Preemption Mechanisms for Push-to-Talk in Ad Hoc Networks

Erlend Larsen*, Lars Landmark*, Vinh Pham*, Paal E. Engelstad† and Øivind Kure*
*Q2S NTNU
†SimTel (Telenor/Simula)
Email: erl@unik.no, larsla@q2s.ntnu.no, {vph, paalee, okure}@unik.no

***Abstract*—Using push-to-talk applications in ad hoc networks is not straightforward. There are no inherent mechanisms to support priority of the voice traffic, to avoid great jitter and packet loss in face of large background traffic loads. This paper presents three preemption mechanisms that can be applied to support push-to-talk traffic in multi-hop ad hoc networks. The mechanisms differ in the way the background traffic is treated: discard, buffering and inter-scheduling. It is shown that there is a trade-off between the impact on the background traffic and the service for the push-to-talk traffic. Discarding or buffering the background traffic leaves the push-to-talk traffic with very little impact by the background traffic, while inserting the low priority packets in the interval between the high priority packets incurs some cost to the push-to-talk traffic.**

I. Introduction

The coordination of emergency and military operations has traditionally been done by Push-to-Talk (PTT) using two-way radio transceivers (walkie-talkies). With a single push on a button, the user switches from voice reception mode to transmit mode. PTT is a half-duplex method, meaning that when a sender transmits, it is unable to hear other radios transmitting at the same time. This inability to interrupt has made PTT best suited for quick communication exchanges between users.

Contrary to the low capacity analog communication supported by walkie-talkie systems, Mobile Ad Hoc Networks (MANETs) are able to support high capacity digital communication in environments where no network infrastructure is available. Because of the important coordination function of PTT, it is vital to support this service also in MANETs. In some circumstances, the PTT service can mean the difference between life and death, e.g. when calling for support or alerting of immediate danger. The combination of inability to interrupt and different urgency of the PTT calls should be reflected by the network in terms of service priority.

Push-to-talk is a one-to-many service, i.e. there is one sender and several receivers. For such applications, multicast can be a more efficient distribution method than unicast. With unicast, each packet is forwarded to one single destination. For each receiver a unique packet must be created and forwarded. An advantage of multicast is that one packet transmission can be received by multiple nodes, and then forwarded by these nodes, distributing the information to the whole or larger parts of the network. However, this efficiency makes for less reliable link layer transmissions, due to more receivers per transmission, where unicast with one receiver can rely on acknowledgment of each packet.

Voice is a traffic type with high network service demands, especially in terms of delay/jitter and loss rate. In a MANET, the end-to-end voice communication is faced with several challenges, including interference, packet loss and congestion. In addition, multicast traffic is troubled by self-interference when a received packet is transmitted almost simultaneously by several forwarders.

This paper proposes three mechanisms based on preempting the lower priority traffic and compares these to the well known priority queuing mechanism. Through simulations it is shown how preempting the lower priority traffic can increase the network performance for the push-to-talk voice traffic.

The rest of the paper is structured in the following way. First, in Section II, the problem of using PTT in ad hoc networks without priority is explained. Second, in Section III, the well known priority queuing mechanism is presented. A first effective preemption mechanism is introduced in Section IV, and then two more gentle preemption mechanisms are investigated in Section V. In Section VI, the behavior of the preemption mechanisms is scrutinized when TCP is used as background traffic transport protocol. Related work is presented in Section VII, and finally, in Section VIII, the conclusions of this paper is presented.

II. Received network service for push-to-talk

The normal network behavior is investigated in this section, and then the results documenting the service problem for PTT in ad hoc networks are shown. First, the simulation setup is presented, and then the results documenting the impact of the background traffic on the network service are shown.

Ns-2 [1] version 2.33 was used to run simulations evaluating the mechanisms and solutions presented in this paper. Unless otherwise specified, the following settings were used for the simulations: The IEEE 802.11 MAC was used for medium access with 2 Mbps data rate and 1 Mbps basic rate. The interface queue size was 100 packets. The interference radius was 550 m, and the transmission radius was 250 m. OLSR was used as routing protocol, and link layer notification was enabled. The simulation topology was 30 nodes moving in a 1500 x 300 m^2 area using random direction with reflection mobility model at a constant velocity of 5 m/s generated by the tools in [2]. The nodes changed direction every 10 s $\pm$ 5 s,

