Jittering Broadcast Transmissions in MANETs: Quantification and Implementation Strategies

Roy Friedman
Dept. of Computer Science
Technion–Israel Inst. of Technology
roy@cs.technion.ac.il

David Hay
Dept. of Electrical Engineering
Politecnico di Torino, Italy
hay@tlc.polito.it

Gabriel Kliot[1]
Microsoft Research
Redmond, WA, USA
Gabriel.Kliot@microsoft.com

***Abstract*—Delaying the transmission time of messages for a short random period, a.k.a. *jittering*, is a known approach for preventing concurrent transmissions, and consequently collisions, in multiple access networks (e.g., RFC 5148). It is particularly useful for increasing the reliability of unacknowledged broadcast messages in the 802.11 protocol. Yet, transmission jittering comes with a price. It increases the average end-to-end latency and could potentially decrease the throughput of the network.**

This paper investigates the relation between the maximal jitter duration and the probability of successful transmissions. Specifically, we propose and compare three implementation strategies: inside the MAC protocol, in the IP layer, and a cross-layer implementation. The comparison establishes that our practical cross-layer implementation minimizes the suggested protocol changes to 802.11 MAC protocol, while incurring the smallest latency, thus being the most attractive one. In fact, under IEEE 802.11e extension, our implementation does not require any MAC change.

Our results show that by carefully designing the system, a very small jitter value of hundreds of microseconds suffices to obtain a high transmission success probability, a network utilization factor of above 70%, and subsequently to ensure high flooding reliability. This is in contrast with prior works, which have suggested to use two orders of magnitude larger jitter (i.e., tens of milliseconds).

All our results are backed up by extensive simulations.

I. Introduction

In a wireless network, simultaneous packet transmissions by nearby nodes are undesirable, as they may cause collisions which result in packet loss. Such collisions are especially detrimental for 802.11 broadcast messages. This is since the 802.11 protocol cannot sense collisions, does not send acknowledgment messages (acks), and does not employ the RTS/CTS collision avoidance mechanism for broadcast messages. Therefore, a node that sends a broadcast message has no way of knowing if it was successfully received by others.

Collisions of broadcast messages are highly likely in protocols that send periodic messages as well as for externally event-triggered messages and message forwarding (flooding), as described in RFC 5148 [1]. Specific examples of periodic messages include HELLO messages sent by MANET routing protocols. Event-triggered messages occur when nodes respond to changes in their environment with an immediate message transmission. For example, in reactive MANET routing protocols such as AODV [2], a node sends a RERR message upon notification of a link breakage with one of its neighbors to all nodes that used to route through this link. Message forwarding may occur in reactive MANET routing protocols such as AODV, which broadcasts a RREQ message to find a new route, proactive MANET routing protocols such as OLSR [3], which broadcasts Topology Control messages, as well as in any other flooding based dissemination protocol.

[1]The work was done when the author was with the Department of Computer Science, The Technion – Israel Institute of Technology.

RFC 5148 [1] recommends to jitter (that is, to randomly delay) broadcast transmissions in mobile ad hoc networks in order to reduce the probability of such collisions, *yet without an exact guidance on how to set this jitter*. In this paper, we investigate the influence of a maximum jitter value (*maxjitter*) on the probability of a collision free transmission in various channel and network models. In contrast with [1], which states that *maxjitter* should be significantly greater than (an order of magnitude) any medium access control frame period, we show that by applying a cross-layer design, the benefits of jitter can be obtained by *relating* maxjitter *to the MAC slot period*, which is significantly shorter. [1] also states that *maxjitter* may depend on nodes density. In our work, we quantify this dependence.

Our Contributions: In this paper, we first consider a theoretical, simplified implementation of the jitter mechanism. We provide a mathematical analysis of the probability of a collision free transmission, in a simplified protocol model, in a single hop ad hoc network (when all nodes are within transmission range of each other). We tight the collision probability with the number of competing transmissions and the number of slots available for these transmissions. This formal analysis, which is performed in a simple to understand model and manner, serves as motivation for the rest of the work, which is more applied.

Second, we discuss how the jitter mechanism can be implemented in the networking stack. Specifically, one possible way to implement this mechanism is at the application or IP layer (or generally speaking, above the MAC layer). However, as we precisely show in this paper, such an implementation results in a very large latency. This is since an implementation that does not rely on low level carrier sensing capabilities of the MAC protocol must relate the *maxjitter* value to the message duration. As a result, such an implementation yields an average jitter of milliseconds. Interestingly, our analysis can actually be used to explain the value of 10 milliseconds proposed in [4] when the jitter is implemented in the application layer, which was previously only determined empirically. Note that the rationale for implementing jitter at the IP layer and not above it is to eliminate the queuing effect that could otherwise intensify synchronization and collisions.

An alternative implementation is inside the 802.11 MAC protocol. Such an implementation is able to achieve the same effect of the jitter mechanism while using a significantly shorter *maxjitter* value of hundreds of microseconds. However, such an implementation is based on changing the MAC protocol. Since the protocol is standardized and its current implementations are ubiquitous, changing its code is impractical.

Therefore, our explicit goal is to implement the jitter mechanism with small *maxjitter* value, while minimizing the changes to the MAC protocol. We show how to do this while requiring to change only one MAC configuration parameter (the length of the contention window) and leaving the protocol code and logic intact. It is important to notice that changing the contention window value is allowed in the *Enhanced Distributed Channel Access* (EDCA) extension of IEEE 802.11e [5], which aims at implementing quality of service (QoS) differentiation in wireless networks. A policy of how exactly to adapt the contention window is a subject of recent studies [6], [7]. Thus, our IP-MAC cross-layer implementation can be fully realized in the current standard limitations, without any MAC changes.

The resulting final implementation achieves tunable high success ratios with extremely short *maxjitter* values (hundreds of micro seconds). This is in contrast with previous works, which have suggested to use much larger *maxjitter* values (e.g., on the order of 10 milliseconds in [4]). Thus, due to our analysis, and by applying a cross-layer design, we are able to reduce the latency of transmissions that use jitter by one or two orders of magnitude compared to previous approaches, without increasing the collision probability.

We also discuss the impact of the *maxjitter* value on multiple hop networks. In particular, we investigate the reliability of flooding (fraction of the network that receive the flooded message) as a function of *maxjitter*. We show by simulations that the small jitter values we computed above are enough to ensure near 100% reliability of flooding.

Our work includes an extensive simulation study, based on a more realistic physical (SINR based) model of a successful reception and uses a full implementation of the 802.11 MAC DCF protocol. We do so for different loads, message sizes, and different network models: single hop networks with and without hidden nodes and for multiple hop flooding. Our simulations validate our theoretical analysis and demonstrate the feasibility of using short *maxjitter* to provide very high channel utilization.

II. Related Work

Using jitter to increase the reliability of broadcast messages in MANET routing protocols was first suggested in [4], which proposed to use a jitter period of 10 millisecond. A jitter mechanism for MANET control messages was formally proposed by [1] to reduce the probability of transmission collisions. Alas, neither of these works provided design criteria on how to set the *maxjitter*, how its value influences the probability of collisions, and how this mechanism should be implemented.

The performance analysis of CSMA protocols in general and 802.11 DCF procedure in particular was subject to an extensive study in various models. The seminal work of Bianchi [8] provides an analysis of unicast transmissions in 802.11 DCF in a single hop network, without hidden nodes. The works in [9] extended this model with broadcast transmissions.

