Gathercast with Active Networks

Yu He
Department of Computer Science
University of Southern California
yuhe@usc.edu

Cauligi S. Raghavendra
Department of Computer Science
University of Southern California
raghu@usc.edu

Steven Berson
Information Sciences Institute
University of Southern California
berson@isi.edu

Abstract

Small packets constitute a large fraction of packets in the Internet. Gathercast is a network layer service that combines small packets going to the same destination into a single packet, reducing the number of packets in the network and saving computations at intermediate routers. In this paper, we propose a new model for gathercast using active network techniques. With this model, gathercast is built throughout the network with low overhead, service parameters can be dynamically adjusted and end-user applications obtain benefits from gathercast without knowing its existence. We also present an implementation of the active gathercast.

1. Introduction

Small packets constitute a large fraction of packets in the Internet. Recent studies show that there are 60 percent of packets in the Internet that are less than 44 bytes [12]. One example of small packets is the ACKs generated by the clients of web servers, which are just 40 bytes long. Another example is the ICP (Internet Cache Protocol) queries generated among web proxies, which are about 60 bytes in length. Gathercast ([6] and [5]) is a network layer service that combines small packets going to the same destination into a single packet. When combined packets arrive at the destination, they are broken into the original small packets. By combining small packets, gathercast reduces the number of packets seen by the intermediate routers in the network. Since each packet requires the same processing overhead at every router irrespective of its size, gathercast saves computation at routers. Simulation studies have shown that gathercast significantly improves network utilization and end-user application performances [6].

In [6], the authors proposed to use transformation tunnels between aggregating routers and destinations to realize gathercast. They discussed signaling protocols between end nodes that are used to build the gathercast service throughout the network. Their study was mainly about a gathercast mechanism using multicast service, because branch points on a multicast tree are naturally good places for aggregations. In this paper, we will justify an active network [14] approach to a general-purpose gathercast that does not depend on multicast.

There are several parameters that effect performance achievable with gathercast. They include size of packets to combine, how many packets to combine, waiting time of small packets in buffers, and what points in the network to do this gathering. There have been general guidelines about how to choose these parameters. For example, defining small packets as 100 bytes or less and combining just two packets into a single one have shown improved performance [6]. There is a tradeoff in choosing the waiting time: the longer the delay, the more small packets can be combined, but the more latency the packets will suffer. Besides, long latencies could trigger off retransmissions of TCP packets. So the waiting time generally should be small. As for the placement of gathercast service, it is observed that the further to senders, the more small packets can be seen in the network; the further to receivers (destinations), the less packets (after aggregations) are in the network. Thus the gathercast service should generally be placed in the middle of the network.

Note that the above are only general guidelines. It is unlikely that fixed values will always achieve the best performance of gathercast. For example, routers on different places of the network should have different parameters since they see different traffic. When network states change, these parameters should also change accordingly based on current conditions of the network. Thus these values should be determined dynamically.

Due to its dynamic and programmable property, active networking [14] is ideally suited to support gathercast. There is an active network service known as concast [10] that also combines packets with same destinations. But as shown in Section 2, concast takes a model with an overhead similar to the one in [6]. In this paper we propose a new model with low overhead for active gathercast. With our model, service parameters of gathercast can adapt to network changes, and applications obtain benefits from gathercast without knowing its existence.

This paper is organized as follows. In Section 2, some previous work on gathercast is discussed. Then a new model for active gathercast is described in Section 3. An implementation of the active gathercast on top of an active execution environment (the ASP with NETIOD) is presented in Section 4. Finally a conclusion is given in Section 5.

2. Previous work on Gathercast

The idea of combining small packets is not new. There has been a transport multiplexing protocol that aggregates small transport-layer packets ([13] and [11]). This protocol is designed to combine large numbers of small data packets that are generated when many Telnet or Rlogin sessions are connected to a few hosts on the network. In this method, aggregation is done only at end nodes: small data packets are combined into a large IP datagram at sender side; the receiver then breaks the large packet and re-generates the small packets. All aggregated packets have the same source and destination addresses. This method take no measure to deal with small packets in the network. It can be regarded as a special case of the gathercast on the end nodes.

