Development of a Reference Implementation of the NSI Connection Service Version 2.0 Standard

Atsuko TAKEFUSA, Tomohiro KUDOH, Ryousei TAKANO
Hidemoto NAKADA, Katsuhiko OHKUBO, Fumihiro OKAZAKI
National Institute of Advanced Industrial Science and Technology (AIST)

Summary

There have been several experiments on dynamic path provisioning over multiple R&E networks provided by heterogeneous management domains, and production services using this feature are being initiated. The OGF NSI working group defines the Connection Service (CS) protocol standard for such a network service. In order to verify the protocol, provider, aggregator, and requester software modules and monitoring tools for the network service are required. We have developed CS v2-based software modules in the GridARS framework. In the CS v2 interoperation experiments, we contributed by providing our reference implementations and monitoring tools. Also, the framework is open-source software, and an application, a provider implementation, and a monitoring tool have been developed by third parties using our framework.

Key words: NSI, Network, Resource Management, Demonstration, Web Services

1 Introduction

Software Defined Networking (SDN) technologies, which dynamically control network switches by using software, are being put to practical use. OpenFlow [1], a typical SDN technology, splits the control function and transfer function from a traditional network switch. The OpenFlow-based architecture consists of a controller software component and multiple OpenFlow-compliant switches.

On Research and Education (R & E) network testbeds, there have been several demonstrations, which dynamically construct and provide an end-to-end "path" over multiple network domains [2, 3, 4, 5]. In order to automatically and easily construct such a path, it is necessary to use a common service interface, which provides the capability to control a path based on each on-demand or advance reservation request. Therefore, the Network Services Interface (NSI) working group [6] in the Open Grid Forum (OGF) defines the Communication Service (CS) protocol, whereby a path network can be provided as a service. The CS protocol defines protocols to request or provide a network service between two Network Service Agents (NSA), such as a requester and a provider, a requester and an aggregator, or an aggregator and a provider. CS enables the requester to reserve, provision, release and terminate a point-to-point connection, that satisfies each requester's requirements, such as bandwidth, latency and VLAN, and the query status of the reserved connection. If a requester requires a path spanning over multiple network domains, a requester can reserve the path by an aggregator via tree-based or chained CS request processes between multiple NSAs.

In order to confirm the feasibility of the CS protocol, we conducted the CS v. 1.0 interoperation experiments between SOAP-based reference implementations in 2011. In 2012 and 2013, the alpha and beta versions of CS were defined and their interoperation experiments were conducted over multiple network domains and testbeds from Asia, Europe, and South and North America. These experiments required various software components including not only a network management system, a provider, an aggregator and a requester, but also graphical or command line interface and a visualization tool for the requester. However, it is difficult for each network domain developer to develop all of these software components because they have to keep up with the CS protocol updates for each version.

We have been developing the GridARS software, that enables us to co-allocate various resources, including network connections, provided by multiple domains [7, 8]. In this study, we developed software modules for a provider, an aggregator and a requester of the NSI CS v. 2.0 (hereafter CS v2) standard, and a monitoring tool, which can visualize the status of connection services based on CS. We contributed the developed software modules to the CS v2 interoperation experiments over eighteen actual networks from Asia, Europe, and South and North America, provided by different organizations. In addition, GridARS is open source software and we are encouraged that third parties have used GridARS to develop for CS-compliant applications or providers, a high-quality video streaming application by using UltraGrid [9], and the SINET [10] CS provider.

2 NSI Connection Service v. 2.0

The OGF NSI working group has been standardizing the Connection Service (CS) protocol as a network service interface, which enables network service providers to provide a path network as a service. We describe the details of the CS protocol after we explain the system architecture the CS protocol assumes.

2.1 NSI architecture

Figure 1 shows the NSI architecture as defined by the NSI working group [11]. The NSI architecture consists of network service Providers, Requesters and Aggregators. They are called Network Service Agents (NSA). Also, each network endpoint in a network managed by each Provider is called a Service Termination Point (STP).

