LINK-AWARE NICE APPLICATION LEVEL MULTICAST PROTOCOL

Dina Helal[1], Amr Naser[1], Mohamed Rehan[1], and Ayman El Naggar[2]
[1] Intel Labs, Cairo, Egypt

[2] Faculty of Computer Science and Information Technology, German University in Cairo, Cairo, Egypt

ABSTRACT

Multicast is one of the most efficient ways to distribute data to multiple users. There are different types of Multicast such as IP Multicast, Overlay Multicast, and Application Layer Multicast (ALM). In this paper, we present a link-aware Application Layer (ALM) Multicast algorithm. Our proposed algorithm, Link Aware-NICE (LA-NICE) [1], is an enhanced version of the NICE protocol [2]. LA-NICE protocol uses the variations of bandwidth or capacity in communication links to improve multicast message delivery and minimize end-to-end delay. OMNeT++ simulation framework [3] was used to evaluate LA-NICE. The evaluation is done through a comparison between LA-NICE and NICE. The simulation results showed that LA-NICE produces an increased percentage of successful message delivery ranging from 2% to 10% compared to NICE. Also, LA-NICE has less average delay and less average message hop count than NICE which reduces the overall latency of message delivery.

KEYWORDS

Application Level Multicast, Multicast tree, Overlay networks, Link Aware.

1. INTRODUCTION

As the number of Internet users increase, data delivery over the Internet becomes more challenging as networks get more overloaded and congested. Currently, data exchange through the Internet is mainly based on unicast (point-to-point between two computers). So, if millions of users try to stream an important broadcast event like the world soccer cup, instead of broadcasting the data to all users, the data source sends a copy of the data to each of the users so the source keeps transmitting the same packet a million times. This leads to redundant traffic in the network in addition to overloading the data source resulting in inefficient data delivery and an increase in packet loss. Multicast was introduced as an alternative to unicast in such cases. In multicast, the source send contents to a sub-server set and each one of those sub-server set forward the content to a different group of users. There are several types of multicast such as IP, overlay, and application level. In IP Multicast, the multicast process is implemented at the IP level during packet transmission. IP multicast provides an efficient multicast technique. However, it was never widely deployed in the Internet due to multiple reasons including the fact that it requires changes at the infrastructural level which slows down the pace of deployment. Also IP multicasting introduces high complexity and serious scaling constraints at the IP layer in order to maintain a state for each multicast group. As a result of the non-acceptance of IP Multicast, the Application layer multicast (ALM) approach was proposed. ALM, also called End-System Multicast, was proposed as an alternative implementation of the multicast technique to the IP Multicast implementation. ALM builds a virtual topology on top of the physical Internet to form an overlay network. Each link in the virtual topology is a unicast link in the physical network [4]. Therefore,

DOI : 10.5121/ijcnc.2014.6302

the IP layer provides a unicast datagram service, while the overlay network implements all the multicast functionality such as dynamic membership maintenance, packet duplication and multicast routing [5].

The rest of the paper is organized as follows. Section 2 describes NICE ALM protocol. Section 3 describes the enhanced Link-Aware NICE protocol. Section 4 presents the simulation design, the implementation and the evaluation and test results. Finally, section 5 presents the conclusion based on the simulation results.

2. ALM OVERLAY CONSTRUCTION

There have been a number of approaches to design ALM protocols. In this section, some of these approaches are introduced. To design a protocol some factors need to be taken into consideration regarding how to manage nodes in multicast groups. This includes how users find out about multicast groups, how they join a group, how they leave the group, abruptly or after notifying the rest of the group, how to handle partitions and splits in the group. These factors of the group management mechanism depend on the nature of the application.

In theory, the overlay network can be viewed as a fully connected graph, as each node can reach every other node in the network via unicast connections. However, only a small subset of the overlay links should be included in the ALM overlay. Hence, the basic functionality of an ALM overlay construction technique is to identify these links and maintain the connectivity of the overlay. The resultant overlay can be in the form of a tree or a mesh which serves two purposes:

- Control topology: is a mesh which provides redundant paths between the members
- Data distribution topology: Which is usually a tree used to ensure loop-free routing

A typical overlay construction technique consists of two phases, the joining phase and the maintenance phase. The joining phase refers to the process where a newcomer is joining an overlay. The maintenance phase manages the connectivity of the overlay. The ALM overlay can be built in a centralized or a decentralized manner.

2.1. Centralized Approach

In the centralized approach, the overlay creation and management are performed by a central controller. Newcomers can learn about the identity of the controller from the Rendezvous Point (RP) that acts as a query server to provide existing members' information to newcomers. The controller uses full knowledge of all members, and potentially the performance metrics (e.g. delay and bandwidth) between the members, to compute a high quality overlay. The computed overlay structure is then distributed to the members in the form of the neighboring relationships (i.e. links) between the nodes.

On receiving such information, the members initiate connections to their assigned neighbors. Once attached to the overlay, the members enter the maintenance phase where they monitor the connections with their respective neighbors. Any changes to the neighbors' status (e.g. leave/fail) gets reported such that the overlay can be repaired.

