

A Distributed Approach for QoS Service Multicast with Geometric Location Awareness

Jingwen Jin and Klara Nahrstedt • University of Illinois at Urbana-Champaign

The service routing problem emerged from the distributed and composable services model. Using service multicast instead of service unicast for group delivery saves bandwidth and resources. The authors describe a distributed approach for building QoS-assured, delay-efficient service multicast trees that uses geometric location information and explore hybrid multicasting (service multicasting plus data multicasting).

The composable services model, which lets developers compose individual services to perform complex tasks, is increasingly used to achieve flexibility in software applications^{1,2}. Assuming services are widely distributed in networks, finding efficient service paths or trees that meet not only quality-of-service requirements, but also service requirements, is important for enabling seamless provisioning of integrated services despite the fact that an integrated service might be available as separate components in distributed hosts.

The literature has addressed unicast service routing reasonably well. In this article, we concentrate on the less investigated, more challenging problem: QoS service multicast routing. The problem appears, for instance, in video distribution applications that involve a single sender but multiple receivers, each requiring a different customization of the original video stream (that is, every end-to-end path must encompass certain transformational services in proper order) in the middle of the network. If the receivers' service requests have common prefixes, instead of delivering the data through multiple individual service paths, one service multicast tree exploring service path sharing could deliver the data, saving both network bandwidth and machine resources. We term this delivery model *service multicast*. Figure 1 gives an example scenario.

