

A QoS Management Framework based on COP oriented Communication Resource Coordination

Hiroyuki Maeomichi, Ryutaro Kawamura, Akihiro Tsutsui and Kouji Yata
NTT Network Innovation Laboratories, NTT Corporation
maeomiti@exa.onlab.ntt.co.jp

Abstract

Due to the diversity of network technologies, network application programs, and user requirements, it is getting difficult to manage Quality of Service (QoS) for the network services. To address this difficulty, we propose a QoS management framework named Communication Service Concierge (CSC). CSC resolves the difficulty by using the Component Oriented Programming approach. CSC downloads components that includes resource coordination algorithm, and combines them to improve the QoS for applications' communication. This approaches allows CSC to provide adoptability to diversity and customizability for various user requirements. We designed CSC as a combination of two frameworks; Communication Resource Management (CRM) and Communication Logic Management (CLM). CRM executes resource coordination by accessing communication resources in the network through outside intervention; CLM handles the data in the communication flow. We developed CSC and some downloadable components and confirmed their feasibility by performance test. Finally, we explain a simple file transfer application as an example of CSC use.

1. Introduction

The rapid progress and increased diversity of network technologies are making Quality of Service (QoS) related problems more diverse and pushing them throughout the entire network. Emerging broadband access technologies such as ADSL and WLAN are accelerating these trends. We must, therefore, handle a number of distributed and quickly changing problems in order to manage end-to-end QoS levels. Meanwhile, the diversity of users' requirements is also increasing due to an increase in the diversity of users, end terminals, and applications for the Internet. Furthermore, even users of the same application have different requirements depending on the occasion. Therefore, QoS management functions must offer not only adaptability but also customizability to meet the users' requirements. Consequently, in order to solve network problems adaptively and provide the QoS required by the user, middleware located between network and applications will have an important role.

For the purpose, middleware researches have been strongly studied. One of the typical researches is an approach from the middleware for distributed computing as typified by CORBA [1] [2]. Such approaches expand its functionality to network and computer resource control based on the distributed object computing paradigm, and put QoS control functionalities into the middleware layer. With these approaches, it is assumed that applications are designed for specific middleware and it's

APIs. In concrete, applications use the API to establish communications with QoS.

Differing from those researches, our approach is that middleware manages QoS from outside of application by coordinating communication resources and keeps the relation between AP and OS as conventional as possible. We adopted Socket API as the interface. Following this approach, we proposed the QoS management framework "Communication Service Concierge (CSC)" [3] [4].

In order to adapt to diversity of QoS related problems and user requirements, and their rapid change, CSC uses the Component Oriented Programming (COP) approach. CSC realizes end-to-end QoS management by combining components that include semantics or algorithms for resource control. Since CSC is COP based, it offers adaptability to solve the QoS problems and customizability to meet the user's requirements.

We introduce the CSC architecture, implementation design, and initial evaluations of prototypes with initial evaluation.

2. QoS management middleware based on adaptive and distributed resource coordination

CSC is network middleware that offers adaptive coor-

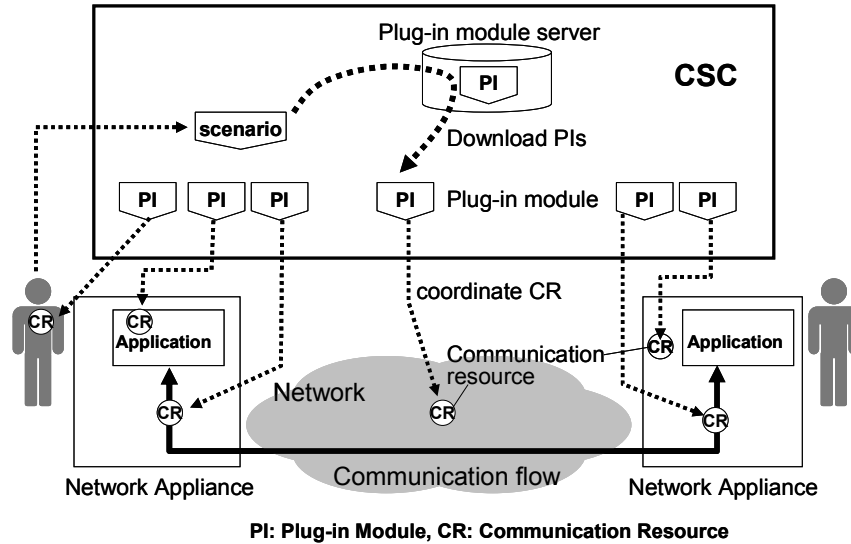


Figure 1: CSC reference model

coordination of distributed resources in the Internet in order to improve the QoS. The key objectives of CSC are to solve the rapidly changing QoS problems in the network and provide the customizability needed to meet user requirements. The methods and strategies of coordination are many and varied. For example, they include network bandwidth control and the selection of the application software. Moreover, the methods and strategies are ever changing due to the heterogeneity of network and user needs. Network middleware must, therefore, adapt to the evolution of coordination methods and strategies, and provide customization. CSC can meet these requirements by casting each method or strategy as a software component that is downloadable via the network; customization is provided by combining the appropriate components.