2.2. Decentralized Approach

In the decentralized approach, the overlay creation and management are done by the members in a distributed fashion. Consider newcomer node *X*, first it requests a list of members in the multicast

group from the RP. From the given list, *X* then selects one or more members as joining targets and sends to each of them a joining request message. When a node, receives a request message, it performs an admission control decision for the requesting node. The main decision criterion is whether it has spare capacity for a new link. Once joined on the overlay, *X* enters the maintenance phase and begins to monitor the status of its neighbors. Any changes to the neighbors' status is normally handled by *X* itself (i.e. the affected member).

While the centralized approach simplifies overlay construction and management, it may not scale well. In particular, the central controller needs to keep track of the information about all members, which is highly dynamic. In addition, it creates a single point of failure problem. Hence, decentralized solutions are more preferable. Decentralized protocols are classified into two groups: Mesh-first and Tree-first.

1) Mesh-First: In the Mesh-first approach, members keep a connected mesh topology among themselves as shown in Figure 1. Usually the source is chosen as a root and a routing algorithm is run over the mesh relative to the root to build the tree [6]. This mesh topology is explicitly created at the beginning, hence it is known. On the other hand, the resulting tree topology is unknown. So the quality of the tree depends on the quality of the mesh chosen. The advantage of Mesh-first approach is that it gives more freedom to refine the tree. It is possible to manipulate the tree topology to a significant extent by selecting mesh neighbors and changing the metrics. A Mesh-first approach is therefore more robust and responsive to tree partitions and is more suitable for multi-source applications, at the cost of higher control overhead. Mesh-first can either be unstructured or structured.

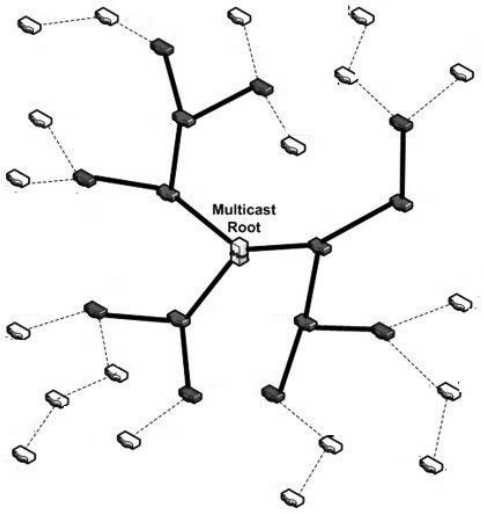

Figure 1. Mesh-first design

Unstructured Mesh: An unstructured mesh is a graph where no special structural information can be inferred from the graph to aid routing and management. Forming such a topology is simple: a newcomer simply attaches itself to some randomly selected members. To obtain loop-free routing trees from this type of topology, a conventional routing protocol such as distance-vector or link-state is needed.

Structured Mesh: While routing in an unstructured mesh is difficult, it is possible to relate the mesh members in a structural manner to simplify routing and management. One of the commonly used structural meshes is distributed hash table (DHT).

2) Tree-First: In the Tree-first approach, the tree is built directly without any mesh. The members explicitly select their parent from the known members in the tree. This may require running an algorithm to detect and avoid loops. The reason for using the Tree-first approach over the Mesh-First approach is that the Tree-First approach gives direct control over the tree. This control is valuable for different aspects such as selecting a best parent neighbor that has enough resources, or responding to the failed members with a minimum impact to the tree. Another advantage of the Tree-First approach is independent actions from each member. It makes the protocol simple as it has a lower communication overhead. For example, when a member changes a parent, it drags all of its descendants with it. This is desirable in the sense that the descendants do not need to change their neighbors; in fact, they are unaware of the incident. However, this can also result in uneven and less efficient trees.

2.3. Existing ALM Protocols

Some of the most popular existing ALM protocols introduced had different designs. As shown in Figure 2, CoopNet [7] an example of a Centralized ALM protocol. NICE [2] and Zigzag [8] as examples for centralized Tree-first protocols. Narada [11] is an example of unstructured mesh and finally Scribe [9] is an example of structured mesh using DHT.

1) CoopNet [7] is a centralized ALM protocol that provides a resilient technique to deliver streaming contents from a single source. It constructs multiple distribution trees, across the range of given nodes, transmitting different multiple description coding MDC) descriptions on every tree. MDC is a method of encoding audio and/or video signal into separate streams, or descriptions, such that any subset of these descriptions can be received and decoded into a signal with distortion (with respect to the original signal) depending on the number of descriptions received, the more descriptions received, the better the quality of the reconstructed signal.

In CoopNet, each sub-stream is delivered using a different distribution tree (formed by the same set of members). This delivery mechanism provides robustness as the probability of all streams concurrently to fail to arrive is very low.

2) Zigzag [8] adopts a similar hierarchical structure to NICE for overlay maintenance. In NICE, the cluster-leaders manage data forwarding to their children which leads to heavier load on higher level nodes. ZIGZAG reduces the bottleneck at the higher level nodes by increasing the size of the cluster in the top layer and sharing the cluster leader load with all the nodes in the cluster [10].