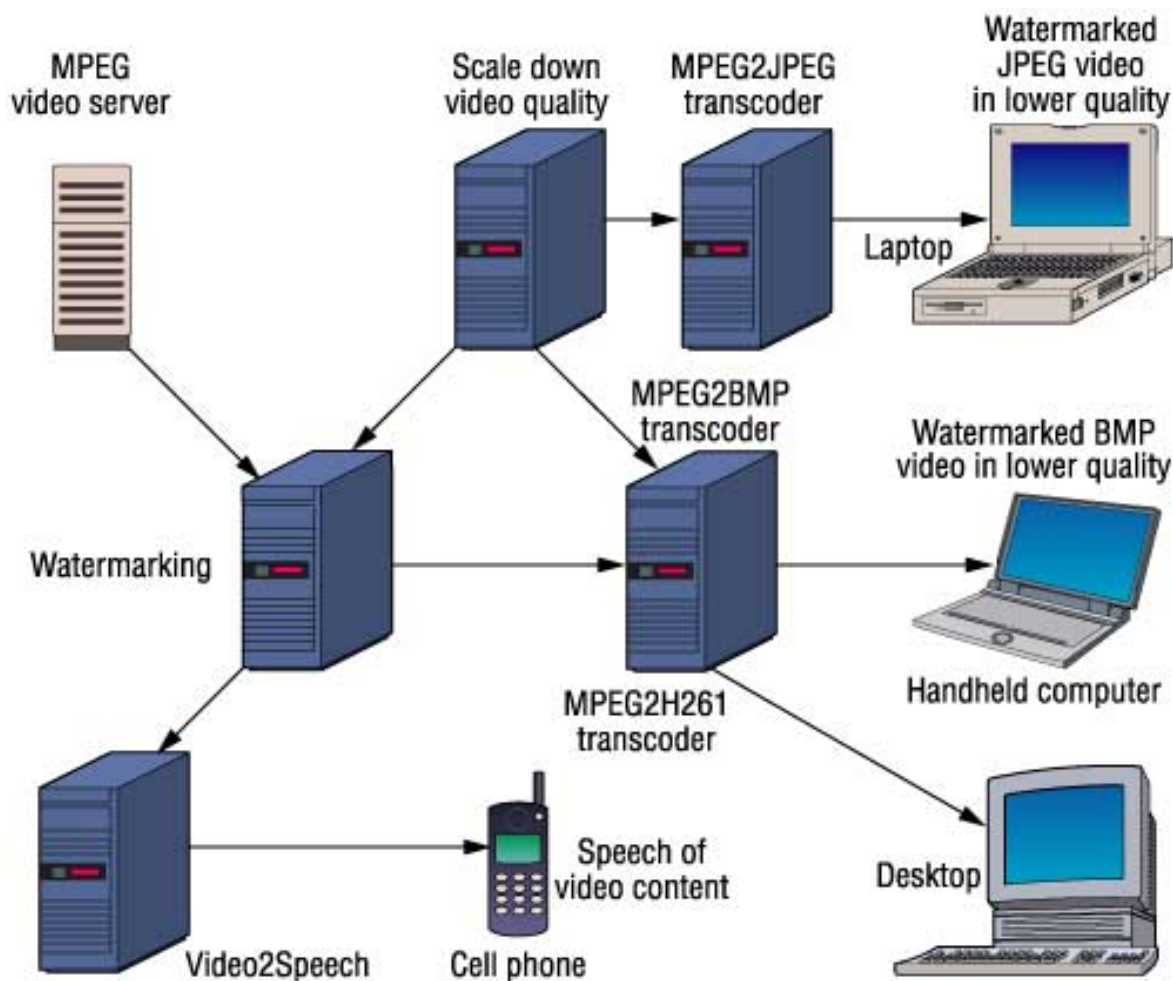


Figure 1. An example scenario of service multicasting.

Traditional QoS (data) routing finds paths based on network connectivity and QoS conditions. QoS service multicast routing, however, must also consider network nodes' service capability and dependency relations among services.

We adopt a distributed approach to building service multicast trees incrementally. We build a service multicast tree by creating one service branch at a time to cover the new member. Each service branch is QoS- and service-satisfied, and is computed hop-by-hop, using the services' geometric location information as guidance (so the constructed service multicast tree is delay efficient). We further exploit data multicasting in service multicasting, thus providing a hybrid multicasting (service multicast plus data multicast) solution.

Assumptions and background

We assume that services are widely distributed over the Internet (in an overlay proxy network or peer-to-peer network, for example), and that we can invoke

service discovery systems such as content-addressable network (CAN)³ to look up locations of service instances. Typically, a discovery system would return the IP addresses of the nodes in which services are located. We propose that service discovery systems also record and return the network nodes' geometric location, so the service lookup system can give the geometric location of each service instance. We can assign geometric coordinates to Internet hosts using mechanisms such as those proposed by T.S. Eugene Ng and Hui Zhang⁴. As we explain later, such geometric location information helps us find more delay-efficient service multicast trees.

Throughout this article, we assume a user's request is a resource-level service graph that specifies the services needed, the services' interdependencies, and the amount of physical resources (network bandwidth and machine capacity) required, because existing QoS compilers such as 2KQ+⁵ can already interpret users' QoS specifications into proper service graphs. Overlay network routing can be performed either in organized topologies^{6,7} or in unorganized topologies⁸. We follow the second approach, and build our service tree on top of a full mesh.

Geometric-location-aware service multicast tree building

Unlike traditional data routing, service routing is complicated by service constraints: the path must include required services in a stated order. In QoS data routing, the routing process usually probes the shortest path between two points. If, at certain point, a network node detects insufficient QoS, the routing probe will deviate to neighboring links or nodes. Whereas in data routing, the shortest network path (maintained by the distance vector or link state protocol, for example) serves as guidance for QoS path finding, in service routing, the network can't maintain similar shortest service paths to let a node quickly look up the best service neighbor—that is, the node that leads to a shortest service path. This is because service dependency relations are only resolved at runtime, after a user has made a service request. Lacking maintenance support, we use the geometric location information of service instances as guidance for a single node to decide, at runtime, its best service neighbor according to a user's request.

Assuming users join a multicast group at different times, we take an incremental approach to building our service multicast tree. That is, we establish a unicast service path for the very first join; for later joins, we identify a graftable on-tree node pg if one exists, and then establish a service branch

from pg to the joining node.

