TCP-Peachtree: A Multicast Transport Protocol for Satellite IP Networks

Ian F. Akyildiz, *Fellow, IEEE,* and Jian Fang

***Abstract*—In this paper, a reliable multicast transport protocol TCP-Peachtree is proposed for satellite Internet protocol (IP) networks. In addition to the acknowledgment implosion and scalability problems in terrestrial wirelined networks, satellite multicasting has additional problems, i.e., different multicast topology, different type of congestion control problems, and low bandwidth feedback link. In TCP-Peachtree, the modified B+ tree logical hierarchical structure is used to form dynamic multicast groups. Local error recovery and acknowledgment (ACK) aggregations are performed within each subgroup and also via logical subgroups. In order to avoid the overall performance degradation caused by some worst receivers, a local relay scheme is designed. Two new algorithms, jump start and quick recovery, which are based on the usage of a type of low-priority segments called NIL segments, are proposed for congestion control. NIL segments are used to probe the availability of network resources and also for error recovery. The delayed selective acknowledgment (SACK) scheme is adopted to address the bandwidth asymmetry problems and a hold state is developed to address persistent fades. The simulation results show that the congestion control algorithms of TCP-Peachtree outperform the TCP-NewReno when combined with our hierarchical groups and improve the throughput performance during rain fades. It is also shown that TCP-Peachtree achieves fairness and is very highly scalability.**

***Index Terms*—Acknowledgment (ACK) aggregation, congestion control, local recovery, reliable multicast, satellite Internet protocol (IP) networks.**

I. Introduction

SATELLITE networks will play a crucial role in the global infrastructure of the Internet. They do not only provide global coverage, but also are capable of sustaining high bandwidth levels and supporting flexible and scalable network configurations. Currently, two thirds of the world still does not have a wired network infrastructure. Locally built networks or individual hosts can be connected to the rest of the world via satellites by simply installing satellite interfaces. Satellite networks can also be used as a backup for existing networks, e.g., in case of congestions or link failures, traffic can be rerouted through satellites.

Multicasting provides an efficient way of disseminating data from a sender to a group of receivers. Instead of sending a separate copy of the data to each individual receiver, the sender just transmits a single copy to all receivers. In this case, a multicast tree is set up in the network, where the sender is the root and the receivers are the leaf nodes. Data generated by the sender flows through the multicast tree, traversing each tree edge exactly once [24].

A large variety of reliable multicast protocols [12], [15], [20], [24], [25], [28], [29] have been proposed for the terrestrial wirelined networks, where packet losses occur mostly due to congestions. These protocols are usually classified into two groups: *sender-initiated* protocols, which use acknowledgments (ACKs), and *receiver-initiated* protocols, which use negative ACKs (NAKs). By this classification, reliable multicast transport protocol (RMTP) [24] is a typical sender-initiated protocol, in which receivers form local groups for *local error recovery*. Scalable reliable multicast (SRM) [12] is a typical receiver-initiated protocol, retransmission is performed in the entire group. Tree-based multicast transport protocol (TMTP) [29] uses both ACKs and NAKs. Multicast transfer control protocol (MTCP) [25] uses error bitmap in ACKs which is similar to TCP selective acknowledgment (SACK) option. However, MTCP introduces high overhead for large congestion window.

Satellite networks have high *bit-error rate* (BER), *long propagation delays*, and *asymmetrical bandwidth* [2]. These characteristics distinguish satellite multicasting from multicasting in terrestrial wirelined networks. In addition to the *acknowledgment implosion* and *scalability problems* in terrestrial wirelined networks, satellite multicasting has the following additional problems.

- *Different Multicast Topology*: In terrestrial multicast networks, the receivers may be located in different places, connected by intermediate routers, and may be several hops distant from the sender. Usually, a multicast tree is created with the help of intermediate routers. While in satellite multicast networks, all receivers receive packets directly from the satellite and there are no intermediate routers between the satellite and the receivers. On the other hand, the receivers send acknowledgments directly to the satellite. In other words, the receivers are only one hop away from the satellite. Therefore, there is no physical hierarchy between the satellite and the receivers. Consequently, the multicast schemes [12], [15], [20], [24], [25], [28], [29] that are based on multicast trees and are developed for terrestrial multicast networks cannot be applied in satellite multicasting. Furthermore, each receiver may experience different data losses for satellite multicasting and there is no guarantee that some receivers may always have better channel conditions. As a result, some mostly

Manuscript received December 15, 2002; revised July 1, 2003 and September 20, 2003. This work was supported in part by the National Aeronautics and Space Administration-Glenn Research Center (NASA-GRC) under Project NAG3-2585.

The authors are with the Broadband and Wireless Networking Laboratory, School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: ian@ece.gatech.edu; jfang@ece.gatech.edu).

Digital Object Identifier 10.1109/JSAC.2003.819991

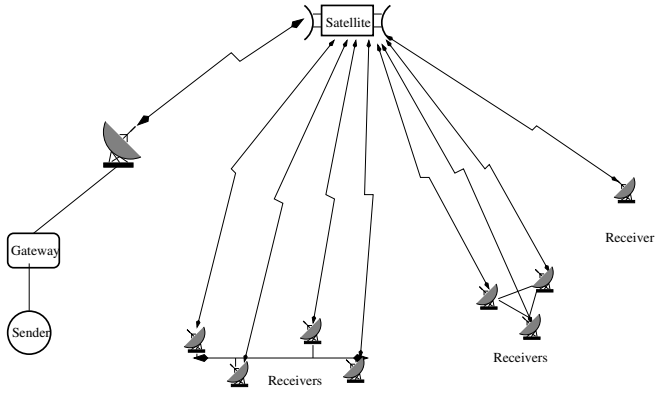

Fig. 1. Satellite multicast scenario.

used terrestrial multicast schemes such as selecting some special servers or some intermediate routers as repairers [15], [20] may not be used here.

- *Congestion Control Problems*: TCP problems in satellite IP networks and related research issues are discussed in [2]. As pointed out in [2], the congestion control for satellite IP networks is difficult due to long propagation delay, high BER, and asymmetrical bandwidth for unicast applications. For satellite multicasting, these problems become more complicated. Traditional TCP schemes usually perform poor in these situations [22]. Furthermore, satellite link channels can experience persistent link fades due to varying weather patterns [18]. The congestion control algorithms should be able to address the persistent fades to reduce performance degradation and unnecessary transmissions when the links are unavailable.
- *Low Bandwidth Feedback Link*: A terrestrial feedback link to the sender has been proposed in [17] and [19]. However, this is difficult to implement in some cases, especially for mobile receivers, which need to use low-bandwidth uplink channel as the feedback link. The feedback link is usually not faster than several hundred kilobits per second for small satellite terminals and a few megabits per second for larger satellite terminals [13].

In the satellite domain, relatively few research has been performed on reliable multicast protocols [17]–[19]. Some key issues to design satellite multicasting are reviewed in [19] and it is shown in [17] that the introduction of a feedback channel is the key to realize bandwidth-efficient, robust, and fully reliable multicast communication via satellites.

In this paper, we consider the satellite multicast scenario as shown in Fig. 1. Here, the sender transmits packets to the satellite through a gateway, then the satellite multicasts packets directly to all receivers. Some receivers may have terrestrial connection among them. Note that not all receivers necessarily need to be connected by terrestrial networks. The receivers use the satellite uplink channel as the feedback link to the sender. Usually, the receiver is a User Earth Station with two way satellite connectivity and terrestrial multicast connectivity. The User Earth Station can work as a proxy for all the terrestrial users connected to it and will be treated as only one receiver in this paper.

Considering the challenges in satellite multicasting, we propose a new reliable multicast transport protocol TCP-Peachtree for satellite IP networks. Our contributions are the following.

- *Hierarchical Logical Groups on the Reverse Path:* Due to the special topology of satellite multicasting, all receivers are only one hop away from the satellite and, hence, there is no physical hierarchy. It is not possible to generate a multicast tree for satellite multicasting as in the case of wirelined networks. The satellite transmits packets directly to all receivers. If all receivers send acknowledgments directly to the satellite, the *acknowledgment implosion problem* becomes very challenging. This is based on the fact that the huge number of receivers are only one hop away from the satellite and no intermediate routers can be used to aggregate the ACKs. Furthermore, the low bandwidth for the feedback link makes the problem worse. In order to address the *acknowledgment implosion* and the *low bandwidth feedback link* problems, we propose the so-called logical hierarchical groups on the reverse path. Based on the logical hierarchical groups, we propose *ACK aggregation, local error recovery*, and *local relay* schemes to suppress the acknowledgments from the receivers and to reduce retransmissions from the sender.
- *New Window-Based Congestion Control Scheme:* As mentioned before, in geosynchronous earth orbit (GEO) satellite multicasting, all receivers have approximately the same round-trip time (RTT) values. Thus, we propose a new window-based congestion control scheme for reliable multicasting in the satellite networks. The new window-based congestion control scheme uses the so-called *NIL segments*. Unlike the *dummy segments* proposed in [3], *NIL segments* are not only used to probe the available network resources, but also to recover packet losses in the receivers. The delayed SACK scheme is adopted to address the bandwidth asymmetry problems. Two new algorithms: *jump start* and *quick recovery* are also proposed to address *long propagation delays* and *high* BERs in the satellite networks. Furthermore, a *hold state* is introduced to address the persistent fades.

