Ontology Modeling of a Dynamic Protocol Stack

LiFeng Zhou[1], Hung Keng Pung[1], Lek Heng Ngoh[2], Tao Gu[1]
[1] *School of Computing, National University of Singapore, Singapore*
[2] *Institute for Infocomm Research, Singapore*
{zhoulife, punghk, gutao}@comp.nus.edu.sg, lhn@i2r.a-star.edu.sg

Abstract

This paper proposes a formal approach for protocol information modeling and validation leveraging on ontological techniques. We demonstrate the advantage of our approach through prototyping a protocol management system for representation of communication protocols and composition of protocol stacks. The prototype has an ontology-based model to describe meta-data of protocols and protocol stacks in a systematic way. Consequently, the retrieval of protocols and the validation of protocol stacks are realized by corresponding operations on the ontology model. Owing to the better expressiveness of RDFS, the ontology model can describe protocols with higher 'fidelity'. Our experimental results show that the ontology-based protocol management system is operable and provides expressive knowledge modeling without compromising the performance.

1. Introduction

With the rapid advancement in media technologies and networks, distributed multimedia applications are expected to be deployed in an environment that is more dynamic and heterogeneous than ever before. Effective QoS provisioning is more challenging as multiple end-to-end components – from applications, hosts' OS and middleware to the underlying networks – are interacting and intertwining in very complex ways. This has led researchers to focus on different aspects of QoS provisioning in a fashion similar to the layered approach in network systems design. As a result, silos of QoS solutions (each particular to one QoS dimension) have been invented, which often led to instability and overall inefficiency of end-to-end QoS provisioning due to poor coordination between respective QoS efforts.

The work reported in this paper has been funded in part by the Agency for Science, Technology and Research (A*STAR) of Singapore under Grant 0520150024.

In view of this, we have proposed a semantic-based QoS control and management framework (QCMF) for a cooperative end-to-end QoS provisioning [5]. It aims at accommodating and coordinating existing QoS mechanisms at three main QoS entity levels - network level, middleware level and user/application level and study their behaviors and inter-relationships.

This paper focuses on the middleware level QoS modeling, with emphasis on the semantic modeling of communication protocols and protocol stacks. Traditional protocol stack in end-hosts is static in nature and hence is not able to re-compose to suit demanding networked multimedia applications over diverse runtime environments. For example, once a protocol stack is established for applications at build time, runtime restructuring of the stack to deal with resource scarcity is normally not supported. In view of this, we have proposed a dynamic protocol framework (DPF) [1], which overcomes such limitations of static protocol stacks by providing protocol stack adaptation at runtime.

The key enabler of DPF is a *protocol management system* which deals with representation of protocols and composition of protocol stacks. This paper proposes an ontology-based approach to the modeling and processing of protocol/stack information. It has the advantages of being expressive, flexible and interoperable with other QoS systems (e.g., QoS-enabled applications) in end-to-end QoS coordination. The proposed ontology-based approach of representing protocol's properties and protocol stack's composition serves as a design paradigm for the semantic modeling of other QoS systems along the end-to-end path. The common semantic modeling approach of QoS systems will facilitate the exchange and processing of their QoS information, which can in turn result in better interoperability in QoS control and management among these systems.

The rest of this paper is organized as follows. Section 2 discusses related work; Section 3 presents an overview of QCMF and DPF. Section 4 describes the ontology model of protocol layers, protocols and stacks.

Implementation results and performance evaluation are presented in Section 5. This is followed by a conclusions and future work in section 6.

2. Related work

Much research has been done in the area of dynamic composition of protocol stacks. However, most of them have taken an ad-hoc approach to design proprietary and exploratory protocol management systems which lacks formality and expressiveness.

Rwanda [4] and its successor Chameleon [11] focus on providing tailored protocol services to support diverse requirements of different media types. Rwanda models a protocol stack as a linear list of protocol objects which represents a QoS such as reliable delivery or encrypted communication. All information is implemented by a Java class and is runtime retrieved via reflection for configuration. However, both Rwanda and Chameleon have only designed a few protocol properties for demonstration purpose. The systematic modeling of protocol properties and efficient processing of protocol information for stack configuration is not considered in their projects.

