Can Multi-rate Radios reduce end-to-end delay in mesh networks? A simulation case study

Luigi Iannone Serge Fdida
LIP6/CNRS – Université Pierre et Marie Curie
Paris – France
{luigi.iannone,serge.fdida}@lip6.fr

***Abstract*— Mesh networks are based on Wireless Mesh Routers (WMRs) that interconnect themselves to form a wireless back-haul able to route traffic from/to end-users. The behavior of radio interfaces used by WMRs has a deep impact on network performance. Various recent wireless technologies are able to perform transmission at multiple different rates, depending on channel condition. How to effectively use this new multi-rate enhancement, in multi-hop mesh networks, is still an open issue.**

In this paper we present a simulation case study that highlights the impact of different rate-adaptation mechanisms, at Transport layer, in terms of throughput and end-to-end delay. In particular we compare MAC-embedded rate adaptation mechanism *vs.* rate-aware routing and how they behave in both congested and not congested networks. Results lead us to argue that using of shorter high rate links does not degrade end-to-end delay, even in congested condition, while preserving throughput capacity.

***Index Terms*— Rate Adaptation Algorithms, Cross-Layer, Mesh Networks, Rate-aware Routing.**

I. Introduction

Wireless Mesh Networks (WMNs) are an emerging technology based on a two layer architecture. The first level of the architecture consists of Wireless Mesh Routers (WMRs), which interconnect themselves in order to form a meshed wireless back-haul. The second level of the architecture consists of end-user terminals, which communicate by means of the WMRs' back-haul. Each WMR covers a region where it offers connectivity by acting as an Access Point. End-users do not need to embed any routing feature since routing is performed exclusively between WMRs.

The behavior of radio interfaces used by WMRs to communicate and to create a wireless back-haul has a deep impact on performances. Various recent wireless technologies, like IEEE 802.11 a/b/g and WiMax, are based on interfaces able to perform transmissions at multiple different rates, according to channel conditions. Data rates higher than the base rate, which is also the lowest, are possible when the signal to noise ratio (SNR) is above a certain threshold. Each transmission rate has a different threshold: the higher the rate, the higher the threshold. This is due to the modulation schemes used in higher rates, which are more sensitive to noise and interference. As a side effect, due to these different thresholds, each rate has also a different transmission range (assuming fixed transmission power): the higher the rate, the smaller the transmission range.

How to effectively use this new multi-rate enhancement in multi-hop mesh networks is still an open issue. When multi-rate capability was introduced in radio interfaces, the first approach in implementing rate adaptation algorithms was in the respect of the protocol stack layering, as defined by the OSI model. Thus, algorithms like Auto Rate Fallback (ARF) were implemented in the MAC layer, *i.e.* in the driver of the wireless cards. While such a solution may have acceptable performance in centralized architectures, like WLANs, we argue that maintaining the rate adaptation mechanism encapsulated in the MAC layer is not an optimal solution for multi-hop mesh networks.

Recently, the research community has started to look at other approaches. One of them consists in improving routing protocols to be rate-aware, *i.e.* able to find path with high transmission rate at each hop. Such a cross-layer approach may overcome some issues concerning rate adaptation mechanism. Nevertheless, with this kind of approach, another concern arises: if only high-rate links are used, the number of hops necessary to reach a destination is increased, since the transmission rate is smaller, what is the impact on the end-to-end delay? Can we dare to ask if end-to-end can be reduced by a rate-aware multi-hop solution?

In this paper we present, and explore by simulation, some mechanisms that perform rate adaptation, focusing on throughput and delay at Transport level. We limit our work to IEEE 802.11b [1] based WMNs, however, since this kind of radio interface is a consolidate standard and the most widely used, our results still have a large and general interest. Usually, papers focusing on rate adaptation algorithm explore only MAC-related issues, not worrying about upper layers. On the other hand, papers that present rate-aware routing algorithms compare them to other routing algorithm and do not care about MAC layer and rate adaptation algorithm. Here we propose a simple simulation case study, having a *cross layer eye* looking what is the effect at Transport layer (UDP/TCP) of different rate adaptation approaches, placed on different layers.

In the next section we present the basic rate adaptation solutions for 802.11, *i.e.* Auto Rate Fallback (ARF) and Receiver Based Auto Rate (RBAR). While the first has been implemented in commercial products, the second remained only a theoretic approach, which we describe in order to give a complete overview of the most important algorithms at MAC level. Then, in section III we take a look in to the real world, presenting the rate adaptation mechanism implemented in the drivers of Atheros chipset based wireless cards, which are widely used. In section IV we introduce a simple rate-aware routing approach. It is out of the scope of this paper to present a brand new rate-aware routing protocol. We just implemented the simplest solution in order to show how different its behavior is compared to MAC-embedded solutions. The results of simulations we performed are presented in section VI, putting in evidence how different solutions have opposite and interesting behaviors. In section VII we conclude the paper with some final remarks.

