Aggregated Hierarchical Multicast for Active Networks

Tilman Wolf and Sumi Y. Choi
{wolf, syc}@arl.wustl.edu
Department of Computer Science
Washington University in St. Louis, MO, USA

Abstract— **Active Networking is the basis for a range of new and innovative applications that make use of computational resources inside network routers. One such application is Aggregated Hierarchical Multicast, which aims at implementing efficient many-to-many communication. In certain scenarios it is possible to transmit less accurate, aggregated data and thus achieve better scalability. Using Active Networks, the aggregation computation can be done transparently by network routers without end system support. We present how aggregated data streams can be structured in a hierarchical fashion to allow easy access of data at the desired aggregation level. We introduce two application examples to illustrate the system design, analyze the performance of the aggregation mechanism, and evaluate it using a prototype implementation.**

Keywords— **active networks, multicast, data aggregation, active network application**

I. Introduction

Many-to-many or multi-source communication is becoming an increasingly important way of communicating. As more and more real-time data is sent by sources distributed over the network, the receiving end-systems become overwhelmed by the amount of data traffic. In a traditional many-to-many communication, each sender is connected to each receiver (be it over unicast or multicast). As a result, a receiver has to deal with as many connections as there are senders. This does not scale well, neither in terms of bandwidth requirements nor in terms of computational demands. Handheld clients with little computational resources or devices connected over low bandwidth wireless links are severely limited in the number of connections they can handle.

A key observation, though, is that for certain applications in many-to-many communications the senders are not of equal importance to the receiver. Thus, it is sufficient to aggregate the information sent by most of the sources, while keeping the full data stream from a few selected sources. To illustrate this concept, we look at two applications: a battlefield information system and an audio conferencing application. We show how aggregation is done for each application and how a hierarchical overlay allows the user to dynamically choose the right level of detail for his needs.

This research was supported by DARPA under contract number N66001-98-C-8510.

The basis for the work lies in the assumption that the interconnection network between the senders and receivers is capable of performing the aggregation of data on-the-fly. Thus, we require what is referred to as an "active network" [1], [2]. An active node in such a network is capable of performing processing of packets as they are being forwarded. Typical implementations range from workstations that act as active routers to high-performance switches that are augmented with per-port network processors. The definition of a unifying node operating system (NodeOS [3]) aims at making these systems interoperable. While Aggregated Hierarchical Multicast is an application for active networks, we do not go into the details of active networking in this paper.

Section II introduces the example applications considered here and shows different information aggregation methods. Section III discusses the hierarchical structure of mutlicast sessions that provides various levels of aggregation detail to the user. Section IV describes a general-purpose aggregation algorithm that is used in our prototype implementation. Section V gives quantitative evaluation of Active Hierarchical Multicast. A summary of the contributions in Section VI concludes this paper.

II. Information Aggregation and its Applications

As discussed above, the limited scalability of many-to-many communications lies in the demands on the network to deliver numerous data streams to the end system and the demands on the end system to process and display the information. To be more concrete, we look at the following two applications.

A. Application I: Battlefield Information System

The battlefield information system is aimed at providing status information of numerous soldiers and

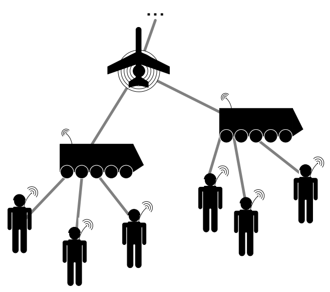

Fig. 1. Battlefield Information Application for Aggregated Hierarchical Multicast. Each node sends its status information to its hierarchical parent, where it is aggregated.

equipment to a large number of commanders and observers. We assume that all soldiers are connected to a common interconnection network (i.e., via wireless links) and have the equipment to periodically transmit their status information (i.e., geographic location, vital statistics, and other easily observable data). The observers can receive this data and display the status of all soldiers accordingly. This application is an example of large multicast with many sources.

The challenge of this application lies in the large amount of data that is received by an observer. Considering only a few thousand soldiers, who transmit every few seconds, gives an average of several hundred data packets per second that have to be processed and displayed.

