

Distributed Core Selection with QoS Support

Wanida Putthividhya Wallapak Tavanapong Minh Tran Johnny Wong
Department of Computer Science
Iowa State University, Ames, IA 50011-1040, USA

Abstract—Core-based routing with Quality of Service (QoS) support is essential to facilitate multi-sender multimedia multicast applications such as video conferencing and virtual collaboration applications. In this paper, we introduce a new distributed core selection protocol that has the following desirable properties. First, the protocol utilizes a new distributed primary core selection algorithm that selects as many primary cores per multicast group as necessary to maximize the number of group members with satisfied QoS requirements. Second, the protocol is distributed, preventing a single router from becoming a hot spot and a single point of failure during core selection. Last, the protocol employs a distributed backup core selection algorithm to provide quick recovery should some primary cores fail. Our analytical experiments show that the proposed protocol significantly satisfies more group members with noticeably less communication overhead than a recent core selection algorithm with QoS support using a single core.

Index Terms—Quality of Service, Multicast, Core-based Routing

I. INTRODUCTION

Quality of Service (QoS) and core-based routing complement each other in facilitating multi-sender multimedia multicast applications such as video conferencing and virtual collaboration. QoS support fulfills multimedia application requirements by allowing end users to specify desired service quality and assuring the specified requirements. Core-based routing provides scalable multicast for multi-sender multicast applications. Regardless of the number of senders participating in a multicast group, only one multicast tree rooted at a single core (router) is used. Data destined to the group is routed towards the single core. The core then distributes the data to all the group members via the multicast tree.

Existing core-based routing with QoS support has two major drawbacks. First, it is rather difficult for group members to specify exact values of the desired service quality as assumed in existing work [1]–[4]. Second, routing with a single core may not satisfy QoS requirements of many distributed group members. As a result, the service can be seriously affected. For instance, a video conference may not be worthwhile if important group members cannot participate with their desired QoS requirements. In addition, a mechanism to handle core failures is not considered in detail.

We introduce a new distributed core selection protocol called *DCSP* to address the above drawbacks. DCSP utilizes our *application-level service class framework* to alleviate the first drawback. The terms “application-level service classes” and “service classes” are used interchangeably in the rest of our discussion. In our service class framework, a multicast

group is associated with a set of pre-defined service classes. A member of the group is not required to specify the service quality himself, but chooses one of these service classes. The service classes indicate different bounds of the same end-to-end QoS metric that can be either delays, network bandwidth, or loss rates. The choices of the bounds and the number of the service classes depend on the types of applications. Table I shows possible end-to-end delay classes for virtual collaboration applications. Note that network-level service classes in Differentiated Services [5] are transparent to the users, which is different from our application-level service classes.

TABLE I
EXAMPLE OF END-TO-END DELAY CLASSES

Delay class	Upper-bound delay (ms)
very_short	66
short	83
medium	101
best_effort	∞

To improve the number of satisfied group members and robustness of multicast with QoS support, DCSP utilizes a new distributed primary core selection algorithm that chooses the smallest set of primary cores for a multicast group while maximizing the number of group members with guaranteed end-to-end QoS requirements under the service class framework. Each participating router runs the same primary core selection algorithm to determine the final set of primary cores for the entire group. DCSP prevents any participating router from becoming a bottleneck and a single point of failure, which impacts the ability to select the primary cores and establish the multicast delivery trees in a reasonable amount of time. DCSP employs a new distributed backup core selection algorithm that selects good backup cores to service group members if some primary core fails. Our work differs from existing primary core selection algorithms that choose only one core per group [3], [6]–[9] and those that use multiple cores per group without QoS support [10], [11]. Finally, we briefly discuss a multicast tree construction protocol to show how to construct a multicast tree rooted at each selected primary core.

The remainder of the paper is organized as follows. In Section 2, we provide background of core-based routing. In Sections 3 and 4, we present our technique and its performance, respectively. Finally, we offer concluding remarks and discuss future work in Section 5.