The throughput and transmission success probability in MANETs, which includes hidden node phenomena, was investigated in the context of CSMA protocols and radio networks (e.g. [10], [11]). For example, Wu and Varshey [10] consider jittering in which the random delay is chosen using a *geometric distribution*. However, in our setting, where the objective is to increase the expected number of successful transmissions, jittering through a geometric distribution gives worse results than applying a uniform distribution (with the same expectation). This is because the more uniform the wait period is, the lower the chances are that two nodes will decide to transmit at the same time. Evidently, RFC 5148 calls for using uniform distribution. Hence, the results of [10] can only serve as a lower bound on the real performance. Note that Tay et al. [12] investigated what is the distribution that achieves a *single* successful transmission with the highest probability and their optimal distribution turned out to be non-uniform. However, these results do not apply to our setting where we are interested in the distribution that attains the maximum average number of successful transmissions. Recently, Kanizo et al. [13, Theorem 3.1] showed in the context of multiple-choice hash-tables that the uniform distribution is optimal, where the objective is to decrease the number of overflowed items from the hash table; their result immediately applies also to our setting, with the assumption that time is slotted.

A new CSMA protocol for synchronized transmissions is proposed in [14]. Our explicit goal is not to introduce a new MAC protocol, but to deal with synchronized transmissions efficiently without any change to an already existing MAC.

The problem of flooding reliability in multihop networks was studied in [15]–[17]. We refer to these works in section VI. Flooding was modeled in [18], [19].

III. System Model

Consider a wireless network comprising of a finite set of nodes S, located arbitrary in the plane. A node is a device with an omni-directional antenna that enables wireless communication. Let X_i for $i = 1, 2, \ldots, |S|$ denote the location of the nodes. We also use X_i to refer to the i^{th} node itself. Denote by $|X_i - X_j|$ the Euclidean distance between X_i and X_j.

We assume that nodes have no geographic knowledge. Yet, each node knows the temporal number of its direct neighbors (nodes that it can exchange messages directly). Such a number can be constructed, e.g., by a simple heartbeat mechanism that is present in any case in many routing algorithms in ad hoc networks (see Section V-A for a detailed discussion). Nodes may join and leave the network at any time.

A. Communication model

We consider two models of a successful transmission reception over one hop, commonly used in the literature [20], [21].

The Protocol model: In this model, each node X_i is associated with a specific transmission range R_i. A transmission from X_i is received successfully by X_j if and only if $|X_i - X_j| \leq R_i$ and for every other simultaneously transmitting node X_k, $|X_k - X_j| \geq (1+\triangle)R_k$. $\triangle > 0$ is the *interference parameter*. To simplify the analysis, we assume that all transmission ranges are the same; namely, for each $i \in n, R_i = R$.

The Physical model: Let $\mathcal{T} \subseteq \mathcal{S}$ be the set of nodes simultaneously transmitting at some time instant, and denote by P_k the transmit power of node $X_k \in \mathcal{T}$. The transmission from node X_i is received successfully by X_j if and only if

$$\frac{\frac{P_i}{|X_i - X_j|^\alpha}}{N_0 + \sum_{X_k \in \mathcal{T}, \mathcal{X}_\| \neq \mathcal{X}_\rangle} \frac{P_k}{|X_k - X_j|^\alpha}} \geq \beta$$

In this model, a minimum signal-to-interference-plus-noise ratio (SINR) of β is necessary for successful reception. The ambient noise power level is denoted N_0 and the signal decays deterministically with distance r as $1/r^\alpha$. As typical to this model, we assume that $\alpha = 2$. In our simulations, we use homogeneous transmit powers: for each $i \in n, P_i = P$.

While we carry our analysis in the protocol model, all our results are validated by a simulation study, performed in a more realistic SINR model of accumulative interference (Sec.VII).

B. 802.11 MAC DCF procedure

In this paper, we target *IEEE 802.11 Distributed Coordination Function (DCF) MAC* for broadcast messages. We now briefly summarize the DCF procedure, as standardized by the IEEE 802.11 protocol. For a detailed presentation refer to [22].

PHY	Slot Time	CW_{min}	CW_{max}	Preamble + PLCP header	Min. packet length *
DSSS	20 μs	31	1023	Long - 192 μs Short - 96 μs	408 μs
FHSS	20 μs	15	1023	128 μs	420 μs
OFDM	9 μs	15	1023	20 μs	332 μs

* with short PHY preamble, PLCP header, MAC header, CRC, IP header and AODV RREQ msg sent at 2Mbps

Fig. 1. 802.11 PHY Characteristics

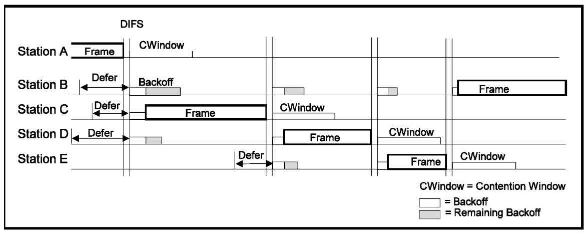


Fig. 2. 802.11 DCF backoff procedure

Before transmitting, a node determines whether the medium is busy or idle. If the channel remains idle for a *distributed inter-frame space* (DIFS) period, transmission can begin. If the channel was initially busy or changed from idle to busy during DIFS, the node defers until the channel turns idle and remains idle for the DIFS period. Then, a node generates a random *backoff* interval before transmitting, to minimize the probability of collision with packets transmitted by other nodes (namely, collision avoidance). If during backoff counting the channel turns busy again, the counting is frozen and is resumed after the channel turns idle again (and remains idle for DIFS). In addition, to avoid channel capture, a node waits an additional random backoff between two consecutive transmissions, even if the medium is sensed idle during the corresponding DIFS[1].

For efficiency reasons, DCF employs a discrete-time backoff. The time immediately following an idle DIFS is slotted, and a node is allowed to transmit only at the beginning of a slot. The MAC slot time size, referred to as *MAC_slot*, is equal to the time needed at any node to detect a transmission from any other node (within its range). This depends on the physical layer, and accounts for the air propagation, for the time required to switch from the receiving to the transmitting state, and for the time the receiver needs to assess the medium and signal to the MAC. The *MAC_slot* is a vulnerable period, during which an ongoing transmission may be corrupted by other nodes, because their MAC layers can only recognize that transmission with some delay. Fig. 1 lists various PHY types used by IEEE 802.11.

The backoff timer used for broadcast messages is picked from the range $[0, CW_{\min}]$, where $CW_{\min}$ is the minimum contention window. There is no exponential increase of backoff windows and no acknowledgments are used to detect successful reception of broadcast messages. Thus, a transmitting node has no way to indicate successful reception or collision. Also no collision avoidance by RTS/CTS mechanism is used. Due to these reasons, broadcast messages are usually sent at the lowest possible rate of 1 or 2 Mbps, as low rates enable better capture.

Fig. 2 illustrates the DCF backoff procedure. It is important to notice that: 1) no backoff is performed if the channel is sensed idle for DIFS period upon new transmission; 2) a backoff procedure is performed after the end of every transmission, even if no additional transmissions are currently queued at this node. We call this additional backoff a *post-backoff*.

[1]If the two transmissions are fragments of the same (large) packet, then MAC does not employ this backoff mechanism and waits for a shorter period between the fragments, called SIFS. We ignore this case in our analysis.

