QoS-Aware service composition and adaptation in Autonomic Communication

Jin Xiao and Raouf Boutaba
School of Computer Science
University of Waterloo
200 University Ave. W., Waterloo, ON, Canada
Email:{j2xiao,rboutaba}@bbcr.uwaterloo.ca

Abstract

Advents in network technology and distributed system design have propelled network communication service beyond best effort data delivery. With the rising complexity of network infrastructures and the need for on-demand provisioning operations, a high degree of self-sufficiency and automation is required in the network service infrastructure. Guided by the autonomic communication principle, this paper first presents an autonomic service provisioning framework for establishing QoS-assured end-to-end communication paths across administratively independent domains. Through graph abstraction, we show that the domain composition and adaptation problem could be reduced to the classic k-MCOP problem. In analyzing existing k-MCOP solutions, we show their inefficiencies when applied to the service provisioning context and establish a number of new domain composition and adaptation algorithms. These new algorithms are designed for the self-configuration, self-optimization and self-adaptation of end-to-end network communications and can provide hard QoS guarantees over domains with relative QoS differentiations. Through in-depth experimentations, we compare the performance of our algorithms with classic k-MCOP solutions and demonstrate the effectiveness of our approach.

Index Terms

Autonomic Communication, Service Provisioning, Service Composition and Adaptation, Quality of Service

I. Introduction

Over the past years, advents in network technology and distributed system design have propelled network communication service beyond best effort data delivery. With the increased dependency on reliable and QoS assured network operations, next generation networks (NGN) are envisioned to support the dynamic

provisioning of network services, catered to the specific QoS requirements of the users/applications. The emergence of new computing paradigms (e.g. peer-to-peer networks, Web services, Grid services, overlay networks, etc.) further accentuates the need for on-demand network service provisioning. It remains that the objective of provisioning is to establish an end-to-end data communication pathway from source to destination with QoS assurance. Most of the existing works focus on provisioning within a single administrative domain. They can be grouped into two general approaches: hard resource allocations (e.g. IETF IntServ approach [1]) or relative traffic differentiation (e.g. IETF DiffServ approach [2]). The latter method is favored in large scale networks due to its scalability and better resource utilization. With the increased complexity of network infrastructures and the rising need for on-demand service provisioning operations, a high degree of self-sufficiency and automation is required in network service infrastructure. This is in part the aim of autonomic communications [3]: to engender self-managed intelligent networks that are capable of self-configuration, self-protection, self-optimization and self-healing. Such infrastructure could significantly reduce the complexity of network management and lessen the degree of human participation in network operations.

To realize automated and dynamic provisioning of network services, it is important to understand the requirements of the upper layer applications, who are the "users" of service, and the characteristics of the underlying network infrastructure, who are the "providers" of service. On the user side, the increasing distributedness of software and systems results in end-to-end communication paths across multiple *domains*. In the context of this paper, we use the term *domain* to mean administratively independent network domains or independent service providers. To date, the issue of inter-domain service provisioning is seldom addressed in literature. In our view, an inter-domain service provisioning mechanism must respect the following two conditions: domains share few information with each other and generally no information about their own networks; each domain independently conducts its own intra-domain provisioning. An end-to-end communication path is thus composed of a set of such consecutive domains selected among a much

larger set of interconnected domains and each of which offers different sets of QoS guarantees at varied cost. For example, the IETF DiffServ approach [2] uses class markings to facilitate differentiated services in a domain, and each domain is entitled to its own class specifications and associated QoS levels. An important question to answer when provisioning an end-to-end network service is: what domains to involve and what service class to choose in each involved domain? We term this the *service composition* problem. On the provider side, the dynamicity in network weather induces rapid changes in QoS conditions of domains. This issue is particularly pronounced for wireless networks, an important part of next generation network infrastructure. As a result, domains that promise specific QoS levels at configuration time may fail to honor these requirements. Therefore, a communication path must monitor, re-evaluate and adjust its domain composition dynamically in order to ensure the required end-to-end QoS level is maintained. In the presence of mobility, as communicating applications/users roam across domains, path reconstruction may also become necessary. We term this the *service adaptation* problem. In addition, solutions to the composition and adaptation problems should be QoS-aware, in that the composition should result in a satisfying end-to-end QoS-assured path based on QoS information of domains, and the adaptation must be performed as to ensure the QoS requirements are upheld whenever network weather fluctuates. It is essential for an autonomic service provisioning infrastructure to effectively and efficiently address these problems.

In this paper, we present a mechanism for QoS-aware service composition and adaptation of end-to-end network service for autonomic communication. First, we introduce a service provisioning framework based on the autonomic communication principle, covering a number of essential functions: domain discovery, domain reacheability, composition, cross-domain contracting, intra-domain provisioning, domain-wide monitoring and adaptation. To achieve self-management intelligence, an efficient method for service composition and adaptation is required at the inter-domain level. Through domain graph abstraction, we reduce the domain composition and adaptation problem to the classic k Multi-Constrained Optimal Path

problem (k-MCOP). However, in the context of service provisioning, we find existing k-MCOP solutions inadequate and inefficient. Following this analysis, we develop a set of new algorithms for QoS-aware service composition and adaptation. With high probability, our composition algorithm finds a series of consecutive domains spanning end-to-end and select appropriate service class in each domain such that the overall QoS requirements are satisfied. The algorithm also minimizes the overall cost of the path. As the network condition changes over time or as the user roams across domains, our adaptation algorithm ensures the QoS requirements of the communication path is respected as long as it is feasible to do so, while minimizing the cost of such adjustments. Together, these algorithms are designed to support self-configuration, self-optimization and self-adaptation of network communication services. The domain abstraction and a simpler version of the algorithms were first introduced in our work on end-to-end service provisioning [4]. As we address the service provisioning problem at the domain level, our algorithms can function over heterogeneous intra-domain provisioning mechanisms, and more importantly, provide hard end-to-end QoS guarantees over "soft" intra-domain QoS schemes (i.e. offered by service differentiation approaches). Through in-depth experimentation with real network topologies, we compare the performance of our algorithms with classic k-MCOP solutions and demonstrate the effectiveness of our approach.

The remainder of the paper is organized as follows. Section II presents our service provisioning framework and related works. Section III reduces the composition and adaptation problems to k-MCOP through domain graph abstraction, and analyzes existing k-MCOP solutions. Section IV presents our algorithms and their application to composition and adaptation. Through detailed experimentations in Section V, we show the performance and effectiveness of our solution. Section VI summarizes our work and contributions.

II. Autonomic Provisioning Framework and Related Works

First, we briefly discuss existing works in inter-domain service composition and provisioning. Raman et. al. [5] 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. Zeng et. al. [6] 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.). They pose the composition problem as a graph search problem, where the graph represents the execution states of the service components. An integer programming technique is proposed to solve this problem. Their experimentation show that the scheme does not scale well beyond 60 states. Furthermore, no QoS adaptation scheme is proposed at the software level. Similarly, Gu et. al. [7] 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 does not perform well (Section V) and only attempts to find a feasible path. They also proposed a simple localized recovery scheme to cope with QoS violations.

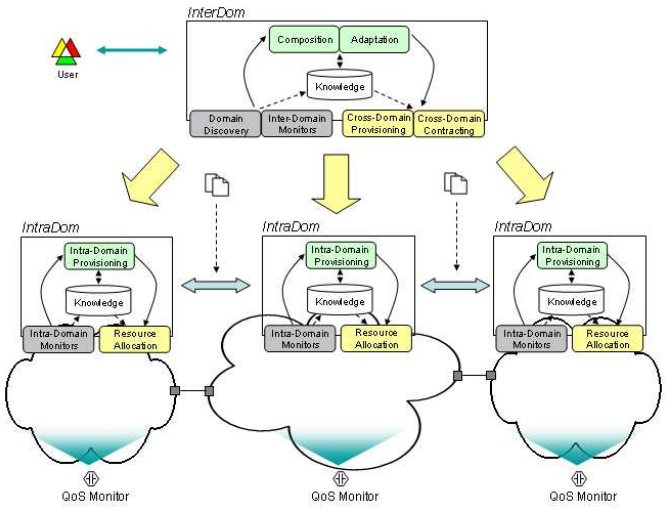


Fig. 1. Autonomic Provisioning Framework

In the network service context, there lacks a guiding framework for autonomic service provisioning of

end-to-end communication paths. For the remainder of this section, we present our view of an autonomic service provisioning framework and discuss its key functions. Figure 1 illustrates our framework. At the intra-domain level, we assume the existence of autonomic component (*IntraDom*) capable of conducting QoS provisioning, performing resource allocations, and monitor QoS conditions within the domain. We focus on the functions of inter-domain autonomic provisioning component (*InterDom*). Three general functions are prescribed for an autonomic component [3]: sensor, analyzer/planner, and actuator. The sensor function of the InterDom consists of two part: domain discovery and inter-domain monitoring. The domain discovery function is able to discover domain connectivities and QoS service class descriptions for domains. Such discovery could be facilitated either through large-scale discovery systems, such as Secure Service Discovery Service [8] and works on Web Service discovery [9][10], or through facilities in the existing network infrastructure, such as BGP [11]. In the case of BGP, the BGP speakers can be pulled periodically by the domain discovery function 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. Information on domain administrative policies and service class descriptions from domain administration could be gathered to provide the needed QoS class descriptions. The inter-domain monitor function is tasked with monitoring the domain-level QoS condition for each service class a domain provides. This function can be facilitated via a set of distributed QoS monitors installed at domain borders, measuring the aggregate QoS condition of each service class in the domain. As InterDom works at the domain-level, for better scalability, only domain-wide QoS measurements are necessary. We note that the monitors of InterDom should not make use of monitoring functions offered by IntraDom for the following reason: monitoring is an essential part of QoS enforcement, a domain could misrepresent its own QoS conditions for obvious economical reasons. Hence domain-level QoS monitoring should be conducted independent of the domain administrations.

