A Peer-to-Peer Tree Based Reliable Multicast Protocol

Min Yang and Yuanyuan Yang
Department of Electrical & Computer Engineering
State University of New York, Stony Brook, NY 11794, USA

***Abstract*—Reliable multicast is critical to multicast based applications as it provides reliability over the unreliable network. Although the primary function of reliable multicast is loss recovery and flow control which are similar to that of reliable unicast, the inherent property of multicast that multiple receivers coexist in one multicast session imposes new challenges such as acknowledge implosion and poor scalability. Among existing reliable multicast protocols, tree based reliable multicast protocols can achieve the reliability in a scalable fashion. They group the receivers into a hierarchy called the ACK tree and the ACK/NACK messages and retransmitted packets are transmitted between adjacent levels. Since current tree based reliable multicast protocols construct the ACK tree based on the multicast tree which is constructed by the multicast routing protocol, the protocol performance greatly depends on the multicast tree. In this paper, we propose a peer-to-peer (P2P) tree based reliable multicast protocol which constructs the ACK tree in a flexible way as the multicast tree is constructed in a P2P system. In our protocol, any two receivers can be adjacent nodes in the ACK tree. The ACK tree construction process is based on a heuristic function which is designed to minimize the retransmission delay. The child node sends ACK/NACK to the parent node and receives retransmitted packets from the parent node. Our protocol uses window based flow control. The window in the parent node will not advance unless the parent node receives all the ACKs from its child nodes. We conducted extensive simulations to evaluate the protocol. The simulation results show that our protocol achieves good scalability with low retransmission delay and high throughput.**

Keywords: Reliable multicast, group communication, ACK tree, transport protocol, peer-to-peer.

I. Introduction and Related Work

Like its unicast counterpart, Transmission Control Protocol (TCP), a reliable multicast protocol should possess the functions such as packet loss detection and recovery, ordering and flow control. However, having multiple receivers makes the reliable multicast vastly different and much more complex than TCP. First, in unicast, the receiver uses positive acknowledgements (ACK) or negative acknowledgements (NACK) to inform the sender its current status. In multicast, many ACK/NACK messages produced by all the receivers will overwhelm the sender and congest the links around the sender, which is called "acknowledge implosion" [1]. Second, it is difficult to adapt the data transmission rate of the sender to the different data reception rates of the heterogenous receivers. Third, the wide range of requirements of the multicast applications makes it impossible to design a one-fit-all reliable multicast protocol [2]. On the other hand, most multicast applications are real-time applications, such as media streaming and audio/video conferencing, which have strict requirements on QoS, especially on delay jitter [3], and it is essential to minimize the retransmission delay when the data is lost in such applications.

There has been a lot of work on reliable multicast in the literature, see, for example, [4]-[8]. In general, reliable multicast protocols are classified into sender-initiated, receiver-initiated and tree based protocols. Both sender-initiated protocols and receiver-initiated protocols suffer the acknowledge implosion problem. Tree based protocols arrange all the receivers into a hierarchy called the ACK tree. Each receiver sends ACK/NACKs to its parent node only, collects the ACK/NACKs from its child nodes and is responsible for retransmitting lost data packets to its child nodes. By limiting the degree of the ACK tree, we can make it possible that no node is overwhelmed by ACK/NACKs.

RMTP (Reliable Multicast Transport Protocol) [7] and TMTP (Tree-based Multicast Transport Protocol) [5] are two typical tree based reliable multicast protocols. In RMTP, receivers are grouped into a hierarchy of local regions, with a Designated Receiver (DR) in each local region. Receivers send ACKs to the DR of its local region, then DRs send ACKs to the DRs in the upper level. The sender is the DR on the top level. DRs cache the data and respond to retransmission requests in the local region. In TMTP, the ACK tree is called *control tree*. Typically, in TMTP the receivers in the same subnet belong to a domain and a single domain manager acts as the parent of the other receivers in the domain. However, since both protocols build the ACK tree based on the multicast tree, this causes the following problems. First, the eligible ACK tree may not exist. In the ACK tree of RMTP, the parent of a receiver must be the ancestor of the receiver in the multicast tree. In an extreme case, all the receivers have to choose the sender as their parent. Thus, the protocol degenerates to a receiver-initiated protocol. It is even worse when there are some constraints, as it may be impossible to find an eligible ACK tree under the constraints. Second, since the parent and child in the ACK tree is the ancestor and descendant in the multicast tree, there must be some correlation among their data loss probabilities. For example, if the data is lost before reaching the parent, the child will not receive the data definitely. If the child sends the NACK to the parent, it will not receive the retransmitted data because the parent is also requesting the data. Third, the superposition of the ACK tree and the multicast tree increases the traffic on the same links, thereby increases the probability of congestion.