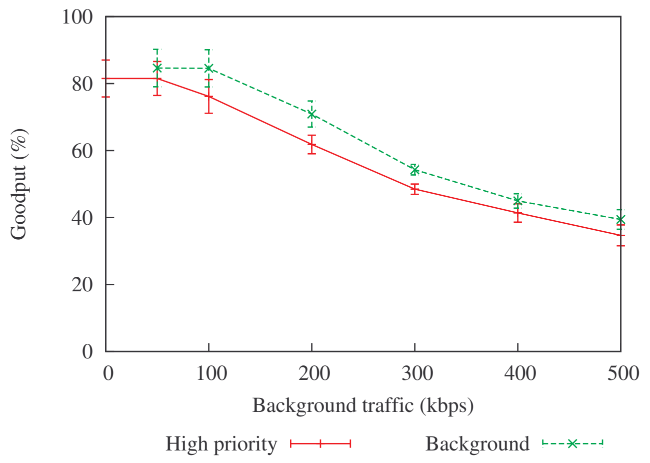

Fig. 1. Normal received network service for push-to-talk traffic with increasing background traffic load.

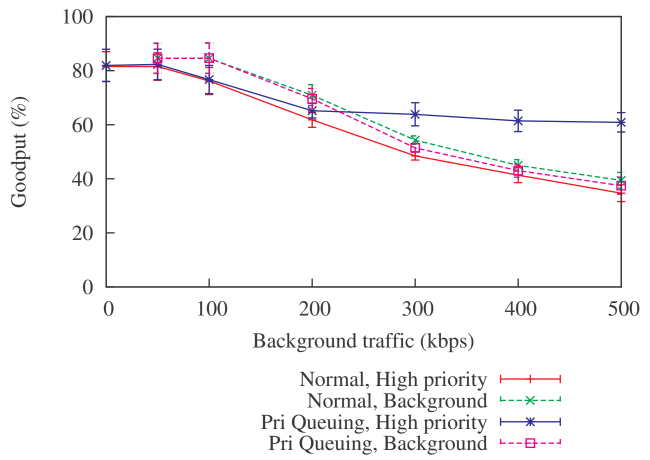

Fig. 2. Goodput with and without the queue priority mechanism.

and the simulations lasted 600 s. Each simulation configuration was run 10 times with different random seed (heuristic) and topology, and the same 10 random topologies were used for all simulations for comparison fairness. The confidence intervals are given with a confidence coefficient of 95%.

The traffic pattern was as follows: At 10 s, the background traffic starts. High priority traffic at constant bit rate 5.3 kbps in 20 byte multicast packets, with an interval of 30 ms, was transmitted in sessions of 5 seconds. This traffic rate corresponds to the G.723.1 voice encoding standard. Simplified Multicast Forwarding (SMF) [3], a mesh-based efficient flooding protocol, was used to forward the multicast traffic. The Source-specific Multi Point Relay (S-MPR) forwarding algorithm was used [4]. For each session, a new random node was designated as the sender, and a 5 s pause separated the sessions. The background traffic was unicast, where all the nodes were divided into two groups. Each node sent traffic to all other nodes in the same group. The packet size for the background traffic was 64 bytes, and UDP was used as transport protocol for both the multicast and the unicast traffic. Measurements were started at 60 s and continued until 590 s.

In the "Normal" case, where there are no mechanisms in place to enhance the service for the priority traffic, the goodput results with increasing background traffic (Fig. 1) show that the priority traffic without any competition from the background traffic manages 80% goodput. The 20% loss is caused by collisions and mobility. As the background traffic is introduced and increased, the priority traffic is impacted very negatively, because the collision rate increases and the interface queues begin filling up, causing tail drops. Preferably, the performance for the priority traffic should be kept at the same level as without any background traffic.

In the following sections, mechanisms that treat packets differently by the priority they are assigned, are analyzed. First, the well known queue priority mechanism is presented. Then, various preemption mechanisms are proposed and analyzed.

III. Priority queuing

When packets have been processed by the routing layer, they are passed through the link layer and queued in the interface queue on the MAC layer. The interface fetches the packet at the head of the queue and transmits it. Normally, no difference is made between packets with different priority classification. This can result in high priority packets being inserted in the interface queue behind several low priority packets, and delayed considerably. It also risks being discarded by tail drop if the queue gets full.

A solution to this problem is to place the new packets in the queue according to priority, so that all the high priority packets are placed in front of any low priority packets. This mechanism is part of the interface queue behavior. Placing the high priority packets in front of the lower priority packets ensures that no high priority packets will be dropped from the queue before all the low priority packets have been discarded. Also, the high priority packets are not delayed by the low priority packets.

