A Service Query Dissemination Algorithm for Accommodating Sophisticated QoS Requirements in a Service Discovery System

Liang Zhang[1], Beihong Jin[1]

[1] Institute of Software, Chinese Academy of Sciences,
Hai Dian, Beijing, PRC
{zhangliang1216, jbh}@otcaix.iscas.ac.cn

Abstract. For many service discovery protocols, user service queries need to be disseminated to some service directories within the system for discovering matched services. The scope of dissemination identifies the scope of service discovery. How to determine the service discovery scope needs to be discussed. In this paper, we present a service query dissemination algorithm in our developing service discovery system, Service CatalogNet. In this system, users can pose different QoS requirements upon their service queries in order to manually control the scope of service discovery. The service query dissemination algorithm will dynamically transform the QoS requirements into a set of directories as well as the routing paths to them. The performance analysis shows a sound result of our algorithm.

Keywords: QoS, multicast, service discovery, mobile computing, NP-complete

1 Introduction

Recent trends in mobile and ubiquitous computing have created new requirements for automatic configuration of network devices. Furthermore, the exploding deployment of network devices in diverse environment has increased the need to simplify the network administration for different kinds of networks. In response to these requirements, a variety of new protocols have been proposed, which attempt to provide automatic discovery and configuration of network devices and services. These protocols are called *service discovery protocol*s [1].

The majority service discovery protocols rely on a special component called *service directory*, or *directory* in short, for maintaining service information. Users post service queries to a directory for services they need. To enhance scalability and robustness, the whole set of service information within a system is normally distributed to a set of directories across the system. In this circumstance, user service queries are first sent to a specific directory called *access directory* and then dynamically forwarded to a set of other directories for service discovery. The challenge is how to select an appropriate subset of directories within the system as well as the routing paths to them. Within the entire spectrum of directory selection, there are two extreme

strategies: *full-directory strategy*, where the user desires the service query to be processed at all the directories within the system, and *no-directory strategy*, where the system restricts the service query only at the access directory for minimizing the transmission cost. Both the user requirement of maximizing the processing directories and the system requirement of minimizing the transmission cost are referred to as QoS requirements. Our problem can be defined as dynamically determining the directories and routing paths for disseminating service queries with the consideration of an ordered list of user and system QoS requirements according to the priority. For example, a service query will be disseminated through the minimum spanning tree covering all the directories within the system in order to fulfill both the above mentioned QoS requirements with the user QoS requirement taking the higher priority than the system QoS requirement.

In this paper, we present a service query dissemination algorithm in our developing service discovery system, Service CatalogNet. In this system, users can pose different QoS requirements upon their service queries in order to manually control the scope of service discovery. The service query dissemination algorithm will dynamically transform the QoS requirements into a set of directories as well as the routing paths to them. The rest of the paper is organized as follows. Section 2 presents a classification scheme for QoS requirements. Section 3 formally models the problem and proposes a solution to it. Section 4 conducts the performance analysis. The final section concludes the paper.

2 QoS Requirement Classification

Before proposing the general algorithm for accommodating QoS requirements in Section 3, let us first study the characteristics of QoS requirements. All QoS requirements have to be expressed in some measurable QoS metrics. There are two types of QoS metrics: link state metrics and server state metrics, which measure the state information of links and servers, respectively.

The link state QoS requirements can be categorized in three dimensions as shown in Fig. 1. For the first dimension, the link state metrics can be divided into subtypes, within which additive, multiplicative and concave are the most common ones [2]. Let $ls(n_i, n_j)$ denote the link state between the two servers n_i and n_j. For any routing path $rp = (n_1, n_2, \ldots, n_{k-1}, n_k)$, the link state metric ls is

