QoS-Aware Service Composition in Large Scale Multi-Domain Networks

Jin Xiao
School of Computer Science
University of Waterloo
200 University Ave. W.
Waterloo, ON, Canada
j2xiao@bbcr.uwaterloo.ca

Raouf Boutaba
School of Computer Science
University of Waterloo
200 University Ave. W.
Waterloo, ON, Canada
rboutaba@bbcr.uwaterloo.ca

Abstract

Next generation networks are envisioned to support dynamic and customizable service compositions at Internet scale. To facilitate the communication between distributed software components, on-demand and QoS-aware network service composition across large scale networks emerges as a key research challenge. This paper presents a fast QoS-aware service composition algorithm for selecting a set of interconnected domains with specific service classes. We further show how such algorithm can be used to support network adaptation and service mobility. In simulation studies performed on large scale networks, the algorithm exhibits very high probability of finding the optimal solution within short execution time. In addition, we present a distributed service composition framework utilizing this algorithm.

Keywords

Service composition and provisioning, network adaptation, multi-domain networks, end-to-end QoS assurance

1. Introduction

Next generation networks will deliver digital information at far greater capability and with far wider reachability than the Internet today. This advancement is envisioned to support advanced and customizable services with stringent QoS requirements. Its evolution coincides with a paradigm shift in distributed applications: dynamic composition of application/service from independent functional components discovered and composed on demand. The proliferation of Grid [7] and Web Service [4] technologies exemplifies this shift. To facilitate the communication between these components, on-demand QoS-aware network service composition across large scale networks emerges as a key research challenge.

To illustrate our problem, consider the following scenario: Alice in Canada wants to make a call to Bob in UK via IP telephony. Because this is a business call, Alice would like a digital copy of the conversation in Chinese for her associates in Hong Kong. A possible method of facilitating this interaction is to use the Session Initiation Protocol (SIP) [10]. Assuming both Alice and Bob have SIP-enabled phones, a SIP proxy that is capable of forking calls to multi-parties must be discovered, as well as a SIP-enabled recording service, a voice to text convertor, and an English-Chinese translator. Three processes must have taken place to grant Alice's wish. First, the components required to perform Alice's action must be discovered. Second, these components must be streamlined together and agree on communication protocols and data formats. Third, network communication paths must be established and monitored for QoS performance.

The first step involves the discovery of service components suiting Alice's criteria. Service discovery protocols such as Service Location Protocol [9], Secure Service Discovery Service [6], and works on Web Service discovery [1][11] could be used to realize this process. The second step involves the selection and composition of service components into a coherent unit. Recent research works on service composition [8][17][23] and work flow languages, such as Web Service Flow Language (WSFL) [14] and Business Process Execution Language for Web Service (BPEL4WS) [2], attempt to address this issue. The third step involves the establishment and monitoring of network paths interconnecting service components with QoS requirements. Due to the distributed nature of these components, such a network path often traverses multiple administrative domains (e.g. Alice is in Canada, Bob is in UK, and the proxy is in US). Given that each domain has its own QoS provisioning scheme (e.g. DiffServ [3], IntServ [22], etc.) and typically offers a set of QoS guarantees at varied prices, then contract-based negotiations can be conducted between neighboring domains to obtain the necessary QoS assurances in each domain and to permit traffic flow across domain boundaries [12][16][20]. However, before this process takes place, a proceeding question must be answered: what QoS assurance should be requested from each domain? This question also has an important implication in DiffServ networks. If two neighboring DiffServ domains each has a number of different QoS class descriptions, class remarking must take place at the border. How should this remarking be determined? To address this problem, it is perhaps better to answer a more general question: given point A and B in a multi-domain network with many interconnecting domains, what is the best combination of domains to choose, and what is the optimal allocation of QoS requirements among them?

In this paper, we first propose a fast QoS-aware multi-domain service composition algorithm. The QoS-aware aspect of the algorithm is two fold. One, the algorithm has high probability of finding a minimal cost communication path (at the domain level) that satisfies the QoS constraints requested by the service components. Two, on detection of QoS violations in a subset of the chosen domains, the algorithm can quickly recompose a new communication path that satisfies the original QoS requirements, while reuse as much of the old path as possible. We further show that our adaptation scheme can cope with service mobility. For scalability, our algorithm only relies on per-hop domain knowledge that can be readily extracted from BGP [18] information. A simple service composition framework utilizing this algorithm is then presented. The three major functions of the framework are: domain discovery, service composition, and network adaptation.

The rest of the paper is organized as follows. Section 2 details the service composition algorithm and section 3 covers the adaptation scheme. Section 4 presents our service composition framework, followed by simulation studies in section 5. Section 6 presents related works and section 7 concludes the paper.

2. Multi-Domain Service Composition

In this section, we first detail the creation of domain graph, which is an abstract representation of the domain connectivity and their service classes. Each service class is considered to have a set of QoS assurances and an associated cost. We focus on three common QoS factors in this paper: delay, availability, and bandwidth. In doing so, we reduce the problem to k-MCOP (k Multi-Constraint Optimization Problem), which is known to be NP-Complete [21]. We then describe the rationale and the design of our algorithm to solve this problem and discuss the runtime complexity.

