

# Multi-domain Topology-Aware Grouping for Application-Layer Multicast\*

Jianqun Cui<sup>1</sup>, Yanxiang He<sup>2</sup>, Libing Wu<sup>2</sup>, Naixue Xiong<sup>3</sup>,  
Hui Jin<sup>2</sup>, and Laurence T. Yang<sup>4</sup>

<sup>1</sup> Department of Computer Science, Huazhong Normal University, Wuhan, 430079, China

<sup>2</sup> School of Computer, Wuhan University, Wuhan, 430072, China

<sup>3</sup> Information Science School, Japan Advanced Institute of Science and Technology, Japan

<sup>4</sup> Department of Computer Science, St. Francis Xavier University, Canada  
jqcui@126.com, {yxhe,wu}@whu.edu.cn, naixue@jaist.ac.jp,  
hjin6@iit.edu, ltyang@stfx.ca

**Abstract.** Application-layer multicast (ALM) can solve most of the problems of IP-based multicast. Topology-aware approach of ALM is more attractive because it exploits underlying network topology data to construct multicast overlay networks. In this paper, a novel mechanism of overlay construction called Multi-domain Topology-Aware Grouping (MTAG) is introduced. MTAG manages nodes in the same domain by a special node named domain manager. It can save the time used to discover topology information and execute the path matching algorithm if there are some multicast members in the same domain. The mechanism can lower the depth of the multicast tree too. Simulation results show that nodes can acquire multicast service more quickly when the group size is large or the percentage of subnet nodes is high.

## 1 Introduction

IP Multicasting [1] implemented at the network layer can provide an efficient delivery service for multiparty communications. It is the most efficient way to perform group data distribution, as it eliminates traffic redundancy and improves bandwidth utilization on the wide-area network. However, although IP Multicasting has been available for many years, today's Internet service providers are still reluctant to provide a wide-area multicast routing service due to some reasons such as forwarding state scalability, full router dependence and so on [2, 15]. In order to achieve global multicast, application-layer multicast (ALM) has recently been proposed, where the group members form an overlay network and data packets are relayed from one member to another via unicast. ALM shifts multicast support from core routers to end systems.

Application-layer multicast can solve most of the problems of IP-based multicast. It can easily be deployed because it does not require any modification to the current Internet infrastructure. However, application-layer overlay multicast technology is not

---

\* Supported by the Important Research Plan of National Natural Science Foundation of China (No. 90104005).

as efficient as IP multicasting. It will incur some delay and bandwidth penalties and less stability of multicast tree. Till now, many ALM protocols [3-9] have been proposed to utilize overlay technique to provide scalable and high quality multicast service to applications. Among these protocols, topology-aware approach is more attractive because it exploits underlying network topology data to construct multicast overlay networks. The constructed tree has low relative delay penalty and a limited number of identical copies of a packet on the same link. TAG (Topology-Aware Grouping) [10, 11] is a typical ALM protocol of this approach. Experiments show that TAG is efficient in reducing delays and duplicating packets with reasonable time and space complexities.

But TAG considers the Internet as a non-domain network. Each node in the multicast tree executes the same algorithms to join or leave the tree. In fact, the Internet is a multi-domain network. Here domain can be considered as an autonomous system or a subnet. Millions of self-similar autonomous systems or subnets form the Internet. TAG will construct a high-depth multicast tree when the multi-domain feature of the Internet is ignored. For example, if there are 10 multicast members in the same domain, the last join node who wants to acquire the multicast data has to wait for the other 9 nodes' forwarding. It will cost a lot and the node will spend more time waiting for the multicast service. Therefore a new mechanism of the overlay construction based on TAG, called Multi-domain Topology-Aware Grouping (MTAG), is introduced in this paper. The mechanism takes the multi-domain feature into account to provide more quickly service for multicast members.

The paper is structured as follows. Section 2 provides an overview of related work and section 3 introduces the proposed MTAG mechanism. Performance of the mechanism is verified by simulation in section 4. Finally, we conclude the paper with a summary of conclusion and future work.

## 2 Related Work

Many application-layer multicast solutions have been put forward with their respective strengths and weaknesses [12]. Several work propose overlay construction schemes that take the topology of the physical network into account. In [13], a distributed binning mechanism is proposed where overlay nodes partition themselves into bins such that nodes that fall within a given bin are relatively close to one another in terms of network latency. In TAG [10], the information about overlap in routes to the sender among group members is used to guide the construction of overlay tree. TAG selects the node that shares the longest common *spath* prefixes, subject to bandwidth, degree, and possibly delay constraints as parent for a new node. The *spath* of a node refers to a sequence of routers comprising the shortest path from source node to this node according to the underlying routing protocol. The resulting tree of TAG has low relative delay penalty, and introduces a limited number of identical packets on the same link.