II. BACKGROUND

Core selection [3], [6]–[11], *multicast tree construction* [10]–[13], *membership dynamics* [1], [2], [4], and *tree/core migration* [14] are four main research areas in core-based routing. Since our work concentrates on core selection, we present only the recent related work in this area.

Distributed Core Multicast (DCM) [10] is a core-based routing protocol designed specifically to support a very large number of multicast groups with few receivers. DCM utilizes multiple cores per group to avoid determining the optimal position of the single core; however, DCM does not support QoS. DCM employs a hash function to select the set of cores from pre-defined candidate cores. Despite the effectiveness of the hash function in balancing the workloads among the candidate cores, the hash function cannot be easily modified to assure end-to-end QoS.

QoS Core Selection Algorithm (QCSA) [3] is a distributed core selection algorithm for IP multicast. The algorithm is performed by each candidate core which is the router with an attached group member. By assuming that membership information is known a priori, each candidate core attempts to construct a path that satisfies the QoS constraints from itself to each of the other candidate cores and uses the maximum number of hops of such paths as its cost. These candidate cores advertise their cost among themselves. The candidate core with the smallest cost becomes the core for the multicast group. While supporting QoS, QCSA has three drawbacks. First, it only guarantees core-to-end QoS constraints although an end-to-end QoS guarantee is more preferable for multimedia applications. Second, QCSA does not address the case that not every candidate core can construct QoS-guaranteed paths to all other candidate cores. Last, QCSA may not maximize the number of members with satisfied QoS requirements since only a single core per group is selected.

III. DISTRIBUTED CORE SELECTION WITH QoS SUPPORT

We first introduce the general concept of QoS core-based routing using a set of necessary primary cores per group under the service class framework. We, then, present DCSP—our distributed core selection protocol that includes a distributed primary core selection algorithm and a distributed backup core selection algorithm. Finally, we discuss our multicast tree construction protocol.

A. New Core-based Routing Under Service Class Framework

We revisit Table I showing four possible end-to-end delay classes for virtual collaboration applications. Each class in the table indicates the upper bound of the end-to-end delay guaranteed to the members requesting for the class. For example, the members choosing the *very_short* delay class expect to experience the end-to-end delays of at most 66 ms. On the other hand, the members requesting for the *short* delay class anticipate to experience at most 83 ms end-to-end delay. Except for the best-effort class, the bounds specified in all the service classes are acceptable to users of virtual collaboration applications [15]. Group members who do not need a high degree of interactivity may request for the class with a longer guaranteed end-to-end delay given a cheaper service fee as an incentive. Group members are either guaranteed the bounds of the requested service class or denied the service if re-negotiation for another service class is not allowed.

We define a *member router* and a *sender router* as a multicast-capable router designated by a group member and a sender, respectively. A designated member router subscribes to all the service classes requested by its designating group

members and is responsible for delivering multicast data to these members. Senders sending data to a group are not required to join the group. One router acts as both a member router and a sender router if designated by both a sender and a group member. We say that a *member router j covers a member router i subscribing to a service class c* if the bound indicated by the class can be guaranteed when the member router *i* receives multicast data from every sender router via the member router *j*. The covering set of the member router *j* for class *c* is the set of all the member routers that subscribe to the service class *c* and are covered by the member router *j*. We focus on guaranteeing end-to-end QoS requirements along the paths between sender routers and member routers. We assume that (i) the routers have the resource reservation capability and (ii) the QoS requirements between a group member and its designated router are always assured since they are typically on the same LAN.

Our approach employs as many primary cores per group as necessary to maximize the number of group members with satisfied QoS requirements. Each primary core is assigned a set of member routers. The sets of member routers assigned to all the primary cores are disjoint. The primary core constructs a multicast tree rooted at itself and spanned all of its assigned member routers. A sender router unicasts data towards each primary core of the group. The data is, then, distributed from the primary cores to the assigned member routers via the corresponding multicast trees. Despite our attempt to use as many primary cores as necessary, a group member requesting for a service class *c* may not have the requirement guaranteed even though its designated member router acts as one of the primary cores. This is because the requirement is too stringent for the current network condition. We call such a group member and its designated member router a *class c insatiable group member* and a *class c insatiable member router*, respectively.