- additive, if $F(ls(rp)) = F(ls(n_1, n_2)) + \ldots + F(ls(n_{k-1}, n_k))$; or
- multiplicative, if $F(ls(rp)) = F(ls(n_1, n_2)) \times \ldots \times F(ls(n_{k-1}, n_k))$; or
- concave, if $F(ls(rp)) = min\{F(ls(n_1, n_2)), \ldots, F(ls(n_{k-1}, n_k))\}$,

where F is a function over link states. For example, transmission delay is an additive QoS metric because the transmission delay of a routing path is the summation of those of the constituent links (Here $F(x) = x$); link-failure rate is a multiplicative QoS metric because the compliment of the link-failure rate of a routing path is the multiplication of the compliment of those of the constituent links (Here $F(x) = 1 - x$); network bandwidth is a concave QoS metric because the network bandwidth of a routing path is the minimum of those of the constituent links (Here $F(x) = x$). Besides, a general link state metric can be expressed by the following equation

$F(ls(rp)) = Func(F(ls(n_1, n_2)), \ldots, F(ls(n_{k-1}, n_k)))$,
where *Func* represents a special function.

For the second dimension, a classification is made between whether the QoS requirement is to find the best routing path or any routing path bounded by a certain value. For example, "transmission delay is minimum" aims to find the routing path with the minimum transmission delay and we call this type of QoS requirements QoS optimization requirements. Another example, "transmission delay < 30 seconds" aims to find the routing path with the transmission delay smaller than 30 seconds and we call this type of QoS requirements QoS constrained requirements.

For the third dimension, a QoS requirement may only pose constraint to the routing path to each receiver; or it may not concern any individual routing path, but desires the overall routing paths to all the receivers to possess a certain characteristic. For example, "transmission cost < 10" specifies the transmission cost to each receiver should be smaller than 10, while "the summation of transmission cost < 100" guarantees the cumulated transmission cost to all the receivers should be smaller than 100. Another example of group QoS requirement is "the variance of transmission delay < 1 second" meaning that the maximum difference of transmission delay to all the receivers should be smaller than 1 second, which is particular important to applications like VoIP Conference. Of course, it is possible for other general group QoS requirements.

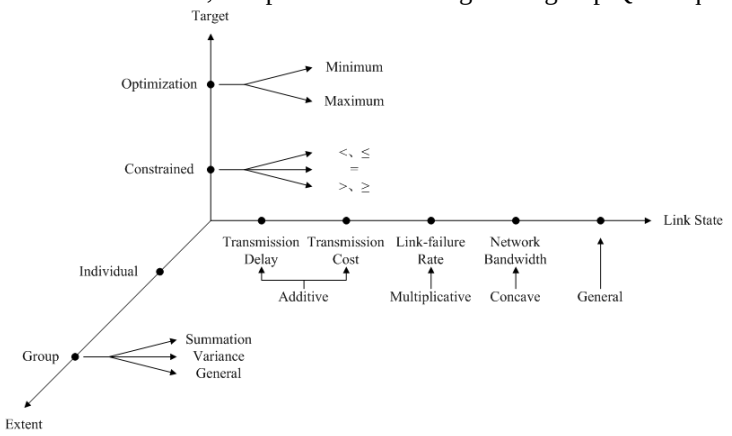

Fig. 1. Link state QoS requirement classification

Similar to the link state QoS requirements, the server state QoS requirements can also be categorized in three dimensions as shown in Fig. 2. The first dimension depicts different server state metrics. The second dimension differentiates optimization and constrained requirements. The third dimension considers either each individual server state or the overall server states of all the servers.

With the above classification scheme defined, any QoS requirement can be modeled in the following format

QoS Requirement = [Extent] + QoS Metric + Target,

where the omission of the optional Extent part represents an individual QoS requirement. With the scheme, we can easily identify different types of QoS requirements with each type representing a specific problem with a certain solution:

- Individual link state QoS requirements are similar to the shortest path tree problem and can be solved by applying the Dijkstra's algorithm either directly or with slight modification.
- Group link state QoS requirements are similar to the minimum spanning tree problem and can be solved by applying the Prim's algorithm either directly or with slight modification.
- Individual server state QoS requirements are simple and can be solved by directly picking the correct servers.
- Group server state QoS requirements are also simple and can be solved by picking the correct sets of servers.

