Multi-hop Broadcast from Theory to Reality: Practical Design for Ad Hoc Networks

Alaeddine El Fawal
EPFL, I&C
CH-1015 Lausanne, Switzerland
alaeddine.elfawal@epfl.ch

Jean-Yves Le Boudec
EPFL, I&C
CH-1015 Lausanne, Switzerland
jean-yves.leboudec@epfl.ch

Kave Salamatian
EPFL, I&C
CH-1015 Lausanne, Switzerland
kave.salamatian@epfl.ch

ABSTRACT

We propose a complete design for a scope limited, multi-hop broadcast middleware, which is adapted to the variability of the ad-hoc environment and works in unlimited ad-hoc networks such as a crowd in a city, or car passengers in a busy highway system. We address practical problems posed by: the impossibility to set the TTL correctly at all times, the poor performance of multiple access protocols in broadcast mode, flow control when there is no acknowledgment and scheduling of multiple concurrent broadcasts. Our design, called "Self Limiting Epidemic Forwarding" (SLEF), automatically adapts its behavior from single hop MAC layer broadcast to epidemic forwarding when the environment changes from being extremely dense to sparse, sporadically connected. A main feature of SLEF is a non-classical manipulation of the TTL field, which combines the usual decrement-when-sending to many very small decrements when receiving. SLEF is intended as a replacement of k-hop limited broadcast for the unlimited ad-hoc setting.

1. INTRODUCTION

Broadcast exists inherently in the wireless channel and is used by several communication systems in ad hoc networks. It can either be single hop (the native MAC layer broadcast), or multihop. In static scenarios, multihop broadcast can simply be implemented as follows: The source generates an IP packet with TTL=k and sends it as a MAC layer broadcast; any node that receives it decrements its TTL and if the result is positive, schedules it for a new transmission as a MAC layer broadcast. In Disruption Tolerant ad-hoc Networks (DTNs), multihop broadcast is implemented by some form of epidemic forwarding: Nodes repeat packets they receive with some probability, possibly more than once, in order to extend the spread (number of nodes that receive the packets) while mitigating redundancy. In single-hop or multi-hop forms, broadcast is used to disseminate information in quickly varying environments (e.g. opportunistic networks), where mobility and self-organization make the classical methods based on distribution trees non-practical. Also, it can be used to support routing, resource discovery protocols or in bootstrapping phases for application layer protocols: for instance the "Spray" phase in the Spray-and-Focus protocol [10] is a form of multi-hop broadcast.

We consider open ad-hoc networks, such as a crowd in a city, or car passengers in a busy highway system. A common feature here is that there is no practical bound on the number of users (unlimited network), and contact times may be short and unpredictable. In practice, implementing multihop broadcast in such settings poses a number of practical challenges, which, if not correctly addressed, may lead to very poor performance. A first issue is how to set the TTL correctly. Consider for example an application that uses multihop broadcast in a vehicular network; the connectivity may range from sparse and sporadic (lightly loaded highway) to very dense (traffic jam, city center). Furthermore, changes from one setting to another may be very sudden. We show in our performance studies that, in a traffic jam with realistic parameters, there are around 200 nodes within range, and any TTL setting $k > 1$ results in congestion collapse. In contrast, in a sparse setting, $k = 1$ results in practically no dissemination. Thus, the TTL, if used in that form, should be set adaptively. A second issue is the absence of acknowledgment in MAC layer broadcasts (e.g. with 802.11). A node cannot know if its transmitted packet is received by someone else, it simply undergoes a collision or it is transmitted in a vacuum. Also, mutual exclusion mechanisms (as CSMA/CA) that manage collisions are usually not implemented in broadcast mode (for example in IEEE 802.11). Therefore, accessing the medium in broadcast mode is similar to ALOHA that performs poorly. A third issue is flow control, i.e. how to control the packet injection rate of the application. This is normally done end-to-end by TCP, but here this probably does not apply. If the injection rate is not adapted to the network conditions, this may result in congestion collapses and failure of the broadcast. A fourth issue is scheduling among competing broadcasts. It is likely that more than one broadcast packets are competing for retransmission at one node, and some form of mechanism is required to know which packet to select next.

In this paper we propose a complete design for a scope limited, multi-hop broadcast middleware that is adapted to the ad-hoc environment and addresses all of the above issues. It performs well in DTNs, as well as in other settings, in particular in very dense networks (as in a traffic jam). We call our system "Self Limiting Epidemic Forwarding" (SLEF). SLEF adapts to a rapidly varying environment in a way that is completely transparent to the application. In a very dense environment, SLEF is equivalent to a single hop broadcast; in a sparse environment, to a k-hop broadcast, with k automatically adapted to the network conditions. In DTNs, it performs as an epidemic system, i.e. packets may be re-transmitted more than once if this is required to achieve a good performance. SLEF achieves these goals

by a number of mechanisms, described in the next section. A main feature of SLEF is a non-classical manipulation of the TTL field, which combines the usual decrement-when-sending with many very small decrements when receiving.

The SLEF middleware offers the following service to the application. It delivers packets as a limited multi-hop broadcast, without the application having to bother about what the current state of the ad-hoc environment is. The service interface is flow controlled, i.e. the application can send only up to a maximum rate determined by SLEF. This rate is controlled so as to ensure a reasonable balance between spread (number of nodes that receive the broadcast) and rate, in an open, unlimited environment.

When using SLEF, an application has to specify two parameters, K_0 and K_1, which control the spread-rate balance. We provide default values for K_0, K_1, which we show by analysis and simulation to perform well; we also give some guidelines on how to set K_0, K_1 if one wished to depart from the default values (in order for example to reach higher spread at the expense of lower rate). As to other SLEF parameters, they are fixed and they are chosen as explained in Sect. 3.3.

This paper is organized as follows. In Section 2, we define the functions achieved by SLEF and explain how they are implemented, which covers the whole design. In section 3, we derive an analytical model in order to tune the design parameters and explain the interaction among its different components. This section ends by giving default values to the parameters of our design. In section 4, we give guidelines on how to set K_0 and K_1 to adjust the spread-rate balance. In section 5, we validate our design through simulations. In section 6, we conclude this paper.

2. FUNCTIONS

In order to achieve its goal as a practical broadcast middleware, SLEF has to implement six mandatory functions. The first function is inhibition, which aims at mitigating redundancy. It adapts the forwarding factor (i.e. the number of times a node forwards a packet) based on the send/receive events seen on the same packet: A packet that is seen for the first time, has a high chance to be forwarded, whereas a packet that has seen several send/receive events is considered as well propagated and its chance to be forwarded is lower. We call it inhibition, as it inhibits nodes from forwarding over-sent/received packets. The second function is spread control. It adapts the spread to the network state in order to guarantee a minimum rate for the application when the network scales. It is based on an aging mechanism that decrements a packet TTL field locally based on the receive events seen by the node. The third function is scheduling, which decides which packet to deliver to MAC for transmission. It is likely that more than one packet are competing for transmission at a node. These packets might belong to different sources and might have seen different numbers of send/receive events. In order to execute the inhibition, our scheduler has to serve packets according to these numbers send/receive events, and thus it can not be based on naive policies such as First In First Out (FIFO). Moreover, our scheduler considers the packet source Ids in order to achieve source-based fairness. The fourth function is congestion control. It consists of adapting the application injection rate, not only to avoid local buffer overflow, but it goes one step further: If the injection rate is higher than the forwarding capacity of the source neighbors, the source packets will be accumulated in the neighbor buffers and dropped before being forwarded. Therefore, we adapt the injection rate to the forwarding capacity in order to allow packets to propagate in the network. The fifth function is buffer management, which decides when to drop packets in order to keep space in the buffer for the new incoming packets. In general, our buffer management drops first the packet the most propagated in the network. The sixth function consists of careful use of the MAC broadcast. To compensate for the absence of the mutual exclusion and the acknowledgment in the MAC broadcast (see Sect. intro), we use two mechanisms: pseudo-broadcast and presence indicator. The former implements a CSMA/CA-based mutual exclusion. The latter returns true if some neighbors were present around a node while it was transmitting a packet. In this case, the node considers that the transmission was successful and was not in the vacuum.

