QoS-Assured Service Composition in Managed Service Overlay Networks*

Xiaohui Gu, Klara Nahrstedt
Department of Computer Science
University of Illinois at Urbana-Champaign
{xgu, klara}@ cs.uiuc.edu

Rong N. Chang, Christopher Ward
Network Hosted Application Services
IBM T.J. Watson Research Center
{rong, cw1}@ us.ibm.com

Abstract

Many value-added and content delivery services are being offered via service level agreements (SLAs). These services can be interconnected to form a service overlay network (SON) over the Internet. Service composition in SON has emerged as a cost-effective approach to quickly creating new services. Previous research has addressed the reliability, adaptability, and compatibility issues for composed services. However, little has been done to manage generic quality-of-service (QoS) provisioning for composed services, based on the SLA contracts of individual services. In this paper, we present QUEST, a QoS assUred composEable Service infrasTructure, to address the problem. QUEST framework provides: (1) initial service composition, which can compose a qualified service path under multiple QoS constraints (e.g., response time, availability). If multiple qualified service paths exist, QUEST chooses the best one according to the load balancing metric; and (2) dynamic service composition, which can dynamically recompose the service path to quickly recover from service outages and QoS violations. Different from the previous work, QUEST can simultaneously achieve QoS assurances and good load balancing in SON.

1 Introduction

The Internet has evolved to become a commercial infrastructure of service delivery instead of merely providing host connectivity. Different forms of *overlay networks* have been developed to provide attractive service provisioning solutions, which are difficult to be implemented and deployed in the IP-layer, such as content delivery overlays [1] and peer-to-peer file sharing overlays [2]. Beyond this, we envision the emergence of *service overlay networks* (SON), illustrated by Figure 1. Each SON node provides not only application-level data routing but also a set of value-added services (e.g., media transcoding, data encryption). Each service component is offered via a *service level agreement* (SLA) contract [15]. *Service composition* in SON has become necessary in order to cost-effectively create new Internet services [8]. Thus, a challenging problem is to *provide a service infrastructure to enable efficient service composition with quality-of-service (QoS) assurances*.

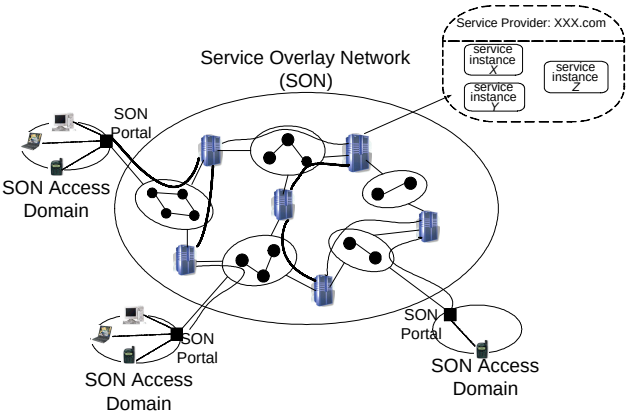

Figure 1. Illustration of the Service Overlay Network Model.

Much research work has addressed the service composition problem. The SAHARA project [8] addressed the *fault-resilience* problem in wide-area service composition. The CANS project [9] addressed the *adaptability* problem in service composition. The SPY-Net [17, 18] framework addressed the problem of *resource contention* while finding a multimedia service path. In the Gaia project [11], we addressed the QoS consistency and load partition issues for composing service path in ubiquitous computing environments. However, little has been done to support generic

*This work was supported in part by the NASA grant under contract number NASA NAG 2-1406, National Science Foundation under contract number 9870736, 9970139, and EIA 99-72884EQ. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation (NSF), NASA or U.S. Government.

QoS provisioning for composed services, based on the SLA contracts of individual service components.

In this paper, we present QUEST, a QoS assUred composEable Service infrasTructure, which can provide both *QoS assurances* under multiple QoS constraints, and *load balancing* in SON. Service composition is performed by the *service composer (SC)* using a composed service provisioning protocol. The protocol is designed based on a network-centric client-SERVICE model [4]. Instead of contacting service providers directly, the client contacts *SC* through a well-known address and specifies its desired services and QoS requirements. Then, *SC*, as an intermediate agent, composes and instantiates a qualified service path in SON. The key algorithms used by *SC* include: (1) *initial service composition*, and (2) *dynamic service composition*. The initial service composition algorithm properly chooses and composes the service instances, based on their SLA contracts and current performances in order to best match the QoS constraints of the user. QUEST achieves not only *QoS assurances* but also *load balancing* in SON by comprehensively considering both QoS and resources of different service instances. Moreover, QUEST provides *dynamic service composition*, which is used during runtime when service outages or QoS violations occur. The algorithm finds an alternative service path in SON, which can quickly recover the failed composed service delivery.