The actuator function of InterDom is composed of the cross-domain contracting function and the cross-

domain provisioning function. The cross-domain contracting function is responsible for establishing the required contracts with each domain for specific QoS classes and setup cross-domain traffic exchange with neighboring domains. Recent research works on service composition [5][6][7] and work flow languages, such as Web Service Flow Language (WSFL) [12] and Business Process Execution Language for Web Service (BPEL4WS) [13] could be leveraged to accomplish this task, especially when combined with works done on contract-based cross-domain management [14][15][16]. The inter-domain provisioning function interfaces with IntraDom to obtain a QoS-assured path segment across the domain at specific border points (as specified by the cross-domain contracting function), or from source/destination to the border point. Such domain provisioning independence grants each domain the autonomy in conducting its own resource management, QoS-based admission control, and pricing strategies. In practice, we expect most of these domains to employ DiffServ. Then a bandwidth broker like architecture can be applied to realize both the intra-domain provisioning and cross-domain negotiation [17].

The core self-management "intelligence" of InterDom lies in its analyzer/planner functions: composition and adaptation. The composition function must utilize the domain connectivity and service class information to determine the best suited domain path from source to destination and their respective service classes. And the adaptation function must react to changing QoS conditions along the path by modifying the existing composition to ensure end-to-end QoS requirements are satisfied. We envision an InterDom is dynamically created specifically for each communication path, utilizing the same underlying functions (e.g. IntraDoms, monitors, etc.). The management composition as determined by InterDom is dynamically adjusted whenever path adjustment takes place. More precisely, when a domain is selected/deselected from the communication path, the associated autonomic components of the domain are connected/disconnected from the InterDom. Since the composition and adaptation functions drive the autonomic behavior of the communication paths, we focus on finding suitable solutions to the *service composition* and *adaptation* problems. In this scope, we deal with domain level information: domain connectivity, service classes and

domain-wide QoS conditions.

III. Domain Graphs and Problem Analysis

In this section, we detail the creation of a 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 assurance and an associated cost. We focus on three common QoS factors in this paper: delay, availability, and bandwidth. In doing so, we reduce our problems to k-MCOP (k Multi-Constraint Optimal Path problem), which is known to be NP-Complete [18]. We then analyze the prominent solutions to the k-MCOP problem, and show their inadequacies in our context. As a result of this analysis, we are motivated to develop new QoS-aware composition and adaptation algorithms for autonomic service provisioning.

A. Domain Graph Creation

A domain exchanges traffic flows with its neighboring domains via border gateways. Each domain has a number of service classes, each with a set of QoS assurances and a price. For domains without QoS differentiations, they are assumed to provide a single set of QoS assurances and price. Figure 2 depicts a typical example of domain connectivity between two service components: S and D.

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 3 illustrates this transformation process. Inter-domain routing policies can be incorporated during this process. For example, domain A routes all traffics transiting from its south neighbor to its east neighbor through border gateway β. This policy is reflected in the graph (Figure 3). With such an 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

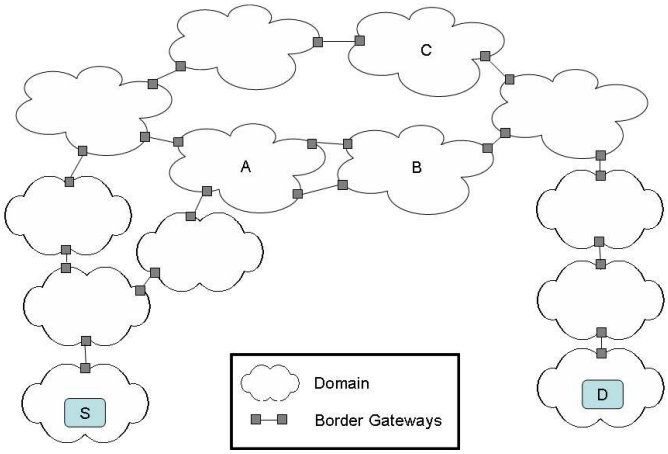


Fig. 2. An Example of Domain Connectivity

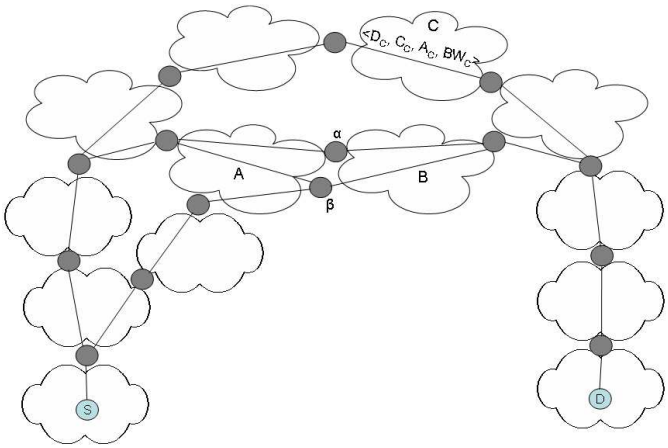


Fig. 3. Graph Representation of Domain Connectivity

C_C, then the edge connecting its border gateways is assigned the weight set $\{D_C, C_C, A_C, BW_C\}$.

As mentioned before, a domain can have a set of service classes (e.g. domain C offers three service classes and domain B offers two). A domain can also have complex QoS class offerings based on its policies. For example, domain A offers two service classes ($\{D_{A1}, C_{A1}, A_{A1}, BW_{A1}\}$ and $\{D_{A2}, C_{A2}, A_{A2}, BW_{A2}\}$) to traffics coming from the domain to its west, and only offers one service class ($\{D_{A3}, C_{A3}, A_{A3}, BW_{A3}\}$) to traffics coming from the domain to its south. To incorporate these service classes into the domain graph, we first associate each edge of the graph with the set of possible service classes. Then, the edge is expanded by introducing a number of "service nodes", where 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 a nil weight set $\{0, 0, 1, \infty\}$. Then it is apparent that the number of service nodes introduced on such an edge is equal to the number of service classes associated with that edge. Figure 4 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 points, but also depicts a selection of

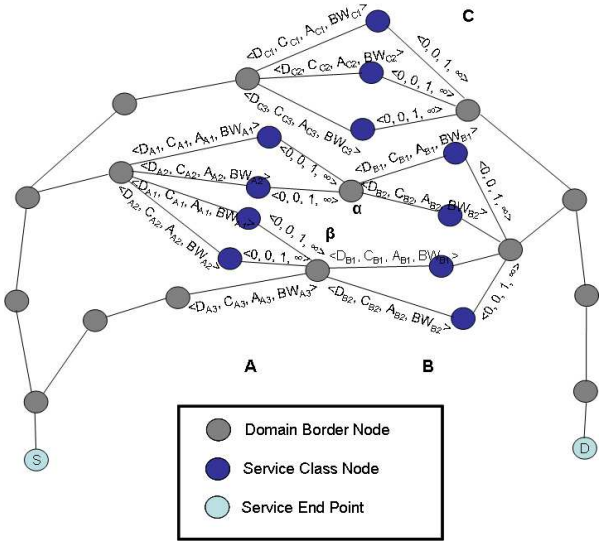


Fig. 4. Domain Graph with Service Class Expansion

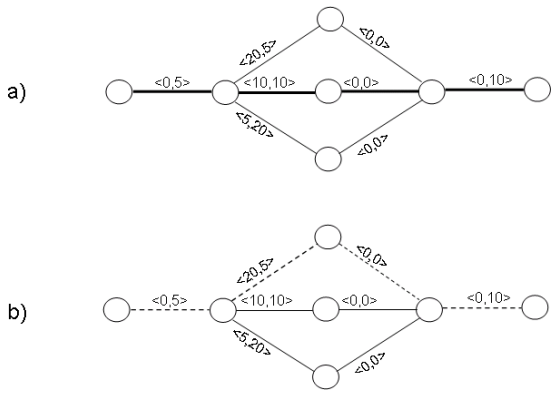


Fig. 5. Example of the Two-hop Problem on Domain Graph

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 composition and adaptation as a graph search problem. Therefore, our composition and adaptation problems 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.