Dynamic Layered Protocol Stack (DLPS) [3] of Microsoft provides a method to dynamically build a protocol stack for data transfer. A stack description file has been designed comprising a plurality of individual protocol layer description so that the plurality of all these layer descriptions together define which protocol layers will be included in the protocol stack when it is constructed. However, DLPS has not touched on the issue of modeling properties of protocol stacks as a collection of individual protocols. Furthermore, each protocol in DLPS is described individually without considering the re-use of common characteristic of protocols. For example, all video codec protocols have properties such as the number of tracks and sampling rate, which can be abstracted as a base knowledge for codecs of that category.

In [12], component description is introduced to represent protocol building blocks. Each component is described by a list of provided properties and required properties. The former declares the functionality that can be provided by a component while the latter defines its conditions to be satisfied by others (e.g., downward and upward dependencies). An algorithm to select building blocks is also presented where the solving strategy of stack composition is to match the requested properties of one component with the provided properties of others. However, their work has only focused on the dependency properties of protocols and neglected the importance of other protocol properties in runtime stack building. For example, the selection of G.723 protocol or MPEG protocol to stream audio flows runtime should depend on the resource availability since they have the same dependency on the RTP protocol. A selection algorithm considering only one searching factor (e.g., dependency as used in [12]) surely would not find an optimal stack composition in practice.

Among all the above projects, protocol modeling is accomplished by individual efforts making use of either programming languages elements or proprietary data structures [14]. These work lack of generality and most likely cannot provide customized and expressive descriptions for protocols and stacks. In this paper, we present our ontology-based protocol model using RDFS that addresses these shortcomings. The ontological approach to QoS modeling is initiated from the semantic web community. However, reported work [7][8][9] so far is limited to the description of web services' QoS properties for service matching and selection. In contrast, we propose in this and other papers semantic modeling and sharing of QoS information for QoS components (e.g., network and middleware) along the end-to-end path, thus forming a knowledge basis for correct QoS configuration and adaptation.

3. QCMF and DPF overview

The semantic-based QoS control and management framework (QCMF) is our research effort for a holistic approach to end-to-end QoS provisioning by considering and coordinating QoS mechanisms from different dimensions for end-to-end benefit. The motivation of QCMF is based on the identification of two shortcomings in current QoS tools and technologies.

Firstly, most current QoS researches focus on individual aspects and areas of the QoS provisioning mechanisms (e.g., within application, middleware or network), with less attention being paid to the collaboration between these facilities. These isolated viewpoints will lead to undesirable or inefficient solutions with respect to end-to-end QoS. For instance, it is not sufficient to rely purely on the QoS provisioning mechanics in end-hosts to guarantee the performance of a multimedia flow from a sender to a receiver. The network in between should also play a role in assuring the quality of the flow if such supports really exist (e.g., in a DiffServ [15] network). Hence, we assert that any useful end-to-end QoS solution must consider the coordination of QoS mechanisms between layers or components (such as those in end-hosts and in network) and manage them in a cohesive and co-operative fashion.

Secondly, current QoS solutions mainly focus on the mathematical calculation of QoS parameters (e.g., [6]) while neglecting the semantic meanings of QoS terms.

In our view, the explicit representation and differentiation of QoS concepts and their relations is equally important for a correct QoS configuration and adaptation end-to-end. A simple example can be illustrated in a grid scenario where a user requests his job to be executed by a UNIX (compatible) server. During QoS negotiation, a service provider running in a Linux environment would not be selected for job execution in a traditional keyword-based matching. However, if we can setup an ontology for relevant QoS knowledge representation (in this case is a QoS ontology about OS information), semantic QoS negotiation can be fulfilled to reason and acknowledge the QoS compatibility between the job requester and supplier.