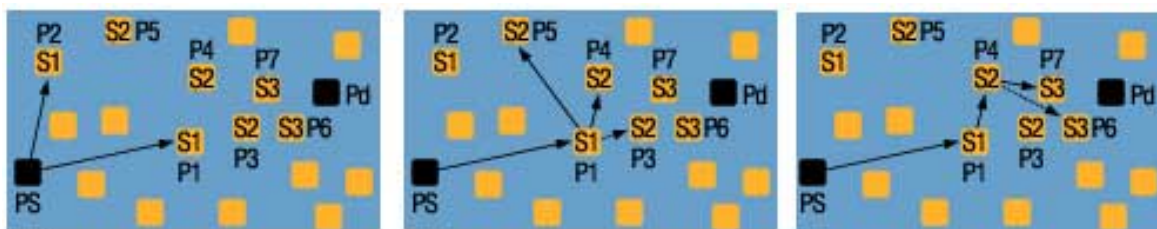
Unicast service path finding

We start our discussion with unicast service path finding, which uses the service instances' geometric location information to find delay-efficient and QoS-assured paths. Without loss of generality, we use the source as the start point and add instances of required services to the path as we route toward the destination. The source might first discover the locations of requested service instances by invoking a service discovery system such as CAN³. Knowing the locations of the service instances, we resolve the service path in a hop-by-hop manner:

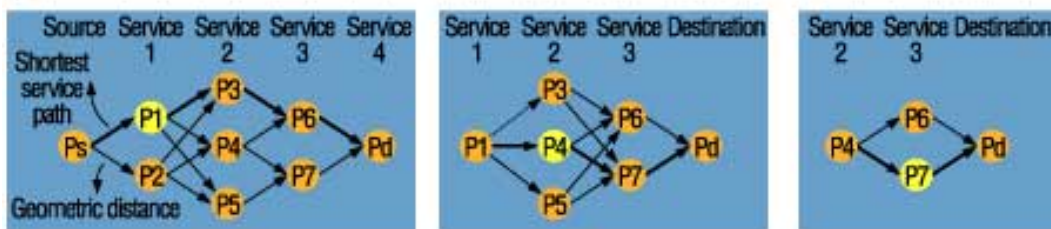
Each hop sends QoS probe messages to all instances of the next service in the request

From the instances that satisfy both bandwidth and machine capacity requirements, the current hop selects as the next hop the instance that leads to the most delay-efficient service path

To select the most delay-efficient service neighbor, the current hop can apply a shortest-path algorithm on top of a derived graph that we call a *service directed acyclic graph* (DAG). In the example in Figure 2a, we want to find a path between source ps and destination pd that includes services $s1$, $s2$, and $s3$ in sequence. Starting from the source, ps probes QoS conditions of both instances of next service in the request, $s1/p1$ and $s1/p2$. Assuming both instances have sufficient resources, ps chooses the one that potentially leads to a shortest service path, or $s1/p1$.



(a)



(b)

Figure 2. (a) Finding a service path hop-by-hop from p_s to p_d that satisfies the service graph $s_1 \rightarrow s_2 \rightarrow s_3$. The final path is in bold lines. Dashed line indicates lack of bandwidth. (b) Selecting the best service neighbors at p_s , p_1 , and p_4 , respectively. The shortest service paths are in bold lines, and the best service neighbors are highlighted.

Figure 2b shows how we compute this: by constructing the corresponding service DAG and applying a shortest-path algorithm, we can calculate a shortest-service path (bold lines) and choose the best delay-efficient service neighbor (shown in shadow).

Once the routing process reaches p_1 , p_1 probes the QoS conditions to three instances of next service in the request s_2/p_3 , s_2/p_4 , and s_2/p_5 . Although p_3 would be the best choice in terms of overall delay, it has insufficient resources. Thus p_1 chooses p_4 as next hop based on the result of the shortest-path algorithm. Such a hop-by-hop process continues until all services in the request have been resolved.

Compared to work by Xiaohui Gu and Klara Nahrstedt⁹, where hop-by-hop service instances are chosen based solely on local properties such as bandwidth and machine capacity (concave QoS metric), our approach further optimizes overall service path distances (additive QoS metric) by using geometric location information.