Our approach improves the robustness of multicast delivery in two ways. First, primary cores are used as backups of each other should some primary core fail. Second, core selection is performed in a distributed fashion, avoiding the problems of a hot spot and a single point of failure. Nonetheless, routing with multiple cores per group comes with a cost. The sender-to-core traffic increases proportionally to the number of cores since each sender requires one unicast stream to each primary core. Thus, to minimize the cost, we minimize the number of primary cores while maximizing the number of group members with guaranteed QoS requirements.

B. Distributed Core Selection Protocol

DCSP determines both primary core(s) for an entire multicast group and a backup core for each member router. Initially, there is neither a core nor a multicast tree for the group.

Step 1: Registration: The purpose of this step is to gather necessary information from both sender and member routers and distribute the received information among them. Each member router registers to a pre-determined bootstrap router by submitting the number of designating group members in each service class. A unique *ID* is randomly assigned to each member router. Each sender router also registers to the bootstrap router. The bootstrap router announces to all the

member routers the IP address of every sender router and the membership information received from every member router along with the ID and the IP address of every member router. The bootstrap router also sends the information received from all the member routers to every sender router¹.

Step 2: Distributed Resource Reservations: All the member routers are the candidate cores. Each sender router independently finds end-to-end QoS guaranteed paths from itself to all the member routers via each candidate core for every service class. This is performed by modifying a QoS unicast routing protocol that reserves resources along the paths [16], [17].

Each candidate core determines its covering set for each service class based on *success* and *failure* control messages of the QoS unicast routing protocol passing through itself. Each candidate core also keeps the path information of the end-to-end QoS assured paths (e.g., list of routers and guaranteed delays along the path) for multicast tree construction.

Step 3: Distributed Primary Core Selection: Each candidate core independently determines the final set of primary cores for the group in a distributed fashion as follows.

Step 3a: Each candidate core executes the algorithm in Fig. 1 to exchange the local covering sets among the candidate cores in two rounds. The candidates are organized into a logical chain. The candidate with the ID i is prior to the candidate with the ID j in the chain if $i < j$. The setup of the logical chain incurs no cost since every candidate knows the IDs and IP addresses of all the candidates by the end of the registration step. The candidate is assumed failed if it does not respond to a message from another candidate within a timeout period after three tries. In this case, the sending candidate announces the detected failure by sending unicast messages to the rest of the candidate cores, and the other immediate neighbor of the failed candidate core becomes the new immediate neighbor of the sending candidate. The failed candidate core is excluded from further computations and treated as a new member router should it recover.

Step 3b: Upon receiving the covering sets of all the other candidates, each candidate core selects a set of primary cores for each service class using a *cover heuristic*, starting from the class with the most stringent requirement. Let R denote the set of member routers (candidate cores) of the multicast group. Let R_c be the set of all member routers designated by the group members requesting for the class c , and R'_c denotes the set of member routers in R_c excluding all the class c insatiable member routers. For each service class c , the greedy algorithm selects one of the remaining candidate cores that *covers* the most number of member routers in R'_c that have not been assigned to any previously selected core. If there is a tie, the candidate covering more group members wins. If there is still a tie, the candidate with the lowest ID wins. An additional core is chosen repeatedly until every member router in R'_c has been assigned to a primary core. By choosing the candidate core that covers the highest number of member routers at each step, the number of chosen primary cores for each service class is minimized. For each service class, every

candidate core selects the same set of primary cores since the deterministic tie breakers are used in the algorithm. A member router whose designating group members have been satisfied for the tighter class is removed from further considerations since the satisfaction of the tighter requirement implies the satisfaction of the less stringent requirement for the same member router.

The greedy algorithm is used in the protocol instead of an optimal algorithm since the problem of selecting the smallest set of primary cores that maximize the number of group members with guaranteed end-to-end requirements is NP-hard. This problem is polynomial time reducible to an NP-hard set-covering problem. The detailed problem formulation can be found in our technical report [18].