TAG considers that all nodes in the multicast tree have no difference. However, the Internet is connected with many domains. Most multicast applications have a characteristic that there are more than one multicast members in the same domain because the spread of a new multicast application is often related to geographic

location. Users who live in the same building or work at the same company have more chances to enjoy the same multicast service. In TAG, supposing there are 10 nodes in the same domain, we can find that the next node out of this domain will have the depth more than 10 even if it is very close to the source node. The multicast tree can not match the real topology very well. For this purpose, our MTAG mechanism modifies the overlay construction of TAG and provides a domain manager to manage the nodes in the same domain. The mechanism can lower the average depth of the whole multicast tree and provide more quickly service for multicast members.

### 3 Multi-domain Topology-Aware Grouping

#### 3.1 TAG Review

Let us review the overlay construction process of TAG before our mechanism is introduced. To simplify the operation, we adopt complete the path matching algorithm to construct the multicast tree.

Fig. 1 shows a multicast tree produced by member join algorithm of TAG. We use a more complex example than [10] so that you can find the disadvantages of TAG.

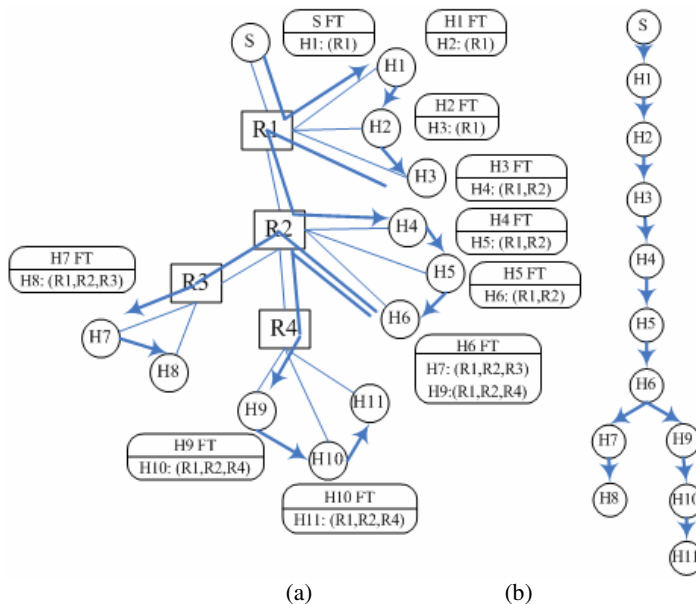


Fig. 1. Multicast overlay constructed by TAG

Source S is the root of the multicast tree, R1 through R4 are routers, and H1 through H11 are hosts in Fig. 1. Thick solid lines denote the current data delivery tree. Each node of the multicast tree maintains a family table (FT) defining parent-child relationship for this node (showing only the children in Fig. 1).

Fig. 1(a) shows the real topology and FT of each non-leaf node. The new node who wants to join the multicast tree will send a JOIN message to S first. When S receives the message, it computes the *spath* to the new node by network path-finding tools such as *traceroute*. After S gets the *spath* of the new node, it starts the path matching algorithm execution. The path matching algorithm will be executed by next nodes until the best parent is found. The details of the algorithm can be found in [10]. In Fig. 1(b), we ignore the topology and other information so that we can get a clear look on the multicast tree. There are just 3 routers from the source node to the node H11, while the depth of the node H11 has already been 9. The data forwarding latency will increase with the depth's increase. Another shortcoming of TAG is that it must compute the same *spath* for many times if there are some nodes (such as H9, H10 and H11) in the same domain. The topology discovery procedure will cost a lot and lead to poor performance.

### 3.2 Mechanism of MTAG

To overcome the weaknesses of TAG mentioned above, we propose another mechanism MTAG to construct and maintain the multicast overlay. We introduce a new role named domain manager to manage nodes in the same domain. The source node will maintain another table called domain table (DT) to save the information of domain managers and the DT is empty at the very beginning. To implement the new mechanism, we add the node's path information into its FT (the old FT just saves the information of its parent and its children).