All these functions are achieved using only local information to the node and do not need any knowledge about the network topology. In the following, we describe in detail the solutions we propose to achieve them. A pseudo-code of the design is available online [1]. For ease of understanding, we begin by defining the main variables used in our design and the terminology that will be used in the explanation.

2.1 Variable and Terminology Definition

Every node maintains one *epidemic buffer*, used to store received and locally originated packets, with the following attributes:

- `sendCount` : how many times this packet was sent by this node
- `rcvCount` : how many times this packet or a duplicate was received by this node
- `vRate` ("virtual rate"): This attribute is derived from `sendCount` and `rcvCount` , using the method described in Sect. 2.2. It is the rate at which this packet would be transmitted if it were alone in the epidemic buffer.
- `age` : combines hop count, real time age (true time to live) and adaptive age, which reflects the amount of competition this packet and its ancestors have encountered so far
- `earliestSendTime` and `pendingSendConfirmation` : see Sect. 2.4 and Sect. 2.7

We call *clone* the set made of an original packet and its duplicates; all packets in the same clone have the same value for source address (IP address) and the identification field. When a packet is received, it is inserted into the epidemic buffer. If this is the first time a packet of this clone is seen by this node, a new entry is created, otherwise if the existing entry is still present, it is overwritten (thus there is always at most one packet per clone in the epidemic buffer). The attributes are updated as explained later. We call self-packets the packets originated by the node, and foreign-packets the packets received from other nodes.

2.2 Inhibition

Inhibition mechanisms aim at preventing nodes from forwarding over-sent or over-received packets in order to minimize redundancy. Different approaches to inhibition are proposed in the literature. They differ in being adaptive or not, and in the information they need about the topology and neighbors. In [8], the authors propose a Gossip-based inhibition mechanisms, where a node decides to forward a packet with a fixed probability p and drop it with $(1-p)$. It

needs information about the number of neighbors to find an appropriate value of p, which is impractical in highly mobile networks specially with short contact time. Further, this inhibition mechanism is not adaptive, as it does not include any mechanism to adapt p. In [9], the authors propose a counter-based inhibition. It consists of discarding a packet when its `rcvCount` reaches a maximum value, assuming that it is well propagated and that forwarding it is redundant. It is an adaptive mechanism. It allows a packet to be forwarded at most one time, then it is discarded. If this transmission fails because of collisions or because it was in the vacuum, the packet will never be retransmitted.

Our inhibition mechanism is adaptive. It allows for forwarding of packets many times to recover from collision or transmitting in the vacuum, as we will see later in Sect. 3.1.2. Note that forwarding packets many times is very useful in highly mobile networks, as several transmissions of the same packet occur in different locations and face different neighbors, and thus, increase the packet spread without adding redundancy.

With our mechanism, a packet in the epidemic buffer is retransmitted with a probability that depends on its `vRate` , which we define as:

$$\texttt{vRate} \leftarrow R_0 a^{\texttt{rcvCount}}\ b^{\texttt{sendCount}}$$

where R_0 is the nominal rate in packets per second of the MAC layer interface and a and b are unit-less constants less than 1. Thus the virtual rate of a packet decreases exponentially with any send/receive event of the same packet. The scheduler (see Sect. 2.4) decides which packet is selected next for transmission by the MAC layer; it serves packets with rates not exceeding their virtual rates. Hence, a packet in the epidemic buffer, which has seen many send/receive events, is scheduled at a very low rate, and it is more likely that it will be dropped by the buffer management (see Sect. 2.6) mechanism before being transmitted.

2.3 Spread Control

We argued in the introduction that spread control is needed to ensure that the transmission rate of any user is satisfactory. Formally speaking, let λ be the user application rate (generating new information to forward), FF the forwarding factor, S the spread and R the available transmission rate over the channel, which includes self and foreign packets . In a symmetric network where self-packets are transmitted only once and foreign-packets forwarded FF times we have:

$$\lambda + FF * \lambda * S = R \Leftrightarrow \lambda = \frac{R}{1 + FF * S} \qquad (1)$$

So the rate-spread trade-off is obvious.

A natural way to limit the spread is to use the classic TTL, which is the method that comes by default with the Internet Protocol (IP). When a packet is created by a source and placed into the epidemic buffer, it receives a TTL value equal to some positive constant `maxTTL` . When the packet is accepted for transmission by the MAC layer, the TTL field of the *transmitted* packet is equal to the value of the TTL field in the packet in the epidemic buffer, minus 1. The TTL field in the packet stored in the epidemic buffer is unchanged.

When a packet created by some other node is received for the first time at this node, the value of the TTL is screened. If it is equal to 0, it cannot be retransmitted and the packet is discarded. Otherwise (TTL$\geq$ 1), the packet is stored in the epidemic buffer, with TTL equal to the value present in the received packet. When and if the packet is later accepted for transmission by the MAC layer, the transmitted TTL field is equal to the stored TTL minus 1, and the stored TTL is unchanged.

A potential problem with the classic TTL is that it does not adapt to the node connectivity. In a very dense network, we should choose a very small value of `maxTTL` to limit the number of hops and the spread. In contrast, a large value of `maxTTL` is preferable in sparse networks.

We propose an aging-based spread control mechanism that adapts itself to the node density and traffic load. With this mechanism, the `age` attribute is inherited when a packet is received for the first time and is equal to 0 for a newly created clone for a self packet. The `age` stored in the epidemic buffer is a floating point number. It increases depending on the events affecting the packet and the state of the epidemic buffer. When transmitting a packet, the complement to `maxTTL`(=255) of `age` , rounded to an integer, is written in the IPv4 TTL field [resp.IPv6 hop count]. Similarly, when a packet is received for the first time, its `age` is extracted from the TTL/hop count field: $\texttt{age} = \texttt{maxTTL} - TTL$.

