A Mobility-Adaptive TDMA MAC for Real-time Data in Wireless Networks

Johannes Lessmann and Dirk Held

University of Paderborn, 33102 Paderborn, Germany
{lessmann, madmax}@upb.de

Abstract. In this paper, we present a TDMA MAC protocol for scenarios with real-time traffic like voice streams. While many previous works assume stationary networks, DynaMAC can quickly adapt to changing topologies while not violating delay guarantees for even a single data packet with high probability. By intelligent data aggregation, DynaMAC also accounts for the fact that the large synchronization preambles of high-speed PHYs make transmissions of very small packets like voice samples extremely inefficient. Further, it introduces a novel segmentation concept, which results in a discrete set of potential per-node delay guarantees, which can be exploited by higher layers to balance the end-to-end delay of routes with different lengths. Finally, the slot allocation strategy tries to ensure a contiguous placement of unassigned slots, so that larger best-effort packets can be efficiently transmitted with a minimum of fragmentation. We validate our propositions with expressive simulations.

1 Introduction

In cases of QoS sensitive traffic, an alternative to contention-based MAC protocols are TDMA MACs. The challenge with TDMA is to create an access schedule for the wireless medium. If the assigned slots are non-overlapping within the two-hop neighborhood of each node, all packets can be sent without risk of collisions. TDMA MACs also naturally lend themselves to periodic real-time traffic like voice or video streams for two reasons. First, they can maintain an established quality level (delay, bandwidth) even if the network load increases. Second, the overhead associated with allocating slots before actually transmitting is affordable since periodic traffic will make use of the same slots for a comparatively long time therefore making up for the initial cost.

While many existing TDMA protocols are designed for sensor networks and assume that the nodes do not move (much), our proposed protocol, DynaMAC, can efficiently handle changing topologies. With high probability, it can not only recover from mobility-induced packet collisions without dropping a single packet, but even uphold the delay guarantees for collided packets.

Another issue, that we explicitly address is the fact that the PHY preambles can be very large compared to individual real-time data packets. A modern voice codec like G.729 [1], for example, produces 40 voice packets per second with

approximately 200 bit per sample. When transmitted with a high-speed PHY like 802.11g at full speed (54 Mbps), this implies a packet length of approximately 4 μs. Compared to the PHY overhead of 26 μs per packet, this results in a goodput reduction of about 86%. With 802.11b, which has a preamble of 192 μs, the overhead at 11 Mbps would be 98%. We therefore propose to aggregate real-time samples first on a per-sender, then on a per-receiver basis to utilize the channel more efficiently.

We finally introduce a novel segmentation concept that allows DynaMAC to offer higher layers (most importantly the routing layer) not only one, but multiple per-node delay guarantees from which they can choose. That way, QoS routes which require many hops on the routing level, can make use of very small per-node delays while short routes are free to choose a more relaxed delay offer. This greatly helps to achieve common end-to-end delays even among routes of different lengths. Traditionally, end-to-end delays are somewhat proportional to the hop count. With DynaMAC, packets on routes with many hops can be routed "faster". Common end-to-end delays are particularly desirable for voice conversations where values of more than 250 ms are not tolerable no matter how many hops the required route might have.

The remainder of this paper is organized as follows. Section 2 discusses previous TDMA MAC protocols and their shortcomings. In Section 3, DynaMAC is described in detail. Section 4 shows DynaMAC's performance using simulation results. We conclude in Section 5.

2 Related Work

Many works have been published in the domain of QoS MACs. Here, we confine ourselves to TDMA approaches. In USAP [2], the author presents a slot allocation scheme where time is divided into frames of variable length. The length of a frame depends on the node density in the neighborhood. When there are many neighbors, the frame length must be doubled to create new transmission opportunities. Unfortunately, [2] does not specify how exactly slots are to be allocated and is also vague as to the details of frame doubling. In [3], the authors try to improve on USAP. Time is again divided into frames where the first slot is always unoccupied. When a new node needs a slot and no unassigned slot can be found, it eventually doubles its frame length (number of slots per frame), copies the schedule into what is now the second half and claims the first slot in the second half. Mobility-induced conflicts are also resolved by doubling the frame length. One conflicting node takes the original slot, the other one claims as described above. Effectively, this approach halves the transmission opportunities for the conflicting slot. This makes it unusable for real-time streams with fixed periodic traffic which we consider in this paper.