2.1 Domain Graph Creation

Since an Autonomous System (AS) is the smallest granularity of administrative independence, we consider each AS as a domain. In this sense, each domain exchanges traffic flow with its neighboring domains via border gateways. We further consider each domain to have a set of service classes, each with a set of QoS assurances and a price. A domain without service class differentiations is assumed to provide a single set of QoS assurances and price.

First, we can abstract the domain connectivity information as an undirected graph, where the nodes of the graph represent the border gateway exchanges between neighboring domains and the edges of the graph represent the connectivity between border gateways in a domain. Figure 1 illustrates this transformation process. Inter-domain routing policies can be incorporated during this process. With such abstraction, it is natural to represent the QoS assurance set and its associated price as a set of weights on each edge. For example, if domain C offers a QoS class with minimum bandwidth BW_C, minimum availability A_C, maximum delay D_C, and price C_C, then the edge connecting its border gateways is assigned the weight set $\{D_C, C_C, A_C, BW_C\}$.

A domain may offer a set of QoS service classes offerings based on its policies. To incorporate these service classes into the domain graph, we perform edge expansion by introducing a number of "service nodes" corresponding to the number of service classes. One node of the original edge now connects to a service node via a new edge with a weight set representing one service class, and the other node of the original edge connects to the service node via a new edge with nil weight set $\{0, 0, 1, \infty\}$. Figure 2 illustrates the edge expansions involving domains A, B and C.

We observe that a path from node S to node D on the expanded domain graph not only represents a possible sequence of interconnecting domains between the two end

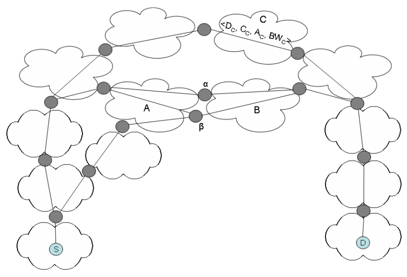


Figure 1: Graph Representation of Domain Connectivity

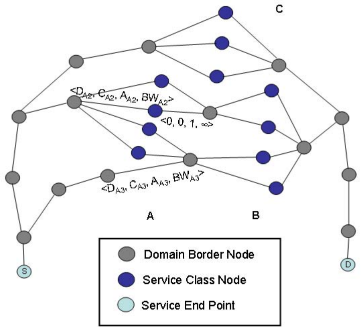

C
<D_{A2}, C_{A2}, A_{A2}, BW_{A2}>
<0, 0, 1, ∞>
<D_{A3}, C_{A3}, A_{A3}, BW_{A3}>
A
B
S
D
Domain Border Node
Service Class Node
Service End Point

Figure 2: Domain Graph with Service Class Expansion

points, but also depicts a selection of respective service classes in these domains. By traversing through all possible paths between S and D, we can exhaustively search all possible service compositions between them. Thus, it is possible to formulate our service composition problem as a graph search problem.

2.2 The Service Composition Algorithm

Our service composition problem can be stated as: given the expanded domain graph $G(V, E)$, find a path $P = (\omega_1, ..., \omega_k, \omega \in E)$ from node v_s to node v_d such that the end-to-end delay $\sum_{i=1...k} D_i$ is below delay constraint κ_D, the end-to-end availability $\prod_{i=1...k} A_i$ is above availability constraint κ_A, the bandwidth BW of all edges in P is above bandwidth constraint κ_{BW}, and the cost $\sum_{i=1...k} C_i$ is below cost constraint κ_C. Such a path is termed a feasible path. Then, a minimal feasible path is a feasible path whose cost is minimal among all feasible paths. Assuming κ_A does not equal to 1, and κ_D and κ_C not equal to 0, We can rewrite the above constraints as:

$$\tau_1 = \frac{\sum_{i=1...k} D_i}{\kappa_D}, \tau_2 = \frac{\sum_{i=1...k} C_i}{\kappa_C} \qquad (1)$$
$$\tau_3 = \frac{1 - \prod_{i=1...k} A_i}{1 - \kappa_A}, \tau_{4i} = \frac{\kappa_{BW}}{BW_i}$$

Thus, the service composition problem can be formalized as:

Given an undirected graph G(V,E) and two nodes in V (v_s and v_d), where each edge in E has weights $\{D, C, A, BW\}$, find a path P=$\{\omega_1$,...,ω_k, $\omega \in E\}$ connecting v_s and v_d such that $\tau_1, \tau_2, \tau_3 \leq 1$, $\tau_{4i} \leq 1$ for all ω_i, and $\sum_{i=1...k} C_i$ is minimal.

This is equivalent to the *k Multi-Constraint Optimization Problem* (k-MCOP), which is known to be NP-Complete [21]. A number of notable heuristics have been developed for this problem [5][13][15], but with our graph constructs, they either do not scale well with increasing number of nodes, or do not yield near optimal solutions. In this subsection, we propose a new heuristic algorithm with fast performance and good optimality rate.