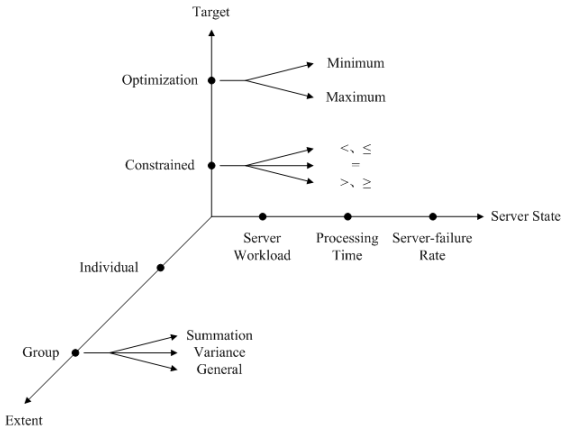

Fig. 2. Server state QoS requirement classification

Although it is easy for a single QoS requirement, it may become extremely challenge to solve even two QoS requirements. [3] points out there are three possible combination of two QoS requirements that will result in NP-complete complexity. Our algorithm aims at providing a general solution for solving a sequence of QoS requirements with polynomial complexity.

3 A General Algorithm

Let us first model our problem formally. Like all the other problems, our problem has input and output. There are five input parameters in our problem (V, v_a, S, L, Q), where V represents all the DSs within the system, v_a stands for the access directory, S keeps the global server state information, L records the global link state information and Q maintains the list of QoS requirements. Notice that we do not introduce an E parameter for all the links within the system as many other papers do. This is because our system adopts a mesh topology that every DS knows all the other DSs, i.e., E is implicitly defined by V. The output of our problem is (V', P'), which respectively means the set of DSs within the service discovery scope as well as the routing paths to them. With the output, the access directory can easily perform source routing for disseminating the service query.

The input parameter Q needs a bit more explanation. Each QoS requirement in Q is associated with a priority. The higher the priority is, the topper a QoS requirement appears in Q, and the earlier it is processed by the general algorithm. There are three types of QoS requirements with different purposes in Q: *user QoS requirements*, *system QoS requirements* and *default QoS requirements*. The service requestor only issues user QoS requirements to manually control the service discovery scope. However, it is usually unwise to leave clients take full control of the system behavior. Certain administrative control is necessary to protect the system from exhausting the resources. For example, the administrator may restrict all the service discovery within 1 kilometer. We name these QoS requirements system QoS requirements. The user QoS requirements have higher priority than the system QoS requirements. To ensure the uniqueness of the output, it is sometimes necessary to append default QoS requirements at the end of Q. For example, the general algorithm suggests two options of DSs and routing paths after processing the user QoS requirements and the system QoS requirements. Then a default "the summation of transmission cost is minimum" QoS requirement may break the tie and leave a unique output. The default QoS requirements must belong to the optimization type. They are added one by one at runtime in a predefined order until the output is unique. The default QoS requirements have the least priority.

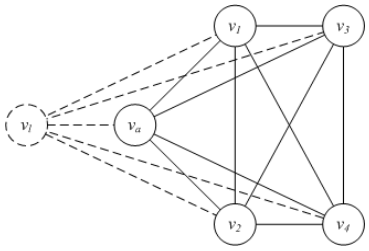

Fig. 3. The logical server

The last thing necessary for mention before presenting the general algorithm is that although the access directory by itself is a DS, we consider there is a separate logical server as shown in Fig. 3 receiving the service query and performing the calculation, and treat the access directory the same as other DSs. That implies the possibility that even the access directory is not included for the dissemination. When the access directory does be included, a logical message is sent to it and executed the same way as a physical message being sent to another DS. We denote the logical server v_l.

Now let us present our general algorithm with the pseudo code below. The main idea is simple. We design a routine procedure for sequentially processing each QoS requirement. During each processing, the algorithm applies the corresponding solution described in the last section either directly or with slight modification for the specific QoS requirement type. The graph theory and the set theory are heavily depended on in our algorithm.