This paper is organized as follows. The multicast procedures in TCP-Peachtree are presented in Section II. Then, the congestion control problem is discussed in Section III. Simulation results are given in Section IV. Conclusions are presented in Section V.

II. Multicast Procedures in TCP-Peachtree

A. B+ Tree Hierarchy

For GEO satellite multicasting, the satellite sends packets directly to all receivers and there is no physical hierarchy. In order to suppress the number of ACKs from the receivers and to reduce retransmissions from the sender, we propose logical hierarchical groups on the reverse path. As some receivers may have terrestrial connections among them, they may form one or more logical hierarchical groups so that each group only needs to send one ACK for each received packet. On the other hand, *local recovery*, and *local relay* schemes can be used for local

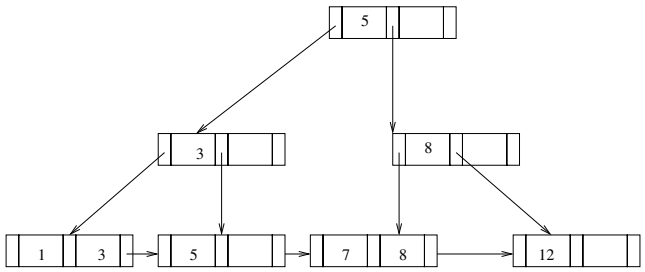

Fig. 2. Typical B+ tree.

```
Random_Selection()
  P = 0;
  K = 1;
  For i = 1 to M
     if (r ≥ P)
        P = P + P_i;
        K = K + 1;
     else
        break;
     end;
  end;
end;
```

Fig. 3. Random selection algorithm.

error recovery to reduce retransmissions from the sender. The satellite still sends packets directly to all receivers, the logical hierarchical group is only formed on the reverse path such that the packet transmissions from the satellite to the receivers are not affected. Note that "reorganizing" receivers in the logical hierarchy does not lead to high overhead and is not error prone.

The logical hierarchy is constructed in a way similar to the B+ tree [9]. B+ tree is a data structure typically used for searching data with data pointers stored only at the leaf nodes of the tree. The main benefit of B+ tree is that it is balanced. A B+ tree of order M has the following characteristics.

- Each internal node is of the form

$$\langle P_1, K_1, P_2, K_2, \ldots, P_{q-1}, K_{q-1}, P_q \rangle$$

  where $q \leq M$ and each P_i is a tree pointer and K_i is the key value for searching.
- Within each internal node, $K_1 < K_2 < \cdots < K_{q-1}$.
- For all search field values X in the subtree pointed at by P_i, we have $K_{i-1} < X \leq K_i$ for $1 < i < q$; $X \leq K_i$ for $i = 1$, and $K_{i-1} < X$ for $i = q$.
- Each internal node has at most M tree pointers.
- Each internal node, except the root, has at least $\lceil M/2 \rceil$ tree pointers. The root node has at least two tree pointers if it is an internal node.
- An internal node with q pointers $q \leq M$ has $q - 1$ search field values.

A typical B+ tree is illustrated in Fig. 2, where six numbers: 1, 3, 5, 7, 8, and 12 are stored in a B+ tree of order 2. All numbers are stored in the leaf nodes. The key values 3, 5, and 8 are used for key searching. The leaf nodes of the B+ tree are usually linked together to provide ordered access to the records.

Here, we use B+ tree to update hierarchical structure automatically instead of searching for a key. So we modify the traditional B+ tree [9] as follows.

- The modified B+ tree forms a multicast group.
- Each leaf node corresponds to a multicast subgroup which consists of physical receivers.
- Each internal node corresponds to a logical subgroup, which is created by choosing the *designated receiver* (DR) from each of its child subgroups. The DR is a node in a subgroup that is responsible for *ACK aggregation, local error recovery*, and *exchanging information* with members in its parent subgroup on behalf of all members in its subgroup.
- In a subgroup, one member is selected as the DR.
- Each node (i.e., a multicast group) has at most M members.
- Each node (i.e., a multicast group), except the root, has at least $\lceil M/2 \rceil$ members. The root node has at least two members, if it is an internal node.

The join and split procedures for the standard B+ tree are also modified to update the modified B+ tree dynamically when a member joins or leaves the multicast group. Using the modified B+ tree, we create a dynamic hierarchical structure for each multicast group.

B. Selecting the DR

Any member can start a multicast group by sending a signaling message to the satellite via the uplink channel. This member becomes the DR initially and it also gets a group ID from the sender, which is used for later transmissions. When members join or leave a multicast group, subgroups may be split or joined accordingly and, consequently, new DRs can be selected. A DR is selected according to the *packet loss statistics*. The DR selection procedures are as follows.

- Each receiver calculates the *packet loss statistics* and sends it to the DR periodically. Let L_i^j be the number of losses from the sender observed by receiver i in epoch j and let L_i be the *packet loss statistics from the sender*, then L_i is updated as follows:

$$L_i = \eta L_{i-1} + L_i^j \tag{1}$$

  where $j \geq 0$ and $0 < \eta \leq 1$. If $\eta = 1, L_i$ is the sum of missing packets from the sender. Usually, $\eta < 1$.
- The DR then ranks all members in an increasing order according to their *packet loss statistics L_i from the sender*.
- The probability P_i for the receiver i to be selected as a DR is calculated from

$$P_i = (1/r_i) \Bigg/ \left(\sum_{i=1}^{M} 1 \,/r_i \right) \tag{2}$$

  where r_i is the rank of the multicast receiver i and M is the number of members in the multicast subgroup.
- Choose a random value $r(0 < r < 1)$.
- The *random selection* algorithm is as shown in Fig. 3 to select the DR (assuming P and K are variables): As a result, the receiver K is chosen as the DR for this multicast group.

The advantage of the *random selection procedure* is to choose the DR according to its packet loss statistics. The receiver with

lower packet loss rate has a higher chance to become a DR. The random selection algorithm can also address the dynamic nature of the network. A receiver with very low packet loss rate does not necessarily mean that it maintains that state forever.

The DR in each subgroup has the following functions.

- Buffer packets received from the sender and use them for *local error recovery*.
- Aggregate ACKs for a packet in this subgroup into one ACK and pass it to the DR in its parent logical subgroup.
- Perform *local retransmission*.
- Forward retransmission requests to its parent logical subgroup.
- Work as a relay for receivers which have very bad channel conditions.
- Manage member joining or leaving.
- Calculate and select a new DR for a new subgroup which is split from the current subgroup.

C. Members Joining a Multicast Group

A new member sends a particular *LOOK_FOR_DR* packet with *time-to-live* (TTL) parameter equal to one. Upon receiving this packet, the DRs send *IM_DR* type of ACK to the new member, which then chooses the DR with the smallest RTT value. If no reply comes back from any DR, the TTL value is increased by one and this procedure is repeated until a DR is found with the RTT value satisfying: $\mathrm{RTT}/2 < \mathrm{TRTT}$, where TRTT is a threshold and is discussed and defined in the performance evaluation in Section IV. Then, the new member joins that multicast group. If no DR can be found with a very large TTL value, it means that either no terrestrial connection exists from the new member to other DRs, or all DRs are too far away. If the DRs are too far away, it takes a long time to do retransmissions and ACK aggregations when the new member joins that group and, hence, affects the response time of that group to the sender and, consequently, the overall performance is degraded. In this case, the new member must initiate a new multicast group by sending a signaling message to the sender and becomes the DR of this multicast group initially. When a new member joins a multicast group, it must first join a subgroup at the leaf in the logical hierarchical structure. If the number of the members in a multicast subgroup exceeds a given threshold M, then we split the group into two subgroups, with one subgroup keeping the current DR, while the other subgroup selecting a new DR. As a result, a new DR in the upper logical multicast group is selected. We repeat this procedure in the upper logical multicast groups if necessary until the highest logical multicast group is reached.

D. Members Leaving a Multicast Group

When a member leaves a multicast group and the number of group members becomes less than $\lceil M/2 \rceil$, then the group members are redistributed to a neighboring group, or two groups are merged into one. In the latter case, a DR is identified for the new subgroup and the according DR is removed from the upper logical multicast group. This procedure is repeated if necessary until the highest logical multicast group is reached. If the last member in the multicast group is removed, it sends a release message to inform the sender that this multicast group is removed.

E. Logical Tree Setup Overhead

In order to setup the logical tree, new members need to send *LOOK_FOR_DR* packets to the DRs and the DRs send *IM_DR* type of ACKs back to the new members. We call the *LOOK_FOR_DR* and *IM_DR* packets *G_JOIN* messages. The number of *G_JOIN* messages used to form logical subgroups is a measurement of logical tree setup overhead [8]. Assume a logical tree has l levels of hierarchies and the average number of *G_JOIN* messages for a group member to join a subgroup is J. In order to calculate the total number of *G_JOIN* messages for a logical tree, we consider two extreme cases.

