Hop-by-Hop Multicast Transport for Mobile Ad Hoc Wireless Networks

Manoj Pandey
Cisco Systems
mpandey@cisco.com

Daniel Zappala
Computer Science Department
Brigham Young University
zappala@cs.byu.edu

Abstract—**Multicast transport is a challenging problem because the source must provide congestion control and reliability for a tree, rather than a single path. This problem is made even more difficult in mobile ad hoc networks due to problems caused by contention, spatial reuse, and mobility. In this paper, we design a hop-by-hop multicast transport protocol, which pushes transport functionality into the core of the network. Although this requires per-flow state, a hop-by-hop approach simplifies congestion control, enables local recovery of lost packets, and provides low delay and efficient use of wireless capacity. We use a simulation study to demonstrate the effectiveness of this approach and compare its efficiency to application-layer multicast.**

I. Introduction

In this paper we consider the problem of multicast transport across mobile ad hoc wireless networks. It is well known that it is difficult to provide unicast transport over multiple wireless hops, due to contention, spatial reuse, mobility, and other challenges of wireless networks [1], [2]. Multicast transport is further complicated because a source must provide reliability and congestion control over a multicast tree, and different branches of the tree may lose different packets and may have different available bandwidths over time. Of primary concern with multicast is scalability – it is infeasible for a source to track state and receive feedback from all of the group members simultaneously.

Current approaches for solving this problem mirror the solutions tried on the Internet: (1) an end-to-end multicast transport layer built on top of an unreliable, network-layer multicast service, and (2) an application-layer multicast overlay built using unicast transport connections between participating hosts.

With a multicast transport layer, packets are sent to group members unreliably using a multicast routing protocol that operates at the network layer. The transport protocol then uses feedback from receivers in the form of ACKs or NACKs to regulate the source's sending rate and to provide end-to-end reliability [3], [4], [5]. The benefit of this approach is that network layer multicast is very efficient, since a tree can be built that minimizes hop count or power consumption. The downside is that it is very difficult to provide an efficient and scalable transport protocol that can manage the differing loss rates and congestion on the tree. Existing protocols slow down severely when congestion occurs, limiting the rate of the source to one packet at a time.

Application-layer multicast avoids some of the complexities of multicast transport by building an overlay using TCP (or other unicast transport protocol) connections [6], [7], [8], [9], [10], [11]. The advantage of this approach is that reliability and congestion control only needs to be provided between pairs of hosts, rather than for a whole tree at once. However, the distribution tree usually imposes greater delay (called stretch) and higher bandwidth usage (stress) as compared to a tree built at the network layer [12]. Because capacity and power are constrained in an ad hoc network, inefficiencies due to both stress and stretch should be avoided.

In this paper, we take a different approach by designing a hop-by-hop multicast transport protocol. Rather than pushing multicast transport to the edges of the network, where it becomes either complex or inefficient, we push transport functionality *into* the network. With hop-by-hop multicast transport, every node in the multicast tree performs both congestion control and reliability. Whenever a node on the tree forwards a packet, it uses credit-based congestion control to ensure that it does not overflow the packet queues of its children. In addition, each node on the tree caches packets for the application, and can repair losses for any group members that are downstream without involving the source.

Pushing more functionality into the network is feasible with an ad hoc wireless network, since wireless devices are usually inexpensive and new versions are often rapidly introduced. This provides a greater opportunity to explore novel network designs, when compared to wired networks. Moreover, ad hoc networks often utilize equipment from a single vendor and typically operate independently of the rest of the network, so interoperability among vendors or with the Internet is not needed. The downside of hop-by-hop protocols is that they typically require per-flow state in routers [13], [14], and this is difficult to scale to large numbers of flows. However, recent work shows that this may be feasible in core Internet routers [15], and this should certainly be less of a concern in ad hoc wireless networks, since they will typically have far fewer flows to handle.

Furthermore, using hop-by-hop transport has a number of advantages in ad hoc networks that make it an attractive alternative. First, since packets are carried on a network-level tree, it can provide low delay and high efficiency. Second, lost packets can be retransmitted by any upstream node that has the missing packets in its cache. This greatly reduces overhead in ad hoc networks, since ACKs or NACKs that must travel multiple hops contend for scarce wireless capacity.