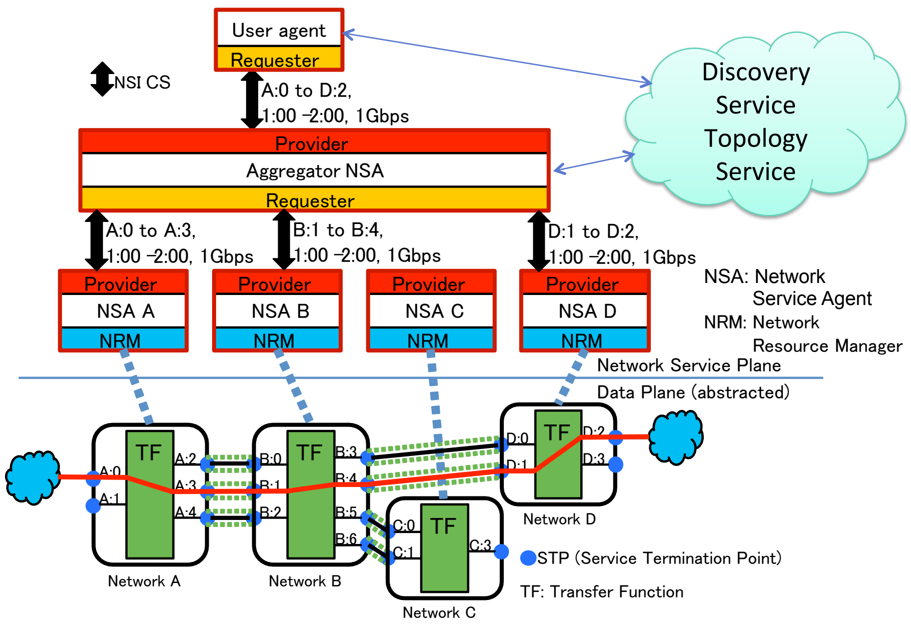


Figure 1: NSI Architecture.

In addition to CS, the NSI working group defines two services: Topology Service and Discovery Service. Topology Service manages the information of multiple domain networks and their adjacency relationships, and the CS providers and STPs of each network. On the other hand, Discovery Service enables the users or aggregators to query and provide details of network services each Network Service Agent (NSA) provides. Based on CS, Topology Service, and Discovery Service, an ultimate Requester can request a performance-assured end-to-end path between different domain STPs to an Aggregator and the Aggregator requests the paths from related Providers, in a hierarchical or chain-based manner. The Requester can reserve, provision, release, terminate and query the requested path whose criteria, such as bandwidth, latency and VLAN, are assured.

Each Provider manages multiple network switches which compose its network, and discloses their network information, such as STPs and a service access point, on a Topology Service server in advance. Based on Topology Service information, a Requester sends a reservation or on-demand request for a path. Here, a Requester can specify the route of the requested path, and so the Requester does not need to specify Explicit Route Objects (EROs). In the latter case, a Requester requests a path to an Aggregator. The Aggregator selects a suitable route for the requested path based on information provided by Topology Service, and sends requests to the related Providers on behalf of the Requester. Each Provider reserves the requested path in its network and control, the requested path following the Requester's operation, based on the CS protocol.

In Figure 1, the Requester (User agent) sends the Aggregator a reservation request for a 1 Gbps path between STP A:0 in Network A and STP D:2 in Network D, from 1:00am to 2:00am. The Aggregator selects the path over Network A, Network B and Network D, and requests paths A:0 to A:3, B:1 to B:4, D:1 to D:2 from the related Providers, respectively. Here, A:3 to B:1 and B:4 to D:1 are physically connected in advance and their relationships are described in topology information provided by Topology Service. The Requester can use the path between A:0 and D:2 after all of the reservation processes have succeeded.

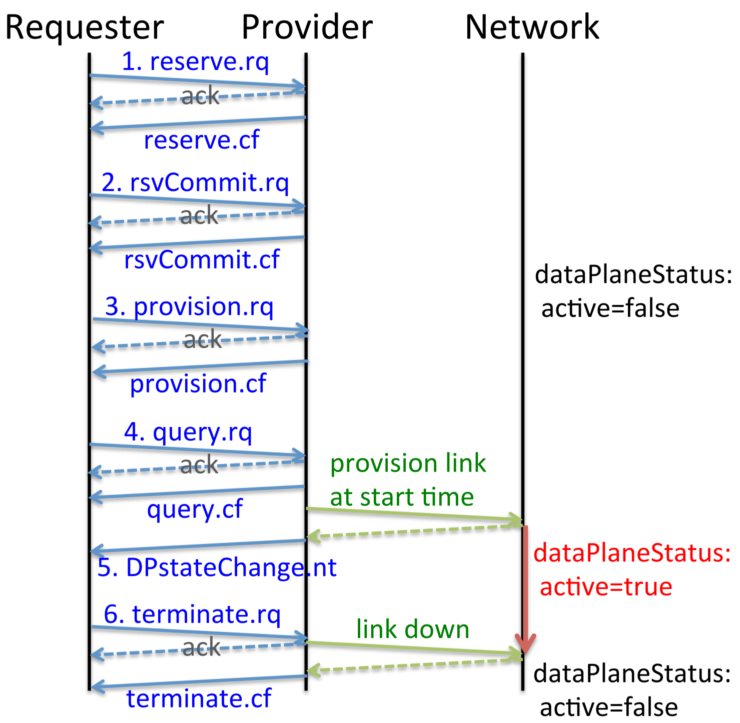


Figure 2: Execution flow based on the CS v. 2.0 protocol.