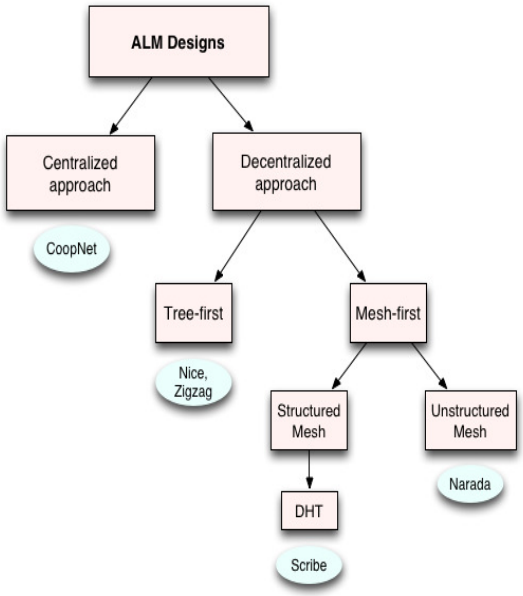

Figure 2. ALM protocols

3) Narada [11] is a protocol suitable for small and sparse groups, for audio/video conferencing, virtual teaching and multiplayer games, not for the delivery applications. The routing control plane of Narada is an unstructured mesh. According to different sending sources, the corresponding data delivery trees are constructed.

In order to obtain a high-quality tree, constructing a good mesh is very important. If a new member *M* wishes to join the group *G*, first of all, it will get a list of group members. The list must contain at least one currently active group member. Then, *M* randomly selects a few group members from the list available to it and sends them messages requesting to be added as a neighbor. This process is repeated until it gets a response. Subsequently, *M* starts exchanging refresh messages with its neighbors. At this time, it has successfully joined the group.

Narada runs a distance vector protocol on top of the mesh. The source trees are constructed by the reverse shortest path between each recipient and the source. Different sources lead to different data delivery tree. A member *M* that receives a packet from source *S* through a neighbor *N* forwards the packet only if *N* is the next hop on the shortest path from *M* to *S*.

One important issue in mesh management is the partition problem. Detecting a partition in a mesh is harder than in a tree. To solve the problem, Narada requires each node to maintain the membership of the overlay, where the dissemination of the membership is integrated with the routing protocol. This, however, leads to a relatively high control overhead. Thus, Narada is effective only for small-scale applications.

4) Pastry is an overlay and routing network for the implementation of a distributed hash table (DHT). In Pastry, each overlay node is assigned a random node ID that is uniformly selected from a one-dimensional circular namespace of 128 bits. Given a message and a destination node ID, Pastry routes the message to a node with the node ID that is numerically closest to the key, among all live nodes. Figure 3 illustrates an example where a message targeted for node **c** is routed from node ***a*** to node ***b***. The Pastry overlay is created in the following order.

- First, a new member uses a deterministic hash function to create a unique node ID.
- It then sends a join message addressed to its own node ID through a close-by overlay node, say node a. Pastry assumes that the knowledge of such a node is learnt from some out-of-band techniques, such as an expanded ring search.
- Node ***d*** then forwards the message hop-by-hop towards the destination node ID.
- Finally, the message reaches node ***b*** which has the closest node ID to ***e***. The newcomer obtains the state information from nodes on the path from ***a*** to ***f*** to establish its routing table and neighbors list.

5) Scribe [9, 12] is a network of Pastry nodes, where each node runs the Scribe application software. Any Scribe node can create a topic and other nodes can register their interest in this topic by becoming members of the topic. A Scribe node with the appropriate credentials for the topic can then publish events related to the topic and Scribe will disseminates these events to the topic's subscribers. Scribe uses best-effort dissemination of events and does not guarantee ordered delivery. However reliability can be implemented on top of Scribe. The Scribe software provides two methods, forward and deliver which are invoked by Pastry whenever a message arrives, the former when the node is an intermediate stop towards the final destination, the latter when the node is the final destination itself. The possible message types in Scribe are:

- Subscribe to a topic
- Unsubscribe from a topic
- Create a topic
- Publish an event to a topic

Each topic has a unique topic ID. The Scribe node with node ID numerically closest to the topic ID acts as the rendezvous point (RP) for that topic. The RV point forms the root of the multicast tree associated to the topic. To create a topic, a Scribe node asks Pastry to route a create message to the RV point of the topic. The Scribe deliver method adds the topic to the set of topics that this node is aware of. The topic ID can be the hash value of the topic's textual name concatenated with its creator's name. The hash is computed using a collision resistant hash function (e.g. SHA-1), which ensures a uniform distribution of topic IDs. Combining this with the uniform distribution of Pastry's node IDs, this ensures an even distribution of topics across nodes.

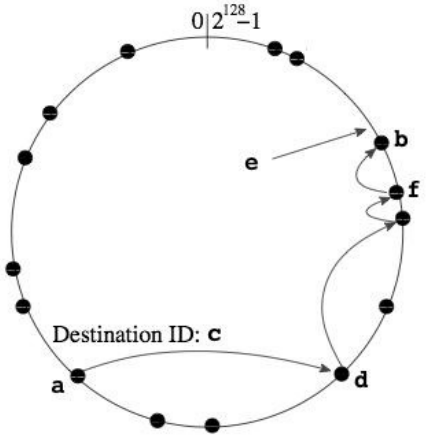

Figure 3. Pastry routing in a circular namespace (each dot depicts a live node in the namespace) [9]

Scribe creates a multicast tree per topic rooted at the RV point of the group to disseminate events

along the tree. The multicast tree is created using reverse path forwarding, since the shortest paths in the overlay towards the RV point are merged. Each node in the multicast tree is a forwarder for the group. A forwarder may or may not be a subscriber to the group. Each forwarder maintains a children table that contains an entry (IP address and node ID) for each of its children in the routing table.