IV. Theoretical Jitter Analysis in Single Hop Networks

We now show how the maximum jitter value (*maxjitter*) influences the collision probability of simultaneous transmissions and how it should be set to maximize the network throughput in single hop networks. We start by presenting a simplified algorithm for jitter implementation, called `SimpleJitterSend`, which receives three parameters: the message to send, the number of simultaneous transmissions n and the desired probability p for each transmission to succeed. Hence, $p{\cdot}n$ is the average number of desired successful transmissions. Notice that in the protocol model, in a single hop network (without hidden nodes), if a particular transmission is successfully received by one node, then it is successfully received by all nodes. We assume all messages have the same length, *msg_length*. Before sending a message, `SimpleJitterSend` waits a random number of slots, picked based on n and p, as listed in Fig. 3.

```
Upon SimpleJitterSend(msg, n, p) do
(01)  $m := \lceil \frac{-2n}{\ln p} \rceil$;
(02)  wait (random (0, $m$) · msg_length);
(03)  lowerLayer.Send (msg);
```

Fig. 3. `SimpleJitterSend` algorithm

Analysis: Somewhat surprisingly, the analysis of `SimpleJitterSend` turns to be similar to the regular Slotted-Unslotted Aloha model [23]. This is not intuitive at the first glance, since `SimpleJitterSend` works in a different model than Aloha in CSMA/CD networks: here, there is no collision detection and no retransmissions, but n (the expected number of simultaneous transmissions) is assumed to be known in advance. As we will shortly demonstrate, this knowledge of n does not help to increase the network throughput versus Aloha. However, it can be used to set the probability of successful packet reception to an arbitrary desired value.

Claim 4.1: `SimpleJitterSend` guarantees at least an average number of pn successful transmissions in a period of *maxjitter* $= m \cdot$ *msg_length* time.

Proof: Imagine the time being divided into slots, every slot equals the message transmission time. Consider two modes of operations: when the slots are synchronized and when they are not synchronized. Collision occurs if two messages are sent in overlapping slots in unsynchronized model and in the same slot in the synchronized model. In addition, we assume every node sends exactly one message in every *maxjitter* period. That is, for every node the time is divided into epochs of m slots (*maxjitter*=$m \cdot$*msg_length*) and every node sends one broadcast every epoch using the `SimpleJitterSend` algorithm. The beginnings of the epochs are not necessarily synchronized.

Denote by n the number of simultaneous transmissions and let X_i be a 0-1 random variable indicating whether slot i includes a successful transmission. $X = \sum_{i=1}^{m} X_i$ is the total number of successful slots. For a slot to include a successful transmission, there should be exactly one transmission in this slot and zero transmissions in the previous slot (in case of synchronized slots, we drop the latter condition). The probability that slot i includes a successful transmission can be expressed:

$$P(X_i = 1) = n\frac{1}{m}\left(1 - \frac{1}{m}\right)^{n-1}\left(1 - \frac{1}{m}\right)^{n-1} =$$
$$\frac{n}{m}\left(1 - \frac{1}{m}\right)^{2(n-1)} \leq \frac{n}{m}e^{-\frac{2(n-1)}{m}} \approx \frac{n}{m}e^{-\frac{2n}{m}}$$

The average number of successful transmissions in *maxjitter* is: $E(X) = P(X_i = 1) \cdot m \approx ne^{-\frac{2n}{m}} = ne^{-\frac{2n}{-2n/\ln p}} = pn.$ ■

Channel Utilization: The channel utilization of the `SimpleJitterSend` algorithm, denoted U_{SJ} is

$$U_{SJ} = \frac{E(X)}{m} = \frac{n}{m}\left(1 - \frac{1}{m}\right)^{2(n-1)} \approx \frac{n}{m}e^{-\frac{2n}{m}}.$$

For $r = \frac{n}{m}$, $U_{SJ} \approx re^{-2r}$. The maximum utilization of $0.5e^{-1} = 18\%$ is achieved for $r = 0.5$. To get a feel of the result, e.g., for $p = 0.8$, one should use $m = \frac{-2n}{\ln 0.8} = 8.96n \approx 9n$ slots. In this case, U_{SJ} is only $re^{-2r} = 1/9e^{-2/9} = 0.089$.

V. Implementation Strategies

We now turn to describe the main contribution of this paper: the implementation strategies of the jitter mechanism. We target an implementation that achieves the best performance while introducing the minimal amount of changes to already existing networking protocols. That is, our explicit goal is to avoid introducing a new MAC protocol or change an existing one.

We consider three different implementation strategies. For didactic purposes, we first present the simplest implementation in the IP layer, then the MAC-based implementation, and lastly the cross-layer implementation. We prove the correctness of each implementation by reduction from the previous implementation.

As we show later, the last, cross-layer implementation is as efficient as the MAC-based implementation, while requiring only a single configuration change to one MAC protocol parameter (which is allowed by 802.11e [5]). Interestingly, in certain practical conditions (certain density and desired transmission success probability), we can obtain the desired performance even *without changing the MAC configuration at all.*

A. Estimating the number of potential simultaneous transmissions

In order to guarantee a desired non-collision probability, the number of potentially simultaneous transmissions must be known in advance. Notice that for the unicast transmissions, the IEEE 802.11 MAC protocol has an adaptive, exponential backoff mechanism, that increases the contention window if the unicast transmission did not succeed, and by that trying to adapt to the number of simultaneous transmissions. However, this relies on acknowledgments, which are not used for broadcasts, implying one cannot use a similar adaptive mechanism.

Fortunately, it is sometimes possible to estimate or upper bound the number of simultaneous transmissions in advance. Specifically, protocols that need to use the jitter mechanism (as specified by RFC 5148 [1]), can usually know, as part of their internal logic, the number of potential simultaneous transmissions. For example, in multi-hop flooding (e.g., as in AODV [2]), every message that is transmitted for the first time is retransmitted by all neighbors. Thus, the number of simultaneous transmissions equals the size of the neighborhood, denoted by n. In probabilistic flooding [15], [16] (discussed in Section VI), when every node retransmits the message with some probability $\mathcal{P}$, the average number of simultaneous transmissions is $n\mathcal{P}$. In protocols operating over a logical overlay, the number of simultaneous transmissions equals the number of one hop overlay neighbors. At any event, the number of neighbors always upper bounds the actual number of simultaneous transmissions, and thus our mechanism will maintain its correctness (guaranteed number of successful transmissions) always, while using too large jitter in some cases.

We thus suggest that the protocol originating the simultaneous transmissions will estimate their potential number and pass this information to the lower stack protocol, that implements the jitter mechanism. This is an example of a *cross layer design*, in which the application determines the policy, while a lower protocol (IP or MAC) implements the actual mechanism.

B. In the IP layer, without Carrier Sense

When considering the networking protocol stack, a simple way to implement the jitter mechanism is in the stack layer that generates potentially synchronized packets. Yet, if we enable high layers to mark messages as potentially synchronized, or if the potentially synchronized packets are generated by a routing protocol implemented inside the IP layer, then the IP layer can implement jittering in a generic manner. A significant advantage of implementing the jitter in the IP layer is eliminating the queuing effects. Under high load, many messages are sent and are usually being queued at the transport or IP layer. If jittering is performed before the queuing, its effect practically diminishes, since queuing introduces the synchronization again. Therefore, jittering should be implemented at the layer immediately above the MAC layer, *such that after the jitter no additional delays, which might potentially re-synchronize the transmissions, will occur.* For this, we simply assign `IPJitterSend := SimpleJitterSend`.