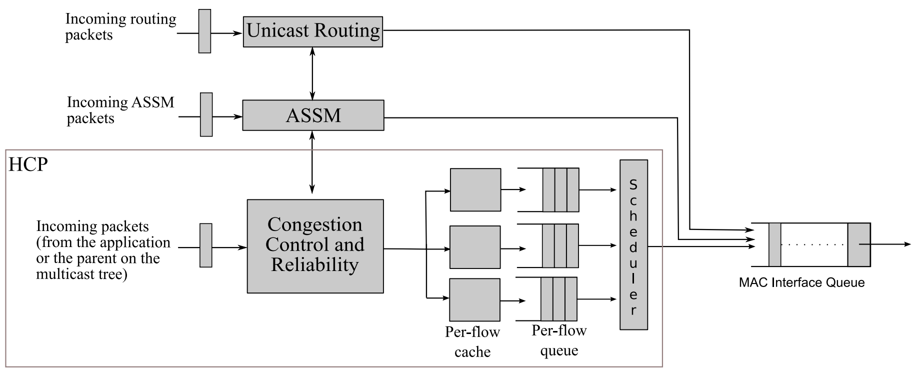


Fig. 1. HCP Architecture

Third, congestion control is simplified since each node only has to control its sending rate for its children in the tree, rather than having to control the rate to all group members.

The simplicity of hop-by-hop congestion control comes from having a feedback loop that operates over a single hop, instead of an entire network. Both of the previous approaches described above use an end-to-end feedback loop, in which the source's sending rate is controlled by ACKs or NACKs sent from a receiving node to a sending node. In wireless networks there is a greater chance that this feedback loop can become delayed or lossy, due to MAC retransmissions and contention. Irregular feedback results in the transport protocol making rate decisions with imperfect knowledge of network conditions. These problems are exacerbated when hosts move because mobility causes sudden changes in available bandwidth. An end-to-end protocol must take some time to probe the available bandwidth on an affected path before it can determine the appropriate sending rate.

By combining hop-by-hop congestion control with per-flow caching, we also enable receivers to obtain different rates for the same multicast source, rather than requiring the source to send at the speed of the slowest receiver. If a node on the multicast tree receives data faster than it can send it, rather than telling its parent to slow down, it caches the packets until it can send them. The size of the cache determines how much of a mismatch there can be between different reception rates. If there is a large difference between the incoming rate and the outgoing rate, then the cache will eventually fill, causing the parent node to slow down.

In this paper, we design HCP, a multicast transport protocol that provides hop-by-hop congestion control and end-to-end reliability with local repair. While the hop-by-hop approach has been previously used for unicast transport protocols, this is the first time this approach has been used for multicast. We use a simulation study to demonstrate that HCP can greatly improve multicast throughput relative to a peer-to-peer multicast overlay. In addition, when packets are lost, HCP is able to retransmit more quickly from a nearby cache, rather than waiting for the source to detect the loss and then retransmit. Our simulations also show how HCP is able to support different receiver rates within the same multicast tree. Finally, we show that only a small cache is needed to provide the benefits of local recovery.

II. Hop-by-Hop Transport

To provide hop-by-hop multicast transport, we have designed the HCP architecture and protocols shown in Figure 1. First, HCP uses the ASSM multicast routing protocol [16] to establish state and per-flow queues at each node on a multicast tree. Then, HCP uses per-flow, hop-by-hop congestion control to ensure that downstream buffers do not overflow; whenever it is safe to transmit a packet to the next hop it places the packet on a per-flow queue that is serviced by a scheduler. If any packets arrive for HCP that do not have associated per-flow state, they are discarded. If a group member loses packets due to contention, HCP can retransmit them from a per-flow cache on one of the upstream nodes.