There are three processes that increase the `age` :

- (hop count): `age` is incremented by a constant amount K_0 whenever either this packet is transmitted or a duplicate is received.
- (real time age): `age` increases at a constant rate $\alpha = 32h^{-1}$. We assume that nodes have free running clocks; there is no need for time synchronization. The constant α is such that a packet lives at most 8 hours.
- (adaptive age): `age` of all packets stored in the epidemic buffer increases by an amount K_1 every time a packet (of an existing or new clone) is received. The adaptive aging constant K_1 is a (possibly non integer) constant less than K_0; its value will be discussed in Sect. 3. For self-packets with $\texttt{sendCount} == 0$, this process is valid up to a threshold that we call Self Age Threshold (SAT). When the `age` reaches SAT, it passes immediately to `maxTTL` and it will never be incremented by K_1 until the packet is transmitted. SAT is computed through a density detection mechanism, explained later in this section.

A packet is killed whenever its `age` is too large to be sent, i.e. when $\texttt{age} \geq \texttt{maxTTL} + 1$ (the +1 is due to rounding). An exception is made for self-packets with $\texttt{sendCount} == 0$, they are not discarded before being transmitted at least once, even if their `age` exceeds $\texttt{maxTTL} + 1$. This happens in very congested networks where self-packets have to stay a long time in the epidemic buffer before being transmitted.

When a packet is received for a clone that is present in the epidemic buffer, the TTL of the received packet is ignored and only the increase by K_0 is applied to the `age` of the packet already present in the epidemic buffer. This is to limit the harm of any spurious malimanipulation of TTL by cheaters [5].

The behavior of hop-count and real-age processes is intuitive, whereas the behavior of the adaptive age needs more explanation. Let N be the number of neighbors a node has, R_0 the MAC nominal capacity in packets/s and γ the channel utilization. In case where each node is running a greedy application (which always has a packet to send), the packet reception rate can be approximated by $\tau_r = \frac{N}{N+1} * \gamma * R_0$, assuming fairness in MAC layer. Thus, the adaptive age increases with a rate equal to $\tau = K_1 * \tau_r$. As we will see in

Sect. 5, a numerical example could be: N = 240 (traffic jam), $R0 = 83 packets/s$ (MAC rate = 1Mbps and packet size of 1500 bytes), $\gamma = 0.7$ and $K_1 = 0.1$. As a result, $\tau = 5.8s^{-1}$. That is, a packet can stay at most $\frac{\texttt{maxTTL}}{\tau} = 44s$ in the epidemic buffer before being rejected. Adding the impact of other `age` components, a packet stay will never reach $\frac{\texttt{maxTTL}}{\tau}$.

A density detection mechanism is implemented in order to strictly limit the communication to one hop in very dense networks. It consists of computing SAT as follows:

1. At the beginning: $SAT \leftarrow SAT_0$ ($SAT_0 = 10$ as computed in Sect. 3.3):
2. Upon each reception, which is considered as indication of high node density, SAT is decremented by K_1 in order to limit the number of hops:
$$SAT \leftarrow max(SAT_0, SAT - K_1)$$
3. Upon each transmission, which is considered as an indication of a low node density, SAT is incremented by SAT_0 in order to allow more hops:
$$SAT \leftarrow min(\texttt{maxTTL}\ , SAT + SAT_0)$$

We set SAT_0 to 10, which corresponds approximately to a maximal spread of 100 nodes (see Sect. 3.3). Thus, SAT will be very close to one of two values in steady state, which is reached in a few seconds. SAT is very close to SAT_0 (=10) in a very dense network where the number of nodes within transmission reach is larger than 100. Hence, a self packet will be transmitted with `age` = `maxTTL` and thus, we ensure only one-hop communication. In contrast, SAT is very close to `maxTTL` in a sparse network that allows several hops communication.

2.4 Scheduler

The scheduler decides which packet in the epidemic buffer is selected for transmission (being passed to the MAC layer).

In order to ensure source-based fairness, the scheduler serves packets per source IP address, using a processor sharing approach. Furthermore, every packet should be served at a rate not exceeding its `vRate` (see Sect. 2.2).

Every packet in the epidemic buffer has a derived attribute `earliestSendTime` , equal to the last time the `vRate` of this packet was modified, plus $\frac{1}{\texttt{vRate}}$. At any time t, a packet is said to be "eligible" if it has `earliestSendTime` $\leq t$. Eligible packets with the same source IP address are linked in one FIFO per source. Each of these FIFOs has an attribute `sourceClaim` , which keeps track of how much this source can claim to be scheduled. It is initially 0 and is decremented by 1 when this source is selected for transmission by the scheduler. It is incremented by 1 divided by the number of sources in the epidemic buffer whenever a packet is scheduled for transmission.

The scheduler issues a blocking send function to the MAC layer that returns whenever the packet is accepted by the MAC layer. When this method returns, the scheduler looks for another packet to deliver to MAC. It selects the source with the highest `sourceClaim` that has eligible packets (none-empty FIFO). In case it does not find eligible packet, it waits until one becomes available.

It can be seen that this algorithm allocates the transmission opportunities according to a water-filling algorithm, thus, it approximates an ideal fluid scheduler that would allocate rates to sources in a max-min fair way, subject to the constraint that the rate of a source does not exceed the sum of the `vRates` of the packets of this source.

2.5 Congestion Control

Our congestion control consists of adapting the application injection rate to the network conditions.

SLEF allows the existence of at most σ self-packets in the epidemic buffer (we set σ to 2). The application is allowed to inject a new packet in the epidemic buffer in one of three cases.

The first is when the number of self-packets in the epidemic buffer is less than σ. It happens either at the bootstrap of the application or when a self-packet is dropped because it has seen numerous send/receive events and its `age` has reached `maxTTL` . Thus, we assume that this packet has lived enough to proliferate in the network.

The second case is when the epidemic buffer contains σ self-packets but, at least one of them has seen a duplicate forwarded by a neighbor. Indeed, SLEF considers the received duplicate as an implicit acknowledgment (Ack) and that the neighborhood has enough capacity to propagate the packet in the network. In this case, the acknowledged packet is dropped when the application injects a new packet.

The third case is also when the epidemic buffer contains σ self-packets but, this time, at least the `sendCount` of one of them has reached 3, which is an indication that the packet is received by some other node. This is to avoid that the application is blocked for long time in case SLEF has not received any implicit Ack for the self-packets existing in the epidemic buffer. Indeed, it might happen that the implicit Ack undergoes a collision and it is not received by the source. Therefore, the source continues transmitting the packet, and at the same time, inhibiting its neighborhood from forwarding it (see Sect. 2.2), and thus, it might never receive an implicit Ack for this packet. Note that, the `sendCount` is not incremented unless we are sure that, with high probability, a transmission is received by other nodes (as explained later in Sect. 2.7).

2.6 Buffer Management

The buffer management aims at cleaning the epidemic buffer to save space for new incoming packets. The cleaning process distinguishes between foreign and self-packets. A foreign-packet is dropped when its `age` becomes larger than `maxTTL` . For nodes with very limited buffer size, this may not be sufficient. If an arriving packets requires space to be freed, the foreign packet with the largest `age` is deleted.

As to a self-packet, it is dropped in one of three cases. The first is when its `age` exceeds `maxTTL` and its `sendCount` is strictly positive. The second is when it is implicitly acknowledged and the epidemic buffer contains σ self-packets. This packet is deleted when the application injects a new packet. The third is similar to the second, except that the `sendCount` of the packet reaches 3 instead of being acknowledged.