Step 3c: The final set of primary cores for the entire group is determined by each candidate core using a deterministic core merging algorithm that consists of the two following steps.

Core Union: Union the sets of primary cores of every service class.

Core Set Reduction: Remove the primary core whose entire covering set can be covered by another primary core and ensure that each member router is assigned to only one of the selected primary cores. The IDs of the primary cores can be used as a tie breaker.

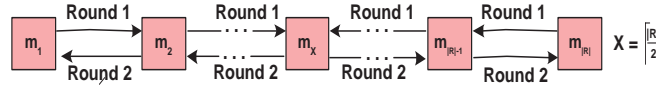
Step 3d: One of the candidate cores, e.g., $m_{|R|}$ informs the bootstrap router and the sender routers that do not act as member routers about the selected primary cores. This is because all the member routers already know the selected cores by the end of Step 3c.

Step 3e: Recall that each end-to-end QoS-assured path constructed in Step 2 has two parts: the part from a sender router to a candidate core and the path from a candidate core to its covered member routers. Each sender router chooses the best path (e.g. the shortest delay route if the end-to-end delay is the QoS metric of interest) from itself to each of the selected primary cores and releases the resources along the non-selected paths. Each sender router also releases the reserved resources along the end-to-end paths corresponding to every candidate core not selected as a primary core. The resources along any core-to-end path from a primary core to a member router covered but not assigned to the core are kept for future use in the case that the primary core is chosen as the backup core for such member router. These resources may be released if failures of primary cores rarely happen and the backup cores are not necessarily selected.

Step 4: Distributed Backup Core Selection: Based on the final set of the primary cores and the covering information determined in the previous step, each member router selects a backup core from the set of the primary cores such that the number of its designating group members whose QoS requirements assured is maximized, if its primary core fails.

The member router finds its backup core which is another primary core that can assure the bound of the service class requested by its designating group members, starting from the most stringent class. If such a primary core is found for one of the service classes, it becomes the backup core for the member router and the search is stopped. Otherwise, the backup core is randomly selected.

¹An implementation alternative of this Registration step is to use routing control packets to carry the information of our protocol.



```

/* R: Set of member routers; mi: Member router with the ID i; myID is the ID of the router executing the algorithm*/
/*Round 1: The incremental covering information is concurrently propagated from both ends to the middle of the chain
of the candidates. Only m_{floor(R/2)} has the complete covering information at the end of this round.*/
If myID == 1, send the local covering sets for every class to m2
Else if myID == |R|, send the local covering sets for every class to m_{|R|-1}
  Else if 2 ≤ myID ≤ floor(R/2),
    Wait for the covering information of m1 ... m_{myID-1} from m_{myID-1} or until a timeout
    Reply to m_{ID-1} and forward both the received and its own covering sets for every class to m_{myID+1};
  Else if floor(R/2) ≤ myID ≤ |R| - 1,
    Wait for the covering information of m_{|R|} ... m_{myID+1} from m_{ID+1} or until a timeout
    Reply to m_{myID+1} and forward both the received and its own covering sets for every class to m_{myID-1};

/*Round 2: The partial covering information from both sides of the chain is propagated to the other side from the middle
of the chain. Every candidate core has the complete covering information at the end of this round.*/
If 1 ≤ myID ≤ floor(R/2) - 1,
  Wait for the covering information of m_{|R|} ... m_{myID+1} from m_{myID+1} or until a timeout
  Reply to m_{myID+1} and If myID > 1, forward both the received and its own covering sets for every class to m_{myID-1};
Else if floor(R/2) + 1 ≤ myID ≤ |R|,
  Wait for the covering information of m1 ... m_{myID-1} from m_{myID-1} or until a timeout
  Reply to m_{myID-1} and If myID < |R|, forward both the received and its own covering sets for every class to m_{myID+1};

```

Fig. 1. Algorithm to exchange covering sets among candidate cores $m_1 \dots m_{|R|}$.