Rather than dealing with heterogenous constraint conditions, we can rewrite the above constraints as:

$$\tau_1 = \frac{\sum_{i=1...k} D_i}{\kappa_D}, \tau_2 = \frac{\sum_{i=1...k} C_i}{\kappa_C} \quad (1)$$

$$\tau_3 = \frac{1 - \prod_{i=1...k} A_i}{1 - \kappa_A}, \tau_{4i} = \frac{\kappa_{BW}}{BW_i}$$

With respect to equations 1, 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 $u \in E$ has weights

$\{D, C, A, BW\}$, find a path P=(ω_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, and $\sum_{i=1...k} C_i$ is minimal.

This is equivalent to the *k Multi-Constraint Optimal Path problem* (k-MCOP), which is known to be NP-Complete [18]. In the following subsection, we present existing solutions to the k-MCOP problem and discuss their inadequacies.

B. Solutions to k-MCOP

The k-MCOP problem is well studied in the literature, particularly in the context of QoS routing. For purpose of analysis, we denote the number of nodes in the graph (including all node types) by N and the number of edges in the graph by E. Chen and Nahrstedt [19] proposed an approximation algorithm (*Chen*) 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 $O(x^2|N|^2)$. The probability of finding a solution with Chen's algorithm is directly related to the value space size of the weights. When a weight can take on a large set of real values, x must also increase proportionally such that there is an integer value in x very close to the weight values of a feasible path. Thus, the value of x must be very large in practice, especially when dealing with delay and availability metrics. Furthermore, Chen's algorithm does not seek optimal solution.

The limited path heuristic (*LPH*) algorithm [20] is constructed based on an extended Bellman-Ford algorithm. As the extended Bellman-Ford algorithm expands each node to keep track of all possible paths from source to the node, the overall runtime is exponential. The LPH algorithm attempts to obtain an approximation by limiting the number of paths stored at any node to X. They further prove that when $X = O(|N|^2 lg|N|)$, LPH can obtain near optimal solutions. However, to obtain this, the runtime of LPH ($O(X^2|N||E|)$) becomes very large for the purpose of adaptation. In addition, Bellman-Ford requires complete domain connectivity information at its initialization stage.

The TAMCRA [21] algorithm (*TAMCRA*) attempts to find a feasible path without optimization. The work proposes to use non-linear constraint mapping functions that maps k constraints to a single value. When designed carefully, such mapping function can produce the desirable effect of amplifying the resulting value when one of the weights is above constraint. The algorithm keeps track of K non-dominating paths at each node. A path is said to be non-dominating over another path if one of its weight values is better than the corresponding weight value of the other path. The runtime of TAMCRA can be quite large when K is high ($O(K|N|log(K|N|)+K^3k|E|)$). As TAMCRA is not optimized, a high value of K is necessary to obtain near optimal solution. An extension to the TAMCRA algorithm is later proposed [22], but the increased runtime complexity makes it impractical for dynamic service provisioning.

Korkmaz and Krunz [23] proposed a heuristic (*H_MCOP*) using a 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. Nevertheless, H_MCOP is fast and does find feasible solutions when the constraint bound is loose.

Some investigations are conducted on the performance of these algorithms [24]. TAMCRA and H_MCOP algorithms are illustrated to provide the best performance. However, we find both of these algorithms are inadequate for service provisioning due to the existence of many "two-hop loop" on a domain graph. Figure 5 depicts such a loop. It is an abstraction of a domain with three service classes. Suppose the constraint set is $\{20,20\}$, both the TAMCRA and H_MCOP algorithms will find the path highlighted in 5a), which is not a feasible path. 5b) highlights a feasible path. This problem arises because the non-linear mapping function only retains information on the highest aggregate weight at any node, and both algorithms greedily explore the minimum of such aggregates. TAMCRA's limited backtracking capability allows it to sometimes recover from this condition, but when there are many two-hop loops on a graph, the algorithm could not perform well.

An in-depth performance evaluation of these algorithms is presented in Section V. We conclude that an effective service composition and adaptation scheme must consider a number of factors. First, the scheme must perform within reasonable time bound for adaptation to be effective (within hundreds of milliseconds). Although the actual efficiency of adaptation is also contingent on the efficiency of the information gathering facilities and the provisioning mechanisms, we believe the composition and adaptation algorithm should not become another significant factor to the overall runtime. Second, the cost and degree of disruption during adaptation should be minimized. Both of these factors are important for self-adaptation. Third, the scheme should have a high probability of finding a feasible path when one exists and provide a near optimal solution with regard to cost. This factor directly impacts the effectiveness of self-configuration and self-optimization. Fourth, the scheme should not introduce excessive communication overhead. Based on these guidelines, we develop new algorithms for the service composition and adaptation problem. In particular, we desire to find a solution that can effectively address the "two-hop loop" problem and provide a much better near optimal solution compared with existing works. Although in this paper we deal with undirected graphs, in practice the QoS conditions across two gateways of a domain could be different depending on the direction of the flow. Using directed domain graph could account for this aspect and since Dijkstra's algorithm could be applied to both directed and undirected graphs, our service composition algorithm exhibits similar characteristic.

IV. Composition and Adaptation for Autonomic Service Provisioning

In this section, we detail the development of new composition (*SComp*) and adaptation algorithms for autonomic service provisioning. We first introduce the construction of our non-linear mapping function and the service composition algorithm. Then we enhance the SComp algorithm to address the two-hop loop problem. Finally, we present the adaptation algorithm and show mobility support as a special case of adaptation.

A. The Composition algorithm

Most of the existing heuristics for k-MCOP utilize the Bellman-Ford or Dijkstra's algorithm, since both of them are simple and fast, while still yield provable shortest path solutions for 1-MCOP. Our heuristic favors 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 a 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 iff. 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:

```
MC_Search(G=(V,E), v_d, κ_D, κ_A, κ_C, κ_BW)
 1  v_d.r = 0, v_d.w_A = 1, v_d.w_D, v_d.w_C = 0;
 2  P = { }, T = {v_d};
 3  while T ≠ { } do
 4     v_c = min(v_i.r), where v_i are nodes in T;
 5     t_list = {neighbors of v_c} \ P;
 6     for each node v_i in t_list do
 7        if E(v_i,v_c).BW ≥ κ_BW then
 8           MC_ Search_Update(v_i, v_c, κ_D, κ_A, κ_C, E(v_i,v_c), T);
 9        end if
10     end for
11     remove v_c from T and add v_c to P;
12  end while
```

Fig. 6. *MC_Search* function

```
MC_Search_Update(v_i, v_c, κ_D, κ_A, κ_C, E(v_i,v_c), T)
 1  let v_t be a temporary node;
 2  v_t.w_D = v_c.w_D + E(v_i,v_c).D;
 3  v_t.w_A = v_c.w_A * E(v_i,v_c).A;
 4  v_t.w_C = v_c.w_C + E(v_i,v_c).C;
 5  v_t.r = max(v_t.w_D/κ_D, (1-v_t.w_A)/(1-κ_A), v_t.w_C/κ_C);
 6  if v_i not in T then
 7     add v_i to T;
 8     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;
 9  else if v_t.r < v_i.r then
10     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;
11  end if
```

Fig. 7. *MC_Search_Update* function

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

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 salient 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 a non-linear function is the basis for a number of k-MCOP heuristics [21][23]. It is found that the maximization function exhibits similar characteristics [23] and we define our mapping function $H(D,C,A)$ accordingly:

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

$H(D,C,A)$ 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 attempts to select paths with good overall QoS values.

The *MC_Search* function is presented in Figure 6. It tries to minimize the maximum 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, and 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. Steps 1 and