Extensive simulation results show that QUEST can provide much better QoS assurances and load balancing for composed services in SON than other common heuristics. The rest of the paper is organized as follows. Section 2 introduces the overall system design. Section 3 describes the design details and the key service composition algorithms. Section 4 presents the performance evaluations. Section 5 discusses related work. Finally, the paper concludes in Section 6.

2 System Overview

In QUEST, SON consists of various service components[1], called SON nodes. Each SON node represents a service component, which is managed by individual *component service provider* (CSP). The connections between SON nodes are called *SON links*, which are application-level virtual links. We assume that each CSP can manage and control the quality levels of its own services in accordance to the SLA contract with the *portal service provider* (PSP). On the other hand, the PSP has an SLA contract with the user for each composed service. In order to allow PSP to monitor and manage the quality levels of the composed service, we introduce SON portals that serve as the entrance/exit points of the SON. In QUEST, SON portals define the management boundary of QoS provisioning for composed applications. We call such a SON with SON portals a *managed* SON.

[1] several service instances can co-located on the same physical host.

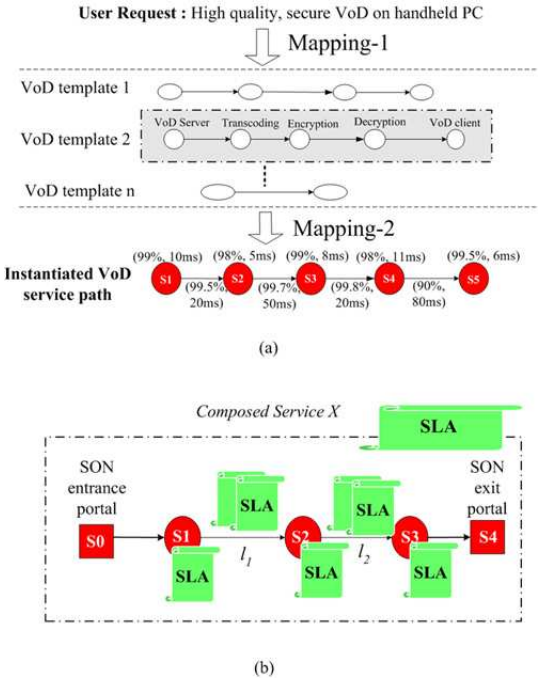

Figure 2. Illustration of the QUEST's service composition model.

The QUEST service composition model includes two mapping steps, illustrated by Figure 2 (a). For each user request, the *service composer* (SC) first maps it to a composite service template, and then maps the template to an instantiated service path. The mapping from the user request to different composite service templates (*mapping-1*) is constrained by the user's application-specific quality requirements and different pervasive client devices, such as PDAs and cell-phones. Mapping-1 has been addressed by several research work [11, 9]. It can be performed based on the application-specific QoS specifications [12] or using automatic composition plan tools [13]. The mapping from the composite service template to an instantiated service path (*mapping-2*) is constrained by distributed performance (e.g., response time) and resource availability conditions. Little research has addressed the mapping-2 that is the focus of this paper.

In QUEST, the composed service delivery, within SON, starts from the entrance portal, traverses through the chosen service instances, ends at the exit portal, illustrated by Figure 2 (b). We assume that each service instance s_i (or overlay link ℓ_i) is associated with an SLA contract specifying its QoS assurances such as *response time*

RT_{s_i} (or *delay* D_{ℓ_i}) and *availability* A_{s_i} (or A_{ℓ_i}). The *availability* is calculated by $(scheduled\ service\ time - total\ period\ of\ unavailability)/scheduled\ service\ time$. The connection between two service instances is called the *service link* l_i, which is mapped to an overlay routing path $l_i = \Gamma : \ell_1 \rightarrow ... \rightarrow \ell_\gamma$. The *availability* A_{l_j} of the service link (l_j) is calculated as $\prod_{\ell_i \in \Gamma} A_{\ell_i}$; and the *delay* D_{l_j} of the service link (l_j) is derived using $\sum_{\ell_i \in \Gamma} D_{\ell_i}$. Each CSP provides an SLA contract to the PSP and is responsible for managing its own service components or overlay links. The composed service X is also associated with an SLA contract specifying the QoS assurance that the PSP promises to the user.