Once a new node sends a JOIN message to the source node, the source node will get the domain information of the node first. How can we know which domain the node belongs to? We can identify the node's domain by the network number of its IP address which can be obtained from the JOIN message if a subnet is regarded as a domain. Otherwise, if there are some other rules to distinguish domains, we need to add the domain information into the JOIN message. Here, for simplicity, we think a domain is a subnet.

Then the source node checks the DT to make sure whether the domain manager of the new node exists. If the domain manager is found, the source node will directly send a CHILD message with the node's information to the domain manager. The domain manager takes the new node as its child and sends a PARENT message to the node. The domain manager provides the multicast service for the node at the same time. What the new node should do is to add the parent information into its FT table when it receives the PARENT message. In another way, if the node's domain manager does not exist, the source node will add the node's information into the DT and send a MANAGER message to the node. Then the source node starts the topology discovery procedure and the path matching algorithm similar to TAG to find the best parent of the node. The node who receives a MANAGER message will identify itself as the domain manager.

A normal member who wants to leave the multicast tree will execute the same algorithm as TAG. The node just needs to send a LEAVE message with its FT to the parent. The parent will remove the node from its FT and add FT entries for the children of the leaving node. If the domain manager wants to leave the tree, it should send a LEAVE message to the source node in addition to the parent. The LEAVE

message includes its FT (if its children are all in other domains) or the new domain manager candidate chosen from its children (if it has any children in the domain). If the parent receives the LEAVE message with the FT, it will execute the same algorithm with TAG, otherwise it will replace the node information with the candidate in its FT. The source node will update or delete the node information from the DT based on the information of the LEAVE message. The last step is the old domain manager sends an UPDATE message to inform its children who is the new domain manager. The new domain manager takes the other nodes as its children and provides the multicast service right now when it receives the UPDATE message. Other children just need to update their parents of the FT. The member leaving process of the domain manager seems more complex than TAG. However, one domain just has one domain manager, so the probability is small.

The mechanism of MTAG is illustrated in Fig. 2.

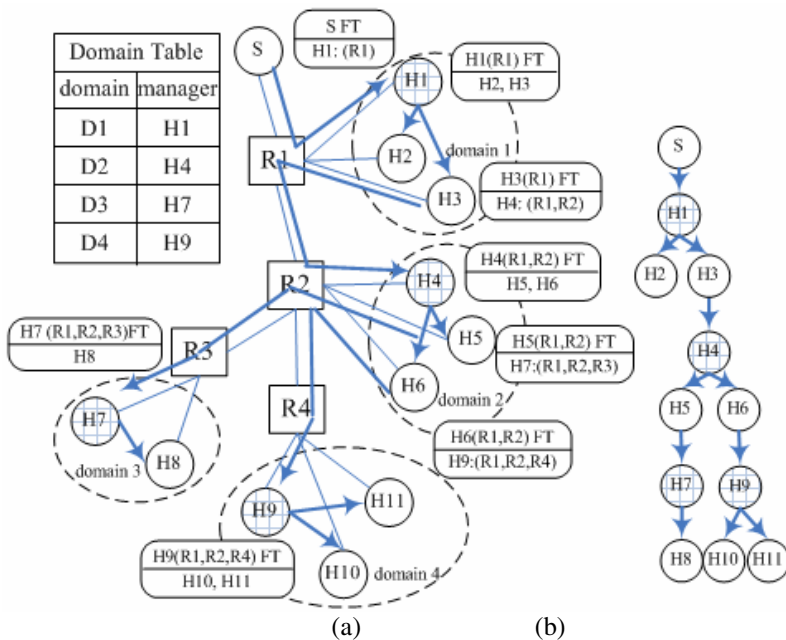


Fig. 2. Multicast overlay constructed by MTAG

We can get a number of advantages from MTAG:

**1. The topology discovery and path matching algorithm are not necessary if the domain manager exist:** Just when the first node in the domain joins the multicast tree, the two steps (cost a lot) need to be executed. This mechanism will shorten the waiting time of the new node if it is not the first member in the domain.

**2. MTAG can lower the average depth of the multicast tree:** From Fig. 2(b), we can find the depth of the multicast tree is 6 instead of 9 (Fig. 1(b)) under the same topology. The reason is that nodes in the same domain all take the domain manager as their parent. The domain manager can afford so many children because the bandwidth

is often high enough in a LAN. The lower the depth of the tree is, the sooner the node acquires the multicast data.