The cost of this mechanism is taken by the low priority traffic. The queue tail drops impact the low priority traffic first, since no high priority packets are discarded unless the queue only contains high priority packets. In fact, the low priority traffic risks starvation if there is capacity only for the high priority traffic. Also, the background traffic always has to wait until all higher priority packets have been transmitted, leading to higher delay and jitter.

Simulations with and without the priority queuing mechanism have been run. With the priority queuing, the goodput (Fig. 2) for all traffic initially drops the same way as without priority queuing. This is due to an increase in collisions for the high priority traffic. However, with the priority queuing, the high priority traffic stabilizes at a background traffic load of 200 kbps and higher. This is because above 200 kbps all new background traffic is lost due to queue tail drops. This is confirmed through observations showing that as the background load increases past 200 kbps, the background traffic losses increase linearly. Without the priority queuing, the priority traffic starts experiencing tail drops as the load increases above 100 kbps, while using the priority queuing avoids all the high priority packet losses due to tail drop.

IV. Preemption by discard

While the queue priority mechanism is able to limit the background traffic in the network as the network approaches congestion, background traffic will still be transmitted in the network. The nodes that are not originating or forwarding the high priority traffic are free to transmit background traffic. So are also the other nodes, after transmitting their high priority packets. Thus, the high priority traffic has to compete with the low priority traffic on the medium, and this reduces the available bandwidth for the high priority traffic.

Preempting the background traffic from the network when the high priority traffic is being sent, is a way to further prevent interference from the background traffic. A crude form of preemption is to discard all the lower priority traffic. Each node discards the background traffic instead of forwarding it during the high priority session. If the lower priority traffic is voice, it is sensible to discard the traffic, as it will not be heard at the receiving nodes, and only expend network resources.

The initialization and end phases are challenging with the discard mechanism. The initialization phase starts with the source of the high priority traffic beginning to send packets down through the routing layer into the interface queue. At the insertion into the interface queue, the preemption mechanism is activated on this first node, and all the lower priority packets are discarded until the preemption times out. It is important that priority queuing is activated, not delaying the high priority packets out of the source node. The high priority packet is transmitted on the medium, and the neighbor nodes hear the packet. Some of the neighbors also forward the packet further out in the network. As the nodes hear the high priority packet, the preemption is activated, and the node stops transmitting low priority traffic. The preemption mechanism should be implemented between the interface queue and the MAC layer, so that the low priority packets already in the interface queue can be easily dropped, instead of being transmitted. Finally, after the first few packets of the high priority traffic flow have been multicasted, all nodes have activated the preemption, and no lower priority traffic is transmitted any longer.

The end phase, when the high priority session is over, represents another challenge. It is difficult for any given node to determine the end of the session, unless the application sends out a disconnect notification. Using a timeout after the last received high priority packet is a simple way to detect the end of the high priority session, but this timeout must be larger than the space between two consecutive packets, since the packets may arrive with different delays, or may even be lost. Although the timeout value could be optimized through measuring the delay between packets, a one second timeout is suggested in this paper, for simplicity.

A benefit of employing the discard mechanism is that any competition between the high priority traffic and the

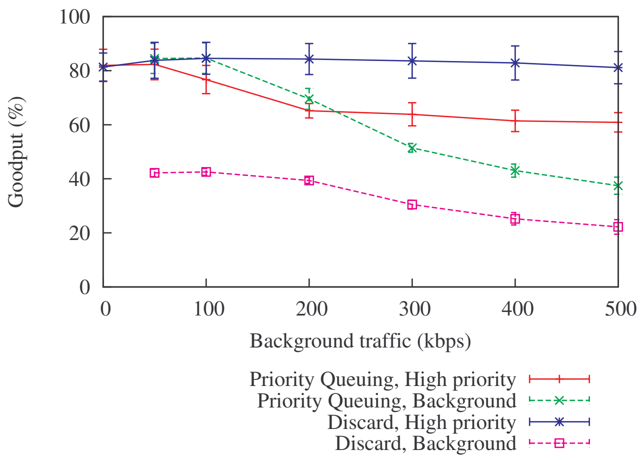

Fig. 3. Goodput for the discard mechanism.

background traffic is effectively avoided. Thus, the resulting goodput for the high priority traffic should be equal to (or close to) the goodput without any background traffic in the network. The cost, on the other hand, are all the discarded background packets that would otherwise have reached their destinations, including traffic already en route to the destination. In addition, since the network is not being fully utilized during, and immediately after, the high priority session, a lot of network resources are wasted.

The preemption mechanisms are dependent on priority queuing to work as proposed. Therefore, these mechanisms have been simulated using priority queuing and are compared to the priority queuing results. Looking at the results for increasing background traffic (Fig. 3), the discard mechanism is very efficient in keeping the high priority goodput at the same level as without the background traffic, but it does so at the expense of the background traffic. Using the discard preemption mechanism, the background traffic sees a reduction from over 80% goodput with the priority queuing, to well below 40%.