As mentioned, the proposed QCMF framework aims at accommodating existing or new QoS mechanisms at three levels and studying their interactions with respect to end-to-end QoS configuration and adaptation. We believe a key step toward the solution lies in the representation and organization of QoS meta-information to facilitate knowledge understanding and exchange along the end-to-end path. This paper focuses on the middleware level QoS modeling, with emphasis on the semantic modeling of communication protocols and protocol stacks.

At middleware level, we have designed a dynamic protocol framework (DPF) [1], which can provide dynamic protocol stack composition at call-setup time and re-composition (i.e., protocol swapping) at runtime. In the context of QCMF, DPF offers one possible dimension of QoS adaptation within the communication protocol stack which can supplement current prevailing QoS solutions at application or network level. The operations of DPF (e.g., runtime re-composition of stacks) are triggered by QCMF as the outcome of a coordinated decision-making process among different QoS dimensions and are transparent to end-users.

4. An Ontology-Based Model for DPF

4.1. Design considerations

In line with the semantic modeling of other QoS components in QCMF, we have chosen an ontology-based approach to model protocols and manage the dynamic composition of protocol stacks for the following reasons: firstly, ontology is a formal description of concepts and relationships, which is expressive in describing notions, their relations and restrictions. Ontology provides a means for formulating semantic models of knowledge while other schemas such as XML can only produce a data model (which is a tree); secondly, the use of ontology enables different QoS components in QCMF to have a common

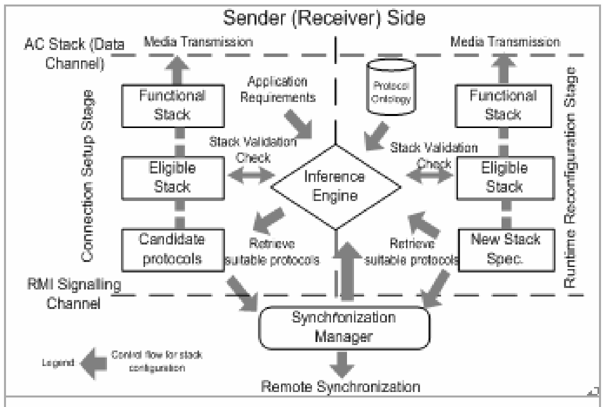


Figure 1. Architecture of DPF with ontology modeling

understanding of QoS knowledge while interacting with one another. The identification of the semantic meanings of QoS concepts is essential in information sharing along the end-to-end QoS provisioning path; lastly, ontology-based approach facilitates the machine processing and reasoning of QoS concepts as is demonstrated in [5].

Semantic QoS modeling also contributes to restoring the runtime portability of QoS demanding applications in heterogeneous environments. Take the protocol stack composition in an end-host as an example. In end-to-end QoS provisioning, the selection of a targeted protocol in forming a stack is dependent on multiple factors such as the availability of that protocol in the end-host and the preference of the communication peer. Hence it is meaningless to explicitly bind an application to a specific protocol beforehand. Instead, our semantic approach for QoS modeling allows applications to specify their protocol need (e.g., resource reservation feature) rather than fixing the name of a protocol. Based on the semantics of each available protocol runtime, appropriate protocols (e.g., RSVP [17]) will be fetched automatically to compose a protocol stack that fulfills application requirements.

To realize our ontology model, RDFS [10], a W3C recommended language for defining ontologies has been employed. The selection of RDFS rather than OWL [21] in our project is based on a realistic balance of language capability and performance: RDFS is sufficient for modeling of protocols (as can be seen from next sections) and is faster than OWL in ontology processing.

The architecture and workflow of DPF is shown in Figure 1. The protocol management system is basically composed of two sub-modules: a *protocol ontology base* which represents and stores protocol information ontologically and an *inference engine* which takes protocol ontology as input and interacts with other components of DPF throughout the lifecycle of a session to guarantee the correct composition of protocol stacks. Details about the modeling and manipulation of protocol information will be presented in the following sections. Particulars about functional manipulation of protocols (.e.g., protocol insertion or swapping) in DPF can be found in our previous paper [1] and patent [2].

