INTERNATIONAL JOURNAL OF COMMUNICATION SYSTEMS
Int. J. Commun. Syst. 2009; **22**:67–86
Published online 1 September 2008 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/dac.959

Locating repair servers in hierarchical reliable multicast networks to reduce the makespan

Sang-Seon Byun[1,*,†] and Chuck Yoo[2]

[1]*Signal Processing Group, Department of Electronics and Telecommunications, Norwegian University of Science and Technology, N7491 Trondheim, Norway*
[2]*Department of Computer Science and Engineering, Korea University, An-am dong 5th street, Seong-buk, Seoul, Korea*

SUMMARY

In a hierarchical reliable multicast (HRM) environment, *makespan* is the time that is required to fully and successfully transmit a packet from the sender to all receivers. Low makespan is vital for achieving high throughput with a TCP-like window-based sending scheme. In HRM methods, the number of repair servers and their locations influence the makespan. In this paper we propose a new method to decide the locations of repair servers that can reduce the makespan in HRM networks. Our method has a formulation based on mixed integer programming to analyze the makespan minimization problem. A notable aspect of the formulation is that heterogeneous links and packet losses are taken into account in the formulation. Three different heuristics are presented to find the locations of repair servers in reasonable time in the formulation. Through simulations, three heuristics are carefully analyzed and compared on networks with different sizes. The results show that the our best heuristic is close to the lower bound by a factor of 2.3 in terms of makespan and by a factor of 5.5 in terms of the number of repair servers. Copyright © 2008 John Wiley & Sons, Ltd.

Received 18 September 2006; Revised 27 February 2008; Accepted 2 July 2008

KEY WORDS: hierarchical reliable multicast; makespan; greedy heuristic; window-based sending scheme

1. INTRODUCTION

Recently many reliable multicast protocols deploy hierarchical structure to achieve scalability. Hierarchical reliable multicast (HRM) constitutes and maintains a control tree that is used to transmit recovery and feedback (ACK/NACK) apart from the multicast delivery tree. On the control tree, there must be repair servers that play a role of local recovery and feedback consolidation.

*Correspondence to: Sang-Seon Byun, Signal Processing Group, Department of Electronics and Telecommunications, Norwegian University of Science and Technology, N7491 Trondheim, Norway.
†E-mail: sang-seon.byun@iet.ntnu.no

Repair server can be set up as an exclusive server [1–5] or designated among adequate receivers [6–8].

The performance of an HRM can be evaluated by the bandwidth overhead due to local recovery and feedback consolidation and the *makespan*. Makespan in HRM is defined as the time that is required to fully and successfully transmit a packet from a sender to all receivers. The bandwidth overhead and makespan are affected by the locations and the number of repair servers. For example, if a repair server is adjacent to a receiver that is susceptible to packet losses and the repair server experiences fewer packet losses than the receiver, the recovery and feedback traffics are limited to the server's domain. Additionally, if the delivery delays between the sender and each receiver are heterogeneous and additional delays are required to recover some packet losses, the arrival time of the packet at each receiver may be different. Therefore, appropriate locating repair servers can reduce packet recovery time, and thus better makespan is obtained.

Low makespan is essential for achieving high throughput in window-based multicast sending schemes. In window-based schemes that provide full reliability, advancing the window is highly dependent on the slowest receiver's successful reception [9–11]. The slowest receiver is the member that takes the longest time to receive a packet from the sender, and this time becomes equivalent to the makespan.

In this paper we propose a new method that can extract locations of repair servers to reduce makespan in heterogeneous HRM networks. First we formulize this problem as an MIP (mixed integer programming). Since it is impossible to obtain an optimal solution to our MIP formulation,[‡] we propose several heuristics that can provide good solutions to this problem in reasonable time. If the network size is n and k proxies are to be located, our MIP has the time complexity of $O((n-k)^n)$ in the worst case. Thus, the computation time increases exponentially as the network size grows. The simulation results show that the greedy heuristic of locating a repair server first on the node that lies on the longest path and has the incident edge showing the largest expected delivery delay is the best among our heuristics. In addition, the best heuristic is close to the lower bound obtained using LP relaxation by a factor of 2.3 in terms of makespan.

Our proposal can be adapted to the real application as follows:

For instance, a service provider may perform file distribution or multi-media streaming using IP multicast or overlay multicast. The service provider collects the statistics of each link—per-link delay and per-link loss rate—and defines appropriate makespan in accordance with the application type and user's requirement. Then using our proposal, they can decide the feasible locations of the repair proxies in order to achieve the makespan in reasonable time. In addition, due to the fluctuation of link statistics and the dynamic membership, the optimal locations of repair proxies must be recomputed as fast as possible.

Moreover, we show that our proposal is beneficial to the achievement of better throughput in a TCP-like multicast congestion control schemes using ns-2 simulation.

This paper is organized as follows. In Section 2 we present some related work. In Section 3 we present the HRM model used in this paper and describe an expected delivery delay model that reflects the locations of the repair servers and heterogeneity of network environment. In Section 4 we provide the MIP that formulates our model. In Section 5 we propose three different heuristics

[‡] With our system, Pentium IV 3.0 GHz and 1.5 GB memory, it took about 4 h GNU glpk [12] to locate only 5 repair servers on a network of 50 nodes. In case of 100 nodes, glpk could not solve integer solution until 24 h have elapsed.

DOI: 10.1002/dac

that can provide solutions in reasonable time, and the MIP for computing the lower bound in terms of the number of repair server. Section 6 provides simulation results on the heterogeneous networks with various sizes. Finally, concluding remarks are given in Section 7.

2. RELATED WORK

2.1. Repair server location scheme in HRM

Nonnemacher *et al.* [13] indicate makespan as an important metric in HRM. However, to the best of our knowledge, there is no previous work to improve the makespan in HRM networks. Related with placement of repair servers, many researches have focused on minimizing bandwidth overhead caused by recovery and feedback traffic [14–18] and load balancing among repair servers [15, 19]. Some researches have not considered optimal placement of repair servers [18, 20]. Also, they assume that every link has a uniform delivery delay and loss rate. Li *et al.* [21] suggest a method to localize proxies to minimizing web distribution time using dynamic programming formulation. However, in order to reduce combinatorial complexity, the available location of a proxy is limited to some area of the web distribution tree.

2.2. Window-based sending schemes

Pragmatic general multicast congestion control (PGMCC) [9] is a TCP-like window-based congestion control scheme that operates on an HRM protocol, pragmatic general multicast (PGM). PGM deploys routers that perform local recovery and feedback consolidation. PGMCC elects a group representative (acker) and mimics the TCP congestion window mechanism between the sender and the acker. The acker is chosen as the receiver with the worst throughput among group members. The acker election process is performed by measuring packet loss rate and round trip time (RTT). RTT is measured through computing the difference between the most recent sequence number sent and the highest known sequence number to a receiver that is noted in a feedback packet. Therefore, if the makespan is reduced through adequate placement of PGM-enabled routers, the RTT between the sender and the acker can also be reduced, and thus the congestion window can advance faster.

Multicast TCP (MTCP) [10] is another congestion control scheme using hierarchical structure for large-scale reliable multicast. In MTCP, sender agents (SAs) are deployed to perform local recovery and feedback consolidation for their children, and between SAs, and between SA and leaf nodes, TCP-like congestion control is performed. The congestion window in each SA is incremented when each SA receives ACKs for a packet from all of its children. Therefore, the reduced makespan can be beneficial to the fast advance of the congestion window.