Due to the pauses between the high priority traffic sessions, the goodput for the background traffic is as high as 35%. The background traffic goodput is completely relient on pauses in the high priority traffic, and the gain in the priority traffic goodput comes at the expense of the background traffic. Thus, there is no definite answer to what the better mechanism is, when comparing the discard and priority queuing mechanisms, since it is dependent on how much one is willing to penalize the background traffic, to achieve lower loss and delay for the priority traffic.

V. Improved preemption mechanisms

It may be difficult to choose either preemption or only priority queuing, since it depends on how the importance of the lower priority traffic is weighed. In this section, attempts are made to find a “both ways” solution, which gives good priority for PTT and, at the same time, low consequences for the background traffic (i.e. mechanisms that help the background traffic).

A. *Buffering of the background traffic*

Instead of discarding the lower priority packets set for transmission during a high priority session, these packets could be held back in the interface queue until the end of the session, and then transmitted. This mechanism is called preemption with buffering. The issues with initializing and ending the preemption are the same as for the discard mechanism. The priority queuing is also required to keep any higher priority packets from being held back in the interface queue along with the lower priority packets, when the preemption is in effect.

An advantage of the buffering mechanism is that the low priority packets created during the high priority session can be transmitted after the session ends, instead of this information being lost. Thus, the goodput would increase. Another advantage is that any packets caught by a high priority session while en route to the destination would not be discarded. These packets have already spent network resources to come somewhere along the path, and so to discard the packets would be to waste these resources.

The mechanism will work best with a low load for the low priority traffic, with a short high priority session and with a large queue, since the queue may fill up fast if the load is high, or if the priority session lasts for a long time. Packets arriving after the queue is filled up will have to be discarded, and the advantage of the buffering approach is reduced.

When packets buffered in the queue have to be discarded, a choice must be made to either discard the oldest or the newest low priority packets. Discarding the newer packets would spare the packets already on the way to the destination, as the forwarding nodes may produce new packets, filling the queue. Discarding the old packets would mean losing the gain of storing already forwarded packets. However, protocols relying on packet acknowledgments, such as TCP, would benefit from discarding the old low priority packets, since these probably would time out before the high priority session is finished.

The buffer mechanism may hold the low priority packets for extensive periods of time. The mobility can cause the next hop of a low priority packet to be gone by the time the packet is due to be transmitted. A packet held in the buffer for a longer period of time will be more susceptible to have lost its next hop. This calls for a mechanism to enable route lookup and next hop insertion right before transmission, instead of the regular way of doing this before inserting the packet into the interface queue. This mechanism is explained below.

B. *Ingress queuing*

All packets to be transmitted by an ad hoc node have to be assigned a next hop by the routing protocol. The normal function is that the routing protocol assigns the packet a next hop, and then puts it in the interface queue. Here, the packet may stay for quite some time, before it reaches the head of the queue. If there is mobility in the network, there is a chance that the next hop assigned by the routing protocol is no longer reachable. The problem increases with higher mobility and with more packets in the queue. This forms a vicious circle where more packets in the queue leads to even more packets in the queue, only limited by the queue length.

Ingress queuing [5] remedies this problem through queuing the packets to be forwarded before assigning them a next hop. The next hop decision is taken as the packet is about to be transmitted, instead of doing so before the queue insertion. Thus, the packet is routed using the current topology information, instead of at the time of queuing.

The ingress queuing mechanism works only on unicast packets, since multicast and broadcast packets are assigned a special broadcast next hop. However, indirectly it works to the advantage for the multicast traffic, since the reduction of the number of unicast packets suffering retransmissions reduces the risk of collisions for the multicast traffic.

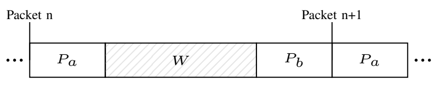

Fig. 4. Preemption with a window to transmit the low priority packets.

C. *Low priority window*

The Low Priority Window (LPW) preemption is the third proposed mechanism for preempting the background traffic, enhancing the buffering mechanism above. Considering that the high priority packets move in waves throughout the network, the interval between any two consecutive high priority packets could be used to transmit the low priority traffic.

