http://dx.doi.org/10.5755/j01.eee.19.10.5909 ELEKTRONIKA IR ELEKTROTECHNIKA, ISSN 1392-1215, VOL. 19, NO. 10, 2013

An Acknowledgement Mechanism for Reliable Application Layer Multicast

Xinchang Zhang[1,2], Weidong Gu[1], Ye Li[1], Haitao Xu[3]
[1]*Shandong Key Laboratory of Computer Networks, Shandong Computer Science Center, Jinan, 250101, China*
[2]*DNSLAB，China Internet Network Information Center, Beijing, 100190, China*
[3]*Department of Computer Science, College of William and Mary, TWilliamsburg, VA 23185, USA*
sddguwd@163.com

[1]***Abstract*—This paper proposes a tree-based acknowledgement mechanism designed for a reliable application layer multicast, called ATACK. The proposed mechanism divides the group members into a hierarchy of logical acknowledgement domains by a self-organized way, and further builds the acknowledgement tree in terms of hierarchical acknowledgement domains. The expected size of the acknowledgement domain is within some fixed range, which promises the efficient acknowledgement and avoids adjusting acknowledgement domains frequently. In each domain, only a part of group members send acknowledgement messages to the corresponding acknowledgement processor according to the error correlation of the ALM delivery tree. Therefore the proposed acknowledgement mechanism can effectively reduce the related load and improves the acknowledgement performance.**

***Index Terms*—Application layer multicast, reliable multicast, acknowledgement mechanism, tree.**

I. Introduction

As an alternative of IP multicast, application layer multicast (ALM) implements multicast functionality at the application layer instead of the IP layer [1]–[7]. In ALM, network infrastructures need no additional modification, which addresses the problem of non-ubiquitous deployment of IP Multicast. There are two main types of ALM services, i.e., reliable service and loss-tolerant service [8]–[9]. In reliable service, the data source distributes the complete and right data to all the active receivers. Examples include file distribution and online games.

In ALMI [2], data distribution along the multicast tree occurs on a hop by hop fashion. Depending on the application, the data transfer between two adjacent members can be reliable or unreliable by deploying TCP or UDP, respectively. When TCP is used, a connection has to be established between two adjacent nodes with one end initiating and the other end accepting the connection.

Manuscript received March 15, 2013; accepted August 12, 2013.
This research was funded by the National Natural Science Foundation of China under Grant No. 61272433 and 61070039, the Outstanding Young and Middle-aged Scholars Foundation of Shandong Province of China under Contract No. BS2011DX033 and the Shandong provincial Natural Science Foundation of China under Grant No. ZR2010FQ008.

Providing end-to-end reliability through TCP is a choice for implementing reliable ALM. However, it is difficult for this approach to obtain effective flow control in the viewpoint of group communication, because the end-to-end transmissions are asynchronous. For the same reason, the approach needs frequent data numbering and renumbering operations. Additionally, some end-to-end transmissions might be broken because the members can randomly leave the multicast session, and it is unsuitable for reliable ALM to use TCP-based approach in short duration multicast services (e.g. news report) [9].

Unlike reliable ALM, reliable IP multicast has been widely researched over the past two decades. Existing reliable IP multicast protocols usually are based on UDP. In UDP-based reliable IP multicast, the tree-based acknowledgement (ACK) is an important mechanism that can achieve good throughput with desirable scalability. This mechanism arranges the group members into a logical tree, so that each leaf sends ACKs to its parent which aggregates them, and passes them on up the tree. The tree-based acknowledgement mechanism is proven most scalable in terms of throughput, because they can ensure that the sender's processing time is bounded by the number of its immediate children, which remains constant in the tree hierarchy regardless of the number of receivers [10]. In addition, the tree-based acknowledgement mechanism can be independent of the loss recovery way (see RMTP-II [11], GAM [12] and TMTP), which enhances the flexibility of the corresponding application.

In the delivery tree of IP multicast, branch nodes (other than the root) are multicast routers, and leaf nodes are group members. In contrast, no-leaf nodes of ALM delivery trees are dynamic group members instead of multicast routers. Therefore there is close error correlation in the application layer multicast, i.e., a loss at some node must result in the same loss at the downstream nodes of the node. Because of the above intrinsic difference, it is unwise for reliable ALM to directly leverage some existing tree-based ACK mechanism approaches designed for reliable IP multicast. To our best knowledge, so far there is no tree-based ACK mechanism designed for reliable application layer multicast.