6) NICE [2] is an ALM protocol that arranges the set of members in a multicast group into a hierarchical control topology. As new members join and existing members leave the group, the basic operation of the protocol is to create and maintain the multicast tree hierarchy. The NICE hierarchy is created by assigning members to different levels (or layers) as illustrated in layer 0, Figure 4. Layers are numbered sequentially with the lowest layer of the hierarchy being layer zero (L_0). Members in each layer are partitioned into a set of clusters [13]. Each cluster is of size between k and $3k-1$ members, where k is a constant (usually $k=3$), and consists of a set of members that are close to each other. Further, each cluster has a cluster leader. The protocol chooses the center of the cluster to be its leader, i.e., the cluster leader has the minimum distance to all other members in the cluster. This choice of the cluster leader ensures that a new joining member is quickly able to find its appropriate position in the hierarchy using a very small number of queries to other members. The leaders in level i are the members of level $i+1$ in the tree, so all the leaders in L_0 belong to L_1 and their leaders belong to L_2 and so on until there is only one leader which is the Rendezvous Point (RP) in the highest level of the tree. Since each cluster in the hierarchy has between k and $3k-1$ members, a host that belongs only to L_0 layer peers with $O(k)$ other hosts for exchange of control messages. In general, a host that belongs to layer L_i and no other higher layer, peers with $O(k)$ other hosts in each of the layers $L_0 \ldots L_i$, which results in control overhead $O(k \times i)$ for this member. Hence, the cluster-leader of the highest layer cluster peers with a total of $O(k \times log\ N)$ neighbors, which is the worst case control overhead at a member. NICE mainly focuses on minimizing end-to-end delay. This is done by computing the distance between the nodes and constructing the tree such that nodes close to each other get assigned to the same cluster. This technique minimizes end-to-end delays as the cluster leader is always centered in the middle of the cluster where the distance between it and the rest of the cluster members is minimum.

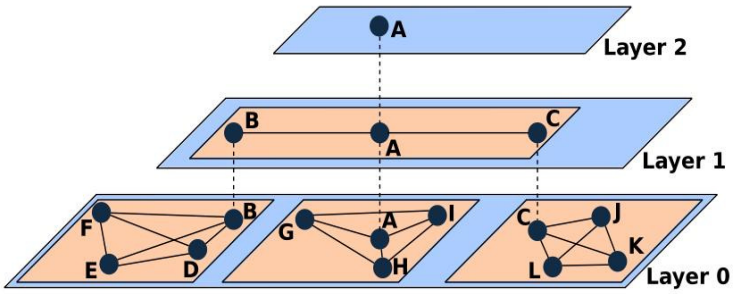

Figure 4. Hierarchical arrangement of hosts in NICE [14]

3. PROPOSED LINK-AWARE NICE (LA-NICE)

LA-NICE is an enhancement to the NICE protocol. NICE mainly focuses on minimizing end-to-end delay. This is done by computing the distance between the nodes and constructing the tree such that nodes close to each other get assigned to the same cluster. This technique minimizes end-to-end delays as the cluster leader is always centered in the middle of the cluster where the distance between it and the rest of the cluster members is minimum. LA-NICE takes into account the fact that different links can have different bandwidths and uses this fact to improve multicast message delivery and minimize end-to-end delay. As shown in Figure 5, LA-NICE doesn't

change the cluster structure or how the cluster splits or merges; it focuses on two phases in the tree management which are the member join and the tree maintenance phases.

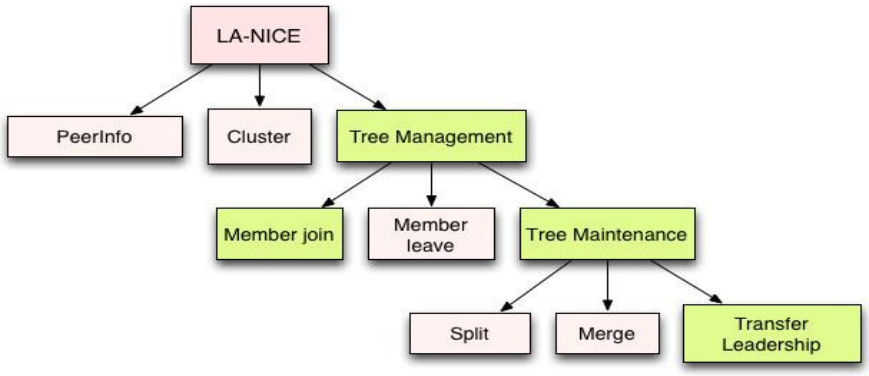

Figure 5. LA-NICE code structure

3.1. Multicast Tree Member Join Procedure

When a new node wants to join the multicast tree, in the original NICE algorithm, it contacts the RP with its join query. The RP responds with the hosts that are present in the current highest level of the tree L_i. The joining host then contacts all members in L_i to locate the member closest to it. This member then informs the joining host of its other members in L_{i-1} layer and so on recursively until the joining host finds its position in the lowest level of the tree. Meanwhile, the joining host gets peered temporarily with the RP as soon as it starts communicating with it to get the multicast messages sent until it finds its appropriate position in the tree. This ensures that the joining node gets connected to the tree as soon as it requests to join. Right before joining the tree in the lowest level and after knowing where exactly it will join, the member node requests to be disconnected from the RP and requests to join its appropriate parent in the tree.