The preemption is initialized in the same way as with the previous preemption mechanisms. Upon hearing a high priority packet, the nodes start buffering the low priority packets. The window for transmitting the low priority traffic is determined based on the time of the last received high priority packet. Fig. 4 shows the receival times of the packets n and $n + 1$, and the low and high priority time spans surrounding them. Before the packets is a time span P_b, wherein the high priority packet is being received by the upstream node. No low priority traffic should be sent in this period. After the high priority packet is received by the current node, there is a time span P_a, wherein the high priority packet is forwarded by the downstream node(s). Here too, no low priority traffic should be sent. The P_a and P_b time spans are hereafter referred to as "guard windows".

After the guard window P_a is the low priority window W. During this time span, until the period P_b starts, the low priority traffic can be transmitted in the network. For each new received high priority packet, the expected time for the next high priority packet is estimated, and the low priority window is set accordingly. This continues until no new high priority packet has been received for an extended period (one second).

A premise for this solution to work is that the high priority packets are transmitted at a relatively constant interval, and that the interval between each high priority packet is so large that jitter does not cause packets to be received out-of-order

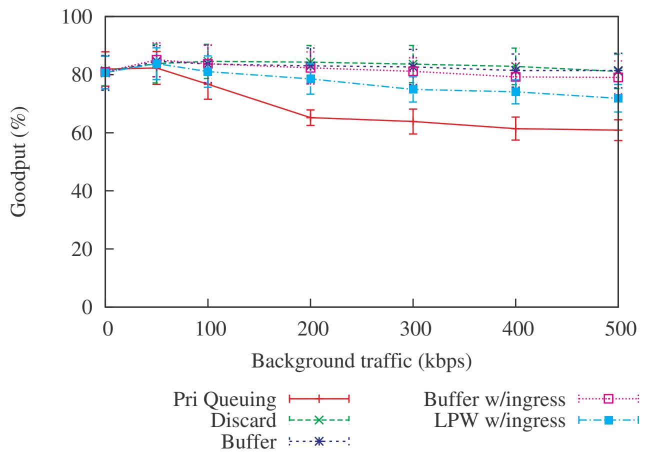

Fig. 5. Priority traffic goodput for the improved preemption mechanisms.

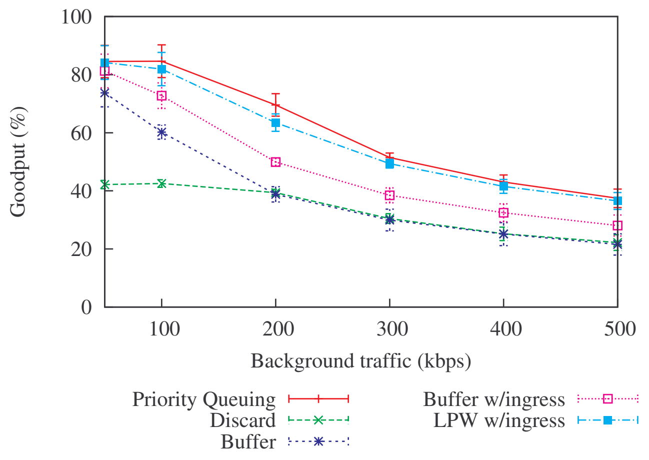

Fig. 6. Background traffic goodput for the improved preemption mechanisms.

at any destination. Packets received out-of-order would mean that the space between any two packets cannot be depended on for use by the low priority traffic, as a high priority packet may be transmitted in the neighborhood at any given time.

The LPW preemption mechanism faces several challenges in calculating the receival of the next high priority packet, thus identifying the start of the P_b time span: First and foremost, it is necessary to know the rate at which the packets are sent from the source. This can either be calculated from an average of the incoming packets, or it can be known through a predetermined codec selection and hard-coded before the network is started up.

Second, it is necessary to detect lost packets. If all the packets of the same flow have an incremental sequence number, it is easy to detect a missing packet. Another way of detecting lost packets is to compare the time of the incoming packets with the known interval between the packets, and see if there is a gap considerably larger than the expected gap between two consecutive packets.

A third challenge is to cope with the jitter between the packets. Jitter can be observed in terms of the variation in delay between any two consecutively received packets. The determination of when the LPW is in effect is done locally, based on the time of the received high priority packet. Thus, the dissemination delay is not a problem, although it leads to nodes operating with local LPWs different from each other. Both the delay and the jitter grow larger for each hop the packet is sent outwards from the sender. Therefore the mechanism is best suited for networks of limited size.