C. Multicast Tree Construction Protocol

We use end-to-end delays as an example QoS metric to demonstrate our multicast tree construction protocol. The protocol is also applicable when using a different end-to-end QoS metric. Each selected primary core independently constructs a multicast tree rooted at itself and spans every member router assigned to it as follows.

- Each primary core determines the shortest guaranteed delay path from itself to each of its assigned member routers using the end-to-end path information from Step 2 of DCSP.
- Each primary core independently sends a *tree construction message* onto each core-to-end path determined in the previous step. Each core fills the source routing option field in the IP header of the tree construction message with the list of the IP addresses of the routers along the corresponding path. The message must be forwarded to these routers in the order specified in the list. The payload of the tree construction message contains the multicast group address, the primary core of the group originating the message, the destination member router, and the cumulative guaranteed delay associated to the portion of the core-to-end path the message has traversed so far. The router that receives the tree construction message creates/updates the routing entry corresponding to the group and the primary core of the group specified in the message.

A routing entry maintained at a participating router has the following fields: *gid*, *cid*, *in*, *delay*, *out*. The *gid* and *cid* fields specify the multicast group and the primary core of the group, respectively. The *in* field indicates the interface that receives multicast data sent from the primary core *cid* of the group *gid*. The *delay* field indicates the cumulative guaranteed delay from the primary core *cid* to the router via the current multicast tree. The *out* field contains a list of interfaces to forward the multicast data. To ensure that routing within a multicast group is loop-free, a router being a part of different multicast trees for the same group maintains more than one routing entries for the group. Each entry corresponds to the multicast tree rooted

at the primary core specified in the *cid* field of the entry. The router has as many routing entries for the group as the number of trees the router involves.

To avoid duplicate multicast data reaching the group members, any router receiving more than one tree construction messages of the same group and the same primary core of the group creates/updates the corresponding routing entry according to the message indicating the shortest cumulative guaranteed delay and prunes the tree branches with longer delays from the primary core to the router. The cumulative assured delay is used to select the shortest guaranteed tree branch to avoid duplicate multicast data to reach the members.

When receiving multicast data packets for the group, each primary core adds its own identity to the packets. The on-tree router checks for the routing entry corresponding to the group and the primary core indicated in each multicast data packet and forwards the packet onto the corresponding outgoing network interfaces. The routing is loop-free and does not cause any duplicate multicast data to arrive at the group members.

D. Mechanism for Handling Core Failures

If the primary core of a member router fails, the member router contacts its backup core to initiate the construction of a new tree branch from the backup core to itself. The new branch is constructed as if the member router was another member router assigned to the backup core. The member router receives multicast data from its backup core while its new primary core is being determined by treating the member router as a new member joining the group. This can be done by modifying existing protocols to handle new members developed for a single core such as QMRP [4]. During the time the primary core of the the group members is down until the time the new primary core for these members is chosen, group members of some service classes may not have their QoS requirements assured, but will continue to receive multicast data from the backup core. This approach is better than letting the members receive no data at all.

IV. PERFORMANCE STUDY

We evaluate our primary core selection algorithm with respect to the optimal solution and the related work QCSA [3]. We quantify the control overhead of our distributed core selection protocol compared to the centralized core selection protocols. The following performance metrics are used.

- **Average number of selected primary cores:** This metric is the average over the number of selected primary cores from several experiments. A small average is desirable as it shows the low sender-to-core traffic.
- **Average percentage of rejected group members:** This metric is the average of the percentages of the group members whose QoS requirement cannot be satisfied by the protocol, including the insatiable members. These members are denied the service. The lower the percentage, the better the core selection protocol.

We also investigate the trade-off between the cost in terms of the increased sender-to-core traffic and benefits of rejecting less group members using our protocol under various scenarios. Due to limited space, interested readers are referred to the results in [18].

A. Experimental Model

The QoS metric used in all the experiments were end-to-end delays. We constructed fifty networks based on the transit-stub model using GT-ITM [19]. Each network has one transit domain with four transit nodes, representing a Wide Area Network with four backbone routers. The transit node has five stub domains, each having five stub nodes on average. The stub domains and stub nodes represent regional multicast capable networks. We map the link distance assigned by GT-ITM to a link delay such that the longest path in the network has the total delay of approximately 70 ms, a coast-to-coast delay observed in United States. Four service classes in Table I were used in the experiments.

