Minimizing the delay and delay variation constraints for collaborative applications based on overlay networks

A. Namachivayam and Kaliyaperumal Karthikeyan
Eritrea Institute of Technology, Asmara, Eritrea, North East Africa.

ABSTRACT
The main objective of this research is to minimize delay in collaborative applications by setting tight bounds. Examples of collaborative applications include video-conferencing, distributed database replication, and online games. The efficiency of this project is analyzed and shown to be superior to its counterparts in the execution time. To provide an efficient heuristic to obtain a multicast sub network on an overlay network, given a source and a set of destinations that is within a specified maximum delay and a specified maximum variation in the delays from a source to the destinations.

Keywords
Multicast routing,
Delay and delay variation,
Overlay networks.

Introduction
The problem of Delay and Delay Variation Bound in Multicasting has been addressed currently using the following four approaches.
-Delay Variation Bound Multicast Algorithm (DVBMA)
-Delay and Delay Variation Constraint Algorithm (DDVCA)
-Dynamic program for Delay Variation Bound (DPPVB)
-Buffering of Messages to reduce delay variation

The main objectives of this minimize delay in collaborative applications by setting tight bounds. Examples of collaborative applications include video-conferencing, distributed database replication, and online games.

The efficiency of this project is analyzed and shown to be superior to its counterparts in the execution time.

Table 1. The List of Paths from V2 to V2, V5 and V8 and their corresponding End-to-End Delays

<table>
<thead>
<tr>
<th>Source</th>
<th>Destination</th>
<th>Path</th>
<th>Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>V2</td>
<td>(a) V3-V1-V2</td>
<td>31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(b) V3-V1-V3-V2-V2</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(c) V3-V1-V4-V5-V2-V2</td>
<td>35</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(d) V3-V1-V3-V2-V2-V2</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>V5</td>
<td>(e) V5-V1-V2-V3-V5</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(f) V5-V1-V2-V3-V5</td>
<td>32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(g) V5-V1-V2-V3-V5</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(h) V5-V1-V2-V3-V5</td>
<td>45</td>
<td></td>
</tr>
<tr>
<td>V8</td>
<td>(i) V5-V1-V5</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(j) V5-V1-V2-V3-V8</td>
<td>43</td>
<td></td>
</tr>
<tr>
<td></td>
<td>(k) V5-V1-V2-V3-V8</td>
<td>46</td>
<td></td>
</tr>
</tbody>
</table>

For example, let us suppose that we choose paths (a), (e), and (i) (the first shortest paths from the source to each of the destinations V2, V5, and V8, respectively). Then, after merging these paths, the resulting sub network will be a shortest path tree. If we remove the delay variation constraint from our problem, then the shortest path tree will be the optimal multicasting tree. Note that destination nodes V2, V5, and V8 receive the message from Vs at time units 31, 26, and 20, resulting in a delay variation of 11.
If the paths that are chosen are (d), (g), and (j), the end-to-end delays for each of the destinations V2, V5, and V8 will be 40, 40, and 43, respectively. These delays are within the desired delay bound of 50, but the delay variation is only 3, smaller then the delay variation of 11 in the previous case. Merging these paths we obtain a sub network as shown in Fig.1

This heuristic achieves the tightest possible bounds on delay variation.

In order to perform a join operation (where a single node joins the multicasting group) or leave operation (where a single node leaves the multicasting group), it has show that the heuristic has a tight delay variation with less complexity.

A source-based heuristic is proposed which does not create overhead at destinations or intermediate nodes.

The heuristic is applicable to both proxy-based and peer-to-peer overlay networks.

Modules:
- Constructing a Multicast Topology
- Finding all Paths and Delay
- Finding Delay Variation and Shortest Path
- Sending a Message

Constructing a multicast topology:

In this Module a Multicasting topology is constructed. The source and destinations are identified.

Finding all paths and delay:

- In this Module all possible paths for the destinations are discovered.
- Delays for the shortest path are found.

Finding delay variation and shortest path:

- Here the delay variation is found out; the delay variation is obtained from the delays.
- Based on the delay variation choose a shortest path. The delay variation of the shortest path is minimum than other delay variance.

Multicast message:

In this module the message is multicast through the shortest path with minimum delay variation to the destinations

Problem Formulation

One approach to ensure minimum delay variation is to buffer the messages at different nodes in the overlay network. This approach can be categorized into three classes: buffering at the source node, buffering at intermediate nodes, and buffering at destination nodes.

Buffering at the source node requires the source node to keep additional information for each destination. The source node will buffer a message for a different amount of time for each destination and transmit the message multiple times over the network; clearly, this is a waste of network bandwidth. Also, buffering at the source node defeats the purpose of multicasting, which is one of conserving network bandwidth.

Buffering at intermediate nodes requires some nodes to be identified as core nodes in the network. Messages are buffered at these core nodes before they are sent to the destinations.

Buffering at destination nodes requires each destination node to buffer the messages before they pass the messages to the application process. In this approach, the source node informs the destination nodes when they can process the received packets.

Networking
TCP/IP Stack

The TCP/IP stack is shorter than the OSI one.

TCP
TCP supplies logic to give a reliable connection-oriented protocol above IP. It provides a virtual circuit that two processes can use to communicate.

UDP
UDP is also connectionless and unreliable. What it adds to IP is a checksum for the contents of the datagram and port numbers. These are used to give a client/server model - see later.

IP Datagram’s

The IP layer provides a connectionless and unreliable delivery system. It considers each datagram independently of the others. Any association between datagram must be supplied by the higher layers. The IP layer supplies a checksum that includes its own header. The header includes the source and destination addresses. The IP layer handles routing through an Internet. It is also responsible for breaking up large datagram into smaller ones for transmission and reassembling them at the other end.