The HCP scheduler uses fair queueing [17] to determine the order in which packets are sent from each of the per-flow queues. This ensures fairness among all of the multicast flows traversing this node. Fairness over wider areas is addressed by existing research [18] and is out of scope for this paper. To prevent the MAC queue from overflowing, the scheduler delivers only one packet at a time. The MAC layer then uses a callback function to inform the scheduler when it is ready for the next packet.

HCP cooperates with ASSM to forward packets semi-reliably on the multicast tree. When a new packet arrives, HCP determines how to forward the packet by getting the forwarding entry from ASSM. This entry contains a list of child nodes to which it should send the packet. Usually, multicast packets are forwarded using broadcast, but this causes multicast to suffer a high loss rate when it must contend with TCP traffic, which uses the RTS/CTS exchange. Instead, we use a form of semi-reliable broadcast. HCP cycles through the child nodes in round-robin order. Each time it forwards a packet it chooses one of the children and forwards it the packet using the same RTS/CTS exchange as a unicast packet. This ensures that one child receives the packet reliably, and the other children attempt to receive the packet through promiscuous listening. As packets are forwarded, each child will get some packets reliably and others by snooping.

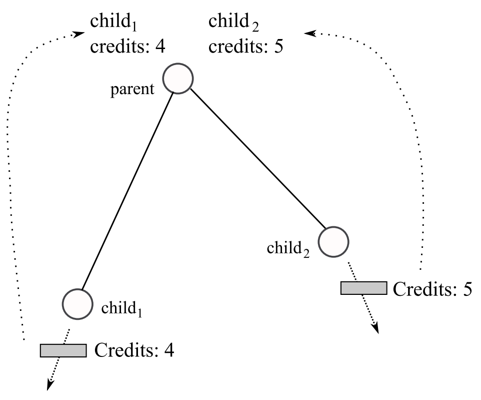


Fig. 2. Passive Feedback for Credit-Based Congestion Control

A fortunate consequence of using semi-reliable broadcast is that HCP can detect when a child moves away, since the RTS/CTS exchange will fail. In this case, HCP removes the forwarding state for that child. If this is the only child downstream from this node, HCP leaves the rest of its state in place and instead waits for a timeout before it deletes the state. This allows the route to be repaired without losing all the packets in this node's cache.

A. Congestion Control

HCP uses credit-based congestion control to ensure that it does not overflow the per-flow caches at the children directly downstream from each node on the tree. With credit-based congestion control, each node maintains a pool of credits for each flow, indicating the amount of buffer space available to that flow at the downstream node. Each time a packet is successfully transmitted for that flow, the pool is decremented. Transmitting is allowed as long as the pool indicates there is space available downstream.

We make several changes to this basic algorithm to take advantage of the nature of wireless communication and to provide multicast congestion control. First, the cache stores packets even after they have been transmitted, so the amount of available buffer space is equal to the number of packets in the cache that have already been sent downstream. These packets can be safely discarded if a new packet arrives. Second, as shown in Figure 2, a node uses passive feedback to learn about the available buffer space at a child. Whenever a node sends a packet downstream, it includes the current buffer space for the flow in a shim header. When the parent node overhears the packet, it is able to synchronize the credit pool for that child with this amount. Finally, a node may have more than one child. It tracks the available credits separately for each child, and only sends a packet downstream if all children have available credits. Thus the total cache size limits the extent to which the multicast tree can accommodate receivers with differing rates

Several complications arise due to our use of passive feedback. First, promiscuous listening is not completely reliable, so a node may miss some feedback messages. This is acceptable, because it can synchronize the credit pool with a subsequent message. As long as passive feedback succeeds most of the time, this scheme works well. However, there may be cases where a parent node is unable to hear several consecutive packets sent downstream. When this occurs, it is possible for the parent node to believe that there is no buffer space available at a child, when in fact there is. Meanwhile, the child may be unable to correct its parent if it has no packets to transmit. To prevent flows from stopping in this case, child nodes without any data to transmit periodically send a message to parents informing them of their current number of available credits.

B. Reliability

Because HCP uses semi-reliable broadcast when forwarding packets, it is likely that some packets will be lost as they are forwarded to receivers. To provide end-to-end reliability, HCP receivers monitor the sequence numbers of incoming packets, detect sequence number gaps, and request retransmissions from upstream nodes using NACKs. Each node on the multicast tree caches the most recent packets it has forwarded and can respond to these requests.