II. Rate Adaptation Solutions for 802.11

In the following subsections, we present the two most important rate adaptation algorithms proposed for 802.11: the ARF and the RBAR. ARF was the first published and commercially implemented rate adaptation algorithm, thus it is also the most studied. RBAR is the algorithm that the literature points as the most effective; however, it remains a theoretic approach because of the need of some modifications in the actual standard.

A. ARF

The original goal of ARF [2] was to optimize the application throughput in WaveLan II devices, which implemented the 802.11b DSSS standard. In WaveLan II there were only two possible rates: 1 and 2 Mbps. Since then, technology has evolved and 802.11b now attains 11 Mbps, while 802.11 a/g go up to 54 Mbps. Nevertheless, ARF is a general approach and can be implemented on all the different *flavors* of 802.11.

In ARF, each sender attempts to increase the transmission rate after certain number of successful transmissions, which have occurred at a given rate. If more than one failure occurs at current rate the algorithm imposes a fallback to a lower rate. At the same time a timer is set. Whether the timer expires or the number of successful transmissions reaches a threshold, the transmission rate is increased to a higher rate and the timer is reset. When the rate is increased, the first transmission is a probe, if transmission fails the rate is immediately decreased and the timer restarted.

Fig. 1 shows the state machine associated with the Auto Rate Fallback algorithm. The state machine shows how starting from the Default Rate (DR), the fallback to the Fallback Rate (FR) is triggered by the loss of one or more ACKs, depending of the actual state. Return to DR is controlled by the number of successfully received acknowledgements or by the expiration of the timer. In the Normal Operation state, the host transmits with the highest transmission rate (*e.g.* 11 Mbps in 802.11b). As long as messages are acknowledged, transmitter stays in this state. If an ACK is missed, the Retransmitting state is entered; however, transmission is still done at the same rate. If a second ACK is missed the Fallback state is entered. As already mentioned, if the associated timer expires or the correct amount of consecutive successful transmissions is reached the Probation state is entered, in order to probe a higher transmission rate. In the other case, if two consecutive ACKs are missed, while in the Fallback state, the transmission rate is lowered again. This is done by transiting through the Fallback Retransmit state.

The ARF algorithm suffers from a certain number of problems:

- In ad-hoc networks, where channel condition variability gets worse due to mobility of hosts, ARF hardly founds a stable state.
- In static condition, the algorithm will periodically try to increase the transmission rate. Since usually it is done every 10 successful transmissions; the result is the introduction of useless retransmission attempt that lowers the application level throughput.
- In Ad-hoc and Mesh networks, since different neighbors may be able to listen to the transmitter at different rates, ARF is not able to find a stable state.

The last drawback of the previous list is common in most rate adaptation algorithms. Typically those algorithms, like ARF, are designed to work well on interfaces set in *infrastructure* mode. In *infrastructure* mode all traffic goes through the Access Point (AP), thus, since the next-hop is fix (the AP), ARF and similar algorithms can find a stable state. In multi-hop mesh networks the next hop is not fixed. A node, may have two differ-

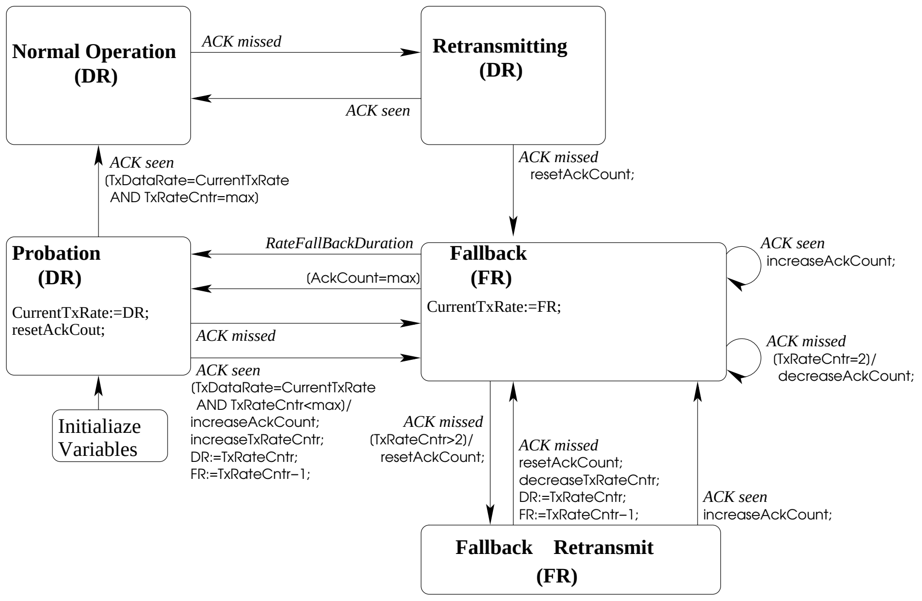
Normal Operation (DR)
ACK missed
Retransmitting (DR)
ACK seen
ACK seen (TxDataRate=CurrentTxRate AND TxRateCntr=max)
ACK missed resetAckCount;
Probation (DR)
CurrentTxRate:=DR; resetAckCout;
RateFallBackDuration
(AckCount=max)
ACK missed
Fallback (FR)
CurrentTxRate:=FR;
ACK seen increaseAckCount;
ACK missed (TxRateCntr=2)/ decreaseAckCount;
ACK seen (TxDataRate=CurrentTxRate AND TxRateCntr<max)/ increaseAckCount; increaseTxRateCntr; DR:=TxRateCntr; FR:=TxRateCntr−1;
Initialiaze Variables
ACK missed (TxRateCntr>2)/ resetAckCount;
ACK missed resetAckCount; decreaseTxRateCntr; DR:=TxRateCntr; FR:=TxRateCntr−1;
ACK seen increaseAckCount;
Fallback Retransmit (FR)