In reliable multicast transport protocol-II (RMTP-II) [11], when receivers and designated receivers (DRs) receive the packet successfully, they respond by generating ACKs and propagating them up the control tree, and back to the sender. DR also performs ACK consolidations and local recovery. When the sender gets an ACK for a data packet it has sent, it can advance its transmission window and free that packet from memory. Therefore, if the makespan becomes shorter, the transmission window can advance faster, and this results in higher throughput and lower memory usage.

In order to alleviate throughput degradation due to the worst receiver, multi-rate sending schemes can be deployed. Chaintreau [22] analytically proves that the window-based sending scheme might

DOI: 10.1002/dac

be harmful with reliable multicast transmission. In addition, they show that this result extends to unreliable application that may find it difficult to manage a 50% drop in the average throughput when the number of receiver increases, and multi-rate sending schemes are best suited to multicast sessions with a large number of receivers. However, the flexibility of the multi-rate scheme is paid in terms of coding costs, some bandwidth inefficiency, and possibly a coarser match of the sender and receiver data rate [9].

3. HRM AND EXPECTED DELIVERY DELAY MODEL

In this section, we describe the HRM and the expected delivery delay model. As described in Section 1, delivery delay is the time required to successfully transmit a packet from the sender to a receiver. In order to extract the locations of repair servers to reduce makespan, HRM and expected delivery model must be established reflecting heterogeneity and locations of repair servers.

3.1. HRM model

This paper assumes the HRM model of the following characteristics (Figure 1).

(1) If a repair server is placed in a node, then the node itself becomes a repair server.
(2) Consider a single-sender multicast tree where the root is the unique sender, all leaves are receivers and all intermediate nodes can be repair servers [1–3].
(3) The topology of control tree is identical to that of its underlying multicast tree (IP multicast tree), and loss probabilities and delays at the links of the control tree are given. It is indeed possible to acquire knowledge about the underlying multicast tree even under dynamic membership situation. Levine *et al.* [23–26] describe a way of establishing a control tree that is identical to its underlying multicast tree and collecting link loss statistics. Especially, Caceres *et al.* [27] and Presti *et al.* [28] propose methods to infer per-link loss rates and delays. Even if the topology and loss rates are not known in real world, our method may still be useful for comparison and assessment purpose.
(4) The control tree is partitioned into subtrees that form a hierarchy rooted at the sender. All nodes in a subtree are combined into a subgroup, and each subgroup has a repair server located at its root. The sender itself is a repair server by default. These features are deployed in [1–3, 7–9, 29].
(5) Repair server multicasts the original data to its own subgroup. Each receiver sends a feedback (NACK or intermediate ACK) to its own repair server when a packet loss is detected, and the repair server retransmits the lost packet to the whole subgroup. We assume that the feedbacks are delivered via out-of-band channel [30]. This assumption simplifies our expected delivery delay model; however, some commercial routers (e.g. Cisco NAC (network admission control) application server) and packet classification service in IPv6 already support the transmission via out-of-band channel.
(6) Feedbacks and transmissions/retransmissions are limited only between a repair server and the receivers of its subgroup and they do not reach the receivers/repair servers of any other subgroup. For this purpose, a new multicast address per subgroup is assigned [2], or TTL (Time to live) may be used to scope subgroup [6]. Additionally subcasting and TTL scoping can be used simultaneously [7].

DOI: 10.1002/dac

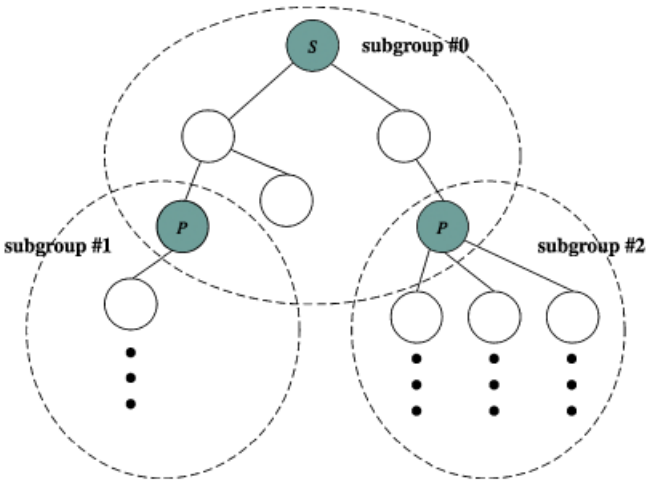

Figure 1. HRM model.

3.2. Expected delivery delay model

We explore an expected delivery delay model that reflects heterogeneity and locations of repair servers. We assume that per-link delay and loss rate are measured in an end-to-end manner. There are some previous studies [27, 28] that describe the methods to infer per-link loss rate and delay in an end-to-end manner. If we assume that the per-link delay is inferred using these methods, the inferred delay must include both queuing and propagation delays. Additionally, as in a real network, if we assume that each router and node processes not only the multicast packet but also its own background traffic, the queuing delay and packet drop rate of each router and node distribute almost randomly. We also emphasize the successful arrival of a packet; thus, parameter t of Equation (3) denotes a current delay incurred from congestion/flow control. It is also assumed that packet losses at each link are independent.

Figure 2 shows the expected delivery model that reflects heterogeneity of per-link delay and loss rate. In Figure 2, $\langle p, d \rangle$ on each link represents ⟨per-link loss rate, per-link delay⟩.

A summary of the used notations is given in Table I.

According to [31], the number of subsequent lost packets follows the expectation of the geometric distribution. Thus, if we denote the number of subsequent lost packets that are lost on the path between proxy a and receiver b as n_{ab}, the following equation holds:

$$n_{ab} = \frac{1 - \prod_{i \in \psi(a,b)} (1 - p_i)}{\prod_{i \in \psi(a,b)} (1 - p_i)} \tag{1}$$

In our HRM model, a receiver recognizes a packet loss by detecting packet sequence missing. Thus, the required time to detect a packet loss at receiver b is expressed as follows:

$$(n_{ab} + 1) \times t \tag{2}$$

DOI: 10.1002/dac

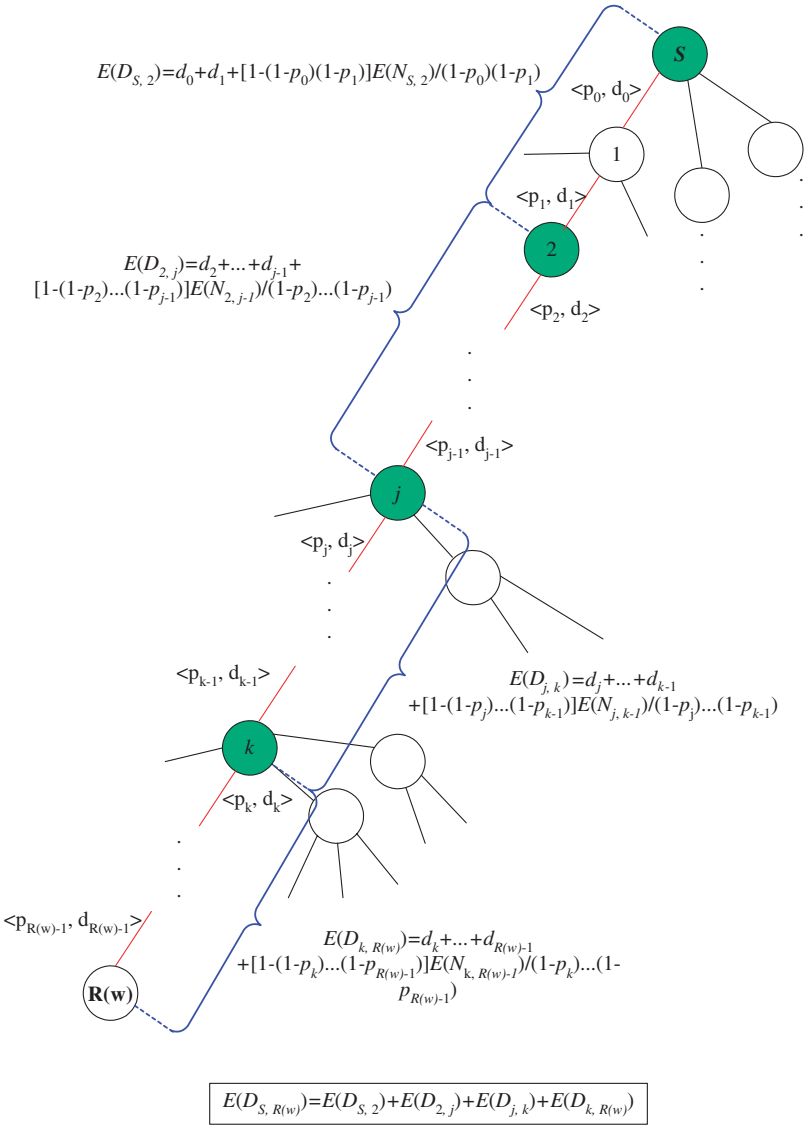

Figure 2. Expected delivery delay model reflecting heterogeneity: expected delivery delay between node S and $R(w)$ when at each node $S, 2, j$ and k, a repair server is located.

where t is the inter-packet delay (gap). Thus, $E(N_{a,b})$ can be expressed as follows:

$$E(N_{a,b}) = (n_{ab}+1)\times t + L_{a,b} \tag{3}$$

where $L_{a,b}$ indicates the delay that is needed by receiver b to deliver NACK or immediate ACK to proxy a through the out-of-band channel. $E(D_{S,R(w)})$ can be expressed in two cases as

DOI: 10.1002/dac

Table I. Notations for the expected delivery model.

Value	Description
S	Sender node
$R(w)$	Receiver node
$L_{a,b}$	Summation of per-link delays on all links between node a and node b
$E(N_{a,b})$	When node a acts as a repair proxy of node b, the expected time for node b to detect a packet loss and deliver a feedback to node a
$p(a,b)$	Set of all repair servers located on the path between node a and node b
$q_{a,b}$	Number of elements in set $p(a,b)$
$E(D_{S,R(w)})$	Expected delivery delay between the sender node s and the receiver node $R(w)$
$y(a,b)$	A set of all links between node a and node b.

follows:

(1) *In case* $\pi(S, R(w)) = \{S\}$ (*S is a unique proxy between S and* $R(w)$): Since there is no proxy between the source S and the receiver $R(w)$, a packet dropped on the path between S and $R(w)$ can be recovered from only S. Thus, $E(D_{S,R(w)})$ is expressed as Equation (4), where $B_{S,R(w)}$ and $Rcv_{S,R(w)}$ are the probability that there is no packet loss and the delay required to recover a packet loss on the path between S and $R(w)$, respectively.

$$E(D_{S,R(w)}) = B_{S,R(w)} \times L_{S,R(w)} + (1 - B_{S,R(w)}) \times Rcv_{S,R(w)} \tag{4}$$

$Rcv_{S,R(w)}$ is composed of the packet loss detection delay at $R(w)$, the delay to deliver a feedback (NACK or immediate ACK) to S, i.e. the first term of Equation (5), and retransmission delay from S. Thus,

$$Rcv_{S,R(w)} = E(N_{S,R(w)}) + E(D_{S,R(w)}) \tag{5}$$

and

$$B_{S,R(w)} = \prod_{i \in \psi(S,R(w))} (1 - p_i) \tag{6}$$

Consequently, $E(D_{S,R(w)})$ is

$$E(D_{S,R(w)}) = \left[\prod_{i \in \psi(S,R(w))} (1 - p_i) \right] L_{S,R(w)} + \left[1 - \prod_{i \in \psi(S,R(w))} (1 - p_i) \right] (E(N_{S,R(w)}) + E(D_{S,R(w)})) \tag{7}$$

By eliminating $E(D_{S,R(w)})$ on the right-hand side of Equation (7), we obtain

$$E(D_{S,R(w)}) = L_{S,R(w)} + \frac{1 - \prod_{i \in \psi(S,R(w))} (1 - p_i)}{\prod_{i \in \psi(S,R(w))} (1 - p_i)} E(N_{S,R(w)}) \tag{8}$$

(2) *In case* $\theta(S, R(w)) \geqslant 2$ *and* $\text{node}(j) \in \pi(S, R(w))$: In this case, $\text{node}(j)$ located on the path between S and $R(w)$ plays a role of a proxy for $R(w)$. Thus, a packet dropped on the path between $\text{node}(j)$ and $R(w)$ can be recovered by $\text{node}(j)$. A packet dropped on the path between

DOI: 10.1002/dac

S and node(j) should be recovered from S. Now the expected delivery delay between S and $R(w)$ is expressed as the time to detect a packet loss at node(j)+feedback transmission delay from node(j) to $S+E(D_{S,R(w)})$.

Therefore, if node(j) is a proxy on the path between source S and receiver $R(w)$ (Refer Figure 4), $E(D_{S,R(w)})$ is expressed as follows:

$$E(D_{S,R(w)}) = B_{S,\text{node}(j)} \times (L_{S,\text{node}(j)} + E(D_{\text{node}(j),R(w)})) + (1 - B_{S,\text{node}(j)}) \times Rcv_{S,R(w)} \tag{9}$$

$Rcv_{S,R(w)}$ is composed of the packet loss detection delay at node(j), the delay to deliver a feedback (NACK or immediate ACK) to S and the expected delivery delay required to retransmit the lost packet to $R(w)$ from S. Thus,

$$Rcv_{S,R(w)} = E(N_{S,\text{node}(j)}) + E(D_{S,R(w)}) \tag{10}$$

Consequently, $E(D_{S,R(w)})$ is

$$\begin{aligned} E(D_{S,R(w)}) = &\left[\prod_{i \in \psi(S,\text{node}(j))} (1 - p_i)\right](L_{S,\text{node}(j)} + E(D_{\text{node}(j),R(w)})) \\ &+ \left[1 - \prod_{i \in \psi(S,\text{node}(j))} (1 - p_i)\right](E(N_{S,\text{node}(j)}) + E(D_{S,R(w)})) \end{aligned} \tag{11}$$

$$E(D_{S,R(w)}) = L_{S,\text{node}(j)} + \frac{1 - \prod_{i \in \psi(S,\text{node}(j))} (1 - p_i)}{\prod_{i \in \psi(S,\text{node}(j))} (1 - p_i)} E(N_{S,\text{node}(j)}) + E(D_{\text{node}(j),R(w)}) \tag{12}$$

By definition of $E(D_{S,R(w)})$,

$$L_{S,\text{node}(j)} + \frac{1 - \prod_{i \in \psi(S,\text{node}(j))} (1 - p_i)}{\prod_{i \in \psi(S,\text{node}(j))} (1 - p_i)} E(N_{S,\text{node}(j)}) \equiv E(D_{S,\text{node}(j)}) \tag{13}$$