In LMAC [4], each node owns exactly one slot in every frame. In the control part of its messages, a node announces the intended recipient for the current message. All nodes must listen to the control part of their neighbors to know whether they are the recipients or not. This approach does not make sense for

small slot sizes like the ones appropriate for real-time samples, since that would imply constant idle listening of all nodes even when they are rarely involved in any actual communication.

DRAND [5] has a schedule allocation strategy which is somewhat similar to ours (allocation by negotiation, schedule propagation). However, DRAND assumes that this schedule has to be constructed only once and for all, it does not provide any method to change or repair schedules. It is explicitly targeted at stationary networks and not applicable to changing mobility. Besides, DRAND does not specify how messages for the initial schedule construction must be exchanged (contention-based or according to some preliminary schedule). Therefore, it is hard to compare its performance with that of DynaMAC, since the initial construction might take any amount of time.

None of the previous TDMA protocols can efficiently meet the requirements imposed by periodic real-time traffic with very small samples in a dynamic network of constantly moving nodes. Additionally, our proposed approach of frame segmentation which allows us to uphold delay guarantees even in the face of mobility-induced collisions is completely novel.

3 Architecture of DynaMAC

In DynaMAC, time is divided into *cycles* (frames). The length of a cycle depends on the rate of the (CBR) real-time traffic that is to be supported. A voice codec like G.729 produces 40 packets per second, i.e. one packet every 25 ms. To support this kind of traffic, the cycle length should therefore be 25 ms. If a mixture of real-time traffic types with different rates is to be routed, the least common multiple must be selected for the cycle length. A cycle is divided into a number l_{cyc} of *slots*.

On a higher abstraction level, a cycle consists of a *random phase* and a *scheduled phase*. The slot count l_{rnd} of the random phase is small compared to that of the scheduled phase. The random phase is accessed using CSMA without exponential backoffs and used to exchange control data like topology information, slot allocations or, to some extent, best-effort traffic. The scheduled phase is accessed according to the TDMA schedule and accomodates the real-time traffic and all best-effort traffic which could not be already transmitted in the random phase. It is divided into *sc segments* of equal size, i.e. each segment has $l_{seg} = \frac{l_{cyc}-l_{rnd}}{sc}$ slots. At the end of each segment are l_{buf} dedicated buffer slots which are used for collision resolution as will be described later. Finally, to allow new nodes to easily synchronize with their neighbors, the random phase is started with one *synch slot*. Each node u sends a synch beacon in this slot with probability $\rho_u = \frac{1}{n_u+1}$ where n_u is the number of neighbors of u. Figure 1 depicts the structure of a DynaMAC cycle.

3.1 Timing Parameters for the Medium Access

Random Phase For the whole random phase, all nodes must switch to listening mode. The random phase is accessed using CSMA without backoff. A node that

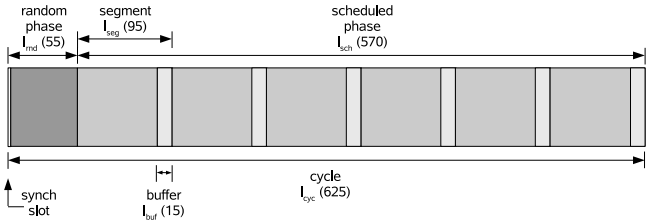

Fig. 1. Structure of a DynaMAC cycle (frame). The numbers in brackets are sample values which are assumed for explanation purposes in this paper.

wants to transmit computes a random slot number from the random phase and sends. In case of collisions, packets must be resent. To detect collisions, nodes which could not successfully receive a message in the random phase have to broadcast a *NACK* message a SIFS after the message. SIFS means "Short Inter-Frame Space" and is adopted from the 802.11 standard. In addition to *NACK*s, unicast messages are acknowledged by the recipients with an *ACK*.