- The Best Case: The number of members in each subgroup is M, hence the maximum number of receivers in this group is M^l. The total number of members S in the logical tree, including physical receivers and DRs is

$$S = \sum_{i=1}^{l} M^i = \frac{M(M^l - 1)}{M - 1}. \tag{3}$$

  Obviously, the total number of *G_JOIN* messages is $S * J$ and the number of *G_JOIN* messages for the physical receivers to join the multicast group is $M^l * J$, hence the number of *G_JOIN* messages used to form logical subgroups is $(S - M^l)J$. As a result, the logical tree setup overhead O_s is

$$O_s = \frac{(S - M^l)J}{M^l J} = \frac{M(M^l - 1)}{(M - 1)M^l} - 1. \tag{4}$$

  Since the total number of receivers in a multicast group, i.e., M^l, is very large, $M^l - 1 \approx M^l$, we get

$$O_s = \frac{1}{M - 1}. \tag{5}$$

- The Worst Case: The root has two members and the other subgroups have $\lceil M/2 \rceil$ members. Thus, the minimum number of receivers is 1 for $l = 1$ and $2\lceil M/2 \rceil^{l-1}$ for $l > 1$. The total number of members S in the logical tree, including physical receivers and DRs is

$$S = \sum_{i=0}^{l-1} 2\left\lceil \frac{M}{2} \right\rceil^i = \frac{2\left(\left\lceil \frac{M}{2} \right\rceil^l - 1\right)}{\left\lceil \frac{M}{2} \right\rceil - 1}. \tag{6}$$

  Similarly, we get the logical tree setup overhead

$$O_s = \frac{1}{\left\lceil \frac{M}{2} \right\rceil - 1}. \tag{7}$$

Consequently, the logical tree setup overhead is

$$\frac{1}{M - 1} \le O_s \le \frac{1}{\left\lceil \frac{M}{2} \right\rceil - 1}. \tag{8}$$

Usually, M is very large and, hence, the logical tree setup overhead is very small. For example, if $M = 20$, the logical tree setup overhead O_s is in the range of 5% to 11%. If M is larger, O_s becomes much smaller.

F. ACK Aggregation

Received packets are reported in ACKs. The DR in a logical multicast group receives ACKs from members in its group. After it receives all ACKs for a packet from all members in its group, the DR sends an ACK to the upper logical subgroup. The DR in the upper logical group also receives ACKs from its members and sends ACKs to its parent logical group. The DR in the highest logical group then sends an ACK to the satellite via the uplink channel. Thus, for each packet only one ACK is sent to the sender and, hence, the feedback implosion problem is solved. Upon receiving ACKs, the sender also aggregates the ACKs from different multicast groups.

In order to further reduce the number of ACKs from this logical hierarchical group, the delayed SACK scheme is adopted here. The DR in the highest level sends one SACK for a given number of data packets received if the sequence numbers of the received packets are increased accumulatively. Otherwise, the DR sends the SACK to the satellite immediately.

G. Local Error Recovery

For satellite multicasting, different receivers may have uncorrelated packet losses. Although forward error correction (FEC) is very efficient to recover uncorrelated packet losses [26], it also introduces high overhead and requires additional software or hardware processing capability at the receiver side. Here, local error recovery is used to address uncorrelated packet losses. Local error recovery may also suffer from "single point of failure" problem when a DR fails, this problem is solved by reassigning another DR to take it over in TCP-Peachtree.

Local error recovery solution is also presented in [24], [25], and [29], however, they are based on the terrestrial multicast tree. In TCP-Peachtree, *local error recovery* is based on the B+ tree hierarchical structure introduced in Section II-A.

Each DR in a subgroup maintains a buffer to hold a number of packets for *local retransmission*. If a receiver is DR in several different logical hierarchical groups, it shares the same buffer for different hierarchical logical levels. The advantage is that this receiver can obtain lost packets from different logical levels in order to increase the chance for local recovery.

The missing packets are reported in NAKs. When a member does not receive a packet correctly, it sends a NAK to the DR in its subgroup. If the lost packet is in the DR's buffer, the DR retransmits the lost packet to the receiver immediately. If not, the DR first multicasts a NAK to all members in its subgroup. Any member having the correct packet in its buffer can unicast that packet to the DR. On the other hand, the DR sets a timer for the lost packet. If the lost packet is received from other members in this subgroup, the DR unicasts the lost packet to the corresponding receiver if only one NAK is received for the lost packet, or multicasts the lost packet to its entire subgroup if multiple NAKs for this lost packet are received. If a timeout occurs, the DR sends a NAK to the DR in its parent logical group and will try to get the lost packet from its upper logical subgroup. This procedure is repeated until the highest logical subgroup is reached. If no success, one NAK is sent to the satellite via the uplink channel. The sender then multicasts the missed packets to all receivers.

H. Local Relay

Due to very bad channel conditions, some receivers may have relatively much higher *packet loss rates* than some other receivers. As a result, the overall performance is degraded since the sender needs to make reliable transmissions to those receivers. In order to improve the overall performance, a *local relay* scheme is designed, which is used if the *packet loss rate k_i from the sender* of member i is higher than a given threshold $\theta (0 < \theta < 1)$, i.e.,

$$k_i \geq \theta. \tag{9}$$

In this scheme, the DR in the corresponding subgroup forward every packet it receives from the satellite to member i in order to reduce retransmission requests from member i. k_i is updated and reported to the DR periodically. If k_i is less than the given threshold θ, the DR stops forwarding packets to member i.

I. DR Handoff

When a DR leaves a multicast group, or when a new DR is selected during group join or merge, handoff is performed between two DRs. The new DR needs to create or update the data structure to store its group member information in each logical hierarchical level in which it is a DR. The new DR only needs to get the information of aggregated ACKs from the current DR or its DR in its parent logical subgroup. The current DR also needs to inform the group members to change their DR. After the new DR is ready to work, it sends a message to current DR and also all members in each logical subgroup in which it is a DR. Upon receiving this message, the subgroup members send ACKs to the new DR. The current DR forward all ACKs it receives to the new DR. ACKs may get lost during DRs handoff. In order to prevent those losses, the new DR multicasts a message to all members in its group to resend ACKs to it. Upon receiving such message, all members send the ACKs to the DR immediately.

J. DR Failure

If a DR fails, a new DR is selected based on the procedure described in Section II-A and the B+ tree type hierarchical structure is updated accordingly. There are two ways to detect a DR failure. In the first case, a DR detects the failure of a member, which is a DR in its child subgroup, by keeping track of the ACKs from that member. If the DR does not get any message from a member for a certain time period, it sends a poll message to this member. Upon receiving this message, the member should send an ACK for this message immediately. If such an ACK is not received during a certain time period, the DR assumes this member as nonexisting and it acts as a DR for the misfunctioning subgroup temporarily until a new DR is selected from that subgroup.

For the DR in the highest logical subgroup, the above method does not work. Thus, the DR in the highest logical subgroup multicasts a poll message to its member periodically. If a member does not receive such a message within a certain time period, it sends the poll message to the DR. If it does not get an ACK for this poll message from the DR within a time period, it assumes that the DR has failed. Thus, it acts as a DR temporarily until a new DR is selected.

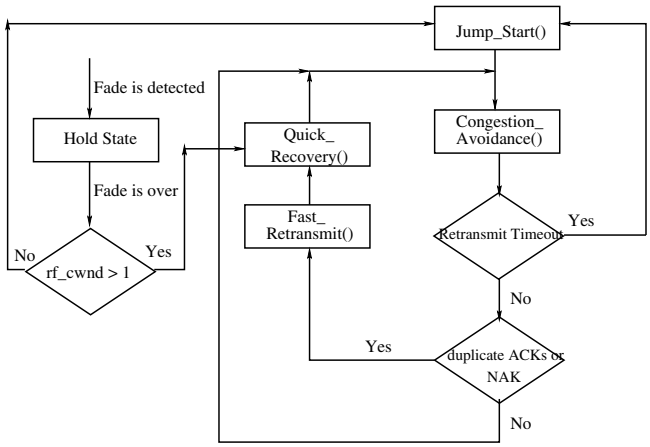

Fig. 4. TCP-Peachtree flow chart.

```
Generate_NIL_Segment()
  Add all unacknowledged packets into
  the queue
  i = 0;
  if a NIL segment is needed
    q = imodQ;
    i = i + 1;
    select the qth packet in the queue as
    the NIL segment;
  end;
  if the jth packet in the queue is ACKed
    remove that packet from queue Q;
    if (j < i and i > 0)
      i = i - 1;
    end;
  end;
  if a new packet is added to the queue Q
    add the new packet at the tail of the queue;
    Q = Q + 1;
  end;
end;
```

Fig. 5. NIL segment generating algorithm.

We could also use another method to cope with the DR failure. A special receiver is selected by the DR as a backup in each subgroup. The DR sends updated information to the backup receiver periodically, while this receiver sends a poll message to the DR periodically to probe the up-to-date status of the DR. If the DR fails, the backup receiver takes over the control and works as a new DR.

III. Congestion Control in TCP-Peachtree

TCP-Peachtree congestion control contains two new algorithms: *jump start* and *quick recovery*, as well as the two traditional TCP algorithms, *congestion avoidance* and *fast retransmit*. The TCP SACK options are also adopted. Moreover, a *hold state* is introduced to address persistent fades. The flow chart of TCP-Peachtree congestion control algorithm is illustrated in Fig. 4.

A. SACK Scheme

Traditional TCP schemes like TCP-Reno have problems when multiple packets are lost in a window, the TCP SACK option is proposed to solve these problems for different applications [5], [6], [10], [27]. Hoe [14] proposed changes to the fast retransmit algorithm so that it can quickly recover from multiple packet losses without waiting unnecessarily for the timeout. In TCP-Peachtree, TCP SACK options are adopted to recover multiple packet losses in one RTT.