3 Initial Service Composition

In this section, we present the *initial service composition* solution, which are used during the setup phase of the composed service delivery. Suppose each service instance s_i (or service link l_j) is offered to the PSP via an SLA contract specifying its provisioning QoS: *availability* A_{s_i} (or A_{l_j}) and *response time* RT_{s_i} (or *delay* D_{l_j}). The QoS assurances (availability: A_x and response time: RT_x) of the composed application X: $s_0 \rightarrow s_1 ... \rightarrow s_{n+1}$ can be derived as follows [2],

$$ln\frac{1}{A_x} = \sum_{i=0}^{n+1} ln\frac{1}{A_{s_i}} + \sum_{j=0}^{n} ln\frac{1}{A_{l_j}} \quad (1)$$

$$RT_x = \sum_{i=0}^{n+1} RT_{s_i} + \sum_{j=0}^{n} D_{l_j} \quad (2)$$

$where\ s_0 : entrance\ portal;\ s_{n+1} : exit\ portal.$

Suppose also that the PSP has an SLA contract with the user to specify its provisioning QoS for the composed service *X* (*availability* A_x^{target} and *response time* RT_x^{target}). In order to guarantee a successful service delivery, the resource requirements of the instantiated service path have to be satisfied. For simplicity, we only consider the CPU resource for end hosts and bandwidth for overlay links. We define the term *cpu ratio* for the service instance s_i as $CR_{s_i} = cpu_{s_i}^{required}/cpu_{s_i}^{available}$. The $cpu_{s_i}^{required}$ represents the required CPU resource for running a new s_i process. The $cpu_{s_i}^{available}$ represents the available CPU resource in the physical hosting environment of s_i. If $CR_{s_i} \leq 1$, then the service request can be admitted. Otherwise, the service request will be denied. The smaller the CR_{s_i}, the more advantageous we choose the service instance in terms of CPU load balancing because we start the new service process on a lightly loaded host. Similarly, we define the term *bandwidth ratio* for the service link l_j as $BR_{l_j} = bandwidth_{l_j}^{required}/bandwidth_{l_j}^{available}$. With the above notations, we can formulate the QoS-assured service composition problem as follows,

DEFINITION 1. QoS-assured Service Composition (QSC) Problem *Suppose we are given a directed graph representing a service overlay network (SON) topology, G = (V, E), where V and E are the sets of N SON nodes and M SON links, respectively. Suppose also each SON node* s_i *is characterized by nonnegative values of 2 additive QoS attributes* ($ln\frac{1}{A_{s_i}}$, RT_{s_i}), *i = 1...N. Each SON link* ℓ_i *is also characterized by nonnegative values of 2 additive QoS attributes* ($ln\frac{1}{A_{\ell_i}}$, D_{ℓ_i}), *i = 1...M. Given the template for a composed service X and user QoS requirements* $ln\frac{1}{A_x^{target}}$ *and* RT_x^{target}, *the problem of QoS-assured service composition is to compose a service path* $p \triangleq s_1 \rightarrow s_2 ... \rightarrow s_n$ *from* s_0 *(entrance portal) to* s_{n+1} *(exit portal), such that* $ln\frac{1}{A_x^{target}} \leq ln\frac{1}{A_x}$ *(equ.(1)) and* $RT_x^{target} \leq RT_x$ *(equ. (2)), and also* $CR_{s_i} \leq 1$, *i = 0... n+1, and* $BR_{l_j} \leq 1$, *j = 0...n.*

We now prove that the QSC problem is NP-complete.

Theorem 1 *QSC problem is NP-complete.*

Proof: We prove this by showing that the *Multiple Constrained Path selection (MCP)* problem, which is known to be NP-complete [10] maps directly to a special case of the QSC problem. The detailed proof is omitted due to the page limitation. □

Besides its NP-completeness, the above QSC problem definition also neglects several important practical issues. First, in the real world SLA contract, the QoS attributes are often specified using *average* values measured over a *long time period* such as a month or a year [15]. However, the composed service session can last for only several minutes or hours. Hence, we need to consider not only SLA contract values but also recent performance conditions of service instances/links. Second, the PSP can concurrently serve thousands of user requests by composing available service instances. To achieve best aggregate QoS assurances for *all* SON users, we also need to consider the *load balancing* problem. Third, because SON is highly dynamic, QoS violations can happen sometimes. Commercial SLA contracts usually make the PSP lose money when QoS violation happens [15]. Hence, the goal of our QSC algorithms is to compose a service path that can best avoid QoS violations or minimize the QoS violation degree if a violation occurs.

We now provide a polynomial heuristic algorithm, called *QSC-basic*, for the QSC problem. The basic idea is to use a modified Dijkstra algorithm by comprehensively considering multiple constraints (e.g., SLA contracts, recent performance, system load). The *QSC-basic* primarily involves two steps: (1) *Generate the weighted candidate*

[2] In order to make the metric *availability* becomes *additive* and *minimum-optimal*, we apply the logrithm and inverse operations on the metric *availability*. We assume that RT_{s_i} and D_{l_j} are measured for an application data unit (e.g., a video frame).