This paper proposes a tree-based ACK mechanism (called ATACK) that can be used in most reliable application layer multicast solutions, to achieve good throughput with desirable scalability. ATACK is dependent of the underlying data recovery way. Therefore it can be used as a complement to existing UDP-based reliable ALM solutions that use the ALM tree to distribute the multicast data. There have been many flow control approaches based on the tree-based ACK mechanism in the field of reliable IP multicast. Since most flow control approaches (e.g., RMTP-II and TMTP) are dependent of the underlying data delivery and loss recovery ways, these approaches can also be used in ALM environment. Therefore this paper only concentrates on the optimized tree-based ACK mechanism in tree-based ALM environment.

II. ATACK Overview

In ALM, tree is the typical data delivery structure. ATACK is designed for reliable ALM that uses tree structure to distribute the multicast data, to achieve good throughput with desirable scalability. ATACK is dependent of the underlying data recovery way and tree building approach.

ATACK groups the members into a hierarchy of logical acknowledgement domains (called ADs) by a self-organized way, and further forms the acknowledgement tree in terms of the hierarchical acknowledgement domains. The expected range of AD size promises the efficient acknowledgement and avoids adjusting ADs frequently. In ATACK, only a part of the nodes of ATACK acknowledgement tree send the ACK messages, which obviously reduces the related load and improves the acknowledgement performance. This paper consists of two main contributions: (1) a feasible way to build the self-organized acknowledgement tree for reliable ALM, and (2) an optimized approach for acknowledging the receipt of multicast data in terms of ALM's error correlation. In the following parts, we introduce the detailed design of ATACK.

III. ATACK Design

A. ATACK Acknowledgement Domains

From the root to each member, there is one unique loop-free path along the ALM tree. In HMTP [1], the member list of this path is called root path. Existing tree-based ALM solutions either use the root path or have the capacity of getting it by a simple extension. In this paper, we assume that each group member knows its root path in the delivery tree.

ATACK organizes group members into a hierarchy of logical acknowledgement domains in terms of the ALM multicast tree (The AD building procedure can be seen in following parts). In each AD, a group member is selected as the acknowledgement processor (AP) to assemble acknowledgement messages in the domain. An *i*-AD consists of its AP, some common members, and APs at level *i*+1. Except for the *i*-AP, all the nodes in an *i*-AD are also called domain members of the AD. In this paper, we say that the AP of an AD stands for the AD. Note that the *i*-AD and *i*-AP mean the AD and AP at level *i*, respectively. The root of the multicast tree is a special AP, and the AD including the root is at the highest level (i.e., level 1). Figure 1 illustrates a two-tier hierarchy of ADs. In the figure, 1-AD A is the parent domain of the 2-AD B.

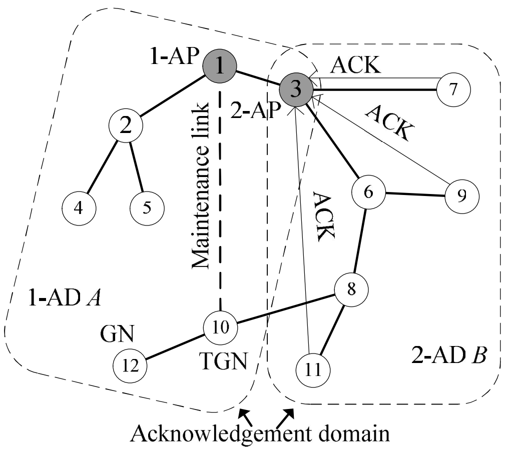

Fig. 1. The hierarchy of acknowledgement domains.

ATACK attempts to divide the multicast group into multiple ADs of size in [λ + 1, δλ + 1] and at most an AD (i.e. the AD that contains the root) of size in [1, δλ + 1], where both λ and δ (δ ≥ 2) are configuration parameters. As noted previously, the range of AD size promises the efficient acknowledgement and avoids adjusting ADs frequently. In a given AD, a domain member is called graft node (GN) if the AP of the AD is not on its root path, and a GN is called TGN if there is no GN on its root path. There is an additional connection (called maintenance link) between a TGN and the corresponding AP. In the example shown in Fig. 1, node 12 and 10 are GN and TGN, respectively.