D. Evaluation of the improved preemption mechanisms

The priority traffic achieves a goodput without any background traffic at around 80% with the buffer mechanism (Fig. 5), and manages to maintain this goodput as the background traffic increases. 80% is the same performance as the discard mechanism. The difference is that the background traffic goodput is increased, compared to the discard mechanism (Fig. 6). As long as the background load is low,

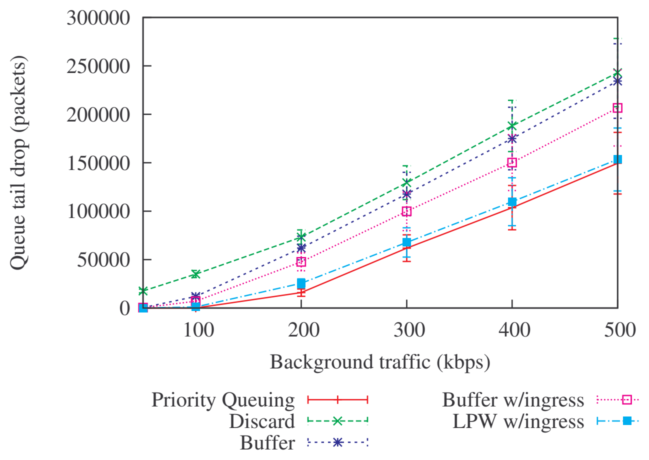

Fig. 7. Interface queue loss for the improved preemption mechanisms.

any low priority packets are buffered until the high priority session is over, and then transmitted. However, for the buffer mechanism more and more packets are lost due to tail drop as the background traffic load increases (Fig. 7). Thus, the low priority goodput is reduced, compared to the priority queuing results.

The results for the buffer mechanism with the ingress queuing (Fig. 6) show that using the ingress queuing increases the background traffic results by some 10%. With the buffer mechanism and ingress queuing, the number of tail drop losses (Fig. 7) is reduced compared to without the ingress queuing, since the packets already in the interface queue are assigned a more correct next hop.

The LPW preemption mechanism has been simulated with the ingress queuing enabled, since the ingress queuing clearly has a positive impact on the background traffic performance. The P_a and P_b guard windows enclosing the high priority packet transmission (Fig. 4) have for simplicity been assigned the same size in the simulations at 10 ms each, leaving 10 ms for the LPW transmissions (30 ms high priority packet interval). In reality, the P_a window, which protects the medium

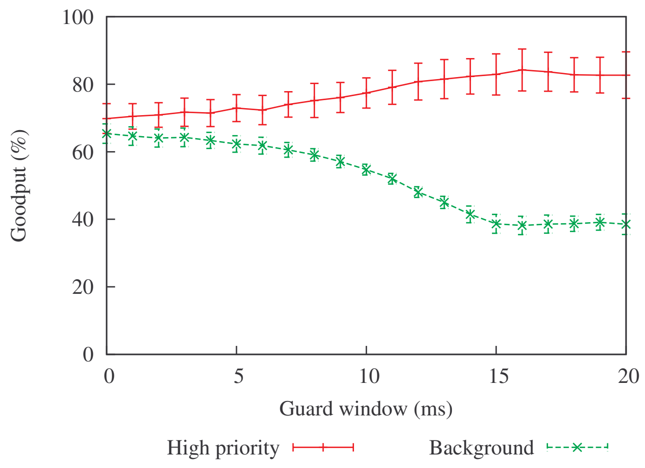

Fig. 8. Goodput with an increasing guard window.

after an actually received packet, is less, since the beginning of the window is determined directly by the receival of the packet. Thus, jitter can only affect the end of this guard window, while P_b, ensuring that the upstream high priority sender and forwarders can use the medium without any interfering low priority traffic, is more exposed to jitter, and should as such be larger than P_a. However, the difference is considered negligible in our study.

The LPW mechanism yields high priority traffic results (Fig. 5) better than using only the priority queuing, but worse than using the buffer or discard preemption mechanisms. It is clearly affected by the increasing background traffic, dropping around 8% when the background traffic is increased from zero to 500 kbps. At the same time, the background traffic goodput is increased compared to both the buffer and discard mechanisms.

However, it is observed that the jitter for the high priority traffic is kept very low (0.01 s), compared with using only the priority queuing (>0.1 s), making the buffer mechanism better suited to support PTT than by using only the priority queuing. Thus, the LPW preemption mechanism could be suited for networks where the PTT traffic can tolerate some loss to accommodate the lower priority traffic.

To better understand the behavior of the LPW mechanism, simulations have been run where the guard window on each side of the packet is increased by one ms at a time with 200 kbps background traffic load. These results (Fig. 8) show that there is no clear optimum where the guard window is perfectly matched to achieve the maximum goodput for both the high priority and the background traffic. The high priority traffic goodput reaches maximum with a 15 ms guard window. This is expected, as the two guard windows P_a and P_b at 15 ms leave no window W where low priority traffic can access the network (the interval between the high priority packets was 30 ms).