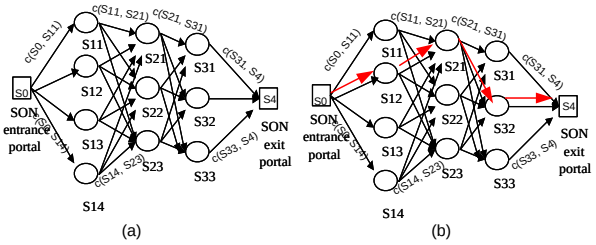

Figure 3. Illustration of the *QSC-basic* algorithm.

graph. Instead of searching the service path in the entire SON, we first generate a candidate graph, illustrated by Figure 3 (a), to minimize the searching range. The ith column in the candidate graph includes all the service instance candidates providing the ith service function in the application template. The "cost value" on the edge $l_k = (s_i, s_j) \in E, s_i, s_j \in V$ is defined using the following integrated metric:

$$c(l_k) = \underbrace{\frac{ln\frac{1}{A_{l_k}}}{ln\frac{1}{A_l}^{max}} + \frac{ln\frac{1}{A_{s_j}}}{ln\frac{1}{A_s}^{max}} + \frac{D_{l_k}}{D^{max}} + \frac{RT_{s_j}}{RT^{max}}}_{average\ performance\ in\ SLA\ contracts}$$
$$+ \underbrace{\frac{ln\frac{1}{A_{l_k}}^{recent}}{ln\frac{1}{A_l}^{max}} + \frac{ln\frac{1}{A_{s_j}}^{recent}}{ln\frac{1}{A_s}^{max}} + \frac{D_{l_k}^{recent}}{D^{max}} + \frac{RT_{s_j}^{recent}}{RT^{max}}}_{recent\ performance}$$
$$+ \underbrace{\frac{CR_{s_j}}{CR^{max}} + \frac{BR_{l_k}}{BR^{max}}}_{load\ conditions} \quad (3)$$

Intuitively, the ratio $\frac{\alpha}{\alpha^{max}}$ (α represents any of the above parameters) represents the *normalized cost* of selecting a portion of the service path in terms of one specific factor (e.g., QoS assurances or load balancing); and (2) *Run the Dijkstra algorithm to find the shortest path*, which is returned as the result of the QoS-assured service composition, illustrated by Figure 3 (b).

However, the above *QSC-basic* algorithm does not consider each individual QoS constraint while composing a service path. We now present an enhanced algorithm, called *QSC-enhanced*. After generating the candidate graph, we associate each edge $l_k = (s_i, s_j) \in E, s_i, s_j \in V$ with a "cost value", which modifies the equation (3) by multiplying a non-negative value w_i $(0 \leq i \leq 9, \sum_{i=0}^{9} w_i = 1)$ with each ratio $\frac{\alpha}{\alpha^{max}}$. The weight w_i represents the significance of the $i'th$ factor while selecting the service instance during each EXTRACT_Min step in the Dijkstra algorithm. The higher the w_i value, the more important the $i'th$ factor. Different from the *QSC-basic*, the *QSC-enhanced* algorithm dynamically changes the importance of different factors by adjusting w_i accordingly. The adjustment of w_i is based on the "pressure" of different QoS constraints. Intuitively, if the current accumulated value of a QoS attribute (e.g., response time) approaches its constraint, we increase its weight in hope that its accumulation will catch up in the later stage of the service path composition. Suppose s_i is the current chosen node by Extract_Min, whose final shortest path from the source s_0 is just determined. We define the response time pressure "P^{RT}", availability pressure "P^{avail}", and the *weight adjustment functions* as follows,

$$P^{RT} = \frac{RT_{s_0 \to s_i}}{RT_x^{target}},\ P^{avail} = \frac{ln\frac{1}{A_{s_0 \to s_i}}}{ln\frac{1}{A_x^{target}}} \quad (4)$$

$$w_0 = w_1 = w_4 = w_5 = \frac{1}{4} \cdot \frac{P^{avail} \cdot (1 - w_8 - w_9)}{P^{avail} + P^{RT}} \quad (5)$$

$$w_2 = w_3 = w_6 = w_7 = \frac{1}{4} \cdot \frac{P^{RT} \cdot (1 - w_8 - w_9)}{P^{avail} + P^{RT}} \quad (6)$$

Both *QSC-basic* and *QSC-enhanced* algorithms have the same computational complexity $O(K^2)$, where K is the number of nodes in the candidate graph.

4 Dynamic Service Composition

SON is a highly dynamic system compared to the IP network infrastructure. First, unlike routers, hosts can dynamically join or leave SON over long time scales. Second, hosts or underlying IP network path can experience performance failures, outages, or degradations over short time scales [3]. Hence, during runtime, an established service path can become broken or violate QoS constraints, particularly for the long-lived application session such as multimedia streaming.

When the service instance (or link) experiences outage or significant quality degradations, the *SC* is notified. It recovers all the affected sessions using the *dynamic service composition* algorithms. We have designed two dynamic service composition algorithms: (1) *DQSC-complete*, which *completely* re-composes the service path without considering the original service path, to recover from failures; and (2) *DQSC-partial*, which *partially* re-composes the service path based on the original service path.