2.2 Connection Service protocol

The CS protocol defines an interface used to request or provide a network service between two NSAs in an asynchronous manner. The both the Requester and Provider NSAs each have global IP address and they can request or respond by using the Requester or Provider Web services interface. Also, the CS v2 provides a two-phase commit reserve operation, which enables use of a modification request for a path provided by multiple networks.

The Provider interface defines reserve, reserveCommit and reserveAbort, provision, release, terminate, and query operations. The reserve operation can be used as a modification request, too. Each reserved path is identified by a Connection ID issued by a Provider. After the first reserve request, the Requester has to specify the issued Connection ID for each request.

The Requester interface provides xxxConfirmed and xxxFailed operations. Here, xxx represents each operation the Requester sent, such as reserve or reserveCommit. Also, there are several notification messages, the Provider sends to the Requester when some event happen on the reserved path, such as dataPlaneStateChange, messageDeliveryTimeout and errorEvent. dataPlaneStateChange reports on a state change of a data plane, e.g., active or inactive. messageDeliveryTimeout and errorEvent report that a timeout or an error happened.

Figure 2 shows the general execution flow between a Requester and a Provider based on the CS v2 protocol. Operations between Provider and Network indicate control passed to the Provider's network switch. Each message can be described in the following way: The Requester sends a 1. reserve request to the Provider and the Provider first returns an ack to the Requester. If the Provider can prepare the requested path, the Provider sends a reserveConfirmed message to the Requester. Otherwise, the Provider sends a reserveFailed message. Then, the Requester sends 2. reserveCommit in order to confirm the requested reservation. The Provider sends reserveCommitConfirmed after the reservation has been confirmed in the Provider system. The Requester can send any of 3. provision, 4. query and 6. terminate request in the same asynchronous manner. After the reservation start time has passed and a provision

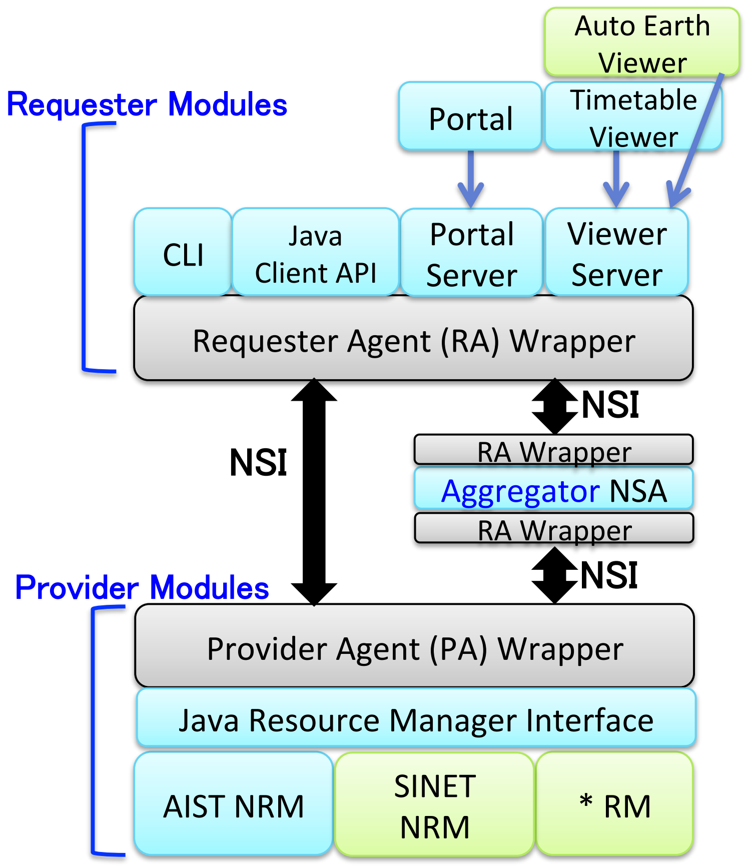


Figure 3: GridARS software modules.

request has been received, the Provider uses the corresponding network switch to provision the reserved path. Then, the Provider sends a 5. dataplaneStateChange notification message to the Requester. From the notification, the Requester knows the requested path has been linked up.

The CS v2 protocol defines a protocol that can be used to request and provide a service in-advance or on-demand. The service descriptions can be defined by each user community. Namely, the CS protocol can be used for not only networks, but also for other resources, such as computers and storage.

3 GridARS framework

We have been developing the GridARS framework, that enables management of inter-cloud resources provided by multiple administrative domains. In this study, we examine CS v2-compliant reference implementations we developed of a Provider, an Aggregator, a Requester, and a software libraries to develop those network service agents, as a part of the GridARS framework. We also developed a monitoring tool, which allows visualization of the reservation and data plane status of NSI experimental environments. First, we describe the overall architecture of the developed software modules, and then explain the details of each software module.