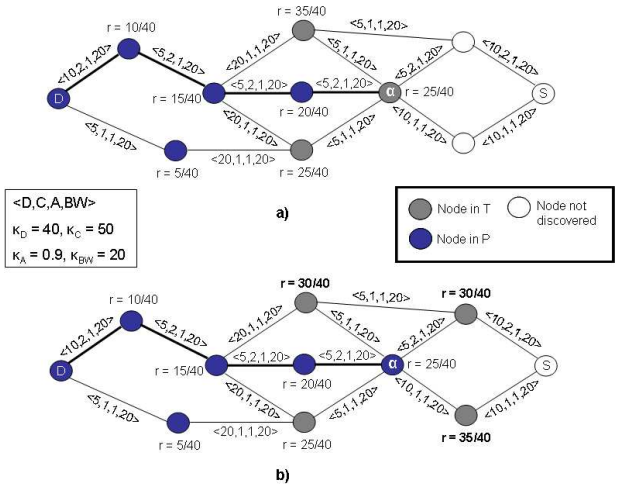


Fig. 8. An Example of *MC_Search*

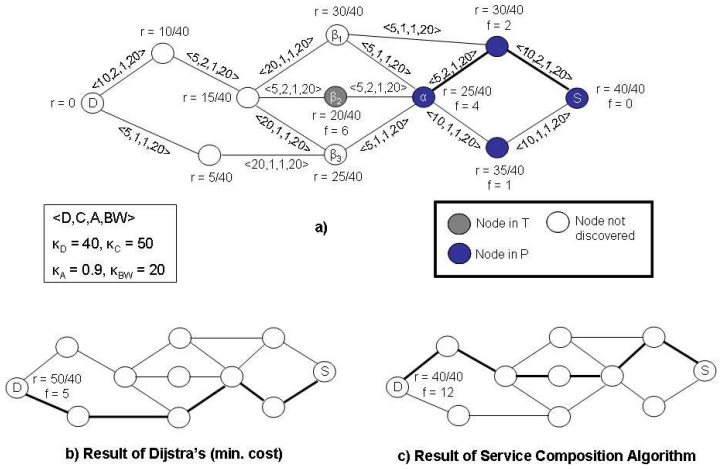


Fig. 9. Example of the Service Composition Algorithm

2 are initialization steps. The list P contains nodes whose r values cannot be further improved. Steps 4 and 5 greedily select a node with the smallest r and obtain its improvable neighbors (i.e. either in T or newly discovered). t_list is a list of such neighbors. Steps 6 to 10 iterate through each member of t_list and update its r value if permissible. Figure 7 details *MC_Search_Update*. Steps 1 to 5 compute the new weights and r of a path from v_d to v_i via v_c, where v_i is the set of neighboring nodes of v_c that has not been added to P yet. If v_i is a newly discovered node, it is added to T (steps 6 to 8), otherwise v_i is updated iff. the new path has a smaller r value than v_i's old path (steps 9 to 11).

Figure 8 illustrates the operation of *MC_Search*. In 8a, node α is the current best node. The value r of node α is obtained by taking the highlighted path. Three neighbors of node α are updated (in 8b) and node α is then added to P as it could not be further improved by the algorithm.

The *MIN_Search* function (Figure 10) 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.

```
MIN_Search(G=(V,E), v_s, κ_D, κ_A, κ_C, κ_BW)
 1  v_s.l = nil, v_s.f = 0, v_s.h_D = 0, v_s.h_A = 1;
 2  P = { }, T = {v_s};
 3  while T ≠ { } do
 4     v_c = min(v_i.f), where v_i are nodes in T;
 5     t_list = {neighbors of v_c} \ P;
 6     for each node v_i in t_list do
 7        if E(v_i,v_c).BW > κ_BW then
 8           MIN_ Search_Update(v_i, v_c, κ_D, κ_A, κ_C, E(v_i,v_c), T);
 9        end if
10     end for
11     remove v_c from T and add v_c to P;
12  end while
```

Fig. 10. *MIN_Search* function

```
MIN_Search_Update(v_i, v_c, κ_D, κ_A, κ_C, E(v_i,v_c), T)
 1  let v_t be a temporary node;
 2  v_t.h_D = v_c.h_D + E(v_i,v_c).D;
 3  v_t.h_A = v_c.h_A * E(v_i,v_c).A;
 4  v_t.f = v_c.f_C + E(v_i,v_c).C;
 5  if (v_t.h_D + v_i.w_D) > κ_D or (v_t.h_A * v_i.w_A) < κ_A then
 6     return;
 7  end if
 8  if v_i not in T then
 9     add v_i to T;
10     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;
11  else if v_t.f < v_i.f then
12     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;
13  end if
```

Fig. 11. *MIN_Search_Update* function

The *MIN_Search_Update* function (Figure 11) first computes the path cost and weights from v_s to v_i via v_c (steps 1 to 4). If no foreseeable feasible path exists from v_s to v_d via v_c and v_i, the function returns (steps 5 to 7). Otherwise, v_i is added to T if it is a newly discovered node (steps 8 to 10), or v_i is updated if the new path has lower cost than the old path (steps 11 to 13).

The operation of *MIN_Search* is illustrated in Figure 9a. Node α has three improvable neighbors β_1, β_2 and β_3. However, only β_2 is added to T, as following β_1 or β_3 does not lead to feasible paths. This look ahead property prevents the algorithm from examining lower cost paths (via β_1 or β_3 in this case) that may not be feasible.

Now, we present the service composition algorithm (Figure 12) 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. Figure 9c illustrates the result of the service composition algorithm, which is a minimal feasible path on the graph. We observe that this path is not the minimal cost path (as generated in Figure 9b) which violates the delay constraint.

Our service composition algorithm has twice the runtime of Dijkstra's algorithm. As Dijkstra's aglorithm

```
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;
```

Fig. 12. The Service Composition Algorithm

```
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 Dijkstra_Delay(G=(V,E), v_d);
3 Dijkstra_Avail(G=(V,E), v_d);
4 If v_s.r <= 1 then
5    MIN_Search(G=(V,E), v_s, v_d, K_D, K_A, K_C, K_BW);
6 end if
7 MIN_Search_Full(G=(V,E), v_s, v_d, K_D, K_A, K_C, K_BW);
8 return the lowest cost path between MIN_Search
  and MIN_Search_Full if any
```

Fig. 13. The Hybrid Service Composition Algorithm

has runtime of O($|E| + |N|log|N|$), our algorithm has a runtime of O(2($|E| + |N|log|N|$)), where $|E|$ is the total number of edges on the expanded domain graph and $|N|$ is the total number of nodes (including service nodes) on the expanded domain graph. This is proportional to the total number of service classes in the domains interconnecting v_s and v_d. The effectiveness and efficiency of our service composition algorithm is demonstrated in Section V.

B. The Hybrid Composition Algorithm

Our service composition algorithm suffers from the two-hop loop problem similar to the TAMCRA and H_MCOP algorithms. To address this issue, we enhance the service composition algorithm with an additional full path heuristic. During the backward feasibility search, in addition to performing *MC_Search*, we also perform Dijkstra's algorithm twice, once to find the minimal delay from any node to the source, and once to find the maximum availability from any node to the source. After this triple backward search step, each node will now hold not only the minimum non-linear mapping values to the source (and its path), but also the minimum delay and maximum availability paths. In general, these three paths are not the same. During the forward minimization step, in addition to performing *MIN_Search*, we also perform *MIN_Search_Full*, which instead of minimizing cost, attempts to minimize the full path cost projection based on the three paths recorded at each node. The hybrid algorithm returns the better result between *MIN_Search* and *MIN_Search_Full*. Figure 13 presents the hybrid service composition algorithm.

MIN_Search_Full can recover some feasible paths that was deemed infeasible by the *MC_Search*. When the

two-hop loop problem is encountered, although the non-linear mapping aggregate will lead to constraint violation, one of the single attribute minimizing path may help to keep such violating weight in check. Figure 14 and Figure 15 present *MIN_Search_Full* and *MIN_Search_Full_Update*. In *MIN_Search_Full_Update*, all three paths recorded by a node are evaluated (steps 4 to 25), and the node is evaluated based on the lowest full path cost projection instead of current path cost. The lowest full path cost is computed based on the current path cost to the node, plus the projected backward costs from destination to the node. In considering full path projections, the problem of greedily exploring lowest cost path that may lead to irrecoverable constraint violations (as in the case of two-hop loop problem) may be avoided. Furthermore, during the forward search, when a particular attribute grows near constraint, taking its minimal path may lead to feasible paths which SComp cannot find. Because *MIN_Search_Full* does not attempt to minimize cost, a path resulting from *MIN_Search_Full* could have higher cost than a path resulting from *MIN_Search*. The converse is also true, the full path cost projection can provide minimal solutions via lookahead, which a simple greedy search may not.

Similar to *MIN_Search*, if *MC_Search* is able to find a feasible path, then *MIN_Search_Full* is guaranteed to return this path if it cannot find a lower cost path. This property holds because the full path projection of the path found through *MC_Search* will always be within constraints and *MIN_Search_Full* will follow another path of a lower cost if and only if that path is also a feasible solution (guaranteed by the full path projection). Figure 16 shows an example in which the hybrid service composition algorithm is able to recover a feasible path where the simple service composition algorithm cannot. After the three backward searches, each node in the graph records the cumulative values of three paths: MC_Search, minimum delay, and maximum availability. For simplicity, only the results of MC_search and minimum delay are presented in Figure 16a. Since MC_Search does not return a feasible path (Figure 16b), the simple service composition algorithm fails. This is caused by the presence of a two-hop loop in the graph. The hybrid algorithm is able to recover a feasible path by using the path projected by the minimal delay search

```
MIN_Search_Full(G=(V,E), v_s, v_d, κ_D, κ_A, κ_C, κ_BW)
 1  v_s.l = nil, v_s.f = 0, v_s.h_D = 0, v_s.h_A = 1; v_s.h_C = 0;
 2  P = { }, T = {v_s};
 3  while T ≠ { } do
 4     v_c = min(v_j.f), where v_j are nodes in T;
 5     If v_c == v_d then
 6        return path;
 7     end if
 8     t_list = {neighbors of v_c} \ P;
 9     for each node v_i in t_list do
10        if E(v_i,v_c).BW > κ_BW then
11           MIN_ Search_Full_Update(v_i, v_c, κ_D, κ_A, κ_C, E(v_i,v_c), T);
12        end if
13     end for
14     remove v_c from T and add v_c to P;
15  end while
16  return nil;
```