Scheduled Phase In the scheduled phase, nodes sleep whenever they are not scheduled from transmission or receiving. Since a particular node u may be scheduled to send in slot s and receive in slot $s+1$ or vice versa, a slot must not be occupied completely, but a SIFS must be left free at the end of each slot to allow the node's radio to switch the radio mode. When mobility-induced collisions occur (cf. Section 3.5), the receiver v must be able to send a *NACK* to the sender u so that the situation can be resolved. Instead of sending a *NACK* as a direct reply (which would consume a lot of time in every slot), the *NACK* is piggybacked in the next message which is sent by v anyway. Since u knows the schedules of all of its one-hop neighbors, it can correctly identify the slot where to expect a potential *NACK* message and switch to listening mode accordingly. Since space must be reserved for the *NACK*s statically (regardless of whether they are actually sent or not), we also piggyback *ACK*s for confirmation of successful receptions. That way, a successful transmission can be distinguished from the case where the recipient has moved out of the sender's transmission range.

3.2 Data Aggregation

As already indicated in Section 1, real-time stream samples can be very small compared to the synchronization preamble of the 802.11 PHY (which we continuously consider in this paper). Hence, transmitting only one sample per packet as a general rule is unacceptable. To our knowledge, we are the first to consider full data aggregation in the context of multi-hop TDMA MACs with delay-sensitive data.

In DynaMAC, we try to combine all samples for which a node u is the sender into one packet. If a packet becomes too large for one slot, u tries to

allocate a neighboring slot so that fragmentation can be avoided. This means that a packet will generally have multiple receivers, namely all nodes, for which a sample is included in the aggregated packet. Hence, all receivers of a packet must listen for u at the particular slots where u transmits. To be able to do so, u must notify its recipients to listen before transmitting by a message in the random phase. In Section 3.3, we will introduce the segmentation concept of DynaMAC. A DynaMAC cycle is divided into several segments to provide different delay bounds. There, we will see that our data aggregation is actually done independently for each segment. Hence, packets contain only those samples for which a node u is the sender and which are scheduled for the same segment.

3.3 Per-Node-Delay Guarantees

In a TDMA MAC, the per-node delay is given by the time difference between the slot when a sample is received by a node u and the slot when it is forwarded by u to the successing hop. Consequently, in a TDMA MAC, where slots are allocated conflict-free, it is principally possible to require (and guarantee) per-node delays in a very fine-grained manner. However, our data aggregation strategy makes this kind of fine-grained approach very counterproductive, since aggregation relies on a certain amount of streams that can be combined. Consequently, since data aggregation is an absolutely essential part of DynaMAC to improve the goodput (cf. Section 3.2), we propose segmentation as a compromise to remedy the problems discussed above. We divide a DynaMAC cycle into a set of sc distinct segments. Guarantees are only given in terms of segments. Within segments, data aggregation can be performed. When we guarantee delivery only for the end of the segment, we do not need to make any statement as to the specific slot in the segment which we chose for transmission. The segmentation constitutes a good combination of the advantages of data aggregation, on the one hand, and fine-grained delay guarantees, on the other hand.

3.4 Slot Allocation

Generally, the goal of the slot allocation strategy is to construct and maintain a conflict-free schedule. We start by explaining the smallest case of just two nodes. Eventually, one of them (say u) will hear the synch beacon of the other one (say v) and synchronizes its cycle to that of its neighbor. Since u does not know whether v is also new to the network or not, it sends a *Schedule Request* (SR) packet to v. Since v is unaware of any other node, it replys with a *Schedule Update* (SU) packet without any allocated slots. Both nodes can then independently allocate their initial slots. The node with the higher ID chooses the first two slots in the first segment, the other one takes the first two slots in the second segment. We call the initial double-slot of a node its *default slot*. Note that the default slot is always a sending double-slot, because it is used to reliably broadcast some control data like SU packets (see below) or $NACK$s.