The results show that preemption with buffering and the ingress queuing can be used to maintain the same performance for the high priority PTT traffic, but avoid the negative impact on the background traffic caused by preemption with discard. If a somewhat lower goodput for the high priority traffic is accepted, however, LPW with the ingress queuing could be preferred instead. Then the background traffic will not suffer the losses that it does with the discard mechanism.

TABLE I
GOODPUT FOR PRIORITY TRAFFIC USING TCP FOR BACKGROUND TRAFFIC.

	Normal	Discard	Buffer[1]	LPW[1]
Goodput (%)	18.3	57.5	53.9	51.9

[1] with ingress queuing enabled.

VI. PREEMPTION AND TCP

The previous simulation results are based on UDP as the transport protocol for the background traffic. However, TCP is the protocol mostly used for packet transportation in the Internet. It is preferred due to its rate control, but in ad hoc networks this feature can be detrimental for the performance [6].

The difference between TCP and UDP is mainly rate control and packet acknowledgments. TCP automatically attempts to take as much as possible of the medium, while not causing congestion. UDP, on the other hand, has no rate control and blindly sends what is received from upper layers. With the setup used in the simulations (i.e. where all nodes send traffic to half of the other nodes in the network), the one hop flows (i.e. the flows going directly between two neighbors) will support the highest throughput. Thus, TCP will end up transmitting most packets via these flows.

Using the TCP protocol for the background traffic yields merely 18% goodput without any extra applied mechanisms (Table I). TCP keeps on pushing packets for the one hop flows at a high rate, corresponding to a very high constant bit rate system load, while reducing the load over multi-hop flows. This limits the propagation of the high priority packets initiating the preemption, resulting in a lower goodput for the discard mechanism. The initialization phase of the preemption is harder to accomplish when TCP is used for background traffic. It is observed that even as long as one second after the first high priority packet is propagated in the network, parts of the network are still transmitting TCP traffic. Due to a combination of hidden node and TCP's high rate transmissions over one hop, nodes that should forward the high priority traffic experience collisions, either when receiving the high priority traffic, or when forwarding it. Thus, although a particular node may have stopped transmitting the lower priority packets, other nodes in the neighborhood are unaware of this.

In fact, with TCP, the initialization problem gets worse when some nodes have received a high priority packet and subsequently stop transmitting the TCP background traffic. This event will make more of the medium available to other TCP flows, and these will quickly increase the load to use the additional capacity. As the high priority traffic spreads outwards from the source, the TCP flows that increase in load act as hidden nodes for the high priority traffic, increasing the probability of collisions and hence the delay in initializing preemption. This can be seen as a race condition, where the high priority traffic, initializing the preemption, competes with the rate control of TCP.

The even lower goodput for the buffer and LPW preemption mechanisms, compared to the discard mechanism, is a result of the always full queues at the start of the high priority sessions. These packets are to be pushed out, either after the high priority session is over, or during the low priority window between the high priority packets.

The results testing the preemption mechanisms with the background traffic carried by TCP were lower than with UDP, as the high priority flow at best, using the discard mechanism, achieved 20% less goodput, going from around 80% to 58% goodput. The two other mechanisms buffer the background packets, and this makes them more vulnerable to TCP's rate control. This was due to the vulnerable preemption initialization phase. However, this problem is closely related to the challenge of using TCP in ad hoc networks, and is not a problem only faced by these preemption mechanisms. Investigating the preemption mechanisms with the use of TCP, and TCP modified for ad hoc networks, is an interesting topic for further work.

VII. Related work

There is little work to be found on preemption in PTT ad hoc networks. Most solutions for Quality of Service (QoS) in ad hoc networks focus on QoS routing [7]. QoS for multicast has been studied, but here too most of the work has been focused on routing and the enhancement of multicast distribution [8].

Some QoS solutions implement call admission control, but only a very few, such as [9] consider preemption. Works on the preemption of traffic flows primarily focus on the preemption of the real time flows, such as [10].

In [11], Elmasry et al. propose a model managing QoS for Secure Tactical Wireless Ad Hoc Networks, directed towards the future US Army tactical backbone network. It is based on traffic characteristics measurements, calculating congestion severity levels and, based on this, generating admission and preemption policies.

For wireless sensor networks, several works on real-time scheduling have been published. A more recent is JiTS [12], a Just-in-Time scheduling protocol which works by delaying packets so they are received "just in time" at the destination node. The result is greater resilience against traffic bursts causing congestion.