In this paper, we propose a new reliable multicast protocol which can minimize the retransmission delay and avoid the problems discussed above. Our protocol takes advantage of both the tree based approach and the peer-to-peer (P2P) [10] technique. In our protocol, any two receivers can be the parent node and child node in the ACK tree. A receiver uses a heuristic function to choose another receiver as its parent. The heuristic function is designed to minimize the retransmission delay. The child node sends ACK/NACK messages to the parent node and receives retransmitted packets from the parent node. The protocol uses window based flow control. The sender can only send the data packets whose sequence numbers are in the window. The receivers can only accept data packets whose sequence numbers are in the window. The window in the parent node will not advance unless the parent node receives all the ACKs from its child nodes.

The research was supported in part by the U.S. National Science Foundation under grant numbers CCR-0073085 and CCR-0207999.

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

II. Assumptions and Notations

We assume that our reliable multicast protocol is deployed in the service-centric multicast architecture proposed in [9] along with the multicast routing protocol SCMP (Service-Centralized Multicast Protocol). The service-centric multicast architecture provides flexible and efficient multicast service by processing most multicast related tasks in a powerful router, which is called *m-router*. The m-router collects JOIN requests, builds the multicast tree and distributes the tree information around the network, while other routers in the network only need to perform minimum functions for routing. One reason for adopting this multicast architecture is that the architecture provides the centralized service which our protocol can use to construct the ACK tree with little protocol overhead. Another reason is that the ACK tree information can be distributed along with the multicast tree information which saves the network bandwidth. The sender in our protocol is always the m-router in the architecture. However, our reliable multicast protocol is not restricted to the multicast architecture in [9]. It can be used in any network running a link state unicast routing protocol.

We also assume that each data packet has a sequence number which represents its order in the data stream. The receivers detect data loss or out of order based on the sequence number of the received data packets. To simplify the presentation, we use "packet n" to represent "the data packet whose sequence number is n" and "packet less than n" to represent "the data packet whose sequence number is less than n."

We use $parent_{ACK}$ to represent the parent of a receiver in the ACK tree. Similarly, we use $child_{ACK}$ to represent a child of a receiver in the ACK tree. For any receiver, there is a unique path on the multicast tree connecting the receiver to the m-router. We denote this unique path by $path_m$, the length of the path by $pathlen_m$, and the delay of the path by $delay_m$.

III. P2P Tree Based Reliable Multicast Protocol

A. Protocol Overview

Under the service-centric multicast architecture, the m-router has the full knowledge of the network topology and the receivers. Each receiver keeps the IP address of its $parent_{ACK}$. The physical links of the ACK tree are determined by unicast routing. Upon receiving a JOIN request from a new receiver, the m-router will graft it to both the multicast tree and the ACK tree. An existing receiver on the ACK tree is selected as the $parent_{ACK}$ of the new receiver. The selection is based on a heuristic function to be described in the next section. To save bandwidth, the m-router encapsulates the $parent_{ACK}$ information in the packet used for constructing the multicast tree. After receiving the $parent_{ACK}$ information, the new receiver sends ACK_JOIN to the $parent_{ACK}$ and the $parent_{ACK}$ adds the new receiver as a $child_{ACK}$.

The data packets flow along the multicast tree first. Once a receiver detects data loss, it will use the ACK tree to recover. The receiver experiencing data loss sends the NACK which includes a sequence number n and a bitmap. All the packets less than n are received correctly up to now. The length of the bitmap is a variable and the bitmap indicates whether the packets following packet n are correctly received or not. The parent retransmits the lost packets by unicasting according to the bitmap if the required packets are in its buffer.