Fig. 1. Auto Rate Fallback algorithm state machine. The content of squared brackets expresses a condition. Squared brackets followed by a slash correspond to the statement *if ... then*.

ent consecutive packets for two different destinations, thus to transmit toward two different next-hops, with different rates. This prevents monolithic rate adaptation algorithms that do not consider who is the next hop from finding a stable state.

B. RBAR

The RBAR algorithm [3], like ARF, focuses on optimizing the application throughput. This algorithm requires important modifications in the actual IEEE 802.11 MAC standard. This explains why, despite the interesting performance it achieves when simulated, it has never been really implemented. The RBAR design, however, supports next-hop changes on a per-packet basis, since it is the receiver that decides the transmission rate of the data packet. This is done in the RTS/CTS (Request To Send/Clear To Send) handshake, which precedes the data packet transmission. Thus the RTS/CTS mechanism becomes mandatory.[1] The algorithm relies on the introduction of some new header fields in both control and data frames and it works in the following manner:

- The sender chooses a data rate based on a heuristic, such the most recent rate that was successful for transmission toward the destination, stores the rate and the data packet size in the outgoing RTS.
- The receiver, based on the information stored in the RTS and the channel conditions (*i.e.* the SNR), selects the appropriate rate and store it, along with the packet size, in the CTS sent back to the sender.
- Neighbors nodes that overhear the RTS/CTS handshake can update their NAV (Network Allocation Vector) to reflect the channel reservation sender/receiver have made.
- The sender transmits the data packet at the rate chosen by the receiver. The channel reservation is confirmed by adding a *Reservation SubHeader* in the data packet.

Compared to ARF, RBAR has the advantage to give very good performances, in the context of mesh networks, since it is the next hop that chooses the transmission rate. Nevertheless, RBAR presents also some disadvantages. The mandate of using RTS/CTS, also for very small packets, increases the overhead. The introduction of new header fields makes RBAR incompatible with the original standard.

III. Rate Control in Atheros chipset based cards

In the last couple of years, Atheros AR5212 [4] chipset based 802.11 wireless cards have become very popular.[2] Atheros chipset embeds very simple MAC controllers. As a consequence the driver of this kind of chipset has to implement a part of the MAC functionalities, like the rate adaptation mechanism. On Linux and BSD systems, the Atheros driver is based on a binary-only Hardware Abstraction Layer (HAL), which

[1]For further information about this kind of handshake refer to [1].

[2]This chipset is not present on Intel Centrino and Cisco Aironet solutions, which are based on proprietary hardware. Both products implement a mechanism called, by the manufacturers, *Auto Rate*, however, no details about the algorithm are available.

hides some hardware specific features, offering a simpler interface to higher levels.

The HAL allows up to 9 FIFO descriptors queues, in order to schedule packets for transmission. Each descriptor contains all the 802.11 specific information, needed to transmit the packet. In particular, there are 4 pairs of rate/counter fields ($Rate_i/Counter_i$; $i = 0..3$). When the wireless medium is available, the descriptor at the head of the FIFO queue and its relative data packet are transferred from the system memory to the device memory. The first transmission attempt is done using rate $Rate_0$. If transmission fails, retransmission occurs at the same rate, up to $Counter_0$ retries. If transmission still fails, the device tries to transmit at rate $Rate_1$ up to $Counter_1$ times, then the rate $Rate_2$ for $Counter_2$ times and finally rate $Rate_3$ for $Counter_3$ times. If transmission fails for $Counter_0 + Counter_1 + Counter_2 + Counter_3$ times in a row, the device gives up transmitting, updates the descriptor and send it back to the system memory.

Whenever a transmission is completed or abandoned, the descriptor is sent back to the system with some statistics, like the number of missed ACKs. This particular information indicates also the rate at which the packet has been sent. For example, if the transmission succeeds at the first attempt, the number of missed ACKs is 0, which means the packet has been transmitted at rate $Rate_0$. If $Counter_0 = 2$ and $Counter_1 = 2$ and the number of missed ACKs is 3, this means that the packet was sent with rate equal to $Rate_1$.

The main idea behind this approach is that short-term variations of the channel conditions can be handled by the four $Rate/Counter$ pairs, while long-term variations can be handled by changing the four pairs $Rate/Counter$. The second task is usually done by a module that regularly updates them, based on the statistics returned by the Atheros device. Two algorithms are implemented by now in the Atheros driver present on Linux and BSD systems ([5], [6]). The ONOE algorithm, whose name is due to its creator Atsushi Onoe, and the AMRR (Adaptive Multi Rate Retry) proposed by the INRIA Research Center in Sophia-Antipolis (France). In the next two subsections we briefly describe both algorithms.