Figure 4 illustrates the complete service re-composition algorithm *DQSC-complete*. Figure 4 (a) shows the candidate graph and the original service path, on which the service instance s_{12} is failed and the service link between s_{21} and s_{31} is broken. The *DQSC-complete* algorithm first modifies the candidate graph by removing those failed or poorly-performing service instances, and also replacing the broken service links with alternate SON routing paths when it is possible, illustrated by Figure 4 (b). In this example, s_{12} is removed from the second column of the candidate graph. The failure service link between s_{21} and s_{31} is replaced with an alternative SON path. The recovered service link is illustrated as a dotted line in Figure 4 (b). Then, we use

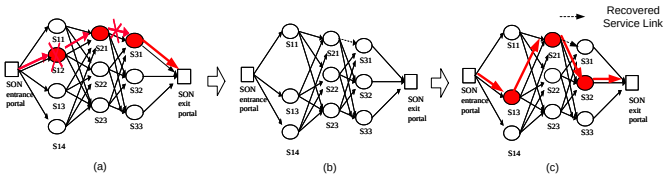

Figure 4. Illustration of the complete service re-composition algorithm *DQSC-complete*.

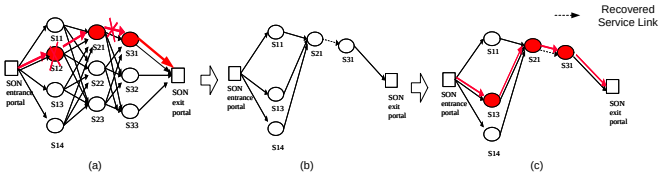

Figure 5. Illustration of the partial service re-composition algorithm *DQSC-partial*.

the *QSC-enhanced* algorithm, described in Section 3.1, to compose a new qualified service path. Thus, the service session can quickly recover from failures or QoS violations by switching from the old service path to the new one [3].

Contrasting with the *DQSC-complete* algorithm, *DQSC-partial* algorithm *partially* re-composes the service path based on the original service path. Figure 5 (a) shows the same service path example as Figure 4 (a). In the original service path *SON entrance portal* $\rightarrow s_{12} \rightarrow s_{21} \rightarrow s_{31} \rightarrow$ *SON exit portal*, service instance s_{12} is failed and service link between s_{21} and s_{31} is broken. In Figure 5 (b), however, we modify the candidate graph by not only removing the poorly-performing or failed service instances (e.g., s_{12}) and recovering the broken service links (e.g., the service link between s_{21} and s_{31}), but also removing, in the column where the old service instance is good, the other candidate service instances. In this example, s_{21} and s_{31} still perform well. Thus, we remove s_{22} and s_{23} in the third column of the candidate graph, and s_{32} and s_{33} in the fourth column. Then, we use the *QSC-enhanced* algorithm to compose a new service path on the modified candidate graph, illustrated by Figure 5 (c). The purpose of such an approach is to keep those original well-performing service instances in the new service path. Hence, we can reduce the migration overhead for switching from the old service path to the new one. The computational complexity of *DQSC-complete* and *DQSC-partial* is still $O(K^2)$, where K is the number of nodes in the candidate graph.

Both *DQSC-complete* and *DQSC-partial* algorithms can quickly re-compose a new service path to recover from failures or QoS violations. However, each of them has both advantages and disadvantages. The advantage of the *DQSC-complete* algorithm is that it can re-compose a better service path in terms of QoS assurances than the *DQSC-partial* algorithm, since it has more choices of service instances. The disadvantage of the *DQSC-complete* algorithm is that the service re-composition takes longer time since it re-composes the entire service path. On the other hand, the advantage of the *DQSC-partial* algorithm is that it is quicker and easier to implement since it only changes part of the service path. However, its disadvantage is that the new composed service path may not be optimal. We will further compare these two different dynamic service composition approaches in the next section.

[3] We assume that the states of all the service instances can be recovered by software.

5 Performance Evaluation

5.1 Evaluation methodology

We evaluate the performance of the initial and dynamic QoS-assured service composition algorithms using extensive simulations. We first use the degree-based Internet topology generator Inet 3.0 [16] to generate a power-law random graph topology with 3200 nodes to represent the Internet topology. We then randomly select 500 nodes as the SON nodes and 40 other nodes as the SON portals. We assume an equal-degree random graph topology for the SON. Each SON node is randomly assigned 5 other SON nodes as its neighbors. Hence, the probing overhead of each SON node is within $5/500 = 1\%$.