Therefore, $E(D_{S,R(w)})$ can be expressed as

$$E(D_{S,R(w)}) = E(D_{S,\text{node}(j)}) + E(D_{\text{node}(j),R(w)}) \tag{14}$$

If node(k) $\in \pi$(node(j), $R(w)$), i.e. if node(k) is a proxy on the path between node(k) and $R(w)$, we obtain

$$E(D_{\text{node}(j),R(w)}) = E(D_{\text{node}(j),\text{node}(k)}) + E(D_{\text{node}(k),R(w)}) \tag{15}$$

Thus, by setting, $\pi(S, R) = \{S, \text{proxy}_0, \text{proxy}_1, \ldots, \text{proxy}_z\}$, we obtain a recursive form as follows:

$$\begin{aligned} E(D_{S,R}) &= E(D_{S,\text{proxy}_0}) + E(D_{\text{proxy}_0,R}) \\ E(D_{\text{proxy}_0,R}) &= E(D_{\text{proxy}_0,\text{proxy}_1}) + E(D_{\text{proxy}_1,R}) \\ &\vdots \\ E(D_{\text{proxy}_{z-1},R}) &= E(D_{\text{proxy}_{z-1},\text{proxy}_z}) + E(D_{\text{proxy}_z,R}) \end{aligned} \tag{16}$$

Therefore, using this recursive form, we compute the expected delivery delays of all receivers if a set of repair servers is configured. Therefore, this expected delivery delay model is used to compute the makespan.

DOI: 10.1002/dac

4. MIXED INTEGER PROGRAMMING FORMULATION

The problem of minimizing makespan in HRM can be formulated as follows:

Given a tree $T(V, E)$, let $n = |V|$ and V_L be the set of leaf nodes in T, and let $m = |V_L|$. Find the optimal locations of k-repair servers in T to minimize the makespan.

The above formulation can be cast into an MIP formulation. The notations for the MIP are given as follows: $x_{ij} : v_j$ is a repair server of v_i, and v_i must be either a repair server or a leaf node. $y_j : v_j$ is selected as a repair server. $z_i : v_i$ is a leaf node. r_{ij}^k: Both v_i and v_j are on the path to the v_k from the sender, and v_i is an ancestor of v_j. p_{ij}: Both v_i and v_j are on the same path. w_{ij}^l: v_l lies on the path between v_i and v_j, and v_j is an ancestor of v_i. d_{ij}: Expected delivery delay between v_i and v_j and calculated using the expected delivery delay model in Section 3.2.

Our objective is

Minimize *Makespan*

subject to

$$\sum_{j=0}^{n-1} y_i = k \qquad \text{Constraint (1)}$$

$$x_{ij} \leqslant y_j \quad \text{for each } v_i, v_j \in V \qquad \text{Constraint (2)}$$

$$x_{ij} \leqslant y_i + z_i \quad \text{for each } v_i, v_j \in V \qquad \text{Constraint (3)}$$

$$x_{ij} \leqslant 1 - w_{ij}^l y_l \quad \text{for each } v_i, v_j, v_l \in V \qquad \text{Constraint (4)}$$

$$x_{ij} \leqslant p_{ij} \quad \text{for each } v_i, v_j \in V \qquad \text{Constraint (5)}$$

$$\sum_{j=0}^{n-1} \sum_{i=0}^{n-1} x_{ij} = k + m - 1 \qquad \text{Constraint (6)}$$

$$\textit{Makespan} \geqslant \sum_{j=0}^{n-1} \sum_{i=0}^{n-1} d_{ij} x_{ij} r_{ij}^k \quad \text{for each } v_k \in V_L \qquad \text{Constraint (7)}$$

$$x_{ij} \in \{0, 1\} \quad \text{for each } v_i, v_j \in V \qquad \text{Constraint (8)}$$

$$y_j \in \{0, 1\} \quad \text{for each } v_j \in V \qquad \text{Constraint (9)}$$

$$z_i \in \{0, 1\} \quad \text{for each } v_i \in V \qquad \text{Constraint (10)}$$

$$w_{ij}^l \in \{0, 1\} \quad \text{for each } v_i, v_j, v_l \in V \qquad \text{Constraint (11)}$$

$$p_{ij} \in \{0, 1\} \quad \text{for each } v_i, v_j \in V \qquad \text{Constraint (12)}$$

$$r_{ij}^k \in \{0, 1\} \quad \text{for each } v_i, v_j \in V \text{ and for each } v_k \in V_L \qquad \text{Constraint (13)}$$

Constraint (1) makes sure that the number of available repair servers is limited to k. Constraint (2) guarantees that node v_j must be a repair server if some other node v_i receives repair packets from node v_j. Constraint (3) ensures that node v_i must be a repair server or a leaf node if the binary

DOI: 10.1002/dac

variable x_{ij} is 1. Also, through the third constraint, constraint (7) guarantees that the expected delivery delay is computed on the path only between repair servers or between a repair server and a leaf node. Constraint (4) makes sure that no repair server exists on the path between node v_j and node v_i except nodes v_j and v_i if node v_i receives retransmissions from node v_j, and also guarantees that node v_j is an ancestor of node v_i. Constraint (5) ensures that node v_i is assigned to node v_j that lies on the same path with node v_i. Constraint (6) assures that the number of sections on which the expected delivery delays are computed is the same as the number of sections between repair servers or a repair server and a leaf node. Constraint (7) ensures that a maximum expected delivery delay cannot exceed the value '*Makespan*', which coincides with the definition of the makespan. Finally, constraints (8)–(13) define the type of each variable and parameter.

5. HEURISTICS FOR THE GREEDY SELECTION OF REPAIR SERVERS

We propose three heuristics in this section. We also give a random selection of repair servers, which serves as a base line for the evaluation of our three heuristics. In each heuristic, we assume that the sender node itself is a repair server by default as in other HRM methods. Therefore, each heuristic selects additional $k-1$ repair servers if we intend to select k repair servers in total.

5.1. Max expected link delay first: HMaxDelay

Our first heuristic is to use an expected delivery delay of outbound[§] incident edge. We refer to this heuristic as *HMaxDelay*. It picks a node to which the largest expected delivery delay edge is incident. The greedy heuristic is based on the intuition that a large expected delivery delay can be brought by high packet loss rate; hence, if we place repair servers at the nodes whose incident edges have larger expected delivery delay than others, more reduced retransmission delays are obtained and this leads to more reduced makespan. Figure 3 illustrates the algorithm of *HMaxDelay* heuristic.

In an HRM tree $T(V, E)$, we assume that every node $v \in V$ is selected as the repair server and then compute the expected delivery delay of each edge $e \in E$ using Equation (16) described in Section 3.2. In our model, the expected delivery delay can be computed between repair servers, or between a repair server and a leaf node. However, since we must obtain the expected delivery delay of each single edge, we just assume that all nodes except leaf nodes are selected as the repair servers. After that, select the node as the repair server to which the highest expected delivery delay edge is incident. This process iterates till $k-1$ repair servers are found or no more repair server candidate is available.

For the *HMaxDelay* algorithm, if there are n repair server candidates and we select $k-1$ repair servers among them, we must iterate the process of finding max delay edge $k-1$ times. Therefore, it takes $O(k \log n)$.[¶]

[§] In this paper, the words 'downward' and 'outbound' indicate the directions from the sender to the leaf, and upward indicates the opposite direction.

[¶] It takes $\log n$ steps to find the maximum key among n elements.