If additional nodes join the network, they will eventually receive a non-collided synch beacon from one of the existing nodes which allows them to

synchronize with their neighbors. They can then broadcast an *SR* packet in the random phase. Since now a schedule is already in place, all nodes which hear the *SR* packet reply with an *SU* packet that contains a bitmask in which each slot of the scheduled phase is represented by one or two bit. The first bit indicates whether the corresponding slot is allocated or not. If it is allocated, there is a second bit for that slot which indicates whether this slot is claimed by the sender of the *SU* packet itself or by one of its one-hop neighbors. The *SU* packets are sent in the default slots of each node. This is done to relieve the random phase from as much burden as possible. Since new nodes are always in listening mode until they have established their own default slots, the packet will certainly be received by the new node.

When the new node receives the *SU* packets from its neighbors, it computes their disjunctive combination to determine the set of unassigned slots in its two-hop neighborhood. Then, it picks the first free slot and broadcasts a *Slot Allocation* (*SA*) message with an "unapproved" flag in the next random phase indicating its slot selection. When no disapproval is sent by any neighbor, the new node sends the same *SA* message again, this time with an "approved" flag. The neighboring nodes receive the second *SA* message and in turn propagate it to their one-hop neighbors in their default slots in the next cycle, so that eventually, all two-hop neighbors are aware of the new slot allocation and can update their internal schedules. The same procedure applies for established nodes to get additional slots.

When two one-hop neighbors want to allocate the same slot in a random phase, one node will be earlier, and, since all nodes are listening, the other node can hear the conflicting allocation and claim another slot. If two two-hop neighbors u and v claim the same slot, there will always be some node w which is a one-hop neighbor to both u and v and therefore hears both nodes' *SA* messages. Hence, w sends an *SU* message to the node with the lower node ID, say u, which includes the new slots allocated by v. Node u interprets this as a disapproval and must send a new *SA* message (unapproved) with different slot allocations. This procedure is repeated until all allocation request are satisfied. When a node wants to release a slot that it previously allocated, the same procedure as described in this section can be utlized.

3.5 Mobility-Induced Collisions

When nodes are moving around, it is possible that two nodes which were more than two hops apart when the current schedule was negotiated, approach each other and therefore become part of each other's two-hop neighborhood. When these nodes have conflicting slot allocations (which was not a problem previously), there can be collisions. We resolve this problem as follows. When packets of two nodes u and v collide in slot s, one or both recipients will send a *NACK* in their next sending slot (i.e. collision-free). Then, one or both of u and v immediately choose a buffer slot located at the end of the current segment and retransmit the packets. If a collision occurs in the buffer slot again (other nodes could also have chosen the same buffer slot due to independent collisions), a

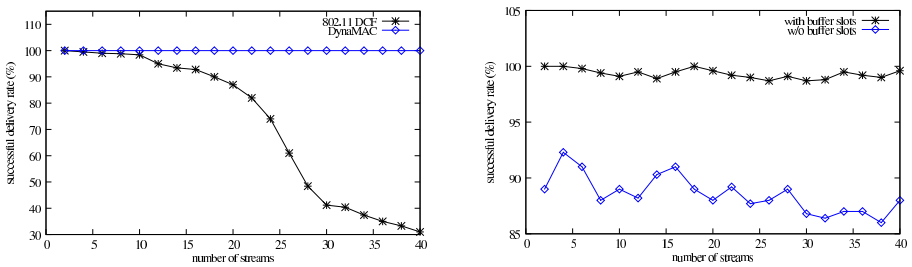

Fig. 2. (a) Successful delivery rate of 802.11 DCF and DynaMAC under increasing load. (b) Successful delivery rate of DynaMAC with and without buffer slots under increasing load.

new buffer slot is computed using a linear congruential generator with the triple (sender ID, current slot number, original slot number) as its input. Eventually, all conflicting nodes will have sequentialized. Since buffer slots are only used for collision resolution, chances are very high that any (even the first) retransmission in a buffer slot is successful. Since in DynaMAC, per-node delay guarantees are only given in terms of segment ends (cf. Section 3.3), a retransmission in a buffer slot at the end of the original segment does not violate the delay guarantee. This is another advantage of our proposed segmentation concept. Generally, the larger the delay between incoming and outgoing slot for a packet, the more buffer slots are in between and the more likely is it that a retransmission *within the guaranteed delay* will succeed.