- At the beginning, there is a mesh graph $\{V, E\}$ where E is implicitly defined by V
- With the logical server v_l included, another mesh graph $G = (\{V, v_l\}, E \cup \{v_l v_i \mid v_i \in V\})$ is defined

- Define $VP = \{V, P\}$ where P[1] $= \{P(v_i) \mid v_i \in V$ and $P(v_i)$ includes all the non-loop routing paths from v_l to v_i in $G\}$
- Define $VP_I = \{VP\}$ and $VP_O = \{\}$
- For the next prioritized QoS requirement q in Q
  - Switch q's major type
    - Case: individual link state optimization
      - For each $VP_i \in VP_I$
        - Define $VP_o.V = VP_i.V$
        - If $VP_i.P = \forall$, then it is a mesh graph
          - Apply the Dijkstra's algorithm over the mesh graph to construct the shortest path tree T
          - $VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o)$ includes the routing path from v_l to v_o in $T\}$
        - Else
          - $VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o)$ includes the optimum routing path in $VP_i.P(v_o)\}$
        - Add VP_o to VP_O
    - Case: individual link state constrained
      - For each $VP_i \in VP_I$
        - Define VP_o
        - If $VP_i.P = \forall$, then it is a mesh graph
          - Apply the Dijkstra's algorithm over the mesh graph to construct the shortest path tree T until further growth of the tree will have the routing path not fulfill the constraint
          - $VP_o.V = T.V$ and **$VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o)$ includes all the routing paths from v_l to v_o in the mesh graph fulfilling the constraint$\}$**
        - Else
          - $VP_o.V = \{v_o \mid v_o \in VP_i.V$ and $\exists\ p \in VP_i.P(v_o)$, p fulfills the constraint$\}$ and $VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o)$ includes all the routing paths in $VP_i.P(v_o)$ fulfilling the constraint$\}$
        - Add VP_o to VP_O
    - Case: group link state optimization
      - For each $VP_i \in VP_I$
        - Apply the Prim's algorithm over the graph defined by VP_i to construct the minimum spanning tree T
        - $VP_i.P = \{P(v_i) \mid v_i \in VP_o.V$ and $P(v_i)$ includes the routing path from v_l to v_o in $T\}$
      - Define $VP_o = VP_{i\text{-opt}}$ where $VP_{i\text{-opt}} \in VP_I$ and $VP_{i\text{-opt}}.V$ has the optimum overall link states
      - Add VP_o to VP_O
    - Case: group link state constrained
      - For each $VP_i \in VP_I$

[1] The initial P is conceptual as it is impractical, if not impossible, to maintain it in the memory. A symbol $\forall$ is used to denote this concept.

- **Define VP_{TMP} include all VP_{tmp} with $VP_{tmp}.V \subseteq VP_i.V$ and $VP_{tmp}.P \subseteq VP_i.P$ has the overall link states fulfilling the constraint**
- Merge VP_{TMP} to VP_O

- Case: individual server state optimization
  - For each $VP_i \in VP_I$
    - Define VP_o
    - If $VP_i.P = \forall$, then it is a mesh graph
      - $VP_o.V = \{v_{o\text{-}opt} \mid v_{o\text{-}opt} \in VP_i.V$ has the optimum server state$\}$ and $VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o) = \{v_l v_o\}\}$
    - Else
      - Define $VP_{tmp}.V = \{v_{tmp} \mid v_{tmp} \in VP_i.V$ and $VP_i.P(v_{tmp})$ contains the routing path $v_l v_{tmp}\}$ and $VP_{tmp}.P = \{P(v_{tmp}) \mid v_{tmp} \in VP_{tmp}.V$ and $P(v_{tmp}) = \{v_l v_{tmp}\}\}$
      - $VP_o.V = \{v_{o\text{-}opt} \mid v_{o\text{-}opt} \in VP_{tmp}.V$ has the optimum server state$\}$ and $VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o) = VP_{tmp}.P(v_o)\}$
    - Add VP_o to VP_O
- Case: individual server state constrained
  - For each $VP_i \in VP_I$
    - Define VP_o
    - If $VP_i.P = \forall$, then it is a mesh graph
      - $VP_o.V = \{v_o \mid$ the server state of $v_o \in VP_i.V$ fulfills the constraint$\}$ and $VP_o.P = \forall$
    - Else
      - $VP_o.V = \{v_o \mid$ the server state of $v_o \in VP_i.V$ fulfills the constraint$\}$ and $VP_o.P = \{P(v_o) \mid v_o \in VP_o.V$ and $P(v_o) = VP_i.P(v_o)\}$
      - For each $v_o \in VP_o.V$
        - Delete all the routing paths from $VP_o.P(v_o)$ that involve DSs not belonging to $VP_o.V$
        - Delete v_o from $VP_o.V$ if $VP_o.P(v_o) = \varnothing$
      - Loop the previous step until VP_o no longer changes
    - Add VP_o to VP_O
- Case: group server state optimization
  - Define $VP_o = VP_{i\text{-opt}}$ where $VP_{i\text{-opt}} \in VP_I$ and $VP_{i\text{-opt}}.V$ has the optimum overall server states
  - Add VP_o to VP_O
- Case: group server state constrained
  - For each $VP_i \in VP_I$
    - **Define VP_{TMP} include all VP_{tmp} with $VP_{tmp}.V \subseteq VP_i.V$ has the overall server states fulfilling the constraint** and $VP_{tmp}.P = \{P(v_{tmp}) \mid v_{tmp} \in VP_{tmp}.V$ and $P(v_{tmp}) = VP_i.P(v_{tmp})\}$
    - For each $VP_{tmp} \in VP_{TMP}$
      - For each $v_{tmp} \in VP_{tmp}$
        - Delete all the routing paths from $VP_{tmp}.P(v_{tmp})$ that involve DSs not belonging to $VP_{tmp}.V$
        - Delete VP_{tmp} from VP_{TMP} if $VP_{tmp}.P(v_{tmp}) = \varnothing$ and break
    - Merge VP_{TMP} to VP_O