When the physical links are unstable or the sending rate of the sender exceeds the processing ability of the receiver, our protocol uses window based flow control to inform the sender to lower its sending rate. Ideally, all the windows should advance at different but close speeds. When some receiver is experiencing data loss, its window will stop advancing which triggers the stop of the window in its $parent_{ACK}$. This chain effect finally stops the window in the sender and then the sender stops sending data packets.

B. Constructing the ACK Tree

Before we give the details of the ACK tree construction, we first introduce a new term. For any two receivers u and v, the two paths $path_m(u)$ and $path_m(v)$ may have some common links. We use $commonlink(u, v)$ to represent this number. The larger the $commonlink(u, v)$, the more correlation between the data loss probabilities of the receivers u and v.

When a new receiver joins the group, any node on the ACK tree is a candidate for the $parent_{ACK}$ of the receiver. The following is some heuristics for selecting $parent_{ACK}$:

- Rule 1: the delay between the receiver and its $parent_{ACK}$ should be minimized.
- Rule 2: the $delay_m(parent_{ACK})$ should be less than the sum of the $delay_m$ (receiver) plus the delay between $parent_{ACK}$ and the receiver.
- Rule 3: the $commonlink(parent_{ACK}, receiver)$ should be minimized.

Rule 1 is intuitive as the delay between the receiver and its $parent_{ACK}$ determines the delay of the ACK/NACK messages and the retransmitted data packets between them. Minimizing the delay helps to minimize the retransmission delay. However, the retransmission delay is affected not only by this delay, but also by the probability that the $parent_{ACK}$ has the required data packets in its buffer. Rules 2 and 3 are trying to maximize this probability. Rule 2 guarantees that the data packets will reach $parent_{ACK}$ earlier than the NACK from the receiver. Rule 3 is to minimize the correlation of the data loss probabilities between the receiver and its $parent_{ACK}$. If such correlation is strong, that is to say a data packet which is lost before arriving at the receiver is very likely lost before arriving at its $parent_{ACK}$, then it is very likely that $parent_{ACK}$ does not have the required packets when receiving the NACK from the receiver.

It is difficult to find a perfect $parent_{ACK}$ satisfying all the three rules. In fact, these three rules contradict each other in some cases. Here we give a preference heuristic function which takes all the three rules into consideration.

$$P(u,v) = \begin{cases} delay(u,v) * e^{delay_m(v) - delay_m(v) - delay(u,v)} * \\ commonlink(u,v) * (1/pathlen_m(u) + 1/pathlen_m(v)) \\ \qquad \text{if } delay_m(v) + delay(u,v) - delay_m(v) \geq 0 \\ \\ MAX \qquad \text{if } delay_m(v) + delay(u,v) - delay_m(v) < 0 \end{cases}$$

where u is a candidate for $parent_{ACK}$ and v is the new receiver, $delay(u, v)$ is the delay between u and v, and MAX is a very large number. The m-router calculates the preference values for each candidate and choose the candidate with the lowest preference value as the $parent_{ACK}$.

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

C. Loss Recovery and Flow Control

Loss recovery and flow control both strongly depend on a data structure - window. Suppose data packets consist of a stream in an ascending order of sequence numbers. A window is a span of continuous sequence numbers. The length of the window is $window\_size$. The first sequence number determines the position of the window.

A receiver uses both ACK and NACK to inform its $parent_{ACK}$ its current status. Each ACK includes a sequence number $current\_seqno$, which indicates that all the packets less than $current\_seqno$ are received correctly and the receiver is waiting for the packet $current\_seqno$. Each NACK includes a sequence number $current\_seqno$ and a bitmap. The sequence number has the same meaning as in ACK. The bitmap is stored in the window and indicates whether the packets after packet $current\_seqno$ are correctly received or not. The $parent_{ACK}$ retransmits the lost data packets according to the bitmap.

All the windows advance at their own speeds until they reach the end of the data stream. A window will not advance until some conditions are satisfied. Based on these conditions, we divide the nodes into three types: *sender* which is the root node of the ACK tree, *i-receiver* which is the inner node of the ACK tree and *l-receiver* which is the leaf node of the ACK tree. Each type of nodes take different actions when receiving ACK/NACKs. We describe each of them next.