**3. MTAG can balance the traffic of the domain:** In TAG, the node chosen to be the parent of the next node (out of the domain) must be the last join node. The node H6 in Fig. 1 is such a node. However, in MTAG, the domain manager has more than one child, so that it can choose randomly its children to be the parent of the next node. We can find the node H5 and H6 in Fig. 2 act as the different nodes' parents respectively. We should not let one node be the parent because the connection is inter-domain.

**4. The domain manager does not need to save the *spath* of its children:** Nodes in the same domain have the same *spath* so that the domain manager just needs to record its own *spath* instead of each child's *spath*. The domain manager can use less memory to store the FT.

Compared with TAG, MTAG has the only obvious drawback: The source node need to maintain the DT. In MTAG the source node just needs to forward the multicast data to one child which should be very close to it. Another task of the source node is to process JOIN/LEAVE messages. The source node just needs to update its DT in most cases because the domain manager has shared some management tasks for the source node. So the maintenance of the DT will not significantly affect the performance of MTAG.

### 3.3 Member Join/Leave Algorithm

Three algorithms are needed to implement the MTAG mechanism. Member\_Joining(S) executed on the new node is used for joining the multicast tree. Member\_Leaving (S) is executed when the node wants to leave the tree. Members of the multicast tree execute the third algorithm Member\_executing() as a daemon.

**Algorithm 1: Member\_Joining (S)**

```
send(S, JOIN_msg);
msg = receive_msg() //receive_msg() will be blocked
till a message is received or timeout
If (isPARENT_msg(msg))
    add_FT_parent(msg.address);
Else If (isMANAGER_msg(msg))
    isManager = true;
Endif
```

**Algorithm 2: Member\_Leaving (S)**

```
if (isManager)
    newManager = getDomainChild(FT);
    If (newManager == NULL)
        LEAVE_msg = new_LEAVE_Msg(FT);
    Else
        LEAVE_msg = new_LEAVE_Msg(newManager);
    Endif
    send(S, LEAVE_msg);
Endif
send(FT.parent, LEAVE_msg);
```

**Algorithm 3: Member\_executing( )**

```

Do while true
  msg = receive_msg();
  If (isJOIN_msg(msg)) //just the source node can
  receive the JOIN message
    manager = checkInDT(msg.domain);
    If (manager == NULL) //there are not any
    nodes in the domain
      updatedT(msg.domain, msg.sourceAddr);
      send(msg.sourceAddr, MANAGER_msg);
      spath = topo_discovery(msg.sourceAddr);
      PathMatch(myself, spath); //execute path
      matching algorithm of TAG
    Else
      send(manager, CHILD_msg);
    Endif
  Else
    If (isCHILD_msg(msg))
      add_FT_child(msg.sourceAddr);
      send(msg.sourceAddr, PARENT_msg);
      send(msg.sourceAddr, multicast_data);
    Else
      If (isLEAVE_msg(msg))
        If (msg.newManager == NULL)
          If (isSourceNode(myself))
            deleteFromDT(msg.domain);
          Else //other nodes
            update_FT_child(msg.FT.children);
          Endif
        Else //has new domain manager
          If (isSourceNode(myself))
            updateFromDT(msg.domain, msg.newManager);
          Else //other nodes
            update_FT_child(msg.sourceAddr,
            msg.newManager);
          Endif
        Endif
      Else
        If (isUPDATE_msg(msg))
          If (msg.newManager == myself)
            add_FT_child(msg.FT.children);
            send(msg.FT.children, multicast_data);
          Else
            update_FT_parent(msg.newManager);
          Endif
        Endif
      Endif
    Endif
  Endif
Endif
Enddo

```

## 4 Evaluation

In this section, we present the simulation results to evaluate the proposed MTAG mechanism and compare it to the original TAG protocol.

### 4.1 Simulation Environment

We generate a Transit-Stub network topology with 500 nodes using GT-ITM [14]. On top of the physical network, the multicast group members are attached to the stub nodes. Nodes belong to the same stub are thought to be in the same subnet. We can control the percentage of nodes in the same subnet. The multicast group size ranges from 50 to 500 members. The transit-to-transit, transit-to-stub, and stub-to-stub link bandwidth are assigned between 10 Mbps and 100 Mbps. The links from edge routers to end systems have the bandwidth between 1 Mbps and 10 Mbps.