- $VP_I = VP_O$, $VP_O = \{\}$
- While VP_I includes more than one member
  - For the next predefined default QoS requirement q
    - Switch q's major type
      - Case: individual link state optimization
        - Same as the corresponding block above
      - Case: group link state optimization
        - Same as the corresponding block above
      - Case: individual server state optimization
        - Same as the corresponding block above
      - Case: group server state optimization
        - Same as the corresponding block above
- $V' = VP_I[0].V$ and $P' = VP_I[0].P$

There are three areas in the pseudo code that cannot be solved in polynomial time as highlighted in boldface. We handle them by approximation as follows.

- Case: individual link state constrained
  Instead of deriving all the routing paths from v_I to v_o fulfilling the constraint, we apply the k-shortest-paths algorithm [4] to only derive k routing paths that best fulfill the constraint.
- Case: group link state constrained
  Apply the Prim's algorithm to construct the minimum spanning tree until further growth of the tree will not fulfill the constraint. Backward the construction of the tree k steps with each step representing an optional solution.
- Case: group server state constrained
  Apply the greedy algorithm to include as many servers as possible until further inclusion will not fulfill the constraint. Backward the greedy algorithm k steps with each step representing an optional solution.

As we can see, the main technique we applied for approximation is to introduce a parameter k to limit the search scope. We call this parameter degree of approximation. The greater k is, the more accurate it is, and the less approximation is.

4 Performance Analysis

In this section, we conduct simulation to evaluate the performance of our algorithm. The performance metric is the average processing time for deriving the set of directories as well as the routing paths to them fulfilling the QoS requirements. We compare the performance of our algorithm with that of two source-based QoS-aware multicast protocols: CST [5] and SunQ [6], both of which deal with the combination of delay-constraint and least-cost QoS requirements and belong to the NP-complete problem identified in [3]. We intentionally post the same two QoS-requirements to our algorithm in order to discover how good our algorithm performs in the NP-complete situation comparing with the benchmark protocols.