3.1 Software architecture

Figure 3 shows the CS v2 protocol-based GridARS software modules, which consist of Requester modules, an Aggregator module, and Provider modules. Requester modules consist of a Command Line Interface (CLI), Java Client API, Portal Server and Portal system, Viewer Server, and a Timetable Viewer.

```
// Create NSI2Client instance
NSI2Client client = new NSI2Client(
    providerNSA, providerURI, requesterNSA, requesterURI, replyWaitMsec, listener);

// Create criteria instance
ReservationRequestCriteriaType criteria =
TypesBuilder.makeRsvReqCriteriaType(
    schedule, srcstp, vlan, deststp, vlan, capacity);

// Send reserve request. rsvReply is returned when cf msg has been received
ReserveReply rsvReply = client.reserve(
    connectionId, globalRsvId, description, criteria);

// Send reserveCommit request
ReserveCommitReply commitReply = client.reserveCommit(reply.getConnectionId());
```

Figure 4: A pseudo code segment using the Java Client API.

Provider modules are composed by the Java Resource Manager Interface, and the AIST Network Resource Management system (NRM). In addition, modules in green indicate software developed by third parties.

The Requester, Aggregator, and Provider modules utilize Requester Agent (RA) and Provider Agent (PA) wrapper modules. The CS v2 protocol defines a SOAP-based Requester and Provider Web services interface described using WSDL (Web Services Description Language). We have used Apache CXF [12], an open source Web services framework, and the Jetty application server [13] in order to develop our RA and PA wrapper modules.

3.2 Requester modules

3.2.1 Command Line Interface (CLI)

CLI allows a user to send CS v2 Web services-based Requester operations, as described in Section 2.2, from the user's command line to Aggregators or Providers. CLI provides `reserve`, `commit/abort`, `provision`, `release`, `terminate`, and `query` commands, and parameters, such as bandwidth, reservation time, and Connection ID, that can be specified in the command options. Also, CLI asynchronously shows confirmed, failed, and notification messages from the Aggregator or the Provider on each user terminal when the messages have been received.

3.2.2 Java Client API

The Java Client API provides a Java programming interface, which enables the development of a CS v2 protocol-compliant Requester application program. The API also supports general authentication mechanisms such as HTTP basic authentication and OAuth2 over Transport Layer Security (TLS).

Figure 4 gives a pseudo code segment implemented by using the Java Client API. First, a user creates an `NSI2Client` instance, `client`, which is a Requester entity. The user provides the `NSI2Client` constructor with a Provider name, the Provider service URL, a Requester name, the Requester service URL, the waiting time for the response, and `listener` information. `listener` is used to receive messages from Aggregators or Providers. `ReservationRequestCriteriaType` is a type which stores reservation request criteria, such as reservation time, source and destination STPs, their VLAN IDs, and capacity information. The `client.reserve()` method sends the `reserve` request to the specified Provider. In the first `reserve` request, `connectionId` is set to `null`, because the Provider issues a `connectionId` for this reservation request. If the Provider returns a `reserveConfirmed` message, the program receives a `ReserveReply` instance, that includes `connectionId`. Then, the program sends the `client.reserveCommit()` method with the received `connectionId` and the reservation is booked.

3.2.3 Portal Server and Portal

In order to support the CS interoperation experiments, we defined a simple REST API and developed Portal Server and Portal as shown in Figure 3. Portal Server implements the REST API, which enables sending of CS v2 requests from web browsers. Portal provides a graphical requester interface, which sends reservation-related requests to the Portal Server. Portal Server was written in Java and developed using OOWeb [14], a light-weight HTTP server framework. Portal was developed using JavaScript and JSON.

3.2.4 Viewer Server and Timetable Viewer

As demonstration tools, we also developed Viewer Server and Timetable Viewer, which enable visualization of CS v2-based service reservation statuses. Viewer Server periodically retrieves all the reservation information managed by Aggregators. We extend the Aggregator's behavior of a CS v2 query request to provide Viewer Server with the managed reservation information because NSI has not defined a monitoring interface, yet. Viewer Server provides Web-based graphical user interface with the retrieved information via the REST API. We developed Viewer Server using Java and OOWeb. We also developed Timetable Viewer as a Java application. Timetable Viewer visualizes reservation statuses of network paths, and each user's requested end-to-end paths.

3.3 Aggregator

Aggregator automatically selects a suitable route for each requested path, and sends a reservation request to the related NSI Providers. Based on topology information provided by Topology Service, Aggregator calculates all the end-to-end paths among all STPs in advance. For each requested path from Requester, Aggregator selects a route from the calculated results and sends requests to the Providers, which manage the paths of the selected route. After Aggregator receives all the `reserveConfirmed` messages from the Providers, Aggregator sends `reserveConfirmed` to the Requester. If a part of a path reservation has failed, Aggregator sends the `reserveFailed` message to the Requester and sends the `terminate` messages to the related Providers. Aggregator also supports HTTP basic authentication and OAuth2 over TLS.