C.1 Operation of the Sender

The window in the sender is shown in Fig.1. The packets colored in red are sent already. $current\_seqno$ is the sequence number of the packet the sender is about to send next. $last\_acked$ is the least sequence number among all the ACK/NACKs collected from the $child_{ACK}$ of the sender. After

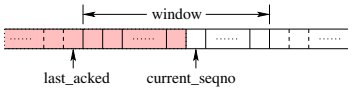


Fig. 1. The window in the sender.

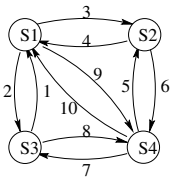
the sender sends a data packet, it increases the $current\_seqno$ by 1. The sender will keep sending data packets until the window is full, and at this time, $last\_acked + window\_size = current\_seqno$. When the sender receives ACK/NACKs from a $child_{ACK}$, it will update the $last\_acked$ if necessary. The increase of the $last\_acked$ results in the advance of the window. When the window is full, the sender will stop sending data packets. It checks the window periodically and resumes to send data packets after the window advances.

C.2 Operation of the I-Receiver

The i-receivers are the most complicated nodes among the three types of nodes because they have to do three things simultaneously: receiving the multicast data through the multicast tree, sending ACK/NACKs to their $parent_{ACK}$ and collecting the ACK/NACKs from their $child_{ACK}$. To simplify the presentation, we use a state transition diagram to describe the operation of the i-receiver. We use four states to represent the status of an i-receiver. Each i-receiver must be in one of the states.

- S_1: the i-receiver does not detect any packet loss or out of order, and the window is not full.
- S_2: the i-receiver detects packet loss or out of order, and the window is not full.
- S_3: the i-receiver does not detect any packet loss or out of order, and the window is full.
- S_4: the i-receiver detects packet loss or out of order, and the window is full.

The state transition diagram is shown in Fig.2.



1. last_acked increased
2. expected packets are received and the window is full
3. receive a packet larger than current_seqno but within the window
4. all lost packets are received
5. last_acked increased and largest_seqno is within the window
6, 8 & 9. receive a packet beyond the window
7. all lost packets are received and current_seqno>last_acked
10. all lost packets are received and current_seqno<=last_acked

Fig. 2. State transition diagram of i-receiver.

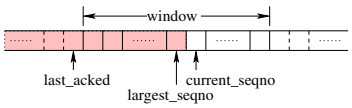


Fig. 3. The window of i-receiver in state S_1 when $last\_acked$ is less than $current\_seqno$.

Ideally, if there is no packet loss or out of order, all the receivers will stay in S_1. The window is as shown in Fig.3, where $last\_acked$ is the same as in the sender, $current\_seqno$ is the sequence number of the packet that the i-receiver is expecting to receive, the packets colored in red are received correctly already, and $largest\_seqno$ is the largest sequence number of the received packets. When the i-receiver receives packet $current\_seqno$, both $current\_seqno$ and $largest\_seqno$ increase by 1. The window advances when the $last\_acked$ increases after receiving ACK/NACK from a $child_{ACK}$.

In Section III-B, we used rule 2 to ensure that the data packets reach the $parent_{ACK}$ before the NACK from a $child_{ACK}$. But when the $parent_{ACK}$ is experiencing a much heavier congestion than the $child_{ACK}$, the window in the $parent_{ACK}$ will lag behind the window in the $child_{ACK}$. As a result, the $last\_acked$ in $parent_{ACK}$ will become larger than $current\_seqno$. This case is shown in Fig.4.

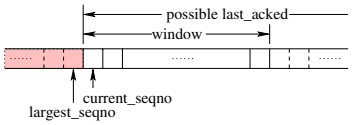


Fig. 4. The window of i-receiver in state S_1 when $last\_acked$ is equal to or greater than $current\_seqno$.

In this case, the window is empty. The only condition that the window will advance is the i-receiver receives packet $current\_seqno$. After $current\_seqno$ catches up with $last\_acked$, the $last\_acked$ will dominate the window advancement again.

If one receiver receives a packet whose sequence number is larger than $current\_seqno$, two cases are possible.