LA-NICE modifies the member join procedure of NICE. NICE members join the closest clusters based on round trip time (RTT) measurements. The RTT of sending a message is the time it takes for the message to be sent plus the time it takes for an acknowledgment of that message to be received. The RP gets a list of the clusters leaders ordered by distance and assigns the new host to the closest cluster to it. In LA-NICE, in a tree of i levels, the RP gets a set of potential clusters (PC) that the new node can join using Eq. (1)

$$PC = min_{RTT}\,(\text{New node}, \text{All leaders}) \qquad (1)$$

Where PC is a set of clusters of size i. Then the RP checks the bandwidth of the leaders of the potential clusters and assigns the new node to the cluster with the highest ratio of leader bandwidth/number of cluster members as shown in Eq. (2).

$$\text{Selected cluster} = \max\left(\frac{\text{bandwidth}}{\text{cluster size}}\right)\,(\text{PC leaders}) \qquad (2)$$

3.2. Multicast Group Member Leave Procedure

When nodes leave the tree, they can leave either gracefully or ungracefully. A graceful leave, which is when a host leaves the multicast group after notifying all the clusters it has previously joined that it's leaving. On the other hand, an ungraceful leave occurs when member fails without being able to send out a leave notification to its clusters, the cluster members then manage to

detect this departure when they don't receive HeartBeat message from that member. A HeartBeat message is a periodic message sent by every member to the rest of the multicast group informing them that it still exists in the group. If a cluster leader left the group, this could lead to a partition in the tree so a new leader needs to be chosen faster. A new leader of the cluster is chosen depending on who is estimated to be closest to the center among these members.

3.3. Multicast Tree Maintenance

A cluster leader periodically checks the size of its cluster, and appropriately splits or merges the cluster when it detects a size bound violation. If a cluster exceeds the cluster size upper bound *3k-1*, it gets split into two equal-sized clusters. Given a set of hosts and the distances between them, the cluster split operation partitions them into subsets that meet the size bounds, such that the maximum radius of the new set of clusters is minimized. The centers of the two partitions are chosen to be the leaders of the new clusters and transfers leadership to the new leaders. If these new clusters still violate the size upper bound, they are split by the new leaders using identical operations.

To maintain the tree structure and detect any unexpected partitioning in the tree, each member of a cluster sends periodic HeartBeat message to each of its cluster members. The message contains the distance estimate from the sender to each other member of the cluster. The cluster leader includes the complete updated cluster membership in its HeartBeat messages to all other members. This allows existing members to set up appropriate peer relationships with new cluster members on the control path. The cluster leaders also periodically send their higher layer cluster membership their cluster.

LA-NICE takes the link load into account when maintaining the tree. So instead of only checking if the cluster leaders need to be modified based on the leader's position with respect to the cluster members, LA-NICE checks the bandwidth of the closest 3 nodes to the center of the cluster (given that the cluster has more than 3 members) and assigns the one with the highest ratio of (bandwidth/number of cluster members) to be the cluster leader. This modification is selecting the cluster leader based on the fact that there is always higher load on the cluster leaders than the other nodes in the cluster since the cluster leaders send the multicast messages to all the cluster members leading to a bottleneck of $O(k \log N)$. So, ensuring that the leader has a relatively high bandwidth in addition to being close to all the members reduces the delay and improves multicast message deliver.

4. RESULTS AND ANALYSIS

4.1. Experiment Setup

To evaluate LA-NICE, we compared it to NICE protocol and Scribe [9] which is another ALM protocol. The evaluation was done using four different test groups of users. Each group had different number of users. The number of users in those groups were 20, 45, 70, and 100 respectively. The simulation was done using OMNeT++ as shown in Figure 6 and it runs for 300 seconds where nodes exchange multicast messages where some users would drop out while other join the group randomly.

4.2. OMNeT++

OMNeT++ [15] is an object-oriented open-source network simulator that is highly flexible and easy to use. The model's structure is described in OMNeT++ Network Description (NED) language. NED facilitates the modular description of a network, which consists of a number of component descriptions (channels, simple/compound module types, gates, etc.). In addition, OMNeT++ provides various statistic collections and visualization tools for results analysis. The simulator supports parallel and distributed simulation with the multiple instances communicating via Message Passing Interface (MPI), as well as support for network emulation through interfaces with real networks and the ability to use real networking code inside the simulator. OMNeT++ simulation models are composed of modules and connections. Connections may have associated channel objects. Channel objects encapsulate channel behavior: propagation and transmission time modeling, error modeling, and possibly others. Channels are also programmable in C++ by the user. Modules and channels are called components. Components are represented with the C++ class cComponent.

4.3. Implementation

The implementation was done using OMNeT++ framework with oversim [16]. Oversim includes a basic implementation of NICE protocol. They were both used in evaluating LA-NICE performance. The implementation of LA-NICE was built using OMNeT++ oversim framework. In LA-NICE, both the member join and the maintenance methods were modified. The code is implemented in C++ to write the logic of the protocol and OMNeT's.Net language to represent the user interface of the network. Datarate channels were used with different datarates to test various network conditions and bandwidth variations.