Since it is unlikely that any observer is interested in the exact status of every single soldier at all times, it is possible to present aggregated status information of a group of soldiers to an observer thereby reducing the number of data streams that are sent to a receiver. Geographic information for a group of people, for example, can be represented by its centroid, the weighted geographic average, or the convex hull. This reduces the amount of detail in the representation, but it also reduces the amount of data that needs to be transmitted. Similarly, vital statistics can be aggregated. For example the predicate ”healthy” of each soldier can be aggregated using a boolean “and” function.

Finally, there is the question of where in the network the aggregation computations should be done. For this purpose, we use a given hierarchical structure (i.e., the typical chain-of-command). Each node sends its information to its hierarchical parent, where the information is aggregated with that of the node's siblings. This is repeated over all levels of the hierarchy. An illustration of this is shown in Figure 1. More details on how to tap the hierarchy at the right level to get the right detail of information is discussed in Section III.

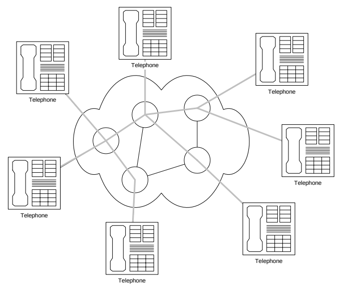

Fig. 2. Audio Conferencing Application for Aggregated Hierarchical Multicast. Each end system sends audio that is mixed at nodes where multiple audio streams merge. The end system receives a single, mixed audio stream.

B. Application II: Audio Conferencing

An audio conferencing application, as illustrated in Figure 2 is another typical example of a many-to-many multicast. Each participant in the conference needs to be able to hear all other participants and therefore needs to receive their audio data stream. In a non-active network, the end system has to receive all these data streams, mix them together, and play the result to the user. In an active network, though, the audio streams can be aggregated as they traverse the network and the end-system is provided with a single mixed audio stream.

C. Data Aggregation

Aggregation of many-to-many multicast data streams in the network brings benefits to the network as well as to the end system.

From the view point of the network, the total amount of traffic is reduced, since only one aggregated data stream has to be delivered to a receiver. Even though the aggregated data stream can potentially be larger than any single data stream, it will always be less than the sum of all single data stream. Also, there can be a reduction in transmission frequency for aggregated data streams. For example, in the battlefield information system, it might be necessary to have high frequency updates for the status of an individual soldier who moves around quickly. The centroid of a group, though, moves at a slower rate and therefore needs to be updated less frequently. A qualitative analysis of this result can be found in Section V.

From the view point of the end system, the received data can be seen as a single unicast connection from the group and displayed directly without the need for complex aggregation processing. In the audio application, for example, the mixed audio stream differs not from a unicast audio stream, except that the audio samples are an aggregate from several sources. This reduces the processing requirements and simplifies the end system application development.

There are several issues that have to be addressed in order to achieve efficient data aggregation. For one, it has to be possible to aggregate the data that is transmitted. In many cases, aggregation can be achieved by downscaling information (e.g., scale change for geographic information) or generating an overlay of different data streams (e.g., mixing of audio). Status information can often be aggregated by simple arithmetic and boolean functions. However, certain data, e.g., text messages, cannot be scaled or aggregated effectively without losing crucial components of the data. In such a case, the information can be concatenated and transmitted unchanged, losing the benefits of reduction in bandwidth and processing requirements.

III. Hierarchy of Source-based Multicast Sessions

In a realistic environment, different observers need different levels of aggregation detail and an individual observer might want to change the level of detail dynamically. To accommodate these requirements, we propose a hierarchy of multicast sessions that provide different levels of aggregation to the end system. Each session is augmented by control information that allows the user to change to a higher or lower level of detail if necessary.

A. Hierarchy Layout

Each layer in the hierarchy represents a different aggregation level. The lowest layer, layer 0, are the data sources (i.e., soldiers, telephones) that send their unaggregated data. Each node in layer 1 aggregates multiple layer 0 sources to a new stream. This stream is sent upwards to layer 2, where it is aggregated with other layer 1 streams. This continues up to the root node. In general, a layer i node aggregates streams from layer $i-1$ and sends it to layer $i+1$. Figure 3 shows this concept.