Each AD has an acknowledgement maintenance subtree (called AMS), which consists of: 1) all domain members and the AP in the AD, 2) all maintenance links in the AD, and 3) the delivery tree edges (called tree links) that connect two different nodes in the AD. The union of all the AMSs is the acknowledgement maintenance tree (called AMT). For example, Fig. 2(b) and Fig. 2(c) show the AMS of AD A and the AMT of the whole group shown in Fig. 3. For distinguishing the AMT from the delivery tree, the parent and child of a node in the AMT are also called maintenance parent and maintenance child of the node, respectively.

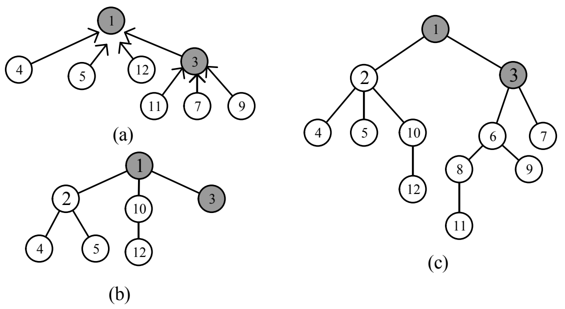

Fig. 2. The examples of the acknowledgement structure: (a) Acknowledgement tree; (b) AMS; (c) AMT.

B. ATACK Acknowledgement Tree

Figure 2(a) illustrates the acknowledgement tree corresponding to Fig. 1. As mentioned previously, the branch nodes of the AMT are group members. Only the leaf nodes of the AMSs send ACKs to the corresponding APs, because a leaf node receives a correct ADU only if all its upstream nodes in the same AD receive the correct ADU. Note that each parent-child relation between two neighbor nodes (not including the AP) in an AMS also holds in the corresponding ALM delivery tree.

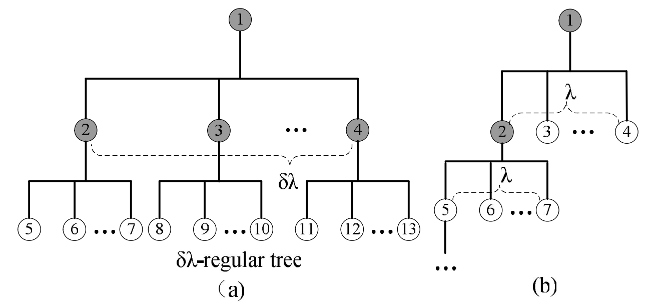

Fig. 3. Examples of the acknowledgement trees.

In this paper, a tree is said to be *k*-regular if and only if each no-leaf node of the tree has *k* children. Let *T* be an acknowledgement tree including *n* nodes, then we have Theorem 1.

Theorem 1. The minimum and maximum heights of *T* are $\lfloor \log_{\lambda\delta} n(\delta\lambda - 1) \rfloor + 1$ and $\lceil (n-1)/\lambda \rceil + 1$, respectively.

Proof. In RALM, there are many ADs of size in [λ + 1, δλ + 1] and at most an AD of size in [1, δλ + 1]. Therefore *T* has the minimum height if only an AD size is in [1, δλ + 1] and all the other AD sizes are equal to δλ + 1, as Fig. 4(a) shows. Let the minimum height of *T* be *h*, then

$$((\delta\lambda)^{h-1} - 1)/(\delta\lambda - 1) < n \le ((\delta\lambda)^{h} - 1)/(\delta\lambda - 1). \quad (1)$$

We can further have $h - 1 \le \log_{\delta\lambda} n(\delta\lambda - 1) < h$. Note that *h* is an integer. Thus $h = \lfloor \log_{\lambda\delta} n(\delta\lambda - 1) \rfloor + 1$. If only a common member node in any AD can become the AP in the lower-level AD, then *T* has the maximum height, as Fig. 4(b) explains. Similarly, we can prove that the maximum height of *T* is $\lceil (n-1)/\lambda \rceil + 1$. Thus the theorem has been proven.

Actually, the maximum height of *T* is only a theoretical threshold, because the acknowledgement tree is built in terms of the ALM tree, and the ALM protocol usually avoids producing the poor ALM tree as Fig. 3(b) illustrates. We also can deduce the following theorem.