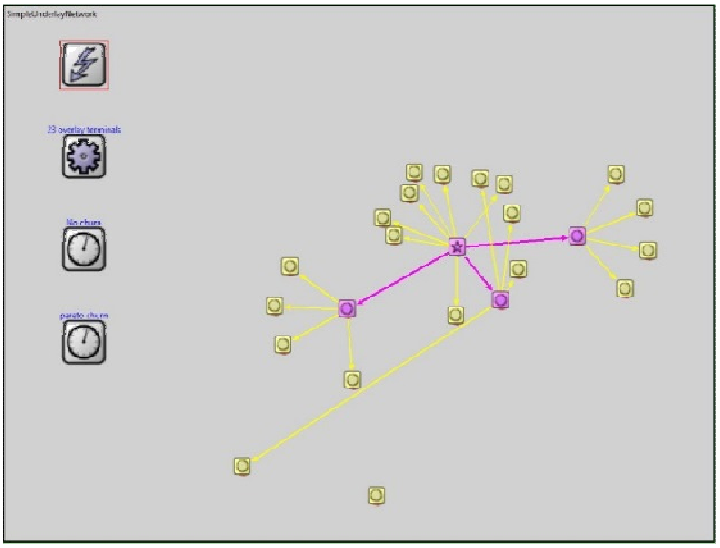

Figure 6. LA-NICE multicast tree simulation using OMNeT++

4.4. Results

Message Delivery: The first evaluation criteria for LA-NICE is the percentage of multicast messages delivered, failure in delivery indicates inefficient tree maintenance due to not detecting node departures in a reasonable amount of time or bad bandwidth utilization, which results in bottlenecks that lead to late delivery and sometimes failure in delivery.

Table 1. Message Delivery Percentage Results

Number of users	NICE	Scribe	LA-NICE		
				% Improvement	
				over NICE	over Scribe
20	90.93	88.57	99.34	8.4%	10.7%
45	96.01	89.65	98.69	2.7%	9.04%
70	90.48	88.71	96.53	6.05%	7.8%
100	93.76	89.62	95.68	1.92%	6.06%

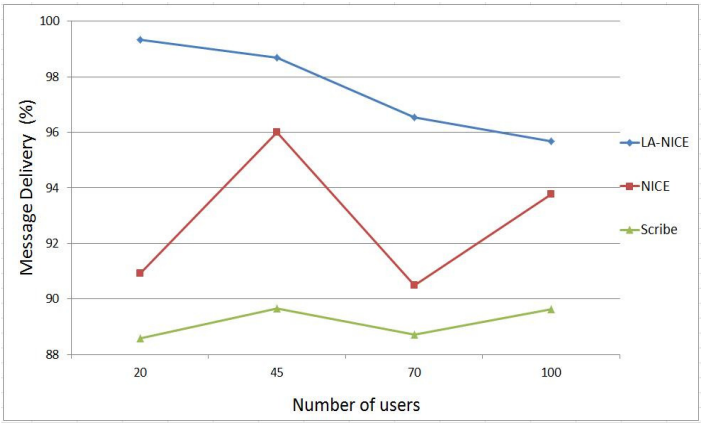

Figure 7. Percentage of Message Delivery of LA-NICE, NICE and Scribe against different number of users

As seen in Table 1 and Figure 7, scribe has lower message delivery percentage. The reason behind the less performance in Scribe is due to the fact that the nodes are assigned random generated IDs. This randomness could lead to a situation where two hosts can be close to each other and yet sending a message from one of them to the other passes by other nodes that are far from them which takes longer time than it should. This is due to the routing table which sends messages to hosts that have the same prefix in their ID (which doesn't always mean that this is the closest node in the table). In addition, messages have a certain time to live (TTL) and then they timeout, so as the delay increases the amount of messages that timeout increase. NICE, on the other hand doesn't have this problem and it sends the messages to the closest nodes graphically without any regard to bandwidth conditions. LA-NICE combines both distance and bandwidth factors.

On the other hand, NICE takes the node's proximity to each other into account when constructing the delivery tree. For example, if two nodes are close to each other, the time taken to send exchange messages between them should be less than the time taken to exchange message between nodes that are further away from each other.

Our main focus is to build on NICE and enhance its performance by making it link-aware. Link-Aware NICE handles the proximity factor in NICE in addition to the bandwidth factor when a new node joins the tree and when maintaining the tree as well. This is done through measuring the cluster leaders' bandwidth and selecting the cluster with the highest leader bandwidth that is relatively close. This approach produced the highest message delivery percentage as seen in

Figure 7 compared to NICE and Scribe.

Table 2 Average Hop Count Results

Number of users	LA-NICE	NICE
20	1.55	1.48
45	1.73	1.77
70	1.79	2.1
100	1.78	2

Table 3. Maximum Hop Count Results

Number of users	LA-NICE	NICE
20	2	2
45	4	4
70	4	7
100	3	6

Message Delay and Hop count: Another evaluation criteria is the average hop count of the multicast messages. The hop count of a packet is defined as the number of routers traversed by a packet between its source and destination [17], which is in this case the number of hosts that a message passes from the source to the destination [18]. As seen in Figures 8 and 9 and Tables 2 and 3 when the number of users is small, the average hop count of LA-NICE and NICE, as well as the maximum hop count is almost the same, however as the number of users increase LA-NICE has less number of hops leading to less delay in delivering the messages.