There are two types of selective ACKs, i.e., positive ACKs (ACKs) and negative ACKs (NAKs). Upon receiving a packet, an ACK is sent immediately to the DR. If the DR received all ACKs for this packet from all its members, an ACK is sent to the DR in its parent logical group until the sender is reached. While one or more missing packets are detected by the receiver, a NAK is sent to the DR for retransmission. If the DR does not have the missing packets in its buffer, it sends a NAK to its parent logical group until the NAK reaches the sender.

B. NIL Segments

In TCP-Peach [3], a type of low-priority segments called *dummy segments* are used to probe the availability of network resources. In TCP-Peachtree, we use low-priority segments as *NIL segments* [4]. There are two main differences between *dummy segments* and *NIL segments*. First, *dummy segments* are used only to probe the availability of network resources, while *NIL segments* are used to probe the availability of network resources and also for error recovery. Another difference is the way how the low-priority segments are generated. In TCP-Peach, *dummy segments* are generated by the sender as a copy of the last transmitted data segment, while *NIL segments* are generated using the NIL segment generating algorithm as shown in Fig. 5 to carry unacknowledged packets, which can be used by the receivers to recover missing packets. Since the probability for a packet to be lost somewhere in the satellite multicast network is rather high, using *NIL segments* to recover errors is advantageous.

Low-priority segments are also used in [23] to improve the performance of TCP *slow start* Algorithm. However, the low-priority segments in [23] are different from NIL segments in that:

- They are not used to probe the availability of network resources. In fact, their objective is to carry information to the receiver more rapidly without harming other flows.
- Since they carry new information to the receiver, they are still data segments, and if they are lost, then they must be recovered.
- They are used only in the beginning of a new connection.

NIL segments are low-priority segments that do not carry any new information. If a router on the connection path is congested, then it discards the *NIL segments* first. Consequently, the transmission of *NIL segments* does not cause a decrease of throughput of actual data segments. If the routers are not congested, then the *NIL segments* can reach the receiver. The sender sets one or more of the six reserved bits in the TCP header to distinguish *NIL segments* from data segments. Therefore, the receiver can recognize the *NIL segments* and for each of them transmits an ACK back to the sender. The ACKs for *NIL segments* are also marked using one or more of the six reserved bits of the TCP header and are carried by low-priority IP segments. Upon receiving ACKs for NIL segments, the sender interprets those ACKs as the evidence that there are unused resources in the network and accordingly, can increase its transmission rate. NIL segments are only transmitted when

```
Jump_Start()
    cwnd = 1;
    τ = RTT/rwnd;
    send(Data_Segment);
    for (i = 1 to rwnd − 1)
        wait(τ);
        send(NIL_Segment);
    end;
    for each received ACK
        cwnd=cwnd+1;
    end
end;
```

Fig. 6. Jump start algorithm.

the sender is in *jump start* phase or when packet losses are first found during the *quick recovery* phase.

Let Q be the queue length and i be a counter, then the *NIL segments* are created as shown in Fig. 5.

Upon receiving *NIL segments* at receivers, they can use them for error recovery and send a NIL ACK to the DR. The DR counts the number of ACK's for a *NIL segment*, if the percentage of reception for this *NIL segment* exceeds a given threshold $T_{\rm nil}$, where $0 < T_{\rm nil} \leq 1$, then the DR sends a NIL ACK to the DR in the parent logical group or to the sender.

C. Jump Start Algorithm

TCP *Slow Start* algorithm is disadvantageous in networks with long propagation delays. Several schemes have been proposed to improve the TCP *slow start* algorithm, TCP fast start [23] transmits low-priority packets during the fast start phase. Since these packets carry new information, they must be recovered if they are lost. In TCP-Peach [3], *dummy segments*, a type of low-priority segments are used to probe the availability of network resources. In TCP-Peachtree, NIL segments described in Section III-B are used to improve the *slow start* algorithm. The algorithm is as shown in Fig. 6.

The basic idea of *jump start* is that in the beginning of a connection the TCP sender sets the congestion window (*cwnd*) to 1 and after the first data segment, it transmits $(\mathrm{rwnd} - 1)$ *NIL segments* created as in Section III-B every $\tau = \mathrm{RTT}/\mathrm{rwnd}$. For each received ACK of a data packet or a NIL segment, the congestion window size is increased by one. As a result, after one RTT, the congestion window size *cwnd* increases very quickly. Note that the TCP sender can estimate RTT during the connection setup phase. Here, RTT is the largest value the sender gets from the receivers.

D. Quick Recovery Algorithm

The *quick recovery* substitutes the classical fast recovery algorithm [16] with the objective of recovering multiple losses in a window and solving the throughput degradation problem due to link errors. As shown in Fig. 4, when one or more segment losses are detected by either duplicate SACKs or a NAK, we use the *fast retransmit* algorithm presented in [16] to transmit a missing packet. After completing the *fast retransmit* algorithm, we apply the *quick recovery* algorithm.

Similar to the TCP SACK proposed in [10], *quick recovery* maintains a data structure called *scoreboard* to update information about missing packets, and a variable called *pipe* that represents the estimated number of packets outstanding in the path. The variable *pipe* is incremented by one when the sender either sends a new packet or retransmits an old packet. It is decremented by one when the sender receives a duplicate ACK packet with a SACK option reporting the new data is received at the receiver. The sender always sends the missing packets first. If no missing packet exists, the sender sends a new packet. Whenever a SACK is accepted, the *retransmit timer* is also reset. The sender exits *quick recovery* when a recovery ACK is received ACKing all data that were outstanding when *quick recovery* was entered.

The following variables or parameters, which are used in *quick recovery*, are summarized as follows.

- *HighAck* is the sequence number of the highest cumulative ACK received at a given point.
- *HighData* is the highest sequence number transmitted just before rapid recovery begins.
- A *duplicate acknowledgment* is defined as an ACK, whose cumulative ACK number is equal to the current value of *HighAck*. It also conveys new SACK information for segment(s) above *HighAck*.
- A *partial ACK* is an ACK that increases the *HighAck* value, but does not acknowledge all of the data up to and including *HighData*.
- *ndupacks* is the number of duplicate packets which triggers the TCP-Peachtree to *fast retransmit* and *quick recovery* phases. This number is assumed to be 3.
- *maxburst* is a parameter which limits the number of packets that can be sent in response to a single incoming ACK, even if the sender's congestion window would allow more packets to be sent. *maxburst* is assumed to be four as proposed in [10].
- *adsn* is the number of NIL segments that the TCP sender is allowed to inject into the network.
- *adps* is the number of allowed packets to be sent.
- *nps* is the number of packets actually sent.
- *END* is used to determine whether the *quick recovery* should be terminated.
- *nsacked* is the number of new packets which are SACKed by an ACK with the SACK option.
- *msacked* is the number of packets which are SACKed and whose SN are larger than *HighAck*.

As shown in Fig. 7, when packet losses are detected, the *quick recovery* behaves conservatively, i.e., the sender halves its congestion window *cwnd*. Since the original congestion window is $2*\mathrm{cwnd}$ and the sender has received *ndupacks* SACKs right before the *quick recovery*, each of which reports a new data packet left the pipe. The sender also transmits a loss packet using the *fast retransmit* algorithm. Consequently, the variable *pipe* is initialized as *HighData - HighAck - msacked*, i.e, there are *HighData - HighAck - msacked* data packets outstanding in the network. The *quick recovery* is terminated roughly within one RTT. NIL segments sent during *quick recovery* are received in the *congestion avoidance* phase and the congestion window *cwnd* is increased by one for each NIL ACK. As a result, the number of allowed NIL segments is set as *cwnd* so that the congestion window becomes the original value before the *quick recovery* very rapidly.

```
Quick_Recovery()
    cwnd = cwnd/2;
    adsn = cwnd;
    adps = 0;
    END = 0;
    pipe = HighData − HighAck − msacked;
    while (END = 0)
        if (ACK_ARRIVAL)
            if( Duplicate ACK or Partial ACK)
                pipe = pipe − nsacked;
            if ( NAK )
                pipe = pipe − 1
            end;
            if (NIL ACK)
                cwnd = cwnd + 1;
                adsn = adsn − 1;
            end;
            if (ACK which acknowledges all data up
            to and cover HighData)
                update HighAck;
                END = 1;
            end;
            update scoreboard;
            adps = cwnd − pipe;
            nps = min(maxburst, adps)
            if ( nps > 0 )
                send nps missing or new packets;
                pipe = pipe + nps;
            end;
            else if (adsn> 0)
                send a NIL packet;
                adsn = adsn − 1;
            end;
        end;
    end;
end;
```

Fig. 7. Quick recovery algorithm.

During the *quick recovery* phase, if a duplicate ACK or partial ACK arrives, *pipe* is reduced by the number of new packets, which are SACKed in the ACK. If a NAK is received, which indicates that a packet left the pipe but is not received correctly, *pipe* is also reduced by one and *scoreboard* is updated to include the missing packet. If a NIL ACK is received, which indicates that there are still more resources available in the network. Accordingly, *cwnd* is increased by one and *adsn* is reduced by one. If an ACK that acknowledges all data up to *HighData* is received, $\mathrm{END} = 1$ is set and the *quick recovery* is terminated.