Discussion: In this simple implementation we do not assume any specific MAC with carrier sensing capabilities, which can stop counting the jitter when other transmissions are in progress. The length of one jitter slot equals the whole duration of a packet transmission. In IEEE 802.11 networks [22], the packet transmission time includes preamble and headers in addition to the actual packet body transmission, which usually happens at the lowest possible rate of 1 or 2 Mbps. These parameters depend on the physical layer. Figure 1 lists different PHY types used by IEEE 802.11. With such overheads, even a very short packet of 24 bytes (such as AODV RREQ message [2]) lasts at least 332 μs.

Applying our analysis of the `SimpleJitterSend` algorithm from Section IV results in a very long jitter. E.g., to guarantee the success for at least 65% of all simultaneous packet transmissions sent in a 7 nodes neighborhood, *maxjitter* should be set to $\frac{-2n}{\ln p}$*slot_size* $= 2 \cdot 7 \cdot 2.3 \cdot 0.332 = 10.69\ ms$. Such a large jitter results in very long latencies, especially for flooded messages that accumulate this latency on every hop. It also greatly decreases the network throughput. This however matches the jitter value proposed in previous studies [4]. Yet, in the past, this value only had an empirical evidence.

C. In the MAC layer, with Carrier Sense (changed MAC)

An alternative way to implement the jitter mechanism is in the MAC layer. As explained in Sec. III-B, IEEE 802.11 MAC already implements a mechanism for random backoff and contention window as part of its collision avoidance mechanism. We now show how this mechanism, with small modifications, provides the same average number of successful transmissions as the `SimpleJitterSend`, yet much more efficiently.

This implementation assumes that a higher layer marks to the MAC potentially synchronized messages, specifies if an additional idle backoff should be used, and what is the new contention window value that should be used for this message backoff. This enables the MAC layer to implement the added jittering functionality for those messages. We eliminate the changes made to the MAC protocol in Section V-D.

Figure 4 depicts the `MACJitterSend` procedure and the changes that should be applied to IEEE 802.11 MAC. When a new packet marked as synchronized broadcast is handled, the MAC should use the new contention window value for any backoff related to this packet (Line 02) (and the regular value of CW_{min} for all other packets). In addition, if the channel is sensed idle by a regular carrier sense procedure (`isCSIdle()` function) and the MAC is not in the middle of a previous

```
Upon MACJitterSend(msg, isSynchBroadcast,
                        newCW, needIdleBackoff) do
(01)    if (isSynchBroadcast) then
(02)        $CW := newCW$;
(03)    else then
(04)        $CW := CW_{min}$;    /*regular CW*/
(05)    if (isCSIdle() && !duringBackoff())
(06)        if (needIdleBackoff) then
(07)            backoff();   /*perform idle backoff*/
(08)    continue as usual;
```

Fig. 4. MACJitterSend implementation. Requires changing the MAC.

backoff, the node should not immediately start the transmission (as currently happens). Instead, it should start an additional random backoff, which we call an *idle backoff* (Line 07). The goal of an idle backoff is to prevent simultaneous synchronized transmissions by nearby nodes. During this backoff, the node performs the regular DCF procedure. In other words, it freezes counting the backoff if the channel becomes busy and sends the packet upon expiration of the backoff timer. The rest of the MAC operations are without changes (e.g, if the channel is sensed busy upon this new packet, the node should not use any additional idle backoff, since it employs a regular backoff).

In order to implement SimpleJitterSend(*msg*, *n*, *p*) in MAC one should issue MACJitterSend(*msg*, *isSynchBroadcast=true*, *newCW*=$\lceil\frac{-2n}{\ln p}\rceil$, *needIdleBackoff=true*). The value of $newCW$ should be set by an upper layer generating the potentially simultaneous transmissions, based on the desired success probability p and the number of such transmissions n. This number can be estimated by the protocol generating these transmissions, e.g., in the way proposed in Section V-A.

Discussion: Slots used by the MAC are considerably shorter (see Fig. 1) and account only for a period long enough to sense a transmission. On the other hand, MAC can sense the carrier and freeze the backoff counting if the channel is sensed busy. Based on these two observations we claim that MACJitterSend with short slots and with carrier sensing achieves at least the same average number of successful transmissions as SimpleJitterSend with long slots without carrier sensing in a significantly shorter period.

Claim 5.1: MACJitterSend algorithm guarantees at least the same average number of successful transmissions as SimpleJitterSend algorithm in a period of $n \cdot$ *msg_length* $+ (2m - n) \cdot$ *MAC_slot* time.

Proof: In the MAC implementation, nodes count backoff in short *MAC_slot*s and at the time of a transmission all other nodes stop the backoff procedure and freeze decrementing their timers. Thus, we can regard the time as being divided into long, occupied, *msg_length* slots (their total number can not be larger than n) and short, idle *MAC_slot*s. Consider only the times when the channel is idle and the backoff procedure is active. In these idle periods, all nodes behave exactly as in SimpleJitterSend. Hence, the number of successful long slots in MAC that hold exactly one transmission is the same as in SimpleJitterSend. Thus, MACJitterSend guarantees the same average number of successful transmissions. After a transmission, the node counts an additional post-backoff. Thus, the total number of *MAC_slot*s is at most $2m$. ■

Note that there is one inaccuracy in the proof above. MAC does not wait the full epoch of m *MAC_slot*s, like SimpleJitterSend does, but may start the next transmission immediately after post-backoff terminates. Thus, it can happen that in the epoch of m *MAC_slot*s there would be more than one transmission attempt by a particular node. This will increase the number of collisions. However, taking this effect into account complicates the analysis significantly without considerable benefit. Thus, we make a simplifying assumption that it will not happen. Indeed, due to the use of a post-backoff, the average (alas not the smallest possible) number of *MAC_slot*s between any two transmission attempts is m.

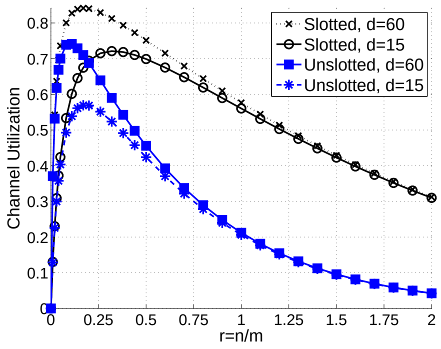


Fig. 5. Channel Utilization of MACJitterSend implementation

Channel Utilization: An interesting aspect of implementing jitter at the MAC layer, is that empty slots are very short compared to occupied ones. Hence, empty slots are much cheaper than slots in which multiple transmissions collided, since the latter are much longer. In particular, we can afford to have a relatively large number of empty slots in order to ensure that most occupied slots will include a successful transmission rather than a collision. This way, the channel utilization will be dominated by the fraction of successful slots, and in particular, will be much higher than can be obtained by the IP implementation presented above.