To send a NACK, a receiver creates a *request* container that contains a list of the missing packets and inserts this container into an ASSM Join message. The list of packets is a set of tuples of the form (l_1, l_2), where l_1 is the first sequence number and l_2 is the last sequence number of the gap. As the Join message travels upstream, each node on the tree examines the list and determines if its cache contains any of the requested packets. If it does, it schedules these packets for retransmission and removes the appropriate tuples from the list. If there are still tuples remaining, then it signals ASSM to continue forwarding the Join message upstream. In the worst case, the source receives the NACK and resends any missing data.

When retransmitting packets, an HCP node follows several policies. First, it always resends data using an RTS/CTS exchange to the child that sent it the request. This ensures that the packet makes it at least one step closer to the receiver that is missing the packet. Second, each node gives equal priority to retransmissions and original transmissions of data currently in the flow's buffer. This allows receivers experiencing loss to get the retransmission, while not penalizing receivers without loss too heavily.

III. Results

We use simulations to evaluate HCP and to compare it to application-layer multicast in an ad hoc network. We have implemented both HCP and ASSM in $ns2$, and use AODV [19] for the unicast routing protocol. The per-flow caches in HCP have a default size of 100 Kbytes, and ASSM uses a refresh period of 1 second. Unless specified otherwise, all our simulations use a default data rate of 11 Mbps and a packet size of 1024 bytes.

To compare HCP to application-layer multicast, we have built a simple protocol that uses an overlay of ATP [20] connections to connect group members. We use ATP rather than TCP, since ATP is designed to avoid many of the problems with unicast transport in ad hoc networks. ATP uses a rate-based congestion control algorithm that measures link characteristics on the forward path to set the proper rate. It is then able to send at the full available rate immediately after a route change. ATP also aggregates ACKs into epochs in order to reduce the overhead of per-packet feedback.

We implement two variants of our simple overlay application: optimal and random. We build an optimal overlay in two steps. First, we build a logical mesh that connects all nodes in the multicast group, where the weight of each link in the mesh is equal to the shortest distance between those two nodes in the ad hoc network. Next, we calculate a minimum spanning tree over this mesh; the links in the spanning tree represent an ATP connection in the overlay. We build a random overlay by choosing random pairs of group members and connecting them with ATP. This approximates an overlay that has been perturbed over time by mobility.

We first evaluate HCP on several simple topologies, to verify the soundness of our design. Due to space limitations this data is not included in this paper. Our results also show that HCP minimizes loss when nodes are mobile, since an upstream node uses RTS/CTS to detect when a downstream child has moved. This enables the upstream node to stop transmitting and wait for a new route to be established.

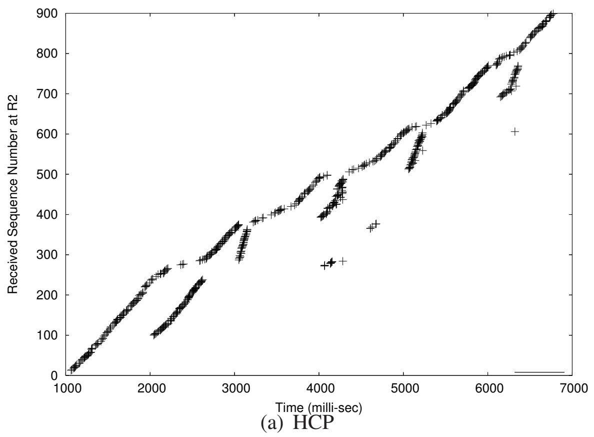
Figure 3 shows a typical packet trace taken at a receiver in a multicast tree. The tails shown in this figure for HCP are packets that are lost and then retransmitted at a later time. Due to its use of semi-reliable broadcast, HCP can suffer substantial loss due to contention when an upstream node has several downstream children. This occurs because only one of the children receives each packet reliably and the others must use promiscuous listening. Despite this source of loss, HCP still achieves double the throughput of a multicast overlay, due to its shorter feedback loop. Whenever HCP loses a packet due to contention, the affected group member can request retransmission from the cache immediately upstream of where the loss occured. An overlay that uses TCP connections, on the other hand, must wait for the packet to be retransmitted by the source, and the source slows down as it invokes its congestion avoidance mechanism.