Whenever an ACK is received, the *scoreboard* is updated to keep the latest information about missing packets. The number of packets which can be sent to the network is (*cwnd-pipe*). In order to avoid injecting bursty packets into the network, the number of packets that can be sent back-to-back is the minimum of *maxburst* and (*cwnd-pipe*). If more data packets are allowed to be sent, the missing packets are transmitted first. If no missing packets are needed to be retransmitted, new data packets are sent. If no data packet is sent, a NIL segment is transmitted as long as $\mathrm{adsn} > 0$.

In TCP-Peachtree, when a retransmission timeout occurs, the sender checks whether it has received any ACK for that packet from any multicast group. If it did receive some ACKs for that packet from some multicast groups, the timeout is due to link error. This is based on the fact that no multicast group can receive a packet if it is lost in the gateway due to congestion. In this case, the sender retransmits that packet and stays in *quick recovery* phase. On the other hand, if there is no ACK for that packet from any multicast group, the timeout is due to congestion, the sender terminates the *quick recovery* and executes the *jump start*.

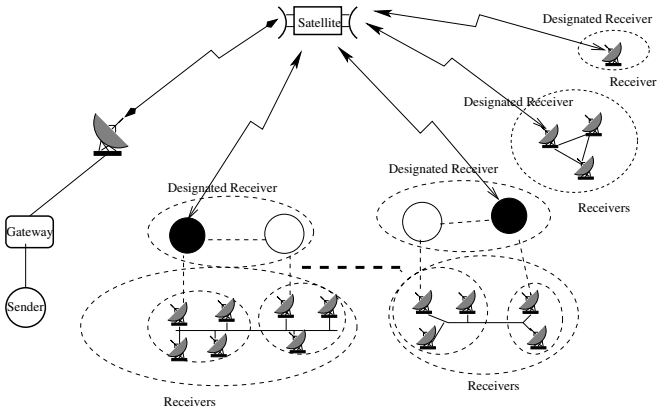

Fig. 8. Logical hierarchical structure for TCP-Peachtree.

E. Hold State

In order to address the persistent fades due to varying weather patterns, such as rain fades, a *hold state* is developed in TCP-Peachtree. The sender receives ACKs for reliability control purposes. If the sender does not receive any ACK from a multicast group or from all multicast groups for a certain period of time T_f, it infers this condition as rain fade and goes to the *hold state*. In the *hold state*, the sender first records the current congestion window *cwnd* in the variable *rf_cwnd*. Then, it freezes all retransmission timers and starts to send probing segments to the receiver periodically. Upon receiving a probing segment, the DR in the highest hierarchical level sends an ACK immediately to the sender to report its current buffer status.

If the sender receives the ACKs for the probing segments, it infers that the rain fade is over and resumes sending data packets. If $\mathrm{rf\_cwnd} = 1$, it goes to the jump start state, otherwise, let $\mathrm{cwnd} = \mathrm{rf\_cwnd}$. In the latter case, the sender first transmits *rwnd-cwnd* NIL segments to probe the available bandwidth and then enters the quick recovery state to recover the missing packets.

IV. Performance Evaluation

We developed our own simulation model. The physical structure for satellite multicasting is shown in Fig. 1, and the logical hierarchical structure is shown in Fig. 8. The sender sends packets to the satellite through a gateway, then the satellite multicasts packets to all receivers. Some receivers may have terrestrial connection among them, but not all of them need to be connected by terrestrial networks. The receivers use the satellite uplink channel as the feedback link to the sender. Since the GEO satellite network is considered, the RTT values (550 ms) from the sender to all receivers are approximately the same.

The measured packet loss probability due to link errors in the channel varies from 10^{-6} to 10^{-2}. Furthermore, we assume the gateway buffer length to be 50 segments and $\mathrm{rwnd} = 64$ segments. We also assume that the link capacity is $c = 1300$ segments/s, which is approximately 10 Mb/s for TCP segments of 1000 bytes. Moreover, there are 11 unicast TCP-NewReno connections from the gateway to the receivers. For those connections, we assume $\mathrm{rwnd} = 64$ segments. Also, assume the application is reliable data distribution and simulation time is 100 s.

TABLE I
RESULTING LOGICAL HIERARCHICAL GROUPS FOR $N = 200$ AND $M = 10$

TRTT (ms)	15	20	25	30	35	40
Group Number	21	10	5	5	3	2
Highest Level	2	2	3	3	3	3

Logical hierarchical multicast groups are first discussed in Section IV-A. In order to illustrate the basic properties of TCP-Peachtree, i.e., throughput performance, overhead, and fairness with respect to packet loss rates due to channel errors, we first assume all receivers have the same packet loss rate, i.e., $p_i = p$. The unbalanced situation where different receivers have different packet loss rates is discussed in Section IV-E. The performances for rain fades and bandwidth asymmetry are illustrated in Sections IV-F and IV-G, respectively. Finally, scalability is discussed in Section IV-H.

A. Logical Hierarchical Multicast Groups

Logical hierarchical groups are formed using the modified B+ tree presented in Section II. The group formation is affected by the following factors.

- *Topology of Receivers:* Receivers form clusters and there is no terrestrial connection among different clusters. Thus, each cluster forms one or more separate multicast groups.
- *RTT Threshold:* TRTT is the threshold used when a receiver selects a DR to join. Assume the RTT value from a new receiver to a DR is RTT, if $\mathrm{RTT}/2 < \mathrm{TRTT}$, the new receiver can join that subgroup. If the RTT value from any DR to the new receiver is larger than $2 * \mathrm{TRTT}$, the new receiver needs to initiate a new multicast group.
- *Maximum Number of Members Allowed in a Subgroup: M* is the maximum member number allowed in a subgroup. If the member number in a subgroup is larger than M, this subgroup is split into two subgroups. After a receiver leaves a multicast group, if the member number in this subgroup is smaller than $M/2$, this subgroup is merged with a neighboring subgroup to form a new subgroup.

Assume $N = 200, M = 10$, and all receivers are in one cluster. The resulting logical hierarchical multicast groups are shown in Table I.

For a given number of receivers, with TRTT increasing, the number of groups decreases, but the highest hierarchical level in a group may increase. If TRTT is too small, there are too many multicast groups.

Although the RTT value from one receiver to the sender is already known. Due to logical hierarchical groups, ACKs are aggregated level by level by the DRs in a group and then sent to the sender, the actual RTT value for one multicast group to the sender should be larger than the given RTT value, i.e., the RTT value for a multicast group should be larger than 550 ms. The actual RTT values are shown in Fig. 9, where the following is demonstrated.

- As TRTT increases, the RTT value also increases, thus, TRTT value should not be very high.
- Usually, if the highest logical hierarchical level increases, the RTT value also increases, but usually three levels of hierarchy are sufficient for most multicast cases.

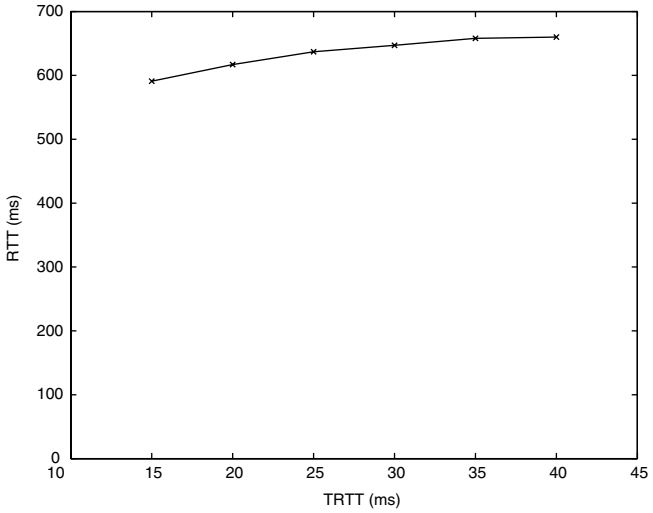

Fig. 9. Average RTT versus TRTT for $N = 200, M = 10$, and $p = 10^{-3}$.

TABLE II
RESULTING HIERARCHICAL GROUPS FOR $N = 200$ AND $\mathrm{TRTT} = 30$ ms

M	5	10	15	20	25	30	35	40
Group Number	1	5	5	6	7	6	8	8
Highest Level	4	3	2	2	2	2	2	2

- The reason why the RTT value increases is that ACKs need to pass along DRs from low hierarchical level to the highest hierarchical level. Another reason is *local error recovery* discussed in Section II-G, the ACK for a packet is sent to the sender only after the packet is received by all receivers in this group, which may also introduce some delays.

Now, we consider how the value of M affects the formation of the logical hierarchical multicast groups. Let $\mathrm{TRTT} = 30$ ms. In a similar way, we get the hierarchical groups as shown in Table II. Obviously, if M decreases, the highest level in a hierarchical group might increase, which is property of the B+ tree. M can also affect how many groups are formed. One interesting observation is that the number of groups formed does not necessarily decrease for increasing M.

As shown in Fig. 10, the effect of M on RTT value is random and the variations of RTT are not large.

Obviously, the aggregated RTT value has very limited sensitivity to the parameter M. Thus, we can choose M sufficiently large to put more receivers in one group so as to reduce the number of ACKs from the receivers and retransmissions from the sender.