Most of the existing heuristics for k-MCOP utilizes the Bellman-Ford or Dijkstra's algorithm for its simplicity. Our heuristic favors the Dijkstra's algorithm, as it relies on per-hop information in its search process. Two major issues are encountered in utilizing Dijkstra's algorithm to solve the k-MCOP problem. One, Dijkstra's algorithm uses greedy search strategy without backtracking (hence the fast runtime bound). However, for k-MCOP, it is often the case that the minimal cost path will violate one or more constraints that forces the algorithm to backtrack. Two, if a mapping function is used to transform the k weights into a single value, it is difficult to ensure the following requirements: a) the function produces smaller values for all feasible paths than the values it produces for infeasible paths. b) a path that minimizes such mapped value is also a minimal cost path.

Let $H(D, C, A)$ be a mapping function that satisfies requirement a), our algorithm first runs Dijkstra's algorithm from v_d to v_s by minimizing $H(D, C, A)$. The purpose of this reverse search step is to determine whether there is a feasible path from every node v_j to v_d. We denote this step as *MC_Search*. Then, we run the Dijkstra's algorithm from v_s to v_d by minimizing the cost. However, we include a node v_j on the shortest path if and only if the entire path from v_s to v_d through v_j is a feasible path. Such look ahead is possible as *MC_Search* provides this feasibility information from v_j to v_d. This prevents our algorithm from following a shortest path that would result in constraint violations at a later point along the path. We denote this cost minimization step as *MIN_Search*. *MIN_Search* also removes requirement b) from $H(D, C, A)$. For the remainder of this subsection, we first develop the mapping function $H(D, C, A)$, and then present the *MC_Search* and *MIN_Search* functions.

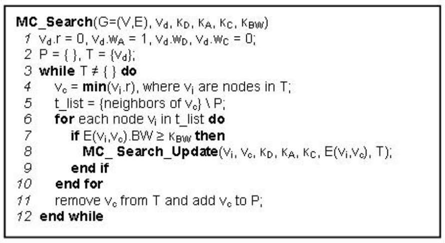
The uniform transformation of constraint conditions (as conducted in Equations 1 yields an interesting property when subject to the following non-linear function:

$$H^*(D, C, A) = \left(\frac{\sum_{i=1...k} D_i}{\kappa_D}\right)^\lambda + \left(\frac{\sum_{i=1...k} C_i}{\kappa_C}\right)^\lambda + \left(\frac{1 - \prod_{i=1...k} A_i}{1 - \kappa_A}\right)^\lambda \quad (2)$$

When λ is set to a large constant value, $H^*(D, C, A)$ will likely return a value no greater than 3 when each of the weights are below 1 (i.e. a feasible path). This property becomes more pronounced when λ is taken to ∞. $H^*(D, C, A)$ will return no greater than 3 when none of the weights violates constraint and return ∞ otherwise. Such non-linear function is the basis for a number of k-MCOP heuristics [13][15]. It is found that the maximization function exhibits similar characteristics [13] and we define our mapping function $H(D, C, A)$ accordingly:

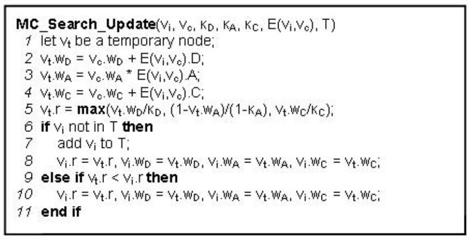
$$H(D, C, A) = \max\left(\frac{\sum_{i=1...k} D_i}{\kappa_D}, \frac{\sum_{i=1...k} C_i}{\kappa_C}, \frac{1 - \prod_{i=1...k} A_i}{1 - \kappa_A}\right) \quad (3)$$

$H(D, C, A)$ will be no greater than 1 when all of the weights are below constraints. Furthermore, the value will always reflect the largest weight value in the set (i.e. closest to the constraint). This additional property is very useful in a greedy search strategy as it



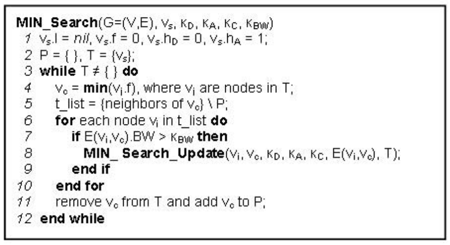
```
MC_Search(G=(V,E), $v_d$, $\kappa_D$, $\kappa_A$, $\kappa_C$, $\kappa_{BW}$)
$v_d.r = 0$, $v_d.w_A = 1$, $v_d.w_D$, $v_d.w_C = 0$;
P = { }, T = {$v_d$};
while T ≠ { } do
    $v_c$ = min($v_i.r$), where $v_i$ are nodes in T;
    t_list = {neighbors of $v_c$} \ P;
    for each node $v_i$ in t_list do
        if $E(v_i,v_c).BW \geq \kappa_{BW}$ then
            MC_ Search_Update($v_i$, $v_c$, $\kappa_D$, $\kappa_A$, $\kappa_C$, $E(v_i,v_c)$, T);
        end if
    end for
    remove $v_c$ from T and add $v_c$ to P;
end while
```