Fig. 14. *MIN_Search_Full* function

```
MIN_Search_Full_Update(v_i, v_c, κ_D, κ_A, κ_C, E(v_i,v_c), T)
 1  let v_t be a temporary node;
 2  let v^kd, v^ka be the node v containing information from
    Dijkstra_Delay and Dijkstra_Avail respectively
 3  v_t.f = ∞;
 4  if (v_c.h_D + E(v_i,v_c).D + v_i.w_D) > κ_D
      or (v_c.h_A * E(v_i,v_c).A * v_i.w_A) < κ_A then
 5     v_t.h_D = v_c.h_D + E(v_i,v_c).D;
 6     v_t.h_A = v_c.h_A * E(v_i,v_c).A;
 7     v_t.h_C = v_c.h_C + E(v_i,v_c).C;
 8     v_t.f = v_c.h_C + v_i.w_C;
 9  end if
10  if (v_c.h_D + E(v_i,v_c).D + v^kd_i.w_D) > κ_D
      or (v_c.h_A * E(v_i,v_c).A * v^kd_i.w_A) < κ_A then
11     if (v_c.h_C + E(v_i,v_c).C + v^kd_i.w_C) < v_t.f then
12        v_t.h_D = v_c.h_D + E(v_i,v_c).D;
13        v_t.h_A = v_c.h_A * E(v_i,v_c).A;
14        v_t.h_C = v_c.h_C + E(v_i,v_c).C;
15        v_t.f = v_c.h_C + v^kd_i.w_C;
16     end if
17  end if
18  if (v_c.h_D + E(v_i,v_c).D + v^ka_i.w_D) > κ_D
      or (v_c.h_A * E(v_i,v_c).A * v^ka_i.w_A) < κ_A then
19     if (v_c.h_C + E(v_i,v_c).C + v^ka_i.w_C) < v_t.f then
20        v_t.h_D = v_c.h_D + E(v_i,v_c).D;
21        v_t.h_A = v_c.h_A * E(v_i,v_c).A;
22        v_t.h_C = v_c.h_C + E(v_i,v_c).C;
23        v_t.f = v_c.h_C + v^ka_i.w_C;
24     end if
25  end if
26  If v_t.f == ∞ then
27     return;
28  end if
29  if v_i not in T then
30     add v_i to T;
31     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;
32  else if v_t.f < v_i.f then
33     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;
34  end if
```

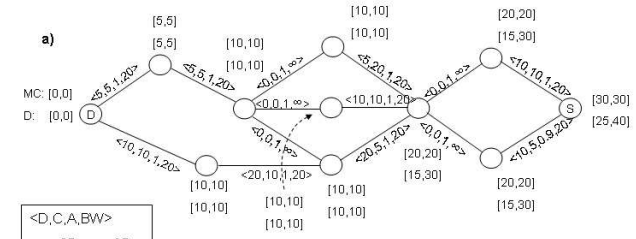
Fig. 15. *MIN_Search_Full_Update* function

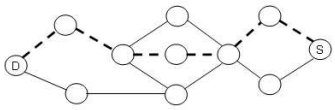
(Figure 16c).

Theoretically, the runtime complexity of our hybrid composition algorithm is identical to that of the simple composition algorithm. However, the process in worst case can take up to O(5($|E| + |N|log|N|$)) instead of O(2($|E| + |N|log|N|$)).

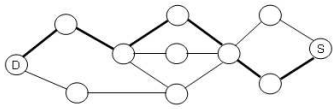
C. Network Adaptation and Mobility Support

Throughout the course of a communication session, the network performance may vary significantly over time. Even with the best service assurance schemes, one or more domains carrying the service traffic may fail to deliver their promised QoS performance. When such an event occurs, self-adaptation should take place in seeking an alternative communication path that satisfies the original QoS requirements, while causing as little service disturbance as possible. In this section, we present the network adaptation





b) Result of Simple SComp



c) Result of Hybrid SComp

Fig. 16. Example of the Hybrid Service Composition Algorithm

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});

Fig. 17. The Network Adaptation Algorithm

algorithm based on our search heuristic. The objective of the algorithm is to find a minimal cost alternative path p_{new} that utilizes as much of the old path p_{old} as possible.

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 17 details the algorithm *SC_Adaptation*. Interestingly, the cost minimization strategy also ensures that many parts of the old path is included in the new alternative path. With our approach, the edges corresponding to the defective domains are not removed from the graph, but rather updated to reflect the new domain condition. We note that setting the costs of these edges to 0 does not prevents the algorithm from selecting a defective domain. It is designed in this way such that it is possible to obtain an alternative path that includes some/all of the defective domains. For example, when a domain in an existing domain composition suffers QoS deterioration, it may be possible to raise the QoS service classes of its upstream and/or downstream domains in the existing composition and hence absorb the QoS deterioration, rather than negotiate for new domain connections going around the defective domain, which is likely to be slow and expensive. Figure 18a shows p_{old} in which two domains fail to deliver

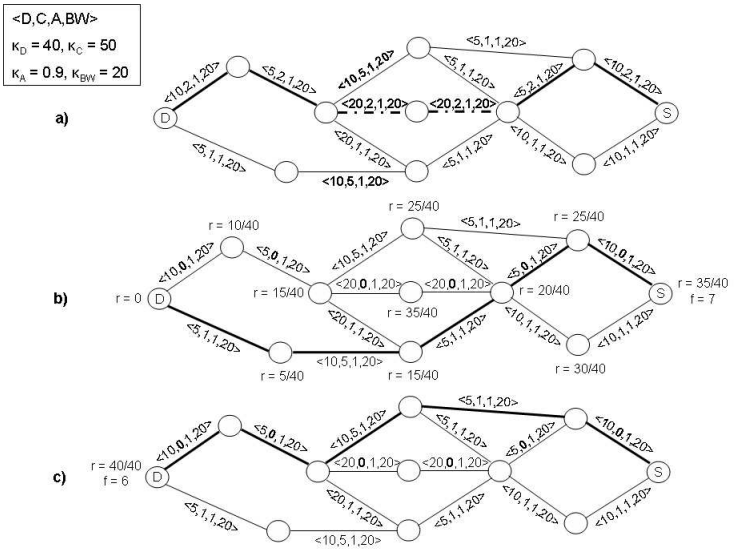


Fig. 18. Example of the Network Adaptation Algorithm

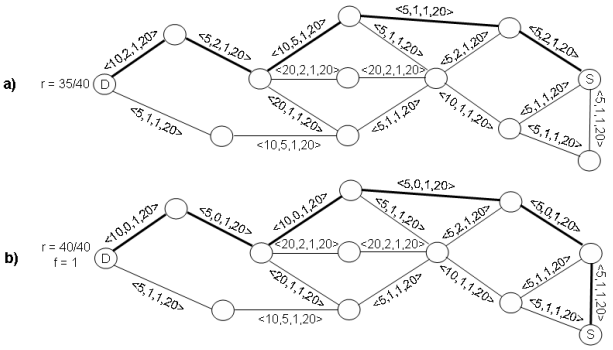


Fig. 19. Example of Mobility Support

their promised delay requirements. The graph further shows two other domains that are willing to deliver better delay assurances at higher price. Figure 18b shows the result of running the *MC_Search* function which generated a feasible path. Figure 18c shows the new path p_{new} (returned by the algorithm) that improves on the cost of the path in 18b. The new path cost is the additional cost that must be absorbed by the violating domains in order to maintain the service.

When the service components roam across domain boundaries, inter-domain network adaptation should also take place. Such service mobility is a specific case of adaptation. First, the expanded domain graph is updated to include the components' new locations, and then *SC_Adaptation* is run. Again, the algorithm will attempt to minimize the service disturbance by reusing parts of the old path. Figure 19 illustrates this scenario. Figure 19a shows the existing path from S to D before S moves. Figure 19b shows the result of running the adaptation algorithm after S moves. In this instance, the new path is a straightforward extension of the old path.

Our adaptation scheme can provide hard QoS guarantees over domains with relative service differentiations. The algorithm does not rely on each domain to fulfill its QoS promises, but actively seek

alternative domain compositions that can satisfy the end-to-end QoS requirements despite changes in network weather. Clearly, such adaptation is only possible if there exists a feasible domain composition at the time of adaptation. Moreover, the adaptation mechanism is inherently self-optimizing, in that when a domain experiences performance degradation, the mere act of adaptation lessens the traffic load on the domain by redirecting traffic elsewhere, and helps in distributing load across the networks.

V. EXPERIMENTATIONS

In this section, we evaluate the runtime performance, success rate and effectiveness of our hybrid composition algorithm compared with prominent solutions in the literature. For completeness, not only are the k-MCOP algorithms presented here, we also include the SC-SON algorithm [7] proposed for service composition in Service Overlay Networks.