B. Throughput Performance

TCP-NewReno [11] is a TCP unicast algorithm, the most important feature of TCP-NewReno is that it can recover multiple missing packets in one window of data. TCP-Peachtree congestion algorithm discussed in Section III is also designed to recover multiple packets in one window of data. In order to evaluate TCP-Peachtree congestion algorithms, we use TCP-NewReno only for congestion control in our simulations and keep the logical hierarchical groups as we introduced in

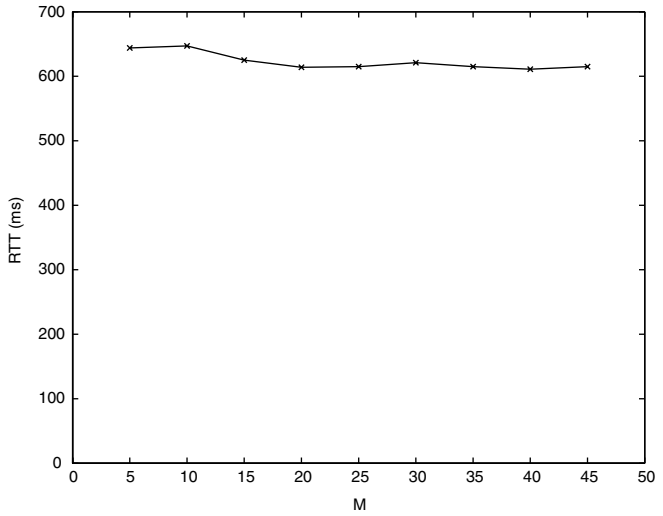

Fig. 10. RTT versus M for $N = 200$, $\mathrm{TRTT} = 30$ ms, and $p = 10^{-3}$.

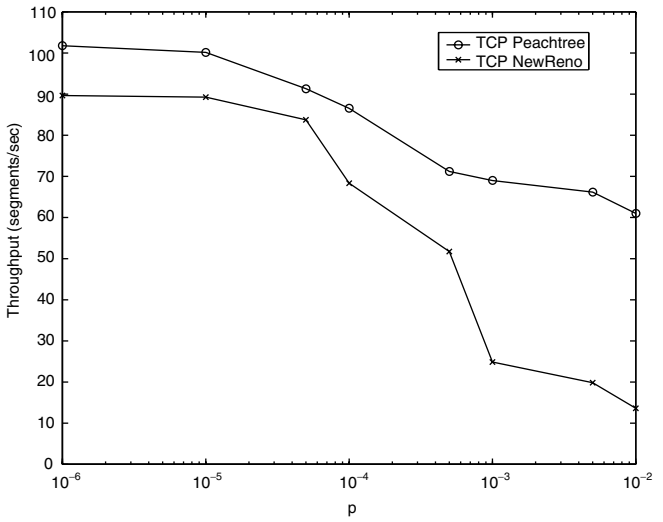

Fig. 11. Throughput performance comparison of TCP-Peachtree and TCP-NewReno for different values of p for $\mathrm{TRTT} = 30$ ms and $M = 10$.

Section II. Thus, we can compare the throughput performance between TCP-Peachtree and TCP-NewReno. For multicast applications, we assume $N = 200, \mathrm{TRTT} = 30$ ms, and $M = 10$, the resulting throughputs are shown in Fig. 11 for different values of p.

Fig. 11 shows that the throughput performance of TCP-Peachtree is much better than the TCP-NewReno, especially when p is large, i.e., TCP-Peachtree is more suitable for reliable satellite multicasting. The reasons are obvious.

- In satellite IP multicasting, due to the *loss path multiplicity problem*, the aggregated link error probability is rather high.
- Using the *jump start* algorithm, TCP-Peachtree can reach $rwnd$ much faster than the TCP-NewReno, which uses the *slow start* algorithm. It is a critical factor in satellite IP networks [2], because the RTT value in satellite IP networks is rather high.
- *NIL segments* are used to exploit the network resources and also to recover lost packets on the receiver side. Due to the *loss path multiplicity problem*, if the receiver number is large, *NIL recovery* can have a high advantage over the TCP-NewReno.
- In *quick recovery*, the information contained in the SACK option is used to improve the performance of TCP-Peachtree.

More detailed performance comparisons of the proposed congestion scheme and other TCP protocols can be found in [1] and [4].

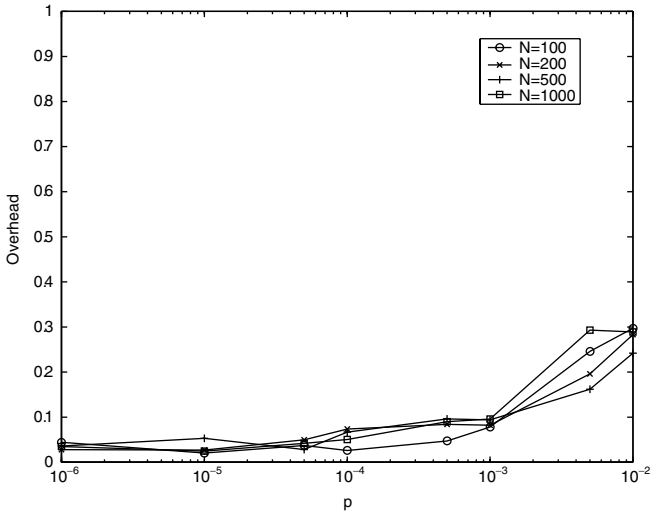

Fig. 12. Overhead versus p for $\mathrm{TRTT} = 30$ ms and $M = 10$.

C. Overhead

In TCP-Peachtree, overhead as shown in Fig. 12 is introduced by NIL segments sent during jump start and quick recovery states. The overhead is measured by dividing the number of NIL segments by the number of total segments sent at the sender.

In general, the overhead decreases with decreasing p from 10^{-2} to 10^{-6}. For $N = 1000$ and $p = 10^{-2}$, the overhead is maximum at 29%. The reason for high overhead is that the aggregated link error probability is very large at this point and the *loss path multiplicity problem* makes it worse. Considering these factors, the overhead here is reasonable. Moreover, the practical channel conditions are below $p \leq 10^{-3}$, where the overhead drops below 10%.

D. Fairness to Unicast TCP Traffic

Usually, the TCP multicast traffic must coexist with other existing TCP unicast traffic. This also applies to satellite IP networks. In order not to overtake bandwidth and starve the unicast traffic, TCP-Peachtree should be fair to unicast traffic.

As described in [25], the fairness index, based on throughput for a bottleneck link is defined as

$$\mathrm{FI} = \frac{\left[\sum_{i=1}^{K} T(x)\right]^2}{N \sum_{i=1}^{K} T(x)^2} \tag{10}$$

where $T(x)$ is the throughput of the xth flow, and K is the number of flows sharing the resource. *FI* always lies between $1/K$ (indicating one of them gets all the bandwidth and all others starve) and 1 (indicating all get an equal share of the bandwidth). Assume there are 11 TCP-NewReno unicast connections from the gateway to the receivers with $\mathrm{rwnd} = 64$,

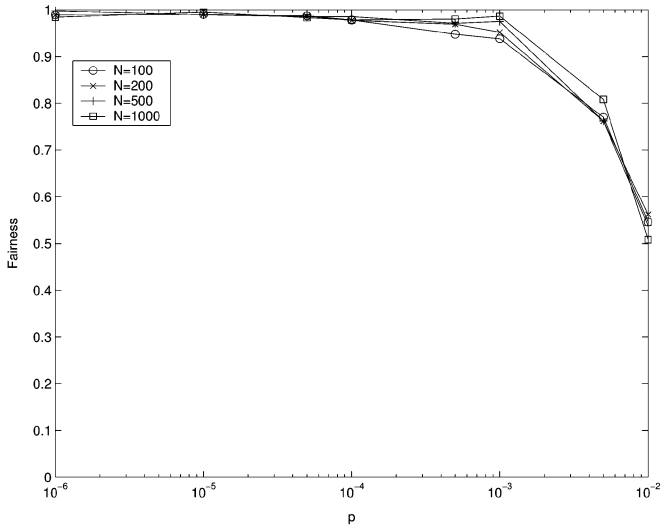

Fig. 13. Fairness versus p for $\mathrm{TRTT} = 30$ ms and $M = 10$.

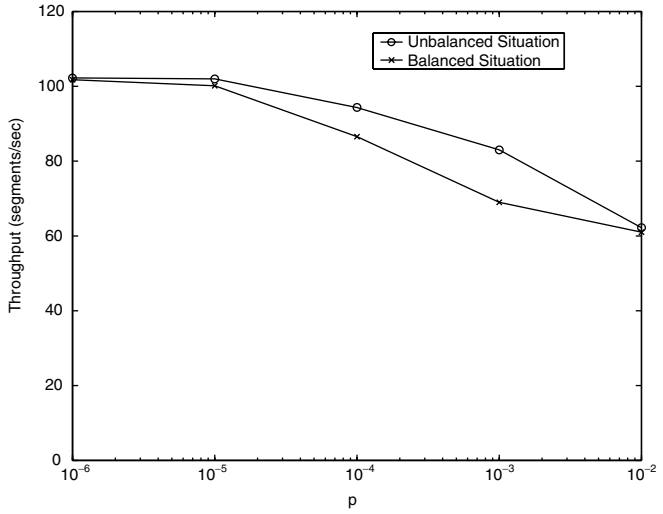

Fig. 14. Throughput comparison for balanced and unbalanced packet loss rates.