Denote $p_i = P(X_i = 1)$ the probability of an arbitrary slot i to include a beginning of a successful transmission and p_{empt} the probability of an arbitrary slot to be empty (not to include any transmission). $p_{empt} \approx e^{-r}$ by the same analysis as in the proof of Claim 4.1. The utilization of the link is defined as the duration of time that includes successful transmissions divided by the total time (successful transmissions, collisions, and empty slots). Thus U_{MACJ}, the link utilization when the MACJitterSend procedure is implemented is:

$$\begin{aligned} U_{MACJ} &= \frac{p_i \cdot \textit{msg\_length}}{p_{empt} \cdot \textit{MAC\_slot} + (1 - p_{empt}) \cdot \textit{msg\_length}} \\ &\approx \frac{re^{-2r} \cdot \textit{msg\_length}}{e^{-r} \cdot \textit{MAC\_slot} + (1 - e^{-r}) \cdot \textit{msg\_length}} \end{aligned}$$

Figure 5 depicts U_{MACJ} as a function of n/m (the ratio between the number of concurrent packets and the number of slots) and the ratio $d = \frac{\textit{msg\_length}}{\textit{MAC\_slot}}$. For example, for $r = 1/9$ (which guarantees $p = 0.8$ according to Section IV) and for $d = 15$ (*MAC_slot* of 20 μs and *msg_length* of 300 μs, which is the shortest packet duration), the utilization is 0.55.

D. Combined IP-MAC Implementation (unchanged MAC)

We now propose a combined IP-MAC implementation. In this implementation, we wait a random backoff in the units of short *MAC_slot*s at the IP layer and then invoke a regular, unchanged MAC send procedure, while only using m as the new contention window. That is, this implementation does not change the MAC protocol code and only requires a reconfiguration of the contention window (CW) parameter. Such a reconfiguration is actually allowed by the recent IEEE 802.11e extension [5]. Thus, this implementation obtains the same performance as MACJitterSend, which is much better

```
Upon IPMACJitterSend(msg, n, p) do
(01)  $m := \lceil \frac{-2n}{\ln p} \rceil$;
(02)  wait (random (0, $m$) · MAC_slot);
(03)  regular MAC Send with CW of m;
```

Fig. 6. Combined IPMACJitterSend implementation. All changes are in IP, no MAC changes.

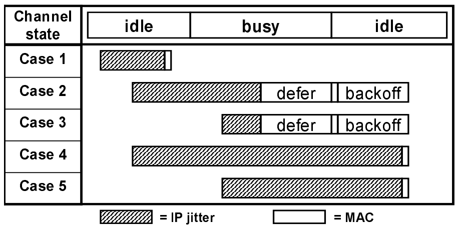


Fig. 7. Channel states during IPMACJitterSend. Same channel for all nodes.

than IPJitterSend, with only a marginal configuration change to the MAC, as listed in Figure 6.

Discussion: We now show that IPMACJitterSend achieves the same average number of successful transmissions as MAC-JitterSend in about the same time period. The intuition is that counting idle jitter in IP is equivalent to counting it in the MAC when the channel is idle. For busy channels, MAC backoff with carrier sensing provides the same jittering effect.

Claim 5.2: IPMACJitterSend algorithm guarantees at least the same average number of successful transmissions as MACJitterSend algorithm in a period of $n \cdot \mathit{msg\_length} + (3m - n) \cdot \mathit{MAC\_slot}$ time.

Proof: Consider the states of a wireless channel (as sensed by the MAC) during various timings of the jitter implementation in IP, as depicted in Fig. 7.

In the first case, the whole jitter waiting in IP is performed while the channel is idle and it is clear that in this case waiting in IP is equivalent to waiting in MAC.

In the second and third cases, counting jitter in IP terminates while the channel is busy. Notice that these are the most frequent cases, since they will almost always happen for all nodes except the one that picked to transmit first. This is since the busy period of the channel, which is equal to the packet transmission time, is usually significantly longer than the jitter waiting time in IP (this jitter is comprised of short *MAC_slot*s). Upon a new message, the MAC will defer the transmission until the channel clears and will perform a backoff using the new contention window value. Thus, in those two cases, simultaneous transmissions would be additionally jittered by the MAC backoff mechanism. Such an additional backoff will introduce the same random jitter as the idle backoff in MAC in the MACJitterSend implementation, since the same size of contention window is used. In total, we will have at least the same number of successful long slots, no more than previously collided long slots and every node will count no more than 3 periods of MAC backoff (one IP jitter, one MAC backoff before the transmission and one MAC post-backoff).

Case number 4 is practically impossible. The jitter value that we propose to use in IP is shorter than the minimal time to transmit a message and thus cannot span such a long duration required for this case to occur.

In case 5, IP wants to transmit the packet while the channel is busy, waits IP jitter and passes the packet to the MAC when the channel is already idle. We further divide this case into two possible scenarios. In the first scenario, the jitter of all competing nodes finished when the channel was already idle. In such a case the fact that there was a busy period in the past does not effect the collision probability, and this scenario is indistinguishable from the case when the whole period was idle (case 1). In the second scenario, at least one other competing node finished its jitter when the channel was still busy. Since this node backed-off when the channel became idle, and since we use the same contention window for this backoff, this scenario is also indistinguishable from case 1. ■

VI. Multiple Hop Flooding

Flooding in multiple hop networks occurs when a source node broadcasts a message to its neighbors, each of which rebroadcast this message to its neighbors once. The goal of a flooding protocol is to deliver the message to all nodes. We refer to the flooding success ratio (or its reliability) as the fraction of nodes that receive the message successfully.

Clearly, for a connected network, if every single hop broadcast is received by all its single hop neighbors, we will also get a 100% flooding reliability, yet with a high level of redundancy. Previous studies [15], [16] have considered a special form of flooding, called probabilistic flooding, whose goal is to obtain high reliability levels with lower message redundancy. We now show a duality between probabilistic flooding insights and our goal of controlling the reliability of flooding with jitter.

In the *Probabilistic Flooding* scheme, each node rebroadcasts a message with a fixed probability $\mathcal{P}$. There is a *threshold probability* $\overline{\mathcal{P}}$, which is around $0.59-0.65$, such that for any $\mathcal{P} > \overline{\mathcal{P}}$ almost all nodes receive the message and for $\mathcal{P} < \overline{\mathcal{P}}$ almost none; the value of $\overline{\mathcal{P}}$ was studied, e.g., in [17] analytically based on percolation theory and in [16] by simulations.

In [16], [17], a perfect (or almost perfect) single hop delivery is assumed. However, as we precisely establish in this work, single-hop success probability is governed by the jitter. In fact, by adjusting *maxjitter* we can control the probability p of a single hop delivery. This probability p could be set to match the above $\mathcal{P}$ rebroadcasting probability, by using our analysis from Section IV. That is, every node rebroadcast the flooding message once, but the probability of its successful reception p within the node's neighborhood will be controlled by the *maxjitter* value so that $p = \mathcal{P}$. This is in contrast with probabilistic flooding, in which only part of the nodes, as determined by $\mathcal{P}$, rebroadcast the message, but it is assumed to be delivered reliably in each one-hop neighborhood.

The work of [16] and RFC 5148 suggest to adapt the jitter based on density and the distance from the source, or maximal accumulated delay. Our scheme already directly accounts for density and can be easily adapted to reflect the distance and accumulated delay.

Finally, notice that in order to guarantee $p = \mathcal{P} \geq 0.65$, one can usually rely on the default contention window value used in the MAC, CW_{min}. For example, for a network with an average density of 7 nodes, CW_{min} of 31 *MAC_slot*s guarantees a probability of $e^{-\frac{2n}{m}} = 0.64$ for every transmission to be successful. Thus, we can achieve the desired $\mathcal{P}$ and as a result a high flooding reliability, with using only short jitter in IP and without changing MAC at all.