4 Simulation Results

We simulated DynaMAC using ShoX [7]. For the simulation, we chose the DynaMAC parameters which we also assumed throughout this paper. The simulated network consisted of 100 nodes randomly distributed in an area of 50 × 50 m. The length of a stream is chosen randomly with a maximum of 15 seconds. For signal propagation, we used the unit disk model with a radius of 20 m. To assess the quality of DynaMAC, we compared it to the 802.11g DCF (Distributed Coordination Function) MAC in ad hoc mode. Our desired end-to-end delay is 100 ms.

Figure 2(a) shows the delivery rate of the 802.11 DCF MAC and DynaMAC under different network loads in a static network. The network load is given in terms of the number of concurrent streams in the network. As can be expected, the 802.11 DCF MAC performs poorly with timely delivery when the load increases. When the number of streams exceeds 30, less than half of the packets are delivered on time. This is due to the fact that, since there is no data aggregation in 802.11, a number of 40 streams implies that there are actually 1600 sample packets per second which compete for the medium. DynaMAC, on the other hand, delivers 100% of the packets regardless of the current load. This

is because of the fact, that the routing layer reserved the complete path between source and destination before sending any sample packets. Collisions due to mobile nodes are covered below.

We measured the impact of our buffer slot concept in order to evaluate and demonstrate its benefits. For that, we induced collisions artificially by having the nodes broadcast interfering sample streams in arbitrary slots. This strategy allows to induce collisions without the need to move nodes, i.e. without routing layer issues. Figure 2(b) clearly shows the benefit of our buffer slot strategy. If no buffer slots are used, all collided packets get lost as long as an interfering "collision stream" lasts. With buffer slots, however, the delivery ratio almost remains at 100 %. The few exceptions are due to situations when multiple collisions of independent streams occur at the same time and different nodes try to switch to the same buffer slots.

5 Conclusion

In this paper, we have presented DynaMAC, a novel TDMA MAC protocol for delay-sensitive data. It is one of the few TDMA MACs which consider mobility of nodes. DynaMAC even allows to uphold packet delay guarantees in the face of constantly moving nodes. DynaMAC also greatly increases the network goodput when it comes to small real-time samples. We have proposed a novel segmentation concept as an excellent compromise between fine-grained delay guarantees and data aggregation. Our proposition to provide buffer slots at the segments' ends and give delay guarantees only in terms of segment ends achieves useful flexibility for DynaMAC to shift outgoing slots back and forth within the whole delay guarantee period, thus enabling optimal inter-segment aggregation.

References

1. International Telecommunication Union, *Recommendation G.729*, `http://www.itu.int/rec/T-REC-G.729/en`
2. D. Young, *USAP Multiple Access: Dynamic Resource Allocation for Mobile Multihop Multichannel Wireless Networking*, IEEE Military Communications Conference Proceedings, 1999
3. A. Kanzaki, T. Hara, and S. Nishio, *An Efcient TDMA Slot Assignment Protocol in Mobile Ad Hoc Networks*, Proceedings of the 2007 ACM symposium on Applied computing (SAC), 2007
4. L. van Hoesel and P. Havinga, *A Lightweight Medium Access Protocol (LMAC) for Wireless Sensor Networks*, First International Workshop on Networked Sensing Systems (INSS), 2004
5. I. Rhee, A. Warrier, J. Min, L. Xu, *DRAND: Distributed Randomized TDMA Scheduling For Wireless Ad-hoc Networks*, ACM MobiHoc 2006
6. V. Rajendran, J. Garcia-Luna-Aceves and K. Obraczka *Energy-Efficient, Application-Aware Medium Access for Sensor Networks*, 2nd IEEE Conf. on Mobile Ad-hoc and Sensor Systems (MASS), 2005
7. The ShoX developers, *ShoX - scalable ad hoc network simulator*, `http://shox.sourceforge.net`