4.2. Ontology modeling of layers

In this section, we introduce our ontology-based protocol modeling and classification. Since each protocol in DPF provides a specific service, we define an overall *service* class in RDFS expression (see Figure 2) to capture the common properties that exist in various protocols. These common properties include layer name (where the protocol lies in the stack), protocol name (what is the protocol) and class name (where to find the functional code of the protocol). Other common properties such as dependency and compatibility will be addressed in the next section.

```
<rdfs:Class rdf:ID="Service"/>
<rdfs:Class rdf:ID="Transport">
   <rdfs:subClassOf rdf:resource="#Service"/>
</rdfs:Class>
…
<rdfs:Property rdf:ID="className">
   <rdfs:domain rdf:resource="#Service"/>
</rdfs:Property>
<rdfs:Property rdf:ID="protocolName">
   <rdfs:domain rdf:resource="#Service"/>
</rdfs:Property>
```

Figure 2. Entry point: class service and its properties

We then classify protocols into *layer* categories according to their positions in the protocol stack. Such classification is based on the observation that protocols of the same layer usually provide similar functionalities. For example, network layer protocols specifically provide connectivity service among hosts. Hence ontology models can be established on a per-layer basis to describe the common functionalities provided by that layer. Each layer extends the fundamental *service* class and also defines layer specific attributes. Figure 3 shows an example of one property specific to the codec layer. The scope that this property applies to is defined by the *domain* expression, which provides isolation of property usage at grammatical level. All properties of a layer, either inherited from the basic description of *service* class or specifically defined in that layer definition, together form the *ontology model of a layer*.

```
<rdfs:Property rdf:ID="supportedInputFormats">
   <rdfs:domain rdf:resource="#Codec"/>
   <rdfs:range rdf:resource="xsd:String"/>
</rdfs:Property>
```

Figure 3. "supportedInputFormats" is a property of codec layer

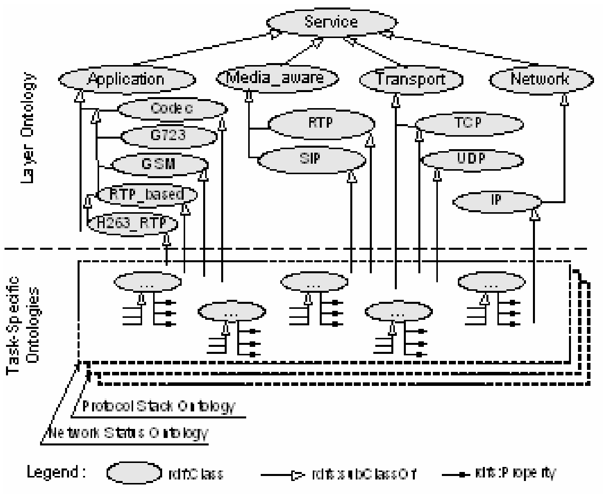


Figure 4. Protocol classification and Ontology modeling

Four layer categories have been developed for DPF currently: *Network layer*, *Transport layer*, *Media-aware layer* and *Application layer*. Network layer contains layer 2 protocols such as IP and IPX while transport layer includes TCP and UDP. Media-aware layer accommodates protocols related to media transmission such as RTP. The composition of application layer is more complex than other layers. Protocols in this layer have been designed to serve applications with diverse nature. There are no uniform criteria to define the common features of application layer protocols. Rather, the characteristics of these protocols depend on the application domain concerned. For example, in the context of multimedia transmission, these application layer protocols refer to codec which provide diverse encoding and decoding functions for media data.