VII. Simulations

We have compared our different implementation strategies by simulations under a wide range of varying conditions. In particular, we have considered different network deployments (single/multiple hop with/without hidden nodes), the impact of the number of nodes and nodes density, message sizes, message load, carrier sense threshold parameter, and mobility.

PHY	
Signal Propagation model (PathLoss)	Two-Ray ground reflection
Signal Interference model	Cumulative noise with SNR and capture effect
Transmit power	15 dBm (31.62 mW)
Receive Thresh (RXThresh in ns2)	-71 dBm (7.9432e-8 mW)
Sensitivity Thresh (CSThresh in ns2)	-77 dBm (1.9952e-8 mW)
Background noise (thermal noise)	-101 dBm (8.0080e-11 mW)
SNR (β) (CPThresh in ns2)	10
TX/RX Antenna gain	0 dBm (1 mW)
Fading	None
Ideal Reception range	200 meter
Carrier Sensing Range	299 meter
MAC	
Modulation	DSSS with long preamble and PLCP header
Slot Time	20 μs
aAirPropagationTime	1 μs
aRxTxTurnaroundTime	4 μs
aCCATime	14 μs
DIFS	50 μs
Bandwidth	2 Mbps
Backoff Contention Window	Varying, default is CW_{min}=31
Simulation Scenarios	
Message length	24 to 512 bytes + IP + MAC + PHY headers
Message duration	504 μs to 2456 μs
Nodes	10 default, varying from 3 to 30, and 100
Number of one hop neighbors	10 default, varying from 3 to 30
Mobility	Static (and mobile random way point - graphs omitted due to the lack of space)
Java pseudo random number generator, initialized with the current time in millis as seed	

Fig. 8. Simulation parameters

A. Setup

Our simulations are conducted using the JiST/SWANS simulator of Cornell university [24], which is a discrete event based network simulator that includes an accurate model of a full networking stack. All parameters are summarized in Figure 8.

The signal propagation model is a two-ray ground reflection model. The signal interference model is RadioNoiseAdditive, which is based on a cumulative noise computation with SINR and capture effect. This is equivalent to the Physical model of successful reception described in Section III and is also identical to the interference model used in Glomosim [25] and ns2 version 2.33 [26] simulators (refer to [27] for more details).

The ideal reception range, without any collisions, is set to 200 meters and the average number of neighbors is between 3 and 30 in various simulations (this is achieved by scaling the number of nodes and the simulation area). The nodes are placed at uniformly random locations on a two-dimensional plane. Each simulation lasts 1,000 simulated seconds and each data point is generated as an average of 10 runs. Simulations start after a 60 seconds initialization period.

We have carefully validated the implementation of the IEEE 802.11 MAC DCF procedure in JiST/SWANS and corrected some bugs. In particular, we have made the backoff counting and pausing discrete, as specified by the standard and added simulation of aCCATime (the time the receiver needs to assess the medium and signal to the MAC). As a result, the vulnerable period, during which an ongoing transmission may be corrupted by other nodes, equals exactly *MAC_slot* time of 20 μs.

We compare four implementations:
(1) IP jitter and unchanged MAC: an IP-only based implementation (`IPJitterSend`, Section V-B) in which the jitter is done only in IP and is counted with granularity of 20 μs. MAC uses the default contention window value;
(2) IP jitter and MAC with adapted Backoff: a cross-layer implementation (`IPMACJitterSend`, Section V-D), in which jitter is done both in the IP and in the MAC layer using an adaptive backoff. The difference between this case and the previous case highlights the extra benefit that comes from altering the contention window configuration MAC parameter;
(3) MAC idle jitter and regular Backoff: a MAC-only implementation, in which idle jitter is done in the MAC layer, but without the adaptive MAC backoff mechanism;
(4) MAC idle jitter and adapted Backoff: a MAC-only implementation (`MACJitterSend`, SectionV-C), in which we employ both the MAC idle jitter mechanism and the adaptive MAC backoff mechanism.

The results are also compared to a theoretical reference of slotted and unslotted `SimpleJitterSend`, which are described in Section IV. Note that the jitter suggested in [4] is not depicted in our graphs, since its proposed value of 10 ms is by orders of magnitude larger than the values we propose.

B. Single Hop without Hidden Nodes

Fig. 9(a) depicts a single hop delivery ratio. 10 nodes were placed uniformly on the plane within the transmission range of each other, thus no hidden nodes are present. Nodes send synchronized simultaneous broadcasts, repeated 500 times. The size of each message is 24 bytes + the headers of the IP, MAC, and PHY layers, which results in message duration of 504 μs.

An IP-only implementation with zero *maxjitter* corresponds to unchanged IP and MAC. In such a setting, all broadcasts collide, as expected. All methods except IP-only jitter perform almost identically. They are quite close to, and even slightly better than what was theoretically predicted. This is since simulations are performed in a physical communication model, while the analysis is in the protocol model. In the physical model, sometimes even simultaneous transmissions are successfully received by some nodes (that are close to one of the transmitting nodes and far from the other). We have manually verified that this occasionally happens.

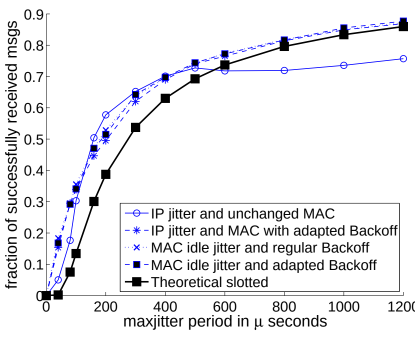
When using an idle backoff in the MAC, it does not matter if the regular backoff uses an adapted contention window or not, because the backoff mechanism almost never kicks in. This is since synchronized broadcasts are sent in epochs under light load and when a new epoch starts, a previous one has already finished in all nodes. Thus, all de-synchronization is performed by the idle MAC backoff.

IP jitter with adapted MAC backoff (`IPMACJitterSend`) performs similarly: the first de-synchronization is done by the IP jitter while the rest are jittered by MAC backoff with an adapted contention window. Recall that the non-adapted MAC backoff contention window is fixed at CW_{min} slots (620 μs). This explains why IP jitter with unmodified MAC does not improve for jitters larger than 620 μs. Yet, for very large IP jitters, it slightly improves. This is since when sending short messages of 500 μs, while *maxjitter* is 1200 μs, IP jitter may span beyond the busy period and its efficiency increases.

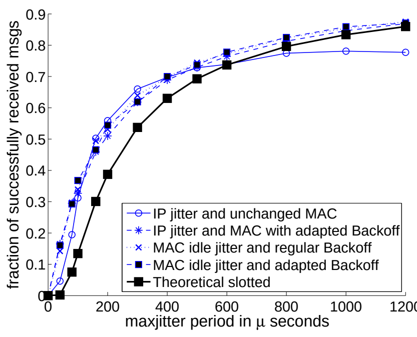
Fig. 9(b) depicts the same scenarios for larger messages of 512 bytes plus headers (duration of 2456 μs). We see that message duration almost does not influence the results. This is expected: as long as the message duration is longer than the jitter, the duration has no effect on the collision probability.

