope with this trend, SCN needs further upgrade in scalability. Major challenges that SCN should address in the future include: establishment of a common service interface, standardization of DSN descriptions, effective mapping and dynamic control of information flows and network configurations through machine learning and other methods, and, discovery and tracking of services that are running on mobile nodes. SCN-related technologies are under active development for wider applications, in relation to social ICT, in view of utilizing cross-cutting participatory sensing to address environmental issues both at normal times and in emergency. The direction of research, in the future, should be geared toward constructing the framework for effective IoT data utilization.

References

1 Presser M., Vestergaard L., and Ganea S. "Smart City Use Cases and Requirements," D2.1, FP7-SMARTCITIES-2013, available at http://www.ict-citypulse.eu/page/sites/default/files/citypulse_d2.1_requirements_v1.0_0.pdf

2 OpenDaylight, http://www.opendaylight.org/

3 Numhauser, B.-M., et al., "Fog Computing Introduction to a New Cloud Evolution. Escrituras silenciadas" paisaje como historiografía. Spain: University of Alcala. pp.111–126. ISBN 978-84-15595-84-7, 2013.

4 Toyomura T., Kimata T., Kim K.-S., and Zettsu K., "Towards Information Service-Controlled Networking," Proceedings of the 5th International Universal Communication Symposium (IUCS2011), pp.155–163, Oct. 2011.

5 Kimata, T., Toyomura, T., Kim, K.-S., and Zettsu, K.: Dynamic Configuration of Network Flow based on Service-Controlled Networking, IEICE Technical Report SC, Vol.112, No.77, pp.7–12, June 2012.

6 Kimata, T., Sugiura, K., Dong, M., and Zettsu, K.: Service-Controlled Networking: A Dynamic Network Control Method based on Application Structure and User Request, The 6th Forum on Data Engineering and Information Management (DEIM2014), C9-3, March 2014.

7 Dong M., Kimata T., and Zettsu K., "Service-Controlled Networking," Dynamic In-Network Data Fusion for Heterogeneous Sensor Networks, Proceedings of the 33rd IEEE International Symposium on Reliable Distributed Systems Workshops (SRDSW), Nara, Japan, pp.94–99, Oct. 2014.

8 Future networks "Objectives and design goals," ITU-T Y.3001, May, 2011.

9 Loo B. T., et. al.: Declarative Networking, Communications of the ACM, Vol.52 No.11, pp.87–95, Nov. 2009.

10 Ramakrishnan R. and Ullman J.D., "A Survey of Research on Deductive Database Systems. Journal of Logic Programming," Vol.23, No.2, pp.125–149, 1993.

11 Nigam V., Jia L., Wang A., Loo B. T., and Scedrov A., "An Operational Semantics for Network Datalog," Proceedings of the Third International Workshop on Logics, Agents, and Mobility (LAM), July 2010.

12 Loo B.T., Condie T., Hellerstein J.M., Maniatis P., Roscoe T., and Stoica I., "Implementing declarative overlays," Proceedings of ACM Symposium on Operating Systems Principles, 2005.

13 P2, "Declarative Networking System" http://p2.cs.berkeley.edu.

14 Foster N., et. al., "Frenetic: A Network Programming Language," Proceedings of the 16th ACM SIGPLAN international conference on Functional programming (ICFP '11), pp.279–291, 2011.

15 Monsanto C., Foster N., Harrison R., and Walker D., "A Compiler and Run-time System for Network Programming Languages," Proceedings of ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL), pp.217–230, Jan. 2012.

16 Canini M., Venzano D., Peres P., Kostic D, and Rexford J., "A NICE Way to Test OpenFlow Applications," Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation (NSDI'12), pp.127–140, 2012.

17 Voellmy A. and Hudak P., "Nettle: Functional Reactive Programming for OpenFlow Networks," Proceedings of the Workshop on Practical Aspects of Declarative Languages pp.235–249, 2011.

18 Voellmy A., Kim H., and Feamster N., "Procera: A Language for High-Level Reactive Network Control," Proceedings of the First Workshop on Hot Topics in Software Defined Networks (HOTSDN'12), pp.43–48, 2012.

19 Ahlgren B., Dannewitz C., Imbrenda C., Kutscher D.,and Ohlman B., "A Survey of Information-Centric Networking, IEEE Communications Magazine," Vol.50, No.7, pp.26–36, July 2012.

20 Jacobson V., et. al., "Networking Named Content," Proceedings of the 5th International Conference on Emerging Networking Experiments and Technologies (CoNEXT '09), pp.1–12, 2009.

21 AMON data format, http://amee.github.io/AMON/ .

22 BOOM, http://boom.cs.berkeley.edu/

23 Alvaro P., Conway N., Hellerstein J. M., and Marczak W. R., "Consistency Analysis in Bloom: a CALM and Collected Approach," Proceedings of the 5th Conference on Innovative Data Systems Research (CIDR '11), pp.249–260, 2011.

24 Stoica I. et. al., "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," ACM SIGCOMM Computer Communication Review, Vol.31, No.4, pp.149–160, 2001.

25 StarBED, http://starbed.nict.go.jp/

26 Open vSwitch, http://openvswitch.org/, accessed at June 2015.

27 NOX OpenFlow Controller, http://www.noxrepo.org/, accessed at June 2015.

28 Japan-wide Orchestrated Smart/Sensor Environment (JOSE), http://www.nict.go.jp/nrh/nwgn/jose.html .

Koji ZETTSU, Ph.D.

Director of Information Services Platform Laboratory, Universal Communication Research Institute/Research Manager, New Generation Network Laboratory, Network Research Headquarters
Data Engineering, Database System, Information Retrieval