3.4 Provider modules

3.4.1 Java Resource Manager Interface

The Java Resource Manager (RM) Interface allows development of a CS v2-compliant Provider. RM Interface provides a Service Provider Interface (SPI) as shown in Figure 5, and calls these methods of a resource management software program, which implements the SPI, when RM Interface has received a CS v2 operation. RM Interface returns a confirmed or failed message to the Requester, based on the result of the resource management software process. Also, a resource management software program can notify RM Interface via the `Notifier` instance provided by RM Interface when the resource management software wants to notify the Requester of a status change, such as a link up or down, or a failure.

3.4.2 AIST Network Resource Management System

We developed a network resource management system (NRM), which manages and controls the network provided by AIST. AIST NRM was developed using the Java RM Interface.

```
public interface NSI2ResourceManager {
 public void setNotifier(notifier);
 public void reserve(header, connectionId,
             globalReservationId, criteria);
 public void modify(header, connectionId,
             criteria);
 public void commit(header, connectionId);
 public void abort(header, connectionId);
 public void provision(header, connectionId);
 public void release(header, connectionId);
 public void terminate(header, connectionId);
}
```

Figure 5: A pseudo code of Java resource management interface.

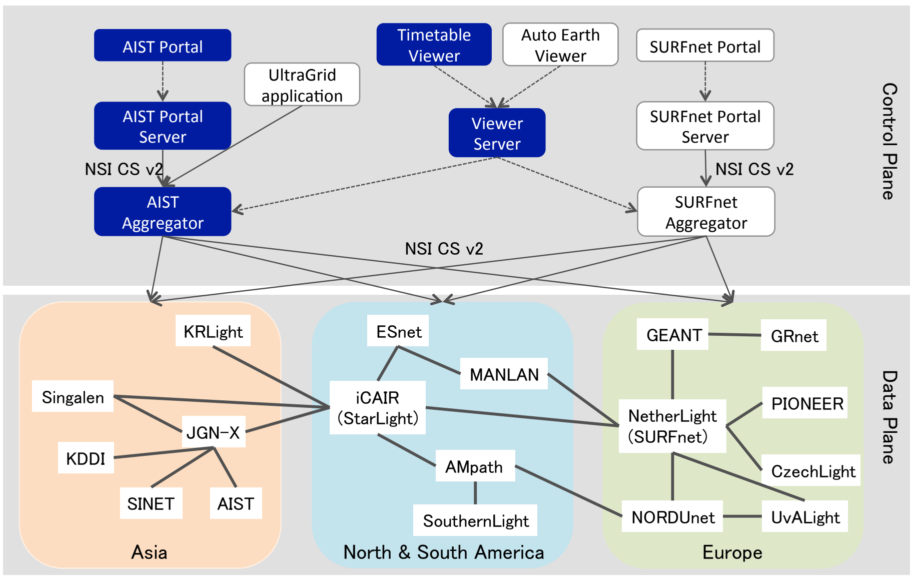


Figure 6: An overview of NSI test bed and software components in the NSI CS v2 interoperation experiments in 2013.