C. Nodes Density Impact

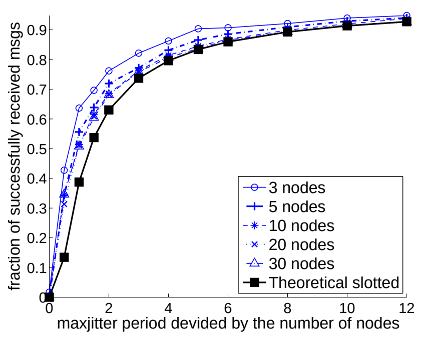
The impact of nodes density is depicted in Fig. 9(c), where the *maxjitter* is normalized by the number of nodes ($\frac{maxjitter}{n}$). All nodes are in the transmission range of each other (no hidden nodes), simultaneously send packets of 512 bytes using `IPMACJitterSend`. We can see that all the curves almost overlap, demonstrating a linear dependence of the successful transmission probability on the number of simultaneous transmissions and thus validating our analysis from Section 3.



(a) 10 nodes, *msg_length*=24B

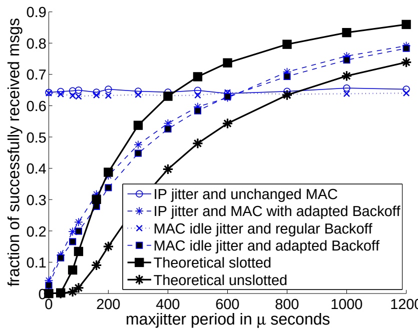


(b) 10 nodes, *msg_length*=512B

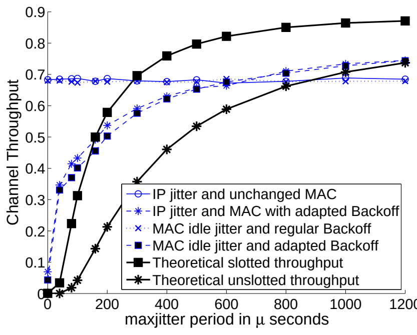


(c) Changing density, `IPMACJitterSend`, x axis is $\frac{maxjitter}{n}$, *msg_length*=512B

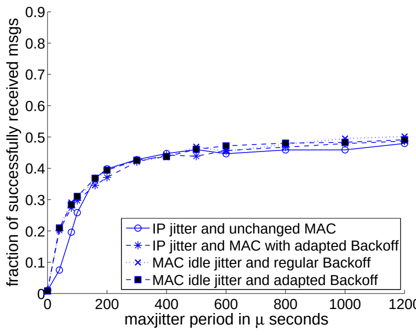
Fig. 9. **Delivery ratio as a function of *maxjitter***: Single hop static network without hidden nodes; simult. broadcasts, repeated 500 times. The jitter suggested by [1], [4] is an order of magnitude larger (10,000 μs).



(a) **Delivery ratio**: No hidden nodes, Saturation load, 10 nodes, *msg_length*=512B



(b) **Channel Throughput**: No hidden nodes, Saturation load, 10 nodes, *msg_length*=512B



(c) **Delivery ratio**: With hidden nodes, 100 nodes, 100 simult. broadcasts, *msg_length*=512B

Fig. 10. **(a,b)**: **Saturation load in Single hop network without hidden nodes**; **(c)**: **Delivery ratio in Single hop network with hidden nodes**: receive range=200 m (10 neighbors, RXThresh=-71dBm); sensing range=300 m (20 neighbors, CSThresh=-77dBm);

D. Performance under Saturation Load in a Single Hop

Figures 10(a) and 10(b) depict the delivery ratio and the throughput of a single hop network under saturation load: nodes broadcast all the time such that IP queues are always full. Thus, the channel is always busy, implying that idle backoff in MAC is never invoked and idle jitter in IP has almost no effect. The transmission success probability is fully governed by the MAC backoff CW. When it is kept constant at CW_{min} slots (620 μs), the delivery ratio does not change. Yet, when it is adapted, the delivery ratio changes accordingly. Notice that the performance of an adaptive backoff is between theoretical slotted and unslotted. This is since under high load the transmissions are not synchronized any more. The 802.11 CSMA behavior under high load is between slotted and unslotted. The capture of a channel by a transmission synchronizes the slots, however since DIFS is 2.5 slots, it unsynchronizes the slots.

Throughput is calculated as the sum of all times when a node successfully receives a message, averaged across all nodes. The throughput is higher than the delivery ratio, since the idle slots are significantly shorter than busy slots that contain transmissions. The throughput is very close to the theoretically predicted one in Fig. 5. Sufficiently large jitter can even achieve the maximal possible MAC throughput of approximately 70%.

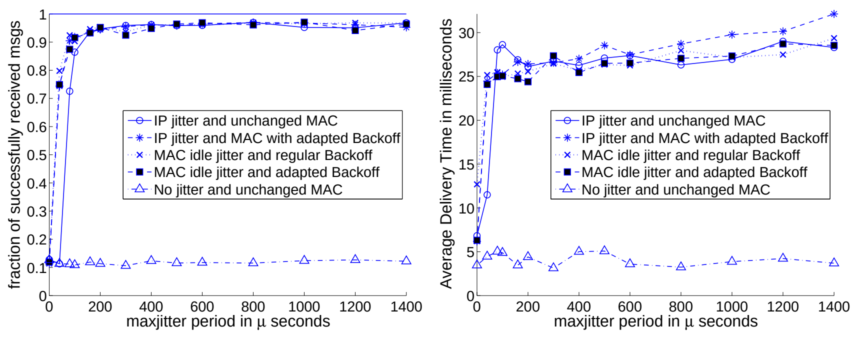
E. Single Hop with Hidden Nodes

We have also investigated the jitter impact on networks with hidden nodes. 100 nodes are placed on a plane of 1000 m × 1000 m with receive range of 200 m (average of 10 neighbors). All nodes send simultaneous broadcasts, 100 broadcasts each. By the theoretical analysis of [10], which is based on a protocol model, one would expect a drastic decrease in the delivery ratio, due to a massive presence of hidden nodes. However, our results indicate differently. In fact, IEEE 802.11 MAC and PHY include a number of mechanisms that assist in overcoming the hidden nodes problem. In particular, under SINR based reception and capture model (physical model), a number of simultaneous transmission can be successfully received; in addition, extended carrier sensing capabilities of the PHY allow a node to sense far away noise and defer its transmission.

Fig. 10(c) depicts the delivery ratio when the carrier sensitivity range was extended to 300 meter (CSThresh=-77dBm resulted in average 20 sensed neighbors). That is, a node can sense a noise in the radius of 300 meters, but it can only successfully capture a message sent within a radius of 200 meters. With no jitter at all, we get only a 5% delivery ratio, which is the performance of a regular unchanged MAC (we have manually validated that these receptions that did occur, happened due to the SINR based interference model and the capture effect). However, extending the jitter improves the delivery ratio significantly. We have also studied the impact of varying the sensitivity threshold. When the sensing range was equal to the receive range, the delivery ratio dropped to 35%. The full results are dropped for lack of space.