In view of this, we classify application layer protocols according to the application domain they serve. Three categories of application protocols have currently been defined in DPF: codec protocols, security protocols and session control protocols. Such a classification is extensible in nature in that more detailed classification is also possible to provide fine-grained descriptions. For example, the category of codec protocols can be further divided into audio and video codec sub-categories, both of which can be further partitioned into RTP-based or non-RTP-based codec types (see Figure 4). A hierarchical layer model can thus be setup and extended on demand to classify protocol types in more details.

4.3. Ontology modeling of protocols

The hierarchical ontology model of layers serves as the template to model protocols belonging to that layer. An individual protocol is defined as an instance of a specific layer. Thus the *ontology model of a protocol* can be setup according to the layer ontology. For example, TCP is defined as an instance protocol of transport layer. Figure 5 shows a partial definition of TCP ontology model. Properties such as port are specific to the transport layer and are defined by the transport layer ontology. Other properties such as className and dependency are derived from *service* class. The layer ontology is reusable among protocols of the same layer. Hence the ontology model of another transport layer protocol (e.g., SPX) can also be portrayed based on transport layer ontology model.

```
<Transport rdf:ID="TCPOut">
 <layerName>protocol.srv.Transport</layerName>
   <protocolName>TCP</protocolName>
   <className>transport.TCPOut</className>
   <upperDepd>NULL</upperDepd>
   <lowerDepdy>Network.IP</lowerDepd>
   <reliability>true</reliability>
   <port>4088</port>
   <polarity>1</polarity>
   ...
</Transport>
```

Figure 5. TCP is of (rdf:) type transport and is modeled according to the transport layer ontology model

The ontology models of layers and protocols are recorded in the *protocol ontology base* as two RDFS documents: *schema* and *instance*. These two files are runtime processed in the *inference engine* for grasping semantic features of protocols and relations among them. At the stage of protocol stack composition/re-composition, suitable protocol components are retrieved from the inference engine to compose a protocol stack. As explained before, to ensure the flexibility of deploying applications in heterogeneous run-time environment, DPF allows applications to describe their desired protocol service features at design-time instead of explicitly bounding a specific protocol. Hence the retrieval of suitable protocols from inference engine is not restricted to just match the name of a protocol. In fact, every property of a protocol in DPF can be used as a criterion to select a protocol component. For example, if the stack specification from users requires a reliable data transmission, reliability will then be used as the keyword to possibly retrieve TCP as one of the stack component.

Our ontological design to protocol and stack modeling has several advantages compared with a traditional java inspection based approach (such as that employed in [4]). Firstly, the maintenance and modification of protocols are expressive and flexible. For example, erasing a protocol instance only requires the update of *instance* file while the modification of layer ontology only involves the refreshment of *schema* file. Secondly, both layer model and protocol model are encoded in RDFS language where information is accessible to other end-to-end QoS components. Thirdly, the ontology-based approach for protocol modeling can easily provide customized and expressive specifications for protocols/stacks and enable code reusability as has been demonstrated. Finally, protocol information can be semantically compared, reasoned and understood by different machines in a distributed environment.

4.4. Stack modeling and validation

In DPF, a protocol stack is a protocol graph that consists of a vector of protocols in the sequence of layers. Hence, *ontology model of a protocol stack* is defined as the integration of ontology models of protocols comprising that stack. As protocol stack is dynamically composed at runtime, stack ontology is a memory model to describe the features of a transient stack. Different combination of protocols will produce stacks with different characteristics. The characterization of stack is essential for deciding whether the stack composition can suffice application/user requirements or not. For example, <G723, RTP, UDP, IP> is a protocol stack for audio transmission. This stack is characterized by having low perception quality (derived from G723 protocol model), unreliable data transmission (derived from UDP protocol model) and supports real time session control (derived from RTP protocol model). If an end-user can accept medium to low audio perception quality, then such kind of stacks can be employed at runtime in case of resource scarcity.