We conducted analytical experiments under various group sizes ranging from 5 to 30 members. These are the group sizes for many multi-sender multimedia multicast applications. For each experiment, the member routers were randomly selected from the routers in the stub domains as the backbone routers are not directly connected to any user. All the member routers joined a single multicast group. In most experiments, each member router was also a sender router unless stated otherwise. We assigned only one service class to each member router, assuming that the member router was designated by only one group member. This restriction is required to formulate the QoS primary core selection problem as an integer programming problem that is solved using the optimal algorithm provided by a linear programming solver `lp_solve` [20]. We generated the number of member routers subscribing to each class based on an assumption that many more group members subscribe to the classes with a reasonable service fee (*short_delay* and *medium_delay*) compared to the most expensive class (*very_short_delay*) or the least expensive class (*best_effort*) that does not provide any guarantee. Within each service class, the member routers are randomly selected repeatedly until the class has the desired number of member routers generated earlier. For QCSA, these member routers ask for the bounds indicated by the classes.

B. Effectiveness of the Algorithms

We present a performance comparison of our distributed core selection algorithm (*DCSP*), the optimal algorithm (*Optimal*), and QCSA. Due to some differences between QCSA and DCSP, a number of modifications were applied to both schemes. Since the original QCSA only guarantees core-to-end delays, we modified it to guarantee end-to-end delays. This modified technique is named *QCSA-e2e*. We also improved *QCSA-e2e* further to handle the case that no candidate core can construct QoS-assured paths to all other candidates. When this happens, the candidate with the biggest covering set is chosen as the primary core. We call this *QCSA-e2e-cover* since it is influenced by our cover heuristic. The plots corresponding to these techniques are labeled accordingly. Each data point in the plot is an average of the results from ten experiments.

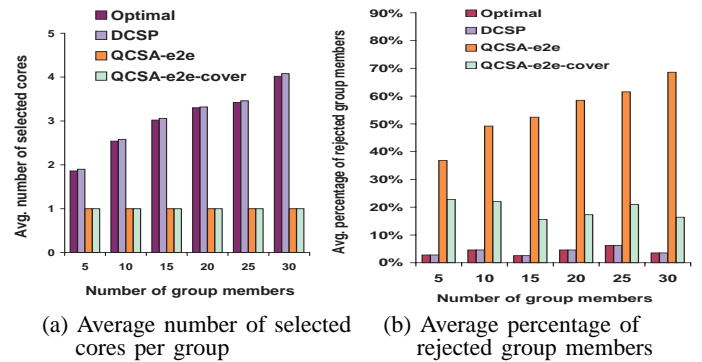


Fig. 2. Effectiveness of core selection algorithms.

DCSP selects slightly more primary cores on average compared to the optimal solution as shown in Fig. 2(a). The good performance of DCSP is attributed by the fact that (i) the cover heuristic minimizes the number of primary cores by selecting the candidate with the biggest covering set at each step and (ii) the merging algorithm further discards the unnecessary primary cores. Fig. 2(a) also demonstrates that both techniques require more primary cores when the group size increases. The rationale is that in our experiments, the members and senders in the bigger group tend to scatter more than those in the smaller group. In contrast, the other techniques in Fig. 2(a) always select one primary core.

Fig. 2(b) shows that QCSA-e2e rejects about 50.45% more group members than DCSP. DCSP and Optimal reject the same number of group members since both techniques select as many primary cores as necessary to satisfy all the QoS requirements except for those of the insatiable members. When compared to DCSP, QCSA-e2e-cover rejects about 15.13% more group members. The rejected group members are insatiable members and the group members whose requirements cannot be satisfied by the single primary core. Fig. 2(b) also shows that QCSA-e2e rejects about 39.59% more group members compared to QCSA-e2e-cover. This is because QCSA-e2e ignores the case when not every candidate core can construct QoS-assured paths to all other candidates. The primary core selection fails and all the group members are rejected in that case. The increase in the group size in Fig. 2(b) does not imply the increase in the number of rejected group members. In particular, the average percentage

of rejected group members appears random, which reflects the randomness in the assignment of the service classes to the group members as explained previously.