Theorem 2. Let T_1 and T_2 be two trees that each consists of *n* nodes, the max degree of nodes of T_1 be *k*, and T_2 be *k*-regular. Then $nl(T_1) \le nl(T_2)$ holds, where $nl(T)$ means the number of leaf nodes of *T*.

Proof. We can traverse T_1 by the breadth-first search. When visiting a node *b* in the search procedure, do as follows: if 1) the number of children of *b* is less than *k*, and 2) there are some leaf nodes which are not at the lowest two levels, one leaf node at the lowest level becomes a child of *b* and is marked with symbol UN. By the above transition procedure, T_1 will become *k*-regular finally. We can notice that the number of leaf nodes is decreased if a node, with the UN mark, becomes a branch node. Thus the theorem has been proven.

We can easily deduce that there are only (*kn*-*n* + 1)/*k* leaf nodes in the k-regular tree including *n* nodes. According to Theorem 2 and the above conclusion, it is safe to say that the number of receivers that need to send ACKs is apparently reduced in ATACK.

C. ATACK AD Formation

We use notation *dn*(*m*) to mean the number of member *m*'s downstream nodes in the AMS of the AD that contains the domain member *m*. For the example shown in Fig. 2©, *dn*(10) and *dn*(2) are 1 and 2, respectively. Particularly, the *dn* value of leaf node of an AMS is 0. Each maintenance child (denoted by *c*) of member *m* records a *cn*(*c*) value, where $cn(c) = \overline{dn(c)} + 1$. Notation $\overline{dn(c)}$ means a proximate *dn*(*c*), $\overline{dn(c)} = \sum_{i=1}^{d} cn(c_i)$, where c_i ($1 \le i \le d$)is a maintenance child of *m* and *d* is the number of maintenance children of *m*. We will further explain $\overline{dn}$ in the next part.

The AD formation procedure is progressively finished as new members join the group. When a new member joins a group, a NEW message is transmitted one by one from its parent node to the root along its root path. When a node receives the NEW message, the corresponding *cn* value is increased by 1. Additionally, the extended refresh procedure updates the states of nodes of the AMT. Therefore each member (denoted by *m*) can get the proximate *dn*(*m*) (i.e., $\overline{dn(m)}$. In the steady group, $\overline{dn(m)}$ is equal to *dn*(*m*). However, it is difficult (even impossible) for member *m* to get the instant *dn*(*m*) in ALM environment because of the dynamics of members, i.e., $\overline{dn(m)}$ might be unequal to *dn*(*m*) in the unsteady group. Note that the $\overline{dn}$ value of an AP implies the proximate size of the corresponding AD that the AP stands for. The root is the first AP, and initial AD only contains the root. When a member joins a group, it contacts the last AP, denoted by *a*, on its root path to try to join the AD *A* that *a* stands for. Let *i* mean the level of the AD *A* and AP *a*. If the proximate size (i.e., *dc*(*a*) + 1) of *A* is larger than δλ + 1, *A* is divided into two ADs as the following rules:

Vertical split rule: If AP *a* has a child *c* such that $\overline{dn(c)} > \lfloor \delta\lambda/2 \rfloor$ holds, a breadth-first search is performed along the AMS of AD *A*. Once finding a node *n* such that $\overline{dn}(n) \le \lfloor \delta\lambda/2 \rfloor$ holds, the search procedure ends, and the maintenance parent of the node becomes a new *i* + 1-AP. Note also that all the downstream nodes of the *i* + 1-AP in the AMS become domain members of the new AD that the *i* + 1-AP stands for.

Horizontal split rule: Assume that AP *a* has *x* maintenance children. If vertical spilt rule fails to implement the partition and $x \ge 3$, AD *A* will be divided in terms of the horizontal split rule. In the rule, the children of *a* is divided into two groups-G_1 and G_2 ($\sum_{i \in G_1} cn(i) \ge \sum_{j \in G_2} cn(j)$), such that

$$\sum_{i \in G_1} cn(i) \ge \lfloor \delta\lambda/2 \rfloor + 1 \quad (2)$$

and

$$\forall q \in G_1, \quad (3)$$

$$\sum_{i \in G_1 - \{q\}} cn(i) < \lfloor \delta\lambda / 2 \rfloor + 1. \quad (4)$$

Then a random node in G1 is selected as a new i + 1-AP, and maintenance links are created between the i + 1-AP and other nodes in G1.