An observer who wants to get information from a node on a certain layer, can directly connect to that node and receive its aggregated data stream. If more detail is required, the observer can connect to a child of the current node. If less detailed information is required, the observer can move up to the parent of the current node.

Since there are many observers present at any time, it is of course possible that multiple observers are connected to a certain node. To make this scalable, any

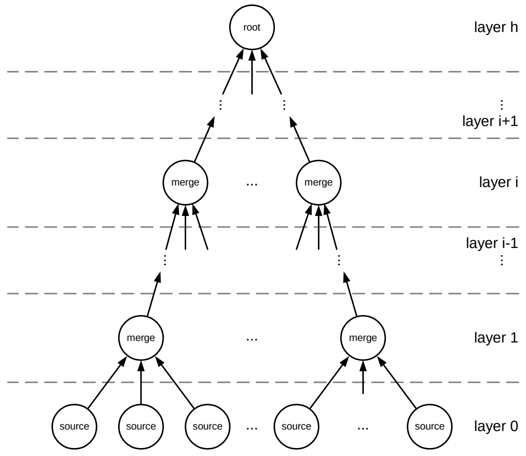

Fig. 3. Hierarchy of Data Aggregation.

node offers its data as a mutlicast session, for which it acts as the source. Any observer can subscribe to this multicast session using well established mutlicast schemes (e.g., by using mutlicast routers that exchange IGMP [4] messages and route using MOSPF [5]).

B. Session Control

The scheme described above requires that nodes are configured to form the tree and aggregate the right set of lower layer streams. Thus, all nodes other than leaves need to know their children at start time. Once the children are known, the node can subscribe to their multicast sessions and aggregate the information.

The control infomation that is distributed with each session needs to contain two components. First, the list of children and their respective session identifiers is included so that an observer can receive data from a lower layer. Second, the session identifier of the parent of the current node is included to allows an observer to step to a higher aggregation layer.

C. Naming Issues

For scalability reasons, the control data of a session cannot contain the names/identifiers of all children, grandchildren, etc., since this would lead to an exponential growth in control information. As a result, at a high level in the hierarchy, say layer i, the contributing sources of a data stream are not known. If an observer wants to step towards a source x, he does not know which child of that node contains the aggregated data stream from x.

To solve this problem, we add a naming scheme to the hierarchy. Each node is identified by its name and the concatenation of names of nodes in the path from the root to the node. This results in a hierarchical naming scheme similar to that used in the Domain Name Service [6] in the Internet or that of class names in object oriented programming languages, like Java [7]. Thus, the full name of leaf x contains the list of higher layer nodes that lead to x, which allows the observer to easily navigate through the tree. For convenience purposes and to abbreviate long names, certain intermediary nodes can be aliased with unique names.

IV. Aggregation Algorithm

The algorithm we discuss here aggregates several data streams on an active node. Each stream is considered to be a periodic sequence of unreliable datagrams. The basic steps consist of the buffering of packets until they can be merged with other packets, the merging of the data, and the transmission of the result. Also, there is a mechanism to detect packet loss and avoid indefinite waiting. Our aggregation algorithm is similar to Concast, described in [8]. While Concast is an implementation of a general purpose many-to-one communications paradigm, it does not specifically consider timeout issues due to packet loss and thus does not lend itself well to a real-time environment.

Consider a router that has to merge k data streams, $s_1 \ldots s_k$, to a new data stream s_a. Assume we are given an aggregation function F that generates a packet p_{s_a} from packets $p_{s_1}, \ldots p_{s_k}$ ($p_{s_a} = F(p_{s_1}, \ldots p_{s_k})$). If a packet p_{s_i} has not been received during a period and the timeout is triggered after $t_{timeout}$, the merging function F can use either an older data packet of stream s_i or use the neutral element 0_F as a placeholder. Let us also assume for now that all sources send packets of the same size (same amount of information or samples) and with the same period. The resulting procedure for buffering and merging packets is shown in detail in Figure 4. There are four parts to the algorithm:

1. *Part I (lines 7-12): A packet arrives and is stored in the buffer.* This requires that the buffer slot for that stream is not yet used.
2. *Part II (lines 13-26): A packet arrives and its buffer slot is already taken.* This happens, when the algorithm was waiting for packets from other streams that were lost or delayed. With the arrival of a second packet from a stream, we know that it is time to send the aggregated packet. Empty buffer slots are filled with null packets and the data is aggregated and sent.
3. *Part III (lines 27-34): All buffer slots are filled.* In this case, one packet from each flow is available and we can aggregate the data and send out the result.
4. *Part IV (lines 36-46): A timeout occurred.* In this case, missing packets are replaced by null packets and the aggregated result is sent.

```
1:  initialization:
2:  clear $b$
3:  $bcount \leftarrow 0$
4:  set $timer$ to $\infty$
5:
6:  receive packet $p_{s_i}$ from stream $s_i$
7:  if $b[i]$ is empty then
8:    $b[i] \leftarrow p_{s_i}$ {store packet in buffer}
9:    $bcount \leftarrow bcount + 1$ {increase buffer counter}
10:   if $bcount = 1$ then
11:     set $timer$ to $t_{timeout}$ {set timer if this is the first packet in
        the buffer}
12:   end if
13: else
14:   {there is already a packet from source $s_i$}
15:   for $j = 1$ to $k$ do
16:     if $b[j]$ is empty then
17:       $b[j] \leftarrow 0_F$ {fill empty buffer slots with neutral elements}
18:     end if
19:   end for
20:   $p_{s_a} \leftarrow F(b)$ {merge packets}
21:   send $p_{s_a}$
22:   clear $b$
23:   $b[i] \leftarrow p_{s_i}$ {store packet that was just received in buffer}
24:   $bcount \leftarrow 1$ {adjust buffer counter}
25:   set $timer$ to $t_{timeout}$ {set timer (since this is the first packet in
      the buffer)}
26: end if
27: if $bcount = n$ then
28:   {there is one packet from each source in the buffer}
29:   $p_{s_a} \leftarrow F(b)$ {merge packets}
30:   send $p_{s_a}$
31:   clear $b$
32:   $bcount \leftarrow 0$ {adjust buffer counter}
33:   set $timer$ to $\infty$
34: end if
35:
36: on $timer = 0$ do: {timeout occurred}
37: for $j = 1$ to $n$ do
38:   if $b[j]$ is empty then
39:     $b[j] \leftarrow 0_F$ {fill empty buffer slots with neutral elements}
40:   end if
41: end for
42: $p_{s_a} \leftarrow F(b)$ {merge packets}
43: send $p_{s_a}$
44: clear $b$
45: $bcount \leftarrow 0$ {adjust buffer counter}
46: set $timer$ to $\infty$
```

Fig. 4. Packet Merging Algorithm.

The packets are stored in the buffer array, b, that has n slots. The variable *bcount* keeps track of the number of valid buffer entries. The *timer* is set to $t_{timeout}$ every time the first packet is put into the empty buffer. This way, no packet is ever stored longer than $t_{timeout}$. The *timer* is cleared (set to ∞) when the buffer is cleared.

V. Evaluation

To show the effectiveness of Aggregated Hierarchical Multicast, we first look at the reduction of link bandwidth for a given scenario. Second, we analyze the complexity and correctness of the aggregation algorithm.

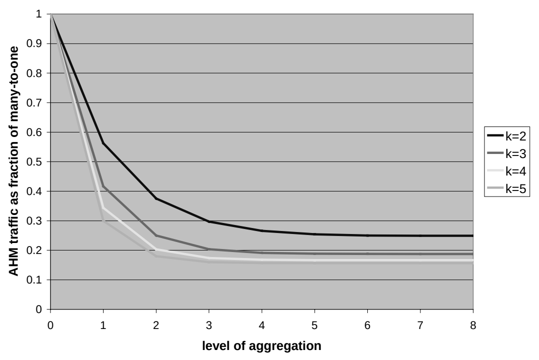

Fig. 5. Traffic comparison between Aggregated Hierarchical Multicast and many-to-one communication.