Two sets of domain graphs are used in our study. The first domain graph is constructed based on the ANSNET (ANSNET) topology (Figure 20), as presented in Chen's work [19]. The second domain graph is constructed based on the Cable&Wireless (CNW) network topology in the US and UK (Figure 21). The CNW topology is of substantially larger size and contains significant number of service classes. In the illustrations, links with boxes depict domains with three service classes, while the other links have a single service class. The actual number of nodes on the graph is larger than shown since the service class nodes are omitted. At the start of each simulation run, the weight of each link is randomly distributed with relative differentiation between different service classes. A service class of higher cost offers better delay and availability values than a lower service class.

The experiments are performed under Redhat Linux 9 on a Pentium 4 2.4GHz PC with 1GB memory. In each run, the graph weights are randomly generated as specified above and each algorithm is then asked to find the minimal cost feasible path between source (S) and destination (D) on ANSNET and between

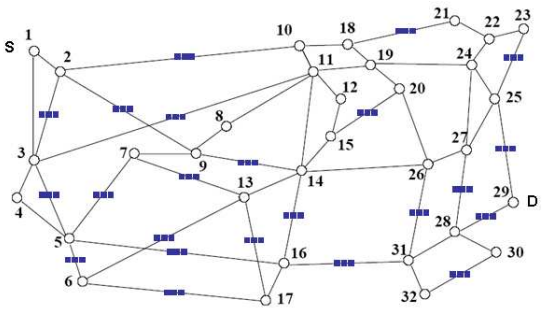


Fig. 20. Domain Graph of ANSNET Topology (ANSNET)

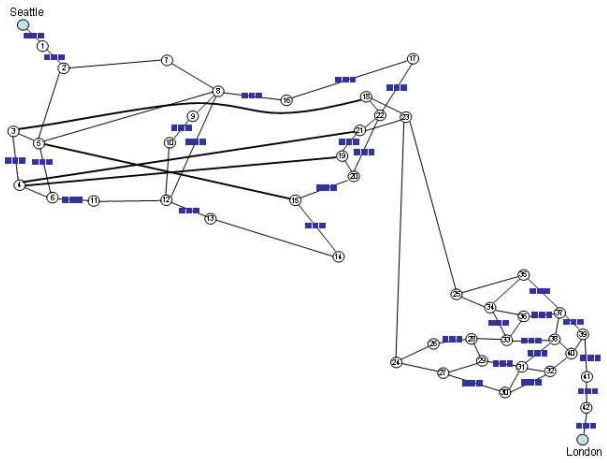


Fig. 21. Domain Graph of Cable&Wireless Topology (CNW)

the node in Seattle and the node in London on CNW. As the base case, depth-first search is performed to find the minimal cost path on the graph.

The performance of each solution is evaluated using randomly generated graphs of size $|N|$. The path length between source and destination is at least 15 links. Figure 22 plot the average runtime of the algorithms over 100 runs for each graph size. As shown through theoretical studies, the LPH and Chen algorithm performs significantly worse than the other algorithms. The performance of our algorithm is roughly on par with TAMCRA when K is set to 3 (i.e. three non-dominate paths are kept at each node). As the expanded graphs of both the ANSNET and CNW topology has 100 or more nodes, we are particularly interested in algorithms with acceptable runtime speed on graphs of this size. Excluding Chen and LPH, most of the algorithms can return a result within 100 millisecond on graphs of size 100. The Chen's algorithm is able to return a result under 200 millisecond. For sake of comparison, depth first search algorithm takes on average 8 minutes with graphs of 50 nodes.

The success rate of a k-MCOP algorithm is the percentage of time the algorithm could return a feasible path when feasible path(s) exist in a graph. To evaluate the success rate of k-MCOP algorithms, most simulation studies in literature apply randomly generated QoS requirements on a graph with random

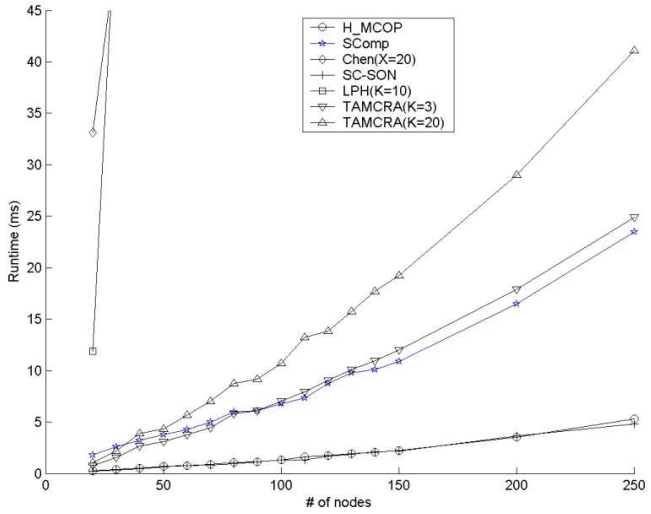


Fig. 22. Runtime Performance on Random Graphs of Varied Size

weight initialization. Such construction does not accurately reflect the success rate of an algorithm, as there is no control over the constraint ratio. More precisely, when the delay requirement of a path is close to the minimum delay bound of the path (i.e. the shortest delay path), one can expect the success rate of an algorithm to drop, as there are few feasible paths. In the following study, we evaluate the success rate of each algorithm over different delay ratios, computed as:

$$DelayRatio = \frac{DelayRequirement}{Minimum\ Delay\ between\ src\ and\ dst} \tag{4}$$

Figure 23 and 24 present the success rate of these algorithms for ANSNET and CNW topology respectively. The difference between ANSNET and CNW topologies are the latter is significantly larger in size and also has many “two-hop loops”. We also consider the impact of attribute correlations on the success rate. Three sets of correlations are evaluated: positive, negative and no correlations between delay and cost. For each delay ratio and attribute correlation, 50 runs are conducted. On the ANSNET topology, our SComp algorithm is able to achieve close to 100% success rate even under tight constraint bound. The TAMCRA algorithm also performs well. On the CNW topology, our hybrid algorithm again shows good performance, but as expected, when constraint bound is tight, the success rate is adversely affected, especially with negative correlation. The TAMCRA algorithm outperforms ours under tight bound conditions, however,

the runtime of TAMCRA (K=20) is significantly higher than our hybrid service composition algorithm. In general, we see that with negative correlations, which is the case in practice, our algorithm can fail to find feasible paths, especially under tight constraints. This problem is somewhat less severe with tunable algorithms such as TAMCRA and LPH, both of which can increase the parameter size for increased success rate, although with significant increase in runtime performance.

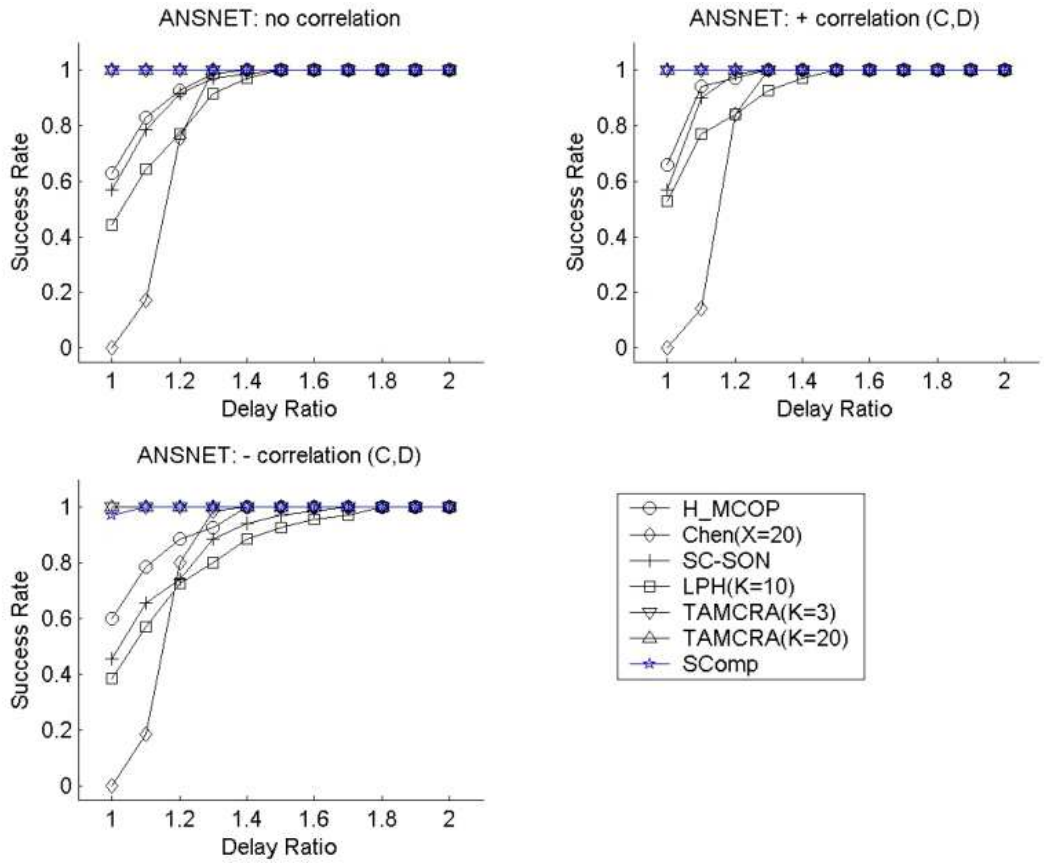


Fig. 23. Success Ratio of Algorithms on ANSNET Topology

The effectiveness of the algorithms is demonstrated in Figure 25 and 26 over 50 runs. The advantage of the forward minimization step is apparent. Our hybrid composition algorithm often finds the optimal solution or near optimal solution. By optimal solution, we mean a feasible path whose cost is minimal. In comparison, the other algorithms perform much worse. In this study, the error percentage ϵ is computed based on the marginal cost difference between the optimal solution C_{opt} (as obtained via depth-first search)