To establish gathercast service throughout the network, the authors in [6] assume that there is a multicast group built in advance. This multicast group consists of multiple senders whose packets are to be gathered. Then they present two ways to establish the service. In the first method, the receiver sends a multicast message to all senders. Then the senders bounce back the messages. The intermediate routers then build gathercast service according to the bounced messages that includes service parameters. This way, the services are established along the unicast routes from senders to the corresponding receiver. The second method is to make use of the existing multicast tree between the receiver and the multiple senders. The receiver multicasts an extended IGMP message to all senders. When the intermediate multicast routers get the message that contains service parameters, they build the services along the reverse-multicast tree. In this method, the multicast routing daemon needs to be modified to intercept the new IGMP message and generate the reverse-multicast routes. Both approaches need a multicast group.

One previous approach to gathercast using active network techniques is the concast service ([10] and [1]). To establish service, concast does not need any multicast protocol support. Instead, it defines its own group, called concast group. This group consists of multiple senders. There is also a signaling protocol for service establishment. Under this signaling protocol, the receiver first initiates the group with an aggregation instruction that includes service parameters. Then senders dynamically join or leave this group by signaling the network. The network contains some predefined concast-capable routers, which respond to the signaling, install the aggregation instruction obtained from the receiver, and build a concast tree with senders as leaves and the receiver as the root and themselves as inner nodes.

In above models ([6] and [10]), service (gathercast or concast) parameters are provided by end nodes (receivers, for example). Routers use these values to establish transformation tunnels. All routers equipped with the service hold the same set of parameters. These models do not provide a flexible way to tune service parameters for the best performance. Besides, there are times when end nodes have no good knowledge of these parameters. We believe that these models are practical only to specific applications where end nodes are aware of service parameters. However, gathercast service should be beneficial no matter what the application is. It should be useful to build gathercast as a general-purpose network service. In this case, routers have better knowledge to dynamically determine these parameters than end nodes since it are routers that actually see passing traffic.

Furthermore, these models require signaling to build and maintain a group (a multicast or a concast group). The main advantage of establishing services using a group is that branch points in the group tree (a multicast tree or a concast tree) are naturally good places for gathercast, thus avoiding unnecessary services at other places. This is beneficial when gathercast is used along with some existing multicast group service since the group is in place anyway.

In general, building and maintaining such groups incur significant overhead. It will be more useful to opportunistically combine small packets into larger ones. Our goal is to build the gathercast as a general-purpose networking service without first establishing a multicast group. We develop a new model to achieve this using active networks that is explained next.

3. Gathercast with Active Networks

In this section, we develop a new model for using active networks to support general-purpose gathercast. This is different from previous models in the sense that we do not need a multicast group to be established first, which leads to a service with better scalability. With this model, gathercast service can be invoked at any node in the network, its parameters are dynamically changeable. For clarification, we first discuss the implication of no multicast group to gathercast before introducing the active model.

3.1. Independent Aggregators and Gatherers

Gathercast service has two functional parts: aggregator and gatherer. The *aggregator* combines small packets into a single packet. The *gatherer* decompose the combined packet and restore the original small packets. There is a tunnel between the aggregator and the gatherer so that the intermediate routers just forward the combined packets as normal packets. The combined packet is called a *gathercast packet*.

In previous models, the aggregators and the gatherers are built by signaling messages between senders and the receivers. With these messages, the services are established at routers roughly corresponding to branch points at multicast trees. Besides, signaling messages also inform routers of the parameters chosen by end nodes. Thus the establishment of the aggregators and the gatherers depends on who the senders and the receivers are.