Service multicast tree building

Service constraints also create extra difficulties in multicast. In data multicast, routers express their join and leave interests using the Internet group management protocol, and routers basically need only be aware of their children. In service multicast, the service issues require the on-tree nodes to maintain the entire functional service tree so each on-tree node can individually search for graftable nodes for other joins. This implies that whenever a service path or branch has been successfully added to the tree, update messages must be sent to all current on-tree nodes.

In traditional data multicast, all on-tree nodes are functionally qualified as graftable nodes for every future join. In service multicast, however, not all on-tree nodes are graftable for future joins due to service constraints. Rather, an on-tree node p is only graftable for a new member whose service request is R , if p 's *up-service-path* (the service path from the root to the graftable node) is a prefix of R .

The first step toward establishing a service branch to cover the new member is

to locate a graftable node. To do this, the joining node sends a join message toward the source through an organized overlay network. The service request might hit some on-tree node p before reaching the source. As stated earlier, not all on-tree nodes are graftable for the current service request. However, since every on-tree node maintains information about the tree T , p can search T for a graftable node pg , to which the request is subsequently directed. Starting from pg and going to the joining node, we can establish a service branch for the request suffix hop-by-hop, using a method similar to that used for unicast service path finding.

Figure 3 is an example of service multicasting with a group size of two: $pd1$ (service graph $s1 \rightarrow s2 \rightarrow s3$) and $pd2$ (service graph $s1 \rightarrow s2 \rightarrow s4$). In Figure 3a, a new member $pd2$ sends a service request message toward the source using compass routing, and the request hits an on-tree node $p1$ before reaching ps . Because every on-tree node maintains T , $p1$ finds that $p4$ is the best graftable node for the current request, and forwards the request to $p4$. In Figure 3b, a service branch (for the suffix of $pd2$'s request) is established hop-by-hop from the graftable node $p4$ to $pd2$.

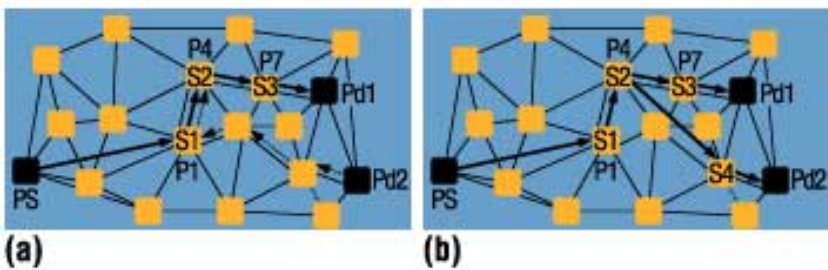


Figure 3. Service multicasting: (a) finding a graftable node and (b) establishing a service branch.

In addition to exploring service path sharing, we'd like to explore data multicasting as follows. If several (functionally) graftable nodes are available, the current branch should be grafted onto a nearby node with enough outgoing network bandwidth. In addition to boosting the service tree's overall cost efficiency, exploring data multicasting would increase our success in finding QoS service branches. Figure 4a depicts a pure service multicast tree. If, for example, $s2/p4$ doesn't have enough outgoing bandwidth to support unicasting data to three hosts ($p1$, $p2$, and $p6$), the delivery model in Figure 4b (combining service multicasting and data multicasting) becomes a better choice. The same applies to $s1/p1$. Due to space limitation, we do not elaborate on this issue.