On the other hand, not every combination of protocols forming a stack is valid. For example, the stack composition <JPEG, RTP, UDP, IPX> for video streaming is not acceptable because UDP is not compatible with IPX. To describe the compatibility among protocols in constituting a protocol stack, we introduce a special property element – *compatibility* – to capture such relationship, as is shown in Figure 6.

```
< rdfs:Property rdf:ID="compatibility">
   <rdfs:domain rdf:resource="#Service"/>
   <rdfs:range rdf:resource="xsd:String"/>
</rdfs:Property>

< rdfs:Property rdf:ID=" upperDepd">
   <rdfs:domain rdf:resource="#Service"/>
   <rdfs:range rdf:resource="xsd:String"/>
</rdfs:Property>
```

Figure 6. RDFS definition for compatibility and dependency

As pointed out previously, dependency is another important property of protocols which describes the reliance of one protocol on other protocols. For example, H263_RTP is a JMF (*Java Media Framework* [18]) video codec whose deployment requires the presence of RTP in the protocol stack. We introduce an additional property element – *lowerDepd* – to describe such a dependency as the desired protocol RTP reside below H263_RTP in the protocol stack. It's also possible that a protocol requires other protocols to appear on top of it in the stack. For example, TCP/UDP and IP are always bundled together in the protocol stack. Thus IP protocol has a dependency on upper side transport layer protocols (either TCP or UDP). We also introduce a property element – *upperDepd* – to describe such a requirement. The definition of the *upperDepd* property is also shown in Figure 6.

Compatibility and dependency are modeled as properties of the basic *service* class since all protocols may have such requirements. The dependency and compatibility knowledge of each protocol is supplied by protocol developers. To ensure that all protocols in the stack can cooperate properly with each other, stack validation is enforced once a stack is defined. The stack validation goes through two stages: *grammatically check* and *specification check*. The former one guarantees that dependencies of every protocol and compatibility of the stack are satisfied. The latter one makes sure that the stack is configured in accordance with user/application requirements. Those stacks that pass the validation check are legal stacks and will be negotiated among communication peers for the initiation of data transmission.

5. Implementation and evaluation

We have implemented the ontology-based protocol management system in Java on J2SE 1.5.0 platform. JMF is used as the runtime environment for multimedia streaming. 11 protocols, namely RTP, TCP, UDP, IP, G711 codec, G723 codec, GSM codec, MPEG codec, H263 codec, JPEG codec and affined encryption/decryption, have been classified into 4 layers and modeled respectively. The layer and protocol ontology consists of 24 classes and 62 properties. The Jena Semantic Web Toolkit [13] has been chosen as the inference library to load RDFS models about protocols and explore relations among layers and protocols (e.g., protocol dependencies).

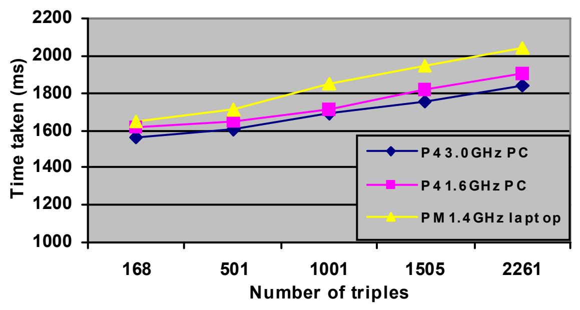
TABLE 1. Time-taken in DPF management functions

Operation		Time (ms)	
		DPF-2	DPF-1
Creation of SyncMaster		453	484
Creation of SyncSlave		250	320
Creation of Ontology Model (total)		1563	-
	Loading of schema/data file	1313	-
	Creation of inference model	250	-
Creation of Java Protocol Registry		-	3645
Load of Java Protocol Registry		-	104
Start-Up Protocol Stack Composition (total)		1446	1632
	Discovery of protocol components	93	500
	Creation of protocol stack	1	1
	Query protocol dependency	2	1
	Check stack validation	31	30
Runtime Protocol Stack Re-composition (total)		148	168
	Discovery of targeted protocol	1	5
	Re-creation of protocol stack	1	1
	Query protocol dependency	1	2
	Check new stack validation	27	29