CSC not only copes with the network-level QoS problems but also handles the various factors associated with the user satisfaction including ease-of-use, safety, security and ubiquitous access.

Figure 1 shows the CSC reference model. The model takes "communication resource" (CR) in the widest possible sense, for example, network bandwidth, CPU time allocation in the network appliance, application software, protocol stack, and actor (human) in the communication session. The coordination functions of CR include observation, control, adjustment, allocation, navigation, and recommendation.

Plug-in modules are the entities that implements CR coordination. A plug-in module is a software component and has both the semantics of CR coordination

and methods to access the CR. Plug-in modules are held in the plug-in module server in the network, and are downloaded as needed. The scenario module is a special module; it has the highest level of coordination strategy. Usually, it is loaded first and then loads the other modules. The Plug-in modules, in combination, coordinate the CRs. Some plug-in modules offer GUIs to allow the actor/human to input requirements and judgments, and to navigate and recommend users' behavior.

One of the important concepts is that CSC tries to coordinate the communication through outside intervention; i.e., CSC affects an already established communication session in order to improve it. In other words, we keep CSC invisible to the application and networks as much as possible. While CSC is not mandatory for communication, it can setup a communication session including application software, if required. Another concept of CSC is best-effort. CSC realizes best-effort oriented coordination rather than guaranteed coordination. CSC pursues improvement rather than rigorously guaranteeing the end-to-end QoS.

Figure 2 shows the CSC control model. CSC consists of two frameworks: CRM (Communication Resource Management) and CLM (Communication Logic Management). CRM plays a central role in CSC's basic concepts. It tries to improve the QoS by CR coordination. The plug-in modules of CRM have the semantics of CR coordination and access methods. Given this perspective, CRM is not typical middleware, which hides the attributes of the lower layer and provides services to the higher layer by using high-level of abstraction. CLM, on the other hand, is directly involved with the communi-

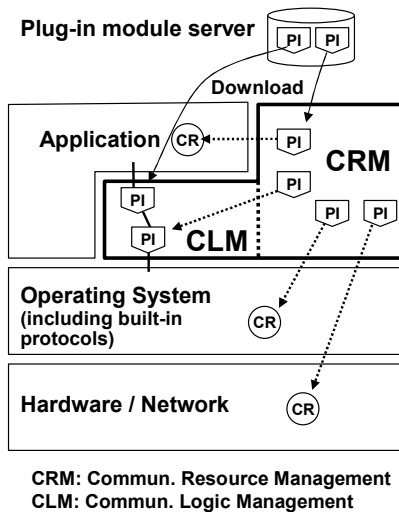


Figure 2: CSC control model

communication protocols. The plug-in modules of CLM are inserted into the middle of data flows in the end systems or network equipment.

We believe that the combination of resource coordination by CRM and protocol coordination by CLM can effectively improve the QoS levels. For example, when a bandwidth bottleneck arises during a video streaming service, two options are possible. In one, the CRM plug-in module tries to raise packet priority to improve video quality if the network supports the QoS / CoS functionalities; however, extra cost may be incurred. In the other, the CLM plug-in module adaptively changes the video encoding quality/algorithm to suit the available bandwidth. Service quality is lower but still better than that realized when many packets are dropped at the bottleneck.

3. Communication Resource Management

Figure 3 overviews the current architecture. Both CRM and CLM have cores (CRM core, CLM core) and plug-in modules (CRM modules, CLM modules). CRM is a framework to coordinate the CRs distributed along the end-to-end communication links. The CRM core formulates the distributed object environment, and executes the CRM modules on it. The CRM core manages CRM module's lifetime and provides a service interface to the CRM modules.

CRM module may control CR directly or delegate operations to another CRM module. In the latter case, a CRM module can either use an existing CRM module or create a new CRM module on any CRM core. CRM cores handle these requests and manage the relationship between using CRM module and serving CRM module. The relationships can be nested to make a tree structure. The top CRM module, called "scenario", usually implements the QoS improvement strategy. With the relationships, CRM cores manage their expiration time to avoid orphaned CRM modules.

Since CRM is a type of distributed agent computing environment, it has to solve the security problems common to this field. For CRM, we prepared three types of security mechanism: (1) mutual authentication between CRM cores using public key cryptography, (2) checking mechanism of digital signature for downloading component, and (3) runtime authorization check of CRM module code.