Our results also verify that HCP is able to share bandwidth fairly among multiple groups using the same path. Moreover, when the network contains some nodes that use a lower wireless rate (1 Mbps instead of 11 Mbps), HCP is able to accommodate both rates in the same multicast tree, due to its combination of hop-by-hop rate control and per-flow caching. Figure 4 shows the instantaneous throughput received by two receivers in the same multicast tree, calculated over a sliding one second window. HCP is able to send at a higher rate for the 11 Mbps receiver while also providing a lower rate for the 1 Mbps receiver. Packets are buffered on the slower path as needed.

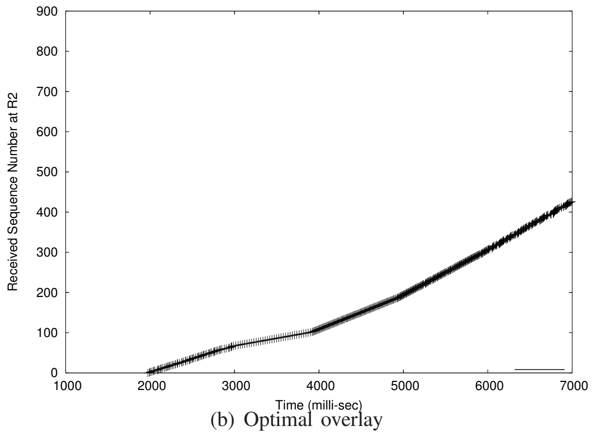
A. *Efficiency*

One of the advantages of HCP as compared to an application-layer overlay is that it can be more efficient. To examine this in more detail, we measure the number of hops in the multicast tree, L_m, which in most cases is equivalent to the number of times a packet must be transmitted to reach all of the group members. The lower this value, the more efficient the multicast tree, and hence the more bandwidth there is available for the group. Since HCP uses semi-reliable broadcast, its efficiency is actually lower than the value of L_m for its tree.

For this experiment we place 50 nodes randomly in a field of size $1000m^2$. We randomly choose a source and a set