The initial resource availability of each IP link and service instance is uniformly distributed in a certain range. The SLA values of each IP link or service instance are also uniformly distributed within certain range. Different values reflect the heterogeneity and diversified quality guarantees in SON. Moreover, to simulate the performance variation in the real world, the QoS attributes of each IP link and service instance are set by uniform distribution functions, with SLA values as the mean values. We assume the Dijkstra shortest path algorithm for both the IP layer and overlay layer routing, using the instantaneous value of delay as the routing metric. The bandwidth of an overlay link is the bottleneck bandwidth along the IP network path. The delay of an overlay link is the addition of the delays along the IP network path.

During each minute, certain number of user requests are generated. The user request is represented by any of 40 composite service templates that comprise 2 to 6 services. Each user session lasts from 15 to 60 minutes. The metrics we use for evaluating the QoS assurances include *QoS violation rate* and *QoS violation degree*. The *QoS violation rate* is measured by the ratio of the sessions during

which QoS violation happens over the total sessions. For each session, the QoS violation is said to happen if the measured average QoS attribute values (i.e., availability, response time) is worse than that specified in the SLA contract. The *QoS violation degree* measures that if a QoS violation occurs, how severe the QoS violation is. It is measured by the ratio of difference between the measured QoS attribute value and its target value, over the target value. Those two metrics are often associated with the financial refund/penalty policies specified in the real world SLA contracts. The minimization of those two metrics means to reduce the financial loss of the service provider.

The metric we use for evaluating the load balancing is the *provisioning success rate*. A composed service provisioning is said to be successful if and only if during its entire session, all the service instances and links' resource requirements on the service path are always satisfied. The *composed service provisioning success rate* is defined as the number of successful requests over the total number of all requests. Given the total amount of resource in SON, higher *provisioning success rate* represents better load balancing in SON.

For comparison, we also implement two common heuristic algorithms for composing service path: *fixed* and *random* algorithms.The *fixed* algorithm always chooses the same service instances for a composed application. The *random* algorithm randomly chooses service instances to compose the service path.

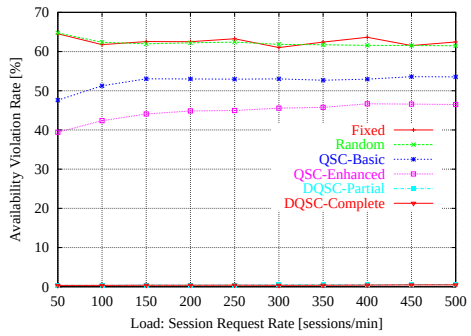

Figure 6. Average availability QoS violation rate under different system load.

5.2 Results and analysis

Figure 6 and Figure 7 show the simulation results about the violation rates of two QoS attributes: *availability* and *response time*, respectively. In Figure 6, the X axis represents different *session request rate*, calculated by the number of composed service session requests per minute. The range of *session request rate* is selected to reflect

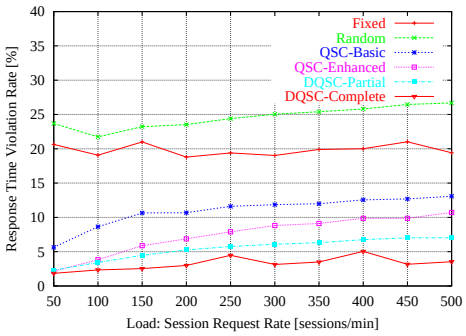

Figure 7. Average response time QoS violation rate under different system load.

different system workload put on the SON. The Y axis shows the average QoS violation rate for the *availability* attribute, achieved by the *fixed*, *random* and our four QSC (QoS-assured service composition) algorithms. *QSC-Basic* and *QSC-Enhanced* represent the two initial service composition algorithms. Both of them do not include any dynamic service re-composition mechanisms. Both *DQSC-Complete* and *DQSC-Partial* use the *QSC-Enhanced* for the initial service composition and also dynamically recovers from the service outage/quality degradations by completely or partially re-composing the service path. Each average availability QoS violation rate (Ψ_1) value is calculated and averaged over a period of 200 minutes for all successfully composed sessions. The results show that all the four QSC algorithms achieve much lower Ψ_1 than the *fixed* and *random* algorithms. The *QSC-Enhanced* has as much as 20% improvements than the *QSC-Basic*. Both *DQSC-Complete* and *DQSC-Partial* further reduce (Ψ_1) to almost 0% lower. The reason is that the application-level service outage recovery can quickly finish in a few seconds while the IP-layer Internet path recovery may take several minutes or even hours [3]. However, the performance difference between *DQSC-Complete* and *DQSC-Partial* is very small.

Similarly, Figure 7 shows the results of the QoS violation rate for the *response time* (Ψ_2). Again, the QSC algorithms achieve much lower Ψ_2 than *fixed* and *random*. The performance order of different QSC algorithms is *QSC-Basic* $\preceq$ (worse than) *QSC-Enhanced* $\preceq$ *DQSC-Partial* $\preceq$ *DQSC-Complete*. The reason why the DQSC algorithms are better than the *QSC-Enhanced* is that service re-composition always uses the most recent performance and load information, which allows it to make better service instance choices in terms of response time.