Figure 3: ***MC_Search*** **function**



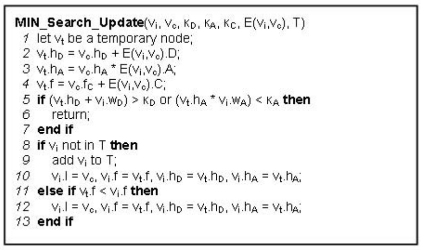
```
MC_Search_Update($v_i$, $v_c$, $\kappa_D$, $\kappa_A$, $\kappa_C$, $E(v_i,v_c)$, T)
let $v_t$ be a temporary node;
$v_t.w_D = v_c.w_D + E(v_i,v_c).D$;
$v_t.w_A = v_c.w_A * E(v_i,v_c).A$;
$v_t.w_C = v_c.w_C + E(v_i,v_c).C$;
$v_t.r$ = max($v_t.w_D/\kappa_D$, $(1-v_t.w_A)/(1-\kappa_A)$, $v_t.w_C/\kappa_C$);
if $v_i$ not in T then
    add $v_i$ to T;
    $v_i.r = v_t.r$, $v_i.w_D = v_t.w_D$, $v_i.w_A = v_t.w_A$, $v_i.w_C = v_t.w_C$;
else if $v_t.r < v_i.r$ then
    $v_i.r = v_t.r$, $v_i.w_D = v_t.w_D$, $v_i.w_A = v_t.w_A$, $v_i.w_C = v_t.w_C$;
end if
```

Figure 4: ***MC_Search_Update*** **function**



```
MIN_Search(G=(V,E), $v_s$, $\kappa_D$, $\kappa_A$, $\kappa_C$, $\kappa_{BW}$)
$v_s.l = nil$, $v_s.f = 0$, $v_s.h_D = 0$, $v_s.h_A = 1$;
P = { }, T = {$v_s$};
while T ≠ { } do
    $v_c$ = min($v_i.f$), where $v_i$ are nodes in T;
    t_list = {neighbors of $v_c$} \ P;
    for each node $v_i$ in t_list do
        if $E(v_i,v_c).BW > \kappa_{BW}$ then
            MIN_ Search_Update($v_i$, $v_c$, $\kappa_D$, $\kappa_A$, $\kappa_C$, $E(v_i,v_c)$, T);
        end if
    end for
    remove $v_c$ from T and add $v_c$ to P;
end while
```

Figure 5: ***MIN_Search*** **function**



```
MIN_Search_Update($v_i$, $v_c$, $\kappa_D$, $\kappa_A$, $\kappa_C$, $E(v_i,v_c)$, T)
let $v_t$ be a temporary node;
$v_t.h_D = v_c.h_D + E(v_i,v_c).D$;
$v_t.h_A = v_c.h_A * E(v_i,v_c).A$;
$v_t.f = v_c.f_C + E(v_i,v_c).C$;
if $(v_t.h_D + v_i.w_D) > \kappa_D$ or $(v_t.h_A * v_i.w_A) < \kappa_A$ then
    return;
end if
if $v_i$ not in T then
    add $v_i$ to T;
    $v_i.l = v_c$, $v_i.f = v_t.f$, $v_i.h_D = v_t.h_D$, $v_i.h_A = v_t.h_A$;
else if $v_t.f < v_i.f$ then
    $v_i.l = v_c$, $v_i.f = v_t.f$, $v_i.h_D = v_t.h_D$, $v_i.h_A = v_t.h_A$;
end if
```

Figure 6: ***MIN_Search_Update*** **function**

attempts to select paths with good overall QoS values.

The *MC_Search* function is presented in Figure 3 and 4. It tries to minimize the highest weight from each node v_j to v_d. Each node keeps track of the following information: the maximum weight r of the minimal path from v_j to v_d, the delay weight w_D, the availability weight w_A and the cost w_C of the path. The w weights are also used to compute complete path information in the *MIN_Search* function.

The *MIN_Search* function (Figure 5) is identical to the *MC_Search* function. Each node v_j keeps track of the cost $v_j.f$ of a minimal feasible path from v_s to v_j, the predecessor $v_j.l$ of v_j on the said path, the delay $v_j.h_D$ of the path, and the availability $v_j.h_A$ of the path. The function tries to find the minimal feasible path from v_s to v_d. The *MIN_Search_Update* function (Figure 6) ensures only foreseeable feasible paths are explored.

Now, we present the service composition algorithm (Figure 7) that utilizes the *MC_Search* and *MIN_Search* functions. The algorithm terminates early if *MC_Search* does not return a feasible path. Otherwise, the algorithm will optimize on such a feasible path p using *MIN_Search*, which yields a feasible path p^* with cost at least as low as p.

Our service composition algorithm has a runtime of $O(m)$, where m is the total number of edges on the expanded domain graph. This is proportional to the total number of service classes in the domains interconnecting v_s and v_d. In densely connected domain

```
Service_Composition(G=(V,E), $v_s$, $v_d$, $K_D$, $K_A$, $K_C$, $K_{BW}$)
  1 MC_Search(G=(V,E), $v_d$, $K_D$, $K_A$, $K_C$, $K_{BW}$);
  2 if $v_s$.r > 1 then
  3     return nil; //no feasible path
  4 end if
  5 MIN_Search(G=(V,E), $v_s$, $K_D$, $K_A$, $K_C$, $K_{BW}$);
  6 return the path by following $v_d$.l;
```

Figure 7: The Service Composition Algorithm

graphs, the runtime is upper-bounded by $O(n^2)$, where n is the number of nodes on the expanded domain graph. The effectiveness and efficiency of our service composition algorithm is demonstrated in Section 5.