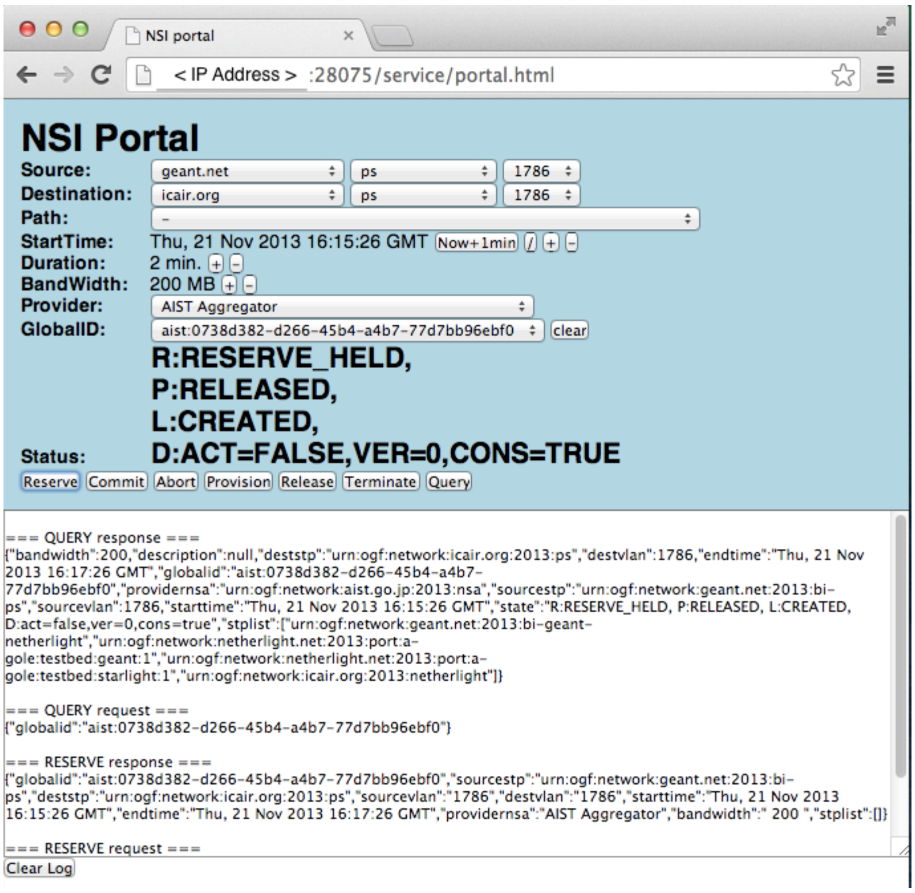
4 Evaluation

In order to show the feasibility of our software modules, we describe the NSI CS v2 interoperation experiments and introduce use cases of our software modules by third parties.

4.1 NSI CS v2 interoperation experiments

The NSI CS v2 interoperation experiments were held at GLIF 2013 and SC13. Figure 6 shows an overview of NSI testbed and software components in the NSI CS v2 interoperation experiments in 2013. The experimental test bed consisted of eighteen networks from Asia, Europe, and South and North America, five different provider implementations, two aggregator implementations, and two requester implementations. Each network is controled by each PA instance although we omit eighteen PAs from Figure 6.

We provided this international experiment with our AIST NRM provider, an aggregator, a Portal (requester), Viewer Server, and Timetable Viewer. Figure 7 shows a snapshot of our portal. The portal



NSI Portal

Source:	geant.net	ps	1786
Destination:	icair.org	ps	1786
Path:	-		
StartTime:	Thu, 21 Nov 2013 16:15:26 GMT		
Duration:	2 min.		
BandWidth:	200 MB		
Provider:	AIST Aggregator		
GlobalID:	aist:0738d382-d266-45b4-a4b7-77d7bb96ebf0		
Status:	R:RESERVE_HELD, P:RELEASED, L:CREATED, D:ACT=FALSE,VER=0,CONS=TRUE		

```
=== QUERY response ===
{"bandwidth":200,"description":null,"deststp":"urn:ogf:network:icair.org:2013:ps","destvlan":1786,"endtime":"Thu, 21 Nov
2013 16:17:26 GMT","globalid":"aist:0738d382-d266-45b4-a4b7-
77d7bb96ebf0","providernsa":"urn:ogf:network:aist.go.jp:2013:nsa","sourcestp":"urn:ogf:network:geant.net:2013:bi-
ps","sourcevlan":1786,"starttime":"Thu, 21 Nov 2013 16:15:26 GMT","state":"R:RESERVE_HELD, P:RELEASED, L:CREATED,
D:act=false,ver=0,cons=true","stplist":["urn:ogf:network:geant.net:2013:bi-geant-
netherlight","urn:ogf:network:netherlight.net:2013:port:a-
gole:testbed:geant:1","urn:ogf:network:netherlight.net:2013:port:a-
gole:testbed:starlight:1","urn:ogf:network:icair.org:2013:netherlight"]}

=== QUERY request ===
{"globalid":"aist:0738d382-d266-45b4-a4b7-77d7bb96ebf0"}

=== RESERVE response ===
{"globalid":"aist:0738d382-d266-45b4-a4b7-77d7bb96ebf0","sourcestp":"urn:ogf:network:geant.net:2013:bi-
ps","deststp":"urn:ogf:network:icair.org:2013:ps","sourcevlan":"1786","destvlan":"1786","starttime":"Thu, 21 Nov 2013
16:15:26 GMT","endtime":"Thu, 21 Nov 2013 16:17:26 GMT","providernsa":"AIST Aggregator","bandwidth":" 200 ","stplist":[]}

=== RESERVE request ===
```

Figure 7: Snapshot of the portal.