A. The ONOE Algorithm

The ONOE algorithm is based on a rate controller, which runs periodically and analyzes transmission statistics for each neighbor/AP. If transmissions look to be working well over a sampling period, the rate controller gives a *raise rate credit*. Otherwise, if transmissions look to be not working well, a credit is subtracted. Once credits reach a threshold the transmit rate is raised. Various error conditions may force the transmission rate to be dropped. The decision to add or subtract a credit is based on the errors and retries accumulated over the sampling period. Usually the counters are set as follows: $Counter_0 = 4$; $Counter_1 = 2$; $Counter_2 = 2$; $Counter_3 = 2$.

Since IEEE 802.11b has only 4 rates available, the ONOE algorithm starts always with the highest rate negotiated. In 802.11 a/g instead, where more possible transmission rates are available, the algorithm usually starts *in the middle*, choosing 24Mbps or 36Mbps rate. Note that the ONOE algorithm is not far from the ARF algorithm. Enhancements consist in the customization for the Atheros chipset and in maintaining separate statistics for each neighbor.

B. The AMRR Algorithm

The main difference between the ONOE algorithm and the AMRR one is the way $Rate/Counter$ pairs are updated. AMRR sets counters as: $Counter_0 = Counter_1 = Counter_2 = Counter_3 = 1$, in order to react rapidly to short-term variations of the wireless medium. The rate $Rate_3$ is always chosen as the lowest rate available (*e.g.* 1 Mbps in 802.11b and 6 Mbps in 802.11a).

A binary exponential backoff algorithm is used in order to change the set of rates in the $Rate/Counter$ pairs. If the number of packet loss in the previous period is less than 10% and the number of different packet transmission attempts is higher than a threshold, the set of rates are increased and the threshold reset to its minimum value. If, instead, more than 33% of packets transmissions failed during the previous period, the threshold doubles. If neither of the two cases occurs the set of rates remains unchanged.

Further details about this algorithm are given [7], where authors also show how this algorithm may improve performances compared to other existing solutions.

IV. Cross-Layer Routing

Cross-layer rate-aware routing protocols are not new in wireless networks. In [8], [9] [10] there are some interesting works on the subject. The only work focusing on the effects of multi-rate in ad-hoc networks is [11]. Nevertheless, as in the other cited works, authors explore only throughput aspects. In this paper, we do not propose any new routing algorithm; we just define a simple rate-aware metric in order to implement a simple routing protocol. Our purpose is to show the effects, in terms of

TABLE I
WIRELESS CARD CHARACTERISTICS

Rate and corresponding Sensitivity	
11 Mbps	-83 dBm
5.5 Mbps	-89 dBm
2 Mbps	-91 dBm
1 Mbps	-94 dBm

both delay and throughput, of integrating rate adaptation algorithm and routing protocol.

In our simple implementation, the cost C_{ij} of a link l_{ij} between node i and node j is defined as:

$$C_{ij} = \frac{1}{R_{ij}}, \tag{1}$$

where R_{ij} is the highest rate that can be used on the link. The cost of a path is defined as:

$$C_{Path} = \max_{\forall l_{ij} \in Path} C_{ij}. \tag{2}$$

Roughly, the cost in equation 2 defines the bottleneck link of the path in terms of rate. The key point in this kind of approach is how to implement the cross-layer solution needed to give to the routing layer the awareness of rates that can be used toward each neighbor. A simple solution would be, for example in a BSD system with ONOE algorithm, to allow the routing demon to access the driver statistics tables. Each time the ONOE rate controller runs and modifies a $< Neighbor, Rate >$ pair, the event should be reported to the routing demon, which should update link costs and the routing table if necessary. We have successfully implemented this routing algorithm on NS-2, by modifying the DSDV Agent, which is the equivalent of a DSDV routing demon of a real system.

V. NS-2 ENHANCEMENT

Before starting the analysis of the results we obtained, we give here few remarks on the work done in order to perform correct simulations. The Network Simulator NS-2 [12], version 2.26, has been used in our tests. NS-2 is a widely used, open source simulator that offers a built-in 802.11 MAC layer implementation, which, however, suffers from some lacks and uncorrectness. Indeed, the radio interface model of NS-2 is still based on old WaveLAN II cards, thus lacking of correct support for rates higher than 2 Mbps. We enhanced the MAC to support multi-rate transmissions as defined in standard 802.11b. We set the interface parameters to model features of the most common wireless cards, summarized in table I. The different level of sensitivity for each transmission rate gives them a different transmission range. With these modifications, the radio interface on NS-2 is closer to real 802.11b wireless cards.

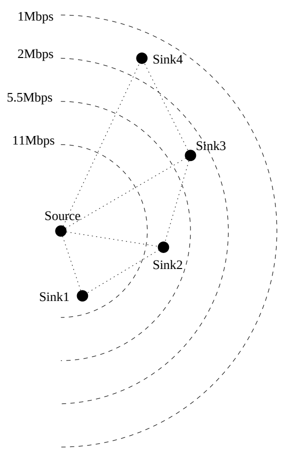

Fig. 2. Simple 5-nodes topology.