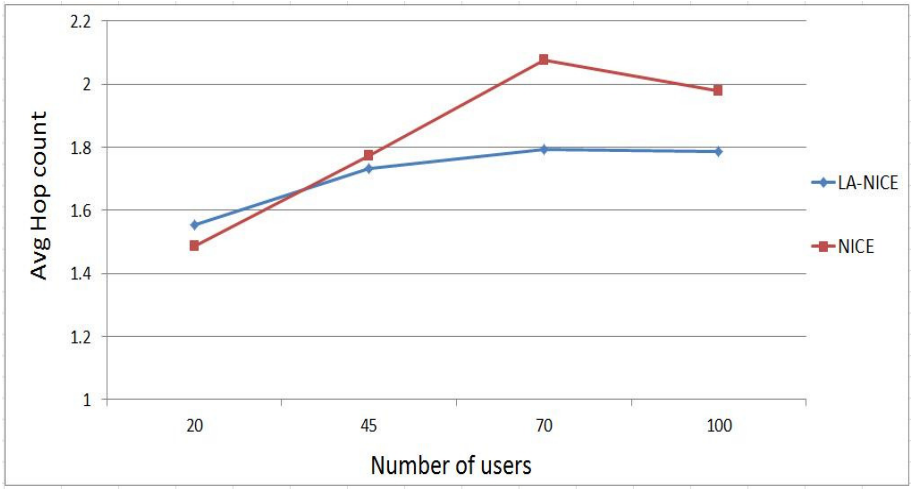

Figure 8. Average hop count in LA-NICE and NICE against different number of users

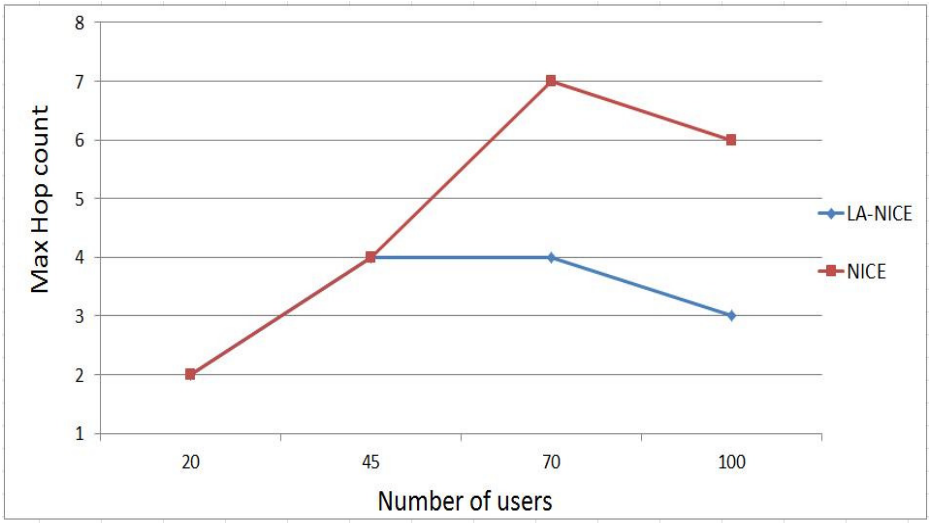

Figure 9. Maximum hop count in LA-NICE and NICE against different number of users

Table 4. Average Delay Results (S)

Number of users	LA-NICE	NICE
20	0.923	0.91
45	0.83	0.9
70	1.54	1.58
100	1.21	1.19

Table 5. Maximum Delay Results (S)

Number of users	LA-NICE	NICE
20	1.67	1.72
45	1.76	1.57
70	2.48	2.48
100	2.01	2.05

The delay factor is related to the hop count. Figures 10 and 11 and Tables 4 and 5 show a comparison between LA-NICE, NICE in terms of delay. The delay and message hops are usually proportional, therefore, with the increase of the hops along the tree, the delay increases. It is clear that LA-NICE outperforms NICE in terms of delay and hop count after taking the links' load factor into account.