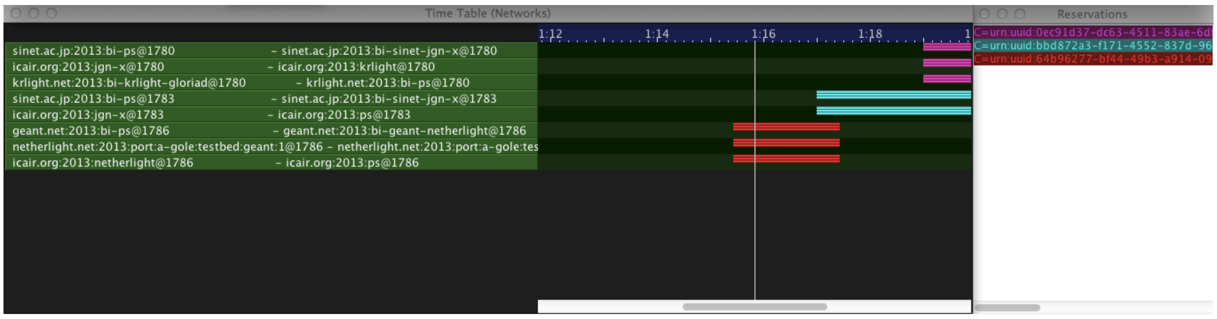
indicates that a user can specify source and destination STP information, path route, reservation time, bandwidth, and a provider or aggregator to which the user sends this request from the portal. When a user ticks a button related to each CS operation, the user can send the request and the request is processed by the specified provider. In this snapshot, the user requests a path from geant.net:ps to icair.org:ps. Figure 8 shows a snapshot of the Timetable Viewer at the experiment. The right-hand side indicates a reservation list which shows the Connection ID of each path reservation and its requester. The left-hand side indicates a timetable for path reservations, and each belt in the timetable means reservation time, from start time to end time. The horizontal axis indicates time and the white line represents the current time. The same colors in the reservation list and the timetable indicate the same reservations. Namely, the path from geant.net:ps to icair.org:ps reserved from the portal consists of three paths provided by geant.net, netherlight and icair.org, and the paths are represented by the three red belts in Figure 8. From this experiment, we confirmed that our software modules can work normally in an actual environment.

4.2 Use cases of GridARS framework by third parties

There are three existing applications developed using the GridARS framework.

UltraGrid application The UltraGrid [9] developer team has developed an on-demand high resolution video streaming application, which dynamically reserves a path over the NSI experimental test bed and sends video streaming over the path. This application used the Java Client API in order to reserve a path based on NSI CS as shown in Figure 6.

SINET provider agent (PA) NII has developed an NSI CS-compliant PA for SINET, which is a re-



```
sinet.ac.jp:2013:bi-ps@1780                   - sinet.ac.jp:2013:bi-sinet-jgn-x@1780
icair.org:2013:jgn-x@1780                     - icair.org:2013:krlight@1780
krlight.net:2013:bi-krlight-gloriad@1780          - krlight.net:2013:bi-ps@1780
sinet.ac.jp:2013:bi-ps@1783                   - sinet.ac.jp:2013:bi-sinet-jgn-x@1783
icair.org:2013:jgn-x@1783                     - icair.org:2013:ps@1783
geant.net:2013:bi-ps@1786                     - geant.net:2013:bi-geant-netherlight@1786
netherlight.net:2013:port:a-gole:testbed:geant:1@1786 - netherlight.net:2013:port:a-gole:tes
icair.org:2013:netherlight@1786               - icair.org:2013:ps@1786
```

```
C=urn:uuid:0ec91d37-dc63-4511-83ae-6d
C=urn:uuid:bbd872a3-f171-4552-837d-96
C=urn:uuid:64b96277-bf44-49b3-a914-09
```

Figure 8: Snapshot of the timetable viewer.

search and educational network in Japan. NII has already provided its own provisioning system. NII has also developed a protocol converter module between NSI CS and their own interface using the Java RM Interface, and has confirmed its interoperability at the SC13 experiment.

Auto Earth viewer KDDI has developed the Auto Earth viewer based on the Google Earth API [15] in order to support the NSI CS interoperation experiments. The Auto Earth viewer retrieves all of the reservation information from our Viewer Server via the REST I/F, and visualizes reserved and active paths over Google Earth.

5 Related work

There have been several software modules developed based on the NSI CS v2. OpenNSA [16] is a Python-based reference implementation developed by the NORDUnet project. The ESnet team developed a Java-based bridge module between the NSI CS protocol and their network resource management system called OSCARS [17]. SURFnet developed a portal system, an aggregator and a provider as open-source software modules in the same way as our GridARS framework does. However, the GridARS framework also provide libraries, such as the Java Client API and RM Interface, and monitoring tools.

6 Conclusion

We have developed NSI CS v2-compliant software modules, such as a provider, an aggregator, a requester and related libraries, and monitoring tools which enables visualization of the statuses of service utilization, just as in the GridARS framework. We contributed the developed software modules to the CS v2 international interoperation experiments over eighteen networks from Asia, Europe, and South and North America, held at GLIF 2013 and SC13. The GridARS framework is open-source software, and we showed the software is being used by third parties, including PA for SINET, an actual production network.