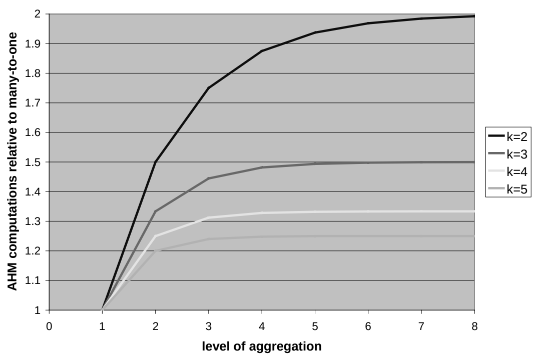

Fig. 6. Computation comparison between Aggregated Hierarchical Multicast and many-to-one communication.

And third, we show measurements from a prototype implementation of the audio conferencing application.

A. Bandwidth and Computation

To analyze the benfits of aggregation in the network, we compare Aggregated Hierarchical Multicast with traditional many-to-one communication, where the receiver aggregates all data. Assume a balanced tree of height h with nodes of degree k. Say the receiver is the root of the tree and it wants to observe all leaves of a subtree with height l. In Aggregated Hierarchical Multicast, each leaf of the subtree sends one message to its parent, which is further aggregated until it reaches the root of the subtree. In traditional many-to-one communication, all leaves have to send a message to the root. Figure 5 shows the fraction of traffic that is necessary to observe a subtree on various levels compared to many-to-one communication. Even for small levels of aggregation ($l \geq 2$), the total traffic is reduced by $60 - 80\%$.

However, aggregation in the network has its price. Each data stream is aggregated possibly multiple times on the way to its destination compared to a many-to-one scheme, where aggregation happens only once at the destination. Figure 6 shows that for small node degrees ($k = 2$) potentially twice as many aggregation computations are necessary. Higher degree trees have less of a computational overhead. Considering that higher degree trees also require fewer transmissions, they are more favorable for Aggregated Hierarchical Multicast.

B. Aggregation Algorithm

Two parts of the aggregation algorithm contribute to its computational complexity. One is the per packet processing and the other is the aggregation of the stored packets. As for the per packet processing, the complexity is constant ($O(1)$). The packet merging, is $O(k)$, since we have to go through all stored packets. If we do an amortized analysis, though, we can add a credit for each packet received, which is used for the $O(k)$ computation. This results in a constant $O(1)$ amortized complexity.

The correctness proof of the algorithm is beyond the scope of this paper, but can be found in detail in [9]. The basic result is that the algorithm operates correctly if the maximum jitter j_{max} is bound to $j_{max} < \frac{1}{4k} \cdot \Delta t$ and the timeout is set to $t_{timeout} = (1 - \frac{1}{k}) \cdot \Delta t + 2 \cdot j_{max}$, where Δt is the time between packets from one source.

C. Measurements

To evaluate the performance of the described algorithm in a real application, we have implemented a prototype of the audio conferencing application described above. The prototype aggregates PCM μ-law encoded data streams and is limited only insofar that it uses unicast connections between nodes and does not implement all control features. All measurements were performed on a heterogenous set of machines with Pentium processors running the NetBSD and Linux operating systems.

C.1 Processing Delay

The processing delay is the time between the arrival of the packet that triggers aggregation and the transmission of the aggregated packet. It varies depending on the size of the packet, the aggregation complexity, and the number of packets in the buffer. Figure 7 shows the processing times for 1000 aggregations for degrees of $k = 1 \ldots 5$ and a packet size of 400 bytes. The processing delay is relatively uniform over the number of packets. Higher degree aggregations require slightly more processing.

To show the effects of varying node degree and packet size, Figure 8 shows the median delay for packets of

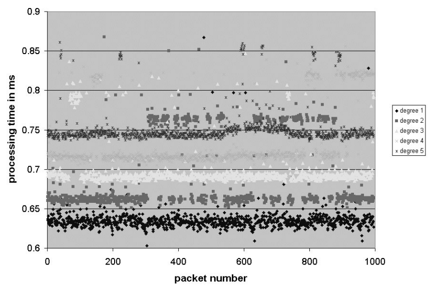

Fig. 7. Processing Delay for 400 byte Packets and Node Degrees of $k = 1 \ldots 5$.