Case 1: If the sequence number of the packet is within the window, the receiver's state transits from S_1 to S_2. The window is as shown in Fig.5. $largest\_seqno$ is updated to the sequence number of the received packet. The bitmap is a bit string with length $largest\_seqno - current\_seqno + 1$, and

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

it indicates whether the packets between $current\_seqno$ and $largest\_seqno$ are received correctly or not, where "1" means the packet is received correctly and "0" means the packet is lost or damaged. The i-receiver sends NACK containing $current\_seqno$ and the bitmap to its $parent_{ACK}$.

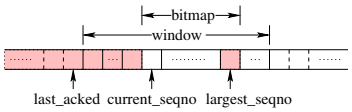


Fig. 5. The window of i-receiver in state S_2 when $last\_acked$ is less than $current\_seqno$.

Similarly, if $last\_acked$ is larger than $current\_seqno$, the window is as shown in Fig.6.

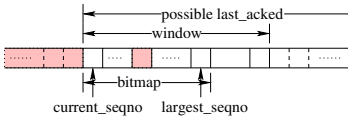


Fig. 6. The window of i-receiver in state S_2 when $last\_acked$ is equal to or greater than $current\_seqno$.

Case 2: If the sequence number of the packet is beyond the window, the receiver's state transits from S_1 to S_4. The window of S_4 is shown in Fig.7. $largest\_seqno$ is still updated to the sequence number of the received packet. But this time the bitmap only includes the packets between $current\_seqno$ and the right end of the window due to the limitation of the window size. If the window advances m packets after updating $last\_acked$, the length of the bitmap increases by m and the values of the new m bits are set to 0.

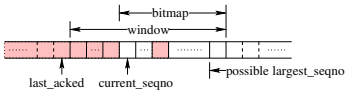


Fig. 7. The window of i-receiver in state S_4 when $last\_acked$ is less than $current\_seqno$.

Similarly, when $last\_acked$ is larger than $current\_seqno$, the window is as shown in Fig.8.

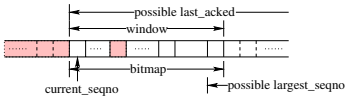


Fig. 8. The window of i-receiver in state S_4 when $last\_acked$ is equal to or greater than $current\_seqno$.

If there is no data loss or out of order, but the $last\_acked$ is not updated in time because a $child_{ACK}$ is experiencing data loss, the window in the $parent_{ACK}$ will become full shortly and the state will transit from S_1 to S_3. The window in S_3 is shown in Fig.9. In this case, the i-receiver will stop increasing $current\_seqno$ even it receives packet $current\_seqno$. But $largest\_seqno$ will be updated similarly as other three states.

When the i-receiver is in S_2, there are two situations resulting in a state transition: if the i-receiver receives a data packet beyond the window, the i-receiver transits from S_2 to S_4; or if the i-receiver receives all the lost packets recorded in the bitmap, the i-receiver transits from S_2 to S_1. When the i-receiver is in S_3, it transits to S_1 if $last\_acked$ is increased due to receiving ACK/NACK, or to S_4 if it receives a data packet beyond

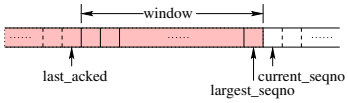


Fig. 9. The window of i-receiver in state S_3.

the window. When the i-receiver is in S_4, its $largest\_seqno$ may be much larger beyond the window. The i-receiver transits to S_2 only if the $last\_acked$ is increased and the window advances further enough that $largest\_seqno$ lies in the window again. If $largest\_seqno$ is equal to $last\_acked + window\_size$ (it should be that $largest\_seqno$ is equal to $current\_seqno + window\_size - 1$ in the case that $last\_acked$ is no less than $current\_seqno$) and all the lost packets recorded in the bitmap are received correctly, the i-receiver transits from S_4 to S_3 (it should be S_1 in the case that $last\_acked$ is no less than $current\_seqno$).

C.3 Operation of the L-Receiver

The l-receivers are similar to the i-receivers except that the l-receivers have no $child_{ACK}$. As they do not need to collect ACK/NACKs, $last\_acked$ is always equal to $current\_seqno - 1$. There are only three states for an l-receiver: S_1, S_2 and S_4. The state transition diagram is shown in Fig.10.