Applying Little formula [4] on our `age` based buffer management, we find that the epidemic buffer size is upper bounded by $\frac{\texttt{maxTTL}\ +1}{K_1}$. To understand this, assume $K_0 = 0$ and a node starts receiving packets, all with `age` $= 0$. Thus, this node starts dropping packets after receiving $\frac{\texttt{maxTTL}\ +1}{K_1}$ packets as the `age` of the first packet received is equal now to `maxTTL` $+1$. This node continues dropping packets with a rate equal to the receiving rate and its epidemic buffer size becomes constant equal to $\frac{\texttt{maxTTL}\ +1}{K_1}$.

2.7 Careful Use of MAC Broadcast

We assume that nodes have a MAC layer capable of receiv-

ing and sending packets in broadcast mode, at a rate that depends on the network conditions (and is likely to be much less than the peak transmission rate R_0 used above). In practice, if we use the IEEE 802.11 MAC broadcast, there is a performance issue, as it does not use the RTS/CTS exchange and collisions during transmission go undetected. To avoid this issue, we use the pseudo-broadcast mode proposed in [6], by which a packet is sent to the MAC address of a neighbor (with RTS/CTS), but can be promiscuously copied by all systems within range. This effectively solves much of the performance issue, but may not always be applicable to our case, since we do not want nodes to spend time discovering their neighbors' MAC addresses. Therefore, we use the following method. The MAC layer has a node global MAC state information that says whether the next packet will be sent in pseudo-broadcast, and if so, to which MAC address, or in broadcast mode. The destination MAC address in the pseudo-broadcast mode is the source MAC address of the last received packet. As soon as the node receives one packet, the MAC state is set to pseudo-broadcast. The next packet is thus sent with an RTS. If no CTS is received in response, the MAC layer backs off for a random time (this is the standard operation of 802.11). If during the back-off time a packet is received, the packet is retransmitted (after expiration of the back-off timer) in pseudo-broadcast mode to the MAC address of the newly received packet. Else the MAC state moves to broadcast, and the packet is re-transmitted in broadcast mode.

There remains one issue, however, where a node does not know if a sent packet was received by another; this might become a problem in the desert highway scenario, where a node would repeatedly send a packet in the vacuum, until it ages out. To avoid this, we use two heuristics when sending in broadcast mode: (1) indication of neighbor presence, and (2) implicit acknowledgment by reception of duplicate. (1) consists in building a function around the MAC layer that says whether, shortly before or after a packet transmission in broadcast mode, the carrier is sensed busy. If a packet is sent in the former case, or in pseudo-broadcast mode (some neighbors are around) then `sendCount` is incremented and a flag we call `pendingSendConfirmation` is set to false. Of course, there is no guarantee that a packet sent in these circumstances is actually received by any one, but the rules for rate adaptation will make it likely for this packet to be retransmitted soon if no duplicate is received (in such a case `vRate` remains large). If in contrast a packet is sent in the latter case (presumably because there is no one around), the flag `pendingSendConfirmation` is set to true for this packet, and the packet is rescheduled at a later date with the same `vRate` . If a packet of the same clone is received while `pendingSendConfirmation` is true, the pending transmission is considered successful. The condition `pendingSendConfirmation` == true can be terminated either by reception of a duplicate or by a subsequent transmission that returns an indication of presence or is in pseudo-broadcast mode.

3. DESIGN TUNING

Through out this section, we derive simple mathematical equations that describe the behavior of our design and show the interaction among its different components. This analysis allows us to find ranges for our design parameters (a,b,K_0, K_1, and SAT_0) whitin which, the system performs well in a wide range of settings. We wanted our analysis to be very simple and intuitive for ease of understanding. Nevertheless, the analysis allows for a deep understanding of the impact of each parameter and the interaction among the different SLEF components, and delivers well-tuned parameter ranges.

We consider only two extreme cases, very sparse and very dense, where we find suitable ranges for the parameters. In intermediate cases, SLEF is able to adapt itself without any need to change its parameters. We validate its capacity to adapt later through simulations (see Sect. 5).

In both scenarios, we assume that each node is running a greedy application (which always has a packet to send) and that the network is symmetric. It follows that:

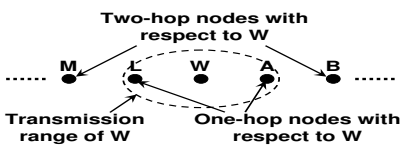
1. The average stay durations of a given packet in each epidemic buffer are the same.
2. All packets have the same average stay duration in a given epidemic buffer.

These two points are ensured by the source-based fairness delivered by the scheduler.

Before starting our analysis, we define our notations:

- R_0: Nominal MAC rate [packets/s].
- γ: Channel utilization.
- N: Number of neighbors within the transmission range of a source (excluding the source).
- S: Spread, including the source itself.
- λ: Application rate.
- H: Minimum number of hops in the absence of collision; neighbors within the transmission range are considered as one hop.
- R: MAC effective transmission rate [packets/s]. We approximate it by:
$$R = \frac{1}{N+1}\gamma R_0 \quad (2)$$
- τ: Adaptive age increasing rate of a packet in the epidemic buffer caused by receive-events. We approximate it by:
$$\tau = \frac{N}{N+1}\gamma R_0 K_1 \quad (3)$$
where ($\frac{N}{N+1}\gamma R_0 = N * R$) is the packet receiving rate. It is clear that the τ unit is [age units/second].
- D_F: Average delay a FIFO undergoes to be served once. This delay is due to the competition among different FIFOs in the epidemic buffer.
- D_M: Average delay a packet undergoes at MAC layer due to the competition among nodes in accessing the medium. We have:
$$D_M = \frac{1}{R} \quad (4)$$
- H_r: Required number of hops; the minimum that we want to ensure in presence of collisions.
- P_c: Probability of collision on one side of a node in the linear grid of the sparse scenario (see Sect. 3.1).

3.1 Sparse Scenario



Each node is connected only to its two closest neighbors: for instance, node W is connected only to nodes L and A. Nodes A and L are the one-hop nodes of W. Nodes M and B are the two-hop nodes of W, and so on.

Figure 1: The linear grid considered in the sparse scenario.

In this scenario, we consider the linear grid in Fig. 1, where each node has only two nodes within transmission range, the previous and the next nodes. As this is a sparse scenario, SAT is equal to `maxTTL` and does not appear in the analysis of this section.

In the following, we derive system equations according to two requirements. The first is that we require H-hop broadcast in the absence of collision. The second is that we require at least H_r hops ($H_r < H$) in presence of collision, where packets are forwarded more than once in case of collision. Then, we finish this section with an interpretation of the obtained analysis and with a parameter tuning.

Before beginning the equation derivation according to the two requirements, we show a general constraint that should be respected in the remaining of this section. In order to let packets propagate beyond the one-hop nodes, we should have:

$$b < a \tag{5}$$

To understand this, let us consider this example: when W in Fig. 1 transmits a self-packet, if it is well received by A, the `vRates` of this packet will be $R_0 b$ and $R_0 a$ at W and A respectively. If b is larger than a, W retransmits the same packet before A and the `vRates` become $R_0 b^2$ and $R_0 a^2$ at W and A respectively, and so on. Thus, if Ineq. 5 is not ensured, the first-hop nodes are inhibited by the source itself and packets will never escape from them.