3. Network Adaptation and Mobility Support

Network performance may vary significantly over time. Even with the best service assurance scheme, one or more domains carrying the service traffic may fail to deliver their promised QoS performance during the course of a service session. When such an event arises, the network should perform self-adaptation by seeking an alternative communication path that satisfies the original QoS requirements, while causing as little service disturbance as possible. Based on our service composition algorithm, we present a simple network adaptation algorithm. The objective is to find a minimal cost alternative path p_{new} that utilizes as much of the old path p_{old} as possible.

```
SC_Adaptation(G=(V,E), $p_{old}$, $v_s$, $v_d$, $K_D$, $K_A$, $K_C$, $K_{BW}$)
  1 set cost of each edge in $p_{old}$ to 0;
  2 for each domain travesed by $p_{old}$ do
  3     set cost of other edges in the domain to the switching cost;
  4 end for
  5 Service_Composition(G=(V,E), $v_s$, $v_d$, $K_D$, $K_A$, $K_C$, $K_{BW}$);
```

Figure 8: The Network Adaptation Algorithm

The algorithm first updates the edge weights on the expanded domain graph to reflect the new domain service conditions. Then, for each domain traversed by p_{old}, set the cost of the edge in p_{old} (i.e. the chosen service class in the domain) to 0, and set the cost of the other edges (i.e. the other service classes in the domain) to the cost of switching to that class. Run the service composition algorithm to obtain a new path.

Figure 8 details the algorithm *SC_Adaptation*. As the cost of the old paths are set to 0, the new alternative path favors path reuse when possible. The new path cost is the additional cost that must be absorbed by the violating domains in order to maintain the service. To cope with mobility, the new location of a service/user is updated on the graph and then *SC_Adaptation* is run.

4. A Framework for Multi-Domain Service Composition

In this section, we present a simple service composition framework that utilizes our algorithm (Figure 9). In our framework, each domain provides three basic services: neighborhood service, provisioning service, and monitor service. The framework assumes BGP is used for inter-domain routing. The neighborhood service periodically pulls the BGP speakers to extract two pieces of information: the neighboring domains it can send traffic to and the list of IP addresses that can be reached via each of these domains. The neighborhood service further augments this information with domain administrative policies and service class descriptions. The provisioning service provides the capability for intra-domain provisioning via the local provisioning mechanism and conduct inter-domain negotiation with the neighboring domains. With this construction, domains can have different local provisioning mechanisms. In practice, we expect most of these domains to employ DiffServ. Then the bandwidth broker based architecture can be used to facilitate both the intra-domain provisioning and cross-domain negotiation [19] with QoS demands given by the provisioning service. Such domain provisioning independence also grants each domain with autonomy in conducting its own resource management, QoS-based admission control, and pricing strategies. The monitoring service is tasked with the monitoring of QoS performance at the domain level. It achieves this goal via an array of QoS monitors installed near network border gateways and at the service components. We only expect the monitoring to be “coarsely grained”, in that only QoS performance across an entire domain (or from the service component to the edge of its home domain) is monitored and only for each service class (rather than per flow).

The service composition framework has three functions: domain discovery, service composition, and network adaptation. The domain discovery function provides information on domain connectivity and service class descriptions by contacting the neighborhood services on per domain basis. It realizes step 5 of the *MC_Search* and *MIN_Search* functions. The service composition function first runs our service composition algorithm to obtain a feasible path consists of the list of connected domains and their chosen service classes. Then it contacts the provisioning function of each chosen domain to perform the actual provisioning activities concurrently. Finally, it initiates an instance of the network adaptation function for the path. The network adaptation function monitors an end-to-end communication path for QoS violations. It fulfills this role by subscribing to a set of monitoring services along the path and supply them with the QoS thresholds for each domain. If the QoS performance of a particular domain is within ε percent of the domain thresholds, the network adaptation function is notified and the adaptation algorithm is run to obtain a new communication path. The provisioning activity is not carried out until the QoS constraint has been violated, at which point the violating domain is asked to accept the additional cost of the path switch. If permitted, the provisioning activities are performed by the service provisioning function. In case

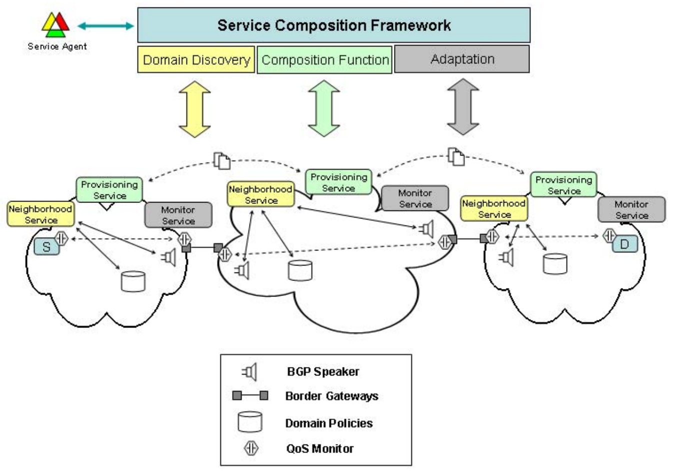


Figure 9: Framework for Multi-Domain Service Composition

of service mobility, the network adaptation function is activated when it is notified of a component's change of location.