In order to use a service, you must be able to find it. The Internet uses an address scheme for machines so that they can be located. The address is a 32 bit integer which gives the IP address. This encodes a network ID and more addressing. The network ID falls into various classes according to the size of the network address.

Network Address

Class A uses 8 bits for the network address with 24 bits left over for other addressing. Class B uses 16 bit network addressing. Class C uses 24 bit network addressing and class D uses all 32.

Subnet Address

Internally, the UNIX network is divided into sub networks. Building 11 is currently on one sub network and uses 10-bit addressing, allowing 1024 different hosts.
**Host Address**

8 bits are finally used for host addresses within our subnet. This places a limit of 256 machines that can be on the subnet.

**Total Address**

```
137.92.11.13
```

The 32 bit address is usually written as 4 integers separated by dots.

**Port Addresses**

A service exists on a host, and is identified by its port. This is a 16 bit number. To send a message to a server, you send it to the port for that service of the host that it is running on. This is not location transparency! Certain of these ports are "well known".

**Sockets**

A socket is a data structure maintained by the system to handle network connections. A socket is created using the call `int socket(int family, int type, int protocol);`

The implementation can be preceded through Socket in java but it will be considered as one to all communication. For proactive broadcasting we need dynamic linking. So java will be more suitable for platform independence and networking concepts. For maintaining route information we go for SQL-server as database back end.

**Time Series Chart Interactivity**

Implement a new (to JFreeChart) feature for interactive time series charts --- to display a separate control that shows a small version of ALL the time series data, with a sliding "view" rectangle that allows you to select the subset of the time series data to display in the main chart.

**Dashboards**

There is currently a lot of interest in dashboard displays. Create a flexible dashboard mechanism that supports a subset of JFreeChart chart types (dials, pies, thermometers, bars, and lines/time series) that can be delivered easily via both Java Web Start and an applet.

**Property Editors**

The property editor mechanism in JFreeChart only handles a small subset of the properties that can be set for charts. Extend (or reimplement) this mechanism to provide greater end-user control over the appearance of the charts.

**Design Overview**

Design involves identification of classes, their relationships as well as their collaboration. In objector, classes were divided into Entity classes, interface classes and the control classes. The Computer Aided Software Engineering tools that are available commercially do not provide any assistance in this transition. Even research CASE tools take advantage of meta modeling are helpful only after the construction of class diagram is completed. In the Fusion method, it used some object-oriented approaches like Object Modeling Technique (OMT), Class, Responsibility, Collaborator (CRC) and Objector, used the term Agents to represent some of the hardware and software systems.

In Fusion method, there was no requirement phase, where in a user will supply the initial requirement document. Any software project is worked out by both analyst and designer. The analyst creates the Use case diagram. The designer creates the Class diagram. But the designer can do this only after the analyst has created the Use case diagram. Once the design is over it is need to decide which software is suitable for the application.

**Implementation**

Implementation is the stage of the project when the theoretical design is turned out into a working system. Thus it can be considered to be the most critical stage in achieving a successful new system and in giving the user, confidence that the new system will work and be effective.

The implementation stage involves careful planning, investigation of the existing system and it’s constraints on implementation, designing of methods to achieve changeover and evaluation of changeover methods.

Implementation is the process of converting a new system design into operation. It is the phase that focuses on user training, site preparation and file conversion for installing a candidate system. The important factor that should be considered here is that the conversion should not disrupt the functioning of the organization.

The implementation can be preceded through Socket in java but it will be considered as one to all communication. For proactive broadcasting we need dynamic linking. So java will be more suitable for platform independence and networking concepts. For maintaining route information we go for SQL-server as database back end.

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the...
functionality of components, sub assemblies, assemblies and/or a finished product.

It is the process of exercising software with the intent of ensuring that the software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

Figure 4: Find All Possible Paths and Delay

Figure 5: Find Delay Variation and Choose Shortest Path

Figure 6: Multicast Message to the Destination:

Figure 7: Case diagram

Conclusion and future enhancement

For dynamic reorganization of the multicasting sub network with the tightest delay variation and bounded delay, it is noticed that, the solution with Chains is more efficient than that of DVMA in terms of time-complexity. As part of the future research work, link delays are considered as time-varying functions and required to develop efficient heuristics for the DVBMN problem.

In this paper, the problem is considered of determining a multicasting sub network with end-to-end delay bound and delay variation bounded for collaborative applications on overlay network. Three well-known heuristics from the literature discussed and exposed their limitations.

Then, the new heuristic Chains are presented, which achieve the tightest delay variation for a given delay bound. At the initial phase of the heuristic, the k is used for shortest path technique proposed by Victor and Andres [16] to find all paths for each destinations for which the delays are less than or equal to the delay bound. Then, using these delays, the delay chain is determined, which gives the minimum delay variation and constructed the multicasting sub network by retrieving the paths from the delays.

Implemented all the heuristics and observed Chains outperform DPDVB and DVMA in terms of execution time. The Chains heuristic also achieves the tightest delay variation bound along with DPDVB. The results are also presented to show that finding k shortest paths for all destinations is not a bottleneck in the solution. It is also observed that Chains require higher values of k to achieve the tightest delay variation when the graph becomes dense or when the end-to-end delay bound increases. For dynamic reorganization of the multicasting sub network with the tightest delay variation and bounded delay, It is noticed that the solution with Chains is more efficient than that of DVMA in terms of time-complexity.

References