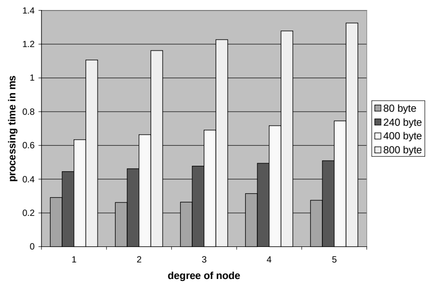

Fig. 8. Median Processing Delay for Varying Packet Sizes and Node Degrees.

sizes 80, 240, 400, and 800 byte. The degree of the node varies again from 1 to 5. The increase in processing time is proportional to the packet size, as should be expected. Over the range of data, there is also an increase in processing time due to higher node degrees, but it is not quite as significant.

C.2 Jitter

The jitter that is introduced by the aggregation affects the correctness of the aggregation and needs to be bound, as discussed in Section V-B. The jitter for an audio data source is plotted in Figure 9. It can be seen that the average jitter is limited to 10 μs. Looking at the jitter after traversing five aggregation steps, Figure 10 shows that it has increased to about 50 μs. This indicates that jitter does increase, but 50 μs is well below 1% of the interpacket time and therefore does not pose any problems.

These measurements indicate that the aggregation algorithm is robust and performs well over a wide range of aggregation levels, node degrees, and packet sizes.

VI. Summary

We have presented a many-to-many communication scheme that uses data aggregation to scale the view of a receiver to the required accuracy level. By reducing the overall data traffic, while increasing the computational requirements only slightly, Aggregated Hierarchical Multicast improves the scalability of many-to-many communication. Measurements on the prototype implementation have shown that the aggregation algorithm is robust and performs efficiently.

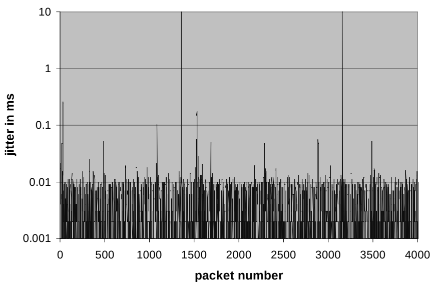

Fig. 9. Jitter for 400 byte Packets at Source.

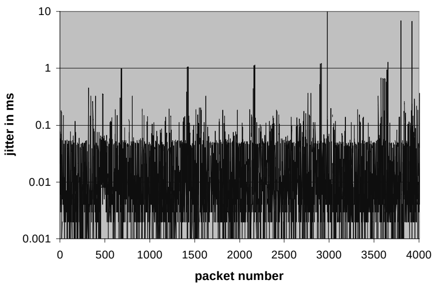

Fig. 10. Jitter for 400 byte Packets after 4 Aggregation Steps.

References

[1] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, Jan. 1997.

[2] Andrew T. Campbell, Herman G. De Meer, Michael E. Kounavis, Kazuho Miki, John B. Vincente, and Daniel Villela, "A survey of programmable networks," *Computer Communication Review*, vol. 29, no. 2, pp. 7–23, Apr. 1999.

[3] Larry Peterson, ed., "NodeOS interface specification," Tech. Rep., AN Node OS Working Group, Feb. 1999.

[4] Steve Deering, "Host extensions for IP multicasting," RFC 1112, Stanford University, Aug. 1989.

[5] John Moy, "MOSPF: Analysis and experience," RFC 1585, Network Working Group, Mar. 1994.

[6] Paul Mockapetris, "Domain names - implementation and specification," RFC 1035, Network Working Group, Nov. 1987.

[7] Bill Joy, Guy Steele, James Gosling, and Gilad Bracha, *The Java Language Specification*, The Java Series. Addison-Wesley, second edition, June 2000.

[8] Kenneth L. Calvert, James Griffioen, Billy Mullins, Amit Sehgal, and Su Wen, "Concast: Design and implementation of an active network service," *IEEE Journal on Selected Areas of Communications*, vol. 19, no. 3, pp. 404–409, Mar. 2001.

[9] Tilman Wolf and Sumi Yunsun Choi, "Aggregated hierarchical multicast for active networks," Tech. Rep. WUCS-01-05, Department of Computer Science, Washington University in St. Louis, Feb. 2001.