F. Multi Hop (with Hidden Nodes)

Fig. 11 depicts the results for flooding in a multi-hop 100 nodes network, with an average density of 10 neighbors, and



(a) Flooding delivery ratio

(b) Flooding average delivery time

Fig. 11. **Flooding in multiple hop static network:** 100 nodes, 10 neighbors, 30 nodes send 100 msgs each, messages of 512 bytes. The impact of jitter on flooding latency is negligible: time to merely transmit a message of 512B (duration of 2.5 ms) over 6 hops is 15 ms, while accumulated jitter is only 1.8 ms.

an average path of 6 hops between any 2 nodes. 30 random nodes send 100 messages of 512 bytes. Without any additional jitter, by simply relying on the unchanged MAC, the delivery ratio is very low. However, with as small as 200 μs jitter in IP or MAC, the delivery ratio jumps to above 0.95. This is as we predicted: jitter of 200 μs guarantees approximately 0.55 single hop delivery ratio (Fig. 9(b)). Such a single hop delivery ratio is above the threshold probability and thus guarantees an almost perfect multi-hop delivery ratio, due to the ability to deliver the same message over multiple paths, as explained in Section VI. This is without using any additional adaptive techniques, which could boost the flooding delivery even further.

The impact of jitter on the flooding latency is depicted in Fig. 11(b). The delivery time is hardly effected by *maxjitter*. This is since *maxjitter* is considerably shorter than the message duration time. The time to merely transmit the message of 512 bytes (duration of 2.5 ms) over 6 hops is 15 ms. A *maxjitter* of 600 μs introduces an accumulated additional average latency of only 1.8 ms. The rest of the latency is due to queueing.

Mobility: We have conducted additional tests in mobile networks with 200 nodes (random waypoint model and speed ranging from 0.5-2 m/s). A jitter of above 200 μs achieved a delivery ratio of 95%. The graphs are omitted for lack of space.

VIII. Discussion

We have proposed a number of implementation strategies of the jittering mechanism in different stack layers.

Implementing jitter at any layer above IP (e.g., at the application layer) brings virtually no benefits, since messages are queued in the IP layer, and thus the jittering just controls the time the messages enter the queue and does not eliminate the synchronization between them when they are serviced.

The previously suggested IP based implementation uses a *maxjitter* of 10 milliseconds [4]. This results in a very poor throughput and long latencies. For example, the theoretical throughput of such an implementation for 10 competing transmissions of short 24 bytes messages with headers is only 17% (Sec. IV).

Implementing jitter completely inside the MAC layer allows tunable *maxjitter* values. Low latency and throughout of up to 70% is achievable with short *maxjitter* of only 600 μs (Fig. 5 and 10(b)). However, this implementation induces a slight modification to the protocol's code.

Our cross layer IP-MAC implementation (`IPMACJitter-Send`) benefits from all worlds: very good performance, short latencies, and only minimal configuration change to the MAC protocol (only adjusting the contention window value, which is in any case allowed under the IEEE 802.11e extension).

References

[1] IETF Mobile Ad-hoc Networks Working Group, "RFC 5148: Jitter considerations in Mobile Ad Hoc Networks (MANETs)." [Online]. www.ietf.org/rfc/rfc5148.txt
[2] ——, "RFC 3561: Ad hoc On-Demand Distance Vector Routing (AODV)." [Online]. www.ietf.org/rfc/rfc3561.txt
[3] ——, "RFC 3626: Optimized Link State Routing Protocol (OLSR)." [Online]. www.ietf.org/rfc/rfc3626.txt
[4] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *MobiCom*, 1998, pp. 85–97.
[5] IEEE Computer Society, "802.11e: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: MAC Enhancements for Quality of Service (QoS)," January 2005.
[6] M. Nassiri, M. Heusse, and A. Duda, "A Novel Access Method for Supporting Absolute and Proportional Priorities in 802.11 WLANs," in *IEEE Infocom*, April 2008, pp. 709–717.
[7] Y. Yang and R. Kravets, "Achieving Delay Guarantees in Ad Hoc Networks through Dynamic Contention Window Adaptation," in *IEEE Infocom*, 2006, pp. 1–12.
[8] G. Bianchi, "Performance analysis of the IEEE 802.11 Distributed Coordination Function," *IEEE J. Select. Areas Commun.*, vol. 18, no. 3, pp. 535–547, March 2000.
[9] R. Oliveira, L. Bernardo, and P. Pinto, "The Influence of Broadcast Traffic on IEEE 802.11 DCF Networks," *Comp. Comm.*, vol. 32(2), Feb. 2009.
[10] L. Wu and P. K. Varshney, "Performance analysis of CSMA and BTMA protocols in multihop networks: (I). Single channel case," *Information Sciences – Informatics and CS: An Intr. Jour.*, vol. 120, no. 1-4, 1999.
[11] H. Takagi and L. Kleinrock, "Optimal transmission ranges for randomly distributed packet radio terminals," *IEEE Trans. Commun.*, vol. 32, pp. 246–257, March 1984.
[12] Y. Tay, K. Jamieson, and H. Balakrishnan, "Collision-Minimizing CSMA and its Applications to Wireless Sensor Networks," *IEEE J. Select. Areas Commun.*, pp. 1048–1057, August 2004.
[13] Y. Kanizo, D. Hay, and I. Keslassy, "Optimal fast hashing," in *IEEE Infocom*, 2009.
[14] T. S. J. Shi, E. Aryafar and E. Knightly, "Synchronized CSMA Contention: Model, Implementation and Evaluation," *IEEE Infocom*, 2009.
[15] Y.-C. Tseng, S.-Y. Ni, Y.-S. Chen, and J.-P. Sheu, "The broadcast storm problem in a mobile ad hoc network," *Wireless Networks*, vol. 8, no. 2/3, pp. 153–167, 2002.
[16] Z. Haas, J. Halpern, and L. Li, "Gossip-Based Ad Hoc Routing," in *INFOCOM*, 2002, pp. 1707–1716.
[17] Y. Sasson, D. Cavin, and A. Schiper, "Probabilistic Broadcast for Flooding in Wireless Mobile Ad hoc Networks," in *WCNC*, March 2003.
[18] K. Viswanath and K. Obraczka, "Modeling the Performance of Flooding in Wireless Multi-Hop Ad Hoc Networks," *Comp. Comm.*, vol. 29(8).
[19] P. Mannersalo, A. Keshavarz-Haddad, and R. Riedi, "Broadcast Flooding Revisited: Survivability and Latency," in *INFOCOM*, 2007, pp. 652–660.
[20] P. Gupta and P. Kumar, "The capacity of wireless networks," *IEEE Trans. Inform. Theory*, vol. 46, no. 2, pp. 388–404, March 2000.
[21] A. Keshavarz-Haddad, V. Ribeiro, and R. Riedi, "Broadcast capacity in multihop wireless networks," in *MobiCom*, 2006, pp. 239–250.
[22] IEEE Computer Society, "802.11: Wireless LAN Media Access Control (MAC) and Physical Layer (PHY) Specifications."
[23] N. Abramson, "The ALOHA System–Another Alternative for Computer Communication," in *Fall Joint Comput. Conf.*, vol. 37, 1970, pp. 281–285.
[24] Cornell University, "JiST/SWANS Java in Simulation Time / Scalable Wireless Ad Hoc Network Simulator." [Online]. http://jist.ece.cornell.edu
[25] "GloMoSim - Global Mobile Information Systems Simulation Library."
[26] "The Network Simulator NS-2." [Online]. http://www.isi.edu/nsnam/ns/
[27] G. Kliot, "Wireless signal interference models made simple." [Online]. www.cs.technion.ac.il/~gabik/Jist-Swans/signal_interference