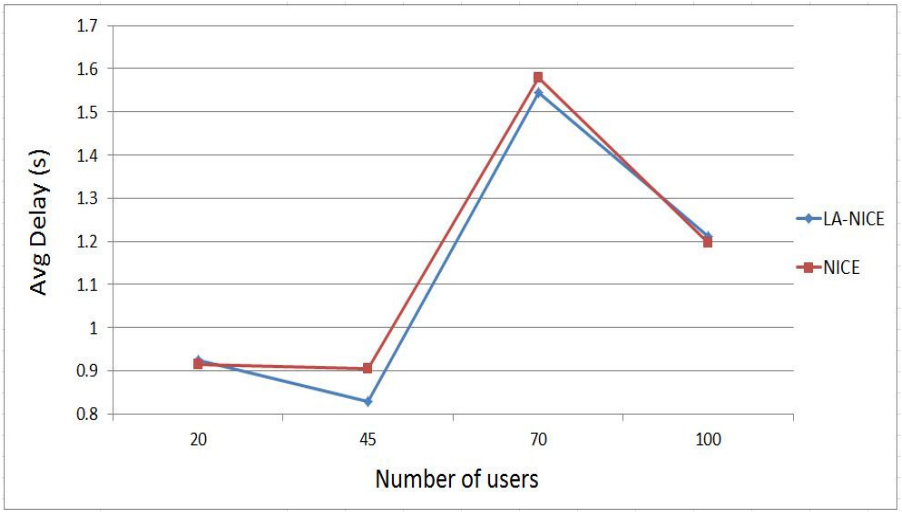

Figure 10. Average Delay in LA-NICE and NICE against different number of users

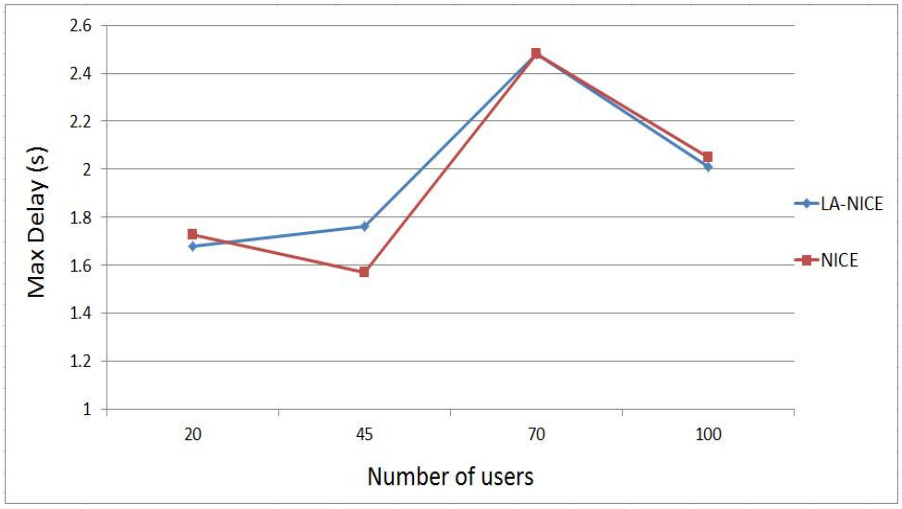

Figure 11. Maximum Delay in LA-NICE and NICE against different number of users

REFERENCES

[1] D. Helal, A. Naser, M. Rehan, and A. ElNaggar, Link-aware nice application level multicast protocol, The Ninth International Conference on Digital Telecommunications (2014) 8–12.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy, Scalable applica-tion layer multicast, vol. 32, no. 4. New York, (2002) 205–217.

[3] A. Varga and R. Hornig, An overview of the omnet++ simulation environment, 1st International Conference on Simulation Tools and Techniques for Communications, Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, Brussels, Belgium (2008) 60:1–60:10.

[4] K. Ke and C. Huang, Performance evaluation of multisource applica-tion layer multicast (alm): Theoretical and simulative aspects, vol. 57, no. 6. New York (2013) 1408–1424.

[5] L. Lao, J.-H. Cui, M. Gerla, and D. Maggiorini, A comparative study of multicast protocols: top, bottom, or in the middle?, INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. (2005) 2809–2814.

[6] B. Rong and A. Hafid, A distributed relay selection algorithm for cooperative multicast in wireless mesh networks. 5th International Conference in Mobile Ad-hoc and Sensor Networks (2009) 49–56.

[7] V. N. Padmanabhan, H. J. Wang, P. A. Chou, and K. Sripanidkulchai, Distributing streaming media content using cooperative networking. NOSSDAV. ACM, (2002) 177–186.
[8] D. A. Tran and et al., Zigzag: An efficient peer-to-peer scheme for media streaming, IN PROC. OF IEEE INFOCOM, 2003.
[9] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron, Scribe: A large-scale and decentralized application-level multicast infrastructure, vol. 20 (2002) 1489 – 1499.
[10] M. Hosseini, D. T. Ahmed, S. Shirmohammadi, and N. D. Georganas, A survey of application-layer multicast protocols, IEEE Commun. Surveys and Tutiruals (2007).
[11] Y. Chu, S. G. Rao, S. Seshan, and H. Zhang, A case for end system multicast, ACM Sigmetrics, (2000) 1–12.
[12] A. Rowstron, A. Kermarrec, M. Castro, and P. Druschel, Scribe: The design of a large-scale event notification infrastructure, Networked Group Communication, 2001, pp. 30–43.
[13] X. Li, X. Zhang, W. Luo, and B. Yan, A clustering scheme in application layer multicast. vol. 30, no. 2, (2011) 335–355.
[14] W. Xijuan, J. Ruisheng, L. Guang, and Y. Xianghong, Research on p2p-based application layer multicast technology for streaming media, IEEE Computer Society, Los Alamitos, (2010) 341–345.
[15] A. Varga, Omnet++ discrete event simulation system user manual (2011).
[16] I. Baumgart, B. Heep, and S. Krause, Oversim: A scalable and flexible overlay framework for simulation and real network applications, Peer-to-Peer computing. IEEE Ninth International Conference on P2P, (2009) 87–88.
[17] C. Hubsch,¨ C. P. Mayer, and O. P. Waldhorst, User-perceived performance of the nice application layer multicast protocol in large and highly dynamic groups, 15th international GI/ITG conference on Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance. Berlin (2010) 62–77.
[18] S. Rizvi, M. Khan, and A. Riasat, Deterministic formulization of bandwidth efficiency for multicast systems, 2nd International Conference on Computer, Control and Communication (2009) 1–6