Graft rule: Assume that the above two rules divide AD A into A_1 and A_2, A_1 contains the new i + 1-AP, and A_2 contains a. If a is not the root and the proximate size of A_2 is less than $\lambda + 1$, one or more nodes of A_2 are designated as the TGNs in A_1, such that the proximate sizes of A_1 and A_2 are both within the expected range (i.e., $[\lambda + 1, \delta\lambda + 1]$). Note that: 1) the number of TGNs should be as small as possible, and 2) all downstream nodes of TGNs in original AD A are GNs in A_1.

Clearly, even partition is helpful for each divided AD to contain as more potential domain members as possible. In addition, even partition can alleviate the negative impact caused by the change of the delivery tree. We can see that the above partitioning procedure is a tradeoff between the capability of containing more potential domain members and the cost of building the ADs.

Assume that the proximate size of an i - AD ($i > 1$) B is less than $\lambda + 1$, the AD is incorporated with its parent AD C, and the AP of C becomes the AP of the new extended AD. If the proximate size of the new AD exceeds $\delta\lambda + 1$, the above AD spilt procedure will be performed. Each AP (denoted by a) periodically checks if $\overline{dn(a)}$ is out of the expected range. If $\overline{dn(a)}$ is larger than $\delta\lambda + 1$, the vertical split rule is first used to divide the corresponding AD. If the above procedure fails, horizontal split rule is used to implement the partition. The graft rule is employed as a possible supplement to the other two rules. This paper does not introduce more trivial details of implementations of the above procedures.

D. ATACK AD Maintenances

Similar to the root path, from the root to each member, there is one unique loop-free path along the AMT. The member list of this path is called acknowledgement root path (ARP). Note that the ARP is labeled by a special mark for distinguishing itself from the root path.

In the ALM tree, the states of members are refreshed by periodic message exchanges between neighbors. For the example of HMTP, each child periodically sends a REFRESH message to its parent, and the parent replies by sending back a PATH message. For simultaneously refreshing the states of members in the AMT and delivery tree, ATACK extends the above REFRESH message. The extended message is denoted by REFRESH_E(s,t, $\overline{dn(s)}$, f), where s and t indicate the sender and receiver of the message respectively, and f ($f \in \{DO, DA, DOA\}$) means a type flag. The extended refresh procedure works as follows:

1. If the maintenance parent mp and parent p of a member m (other than the root) are not the same node, m periodically sends REFRESH_E(m,p,0,DA) to its parent p, and sends another refreshing message (REFRESH_E(m,mp, $\overline{dn(m)}$,DO)) to its maintenance parent mp. Otherwise, m periodically sends REFRESH_E(m,mp, $\overline{dn(m)}$,DOA) to p(p=mp);
2. Once receiving REFRESH_E(m,p,0,DA), p sends back the APATH message that contains p's root path. Note that the APATH message is similar to the PATH message. When mp receives REFRESH_E(m,mp,,DO), it updates the corresponding $cn(m)$ value in terms of $\overline{dn(m)}$, and sends the APATH message that contains its ARP to member m. If p receives REFRESH_E(m,p, $\overline{dn(m)}$,DOA), it also updates the corresponding $cn(m)$ value, and sends back the APATH message, with its ARP and root path. Member m updates the related information (ARP or root path) when it receives the APATH message.

Each leaf node of an AMS and the corresponding AP should periodically exchange LIVE_A messages to keep them active. If the leaf node m of an AMS finds that the AP does not work, it sends NEWAP_R messages to find the active downstream node u of the AP in m's ARP, such that u is closest to the root among all the active downstream nodes of the AP on m's ARP. Then u contacts the closest active AP on its ARP, and becomes a member of the AD that the found AP stands for. The above procedure is called AP recovery. Additionally, some messages are exchanged for informing m of the new AP selection. If there is no node that satisfies the above conditions, m initiates the AP recovery procedure. Similarly, m initiates the above AP recovery procedure if it receives no notice from a new AP in a long time. When an AP finds that some leaf node has left the corresponding AD, it only deletes the correlative information.

In a given AD, when a TGN leaves the group session, its children (GNs) in the AD become TGNs. If a member switches to a new parent in the tree-rebuilding and maintenance procedures, it tells its maintenance parent of this, and the AMT will make some correlative adjustments in terms of the new delivery tree. The above two procedures can be implemented easily. This paper does not discuss more detail on the AD maintenances.