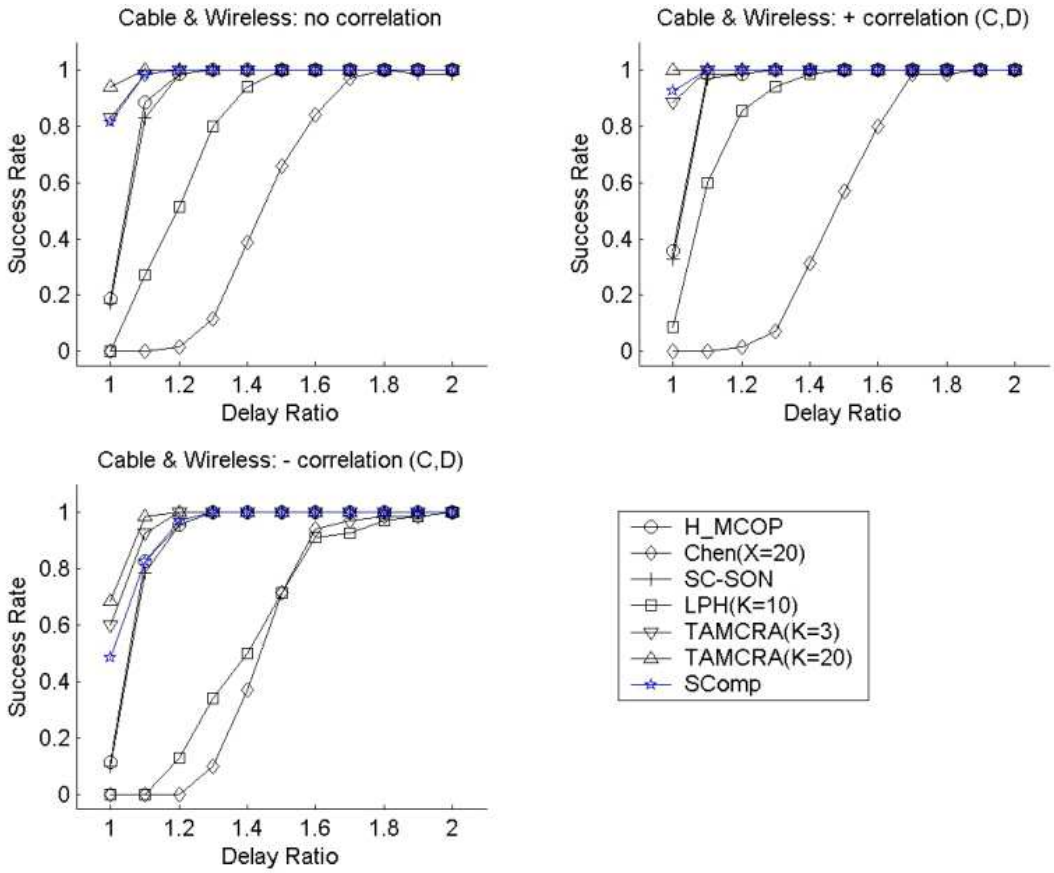


Fig. 24. Success Ratio of Algorithms on CNW Topology

and the solution obtained from the algorithms C_{algo}:

$$\epsilon = (C_{algo} - C_{opt})/C_{opt} \tag{5}$$

Figure 27 illustrates the percentage of times each algorithm is able to find the optimal solution on CNW topology. It further demonstrates the effectiveness of our algorithm in addressing the "two-hop loop" problem, compared with other prominent algorithms.

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. Our studies show

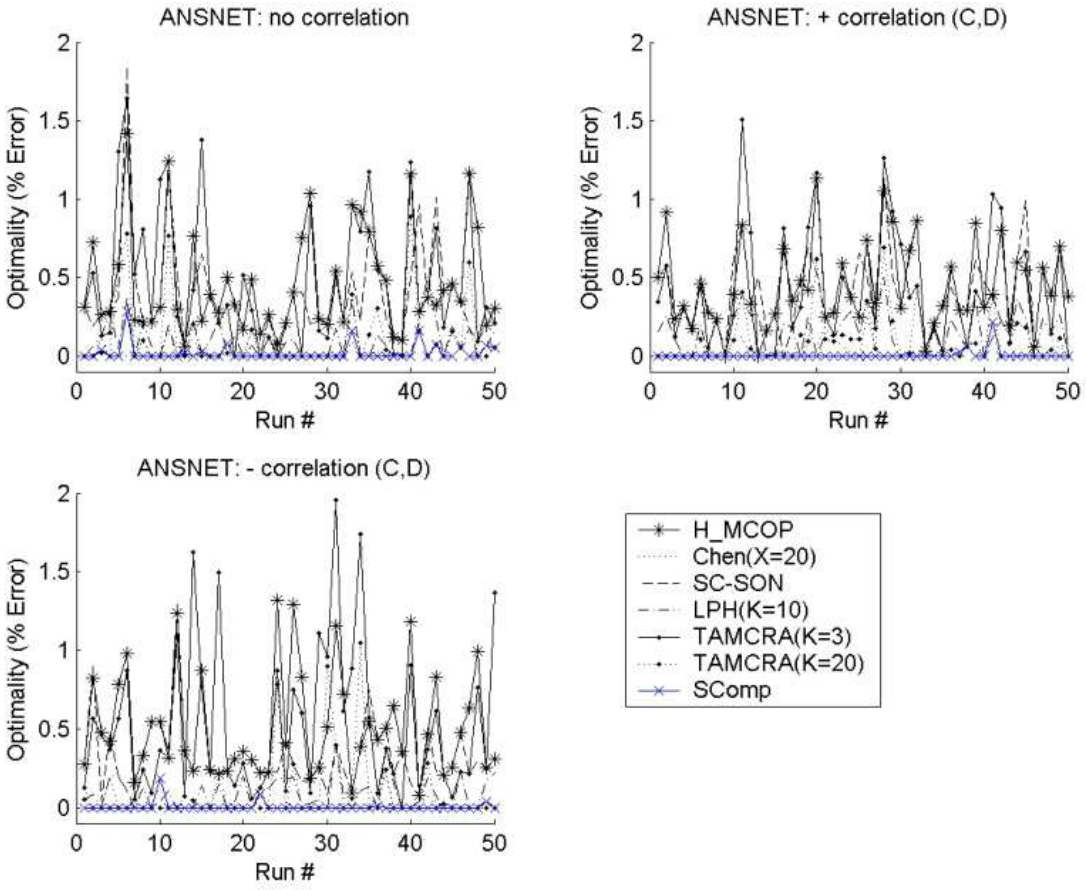


Fig. 25. Relative optimality of Algorithms on ANSNET Topology

that 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 fast runtime performance and good success rate compared to the classic k-MCOP algorithms, our hybrid composition algorithm is able to find minimal cost feasible paths or near minimal cost paths. In particular, when subject to large scale networks with many service classes, our algorithm significantly outperforms the others. Moreover, the network adaptation scheme achieves good path reuse rate, while finding alternative feasible paths with low switching cost.