3.1.1 Reaching H Hops in the absence of Collision

To reach at least H hops, the `age` of a packet after (H-1) hops should be smaller than `maxTTL` to be able to do the last (H^{th}) hop. Thus, the following inequation should be satisfied:

$$X_1 + X_2 + X_3 < \texttt{maxTTL} \tag{6}$$

where the left hand terms are as follows: X_1 is the `age` increase average of a packet during its stay in the epidemic buffer at the self-node (the node generating this packet) before being delivered for the first time to MAC for transmission. We can write:

$$X_1 \leq \sigma D_F \tau \tag{7}$$

We do not consider the delay in MAC, D_M, as the `age` increase during that delay is not transmitted to the next nodes. X_2 is the `age` increase average of a packet during its stay at foreign nodes (nodes that are not the source of this packet) of the $(H-1)$ hops. We have:

$$X_2 = (H-1)(\frac{1}{R_0 a} + D_F)\tau \tag{8}$$

The delay at MAC layer (D_M) is not considered for the same reason as above. X_3 is the `age` increase due to the hop-count component during $(H-1)$ hops. We have:

$$X_3 = (H-1)K_0 \tag{9}$$

In the optimal case and in the absence of collision, a node receives packets from its spread, S, and forwards all of them, except those belonging to the two farthest nodes in the spread (as their packets should not go beyond this node), once and only once. Thus, the number of FIFOs in this node that have eligible packets and competing to be served by the scheduler at a given time is upper bounded by $(S-2)$. Hence, we can write:

$$D_F \leq \frac{1}{R}(S-2) \tag{10}$$

with

$$S \geq 2H+1$$

Plugging Ineq. 10 in Ineq. 7 and Ineq. 8, we obtain upper bounds of X_1 and X_2. By Using Upper bounds of X_1 and X_2, and assuming that we require only H hops (that is $S = 2H+1$), we obtain an upper bound of the lower bound of a, given K_0 and K_1. In this case we have:

$$(\sigma\frac{1}{R}(2H-1) + (H-1)[\frac{1}{R_0 a} + \frac{1}{R}(2H-1)])\tau + (H-1)K_0 < \texttt{maxTTL} \tag{11}$$

3.1.2 Reaching at Least H_r hops in the presence of Collision

Consider the two successive neighbors A and B in Fig. 1. A forwards a foreign packet for the first time to B. At the beginning, the `vRate` is $R_{1,0} = R_0 a$. Recall that $\frac{1}{\texttt{vRate}}$ is the time a packet waits to be eligible after the last update of its `vRate` (due to a send/receive event on the same packet). After forwarding, the `vRate` of the same packet at A is reduced to $R_{1,1} = R_0 ab$, whereas it is again $R_{1,0}$ at B if it is well received. B will forward the packet before that A forwards it for the second time, as B has larger `vRate` . When A receives the same packet from B, it reduces its `vRate` to $R_{2,1} = R_0 a^2 b$.

In the absence of collisions, the simplest thing we can do to avoid redundancy is to select b small enough (that is $R_{1,1}$ small enough) so that the packet dies (its `age` reaches `maxTTL`) before being forwarded for the second time by A.

When the probability of collision is high, the above constraint on b does not hold anymore: It is enough that one collision occurs in one side of the source to stop propagating the packet beyond the collision place, which results in a high reduction in the spread, as the probability of collision is high even at the source level because of the hidden node, a well known problem with CSMA/CA systems.

To solve this problem, we play with both, $R_{1,1}$ and $R_{2,1}$. On one side, (1) we want $R_{1,1}$ large enough to allow A to forward again the packet if no duplicate is received from B, with the constraint that B corresponds to a hop number less or equal than the required number of hops, H_r, which is less than H. On the other side, (2) we want $R_{2,1}$ small enough so that the packet dies before being forwarded for the third time by A. In the following we develop these constraints on $R_{1,1}$ and $R_{2,1}$ separately.

3.1.2.1 Constraint on $R_{1,1}$.

In the following, we show corresponding equations assuming that the packet escapes from the source on the side we apply the equations and that the probability that two collisions occur on the same packet on the same side of a node is negligible. We write this condition as:

$$Y_1 + Y_2 + Y_3 + Y_4 < \texttt{maxTTL} \tag{12}$$

where the left hand terms are as follows: Y_1 is the `age` increase average of a packet during its stay in the epidemic buffer at the self-node until being delivered to MAC for transmission for the first time. We consider only the first transmission by a self-node. Indeed, if W is the self-node and it transmits a self-packet, this transmission faces one of three cases: (1) No collision on both sides and the packet is well received by A and L. (2) A collision occurs on one side, say L-side, then A receives well the packet and forwards it later, which consists an implicit Ack for W, which will drop the packet to allow the application to inject a new one (see Sect. 2.5). We neglect the case where the implicit Ack undergoes a collision from A to W, as this makes appear P_c^2 in the inequation (first, collision from W to L and, then, from

A to W) and P_c^2 is too small compared to P_c. (3) The self-packet undergoes a collision on both side, which happens with negligible probability. Thus, all retransmission possibilities of self-packet are negligible. Therefore, we consider only the first transmission. Similarly to X_1, we have:

$$Y_1 \leq \sigma \frac{1}{R}(S-2)\tau \quad (13)$$

The delay at MAC layer (D_M) is not considered for the same reason as above. Y_2 is the `age` increase average of a packet during its stay until being delivered to MAC for transmission the first time at foreign nodes of the $(H_r - 1)$ hops. Thus, Y_2 consists of the delay because of the `vRate` , which is $\frac{1}{R_{1,0}}$, and D_F. This latter is always upper bounded by $(S-2)\frac{1}{R}$, as the average length of FIFOs is 1. Indeed, our congestion control mechanism together with the source-based fairness of the scheduler ensure that the waiting time average of a packet in the FIFO is less than or equal to the inter-arrival time average of new packets: The application does not inject a new packet unless the previous one is implicitly acknowledged and all the FIFOs have the same scheduling share in all nodes. Applying Little formula [4], we obtain a FIFOs length average equal to 1. We can write:

$$Y_2 \leq (H_r - 1)(\frac{1}{R_0 a} + \frac{1}{R}(S-2))\tau \quad (14)$$

Y_3 is the `age` increase average of a packet during its stay after being delivered to MAC for the first transmission at foreign nodes of the $(H_r - 1)$ hops until being delivered to MAC for the second transmission, because of collision. We can write:

$$Y_3 \leq P_c(H_r - 1)(\frac{1}{R_0 ab} + \frac{1}{R}(S-2) + \frac{1}{R})\tau \quad (15)$$