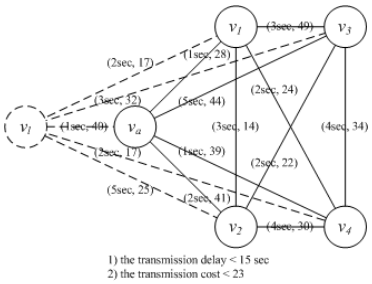

Fig. 4. An example of a network and QoS requirements

The experimental settings are as follows. The program simulates a number of fully-connected directories. For each link between two directories, a pair of integers (*X*, *Y*) are generated for identifying the transmission delay and the transmission cost, respectively. The two QoS requirements are 1) the transmission delay < *Z* seconds and 2) the transmission cost is minimum, where *Z* is also a generated integer. Fig. 4 shows an example of a generated network as well as an example of generated QoS requirements. In our simulation, the transmission delay of a link (*X*) is randomly generated between 1 second and 5 seconds, the transmission cost of a link (*Y*) is randomly generated between 10 and 50, and the transmission cost of a routing path (*Z*) is randomly generated between 10 seconds and 30 seconds. The test-bed computer is ThinkPad x60 with 1.83 GHz Intel Core Duo T4200 CPU and 1 GB RAM. Each experiment is conducted 1000 times to compute the average.

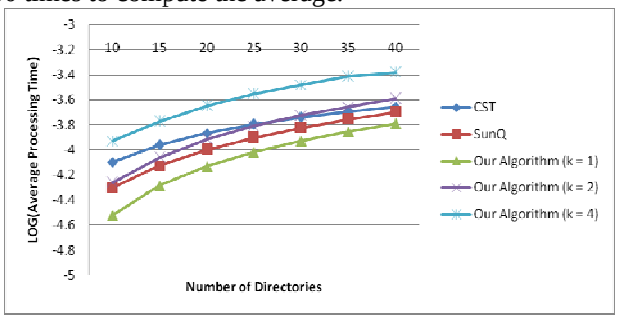

Fig. 5. Performance analysis

Fig. 5 shows the performance results with the number of directories as the x-axis and the logarithm of the average processing time as the y-axis. It is easy to discover that the average processing time of both our algorithm and the two benchmark protocols increase as the number of directories increase. However, this increment is slower than a linear one, which means all of them can effectively reduce the NP-complete problem to polynomial with certain approximation. The second observation is that as the degree of approximation of our algorithm decreases, i.e., *k* increases, the average processing time increases because more routing paths are derived from those fulfilling the delay-constrained QoS requirement. Although theoretically our algorithm is a general solution to QoS-aware routing and cannot make the optimization with respect

to the dealing QoS requirements, we can always maintain the performance of our algorithm by simply adjusting the parameter *k*.

5 Conclusion

In this paper, we present a service query dissemination algorithm in our developing service discovery system, Service CatalogNet. In this system, users can pose different QoS requirements upon their service queries in order to manually control the scope of service discovery. With the defined classification scheme for QoS requirements, the service query dissemination algorithm performs a routine procedure by sequentially processing each QoS requirement. In this way, our algorithm can deal with whatever combination of QoS requirements and can eventually transform them into a set of directories as well as the routing paths to them. With slight modification, our algorithm can also be applied to other systems supporting QoS-aware routing. Finally, the performance analysis shows a sound result of our algorithm.

Acknowledgment

This work was supported by the National Natural Science Foundation of China under Grant No. 60673123 and the National Hi-Tech Research and Development 863 Program of China under Grant No. 2006AA01Z231.

References

1. McGrath R. E., "Discovery and Its Discontents: Discovery Protocols for Ubiquitous Computing", *UIUCDCS-R-99-2132*, Department of Computer Science, University of Illinois Urbana-Champaign, March 2000
2. Chen S. G., "Routing Support for Providing Guaranteed End-to-end Quality-of-service", *Ph.D. Thesis*, Department of Computer Science, University of Illinois Urbana-Champaign, May 1999
3. Chen S. G. and Nahrstedt K., "An Overview of Quality-of-service Routing for the Next Generation High-speed Networks: Problems and Solutions", *IEEE Network*, 12(6):64-79, November 1998
4. Macgegor M. H. and Grover W. D., "Optimized k-shortest-paths Algorithm for Facility Restoration, *Software Practice and Experience*, 24(9):823-828, September 1994
5. Kompella V. P., Pasquale J. C. and Polyzos G. C., "Multicast Routing for Multimedia Communication," *IEEE/ACM Transactions on Networking*, 1(3):286- 292, June 1993
6. Sun Q. and Langendorfer H., "A New Distributed Routing Algorithm with End-to-end Delay Guarantee", *Workshop on Protocols for Multimedia Systems*, October 1995