(a) HCP



(b) Optimal overlay

Fig. 3. Packet Traces with Contention

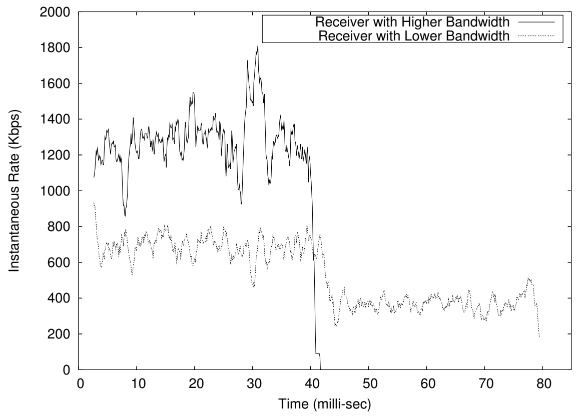


Fig. 4. HCP: Multiple Rates in a Single Multicast Tree

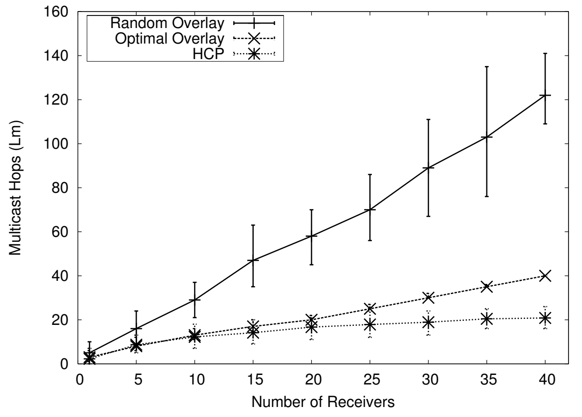


Fig. 5. L_m as a function of group size

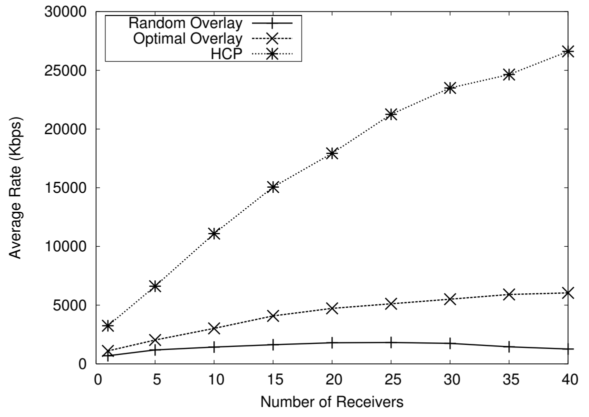


Fig. 6. Throughput as a function of group size

of group members, varying the size of the group from 1 to 40 members. After the multicast tree has been computed, we count the number of links in the tree. We repeat this simulation for 25 seeds of the random number generator for each group size, and plot the average, minimum, and maximum values of L_m for each multicast protocol. We also plot the throughput for each case.

Figure 5 shows that HCP is very efficient, since it uses ASSM to build a shortest-path tree. The penalty for an optimal overlay is close to twice the shortest-path tree as the group becomes larger. Moreover, the overlay must be recomputed periodically in order to maintain its optimality. The random overlay shows how much worse the overlay may become if there is significant mobility before it is rebuilt. This is important because greater efficiency can lead to higher throughput. Figure 6, shows that HCP puts this efficiency to good use, providing up to five times the throughput of even an optimal overlay for larger groups.

B. The Impact of Cache Size

An important factor in the performance of HCP is its cache size. In all previous experiments, each flow is allocated a cache of of 100 Kbytes. Since mobile devices may have memory constraints, we explore the effectiveness of HCP as the cache size changes. For this section, we use the topology shown in

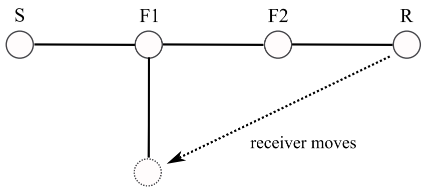


Fig. 7. Cache Size Topology

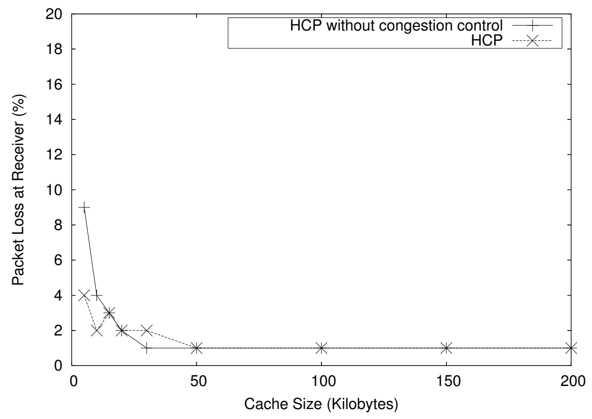


Fig. 8. HCP: Packet Loss versus Cache Size

Figure 7, which consists of a chain of four nodes. We vary the cache size at each node from 5 to 200 Kbytes and transfer a 5 GB from the source to the receiver.

We first examine the effects of cache size when there is contention, by keeping all nodes static. When contention occurs, an HCP receiver must request missing packets using a NACK, and the nearest upstream node that has the available data schedules a retransmission. As the cache size decreases, there is a greater chance that a node does not have the requested data and must forward the request upstream. With small enough caches, the source will eventually need to handle all retransmission requests.

Figure 8 shows the percentage of lost packets for the receiver. Note that there is a steady rate of about 1% loss regardless of cache size, due to contention along the chain. More loss occurs as the cache size decreases, because there are times when an upstream node thinks there is available space downstream when there isn't (due to the failure of passive feedback). If the cache is greater than 30 Kbytes, this never occurs, but loss rises to about 4% when the cache is 5 Kbytes. We also plot loss for when HCP uses no congestion control, so that packets may also be lost because the source sends too fast. This shows that congestion control primarily helps when the cache is small, since loss in this case can rise to about 10%.