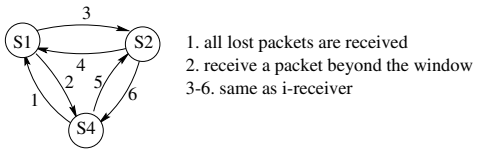


Fig. 10. State transition diagram of l-receiver.

D. Timers

Our protocol uses different timers to improve robustness and performance. There are three types of timers: poll timer, NACK timer and status timer.

Poll timer is scheduled in the sender when the window is full. When the timer is timeout, the sender checks whether the window advances. If the window advances, the sender resets the timer and resumes sending data packets. If the window is still full, the sender reschedules the timer.

When a receiver detects data loss, it sends an NACK to its $parent_{ACK}$ immediately. Although this helps to decrease the retransmission delay, it may waste a lot of bandwidth. For example, the "lost" packet is not really lost, it simply experiences a longer delay than the packet after it. In this case, the NACK is redundant and unnecessary. We use the NACK timer to avoid such situation. Every time the receiver detects data loss, it schedules an NACK timer. If the packet is still missing when the timer is timeout, the receiver then sends out the NACK.

Status timer is used by a $child_{ACK}$ to periodically send status information to its $parent_{ACK}$. When the timer is timeout, the receiver will send an ACK or NACK based on its current status. If the receiver is in states S_1 or S_3, it will send an ACK; if it is in states S_2 or S_4, it will send an NACK.

IV. Performance Evaluation

We have implemented our protocol on the NS2 simulator and evaluated the performance through simulations. For compari-

1-4244-0357-X/06/$20.00 ©2006 IEEE
This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.

son purpose, we also implemented another protocol, which can be considered as a simplified version of our protocol. In this variation protocol, all the receivers send ACK/NACKs directly to the sender. Compared to the original protocol, the variation protocol does not have a hierarchical structure for loss recovery and flow control, and the height of the ACK tree in the variation protocol is always 2. We focus on two metrics: average retransmission delay and throughput. The retransmission delay is the time experienced by a receiver from it sends out an NACK packet until it receives all the required data packets. The throughput is defined as the number of packets the sender has sent before the simulation completes, which is equal to the $current\_seqno$ of the sender.

The network topologies are generated by GT-ITM [11]. Nodes are picked randomly to join a multicast group. The sender sends data packets at a constant rate as long as the window is not full. The simulation time is 50 seconds. There are two tunable parameters: packet drop probability p and the window size $window\_size$. Each link drops the data packets randomly with probability p.

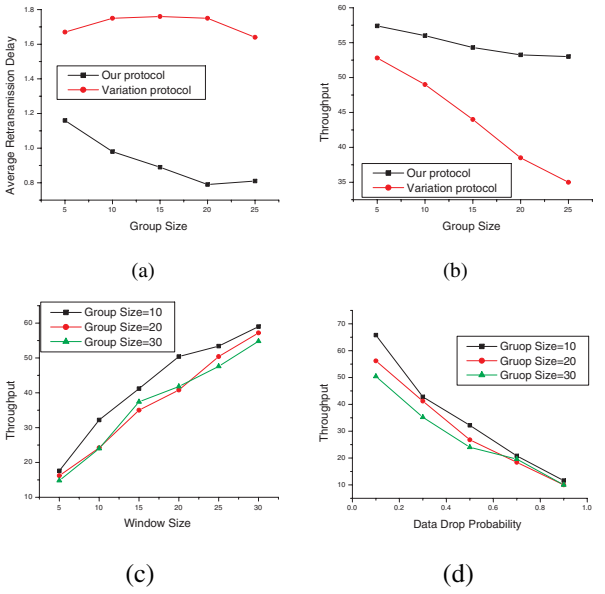


Fig. 11. Simulation results. (a) Average retransmission delay vs. group size; (b) Throughput vs. group size; (c) Throughput vs. window size; (d) Throughput vs. packet drop probabilities.