DOI: 10.1002/dac

```
HMaxDelay(T(V, E), repair_server_size k-1) {
1.    V_L=all leaf nodes in T;
2.    list L=V-V_L;
3.    list Repair_Server_Set=ø;
4.    Compute_Expected_Delay(e∈E) /*Compute the expected delivery delay of each edge e∈E*/
5.    server_size = 0;
6.    while (server_size < k-1 || L is not empty) {
7.      list_index = Find_Largest_Delay(L); /*the index in L whose outbound incident edge has the
      largest expected delivery delay*/
8.      if (L[list_index] is not the root) {
9.          Repair_Server_Set[server_size] =  L[list_index];
10.         Clear L[list_index];
11.         server_size++;
12.     } else
13.         Clear L[list_index];
      }
}
```

Figure 3. Heuristic algorithm of *HMaxDelay*.

5.2. Max out-degree first: HMaxDegree

Our second heuristic is to select repair servers based on node degree. We refer to this heuristic as *HMaxDegree*. It picks a node as the repair server that has the highest degree. The intuition is that by selecting repair servers based on their own degree, more nodes can be serviced fast, and the number of repair servers required can be reduced. Therefore, it is more probable to obtain reduced makespan. The algorithm of *HMaxDegree* heuristic is described in Figure 4. *HMaxDegree* also takes $O(k \log n)$ steps.

5.3. Longest path first: HLongPath

Let P be a set of paths from the sender to all leaf nodes. The third heuristic is to find the longest path $p_i \in P$—that has the largest expected delivery delay—and select a repair server on the path p_i to which the largest expected delivery delay edge is incident. We refer to this heuristic as *HLongPath*. The intuition is that because makespan is determined by the longest path, we can reduce makespan through reducing the expected delivery delay of the longest path first. The *HLongPath* algorithm is illustrated in Figure 5.

First the algorithm finds the longest path p_i. Then we assume that every node in the path p_i is a repair server and selects a repair server on the path p_i whose outbound incident edge has the largest expected delivery delay. After selecting the repair server, if more repair servers are available, computing the expected delivery delay of each path, finding the longest path and selecting a repair server are repeated, respectively, until k repair servers are—including the sender node—selected.

If there are m leaf nodes, it takes $\log m$ step to find the longest path p_i. Then locating j repair server candidates on the path p_i takes $\log j$ step to find the node whose incident edge has the

DOI: 10.1002/dac

```
HMaxDegree(T(V, E), repair_server_size k-1) {
1.      V_L=all leaf nodes in T;
2.      list L=V-V_L;
3.      list Repair_Server_Set=ø;
4.      server_size = 0;
5.      while (server_size < k-1 || L is not empty) {
6.        list_index = Find_Largest_Degree(L); /*the index in L who has the highest degree*/
7.        if (L[list_index] is not the root) {
8.            Repair_Server_Set[server_size] =  L[list_index];
9.            Clear L[list_index];
10.           server_size++;
11.       } else
12.           Clear L[list_index];
        }
}
```

Figure 4. Heuristic algorithm of *HMaxDegree*.

```
HLongPath(T(V, E), repair_server_size k-1) {
1.      V_L = all leaf nodes in T;
2.      P = a set of paths between the root and all v ∈ V_L;
3.      list Repair_Server_Set=ø;
4.      server_size = 0;
5.      while (server_size < k-1) || (L is not empty) {
6.        Find_Largest_Path(p_i ∈ P); /*Find the path p_i ∈ P that has the largest expected delivery
7.      delay*/
8.        Locate_Repair_Server(p_i ∈ P); /*Assume that repair servers are located in all nodes but v ∈ V_L
9.      on the path p_i*/
10.       Compute_Expected_Delay(e ∈ p_i); /*Compute expected delivery delay of each edge on the
        path p_i*/
11.       V_y = all nodes on the path p_i;
12.       v_j= Find_Unmarked(V_y); /*Find a unmarked node v_i in V_y such that its outbound incident edge
        has the largest expected delivery delay*/
13.       if (v_j found) {
14.         Repair_Server_Set[server_size] =  v_i;
15.           Mark v_i;
16.         server_size++;
          }
        }
}
```

Figure 5. Heuristic algorithm of *HLongPath*.

DOI: 10.1002/dac

largest expected delivery delay. If all paths have j repair server candidates, and in every repetition, a repair server is selected from a different path, this algorithm needs $(k-1)(\log j+\log m)$ steps. Therefore, this algorithm takes $O(k\log m)$ or $O(k\log j)$.

5.4. Lower bound of the number of repair servers

We would like to compare the required number of repair servers by our heuristic to the minimum number of repair servers assuming a given makespan. If the obtained makespan using our heuristic is applied to the following MIP formulation, we can compute the minimum required number of repair servers. Therefore, we can evaluate our heuristic in terms of the required number of repair servers that satisfy the makespan.

The following MIP also cannot be solved in reasonable time due to its exponential time complexity; hence, we obtain a crude lower bound (LP bound) and compare our heuristic with the lower bound. We can obtain the LP bound in a short time through LP-relaxation of the MIP. LP bound is obtained by removing all the integer constraints of the MIP. Thus, if the objective is minimization, the LP bound is the lower bound of the MIP integer solution.

Minimize $\sum_{j=0}^{n-1} y_i$

subject to

$x_{ij} \leqslant y_j$ for each $v_i, v_j \in V$ Constraint (1)

$x_{ij} \leqslant y_i + z_i$ for each $v_i, v_j \in V$ Constraint (2)

$x_{ij} \leqslant 1 - w_{ij}^l y_l$ for each $v_i, v_j, v_l \in V$ Constraint (3)

$x_{ij} \leqslant p_{ij}$ for each $v_i, v_j \in V$ Constraint (4)

$$\sum_{j=0}^{n-1}\sum_{i=0}^{n-1} x_{ij} = \sum_{j=0}^{n-1} y_i + m - 1 \quad \text{Constraint (5)}$$

$$C \geqslant \sum_{j=0}^{n-1}\sum_{i=0}^{n-1} d_{ij} x_{ij} r_{ij}^k \quad \text{for each } v_k \in V_L \quad \text{Constraint (6)}$$

$x_{ij} \in \{0,1\}$ for each $v_i, v_j \in V$ Constraint (7)

$y_j \in \{0,1\}$ for each $v_j \in V$ Constraint (8)

$z_i \in \{0,1\}$ for each $v_i \in V$ Constraint (9)

$w_{ij}^l \in \{0,1\}$ for each $v_i, v_j, v_l \in V$ Constraint (10)

$p_{ij} \in \{0,1\}$ for each $v_i, v_j \in V$ Constraint (11)

$r_{ij}^k \in \{0,1\}$ for each $v_i, v_j \in V$ and for each $v_k \in V_L$ Constraint (12)

DOI: 10.1002/dac

6. SIMULATION RESULTS

In this section, we compare the performance of our heuristics on several different network sizes. Ideally we would like to observe how different our best heuristic is from the integer optimal solution. However, we are unable to obtain the optimal solution even for a small network of just 50 nodes in reasonable time due to its exponential time complexity. Thus, our best heuristic is compared with the lower bound obtained by LP relaxation.