Figure 8 and Figure 9 show the results about the QoS violation degree. Figure 8 shows the availability QoS violation degree (Ψ_3). Once again, the QSC algorithms consistently achieve better performance than *fixed* and *random*. Figure

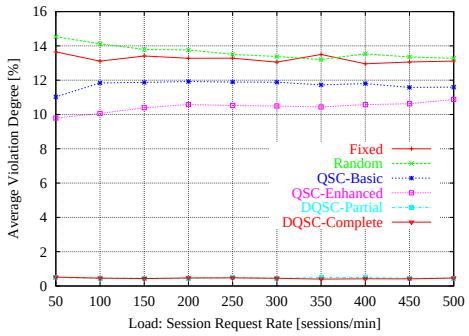

Figure 8. Average availability QoS violation degree under different system load.

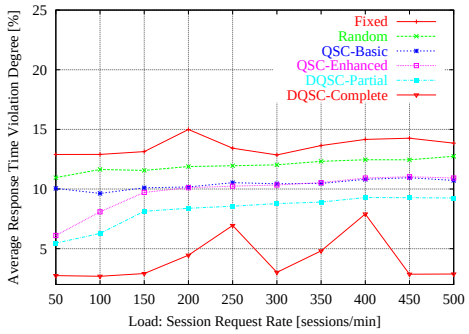

Figure 9. Average response time QoS violation degree under different system load.

9 shows similar results for the response time QoS violation degree. Hence, the simulation results further validate our algorithms by showing that QSC algorithms can not only greatly reduce the violation rate but also achieve lower violation degree when QoS violations occur.

Figure 10 shows the results about the composed service *provisioning success rate*. Similar to the above experiments, each *provisioning success rate* value is calculated and averaged over a period of 200 minutes. The results show that all four QSC algorithms can similarly achieve much higher *provisioning success rate*, namely better load balancing, than the *fixed* and *random*.

In all of the above experiments, we observe that the performance gains of the *DQSC-Complete* are very small compared to the *DQSC-Partial*. The reason is that the initial service composition algorithm *QSC-Enhanced* is already very good. Hence, the selected service instances in the old service path are still, by large probability, the best ones when we re-compose the service path. Hence, it is a near optimal solution that we keep the old good service instances in the new service path, which is exactly the *DQSC-Partial* algorithm.

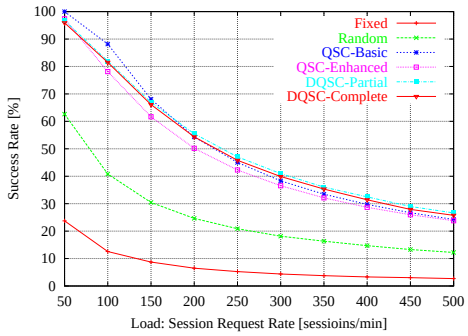

Figure 10. Average provisioning success rate under different system load (load balancing).

6 Related Work

Besides to the related work mentioned in the Introduction, much other research work has also addressed the service composition problem. The SWORD project at Stanford [13] provided a developer toolkit for the web service composition. It can automatically generate a functional composition plan given the functional requirements for the composed application. However, SWORD only addressed the mapping-1 problem defined in our framework. The eFlow project at HP labs [7] provided an adaptive and dynamic service composition mechanism for the commercial e-business process management. It did not address the end-to-end QoS assurances for the composed service. Our work is also different from the traditional IP-layer QoS routing problem [5] because: (1) The goal of IP-layer QoS routing is to find a network routing path satisfying QoS constraints while QUEST addresses two mapping problems to achieve not only QoS assurances but also load balancing and fault-tolerance; and (2) IP-layer QoS routing only considers the network resource while QUEST considers not only network resources but also end-system resources (e.g., CPU).

Other closely related work includes various overlay networks. Anderson et al. proposed a *resilient overlay network* (RON) architecture [3] to allow distributed applications to quickly detect and recover from the Internet path failure. RON can recover from network path outage within several seconds using the application-level routing. RON is useful and beneficial to QUEST although it only solved a subset of the dynamic service composition problems. In [6], Duan et. al. also proposed the concept of SON, which can provide value-added services with QoS assurances to the user via SLA contracts. However, they only addressed the bandwidth provisioning problem for SON, while QUEST considers not only resource provisioning (e.g., bandwidth and CPU), but also various service QoS (e.g., response time and availability). The OverQoS [14] proposed an architec-

ture to provide Internet QoS (e.g., statistical bandwidth and loss rate assurances) using overlay networks. QUEST is different from OverQoS by providing QoS assurances for composed services, based on SLA contracts of individual service components.