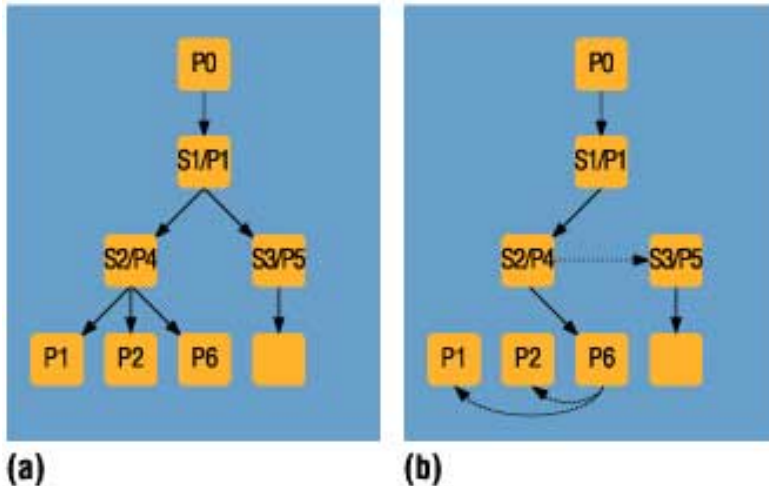


Figure 4. Multicast delivery models: (a) pure service multicast and (b) hybrid multicasting (service multicasting plus data multicasting).

We have sketched a distributed and hybrid (service multicasting plus data multicasting) approach for building service trees. We omitted certain issues, such as service tree state maintenance and failure recovery mechanisms, because of space limitations. We plan to validate the described mechanisms in NS-2 and study their performance in the future.

Acknowledgments

This work was supported by DARPA grant F30602-97-2-0121, US National Science Foundation grants CCR-9988199 and EIA 99-72884 EQ, and NASA grant NAG2-1406. Jingwen Jin is also supported by CAPES/Brazil.

References

1. F. Kon et al., "2K: A Distributed Operating System for Dynamic Heterogeneous Environments," *Proc. 9th IEEE Int'l Symp. High Performance Distributed Computing*, IEEE CS Press, 2000, pp. 201-208.
2. S.D. Gribble et al., "The Ninja Architecture for Robust Internet-Scale Systems and Services," *IEEE Computer Networks* (special issue on pervasive computing), vol. 35, no. 4, Mar. 2001.
3. S. Ratnasamy et al., "A Scalable Content-Addressable Network," *Proc. ACM Sigcomm*, ACM Press, 2001.
4. T.S. Eugene Ng and Hui Zhang, "Predicting Internet Network Distance with Coordinates-Based Approaches," *Proc. IEEE Infocom*, IEEE CS Press, 2002.
5. D. Wichadakul et al., "2KQ+: An Integrated Approach of QoS Compilation and Component-Based, Run-Time Middleware for the Unified QoS Management Framework," *Proc. IFIP/ACM Int'l Conf. Distributed Systems Platforms (Middleware)*, ACM Press, 2001.

6. J. Jin and K. Nahrstedt, "On Construction of Service Multicast Trees," *Proc. IEEE Int'l Conf. Comm. (ICC)*, IEEE Press, 2003.
7. Y. Chu, S.G. Rao, and H. Zhang, "A Case For End System Multicast," *Proc. ACM Sigmetrics*, ACM Press, 2000, pp. 1-12.
8. J. Jannotti et al., "Overcast: Reliable Multicasting with an Overlay Network," *Proc. 4th Symp. Operating Systems Design and Implementation*, Usenix, 2000.
9. X. Gu and K. Nahrstedt, "A Scalable QoS-Aware Service Aggregation Model for Peer-to-Peer Computing Grids," *Proc. High Performance Distributed Computing*, IEEE CS Press, 2002.

Jingwen Jin is a PhD candidate in the Department of Computer Science at the University of Illinois at Urbana-Champaign. Her research interests include overlay networks, service composition, quality-of-service routing, and multicast. She received a BS in physics and an MS in computer science, both from the Federal University of Pernambuco. Contact her at jjin1@cs.uiuc.edu.

Klara Nahrstedt is an associate professor of computer science at the University of Illinois at Urbana-Champaign. Her research interests are directed toward multimedia services, middleware, protocols, security, end-point architectures for multimedia, quality-of-service provision, and resource management in real-time multimedia distributed systems. She received a PhD in computer and information science from the University of Pennsylvania. She is a member of the ACM and the IEEE Computer Society. Contact her at klara@cs.uiuc.edu.