### 4.2 Performance Metrics

A simulation is carried out to contrast the performance of our proposed MTAG and TAG. Because the main idea of the overlay construction has not been changed by MTAG, the mean RDP (Relative Delay Penalty) and Link Stress are same as TAG. So we will not evaluate these two performance metrics which can be found in [10]. The performance metrics used for our experiments are Member Join Latency (MJL) and Data Acquisition Latency (DAL).

Member Join Latency denotes how long it takes for the join operation. If  $T_{JOIN}$  denotes the time when the new node sends the first JOIN message and  $T_{data}$  denotes the time when it receives the first packet of multicast data from the parent, the value of the MJL can be measured by  $T_{data} - T_{JOIN}$ . We report the mean of the latency of all  $n$  members:

$$\text{Mean Member Join Latency} = \frac{1}{n} \sum_{i=1}^n (T_{data_i} - T_{JOIN_i}) \quad (1)$$

Data Acquisition Latency of a node is an expression of how much time it takes for a packet of multicast data to get from the source node to this node. To get the DAL of each node, we add the additional mechanism into all nodes. It is not necessary for ALM. The source node sends a PROBE message (very small) to its children in a certain interval. The node who receives the message will send a HELLO message to its parent first and forward the PROBE message to its children at the same time. So the PROBE message can be received by all nodes in the multicast tree. Because the path of the round trip is same, the source node can calculate approximately the DAL by  $RTT/2$ . The source node will probe each node for  $m$  times to get the average value. We also use the mean of the DAL to evaluate the performance:

$$\text{Mean Data Acquisition Latency} = \frac{1}{n} \sum_{i=1}^n \left( \frac{1}{m} \sum_{j=1}^m (RTT_{ij} / 2) \right) \quad (2)$$

### 4.3 Results and Analysis

We will evaluate the two metrics in two different conditions. We attach the multicast group members to the stub nodes randomly and adjust the multicast group size from 50 to 500 in the first experiment. The relationship between the performance metrics and the group size is shown in Fig. 3.

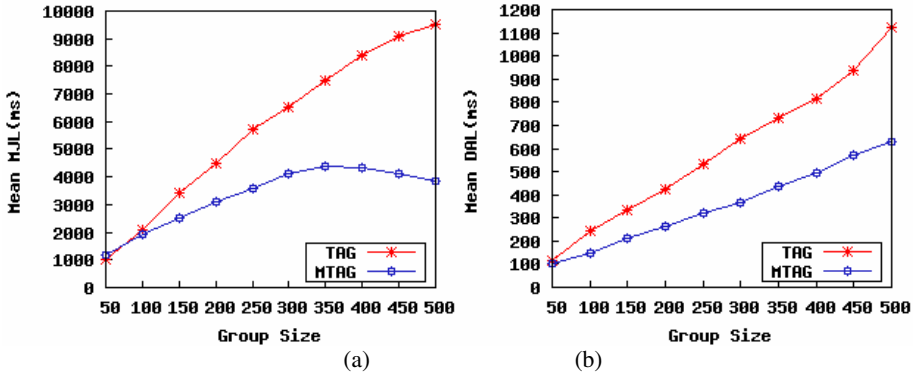


Fig. 3. Performance metrics versus group size

We find the mean MJL of TAG become higher when the group size increases from the experimental data showed in Fig. 3 (a). However, MTAG can get lower MJL when the group size reaches a certain number. The reason is that we have more nodes in one subnet when the group size is large. MTAG will save more join time if there are more nodes in a subnet.

Fig. 3 (b) shows that the mean DAL will increase with the group size in both mechanisms. We also find the mean DAL of MTAG is much less than that of TAG. MTAG can build a multicast tree with lower depth if there are some nodes in the same domain. The low depth multicast tree will shorten the latency of data forwarding.

In the second experiment, we give a more obvious percentage of the nodes in the same subnet and fix the group size of 300. For example, 20% nodes in the same subnet means there are 20% nodes in such subnet where there are two or more member nodes. These 20% nodes needn't be in one subnet. Considering the size of stub nodes and member nodes, the percentage ranges from 20% to 100%. The result is shown in Fig. 4.

Fig. 4 shows that the percentage of the nodes in the same subnet has little effect on TAG's performance while the mean MJL and DAL of MTAG decrease sharply when the percentage increases.

It can be concluded from the simulation results that MTAG can decrease the mean MJL and DAL than TAG when the group size is large or the percentage of the subnet nodes is high.