7 Conclusion

We have presented a QoS-assured composed service delivery framework, called QUEST, for a managed service overlay network (SON). The major contributions of this paper include: (1) formally define the QoS-assured service composition problem and prove that it is NP-complete. We then design efficient approximate optimal algorithms to compose service paths under multiple QoS constraints. Moreover, QUEST can achieve sound load balancing in SON to provide best possible QoS for all SON users; (2) provide both partial and complete dynamic service re-composition algorithms, which can quickly recover the service path from failures or QoS violations. We have implemented a large-scale simulation test-bed and our extensive simulation results show that QUEST can provide both QoS assurances and load balancing for composed services in SON. The simulation results also indicate that our partial dynamic service re-composition algorithm can achieve almost the same level of QoS assurance as the complete re-composition algorithm, but with much lower overhead.

8 Acknowledgment

We would like to thank Dr. Eric Wu at IBM T.J. Watson research center for his helpful input to our work. We wish to thank anonymous reviewers for their helpful suggestions.

References

[1] Akamai Inc. *http://www.akamai.com/*.

[2] Gnutella. *http://gnutella.wego.com/*.

[3] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient Overlay Networks. *Proc. of 18th ACM SOSP 2001, Banff, Canada*, October 2001.

[4] R. Chang and C. Ravishankar. A Service Acquisition Mechamism for Server-Based Heterogeneous Distributed Systems. *IEEE Transactions on Parallel and Distributed Systems, 5(2)*, February 1994.

[5] S. Chen and K. Nahrstedt. An Overview of Quality-of-Service Routing for the Next Generation High-Speed Networks: Problems and Solutions. *IEEE Network Magazine, Special Issue on Transmission and Distribution of Digital Video, 12(6), pp. 64-79*, 1998.

[6] Z. Duan, Z.-L. Zhang, and T. Hou. Service Overlay networks : SLAs, QoS and Bandwidth Provisioning. *Proc. of 10th IEEE International Conference on Network Protocols(ICNP2002), Paris, France*, November 2002.

[7] B. Raman et. al. The SAHARA Model for Service Composition Across Multiple Providers. *Proc. of International Conference on Pervasive Computing (Pervasive 2002)*, Aug 2002.

[8] F. Casati et. al. Adaptive and Dynamic Service Composition in eFlow. *HP technical Report HPL-2000-39*, March 2000.

[9] X. Fu, W. Shi, A. Akkerman, and V. Karamcheti. CANS: Composable, Adaptive Network Services Infrastructure . *Proc. of 3rd USENIX Symposium on Internet Technologies and Systems*, March 2001.

[10] M. R. Garey and D. S. Johnson. Computers and Intractability. *A Guide to the Theory of NP-Completeness, Freeman, San Francisco*, 1979.

[11] X. Gu and K. Nahrstedt. Dynamic QoS-Aware Multimedia Service Confi guration in Ubiquitous Computing Environments. *Proc. of IEEE 22nd International Conference on Distributed Computing Systems (ICDCS 2002)*, July 2002.

[12] X. Gu, K. Nahrstedt, W. Yuan, D. Wichadakul, and D. Xu. An XML-based Quality of Service Enabling Language for the Web. *Journal of Visual Language and Computing, Special Issue on Multimedia Language for the Web, 13(1), pp. 61-95*, February 2002.

[13] S.R. Ponnekanti and A. Fox. SWORD: A Developer Toolkit for Building Composite Web Services. *Proc. of the Eleventh World Wide Web Conference (Web Engineering Track), Honolulu, Hawaii*, May 2002.

[14] L. Subramanian, I. Stoica, H. Balakrishnan, and R. H. Katz. OverQoS: Offering QoS using Overlays. *Proc. of First Workshop on Hop Topics in Networks (HotNets-I), Princeton, New Jersey*, October 2002.

[15] C. Ward, M. Buco, R. Chang, and L. Luan. A Generic SLA Semantic Model for the Execution Management of e-Bussiness Outsourcing Contracts. *Proc. of 3rd International Conference on e-Commerce and Web Technologies (EC-Web 2002)*, September 2002.

[16] J. Winick and S. Jamin. Inet3.0: Internet Topology Generator. *Tech Report UM-CSE-TR-456-02 (http://irl.eecs.umich.edu/jamin/)*, 2002.

[17] D. Xu and K. Nahrstedt. Finding Service Paths in a Media Service Proxy Network. *Proc. of SPIE/ACM Multimedia Computing and Networking Conference, San Jose, CA*, January 2002.

[18] D. Xu, K. Nahrstedt, and D. Wichadakul. QoS and Contention Aware Multi-Resource Reservation. *Cluster Computing, the Journal of Networks, Software Tools and Applications, 4(2), Kluwer Academic Publishers*, 2001.