We next examine the effects of cache size when a receiver moves. In this simulation, the receiver moves from its first location to the second location indicated in the topology. In this case, the receiver will miss some packets, then request them when it joins the tree again. As the cache size decreases, there is a greater chance the lost packets will need to be

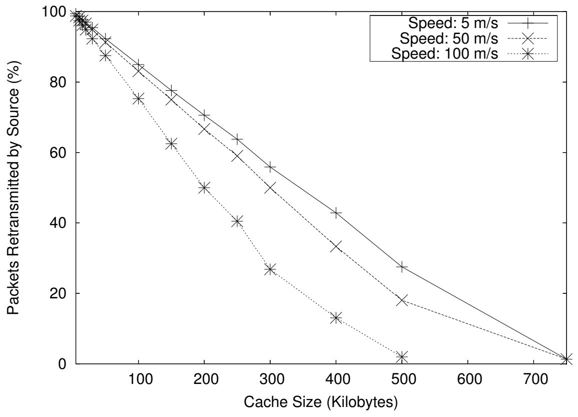


Fig. 9. HCP: Source Retransmission versus Cache Size

requested from the source.

Figure 9 plots the number of lost packets that must be retransmitted by the source for different movement speeds and cache sizes. Two trends are evident: more buffering is needed when the receiver moves more slowly, and larger cache sizes provide more local recovery. The slower a receiver moves, the more packets it may miss, so a larger cache helps it to catch up once it is reconnected to the tree. Although a large cache size is needed to provide local recovery, Figure 8 shows that the percentage of loss with a cache of 50 Kbytes is very small.

We examine traces of this data to determine the percent of lost packets retransmitted by each upstream node. We find that as the cache size decreases, fewer packets are retransmitted by the direct upstream forwarder, $F2$, and they are instead retransmitted by the source. No packets are retransmitted by $F1$. This occurs because all nodes use the same cache replacement algorithm, which discards the oldest packets first. A different cache replacement scheme could allow greater local repair, and smaller cache sizes, by retaining different sets of packets at $F1$ and $F2$.

IV. Conclusion

Our simulations demonstrate some of the advantages of a hop-by-hop multicast transport protocol in an ad hoc network. Because HCP uses credit-based, hop-by-hop congestion control, it can quickly determine the appropriate sending rate when network conditions change. Using semi-reliable broadcast enables many packets to be transmitted reliably by the MAC layer. Using caching provides local recovery and supports different rates within the same multicast tree. All of this is built on top of a shortest-path multicast tree, which provides greater efficiency than an application-layer overlay.

We have identified several areas where additional work needs to be done. First, it would be helpful to reduce the amount of packet loss due to contention. One possibility would be to use rate-based congestion control to slow down the forwarding nodes, so that promiscuous listening succeeds more often. Another possibility is to substitute a reliable MAC-layer multicast for our semi-reliable algorithm; BMW [21] is likely a good starting point.

Another important issue is to enable faster recovery when the tree breaks due to mobility. A receiver should be able to measure the average inter-packet delay and then trigger a new Join if this delay grows too large.

Finally, we would like to further study the tradeoffs of buffering versus local recovery. For example, perhaps a parent and a child should cache different packets, so there is chance that if one of them is missing a packet the other one has it. In the absence of a cache coordination protocol, random cache replacement might provide this benefit without any overhead.

References

[1] Z. Fu, P. Zerfos, H. Luo, S. Lu, L. Zhang, and M. Gerla, "The Impact of Multihop Wireless Channel on TCP Throughput and Loss," in *IEEE INFOCOM*, 2003.

[2] J. Liu and S. Singh, "ATCP: TCP for Mobile Ad Hoc Networks," *IEEE Journals on Selected Areas in Communications*, vol. 19, no. 7, pp. 1300–1315, 7 2001.