We use ToGenD topology generator [32] for our simulation and generate topologies of 100, 200, 500, and 1000 nodes. The numbers of links are 470, 806, 1721 and 3321, respectively. In each topology, we define the node that has index 0 as a sender node and generate the HRM tree using the shortest path algorithm in order to minimize the total delay of each sender–receiver pair. In each HRM tree, a sender node is a repair server by default and every node can be selected as the repair server except leaf nodes. Per-link delays are randomly assigned from the range of 10 to 40 ms and packet loss rates are chosen from the range of 0.0 to 0.1. In each HRM tree, the average out degree at each node is 3.

6.1. Performance comparison among greedy heuristics

Figure 6 shows measured makespan of each heuristic with respect to the number of repair servers on each network size. Each makespan is computed using the expected delivery delay model—Equations (8) and (13)—described in Section 3.

As shown in Figure 6, we observe that the heuristic *HLongPath* outperforms other heuristics in terms of makespan regardless of network size. We can also see that if each makespan arrives at some lower limit, makespan does not decrease even though we increase the number of repair servers. For instance, if no more repair server candidates are available on the path that determines the makespan, additional placement of repair servers on other paths does not alter the makespan. In the case of network size of 100 nodes, the available number of nodes that can be selected as a repair server is measured as 25. The heuristic *HRandom* arrives at the lower limit of makespan when 24 repair servers are selected. The *HMaxDelay*, *HMaxDegree* and *HLongPath* need 15, 22 and 10 repair servers, respectively, in order to arrive at the lower limit of makespan. Table II shows the lower limit of makespan and the minimum number of repair servers to attain this lower limit with respect to each heuristic on each network size. The parenthesized number in the field of network size means the available number of nodes that can be selected as a repair server in accordance with each network size. We observe that the *HLongPath* heuristic needs fewer repair servers than the other three heuristics in order to attain the lower limit of makespan. Other two heuristics, i.e. *HMaxDelay* and *HMaxDegree*, decide the locations of repair servers based on the statistics of links and nodes only. On the other hand, *HLongPath* heuristic locates repair servers on the basis of path statistics and locates them on the longest path first. Thus, *HLongPath* is more beneficial to reduce the makespan than the other heuristics.

6.2. Comparison with lower bound

Since *HLongPath* performs the best among the heuristics, we only need to compare *HLongPath* with the lower bound from the LP relaxation. The LP-relaxed MIP is computed using the simplex method. Figure 7(a) plots the makespan of *HLongPath* and its lower bound. It is observed that the result of our best heuristic, *HLongPath*, is close to the lower bound by a factor of 2.3 maximally in terms of the makespan. Figure 7(b) plots the minimum number of repair servers when it is

DOI: 10.1002/dac

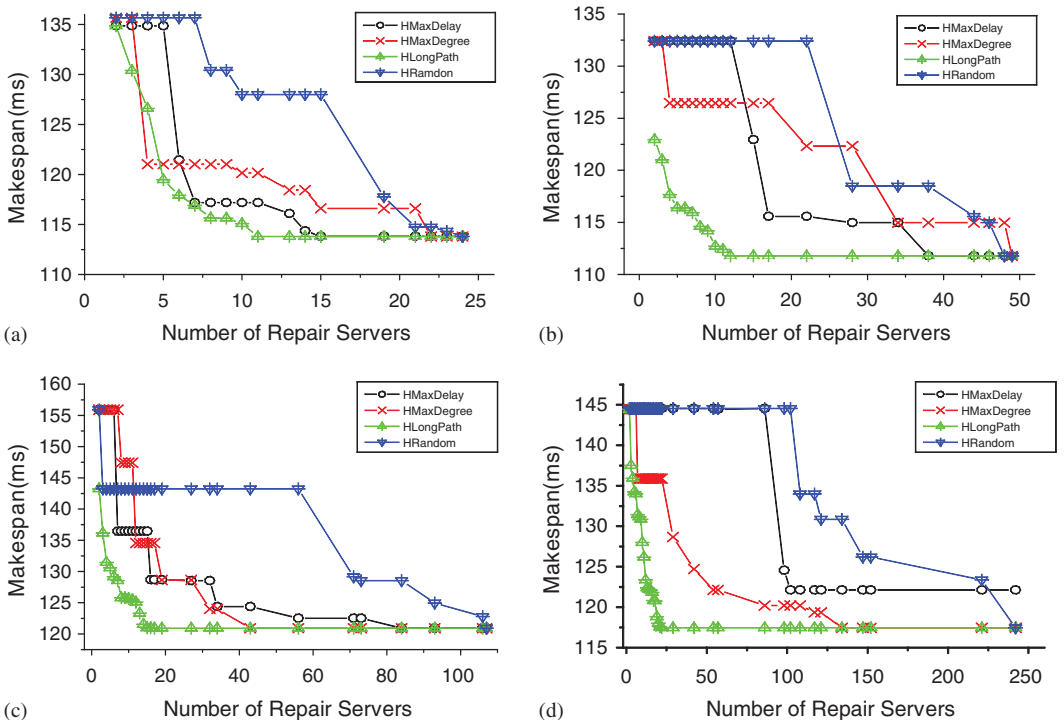

Figure 6. Makespan of each heuristic as a function of the number of repair servers. In the case of network size of (a) 100 nodes, (b) 200 nodes, (c) 500 nodes and (d) 1000 nodes.

Table II. Each tuple—(*Lower limit of a makespan* (*ms*), *minimum number of repair servers to attain this lower limit*) in accordance with each heuristic and network size.

	Network size			
Heuristic	100 (25)	200 (50)	500 (113)	1000 (244)
HRandom	(113.81, 18)	(111.78, 48)	(120.92, 107)	(117.46, 242)
HMaxDelay	(113.84, 13)	(111.78, 38)	(120.92, 84)	(122.14, 102)
HMaxDegree	(113.81, 22)	(111.78, 49)	(120.92, 43)	(117.46, 134)
HLongPath	(113.81, 11)	(111.78, 12)	(120.92, 16)	(117.46, 22)

assumed that the makespans obtained by *HLongPath* are given.[‖] The lower bound obtained from the LP relaxation of MIP in Section 5.4 is also plotted. We see that our result is close to the lower bound by a factor of 5.5 in terms of the number of repair servers when the given makespan is 113. 81 ms. On the basis of the previous work with respect to MIP optimization [33], we regard

[‖]In our system, our GLPK cannot obtain the result of LP relaxation on the 200 nodes case.

DOI: 10.1002/dac

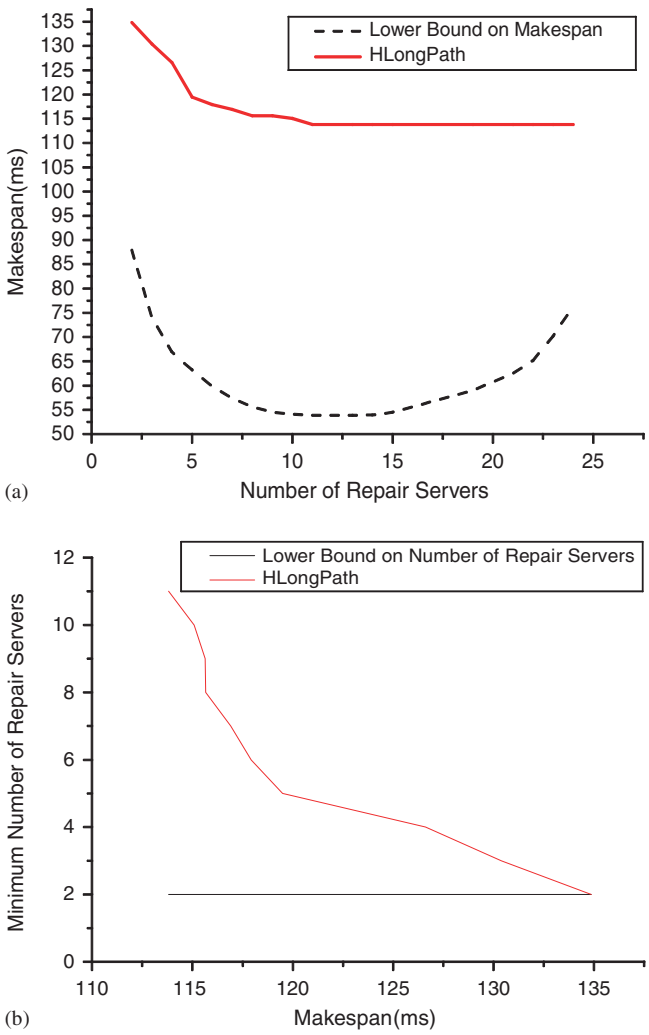

Figure 7. Comparison of *HLongPath* in a network size of 100 nodes: (a) comparison with lower bound on makespan and (b) comparison with lower bound on makespan.