When there is no such multicast tree in the network, the aggregators and gatherers do not have to know the senders and the receivers in order to operate. To aggregators, any packets, no matter who sends them, can be combined as long as they meet the combining rules. To gatherers, any gathercast packet they get, no matter who the aggregator was, is decomposed into original packets. The service parameters can be dynamically determined by routers. Thus there is no need for the signaling messages between the senders and the receivers. End-user applications associated with the senders and receivers obtain benefits from gathercast without the knowledge of its existence. Besides, the scalability of the gathercast becomes not limited by the underlying multicast service.

3.2. Active Gathercast Model

In an active network, a network service is dynamically invoked when specific packets arrive. For other packets, the network functions as a normal IP network. This property adds great flexibility to the placement and establishment of gathercast.

In gathercast, a packet could be combined multiple times along its way to the destination. But with the increase of the aggregation times, the packet latency also adds up. Since TCP is sensitive to the delay, to get the minimal delay, we

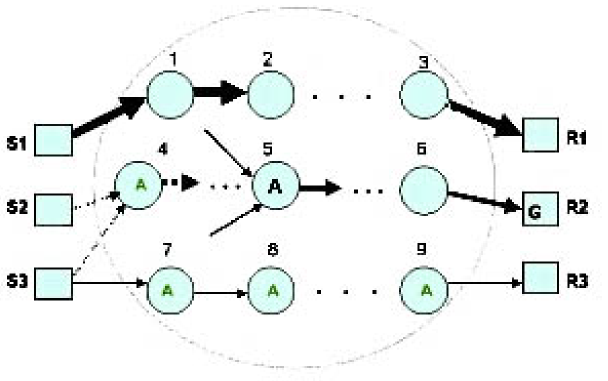

Figure 1. Gathercast Model in Active Network

choose to allow packets to be combined only once. This also simplifies our model as shown below.

We define an IPPROTO_GATHERCAST protocol number to be reserved for gathercast. This protocol number is used to discriminate gathercast packets from normal ones. The gatherers are put onto hosts since every end node is a potential receiver. The gatherers are installed whenever they receive packets containing the protocol number IPPROTO_GATHERCAST.

The aggregator's installation is triggered by a suitable passing packet. It is invoked if the passing packet meets the following two conditions: 1) the packet meets the length limit predefined in the aggregator; 2) the packet does not contain the protocol number of IPPROTO_GATHERCAST. These two conditions avoid the establishment of aggregators on routers that only see large packets or combined packets.

It is possible that a small packet triggers an installation of the aggregator but does not get combined because it does not meet the combining condition. This kind of aggregator brings unnecessary delay to small packets. To reduce the number of aggregators, a timer is enforced on each aggregator. If there is no combination happening for some predefined threshold of consecutive small packets arrivals, the aggregator is uninstalled and the timer is started. During the interval of the timer passing packets meeting the above two conditions do not trigger aggregators installation. But after the timer expires, the aggregator can be triggered again. In this way, the aggregator installation dynamically reflects the changes of the network and only those routers that frequently see a significant number of combinable small packets keep the aggregators. Figure 1 shows how the model works.

In Figure 1, the large arrows represent traffic with big packets; the small arrows are traffic with small packets; the medium arrows are traffic generated by gathercast packets. The arrows with solid lines represent continuous traffic, while the arrows with dashed lines are intermittent traffic. All hosts (S1-S3 and R1-R3) could install a gatherer. But only R2 sees the gathercast traffic. So R2 is the only host installing a gatherer. Routers 1, 2, and 3 see big packets and thus no aggregators are installed. Router 6 sees gathercast packets and thus is not installed with an aggregator. Routers 7, 8 and 9 see small packets, so periodically the aggregators are installed on these routers. But most of the time aggregators are uninstalled on these routers since small packets arriving at these nodes are not combinable. Router 4 see small packets occasionally that are combinable and an aggregator is periodically installed. But since the threshold of the number of passing small packets can not be reached during the timer interval, the aggregator is uninstalled most of the time. Router 5 is in the middle of the network where there are more packets. It frequently sees combinable small packets, and it keeps the aggregator for a long period of time.