C. Protocol Overhead

The overhead comparison between QCSA, our distributed core selection protocol (DCSP) and its centralized counterpart (CCSP) is shown in Fig. 3. In CCSP, each member router tells the bootstrap router about its covering sets for all the service classes requested by its designating members. The bootstrap router performs the same greedy core selection algorithm to select the primary cores and informs the selected cores and all the sender routers about the core set. The overhead of each protocol is measured by the total number of control messages caused by each protocol. This includes control messages sent during the registration step, during the core selection step, and after the core selection step in each protocol.

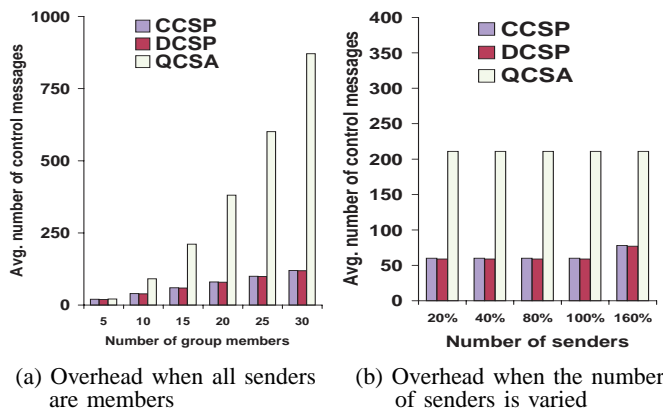


Fig. 3. Effect of group sizes and the number of senders.

Fig. 3(a) shows the overhead comparison among the three protocols when all the group members are also senders. The overhead caused by every protocol increases as the group size increases. More members mean more messages during the registration step and during the exchange of the covering information among the routers. DCSP and the CCSP incur significantly less overhead than QCSA. This is because the local covering information is exchanged via a logical chain in DCSP or sent to the only bootstrap router in CCSP. In QCSA, each member router unicasts its cost to each of the member routers to independently select the least cost member router as the core.

Fig. 3(b) depicts the overhead comparison when the number of senders is varied from 20% to 160% of the total number of group members. The group size was fixed at fifteen which is the middle-sized group. The increase in the percentage of senders has no impact on QCSA since senders and members behave similarly in QCSA. However, the overhead caused by DCSP and CCSP increases when the set of sender routers is larger than the set of member routers. The rationale behind this is that more sender routers imply more number of control messages when registering with the bootstrap router and informing the sender routers about the selected cores of the group. For any number of senders, DCSP incurs slightly less overhead than CCSP since only the sender routers which are not the member routers are informed of the selected cores. The sender routers that are also the member routers already know

the selected cores. In CCSP, all the sender routers have to be informed. Considering both robustness and overhead, DCSP is more attractive for multi-sender applications than CCSP and QCSA.

V. CONCLUDING REMARKS

This paper presents a new distributed core selection protocol with QoS support for core-based routing using as many primary cores per group as necessary. The proposed protocol has been shown to accept about 50% more group members compared to a recent QoS core selection protocol with significantly less communication overhead. As important, the proposed distributed protocol enhances the robustness of core-based routing in two ways. First, it prevents a single router from being a bottleneck or a single point of failure. Second, the protocol selects a good backup core for each member router should its primary core fail.