the solution of a heuristic as relatively good, if the solution is close to the LP bound with a factor of approximating less than 5.

6.3. Comparison with the optimal integer solution in a small-sized network

We can obtain the integer solution of our MIP using glpk in a small-sized network. Thus we compare the result of our best heuristic with the integer solution in the network size of 25, 30, 40 and 50. The integer solution is obtained by the branch and bound. Figure 8 and Table III show the obtained results. First we compute the makespan of our heuristic and obtain the proxy sizes that cause reduction in makespan. In a network size of 25, 30 and 50, makespan reduction occurs

DOI: 10.1002/dac

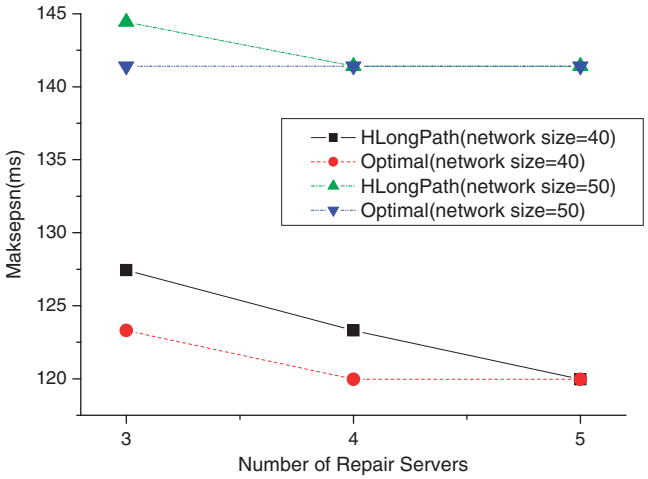

Figure 8. Comparison with optimal in a network size of 40 and 50 nodes.

Table III. Makespan of our heuristic and the integer optimal solution.

Proxy size		Network size 25	30	40	50
3	Optimal	123.31	123.31	**123.31**	**141.41**
	Greedy	123.31	123.31	**127.44**	**144.43**
4	Optimal	119.96	119.96	**119.96**	**141.41**
	Greedy	119.96	119.96	**123.31**	141.41
5	Optimal	—	—	119.96	—
	Greedy	—	—	119.96	—

at the proxy size of 3 and 4. In a network size of 40, makespan is reduced at the proxy size of 3, 4 and 5. In a network size of 25 and 30, there is no difference between our greedy heuristic and the optimal solution. However, in a network size of 40 and 50, the optimal solution shows better performance than our greedy method. In a network size of 40, our heuristic needs five proxies to obtain the makespan of 119.96; on the other hand, the optimal solution needs four proxies. Thus, we can infer that as the network size grows, the makespan difference between our heuristic and the optimal solution also increases. However, our heuristic can be useful to obtain a relatively good solution in reasonable time.

6.4. Impact on the throughput of a multicast congestion scheme

The main objective of our proposal is to provide a multicast congestion control scheme with better performance, especially sending rate. Thus, we adapt our proposal on the PGMCC through ns-2 simulation. We generate a topology with a size of 1000 nodes and 3321 links and impose a random loss rate (between 0.0 and 0.1) and delay (between 10 and 40 ms) on each link. All

DOI: 10.1002/dac

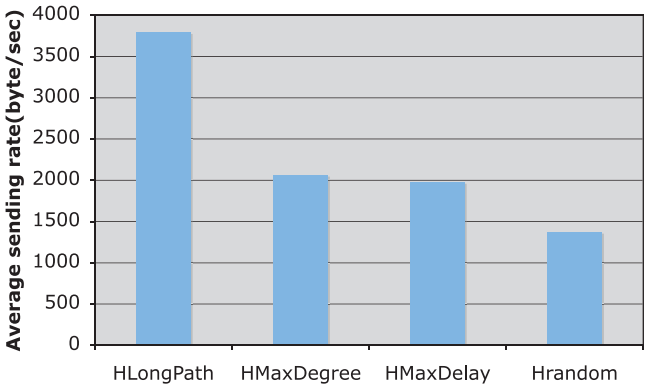

Figure 9. The sending rates in accordance with the four heuristics.

the links have a bandwidth of 1.5 Mbps. Then we construct an HRM tree using the shortest path algorithm. In the HRM tree, the average out degree at each node is 3. Also the various background traffics are generated randomly, i.e. exponential, CBR, and TCP. Each packet size is 1000 bytes and the simulation duration is 4 min. We select 20 repair proxies based on our four heuristics, i.e. *HLongPath*, *HMaxDelay*, *HMaxDegree*, and *HRandom* and impose the role of a PGM repair server on the selected proxies. Only the sender performs the congestion control.

Figure 9 illustrates the average sending rates during the simulation in accordance with the four heuristics. As shown in this figure, we observe that *HLongPath* shows the best performance among the four heuristics. *HRandom* shows the worst performance. Consequently, we are sure that, as a makespan becomes shorter, the sending rate of the PGMCC scheme becomes higher. Moreover, a shorter makespan can be obtained from the heuristic *HLongPath*.

7. CONCLUSION

Many HRM methods deploy local recovery using repair servers to reduce the delay that is required to recover packet loss, and this delay is strongly affected by the locations of repair servers.

We define makespan in HRM methods, as the time required to fully and successfully transmit a packet from the sender to all receivers. Makespan includes transmission delay and feedback delay, and if a packet loss has occurred, it includes recovery delay also. Thus, makespan can be reduced through adequate placement of repair servers. Especially, in window-based sending schemes, the more the makespan is reduced, the faster the transmission window advances, and therefore the better the throughput is achieved.

In this paper, we propose a new method to place repair serves that can reduce the makespan in heterogeneous HRM networks. We use an expected delivery delay model to reflect heterogeneity and locations of repair servers and formulate the makespan minimization problem into MIP. In addition, we propose three heuristics except the random placement policy in order to obtain the locations of repair servers in reasonable time. Through our simulations, we observe that the heuristic *HLongPath* is the best in that fewer repair servers are needed to attain the lower limit of a makespan. For a network of about 1000 nodes we can obtain the locations of repair servers in just

DOI: 10.1002/dac

a few scores of seconds, which will allow the network managers to identify the locations of repair servers in reasonable time when our method in this paper is deployed. In addition, we observe that, as a makespan becomes shorter, the sending rate of a window-based congestion control scheme, i.e. PGMCC, becomes higher.