Further, NS-2 does not support natively any rate adaptation algorithm. We first implemented the ARF algorithm in the MAC layer of the simulator, following the state machine described in fig. 1. Based on the code produced to implement ARF, we developed the ONOE algorithm. The modifications made to ARF are basically to collect different statistics for each neighbor. The implementation of the routing approach needed more coding work. The rate adaptation mechanism has been included in a routing agent implemented starting from the DSDV agent, which is part of the NS-2 distribution. The resulting routing agent is rate-aware, thus it can implement the simple rate-metric presented in section IV.

VI. SIMULATION ANALYSIS

We propose here the analysis of ARF, ONOE, and rate-aware routing rate adaptation algorithms. First we performed the analysis on a simple 5-nodes scenario (see fig. 2), with heavy traffic condition. Despite of its simplicity, this topology well exploits issues related to rate adaptation in mesh networks. Then, while discussing the results obtained, we will explain them by analyzing the same algorithms, still on the 5-nodes topology, but in light traffic condition. Finally we compare ARF and the routing approach on a larger topology, in order to find some general conclusions.

A. 5-nodes Topology in heavy traffic condition

In the 5-nodes topology of fig. 2 there is one single source that generates traffic toward the other four nodes, which act as sinks. As the figure shows, sinks are placed at different distances from source in order to have

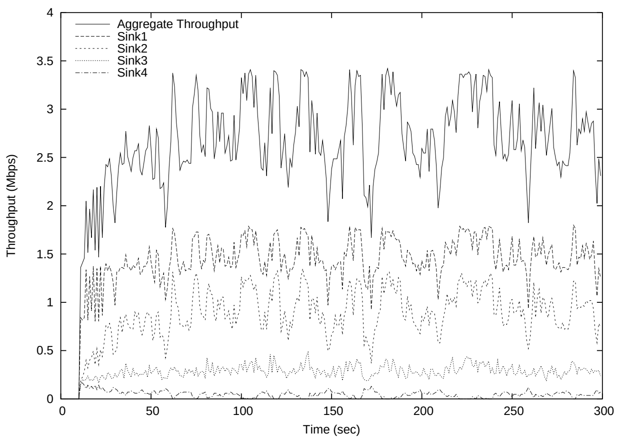

Fig. 3. ARF throughput in heavy traffic condition.

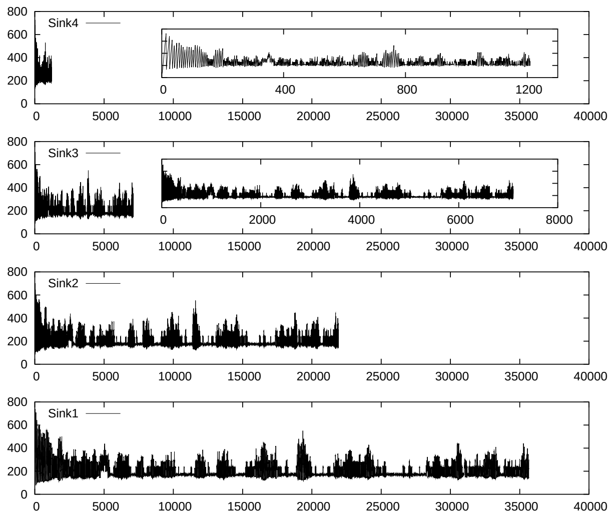

Fig. 4. Delay in heavy traffic condition, expressed in milliseconds, for each received packet when ARF is used.

different maximum transmission rates achievable. At the same time, the distance between each pair of successive sinks $< Sink_i, Sink_{i+1} >$ is tailored to have 11 Mbps maximum transmission rate on the corresponding link. Our topology is the worse multi-hop case when using only 11 Mbps links. Indeed, if for example we add a node between the $Source$ and $Sink_4$, distance would let the two nodes send traffic to each other, at 11 Mbps, through the new node, thus with only 1 hop. In our topology, instead, if the $Source$ and $Sink_4$ wish to send traffic to each other at 11 Mbps, they are forced to pass through $Sink_1$, $Sink_2$, and $Sink_3$, thus having 3 hops.

On the source node there are 4 UDP Constant Bit Rate (CBR) traffic generators, one for each possible sink. Each CBR generates packets of 1500 bytes at a constant rate. To simulate a heavy traffic condition, CBRs send a packet each 6.5 $msec$. In this way the output queue of the source node, with 50 packets size, is never empty. The measured delay in this case is shaped by the time the packet spends in the output queue. Each simulation run lasts 300 seconds of simulated time, with traffic starting after 10 seconds, in order to give the network the possibility to get stable when the routing agent is used.