The current CRM implementation is based on Java.

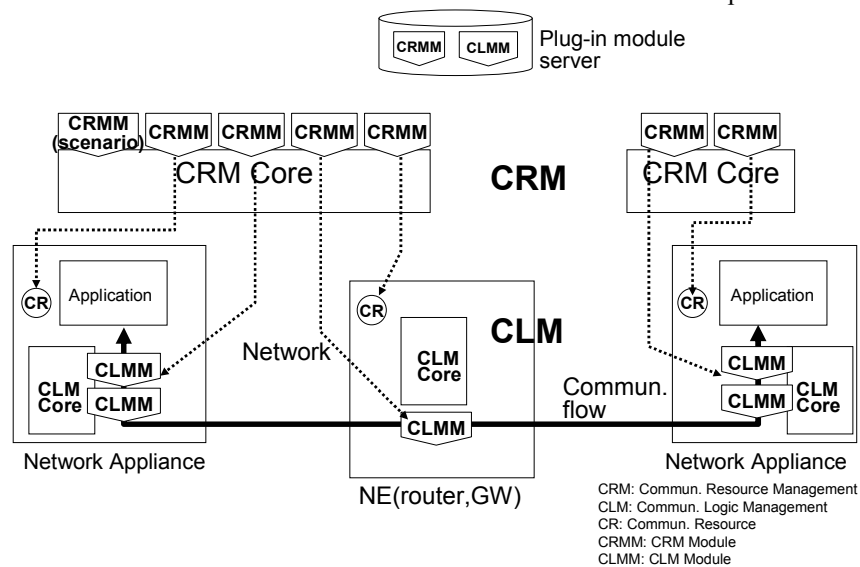


Figure 3: CSC architecture overview

The reason for choosing Java is that it is the de facto standard of OS-independent computing and various kinds of devices support it. Java has the basic functionalities required for CRM such as object class downloading. The current CRM implementation uses a part of the Open Services Gateway initiative (OSGi) middleware [5]. OSGi is thin middleware running on Java and offers software component (called "Bundle") download, execute, stop and so on.

4. Communication Logic Management

CLM is a framework that is involved in controlling the communication data directly, unlike CRM. CLM also consists of a core (CLM core) that provides the execution environment and CLM modules that hold the semantics. CLM module examples include encryption, compression, QoS monitoring, shaping, policing, synchronization, packet filtering, dynamic transcoding of video stream, and so on. In fact, most of them can be categorized as some sort of protocol or filter.

The CLM core uses socket abstraction to provide the API to the applications. This simplifies the use of existing applications that use socket API. Moreover, we believe that loosely coupling between the applications and the middleware layer results in fewer and simpler interfaces. This is beneficial in terms of adaptability to QoS problems and customizability. Based on this concept, CLM uses the Java Socket compatible interface (CSC Socket API) instead of its own abstraction for the API. The data received from an application is processed sequentially in each CLM module, which makes a data process chain underneath the CSC Socket API.

We consider that CLM is a type of dynamic protocol stack framework as typified by STREAMS/XTI [6] and FilterStreams in Java. However, CLM is different from these frameworks in that it can download plug-in modules (protocols) dynamically from the network; modules can be manipulated from outside the application. Moreover, CLM can reconstruct CLM module chains without any data loss even if the application is still communicating.

CLM is also used in network equipment. For example, if the end system has very low-performance, other equipment, such a home router, can offer CLM functionality on behalf of the end system. In this case, CLM acts as a sort of "proxy", and incoming data from the network runs through the CLM module chain to finally be released to the network. This proxy also allows existing applications, including non-Java ones, to use CLM functionalities.

The current implementation is based on Java J2SE 1.4.1. Each CLM module is constructed using JavaBeans, and inter module data passing is realized based on the event model of JavaBeans.

One focus of CLM design was performance. To enhance performance, we designed it to minimize memory copy during data passing. We conducted a performance test using two PCs (Pentium4 2.4GHz, 1GB memory, Windows XP) connected via a gigabit Ethernet switch. The test application simply generated data, sent it at a specified rate using TCP, and received it at the other side. The block length of the generated data was held constant at 1024 bytes. We developed the fol-