Besides the dynamic placement, this model enables routers to adapt other service parameters to network changes. Routers can obtain statistical data from passing small packets by simply checking their IP headers. The available information includes the average size of small packets passed during a time interval, the most frequent protocol number, and the number of combinable packets from different sources. These data reflect most recent network conditions. Routers thus can take advantage of the statistics to estimate proper service parameters for next time interval.

4. Gathercast Implementation with ASP and NETIOD

We implemented gathercast on an active environment including the ASP EE and NETIOD.

4.1. Outline of ASP and NETIOD

The ASP Execution Environment [7] is developed using Java by USC/ISI and follows the node reference model [3]. It runs on top of a node OS and is the operating environment for active applications (AAs, namely, active network services). Its features include dynamically loading of AA code, security and resource protection, network I/O, timing service, and controlled inter-AA communication. With dynamical code-loading, the AA can be loaded out-of-band locally, from neighbors or from remote code servers. The timing service can be associated with the state information in AAs. With this service, AAs are able to maintain their state information as soft state. The node structure with ASP EE is shown in Figure 2.

NETIOD (network I/O daemon) [9] is developed using C by USC/ISI and follows the node OS reference model

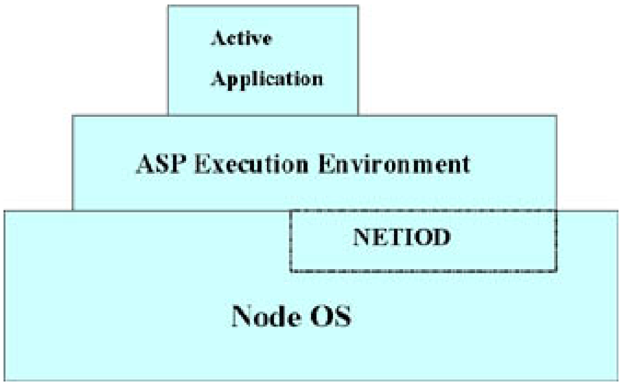

Figure 2. Active Node Architecture with ASP and NETIOD

[4]. NETIOD directly sits on top of Unix operating system. It is logically a part of the Node OS. Although currently ASP can handle some network I/O by itself, NETIOD is able to meet all network-I/O requests from all EEs (including other EEs than ASP). With NETIOD, AAs submit their specific requests for input and output packets to the ASP EE, which then uses the NETIOD to open specific input and output channels. When the required packets arrive at the input channel, the AA code is loaded by the ASP EE if it has not already been in the node. The corresponding AA is then activated, processes the incoming packets and sends the output packets via the output channel.

Currently the latest versions of ASP (1.5+) and NETIOD (1.0b) are used in our implementation.

4.2. Gathercast AA

The gathercast service is implemented as an AA. There are two parts for the gathercast AA - aggregator and gatherer.

(1) Aggregator

The aggregator looks for small IP packets meeting its specific length limit and having a protocol number other than IPPROTO_GATHERCAST. The aggregator requests that the ASP EE open an input channel for these packets. When NETIOD gets the incoming packets, it checks if they meet the request of the AA. If not, the packets are not diverted to AA but put back to normal processing. If the incoming packets meet the request, they are put into the input channel for the EE, which causes the aggregator to be loaded.

The aggregator is invoked with a delay parameter. It processes the packets coming from the input channel according to the following algorithm.

For packet p:

```
if(delayed_packet exists) {
    if(p and delayed packet can be combined) {
        clear the delay timer;
        combine and send the gathercast packet;
    }
    else {
        send delayed_packet without change;
        clear the delay timer;
        delayed_packet = p;
        set the delay timer;
    }
}
else {
    delayed_packet = p;
    set the delay timer;
}
```

From the above algorithm, the aggregator currently delays at most one packet and only combines two qualified packets. This way, the delay is kept to a minimum. Currently two packets can be combined if they are from different sources with the same destination address. This condition is necessary because there would be an ACK compression problem for TCP traffic if ACKs from the same source got aggregated. The combined packet is given an IP header with the destination address and a specific protocol number - IPPROTO_GATHERCAST. A tunnel is thus built between the aggregator and the destination. Intermediate routers forward the gathercast packets without any knowledge of the service.