1) ARF Simulation: The behavior of ARF, in terms of throughput, in the case of heavy loaded network, is depicted in fig. 3. The throughput is sampled at regular intervals of one second, taking in to account only packets correctly received by the intended sink. No routing agent is used during simulation of ARF, thus the $Source$ tries to talk directly to each sink. As can be remarked, the throughput is highly variable. The cause of this behavior is the ARF that does not separate statistics based on the next-hop. If, for instance, the source sends a burst of successfully delivered packets to $Sink_1$, its ARF may decide that the best rate to use to send packets is 11 Mbps. If the next packet must be delivered to $Sink_4$, the MAC layer will waste time in several retransmissions before reaching the transmission rate of 1 Mbps that is the only one $Sink_4$ is able to correctly decode. In other words, since the intended next-hop may change for different packets, the ARF is not able to get stable. The instability of the ARF in mesh networks can be also observed in fig. 4. In this figure the y-axis represents the delays, in milliseconds, of each successive packet correctly received (x-axis). The figure is composed by four graphs, one for each sink. When necessary a zoom picture has been added, to better show the behavior of the related sink. The delay has big fluctuations for all sinks; up to 600 $msec$. We can also observe the difference of received packets, due to the different transmission rate. While $Sink_4$ receives only around 1200 packets, $Sink_1$ receives more than 35000.

2) ONOE Simulation: The ONOE algorithm has been simulated in the same conditions as ARF. In this case, no routing agent is necessary either, since source will send packets directly to each intended sink. Collecting statistics for each neighbor eliminates the major drawback of ARF. Transmission rate is now triggered on the intended next-hop. Even if the next-hop changes on a per-packet basis, this approach avoids a lot of retransmissions improving stability, as fig. 5 shows. We can remark that throughput shape is more stable for all sinks. The aggregate throughput, *i.e.* the sum of the throughput toward each sink, is also improved, with an average above 3.5 Mbps, while ARF rarely achieved a throughput of 3.3 Mbps. The same stability behavior can be observed for the delay in fig. 6. This time, the average delay is always lower than 200 $msec$. There

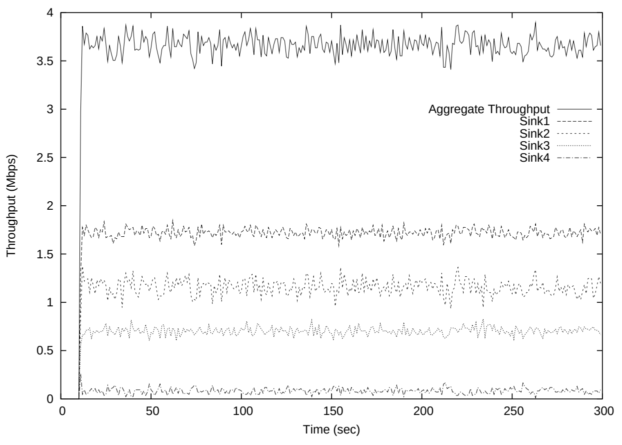

Fig. 5. ONOE throughput in heavy traffic condition.

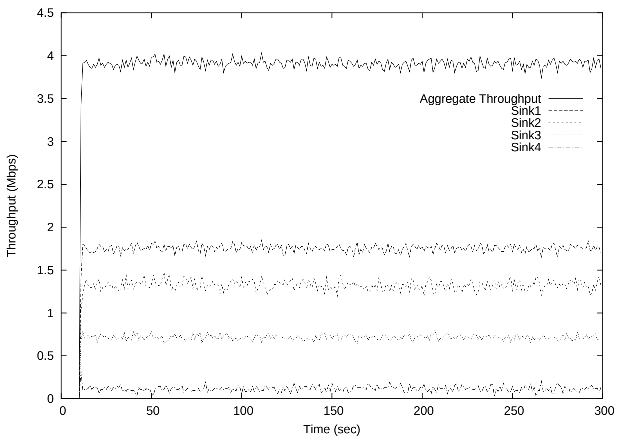

Fig. 7. Routing approach throughput in heavy traffic condition.

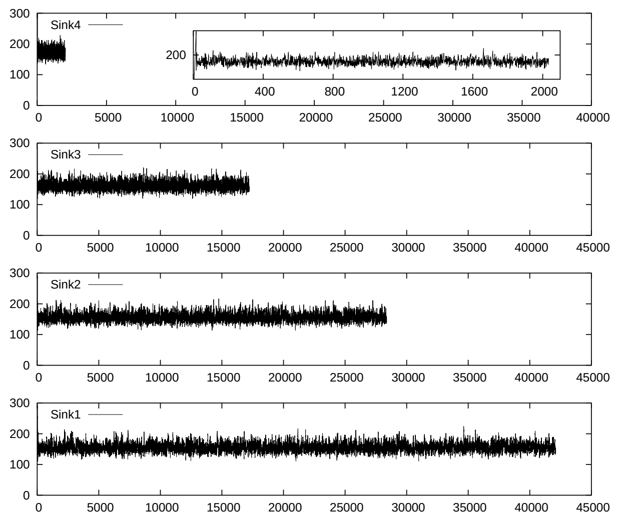

Fig. 6. Delay in heavy traffic condition, expressed in milliseconds, for each received packet when ONOE is used.