Experiments have been conducted to evaluate the performance of ontology-based DPF (named as DPF-2) with emphasis on the semantic protocol discovery and stack validation. The configuration of the testbed is as follows: two DELL PCs with Pentium IV processor 2.4 GHz and 512MB memory are employed as end-hosts. Microsoft Windows XP is the primary operating system on these machines. Every end-host holds a local protocol ontology base which provides ontology models for layers and protocols. For end-to-end communication, these two end-hosts act as media sender and receiver respectively. One of the end-host also serves as the synchronization manager (SyncMaster) to coordinate the communication. We obtained our measurements from initiation of audio/video streaming and runtime swapping of codecs (or transport layer protocols). The results of the experiment are shown in Table 1. For comparison, we have also implemented a Java introspection based protocol system (named as DPF-1) similar to that of Rwanda [4] and gathered measurement under the same environment setting.

As can be seen from Table 1, the overall performance of DPF-2 is slightly better than its DPF-1 except that a one time initiation delay around 1.5 seconds is needed to initiate the ontology model. This overhead is incurred by Jena to read in and analyze the protocol ontology (*schema* and *instance* RDFS file) which contain ontology definition about layers and protocols (1313 ms) and to create inference engine (250 ms). Despite such an initiation overhead, ontology-based DPF-2 can offer better expressiveness and flexibility than a Java inspection-based one as we have explained earlier. The time for creation of SyncMaster and SyncSlave is largely spent on establishing control channel between sender and receiver. Such control channel is launched using RMI (*Remote Method Invocation*) [19] registry

Figure 7. Initiation delay of ontology model



technology which is a server-side name service that allows RMI clients to get a reference to the server object. In DPF-2, this RMI registry is used for SyncMaster to announce its presence and for SyncSlave to retrieve and register with it. These two processes take 453 ms and 250 ms respectively. Comparatively, DPF-1 leverages on JINI [20] technology to publish and discover SyncMaster which takes 484/320 ms to finish. Hence the invocation delay of RMI and JINI are of the same level in our test.

In DPF-1, the discovery of appropriate protocols from the protocol registry (a java class that records protocol properties) is achieved through introspection and takes half second. In DPF-2, such retrieval is performed by SPARQL [16] query language built-in Jena. We found that each query can be answered at the average rate of a few tens of milliseconds. Protocol dependency and compatibility is similarly checked by querying relevant information from inference engine, which can also be completed within a few milliseconds.

We have also conducted a series of experiments to further evaluate the performance of ontology model over different scales of dataset. The size of dataset is measured in term of the number of RDF triples, each of which represents a single statement (S-V-O predicate). These triples are generated within RDFS inference engine by parsing and merging RDF class definition and instances contained in the protocol ontology base. We have focused on the loading and merging time of the protocol ontology which also involves checking the ontologies for inconsistencies and generating of RDF triples. Figure 7 shows the results of experiments on PCs and laptops of different CPU speeds. It's not surprising to see that the initiation delay of RDF inference engine is proportional to the size of input dataset. However, as increments are linear, the delay is still acceptable even for a large dataset of 2261 triples (corresponding to 302 RDFS classes, which are sufficiently large enough for modeling more than 100 protocols – created for testing purpose only); the initiation delay also depends on the CPU speed. A PC with higher CPU clock speed will require less time to prepare the inference engine. However, the difference in performance is not significant. On the other hand, we have found that after preparation, the inference engine can answer a query within tens of milliseconds in both small and large scale dataset settings. All these results suggest that it is feasible to employ RDF based ontology model to manage protocol knowledge even on less powerful hosts such as laptops.

6. Conclusions and future work

In this paper, we have presented a formal ontology-based methodology to represent, access and retrieve protocol information. Protocol knowledge is modeled to (1) semantically select appropriate protocols that meet application requirements, (2) validate the composition of a protocol stack for build-time construction and runtime re-composition, and (3) facilitate QoS information exchange among end-to-end QoS components within the scope of our QCMF framework.