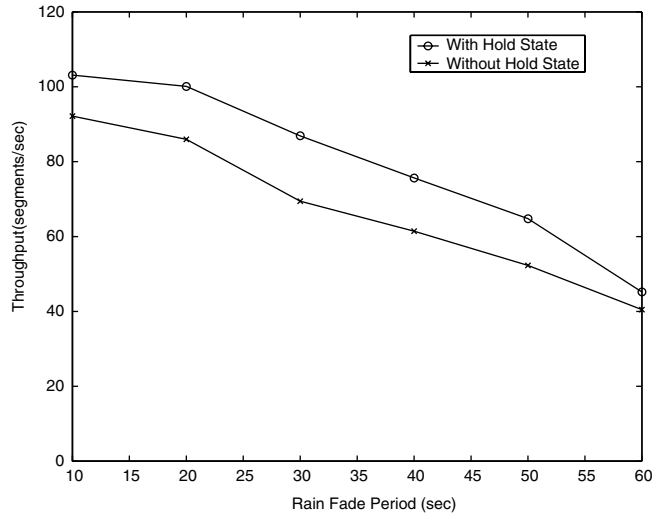

Fig. 15. Throughput versus rain fade period.

they compete for the bandwidth with TCP-Peachtree. The resulting fairness is shown in Fig. 13.

For $p = 10^{-2}$, the fairness is not good. However, it does not mean that the TCP-Peachtree is unfair to the unicast TCP-NewReno traffic. As the link error probability is very large, the TCP-NewReno cannot fully utilize the bandwidth, while TCP-Peachtree can utilize the bandwidth more efficiently. The reason for the result is based on the fact that the bandwidth is not fully utilized by TCP-NewReno at this moment.

For decreasing p, the fairness also increases. For $p \leq 10^{-3}$, fairness becomes larger than 0.9. Moreover, fairness keeps increasing with decreasing p and eventually it reaches one. As a result, the TCP-Peachtree is fair to TCP unicast traffic. The reason is that TCP-Peachtree only uses NIL segments to probe the network resources. When congestion occurs, NIL segments are usually dropped first. Consequently, TCP-Peachtree increases its sending rate much slower. In the worst case, it acts very similar to the TCP-NewReno.

E. Unbalanced Situation

For practical satellite multicast applications, different receivers may experience different channel conditions, thus, they may have different packet loss rates due to channel errors. Assume that the packet loss rate p_i due to channel errors for receiver i is randomly chosen in the range of p_l to p_h. Here, p_l is chosen as 10^{-6} and $p_h = p$ increases from 10^{-6} to 10^{-2}, the throughput performance with respect to p_h is shown in Fig. 14. The throughput of the balanced situation, i.e., all $p_i = p$, is also shown in Fig. 14.

The throughput for the unbalanced situation is always higher than that for the balanced situation. Obviously, the throughput difference is quite small when p_h is close to p_l, i.e., 10^{-6}, but it increases when p_h increases from 10^{-6} to 10^{-3} and it reaches the highest value at $p_h = 10^{-3}$. The reason is that local error recovery is very helpful to recover lost packets for the unbalanced situation. However, the throughput difference drops dramatically at $p_h = 10^{-2}$. This is because most receivers are in worse channel conditions and local error recovery is not efficient in such situations, thus, the throughput performance is degraded greatly.

F. Rain Fade

Assume rain fades last for a short duration of not more than 60 s similar to the assumption made in [18] and rain fade occurs at time $t = 20$ s. The timeout threshold T_f for detecting rain fades is chosen to be the same as the retransmission timeout threshold. The sender may go to the jump start state before it can infer rain fade, thus, it always records the current congestion window *cwnd* into the variable *rf_cwnd* before it goes to the jump start state. During the hold state, the sender stops sending any data packets and freezes all retransmission timers to avoid unnecessary transmission. However, it sends probing packets periodically to detect when rain fade is over. To investigate the performance improvement by the hold state, we also consider TCP-Peachtree without the hold state, where the sender keeps going into the jump start state when the transmission timer expires. The throughput performance for TCP-Peachtree with and without the hold state is shown in Fig. 15.

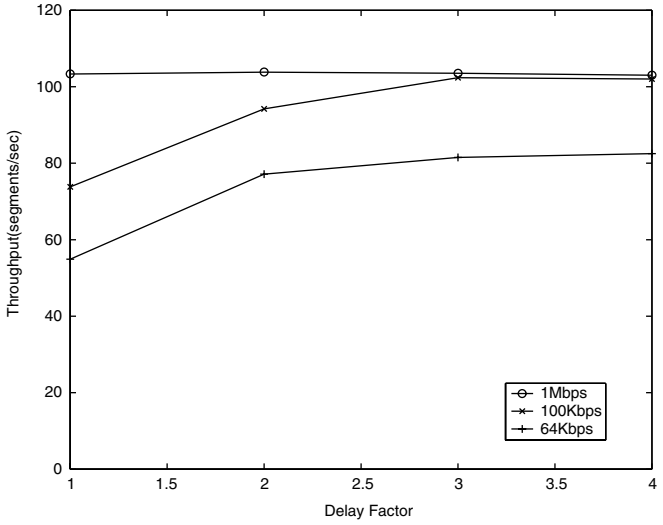

Fig. 16. Throughput versus delay factor.

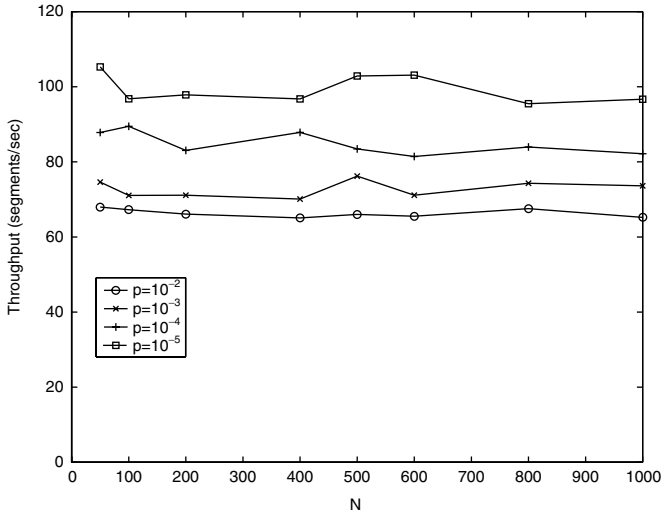

Fig. 17. Scalability for $\mathrm{TRTT} = 30$ ms and $M = 10$.

The throughput with the hold state is always higher than that without the hold state and the throughput difference is approximately constant for different rain fade periods. The reasons that TCP-Peachtree with the hold state achieves higher throughput are as follows.

- In the hold state, the sender uses probing segments to obtain the exact information about rain fade. On the other hand, for TCP-Peachtree without the hold state, the sender has to wait for the retransmission timer to expire and then goes to the jump start state to send packets to the receiver.
- The sender records its current congestion window when rain fade occurs and keeps this congestion window after rain fade is over. While for TCP-Peachtree without the hold state, the sender always goes to the jump start state and the congestion window is set to one.

G. Bandwidth Asymmetry

Since the feedback link capacity is usually very low for satellite networks, the delayed SACK scheme discussed in Section II-F is adopted to address the bandwidth asymmetry problem, i.e., the DR in the highest level sends one SACK for a given number of received data packets. This number is called the delay factor. We investigate three cases of the bandwidth asymmetry problems, i.e., the feedback link bandwidth is 1 Mb/s, 100 kb/s, and 64 kb/s, respectively. Also, assume the link capacity from the sender to the receivers is 10 Mb/s. The throughput performance with respect to different delay factors is shown in Fig. 16.

When the feedback link bandwidth is 1 Mb/s, the throughput performance is not degraded for this bandwidth asymmetry ratio. The reason is that the ACK size is usually about 40 Bytes, which is much smaller than the data packet size 1 kB, thus, the feedback link is not congested. However, the throughput performance decreases with increasing bandwidth asymmetry ratio. For example, the throughput drops to 73.81 from 103.31 when the feedback link bandwidth is 100 kb/s and the delayed SACK scheme is not used. The throughput increases with increasing delay factor and reaches the highest value when delay factor is three, which is close to the throughput for 1 Mb/s. Since TCP relies on the ACK clock to transmit packets, the delay factor can not be too large, thus, the throughput performance can be degraded for very high bandwidth asymmetry ratios. For example, the throughput decreases approximately 20% when the feedback link bandwidth is 64 kb/s and the delay factor is 4.

H. Scalability

Scalability is one of the most important metrics of IP multicast schemes. Usually, scalability is measured by the performance over different numbers of receivers in a multicast session. Schemes with good scalability can be applied to very large size of receivers. Here, we use throughput as the performance measurement. For $\mathrm{TRTT} = 30$ ms and $M = 10$, the resulting scalability is shown in Fig. 17, where we observe that the throughput is constant and does not decrease with N increasing from 50 to 1000. We can conclude that TCP-Peachtree has good scalability. The reasons are based on the facts.