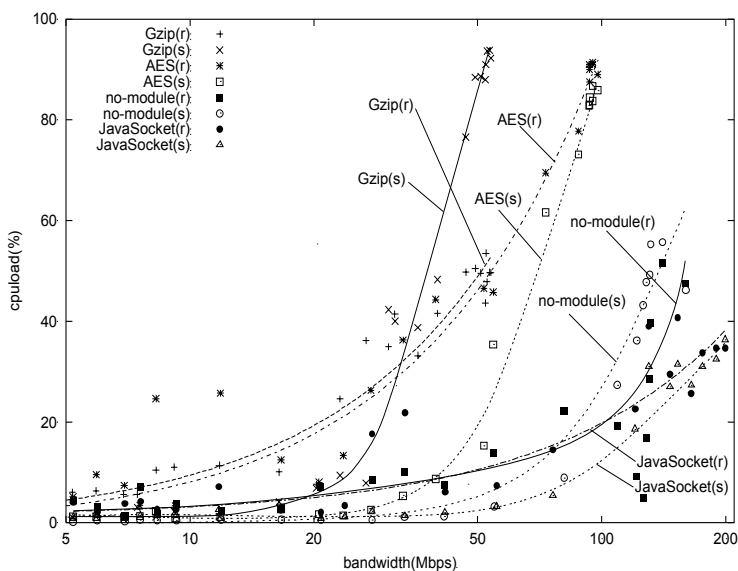


Figure 4: Results of CLM Performance Evaluation

lowing two CLM modules: (1) data compression / decompression module using the DEFLATER algorithm [7], and (2) an Advanced Encryption Standard (AES) [8] based encryption/decryption module. We conducted a performance evaluation test and observed the relationship between CPU load and throughput, which is the average bandwidth sent/received by the application. We used text data of "rfc1951.txt" as the data compress module and randomly generated data for other cases. The block length and key lengths of the AES modules were 128 bits.

Figure 4 shows initial results from performance tests. "no-module" shows the result with no CLM module (i.e. simple connection to the Java Socket (TCP)). "JavaSocket" means the conventional Java Socket without CLM. Data for sending side are marked "(s)"; data for receiving side are marked "(r)". From this graph, we can see that the CLM structure (no-module) does not increase CPU load seriously compared to the conventional Java Socket. The CPU load for AES modules and compression modules are relatively high. However, they still can be used in the 20 Mbps bandwidth range under 20% CPU load. Considering CPU load for more complex applications running on CSC, we estimated that CPU load for CSC should not exceed around 20%. Given this estimation, the results indicate that applicability is reasonable up to 20 Mbps.

We confirmed that these modules and CLM core's design and implementation achieve feasible performance.

5. Use Example

As an example of CSC control, we describe a simple file transfer application consisting of a receiver program and a sender program

When the transfer rate is insufficient due to a bottleneck in the network, CSC may activate a compression and decompression CLM modules to enhance the effective transfer rate.

If the transfer is via a CoS / QoS enabled network such as diffserv, CSC may download and use a CRM module to control QoS parameters of the network for the flow that the applications are handling. Of course, this coordination can be use with the coordination by the data compression/decompression modules described above.

Meanwhile, if the flow pass through a high-risk

network and the application user requires the high security, CSC may activate encryption / decryption modules in the end terminals to prevent information leaks. We consider that security is also an aspect of QoS.

As shown above, CSC improves communication in various ways without any modification of the original application programs.

6. Conclusions

In order to adapt to diverse and distributed network problems and satisfy diverse users' requirements, we proposed an end-to-end QoS management framework, CSC. We designed CSC using the COP approach to provide adaptability to various environments. CSC downloads and executes components, which realize the QoS control algorithms needed. CSC consists of two frameworks: CRM and CLM. CRM coordinates network resources, while CLM access the data that applications are communicating with. The full paper will introduce the CSC architecture, implementation design, and describe the evaluation of a prototype implementation.

References

- [1] D. C. Schmidt et al., *Developing Next-Generation Distributed Application with QoS-Enabled DPE Middleware*, IEEE Communications Magazine, October 2000.
- [2] G. Coulson et al., *Supporting Mobile Multimedia Applications Through Adaptive Middleware*, IEEE J-SAC, Vol. 17, No. 9, September 1999.
- [3] R. Kawamura et al., *A Middleware Architecture for Adaptive and Distributed Resource Coordination in IP Networks*, submitted to IEEE Network Magazine.
- [4] H. Maeomichi et al., *A User-Side QoS Management Framework: Communication Service Concierge*, IEEE CQR 2002 (International Workshop on Communications Quality and Reliability), May 2002.
- [5] *Open Service Gateway Initiative Specification*, http://www.osgi.org/resources/spec_overview.asp
- [6] AT&T, *Unix System V Release 4: Programmer's Guid : Streams*, June 1992, ISBN 0130206601
- [7] A. Feldspar, *An Explanation of the DEFLATE Algorithm*, August 1997, <http://www.gzip.org/deflate.html>
- [8] *Advanced Encryption Standard Home Page*, <http://csrc.nist.gov/CryptoToolkit/aes>