Note that Y_3 considers D_M that corresponds to the first transmission. Y_4 is the `age` increase due to the hop-count component during $(H_r - 1)$ hops. We have:

$$Y_4 = K_0 + (1 + P_c)(H_r - 2)K_0 \quad (16)$$

The first right-hand term corresponds to the transmission by a self-node, it does not include P_c because we consider only the first transmission (as discussed above). Finally, by using upper bounds of Y_1, Y_2 and Y_3 and plugging Ineq. 13, Ineq. 14, Ineq. 15 and Ineq. 16 in Ineq. 12, this gives an upper bound of the lower bound of b, when other parameters are specified. Ineq. 12 becomes:

$$\begin{aligned} \sigma \frac{1}{R}(2H-1)\tau &+ \\ (H_r - 1)(\frac{1}{R_0 a} + \frac{1}{R}(2H-1))\tau &+ \\ P_c(H_r - 1)(\frac{1}{R_0 ab} + \frac{1}{R}(2H-1) + \frac{1}{R})\tau &+ \\ K_0 + (1 + P_c)(H_r - 2)K_0 &< \texttt{maxTTL} \end{aligned} \quad (17)$$

3.1.2.2 Constraint on $R_{2,1}$.

If this constraint is satisfied for the one-hop nodes, it is satisfied for farther nodes. Therefore, we consider only the one-hop nodes in the following. We write this condition as:

$$Z_1 + Z_2 + Z_3 + Z_4 + Z_5 + Z_6 > \texttt{maxTTL} \quad (18)$$

where the right-hand terms are as follows: Z_1 is the `age` increase average of a packet during its stay in the epidemic buffer at the self-node (say W in Fig. 1) until being delivered to MAC for transmission for the first time. We have:

$$Z_1 = \sigma D_F \tau \quad (19)$$

Z_2 is the `age` increase average of a packet during its stay in the one-hop node (say A in Fig. 1) until being transmitted for the first time by the same node. We have:

$$Z_2 = (\frac{1}{R_0 a} + D_F + D_M)\tau \quad (20)$$

Z_3 is the `age` increase average of a packet during its stay in the one-hop node (A) after being transmitted for the first time at the one-hop node (A) until being transmitted for the first time at the two-hop node (B in Fig. 1). We have:

$$Z_3 = Z_2 \quad (21)$$

Z_4 is the `age` increase average of a packet during its stay in the one-hop node (A) after being transmitted for the first time at the two-hop node B (and then received by the one-hop node (A) that updates its `vRate`) until being delivered to MAC for transmission for the second time at the one-hop node (A). We have:

$$Z_4 = (\frac{1}{R_0 a^2 b} + D_F)\tau \quad (22)$$

Z_5 is the `age` increase due to the hop-count component because of transmissions at self and one-hop nodes. We have:

$$Z_5 = 2K_0 \quad (23)$$

Z_6 is the `age` increase due to the hop-count component when the one-hop node (A) receives the transmission of the two-hop node (B). We have:

$$Z_6 = K_0 \quad (24)$$

Plugging Ineq. 19 till Ineq. 24 in Ineq. 18 while neglecting D_F and D_M, gives a lower bound of the upper bound of b when other parameters are specified. That gives:

$$(\frac{1}{R_0 a} + \frac{1}{R_0 a} + \frac{1}{R_0 a^2 b})\tau + 3K_0 > \texttt{maxTTL} \quad (25)$$

3.1.3 Analysis Interpretation

In this section, we interpret the three main inequations Ineq. 11, Ineq. 17 and Ineq. 25, and we show how they can be used to find appropriate ranges for the parameters. Replacing τ and R in Ineq. 11, Ineq. 17 and Ineq. 25 by their expressions in Eq. 2 and Eq. 3, we get respectively:

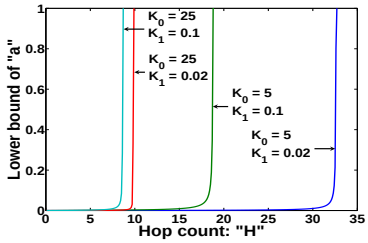
$$\begin{aligned} \sigma N K_1(2H-1) &+ \\ (H-1)[\frac{\gamma N K_1}{a(N+1)} + N K_1(2H-1)] &+ \\ (H-1)K_0 &< \texttt{maxTTL} \end{aligned} \quad (26)$$

$$\begin{aligned} \sigma N K_1(2H-1) &+ \\ (H_r - 1)[\frac{\gamma N K_1}{a(N+1)} + N K_1(2H-1)] &+ \\ P_c(H_r - 1)[\frac{\gamma N K_1}{ab(N+1)} + N K_1(2H-2)]\tau &+ \\ K_0 + (1 + P_c)(H_r - 2)K_0 &< \texttt{maxTTL} \end{aligned} \quad (27)$$

$$(\frac{2}{a} + \frac{1}{a^2 b})\frac{N}{N+1}\gamma K_1 + 3K_0 > \texttt{maxTTL} \quad (28)$$

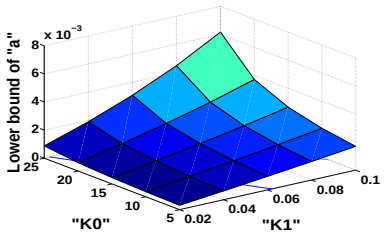
We notice that Ineq. 26, Ineq. 27 and Ineq. 28 are independent of R_0, and thus this analysis is applicable to any MAC layer with any nominal rate.

Ineq. 26 gives a lower bound of a in order to ensure H hops in the absence of collisions. This inequation should be combined with the constraint $0 \leq a \leq 1$. Fig. 2-(a) shows this bound according to H for different combinations of K_0 and K_1. The asymptotes in Fig. 2-(a) correspond to the maximum number of hops reachable for a given combination (K_0, K_1), which is a decreasing function of K_0 and



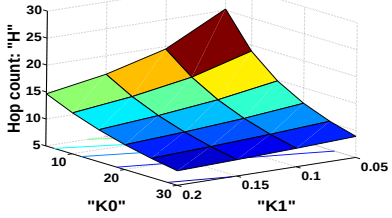
(a)

The lower bound of a is drawn according to H for different combinations (K_0, K_1). The asymtotes correspond to the maximal value of H reachable for a given combination of (K_0, K_1).



(b)

The lower bound of a is drawn as a function of K_0 and K_1. H is set to 8, which is reachable with the used ranges of K_0 and K_1.



(c)

H is drawn as a function of K_0 and K_1 for a equal to 0.1 . This value of a corresponds to a H very close to the asymtote (see Fig. 2-(a)), which is the maximal H reachable for a given combination of (K_0, K_1).

Figure 2: Relations among a, H, K_0 and K_1 based on Ineq. 26.

K_1: when K_0 increases, the maximum number of hops decreases, as it is limited by $\frac{\texttt{maxTTL}}{K_0}$, and when K_1 increases, τ increases and the packets age out faster. In Fig. 2-(b), we apply Ineq. 26 and show the lower bound of a as a function of K_0 and K_1. We set H to 8, which is reachable with the shown ranges of K_0 and K_1. The lower bound of a is an increasing function of K_0 and K_1. Indeed, with increasing K_0 and K_1, a packet ages faster and we need to increase its `vRate` in order to ensure the same number of hops.

From Fig. 2-(a), we notice that $a = 0.1$ corresponds to a H very close to the asymptote with the used ranges of K_0 and K_1. Thus, we fix a to 0.1 in the remaining of the paper. Consequently, H is tuned through the 2 parameters K_0 and K_1 and by applying Ineq. 26. This tuning is shown in Fig. 2-(c).