ACKNOWLEDGEMENTS

This work was carried out during the tenure of an ERCIM 'Alain Bensoussan' Fellowship Programme.

REFERENCES

1. Chawathe Y, McCanne S, Brewer E. RMX: reliable multicast in heterogeneous networks. *Proceedings of INFOCOM*, vol. 2, Tel Aviv, Israel, 2000; 795–804.
2. Hoffman M. *A Generic Concept of Large-scale Multicast*. Lecture Notes in Computer Science, vol. 1044. Springer: Berlin, 1994; 95–106.
3. Holbrook H, Singhal S, Cheriton D. Log-based receiver-reliable multicast for distributed interactive simulation. *Proceedings of SIGCOMM*, vol. 25(4), Massachusetts, U.S.A., 1995; 328–341.
4. Holland O, Aghvami AH. Bitmapped NAK feedback aggregation for improved reliability scalability. *IEE Electronic Letters* 2005; **41**(14):828–829.
5. Yeung KL, Feng G. Cache partitioning for multiple sessions in local loss recovery of reliable multicast. *IEE Proceedings—Communications* 2005; **152**(6):866–876.
6. Floyd S, Jacobson V, Liu C-G, McCanne S, Zhang L. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking* 1997; **5**:784–803.
7. Lin J, Paul S. RMTP: a reliable multicast transport protocol. *Proceedings of INFOCOM*, vol. 3, San Francisco, U.S.A., 1998; 1414–1424.
8. Malhotra N, Ranjan S, Bagchi S. LRRM: a randomized reliable multicast protocol for optimizing recovery latency and buffer utilization. *Proceedings of SRDS*, Orlando, U.S.A., 2005; 215–225.
9. Rizzo L. PGMCC: a TCP-friendly single-rate multicast congestion control scheme. *Proceedings of SIGCOMM*, vol. 30(4), Stockholm, Sweden, 2000; 17–28.
10. Rhee I, Balaguru N, Rouskas GN. MTCP: scalable TCP-like congestion control for reliable multicast. *Proceedings of INFOCOM*, vol. 3, New York, U.S.A., 1999; 1265–1273.
11. Whetten B, Taskale G. An overview of reliable multicast transport protocol ii. *IEEE Network* 2000; **1**(14):37–47.
12. GLPK—GNU Linear Programming Kit. www.gnu.org/software/glpk/glpk.html (January 2006).
13. Nonnemacher J, Lacher M, Jung M, Biersack E, Carle G. How bad is reliable multicast without local recovery? *Proceedings of INFOCOM*, vol. 3, San Francisco, U.S.A., 1998; 972–979.
14. Markpoulou A, Tobagi F. Hierarchical reliable multicast: performance analysis and placement of proxies. *Proceedings of SIGCOMM*, Stockholm, Sweden, 2000.
15. Lin H, Yang K. Placement of repair servers to support server-based reliable multicast. *Proceedings of ICC*, Helsinki, Finland, 2001.
16. Wan Z, Kadoch M, Elhakeem A. Performance evaluation of tree-based reliable multicast. *Proceedings of ICCCN*, Dallas, U.S.A., 2003.
17. Tau C, Wang T. Performance evaluation of the loss-collected retransmission scheme in reliable multicast protocol. *IEE Proceedings—Communications* 2006; **153**:376–382.
18. Li B, Chen F, Yin L. Server replication and its placement for reliable multicast. *Proceedings of ICCCN*, Las Vegas, U.S.A., 2000.
19. Daescu O, Jothi R, Raghavachari B, Sarac K. Optimal placement of NAK-suppressing agents for reliable multicast: a partial deployment case. *Proceedings of ACM SAC*, Nicosia, Cyprus, 2004.
20. Ratasamy S, McCanne S. Scaling end-to-end multicast transports with a topologically-sensitive grouped formation protocol. *Proceedings of ICNP*, Toronto, Canada, 1999.
21. Li B, Chen F, Yin L. On the optimal placement of web proxies in the internet. *Proceedings of INFOCOM*, New York, U.S.A., 1999.
22. Chaintreau A, Baccelli F, Diot C. Impact of network delay variations on multicast sessions with TCP-like congestion control. *Proceedings of INFOCOM*, vol. 2, Anchorage, U.S.A., 2001; 1133–1142.

DOI: 10.1002/dac

23. Levine B, Paul S, Garcia-Luna-Aceves J. Organizing multicast receivers deterministically by packet-loss correlation. *Proceedings of ACM International Conference Multimedia*, Bristol, England, 1998; 201–210.
24. Casner S, Thyagarajan A. *MTRACE(8): Tool to Print Multicast Path form a Source to a Receiver*, *UNIX Manual Command*.
25. Speakman T, Crowcroft J, Gemmell J, Farinacci D, Lin S, Leshchiner D, Luby M, Montgomery T, Rizzo L, Tweedly A, Bhaskar N, Edmonstone R, Sumanasekera R, Vicisano L. PGM reliable transport protocol specification. *RFC 3208*, 2001.
26. Sarac K, Almeroth KC. Tracetree: a scalable mechanism to discover multicast tree topologies in the Internet. *IEEE/ACM Transactions on Networking* 2004; **12**(5):795–808.
27. Caceres R, Duffield NG, Horowitz J, Towsley DF. Multicast-based inference of network-internal loss characteristics. *IEEE Transactions on Information Theory* 1999; **45**(7):2462–2480.
28. Presti FL, Duffield NG, Horowitz J, Towsley D. Multicast-based inference of network-internal delay distributions. *IEEE/ACM Transactions on Networking* 2002; **10**(6):761–775.
29. Li D, Cheriton DR. OTERS (on-tree efficient recovery using subcasting): a reliable multicast protocol. *Proceedings of ICNP*, Austin, U.S.A., 1998; 237–245.
30. Zhang X, Shin KG. Delay analysis of feedback-synchronization signaling for multicast flow control. *IEEE/ACM Transactions on Networking* 2003; **11**(3):436–450.
31. Mailhofer C, Rothermel K. A delay analysis of tree-based reliable multicast protocols. *Proceedings of Computer Communications and Networks*, Scottsdale, U.S.A., 2001.
32. ToGenD—A Notre-Dame based Topology Generator. http://www.eng.tau.ac.il/∼osnaty/togend.html (January 2006).
33. Li L, Thottan M, Yao B, Paul S. Distributed network monitoring with bounded link utilization in IP networks. *Proceedings of INFOCOM*, vol. 2, San Francisco, U.S.A., 2003; 1189–1198.

AUTHORS' BIOGRAPHIES

Sang-Seon Byun received the BS and MS degrees in computer science from Korea University, Seoul, Korea, in 1996 and 2002, respectively. He received the PhD degree in computer science at Korea University, Seoul, Korea, in 2007. He was a research professor of Graduate School of Embedded Software at Korea University in 2007. He is now a research fellow in the Department of Electronics and Telecommunications, NTNU, Trondheim, Norway. His Current interests are in network measurement modeling and optimizations.

Chuck Yoo received the BS and MS degrees in electronic engineering from Seoul National University, Seoul, Korea, and the MS and PhD in computer science from the University of Michigan. From 1990 to 1995, he worked as a researcher in Sun Microsystems Lab. He is now a professor in the Department of Computer Science and Engineering, Korea University, Seoul, Korea. His research interests include high-performance network, multimedia streaming, and operating systems. He served as a member of the organizing committee for NOSSDAV 2001.

DOI: 10.1002/dac