Fig.11(a) shows the average retransmission delay under different group sizes. We can see that the average retransmission delay of our protocol is much shorter than that of the variation protocol no matter what the group size is. The average retransmission delay of the variation protocol increases slightly as the group size increases. On the other hand, as the increase of the group size, the average retransmission delay of our protocol decreases. This is because that when the group size increases, the number of the $parent_{ACK}$ candidates increases too. It is more likely to find an eligible $parent_{ACK}$ close to the receiver. When the group size becomes large enough, the average retransmission delay almost remains constant.

Fig.11(b) shows the throughput under different group sizes. As can be seen, the throughput of our protocol is always greater than that of the variation protocol. The throughput of both protocols decreases as the group size increases. But the throughput of our protocol decreases at a much lower pace. When the group size becomes large enough, the decrease of the throughput is slim. From these observations, we can see that our protocol scales well when the group size increases.

Fig.11(c) shows the throughput under different window sizes. We can observe that as the increase of the window size, the throughput increases as well. The throughput when the group size is 10 is better than that when the group size is 20 or 30. The two curves of group size is 20 and 30 interweave with the increase of the window size. It means that the group size has little impact on throughput when it is large enough.

Fig.11(d) shows the throughput under different packet drop probabilities. We can see that as the increase of the packet drop probability, the throughput decreases. Still, the throughput when the group size is smaller is better than that when the group size is larger. However, this advantage diminishes when the packet drop probability is high enough.

V. Conclusions and Future Work

In this paper, we have proposed a P2P tree based reliable multicast protocol. Compared to existing reliable multicast protocols, our protocol can avoid the acknowledge implosion and minimize the retransmission delay. Constructing the ACK tree in a P2P fashion makes our protocol transparent to routers and easy to deploy. In the proposed window based loss recovery and flow control scheme, the window in the child node can advance faster than the window in the parent node, which can greatly increase the throughput. Our simulation results show that the new protocol can achieve good scalability. Its average retransmission delay and throughput are much better than the variation reliable multicast protocol. In the new protocol, the throughput increases when the window size increases or the drop probability decreases, and the throughput differences between different group sizes are marginal. Our future work will focus on how to increase the throughput by using multiple $parent_{ACK}$ and how to adjust the ACK tree dynamically.

References

[1] B. Rajagopalan, "Reliability and Scaling issues in Multicast Communication," *ACM SIGCOMM '92*, 1992, pp. 188-198.

[2] S. Floyd, et al., "A Reliable Multicast Framework for Light-weight Sessions and Application Level Framing," *IEEE/ACM Trans. Networking*, vol.5, no.6, 1997, pp. 784-803.

[3] X. Li, M.H. Ammar and S. Paul, "Video Multicast over the Internet," *IEEE Network Magazine*, April 1999. pp. 46-60.

[4] J.W Atwood, "A Classification of Reliable Multicast Protocols," *IEEE Network*, vol.13, no.3, 2004, pp. 24-34.

[5] R. Yavatkar, J. Griffioen and M. Sudan, "A reliable dissemination protocol for interactive collaborative applications," *Proc. ACM Multimedia*, San Francisco, CA, 1995, pp. 333-344.

[6] B.N. Levine and J.J. Garcia-Luna-Aceves, "A Comparison of Reliable Multicast Protocols," *Multimedia System*, vol.6, no.5, 1998, pp. 334-348.

[7] J. Lin and S. Paul, "RMTP: A reliable multicast transport protocol," *IEEE INFOCOM '96* , 1996, pp. 1414-1424.

[8] P. Radoslavov et al., "A Comparison of Application-level and Router-assisted Hierarchical Schemes for Reliable Multicast," *IEEE/ACM Trans. Networking*, vol.12, no.3, 2004, pp. 469-482.

[9] Y. Yang, J. Wang and M. Yang, "A Service-Centric Multicast Architecture and Routing Protocol," *Proc. Int'l Conf. Parallel Processing(ICPP '06)*.

[10] E-K Lua, J. Crowcroft, M. Pias, R. Sharma and S. Lim, "A Survey and Comparison of Peer-to-Peer Overlay Network Schemes," *IEEE Communications*, Mar. 2004.

[11] http://www.cc.gatech.edu/projects/gtitm/.

1-4244-0357-X/06/$20.00 ©2006 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE GLOBECOM 2006 proceedings.