Once parameters a, H, K_0 and K_1 are fixed, we apply Ineq. 27 and Ineq. 28 to find lower and upper bounds of b, respectively. Fig. 3 shows that these bounds are increasing functions of K_0 and K_1. Also, as it is expected, these bounds are more sensible to K_1 than K_0. Indeed, b is related to the stay duration of a packet in a node, and it is K_1 that increases the packet `age` during this stay.

3.2 Dense Scenario

Through this analysis, we aim at tuning SAT_0. We want to strictly limit the broadcast to one-hop when the number of neighbors within the transmission range exceeds N^*. In this case, a node transmits only self-packets and the application rate in this case, λ^*, is equal to the transmission rate R^*. Thus, the FIFO belonging to self-packets is served on average each $\frac{1}{R^*}$ seconds. During this time, the `age` increase of a packet in the FIFO is $\frac{1}{R^*}\tau = N^*K_1$. Thus, we set SAT_0 to N^*K_1. If $R > R^*$, e.g $N < N^*$, SAT is incremented by $SAT_0 = N^*K_1$ each $\frac{1}{R}$ seconds, but decreased linearly during this time by $\frac{1}{R}\tau = NK_1 < N^*K_1$. Thus, SAT continues increasing on average with a slope equal to $(N^* - N)K_1R$ until it reaches `maxTTL` (see Sect. 2.3). In contrast, if $R < R^*$, e.g $N > N^*$, SAT is decremented more than incremented and thus, SAT is kept very close to SAT_0. The behavior of SAT is shown in Fig. 4 for $N > N^*$ and $N > N^*$ where it shows a very short transient phase. We require that N^* is equal to 100. Hence, we have: $SAT_0 = 100K_1$.

3.3 Default Values

In this section, we give default values to the SLEF parameters based on the above analysis. We set K_0 to 25, as we want to allow at most 10 hops. We set a to 0.1, which corresponds to a number of hops, H, very close to the maximum reachable for a given combination (K_0, K_1) (see Fig. 2-(a)). As to b, if it is too small, the second forwarding of a packet in case of collision is largely delayed. In order to avoid this delay, b should be very close to the upper bound in Fig. 3-(b). We choose the value 0.01. K_1 is set to 0.1, which is in accordance with the selected value of b (see Fig. 3). Furthermore, with $K_1 = 0.1$, the maximum epidemic buffer size needed is $\frac{\texttt{maxTTL}}{K_1} = 2550$ (see Sect. 2.6), which is a reasonable size. Finally, for $K_1 = 0.1$, SAT_0 is equal to 10, as discussed in Sect. 3.2. An application might need to change only K_0 and K_1 to adjust the spread-rate balance (see Sect. 4) while other parameters are fixed.

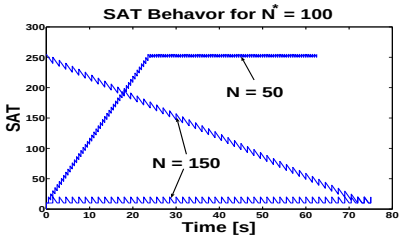
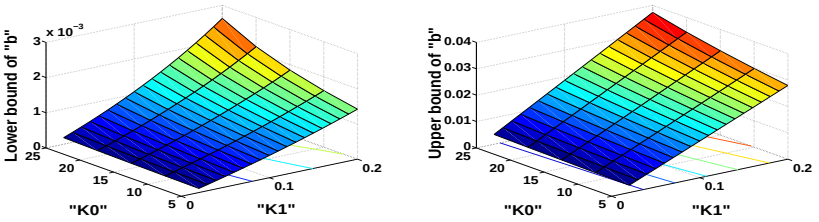


Figure 4: SAT behavior according to the density detection mechanism: $N^* = 100$, $K_1 = 0.1$ and $SAT_0 = 10$. We assume 802.11 MAC layer with a nominal rate of 1Mbps and a packet length of 1500 bytes.

4. SPREAD-RATE BALANCE

One of the main features of SLEF is to limit adaptively the spread in order to keep some balance between rate and spread (see Eq. 1). However, an application might need to move this balance in favor of one or the other. For instance, an application might have a very small rate but it requires a very large spread. In order to adjust this balance, SLEF offers to the application two degrees of freedom, which are the two main parameters of the spread control function: K_0 and K_1. These two degrees play complementary roles: One is dominant in some network settings, the other is dominant in those settings just opposite.

K_1 is related to the adaptive age component. This component is incremented during the stay of a packet in the epidemic buffer by K_1 for any receive event. The longer the packet stays before being transmitted (or forwarded), the higher its adaptive age is and the less the spread is. Thus



(a) (b)

Figure 3: Bounds on b drawn according to K_0 and K_1. These bounds are obtained through Ineq. 27 for (a) and Ineq. 28 for (b). $a = 0.1$, $H = 8$ and $P_c = 0.1$.

the effect of K_1 is dominant when this stay is long. This happens in two cases: (1) either the traffic load is very high and the number of competing packets in the epidemic buffer is very large, or/and (2) the network is dense and the number of nodes competing to access the medium is large. In both cases, a packet transmission is delayed and the packet might be dropped before being transmitted or, if transmitted, it will have a large age that does not allow it to go very far. Hence, the spread is limited by mainly the adaptive age. Consequently, playing with K_1 in these settings indeed has an impact on the spread-rate balance.

In contrast, K_0 is dominant in sparse networks, in particular with low traffic load. In this case, the spread is mainly limited by the hop-count component of the age. That is, the spread corresponds to a number of hops close to the maximum reachable ($\frac{\texttt{maxTTL}}{K_0}$).

An application that has to adjust the spread-rate balance according to its needs may proceed as follows: Specify the number of hops it needs in a sparse network and set K_0 accordingly ($K_0 \leq \frac{\texttt{maxTTL}}{number\ of\ hops\ needed}$). Then, it has to decrease or increase the default value of K_1 in order to adjust the balance in dense congested networks.

5. DESIGN VALIDATION

We validate our design through simulation. Our simulations are carried out through JIST-SWANS [2], an open source simulator for ad hoc networks. The MAC layer is a very accurate implementation of 802.11b in DCF mode with the basic rate of 1 Mbps, as we transmit in broadcast (pseudo-broadcast). As for the radio, we use the capture effect to approach the real WIFI cards, which all implement it [7]. We consider fading channels with free space path-loss.

We applied SLEF to vehicular networks. We use an extension of JIST-SWANS called STRAW [3], which simulates the vehicular traffic and provides a mobility model based on the operation of the real vehicular traffic. We simulate vehicles on an urban road with two lanes in each direction and a speed limit of $80\,Km/h$. Results for other scenarios (static and different mobility models) are omitted, as they show the same behavior.

Throughout our simulations, we adopt the default values set in Sect. 3.3, unless it is indicated elsewise. Our results focus mainly on (1) the adaptation of the spread to the rate, (2) the adaptation of the forwarding factor to the density, (3) the need of the pseudo-broadcast and (4) the spread-rate balance.