After the experiments in 2013, the CS v2 standard document was published in June, 2014 [18] and the experiments of CS v2 over Transfer Layer Security (TLS) were conducted in 2014. We continue to develop software modules for the latest CS protocol and aim to standardize interfaces for portal and monitoring systems in order to achieve improved usability of network services in future work. In addition, we will improve the performance, scalability, and robustness of our software framework.

Acknowledgments

This work was partly funded by the FEderated testbeds for Large-scale Infrastructure eXperiments (FELIX) project of the National Institute of Information and Communications Technology (NICT), Japan, and the Project for Developing Innovation Systems of MEXT, Japan.

References

[1] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "OpenFlow: Enabling Innovation in Campus Networks," ACM SIGCOMM Computer Communication Review, vol.38, pp.69–74, 2008.

[2] A. Takefusa, M. Hayashi, N. Nagatsu, H. Nakada, T. Kudoh, T. Miyamoto, T. Otani, H. Tanaka, M. Suzuki, Y. Sameshima, W. Imajuku, M. Jinno, Y. Takigawa, S. Okamoto, Y. Tanaka, and S. Sekiguchi, "G-lambda: Coordination of a Grid Scheduler and Lambda Path Service over GMPLS," Future Generation Computing Systems, vol.22 (2006), pp.868–875, 2006.

[3] S.R. Thorpe, L. Battestilli, G. Karmous-Edwards, A. Hutanu, J. MacLaren, J. Mambretti, J.H. Moore, K.S. Sundar, Y. Xin, A. Takefusa, M. Hayashi, A. Hirano, S. Okamoto, T. Kudoh, T. Miyamoto, Y. Tsukishima, T. Otani, H. Nakada, H. Tanaka, A. Taniguchi, Y. Sameshima, and M. Jinno, "G-lambda and EnLIGHTened: Wrapped In Middleware Co-allocating Compute and Network Resources Across Japan and the US," Proc. GridNets2007, 2007.

[4] C. Barz, M. Pilz, T. Eickermann, L. Kirtchakova, O. Waldrich, and W. Ziegler, "Co-Allocation of Compute and Network Resources in the VIOLA Testbed," TR 0051, CoreGrid, 9 2006.

[5] I. Baldine, Y. Xin, A. Mandal, C. Heermann, J. Chase, V. Marupadi, A. Yumerefendi, and D. Irwin, "Autonomic Cloud Network Orchestration: A GENI Perspective," Proc. the 2nd International Workshop on Management of Emerging Networks and Services (IEEE MENS '10), 12 2010.

[6] Open Grid Forum (OGF) Network Service Interface. http://redmine.ogf.org/projects/nsi-wg/.

[7] A. Takefusa, H. Nakada, T. Kudoh, Y. Tanaka, and S. Sekiguchi, "GridARS: An Advance Reservation-based Grid Co-allocation Framework for Distributed Computing and Network Resources," Job Scheduling Strategies for Parallel Processing, pp.152–168, Springer Berlin / Heidelberg, 4 2008.

[8] A. Takefusa, H. Nakada, R. Takano, S. Yanagita, K. Ookubo, T. Kudoh, and Y. Tanaka, "Resource Management Framework for Multi-domain Cloud," IEICE TRANSACTIONS on Information and Systems, vol.J94-B, no.10, pp.1332–1340, 10 2011. (in Japanese).

[9] UltraGrid Software. http://www.ultragrid.cz/en.

[10] Science Information NETwork 4 (SINET4). http://www.sinet.ad.jp/.

[11] G. Roberts, T. Kudoh, I. Monga, J. Sobieski, and J. Vollbrecht, "Network Services Framework v1.0," GFD. 173, 2010.

[12] Apache CXF. http://cxf.apache.org/.

[13] Jetty. http://www.eclipse.org/jetty/.

[14] OOWeb. http://ooweb.sourceforge.net/.

[15] Google Earth API. https://developers.google.com/earth/.

[16] OpenNSA. https://github.com/NORDUnet/opennsa.

[17] Virtual Circuits (OSCARS). http://www.es.net/services/virtual-circuits-oscars/.

[18] G. Roberts, T. Kudoh, I. Monga, J. Sobieski, J. MacAuley, and C. Guok, "NSI Connection Sevice v2.0," GFD. 212, 2014.

Development of a Reference Implementation of the NSI Connection Service Version 2.0 Standard.
31 March 2015
Information Technology Research Institute,
National Institute of Advanced Industrial Science and Technology (AIST)
1-1-1 Umezono, Tsukuba-shi, Ibaraki
305–8568, Japan
atsuko.takefusa@aist.go.jp
Reproduction in whole or in part without written permission is prohibited.