When a request for a network communication path is made to the service composition framework, a service agent embedding the three functions is instantiated for the requestor. There is no central unit of processing or information storage in the framework, and no single entity holds global knowledge of the domain connectivity or network states. Furthermore, each domain retains its own administrative independence and is not required to share any sub-domain infrastructural or operational information at the domain level.

5. Simulation Studies

In this section, we compare the performance of our service composition algorithm (denoted as *SC*) with two other algorithms with similar runtime. The first algorithm (denoted as *Korkmaz*) is a well known k-MCOP heuristic developed by Korkmaz and Krunz [13]. The second algorithm (denoted as *SC-SON*) is used in the service composition scheme proposed by Gu et. al. [8]. Both algorithms have good runtime performance and high success rate of finding a feasible service composition. We compare these algorithms in terms of success rate and runtime speed, the path cost of the solutions, and the optimality of the solutions.

For simulation setup, we constructed a domain graph based roughly on the Cable&Wireless network topology in the US and UK (Figure 10). On this graph, the edges with boxes are domains offering three service classes, while the other edges are

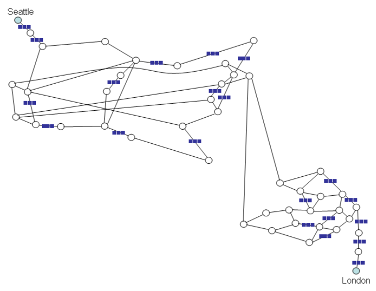


Figure 10: Domain Graph: Simulation Topology

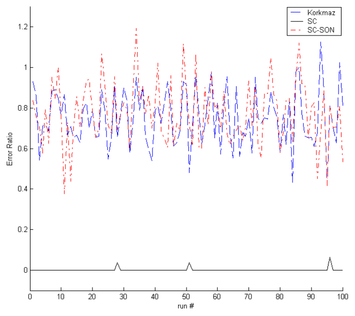


Figure 11: Optimality Comparison

domains with single service class. Furthermore, each edge is assigned a weight set: delay, cost, availability and available bandwidth. The value of available bandwidth is set to 2Mb/s and the availability is a random value chosen between 0.990 and 0.998. The delay (in ms) and cost values are also randomly determined from a range. The range is dependent on the edge length, where longer edges have higher delay and cost ranges. The three cross continental US links (in bold) are assigned relatively low delay ranges and high cost ranges. In addition, for edges with multiple service classes, each class offers progressively better QoS values at higher cost.

The simulations are performed with Redhat Linux 9 on a Pentium 4 1.4GHz PC with 512MB memory. Given the constraint set $\{\kappa_d = 110ms$, $\kappa_c = 150$, $\kappa_a = 0.9$, $\kappa_{BW} = 2Mb/s\}$ and a random weight initialization on the graph, each algorithm is required to find the minimal cost feasible path between the node in Seattle and the node in London. As base case, depth-first search is performed to find the minimal cost feasible path on the graph. Performance data are collected from 100 such runs. For all of the runs, all three algorithms are able to return a feasible solution. The runtime of the three algorithms are identically clocked at under 100ms, with the SC-SON algorithm performing slightly better due to simpler implementation. In contrast, the depth first search algorithm takes from 6 to 9 minutes to complete.

Figure 12 illustrates the path cost of the solutions under different delay constraints. We observe in 12a, the cost of our solution is much lower than the cost of the Korkmaz or SC-SON solutions. In fact, the optimal cost (solution of DFS) is not visible in 12a because our solution finds this optimal cost most of the time. In Figure 12b, the delay constraint is set to the delay of the optimal cost path. Our algorithm is able to return the same optimal solution as in 12a. Finally, in Figure 12c, the delay constraint is set close to the minimal path delay on the graph (from Seattle to London), our algorithm is still able to return path with lower cost than the other two algorithms. However, the gap is not as significant as in 12a and 12b, because with the reduction in number of feasible paths, the margin for improvement is much smaller. Figure 12d presents the path delay of the