VIII. Conclusions and future work

In this paper we have showed how employing priority mechanisms can improve multicast priority traffic conditions in ad hoc networks. The individual effects of the mechanisms were investigated. It was shown that priority queuing can stop the negative impact of background traffic on the priority traffic, but only through interface queue tail drops. To be able to maintain the same goodput as without any background traffic, it was necessary to use discard or buffering preemption of the background traffic, thus effectively removing the low priority traffic from the medium as the high priority traffic flow was active. The low priority window preemption was a compromise between the buffer preemption and only priority queuing, increasing the guard windows to enhance the high priority traffic performance at the expense of the background traffic.

The simulations using TCP for the background traffic showed that although the preemption with discard or buffering mechanisms were very effective with UDP, TCP poses different challenges. The analysis on the UDP background traffic presented in this paper is a good starting point for continued work on the preemption mechanisms and TCP background traffic. One solution to mend the initialization problem could be to transmit special initialization packets at very short intervals beginning the initialization phase. This way, the TCP algorithm would not be able to increase the rate as much as with only ordinary voice traffic with much larger intervals between the packets.

Initial studies on the QoS support of IEEE 802.11 (802.11e) MAC protocol [13] were performed. However, these showed that the performance of the high priority traffic was reduced compared to not using the QoS mechanism, due to the increased number of collisions stemming from the reduced contention window settings for the high priority traffic. Therefore, the 802.11e was not investigated further in this paper, but could be considered as future work.

References

[1] "Network simulator 2 - ns2." [Online]. Available: http://nsnam.isi.edu/nsnam/index.php/Main_Page

[2] "ns-2 code for random trip mobility model." [Online]. Available: http://monarch.cs.rice.edu/~santa/research/mobility/

[3] J. P. Macker, J. Dean, and W. Chao, "Simplified multicast forwarding in mobile ad hoc networks," *Military Communications Conference, 2004. MILCOM 2004. IEEE*, vol. 2, pp. 744–750 Vol. 2, 2004. [Online]. Available: http://dx.doi.org/10.1109/MILCOM.2004.1494892

[4] A. Hafslund, T. T. Hoang, and O. Kure, "Push-to-talk applications in mobile ad hoc networks," *Vehicular Technology Conference, 2005. VTC 2005-Spring. 2005 IEEE 61st*, vol. 4, pp. 2410–2414 Vol. 4, May-1 June 2005.

[5] L. Landmark, K. Øvsthus, and O. Kure, "Alternative packet forwarding for otherwise discarded packets," in *Future generation communication and networking (fgcn 2007)*, vol. 1, Dec. 2007, pp. 8–15.

[6] S. Xu and T. Saadawi, "Does the ieee 802.11 mac protocol work well in multihop wireless ad hoc networks?" *Communications Magazine, IEEE*, vol. 39, no. 6, pp. 130–137, Jun 2001.

[7] P. Mohapatra, J. Li, and C. Gui, "Qos in mobile a hoc networks," *Wireless Communications, IEEE*, vol. 10, no. 3, pp. 44–52, June 2003.

[8] A.-H. A. Hashim, M. M. Qabajeh, O. Khalifa, and L. Qabajeh, "Review of multicast qos routing protocols for mobile ad hoc networks," *International Journal of Computer Science and Network Security*, vol. 8, no. 12, pp. 108–117, December 2008.

[9] M. Canales, J. Gallego, A. Hernandez-Solana, and A. Valdovinos, "Cross-layer routing for qos provision in multiservice mobile ad hoc networks," *Personal, Indoor and Mobile Radio Communications, 2006 IEEE 17th International Symposium on*, pp. 1–5, Sept. 2006.

[10] G.-S. Ahn, A. Campbell, A. Veres, and L.-H. Sun, "Swan: service differentiation in stateless wireless ad hoc networks," *INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, vol. 2, pp. 457–466 vol.2, 2002.

[11] G. Elmasry, C. McCann, and R. Welsh, “Partitioning qos management for secure tactical wireless ad hoc networks,” *Communications Magazine, IEEE*, vol. 43, no. 11, pp. 116–123, Nov. 2005.

[12] K. Liu, N. Abu-Ghazaleh, and K.-D. Kang, “Jits: just-in-time scheduling for real-time sensor data dissemination,” in *Pervasive Computing and Communications, 2006. PerCom 2006. Fourth Annual IEEE International Conference on*, March 2006, pp. 41–46.

[13] IEEE, “Wireless LAN medium access control (MAC) and physical layer (PHY) specification,” IEEE standard 802.11-2007, June 2007.