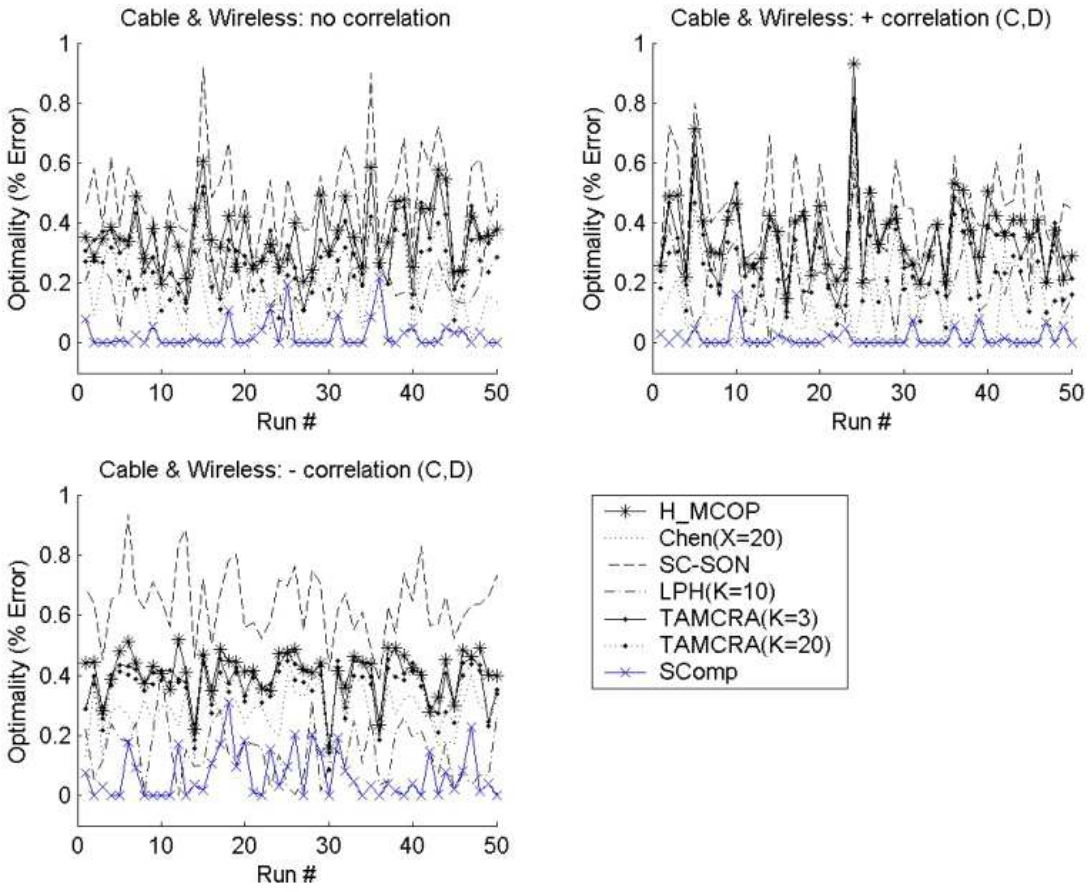


Fig. 26. Relative optimality of Algorithms on CNW Topology

VI. Conclusion

Following the autonomic communication principle, we first presented a framework for autonomic network service composition in this paper. The ultimate aim of such a framework is to create self-managed end-to-end inter-domain communication paths with QoS assurance. The self-configuration, self-optimization and self-adaptation behavior of this framework revolves around solving the service composition and adaptation problems. 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 abstraction, we reduced the problem to k-MCOP problem and shown the inadequacies of classic k-MCOP solutions in this context. Based on our analysis, a set of new service composition and adaptation algorithms were developed and our in-depth

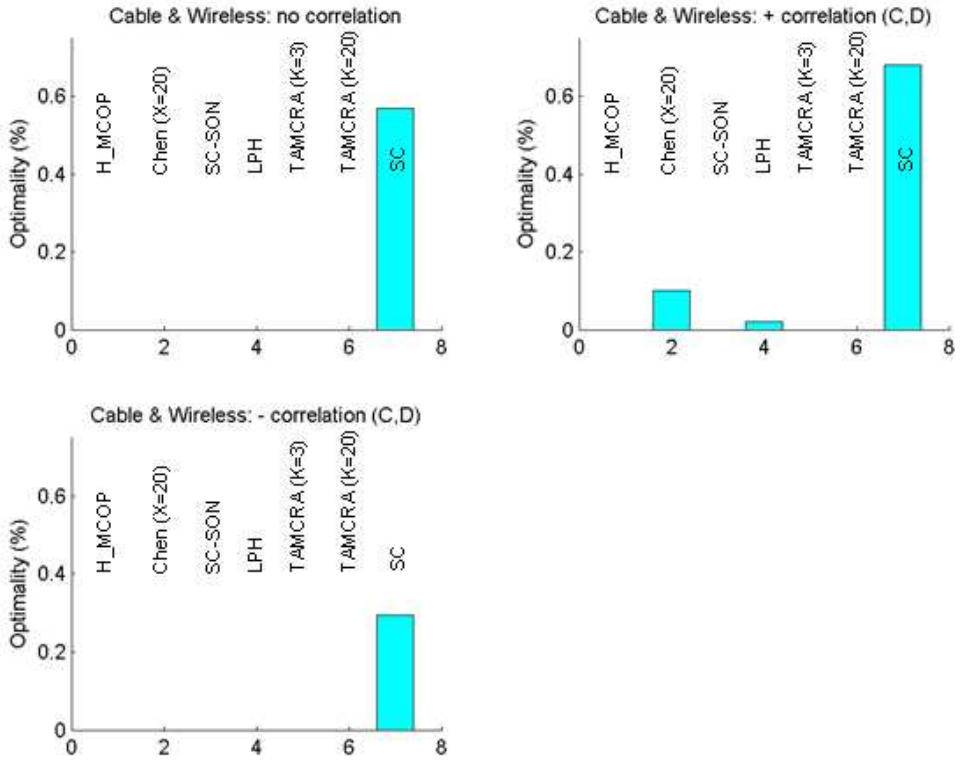


Fig. 27. % optimality of Algorithms on CNW Topology

experimentation showed that the SComp algorithm has very high probability of finding the optimal or near optimal solution and generates solutions with path cost much lower than those generated by current best algorithms. Furthermore, our algorithm only relies on per hop domain information obtainable from BGP information. In the event of QoS violation, our algorithm can quickly find an alternative low cost path that reuses much of the original path. Our composition and adaptation scheme is capable of providing hard QoS guarantees over domains with soft guarantees and exhibit the property of self-optimization via load distribution in adaptation. In the autonomic provisioning framework, the algorithm provides the intelligence to automatically perform domain composition and adaptation based on domain information gathered by the monitoring facilities. The required provisioning operations are to be carried out by the provisioning mechanisms.

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 seek mathematical bound on the near-optimality and feasibility of our algorithms. On the application side, a prototype implementation of the autonomic service provisioning framework should be undertaken to evaluate the applicability of our framework and algorithms in real networks.

REFERENCES

[1] J. Wroclawski, "The use of RSVP with IETF integrated services," RFC2210, September 1997.

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

[3] J. Kephart and D. Chess, "The vision of autonomic computing," *IEEE Computer Magazine*, 2003.

[4] J. Xiao and R. Boutaba, "Qos-aware service composition in large scale multi-domain networks," in *Proceedings of the IEEE/IFIP Symposium on Integrated Network and System Management (IM'05)*, 2005.

[5] 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.

[6] 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*, vol. 30, no. 5, pp. 311–327, May 2004.

[7] 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.

[8] 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.

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

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

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

[12] 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.

[13] 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/.

[14] 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.

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

[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] 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)*. IEEE, May 2003, vol. 3, pp. 1912–1916.

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

[19] S. Chen and K. Nahrstedt, "On finding multi-constrained paths," in *Proceedings of IEEE International Conference on Communications (ICC98)*. IEEE, June 1998, vol. 2, pp. 874–879.

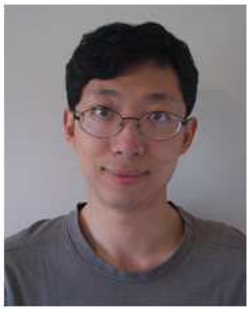
[20] X. Yuan, "Heuristic algorithms for multiconstrained quality-of-service routing," *IEEE/ACM Transactions on Networking*, vol. 10, no. 2, April 2002.

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

[22] P. Van Mieghem, H. De Neve, and F. Kuipers, "Hop-by-hop quality of service routing," *Computer Networks*, , no. 37, pp. 407–423, 2001.

[23] T. Korkmaz and M. Krunz, "Multi-constrained optimal path selection," in *Proceedings of IEEE INFOCOM 2001*. IEEE, April 2001, vol. 2, pp. 834–843.

[24] F. A. Kuipers, T. Korkmaz, M. Krunz, and P. Van Meighem, "Performance evaluation of constraint-based path selection," *IEEE Network*, vol. 18, no. 5, pp. 16–23, September/October 2004.



Jin Xiao received the BSc. degree in computer science from University of Calgary, Canada. He is currently a PhD. candidate at School of Computer Science, University of Waterloo, Canada. His research interests include autonomous network service provisioning and composition, network QoS awareness and adaptation, and autonomous network service management architectures.

Raouf Boutaba is currently an Associate Professor in the School of Computer Science of the University of Waterloo. He conducts research in the areas of network and distributed systems management and resource management in multimedia wired and wireless networks. Dr. Boutaba is the Chairman of the Working Group on Networks and Distributed Systems of the International Federation for Information Processing (IFIP), the Vice Chair of the IEEE Communications Society Technical Committee on Information Infrastructure, and the Director of standards board of the IEEE Communications Society and the editor of many IEEE journals, including guest editor for a number of JSAC issues.

LIST OF FIGURES