REFERENCES

- [1] K. Carlberg and J. Crowcroft, "Building shared trees using a one-to-many joining mechanism," *Computer Communication Review*, pp. 5–11, January 1997.
- [2] M. Faloutsos, A. Banerjee, and R. Pankaj, "Qosmic: Quality of service sensitive multicast internet protocol," in *Proc. of ACM SIGCOMM*, September 1998, pp. 144–153.
- [3] S.M. Chung and C.H. Youn, "Core selection algorithm for multicast routing under multiple qos constraints," *Electronic Letters*, vol. 36, no. 4, pp. 378–379, February 2000.
- [4] S. Chen, K. Nahrstedt, and Y. Shavitt, "A qos-aware multicast routing protocol," *IEEE journal on Selected Areas in Communications*, vol. 18, no. 12, pp. 2490–2498, December 2000.
- [5] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss, "An architecture for differentiated service," in *RFC 2475, IETF*, available at <http://www.faqs.org/rfcs/rfc2475.html>, December 1998.
- [6] K.L. Calvert, E.W. Zegura, and M.J. Donahoo, "Core selection methods for multicast routing," in *Proc. of Int'l Conf. on Computer Communications and Networks*, September 1995, pp. 638–642.
- [7] D.G. Thaler and C.V. Ravishanker, "Distributed center-location algorithms," *IEEE journal on Selected Areas in Communications*, vol. 15, no. 3, pp. 273–276, April 1997.
- [8] Y. Huang, E. Fleury, and P.K. McKinley, "LCM: A multicast core management protocol for link-state routing networks," in *Proc. of IEEE Int'l Conf. on Communications*, June 1998, vol. 2, pp. 1197–1201.
- [9] E. Fleury and Y. Huang, "On the performance and feasibility of multicast core selection heuristics," in *Proc. of 7th Int'l Conf. on Computer Communications and Networks*, Lafayette, LA, October 1998, pp. 296–303.
- [10] L. Blazevic and J.Y. Le Boudec, "Distributed core multicast(dcm): a multicast routing protocol for many groups with few receivers," in *ACM SIGCOMM Computer Communication Review*, October 1999, vol. 29, pp. 6–21.
- [11] S. Deering, B. Fenner, D. Estrin, A. Helmy, D. Farinacci, L. Wei, M. Handley, V. Jacobson, and D. Thaler, "Hierarchical pim-sm architecture for inter-domain multicast routing," in *Internet Draft*, available at <http://netweb.usc.edu/ahelmy/interdomain-multicast/interdom-1-19.ps>, December 1995.
- [12] T. Ballardie, P. Francis, and J. Crowcroft, "Core based tree (CBT): An architecture for scalable inter-domain multicast routing," in *Proc. of ACM SIGCOMM*, Ithaca, NY, USA., September 1993, pp. 85–95.
- [13] S. Deering, D. Estrin, D. Farinacci, V. Jacobson, C.G. Liu, and L. Wei, "An architecture for wide-area multicast routing," in *Proc. of ACM SIGCOMM*, August 1994, pp. 125–135.
- [14] M.J. Donahoo and E.W. Zegura, "Core migration for dynamic multicast routing," *Int'l Conf. on Computer Communications and Networks*, 1996.
- [15] VRAC, "Virtual reality applications center," <http://www.vrac.iastate.edu>.
- [16] G. Manimaran, H. S. Rahul, and C. Siva Ram Murthy, "A new distributed route selection approach for channel establishment in real-time networks," *IEEE/ACM Transactions on Networking*, vol. 7, no. 5, pp. 698–709, October 1997.
- [17] R. Sriram, G. Manimaran, and C. Siva Ram Murthy, "Preferred link based delay-constrained least-cost routing in wide area networks," *Computer Communications*, vol. 21, pp. 1655–1669, May 1998.
- [18] W. Putthividhya, W. Tavanapong, J. Wong, and M. Tran, "A novel core selection with end-to-end qos support for multi-sender multimedia multicast applications," *Technical Report: Department of Computer Science, Iowa State University*, no. TR-03-07, 2003.
- [19] K. Calvert and E.W. Zegura, "Georgia tech internetwork topology models." Available from <http://www.cc.gatech.edu/projects/gtintm/>, 1996.
- [20] M. Berkelaar, "Lp_solve: linear programming code," Available from <http://www.cs.sunysb.edu/algo-rith/Implement/lpsolve/Implement.shtml>, 1996.