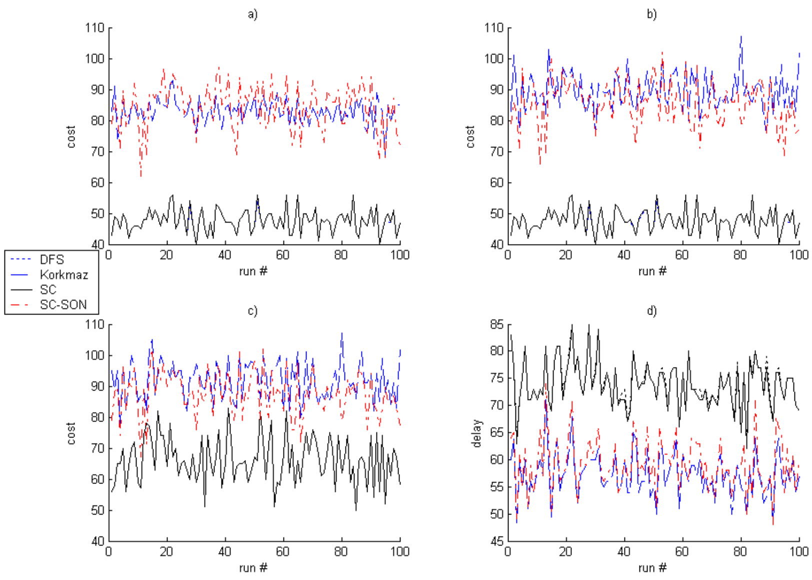


Figure 12: Performance Comparison

solutions from 12c. We observe that our algorithm is able to obtain near optimal cost by finding paths whose delays are close to the delay constraint. In fact, the delay constraint line (illustrated by dotted line) is hardly visible as our solution has path delay values equal or close to the delay constraint values.

The optimality of the solutions from Figure 12a is plotted in Figure 11. Over the 100 runs, our algorithm is able to find the minimal cost feasible path 97% of the time, and is always able to find a near optimal solution. In comparison, the Korkmaz and SC-SON algorithms can find solutions whose path cost is 80% (on average) off the optimal path cost. In the graph, the error percentage ϵ is computed based on the marginal cost difference between the optimal solution C_{opt} and the solution obtained from the algorithms C_{algo}:

$$\epsilon = (C_{algo} - C_{opt})/C_{opt} \tag{4}$$

Furthermore, for the Korkmaz and SC-SON algorithms, ϵ varies significantly between runs, ranging from 0.38 to 1.2.

To evaluate the performance of the network adaptation scheme, the path returned by our composition algorithm is subjected to domain defection. Each edge along the path has a 10% independent probability of defection. On average, a path consists of 15 edges and each defective edge is assigned available bandwidth of 0Mb/s. The network adaptation scheme is then performed in each run. The results show when the number of defective domains is reasonable (i.e. up to 4 domains), our adaptation scheme can reuse most of the old edges in the new path (69% or more). Even when subject to high defection rate (6 to 7 domains), the scheme can still reuse 60% of the original path.

With comparable runtime performance to the Korkmaz and SC-SON algorithms, our algorithm can find a service composition that satisfies the QoS constraints while achieving cost minimization most of the time. As our algorithm uses a greedy cost minimization strategy, the path cost generated by our solution is much lower than the path cost generated by the other two algorithms. In addition, the network adaptation scheme achieves relatively high path reuse rate, while still generating alternative feasible paths with low path cost.

6. Related Works

Raman et. al. [17] presented a general framework for service composition across multiple providers. They stress on the importance of inter-domain cooperations among service providers and the need for performance aware service composition. In their reference architecture, they envision the service composition to encompass both the application and the connectivity planes wherein "end-to-end network with desirable properties" should be constructed at the connectivity plane. The primary focus of our work is to enable the creation of such communication paths, that can also self-adapt to varying network conditions. Zeng et. al. [23] proposed a QoS-aware service composition scheme for web services. They are concerned with finding the optimal service execution plan among the set of candidate service components, while taking into account the QoS characteristics of these components. The QoS attributes investigated in this work are software quality oriented (e.g. execution duration, reputation, success rate, etc.). Similar in our work, they pose the composition problem as a graph search problem, except their graph represents the execution states of the service components. An integer programming technique is proposed to solve this problem. The scheme does not scale well beyond 60 states. Although titled "QoS-aware", no QoS adaptation scheme is proposed at the software level. Similarly, Gu et. al. [8] proposed a QoS-assured service composition mechanism for Service Overlay Networks (SON). They attempt to find a feasible service component flow via a linear multi-constraint mapping function. However, the search heuristic only attempts to find a feasible path, rather than an optimal one. We compare the performance of our search heuristic with theirs in Section 5. They also proposed a simple localized recovery scheme to cope with QoS violations.

The k-MCOP problem is well studied in the literature, particularly in the context of QoS routing. Chen and Nahrstedt [5] proposed an approximation algorithm for finding a feasible path. Their algorithm involves mapping $k-1$ real weights to $k-1$ integers in the range of 0 to x. A dynamic programming scheme is then used to obtain a feasible path. The runtime complexity of their algorithm is $O(x^2n^2)$, where in practice x must be set to a very large integer. The TAMCRA [15] algorithm attempts to find a feasible path without optimization. The work proposes to use non-linear k constraint mapping functions. The design of our mapping function follows a similar idea. Korkmaz and Krunz [13] first proposed a heuristic using two pass Dijkstra's algorithm with look ahead. However, their look ahead heuristic is overly simplistic and their minimization step does not yield near optimal solutions. The performance of their heuristic is compared with others in Section 5.