IV. EXPERIMENTAL RESULTS

We used the BRITE Generator to generate a 5000-node graph as the underlying network topology. Additionally, we generated 1000 nodes as member hosts (receivers) and a node as the multicast source. Each member node was connected to a random router node. The fanout (i.e. node degree) value of each member node was between 2 and 4, and the expected AD size was within the range from 6 to 18.

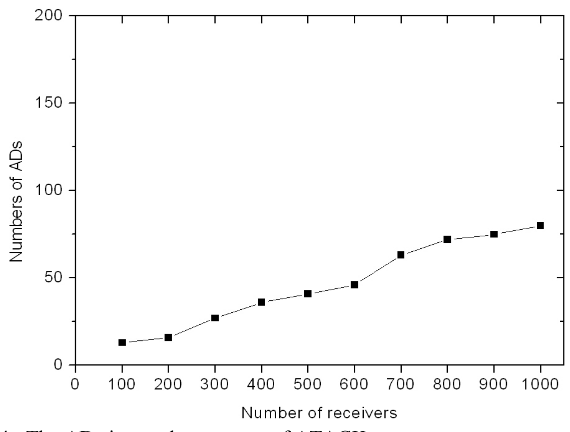

Fig. 4. The ADs in steady structure of ATACK.

We used HMTP protocol to build ALM delivery trees, and simulated ATACK based on the HMTP trees. We simulated HMTP and ATACK with the network simulator NS-2.

Figure 4–Fig. 6 give some features of the stable structure of corresponding acknowledgement trees. In Fig. 6, AN means the node that needs to send ACK messages. From Fig. 4, we can see that about 9 % group members become AD nodes. Figure 5 tells us that the sizes of ADs are within the expected range. The above two figures show that the AD building approach of ATACK has an expected performance in the steady cases. From Fig. 6, we can notice that the number of receivers which send ACK messages is obviously reduced, which means that ATACK has high effectiveness of acknowledging the received packets.

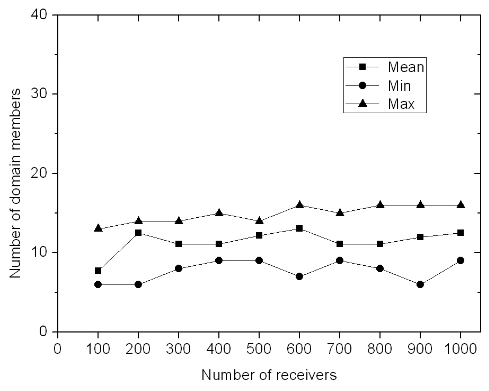

Fig. 5. The domain members in steady structure of ATACK.

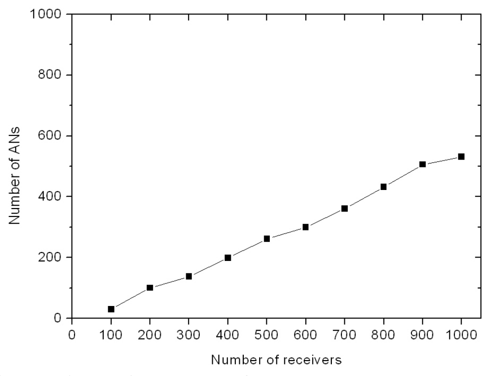

Fig. 6. The ANs in steady structure of ATACK.

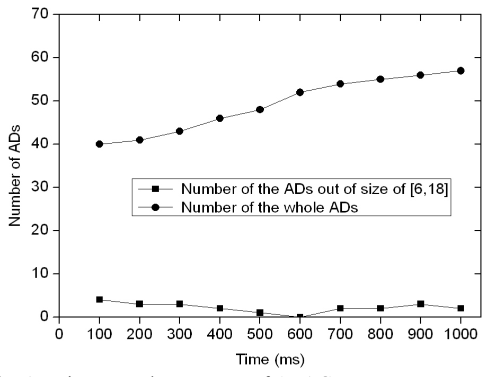

Fig. 7. The ADs in unsteady structure of ATACK.

Figure 7 shows the numbers of the whole ADs and the ADs whose size is out of the range from 6 to 18 in an unsteady case. In this experiment, we first got a steady multicast tree, including 500 receivers. Then each existing receiver left the group with probability of $e^{-0.05t}$. Additionally, another 100 receivers each joined the group at a random moment between 1th ms and 1000th ms. The time interval of sending REFRESH_E messages was 3 ms. From the results shown in the figure, we can see that only a small amount of ADs are out of the excepted AD size (e.g., from 6 to 18), which shows that ATACK's AD formation and related maintenance procedures work well in the unsteady group.