- The *acknowledgment implosion problem* is solved by the ACK fusion procedure performed by the DRs of the hierarchical multicast groups and the sender.
- The *loss path multiplicity problem* can be solved by *local error recovery* and *NIL recovery*.
- The congestion control algorithm in TCP-Peachtree can recover multiple packet losses in one window of data so that TCP-Peachtree is very suitable to work in applications with high packet loss rates due to link errors.

V. Conclusion

In this paper, the TCP-Peachtree is proposed for reliable multicast in satellite IP networks. TCP-Peachtree uses a modified B+ tree-like hierarchical multicast group to solve the acknowledgment implosion and scalability problems in reliable IP multicast applications. Two new congestion control algorithms are also presented, i.e., *jump start*, and *quick recovery*, so that TCP-Peachtree is suitable for satellite IP networks with long propagation delays and high BERs. NIL segments are used to exploit the availability of network resources and recover lost packets on receiver side. The delayed SACK scheme is adopted to address the bandwidth asymmetry problems. Furthermore, a hold

state is developed to address persistent fades. Simulation results show that the congestion control algorithms in TCP-Peachtree perform better than that of the TCP-NewReno and improve the throughput performance during rain fades. It is also shown that TCP-Peachtree achieves fairness and also has high scalability.

REFERENCES

[1] O. B. Akan, J. Fang, and I. F. Akyildiz, "Performance of TCP protocols in deep space communication networks," *IEEE Commun. Lett.*, vol. l6, pp. 478–480, Nov. 2002.

[2] I. F. Akyildiz, G. Morabito, and S. Palazzo, "Research issues for transport protocols in satellite IP networks," *IEEE Pers. Commun.*, vol. 8, pp. 44–48, June 2001.

[3] ——, "TCP peach: A new congestion control scheme for satellite IP networks," *IEEE/ACM Trans. Networking*, vol. 9, pp. 307–321, June 2001.

[4] I. F. Akyildiz, X. Zhang, and J. Fang, "TCP-peach+: Enhancement of TCP-peach for satellite IP networks," *IEEE Commun. Lett.*, vol. 6, pp. 303–305, July 2002.

[5] M. Allman, D. Glover, and L. Sanchez, "Enhancing TCP over satellite channels using standard mechanisms," RFC 2488, Jan. 1999.

[6] H. Balakrishnan, V. Padmanabhan, S. Seshan, and R. Katz, "A comparison of mechanisms for improving TCP performance over wireless links," *IEEE/ACM Trans. Networking*, vol. 5, pp. 756–769, Dec. 1997.

[7] S. Bhattacharyya, D. Towsley, and J. Kurose, "The loss path multiplicity problem in multicast congestion control," in *Proc. IEEE INFOCOM'99*, New York, NY, Mar. 1999, pp. 856–863.

[8] J. H. Cui, L. Lao, D. Maggiorini, and M. Gerla, "BEAM: A distributed aggregated multicast protocol using bi-directional trees," in *Proc. IEEE ICC'2003*, Anchorage, AK, May 11–15, 2003, pp. 689–695.

[9] R. Elmasri and S. B. Navathe, *Fundamentals of Database Systems*, 3rd ed. Reading, MA: Addison-Wesley, 2000.

[10] K. Fall and S. Floyd, "Simulation-based comparisons of tahoe, Reno, and SACK TCP," *ACM Comput. Commun. Rev.*, vol. 26, pp. 5–21, July 1996.

[11] S. Floyd and T. Henderson, "The NewReno modification to TCP's fast recovery algorithm," RFC 2582, Apr. 1999.

[12] S. Floyd, V. Jacobson, and C. G. Liu, "A reliable multicast framework for light weight session and application level framing," in *Proc. ACM SIGCOMM'95*, Aug. 1995, pp. 342–356.

[13] T. R. Henderson and R. H. Katz, "Transport protocols for Internet-compatible satellite networks," *IEEE J. Select. Areas Commun.*, vol. 17, pp. 326–344, Feb. 1999.

[14] J. Hoe, "Improving the start-up behavior of a congestion control scheme for TCP," in *Proc. ACM SIGCOMM'96*, Aug. 1996, pp. 270–280.

[15] H. W. Holbrook, S. K. Singhal, and D. R. Cheriton, "Log-based receiver-reliable multicast for distributed interactive simulation," in *Proc. ACM SIGCOMM'95*, Cambridge, MA, Aug. 1995, pp. 342–357.

[16] V. Jacobson, "Congestion Avoidance and Control," Tech. Rep., Apr. 1990.

[17] M. Jung, J. Nonnenmacher, and E. W. Biersack, "Reliable multicast via satellite: Uni-directional vs. bi-directional communication," presented at the KiVS'99, Darmstadt, Germany, Mar. 1999.

[18] M. W. Koyabe and G. Fairhurst, "Performance of reliable multicast protocols via satellite at EHF with persistent fades," presented at the 7th Ka-Band Utilization Conf., Santa Margherita Ligure, Genoa, Italy, Sept. 2001.

[19] M. Koyabe and G. Fairhurst, "Reliable multicast via satellite: A comparison survey and taxonomy," Int. J. Satell. Commun., vol. 19, pp. 3–28, Jan./Feb. 2001.

[20] L. H. Lehman, S. J. Garland, and D. L. Tennenhouse, "Active reliable multicast," presented at the IEEE INFOCOM'98, San Francisco, CA, Mar. 1998.

[21] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow, "TCP Selective Acknowledgment Options," RFC 2018, Oct. 1996.

[22] M. Marchese, "Performance analysis of the TCP behavior in a GEO satellite environment," *Comput. Commun. J.*, vol. 24, pp. 877–888, 2001.

[23] V. N. Padmanabhan and R. H. Katz, "TCP fast start: A technique for speeding up web transfers," presented at the IEEE GLOBECOM'98 Internet Mini-Conf., Sydney, Australia, Nov. 1998.

[24] S. Paul and K. K. Sabnani, "Reliable multicast transport protocol (RMTP)," *IEEE J. Select. Areas Commun.*, vol. 15, pp. 407–421, Apr. 1997.

[25] I. Rhee, N. Balaguru, and G. N. Rouskas, "MTCP: Scalable TCP-like congestion control for reliable multicast," *Comput. Networks J.*, vol. 38, pp. 553–575, Apr. 2002.

[26] L. Rizzo and L. Vicisano, "RMDP: An FEC-based reliable multicast protocol for wireless environments," in *Proc. ACM Mobile Computer Communication Review*, vol. 2, Apr. 1998, pp. 23–31.

[27] P. Sinha, N. Venkitaraman, R. Sivakumar, and V. Bharghavan, "WTCP: A reliable transport protocol for wireless wide-area networks," presented at the ACM MOBICOM'99, Seattle, WA, Aug. 1999.

[28] B. Whetten and G. Taskale, "An overview of reliable multicast transport protocol II," *IEEE Network*, pp. 37–47, Jan./Feb. 2000.

[29] R. Yavatkar, J. Griffioen, and M. Sudan, "A reliable dissemination protocol for interactive collaborative applications," in *Proc. ACM Multimedia'95*, Nov. 1995, pp. 333–343.

Ian F. Akyildiz (M'86–SM'89–F'96) received the B.S., M.S., and Ph.D. degrees in computer engineering from the University of Erlangen-Nuernberg, Erlangen, Germany, in 1978, 1981, and 1984, respectively.

Currently, he is the Ken Byers Distinguished Chair Professor with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, and Director of Broadband and Wireless Networking Laboratory. He is an Editor-in-Chief of *Computer Networks* and for the newly launched *Ad Hoc Networks Journal* and an Editor for *ACM Journal of Wireless Networks*. His current research interests are in sensor networks, IPN Internet, wireless networks, and satellite networks.

Dr. Akyildiz received the Don Federico Santa Maria Medal for his services to the Universidad of Federico Santa Maria, in 1986. From 1989 to 1998, he served as a National Lecturer for ACM and received the ACM Outstanding Distinguished Lecturer Award in 1994. He received the 1997 IEEE Leonard G. Abraham Prize Award (IEEE Communications Society) for his paper entitled "Multimedia Group Synchronization Protocols for Integrated Services Architectures" published in the IEEE JOURNAL OF SELECTED AREAS IN COMMUNICATIONS (JSAC) in January 1996. He received the 2002 IEEE Harry M. Goode Memorial Award (IEEE Computer Society) with the citation "for significant and pioneering contributions to advanced architectures and protocols for wireless and satellite networking." He received the 2003 IEEE Best Tutorial Award (IEEE Communicaton Society) for his paper entitled "A Survey on Sensor Networks," published in the *IEEE Communication Magazine*, in August 2002. He also received the 2003 ACM Sigmobile Outstanding Contribution Award with the citation "for pioneering contributions in the area of mobility and resource management for wireless communication networks." He has been a Fellow of the Association for Computing Machinery (ACM) since 1996.

Jian Fang received the B.S., M.S., and Ph.D. degrees from Shanghai Jiao Tong University, Shanghai, China, and is working toward the Ph.D. degree from Georgia Institute of Technology, Atlanta.

Currently, he is a Research Assistant in the Broadband and Wireless Networking Laboratory, Georgia Institute of Technology. He was a Research Associate in the Department of Computer Science and Engineering, the Chinese University of Hong Kong, Shatin, for one year. After that, he became a Lecturer with Shanghai Jiao Tong University. His current research interests are in interplanetary Internet and satellite networks.