Note that the delayed_packet is maintained as a soft state at aggregators. Its associated timer is the delay timer. When this timer is expired, the delayed packet is sent out without any change.

(2) Gatherer

The gatherer requests an input channel from NETIOD for IP packets with a protocol number of IPPROTO_GATHERCAST. When such packets arrive, the gatherer is loaded. The gathercast packets are then decomposed and the original small IP packets are restored.

Note that neither NETIOD nor the gathercast AA needs to check the IP packet payloads for the service. All operations are carried out on IP headers. Although gathercast is implemented on top of the ASP EE that is a Java environment, there is no difficulty in re-implementing it as a router module directly on top of the machine OS. We believe that gathercast with our model is deployable in the Internet.

5. Conclusion and Future Plan

In this paper, we proposed an active network model for gathercast. This model does not need an underlying multicast group to establish the service. Thus the group overhead is avoided and the service's scalability is not limited by multicast. With this model, gathercast can dynamically adapt its service parameters to network changes. End-user applications obtain benefits from gathercast without the knowledge of its existence. A prototype implementation of gathercast on top of ASP and NETIOD was presented.

We have deployed our gathercast AA on several UNIX machines. We tested its impact on ICMP, UDP and TCP traffic and observed network packets reduction after involved small IP packets get combined. To get meaningful quantities, we need to deploy the gathercast service in a significantly large network. The ABone active network testbed ([8] and [2]) currently has around 100 registered nodes. It should be a good environment to get real performance measurements for the active gathercast.

References

[1] *http://protocols.netlab.uky.edu/ acast/concast.html.*

[2] Abone coordination center (abocc). *http://www.isi.edu/abone/abocc.html.*

[3] active network architecture working group. Architectural framework for active networks. *http://www.cc.gatech.edu/projects/canes/papers/arch-1-0.ps.gz*, July 1999.

[4] active network node operating system working group. Active node operating system interface specification. *http://www.cs.princeton.edu/nsg/papers/nodeos.ps*, January 2000.

[5] B. R. Badrinath and P. Sudame. Gathercast: an efficient multi-point to point aggregation mechanism in ip networks. *Rutgers University CS Technical Report DCS-TR-362*, 1998.

[6] B. R. Badrinath and P. Sudame. Gathercast: the design and implementation of a programmable aggregation mechanism for the internet. *ICNP*, 1998.

[7] S. Berson, R. Braden, T. Faber, and B. Lindell. The asp ee: An active network execution environment. *DARPA Active Networks Conference and Exposition*, May 2002.

[8] S. Berson, R. Braden, and L. Ricciulli. Introduction to the abone. *http://www.isi.edu/abone/DOCUMENTS/ABarch/.*

[9] S. Berson, S. Dawson, and R. Braden. Evolution of active networks testbed, accepted for publication. *DARPA Active Networks Conference and Exposition*, May 2002.

[10] K. L. Calvert, J. Griffioen, B. Mullins, A. Sehgal, and S. Wen. Concast: Design and implementation of an active network service. *IEEE Journal on Selected Area in Communications (JSAC)*, 19(3), March 2001.

[11] D. Cameron, D. Crocker, D. Cohen, and J. Postel. rfc 1692 transport multiplexing protocol (tmux). August 1994.

[12] K. Claffy, G. Miller, and K. Thompson. The nature of the beast: Recent traffic measurements from an internet backbone. *INET*, 1999.

[13] D. Cohen and J. Postel. Multiplexing protcol. *IEN 90*, May 1979.

[14] D. Tenenhouse, J. Smith, W. Sincoskie, D. Wetherall, and G. Minden. A survey of active network research. *IEEE Communication Magazine*, pages 80–86, January 1997.