V. CONCLUSIONS

This paper presented a tree-based ACK mechanism ATACK for reliable application layer multicast. ATACK is dependent of the underlying data recovery way and tree building approach. ATACK employs a self-organized way to group the members into a hierarchy of logical acknowledgement domains, which each have an acknowledgement processor. ATACK attempts to confine the size of an acknowledgement domain into some expected range, which promises the efficient acknowledgement and avoids adjusting acknowledgement domains frequently. ATACK employs the acknowledgement processors to aggregate ACK messages from other nodes in the same domain. In a designed domain, only the leaf nodes of the AMS send ACKs to the corresponding acknowledgement processor, because a leaf node receives a correct packet only if all its upstream nodes in the AMS receive the same correct packet. The analysis and experiments show that ATACK has desirable acknowledgement performance.

REFERENCES

[1] B. Zhang, S. Jamin, L. Zhang, “Host multicast: A framework for delivering multicast to end users”, in *Proc. of IEEE INFORCOM*, pp. 1366–1375, 2002.

[2] D. Pendarakis, S. Shi, D. Verma, M. Waldvogel, “ALMI: an application level multicast infrastructure”, in *Proc. of 3rd Usenix Symposium on Internet Technologies & Systems*, pp. 49–60, 2001.

[3] F. Wang, Y.Q. Xiong, J.C. Liu, “mTreebone: A collaborative tree-mesh overlay network for multicast video streaming”, *IEEE Trans. Parallel and Distributed Systems*, vol. 21, no. 3, pp. 379–392, 2006. [Online]. Available: http://dx.doi.org/10.1109/TPDS.2009.77

[4] Jijun Cao, Jing Xie, Feng Chen, “DSD-D: A distributed algorithm for constructing high-stability application-layer multicast tree”, in *Proc. of Fifth Int. Conf. on frontier of computer science and technology*, pp. 122–128, 2010.

[5] T. Strufe, S. Günter, A. Chang, “BCBS: An efficient load balancing strategy for cooperative overlay live-streaming”, in *Proc. of IEEE ICC*, pp. 304–309, 2006.

[6] X.L. Li, A.D. Striegel, “A case for passive application layer multicast”, *Computer Networks*, vol. 51, no. 11, pp. 3157–3171, 2007. [Online]. Available: http://dx.doi.org/10.1016/j.comnet.2007.01.016

[7] L. Dan, W. Jianping, L. Jiangchuan, C. Yong, X. Ke, “Defending against distance cheating in link-weighted application-layer multicast”, *IEEE/ACM Trans. Networking*, vol. 19, no. 5, pp. 1448–1457, 2010.

[8] X. Jin, W.–P.K. Yiu, S.–H.G.Chan, “Loss recovery in application-layer multicast”, *IEEE Multimedia*, vol. 15, no. 1, pp. 18–27, 2008. [Online]. Available: http://dx.doi.org/10.1109/MMUL.2008.14

[9] K.-F. S. Wong, S.-H. G. Chan, Wan-Ching Wong, Qian Zhang, Wen-Wu Zhu, Ya-Qin Zhang, “Lateral error recovery for application-level multicast”, *IEEE Trans. on multimedia*, vol.8, no.6, pp. 219–230, 2006.

[10] B. N. Levine, S. Paul, J. J. Garcia-Luna-Aceves, “Organizing multicast receivers deterministically by parcket-loss correlation”, *Multimedia Systems*, vol. 9, no. 1, pp. 3–14, 2003. [Online]. Available: http://dx.doi.org/10.1007/s00530-003-0050-2

[11] B. Whetten, G. Taskale, “An overview of reliable multicast transport protocol II”, *IEEE Network*, vol.4, no.1, pp. 37–47, 2000. [Online]. Available: http://dx.doi.org/10.1109/65.819170

[12] Y. Wonyong, L. Dongman, hy Youn, S. Lee, “A combined group/tree approach for scalable many-to-many reliable multicast”, *Computer Communications*, vol. 29, no. 18, pp. 3863–3876, 2006. [Online]. Available: http://dx.doi.org/10.1016/j.comcom.2006.06.011