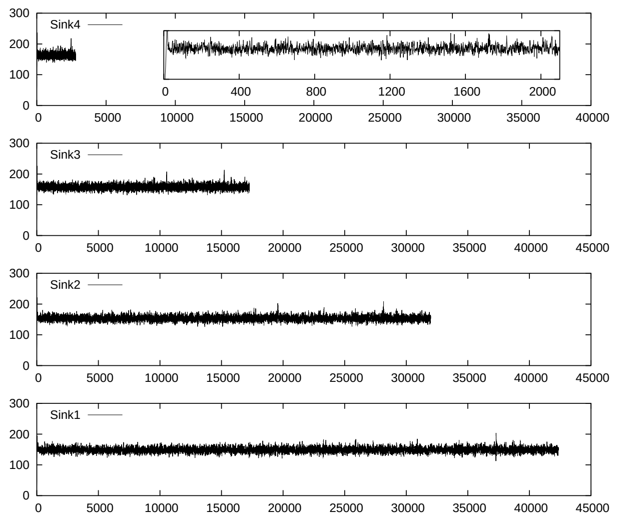

Fig. 8. Delay in heavy traffic condition, expressed in milliseconds, for each received packet when the routing approach is used.

is also an increase in the number of packets correctly received, almost double for $Sink_4$, and around 42000 for $Sink_1$. Once again the different transmission time for the different rates plays a key role on the number of packets that can be delivered correctly in a fixed time window (300 sec simulated time in our case).

3) Routing Approach Simulation: Finally, let us take a look to the results of the rate-aware routing approach. As already stated, we placed sinks in such a way to have 11 Mbps links between them. Our routing agent correctly finds highest rate paths, from $Source$ to each sink, which are:

$\mathbf{Sink_1}$ $Source{\rightarrow}Sink_1$;

$\mathbf{Sink_2}$ $Source{\rightarrow}Sink_1{\rightarrow}Sink_2$;

$\mathbf{Sink_3}$ $Source{\rightarrow}Sink_1{\rightarrow}Sink_2{\rightarrow}Sink_3$;

$\mathbf{Sink_4}$ $Source{\rightarrow}Sink_1{\rightarrow}Sink_2{\rightarrow}Sink_3{\rightarrow}Sink_4$.

Since transmission statistics are collected on a per-neighbor basis, the routing solution presents the same stability behavior as ONOE. All transmission are now performed at 11 Mbps, thus the aggregate throughput is improved, reaching almost 4 Mbps, as shown in fig. 7. Of course, the number of collisions is increased, since performing multi-hop means that more than one node wish to access the channel. Nevertheless, this drawback is largely overcome by the gain in terms of reduced transmission time, which in turns increases global throughput. It is interesting to remark the results concerning the delay, showed in fig. 8. The number of delivered packets is increased compared to both ARF and ONOE, even if there is not a large gain versus ONOE. The delay variation is reduced; indeed the plot is thinner, while the average is the same as for ONOE. The last point seems to go against the common intuition that since the routing approach increases the number of hops, the end-to-end delay is increased. The main reason for this behavior is that the latency of packets in the output queue of

TABLE II
NUMBER OF DELIVERED PACKETS IN HEAVY TRAFFIC CONDITION

Sink	ARFB	ONOE	Routing
$Sink_1$	35672	42114	42362
$Sink_2$	21932	28381	31991
$Sink_3$	7106	17217	17276
$Sink_4$	1210	2034	2803

the $Source$ is far higher than the time needed to reach the destination and this latency shapes the delay. Further details are given in the next section.

B. 5-nodes Topology in light traffic condition

What comes out from the simulation study we performed in the previous section is that enhancing the routing layer by rate-awareness seems to give a gain in terms of throughput. The reason is that the routing approach is able to deliver a higher quantity of packets compared to MAC layer encapsulated approaches, as shown in table II. The routing approach exceeds (slightly) ONOE which in turns exceeds ARF. Compared to ARF, ONOE is able to deliver more packets, in the time window we simulated, because it collects statistics on a per-neighbor basis. This saves retransmissions, needed to adapt the rate, which instead occurs in ARF when the next-hop of two successive packets changes. The routing approach increases the number of delivered packets, compared to ONOE, because all transmissions are shorter, since only 11 Mbps links, through multi-hop, are used. In heavy traffic condition, ONOE greatly stabilizes performance, and slightly reduces the average delay, compared to ARF. This result is due, again, to the reduced number of retransmissions that occur when ONOE is used instead of ARF. The routing approach, compared to ONOE, reduces even more the delay fluctuations. The reason of these achievements is that, with the routing approach, all packets that are in the output queue have the same transmission time. What remains variable is the channel access time, which depends on the random backoff mechanism of the MAC layer.