Based on the ontology model for layers, protocols and protocol stacks, an adaptive middleware can be designed to support dynamic discovery of protocols, building/validation and runtime re-composition of the protocol stack. The evaluation results demonstrate a reasonable performance of our system on desktops. On the other hand, we observe ontology processing a quite time-consuming process, which may not be appropriate for resource-constrained devices such as PDA or handset. We plan to introduce a set of proxies in the future to take over the ontology processing task in the case of mobile multimedia streaming. The design of proxy architecture and efficient distribution of proxies for multiparty communications (e.g., distant conferencing) should also be carefully studied.

References

[1] Liming An, Hung Keng Pung, Lifeng Zhou, "Design and Implementation of a Dynamic Protocol Framework", *Journal of Computer Communications*, 2005

[2] Hung Keng Pung, Liming An, "System and Method For A Dynamic Protocol Framework", *US patent 2005/0238050 A1*, October 2005

[3] Pearson Malcom E, "Dynamic layered protocol stack", *US patent US5903754*, 1999

[4] Gerard Parr, Kevin Curran, "A Paradigm Shift in the Distribution of Multimedia", *Communications of the ACM*, Vol. 43, No. 6, June 2000

[5] LiFeng Zhou, Hung Keng Pung, Lek Heng Ngoh, Tao Gu, "Knowledge Modeling for End-to-End QoS Provisioning", *Proc. 1st International Conference on Telecommunications and Networking in China (ChinaCom)*, 2006

[6] Zoubir Mammeri, "Towards a Formal Model for QoS Specification and Handling in Networks", *Proc.*

International Workshop on Quality of Service (IWQoS), 2004

[7] Chen Zhou, Liang-Tien Chia, and Bu-Sung Lee, "QoS Measurement Issues with DAML-QoS Ontology", *Proc. IEEE International Conference on e-Business Engineering (ICEBE),* 2005

[8] Glen Dobson, Russell Lock, Ian Sommerville, "QoSOnt: an Ontology for QoS in Service-Centric Systems", *Proc. UK e-Science AHM*, 2005

[9] E. Michael Maximilien, Munindar P. Singh, "A Framework and Ontology for Dynamic Web Services Selection", *IEEE Internet Computing*, 8(5):84-93, September-October 2004

[10] Dan Brickley, R.V. Guha, "RDF Vocabulary Description Language 1.0: RDF Schema", *World Wide Web Consortium*, January 2003

[11] Kevin Curran, Gerard Parr, "A middleware architecture for streaming media over IP networks to mobile devices", *Proc. IEEE Wireless Communications and Networking Conference (WCNC)*, 2003

[12] I. Sora, etc. "Policies for dynamic stack composition", *Technical Report*, Dept. of CS, Leuven, Belgium, 2001

[13] HP lab, "Jena 2 - A Semantic Web Framework", http://www.hpl.hp.com/semweb/jena2.htm

[14] Vangelis Gazis, etc,. "Metadata Design for Reconfigurable Protocol Stacks in Systems Beyond 3G", *Wireless Personal Communications*, January 2006

[15] S. Blake et al., "An Architecture for Differentiated Services," *RFC 2475*, December, 1998

[16] Eric Prud hommeaux, etc., "SPARQL Query Language for RDF", *World Wide Web Consortium* http://www.w3.org/TR/rdf-sparql-query/

[17] R. Braden et al., "Resource Reservation Protocol (RSVP) Version 1 Functional Specification", *RFC 2205*, September 1997

[18] Sun Microsystems, Inc., "Java Media Framework API", http://java.sun.com/products/java-media/jmf/

[19] Sun Microsystems, Inc., "Java Remote Method Invocation", http://java.sun.com/products/jdk/rmi/

[20] Sun Microsystems, Inc., "Jini Network Technology", http://www.sun.com/software/jini/

[21] Deborah L. McGuinness, Frank van Harmelen, "OWL Web Ontology Language Overview", *World Wide Web Consortium*, Feburary 2004