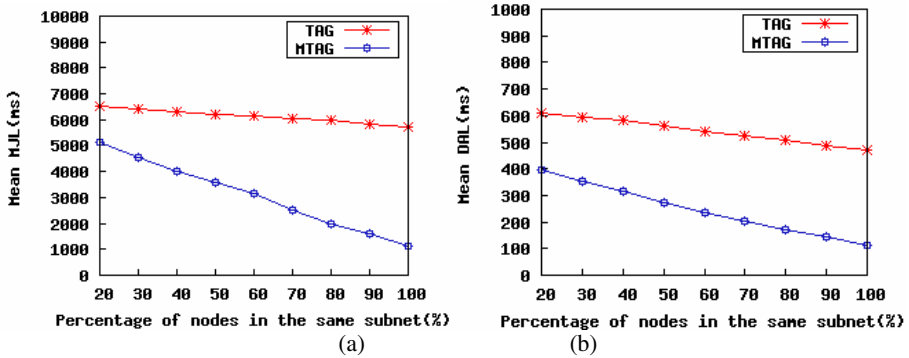


Fig. 4. Performance metrics versus percentage of nodes in the same subnet

## 5 Conclusion and Future Work

In this paper we have proposed a novel overlay construction mechanism MTAG based on the topology-aware application-layer multicast. We introduce the domain manager to manage nodes in the same domain. The mechanism can shorten the join time of a new node and lower the depth of the multicast tree. The results show that the new nodes can acquire the multicast service more quickly when the group size is large or percentage of the subnet nodes is high.

For simplicity, we do not take the bandwidth into account when we use the path matching algorithm of TAG. The topology of simulation environment is not complex and sweeping enough. Future work will be done on testing and optimizing MTAG in a more complex environment.

## References

1. Kosiur, D.: IP Multicasting: The Complete Guide to Interactive Corporate Networks. John Wiley & Sons, Inc. Chichester (1998)
2. Chu, Y., et al.: A Case for End System Multicast. IEEE Journal on Selected Areas in Communication (JSAC), Special Issue on Networking Support for Multicast (2002)
3. Chawathe, Y.: Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service. Ph.D. Thesis, University of California, Berkeley (2000)
4. Chu, Y.-H., Rao, S.G., Zhang, H.: A Case for End System Multicast. In: Proceedings of ACM SIGMETRICS, ACM Press, New York (2000)
5. Francis, P.: Yoid: Extending the Multicast Internet Architecture, White paper (1999), <http://www.aciri.org/yoid/>
6. Jannotti, J., Gifford, D., Johnson, K., Kaashoek, M., O'Toole, J.: Overcast: Reliable Multicasting with an Overlay Network. In: Proceedings of the 4th Symposium on Operating Systems Design and Implementation (2000)
7. Pendarakis, D., Shi, S., Verma, D., Waldvogel, M.: ALMI: An Application Level Multicast Infrastructure. In: Proceedings of 3rd Usenix Symposium on Internet Technologies & Systems (2001)

8. Ratnasamy, S., Handley, M., Karp, R., Shenker, S.: Application-level multicast using content-addressable networks. In: Proceedings of 3rd International Workshop on Networked Group Communication (2001)
9. Zhuang, S.Q., Zhao, B.Y., Joseph, A.D., Katz, R., Kubiawicz, J.: Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In: NOSSDAV 2001 (2001)
10. Kwon, M., Fahmy, S.: Path-aware overlay multicast, *Computer Networks. The International Journal of Computer and Telecommunications Networking* 47(1), 23–45 (2005)
11. Kwon, M., Fahmy, S.: Topology-Aware Overlay Networks for Group Communication. In: Proc. of ACM NOSSDAV, pp. 127–136. ACM Press, New York (2002)
12. Yeo, C.K., Lee, B.S., Er, M.H.A.: Survey of application level multicast techniques. *Computer Communications*, 1547–1568 (2004)
13. Ratnasmy, S., et al.: Topology-Aware Overlay Construction and Server Selection. In: IEEE INFOCOM 2002 (2002)
14. Doar, M.: A better model for generating test networks. In: GLOBECOM 1996, London, UK, pp. 83–96. IEEE Computer Society Press, Los Alamitos (1996)
15. Xiong, N., Défago, X., Jia, X., Yang, Y., He, Y.: Design and Analysis of a Self-tuning Proportional and Integral Controller for Active Queue Management Routers to Support TCP Flows. In: IEEE Infocomm 2006, Barcelona, Spain (April 23–29, 2006)