In order to cover these different aspects, we use the following metrics:

1. Rate: This is the application injection rate in packet/s. We consider a packet size of 1500 bytes.
2. Spread: This is already defined in this paper. It is the average number of nodes that receives a packet.
3. Forwarding factor: Again, this is already defined in this paper. It is the number of times a node forward a packet. We compute it as the number of duplicates circulated in the network divided by the spread.
4. Channel utilization: We use this metric only with the traffic jam (very dense network). It is approximated by the receiving rate divided by the nominal transmission rate. Note that, the receiving rate considers only successfully received packets. The channel utilization should include the successful transmission rate. We neglect this as the network is very dense and it is too small compared to the receiving rate.

5.1 Adaptation of the Spread to the Rate

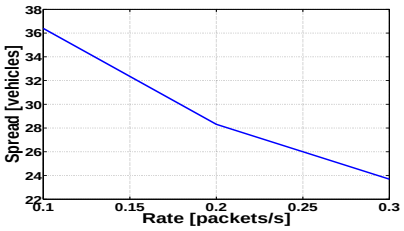


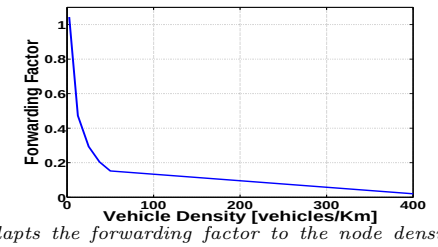
Figure 5: SLEF adapts the spread to the rate. This equivalent to an adaptive TTL. The curve corresponds to the highway scenario with a vehicle density of $12.5\,vehicles/Km$.

In this scenario, we consider a fixed vehicle density of $12.5\,vehicles/Km$. The application is not greedy. It injects packets with a fixed rate that ranges from 0.1 till $0.3\,packets/s$. This range is less than the maximal rate allowed for this scenario by the congestion control mechanism of SLEF, which is around $0.4\,packets/s$ (obtained by simulation). Fig. 5 shows the spread according to the rate. As it is expected, SLEF adapts the spread to the traffic load: it decreases the spread with increasing rate. This is equivalent to adapting the TTL.

5.2 Adaptation of the Forwarding Factor to the Density

An efficient inhibition mechanism reduces the forwarding factor with increasing node density in order to limit redundancy. This is ensured by SLEF and shown in Fig. 6, where each node runs a greedy application and thus, it transmits at the maximal rate allowed by the congestion control mechanism.

5.3 Pseudo-Broadcast



SLEF adapts the forwarding factor to the node density in order to mitigate redundancy. This is the achievement of the inhibition function. The curve corresponds to the highway scenario with a vehicles running greedy applications. Thus, the rate corresponds to the maximal allowed by the congestion control mechanism that SLEF implements.

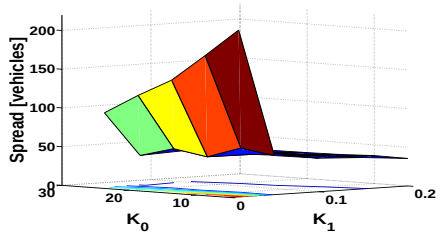
Figure 6: Forwarding factor vs vehicle density.

	Normal broadcast	Pseudo-broadcast
Channel utilization	0.02	0.7

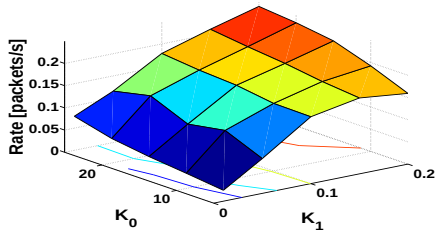
Table 1: Channel utilization in a traffic jam.

In order to show the need of pseudo-broadcast, we have chosen a very challenging scenario: a traffic jam where each vehicle has around 240 others within its transmission range and runs a greedy application. The results are shown in Table 1. The pseudo-broadcast solves very efficiently the medium access problem. It achieves a channel utilization of 0.7, whereas it is less than 0.02 with the normal MAC broadcast, which does not implement a mutual exclusion mechanism.

5.4 Spread-Rate Balance



(a)



(b)

Figure 7: Spread-rate balance: This balance is adjusted through K_0 and K_1. This is the highway scenario with vehicle density of $12.5\,vehicles/Km$. Each node runs a greedy application.

In this section, we validate what is discussed in Sect. 4 about adjusting the spread-rate balance by showing the impact of K_0 and K_1 in a sample scenario. The scenario is the highway with vehicle density of $12.5\,vehicles/Km$. Each node runs a greedy application. The results are drawn in Fig. 7. By increasing K_0 and K_1, the application increases the rate on the expense of the spread.

6. CONCLUSIONS

We propose SLEF, a complete practical middleware for multi-hop broadcast in ad hoc networks. It adapts itself to the variability of the ad hoc network environments. This includes the implementation of an adaptive TTL (through the spread control), an adaptive forwarding factor (inhibition) and congestion control. In addition, SLEF achieves buffer management, an efficient use of the MAC broadcast and source-based fairness. All these functions are achieved using only local information to the node and do not need any knowledge about the network topology. We derive simple system equations in order to tune the SLEF parameters, and we deliver default values for them. We validate our design through simulations applied on different vehicular network scenarios ranging from very sparse (DTN like) to very dense (traffic jam). SLEF shows a good adaptation and succeeds in avoiding congestion collapse, even in the extreme scenarios where other multi-hop broadcast schemes fail. Finally, SLEF offers to the application two parameters to adjust the spread-rate balance if it needs to depart from the default values.

7. REFERENCES

[1] http://icawww1.epfl.ch/haggle/slef.html.

[2] Java in simulation time / scalable wireless ad hoc network simulator , jist/swans, http://jist.ece.cornell.edu/.

[3] Street random waypoint / vehicular mobility model for network simulations , straw, http://www.aqualab.cs.northwestern.edu/projects/straw/.

[4] J.-Y. L. Boudec. Performance evaluation. http://ica1www.epfl.ch/perfeval/printMe/perf.pdf.

[5] A. El Fawal, J.-Y. Le Boudec, and K. Salamatian. Vulnerabilities in Epidemic Forwarding. In *The First IEEE WoWMoM Workshop on Autonomic and Opportunistic Communications (AOC2007)*, 2007.

[6] S. Katti, D. Katabi, W. Hu, H. Rahul, and M. Medard. The importance of being opportunistic: Practical network coding for wireless environments. In *Allerton conference*, 2005.

[7] A. Kochut, A. Vasan, A. U. Shankar, and A. Agrawala. Sniffing out the correct physical layer capture model in 802.11b, berlin, germany. In *IEEE International Conference on Network Protocols (ICNP 04)*, pages 252– 261, October 2004.

[8] S.-D. Modiano, E. and G. Zussman. Maximizing throughput in wireless networks via gossiping. In *ACM SIGMETRICS / IFIP Performance'06*, June 2006.

[9] S.-Y. Ni, Y.-C. Tseng, Y.-S. Chen, and J.-P. Sheu. The broadcast storm problem in a mobile ad hoc network. In *Mobicom, Seattle, Washington, United States, August 15 - 19, 1999*, pages 151–162.

[10] T. Spyropoulos, K. Psounis, and C. Raghavendra. Spray and focus: Efficient mobility-assisted routing for heterogeneous and correlated mobility. In *Proceedings of IEEE PERCOM, on the International Workshop on Intermittently Connected Mobile Ad hoc Networks (ICMAN)*, March 2007.