[3] K. Tang, K. Obraczka, S.-J. Lee, and M. Gerla, "A Reliable, Congestion-Controlled Multicast Transport Protocol in Multimedia Multi-hop Networks," *5th International Symposium on Wireless Personal Multimedia Communications*, pp. 252–256, Oct, 2002.

[4] S. Wen, J. Griffioen, and K. Calvert, "Calm: congestion-aware layered multicast," *Open Architectures and Network Programming Proceedings, 2002 IEEE*, pp. 179–190, 2002.

[5] V. Rajendran, K. Obraczka, Y. Yi, S.-J. Lee, K. Tang, and M. Gerla, "Combining Source- and Localized Recovery to Achieve Reliable Multicast in Multi-Hop Ad Hoc Networks," *IFIP-TC6 Networking Conference 2004*, pp. 112–124, May, 2004.

[6] C. Gui and P. Mohapatra, "Overlay multicast for manets using dynamic virtual mesh," *Wirel. Netw.*, vol. 13, no. 1, pp. 77–91, 2007.

[7] M. Ge, S. Krishnamurhty, and M. Faloutsos, "Application versus network layer multicasting in ad hoc networks: the alma routing protocol," *Ad Hoc Networks Journal*, vol. 4, no. 2, pp. 283–300, 2006.

[8] J. Xie, R. R. Talpade, A. Mcauley, and M. Liu, "Amroute: ad hoc multicast routing protocol," pp. 429–439, 2002.

[9] K. Chen and K. Nahrstedt, "Effective location-guided tree construction algorithms for small group multicast in MANET," in *IEEE INFOCOM*, June 2002.

[10] L. Xiao, A. P. Patil, Y. Liu, L. M. Ni, and A.-H. Esfahanian, "Prioritized Overlay Multicast in Mobile Ad Hoc Environments," *IEEE Computer 37(2): 67-74*, 2004.

[11] A. Passarella and F. Delmastro, "Usability of legacy p2p multicast in multihop ad hoc networks: an experimental study," *EURASIP J. Wirel. Commun. Netw.*, vol. 2007, no. 1, pp. 38–38, 2007.

[12] Y.-H. Chu, S. G. Rao, and H. Zhang, "A case for End System Multicast," in *Measurement and Modeling of Computer Systems*, 2000, pp. 1–12. [Online]. Available: citeseer.ist.psu.edu/article/chu01case.html

[13] P. P. Mishra and H. Kanakia, "A Hop by Hop Rate-Based Congestion Control Scheme," in *ACM SIGCOMM*, 1992.

[14] C. Özveren, R. Simcoe, and G. Varghese, "Reliable and Efficient Hop-by-Hop Flow Control," in *ACM SIGCOMM*, 1994.

[15] A. Kortebi, L. Muscariello, S. Oueslati, and J. Roberts, "On the Scalability of Fair Queueing," in *ACM HotNets*, 2004.

[16] M. Pandey and D. Zappala, "Scalable Multicast Routing for Ad Hoc Networks," in *International Workshop on Localized Communication and Topology Protocols for Ad hoc Networks*, September 2008.

[17] A. Demers, S. Keshav, and S. Shenker, "Analysis and Simulation of a Fair Queueing Algorithm," in *ACM SIGCOMM*, 1989.

[18] K. Xu, M. Gerla, L. Qi, and Y. Shu, "Enhancing TCP Fairness in Ad Hoc Wireless Networks Using Neighborhood RED," in *ACM MobiCom*, 2003.

[19] C. Perkins and E. Royer, "Ad Hoc On Demand Distance Vector (AODV) Algorithm," in *IEEE Workshop on Mobile Computing Systems and Applications*, 1999.

[20] K. Sundaresan, V. Anantharaman, H.-Y. Hsieh, and R. Sivakumar, "ATP: a reliable transport protocol for ad-hoc networks ," *Mobihoc*, 2003.

[21] K. Tang, K. Obraczka, S.-J. Lee, and M. Gerla, "A Reliable, Congestion-Controlled Multicast Transport Protocol in Multimedia Multi-hop Networks ," in *WPMC*, 2002.