7. Conclusion

In this paper, we investigated the composition of QoS-aware network communication path across large scale multi-domain networks. In order to support dynamic service composition, it is imperative to establish end-to-end QoS assured communication paths between distributed service components, and to self-adapt to varying network conditions. Given that each domain has its own QoS provisioning mechanisms and offers a set of service classes, we formalize the problem as finding a selection of service classes in interconnected domains between two service components, such that the end-to-end QoS constraints are satisfied and the cost of the composition is minimal. Via domain graph expansion technique, we reduce the problem to k-MCOP problem and developed a fast search heuristic. Simulation studies show that our service composition algorithm has very high probability of finding the optimal solution and generates solutions with path cost much lower than those generated by the well known search heuristics today. Furthermore, our algorithm only relies on per hop domain information obtainable from BGP information. We further proposed a simple service composition framework that utilizes this algorithm.

Our work lends itself to extensions in both the theoretical and the application domains. On the theoretical side, further analysis and comparison should be conducted to formally prove the optimality and effectiveness of our service composition scheme. On the application side, a prototype implementation of the service composition framework should be undertaken to evaluate the runtime performance and network cost of our algorithm.

References

[1] Universal description, discovery and integration technical white paper. UDDI.org, September 2000.

[2] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web service v1.1. IBM Technical White Paper, May 2003. http://www.ibm.com/developerworks/library/ws-bpel/.

[3] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An architecture for differentiated services. RFC2475, December 1998.

[4] D. Booth, H. Hass, F. McCabe, E. Newcomer, M. Champion, C. Ferris, and D. Orchard. Web services architecture. W3C Working Group Note 11, February 2004. http://www.w3.org/TR/2004/NOTE-ws-arch-20040211/.

[5] S. Chen and K. Nahrstedt. On finding multi-constrained paths. In *Proceedings of IEEE International Conference on Communications (ICC98)*, volume 2, pages 874–879. IEEE, June 1998.

[6] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An architecture for a secure service discovery service. In *Proceedings of Fifth Annual International Conference on Mobile Computing and Networks (MobiCom '99)*. IEEE, August 1999.

[7] I. Foster (Editor). The open grid service architecture, v1.0. Grid Forum OGSA Working Group Draft, July 2004. http://forge.gridforum.org/projects/ogsa-wg.

[8] X. Gu, K. Nahrstedt, R. N. Chang, and C. Ward. QoS-assured service composition in managed service overlay networks. In *Proceedings of the 23rd International Conference on Distributed Computing Systems*. ACM, May 2003.

[9] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol v2. RFC2608, June 1999.

[10] M. Handley, H. Schulzrinne, E. Schooler, and J. Rosenberg. SIP: Session initiation protocol. RFC2543, March 1999.

[11] W. Hoschek. The web service discovery architecture. In *Proceedings of ACM/IEEE Conference on Supercomputing 2002*, 2002.

[12] A. Keller, G. Kar, H. Ludwig, A. Dan, and J. L. Hellerstein. Managing dynamic services: A contract based approach to a conceptual architecture. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2002 (NOMS2002)*. IEEE/IFIP, April 2002.

[13] T. Korkmaz and M. Krunz. Multi-constrained optimal path selection. In *Proceedings of IEEE INFOCOM 2001*, volume 2, pages 834–843. IEEE, April 2001.

[14] F. Leymann. Web service flow language (WSFL 1.0). IBM Software Group, May 2001. http://www-3.ibm.com/software/ solutions/webservices/pdf/WSFL.pdf.

[15] H. De Neve and P. Van Mieghem. A multiple quality of service routing algorithm for PNNI. In *Proceedings of the ATM Workshop*, pages 324–328. IEEE, May 1998.

[16] G. Piccinelli, C. Preist, and C. Bartolini. E-service composition: Supporting dynamic definition of process-oriented negotiation parameters. In *Proceedings of 12th International Workshop on Database and Expert Systems Applications*. IEEE, September 2001.

[17] B. Raman, S. Agarwal, Y. Chen, M. Caesar, W. Cui, P. Johansson, K. Lai, T. Lavian, S. Machiraju, Z. Morley-Mao, G. Porter, T. Roscoe, M. Seshadri, J. S. Shih, K. Sklower, L. Subramanian, T. Suzuki, S. Zhuang, A. D. Joseph, R. H. Katz, and I. Stoica. The SAHARA model for service composition across multiple providers. In *Proceedings of the First International Conference on Pervasive Computing*. ACM, August 2002.

[18] Y. Rekhter and P. Gross. Application of the border gateway protocol in the internet. RFC1772, March 1995.

[19] W. Rhee, J. Lee, M. Yang, I. Lee, J. Yu, and S. Kim. Dynamic provisioning mechanism for heterogeneous QoS guarantee in differentiated service networks. In *Proceedings of IEEE International Conference on Communications 2003 (ICC03)*, volume 3, pages 1912–1916. IEEE, May 2003.

[20] M. Salle and C. Bartolini. Management by contract. In *Proceedings of IEEE/IFIP Network Operations and Management Symposium 2004 (NOMS2004)*. IEEE/IFIP, April 2004.

[21] Z. Wang and J. Crowcroft. QoS routing for supporting resource reservation. *IEEE Journal on Selected Areas in Communication*, 1996.

[22] J. Wroclawski. The use of RSVP with IETF integratd services. RFC2210, September 1997.

[23] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-aware middleware for web service composition. *IEEE Transactions on Software Engineering*, 30(5):311–327, May 2004.