Based on the above considerations, we performed simulation in light traffic condition in order to put in evidence the transmission time of the packets in each approach. In this the second type of simulation, CBRs generate a packet every 250 $msec$ (*i.e.* 4 packets/second/CBR). Opposite to the first case, this time the output queue is always empty when a packet is generated. The measured delay in this case is shaped by the transmission time and the latency of the packet when traversing the layers of the protocol stack at each hop. The results are shown in fig. 9, where there is the

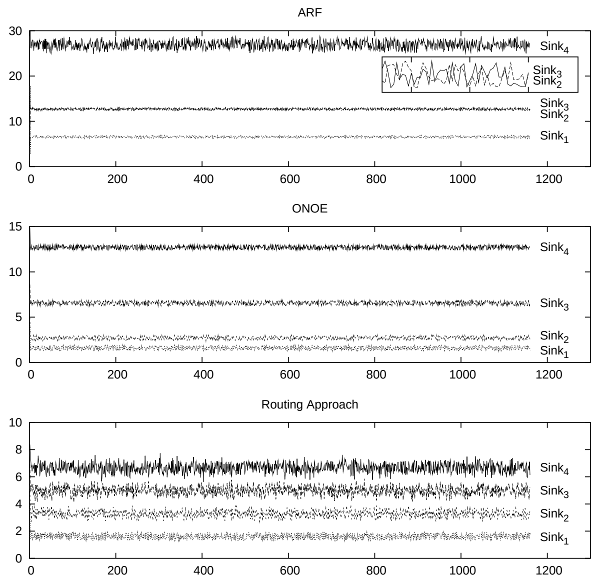

Fig. 9. Delay in light traffi c condition, expressed in milliseconds, for each received packet, for all proposed solutions.

plot of the end-to-end delay of all delivered packet, for all the proposed approaches. Plotting the throughput is useless because the traffic produced by the four CBRs is far lower than the network throughput capacity.

Sending a packet to $Sink_4$ in the case of ARF, the end-to-end delay is more than three times the end-to-end delay when using the routing approach. The high latency of ARF is justified by the fact that some retransmissions occur before ARF finds the correct transmission rate. Moreover, these retransmissions increase the contention window of the backoff mechanism, deteriorating even more the performances. Note that the fact that in fig. 9 the plot of $Sink_2$ and $Sink_3$ almost totally overlap is due to a synchronization phenomenon occurring with CBR traffic generators. A zoom has been added to the picture in order to show the difference. The reason of this behavior is that in light traffic conditions packets are always sent in the same cyclic sequence. Meaning that, also the transitions in the ARF state machine are cyclic. The total time spent in retransmissions to find the right rate and the time spent in the successful transmission is the same for both sinks.

ONOE reduces some part of latency due to retransmissions, maintaining also a smaller contention window. Yet, slow transmissions do not allow ONOE to achieve low delay like in the routing case. The delay when sending packets toward $Sink_4$ remains higher than 12 $msec$. For $Sink_1$ and $Sink_2$, the delay is the same as for the routing approach. For $Sink_1$ the explanation is trivial, the delay is due only to channel access time and transmission time, which in both cases is the same. For $Sink_2$, the reason of having the same delay is that ONOE transmits at 5.5 Mbps, while the routing approach transmits two times at 11 Mbps, in order to reach $Sink_2$. This makes an average transmission rate of 5.5 Mbps with some overhead added due to the forwarding operation.

The rate-aware routing approach presents the most linear behavior. Due to the absence of other traffic, the end-to-end delay is just the sum of the time need to reach each hop, no queuing time is present. In particular, to reach $Sink_4$ the delay results lower then the delay experienced when transmitting directly at 1 Mbps as in the case of ONOE.

C. Random Topology

Insofar, we analyzed the relation between rate adaptation mechanism and end-to-end delay in a simple 5 nodes topology. Results suggest that performing multi-hop, by using a rate-aware routing approach, may reduce, or at least not deteriorate the performances. What seems to come out is that rate-aware routing is the best solution in light traffic condition, while in heavy traffic condition ONOE and rate-aware routing are equivalent, while ARF performs the worst in all cases. Due to the simple topology we used, this conclusion cannot be assumed as a general behavior of multi-rate mesh networks.

To go deeper in the analysis, looking at what happens in a more general topology, with different traffic loads, we simulated a random topology of 52 nodes in an area of 2000x2000 meters. We fixed our attention on one single flow; we call this the *sample flow*. Besides, other flows are also generated, for different source/destination pairs and the traffic injected by each flow is also varied. The goal is to better highlight what happens to a flow, in terms of delay, when congestion of the network (*i.e.* flow numbers, traffic load per flow) changes. Flows consist of UDP packets generated by a CBR source, all with the same size of 1500 bytes. We performed several simulation runs, changing at each run the packet interval time, in order to change the traffic load. The smaller the packet interval time, the higher the traffic load. We first observed the sample flow alone and then we added more flows. Each time a flow is added, we measured the delay for the whole set of packet interval time. Fig. 10 and fig. 11 show the delay of the sample flow for, respectively, ARF and rate-aware routing. In the case of ARF, we used AODV as a routing protocol.

For ARF, the sample flow has a low (less than 100 $msec$) and stable delay when it is alone, or with only one more flow. When more than one flow are added, the delay becomes more variable and when the packet interval time falls under 0.1 $msec$, it explodes to very high

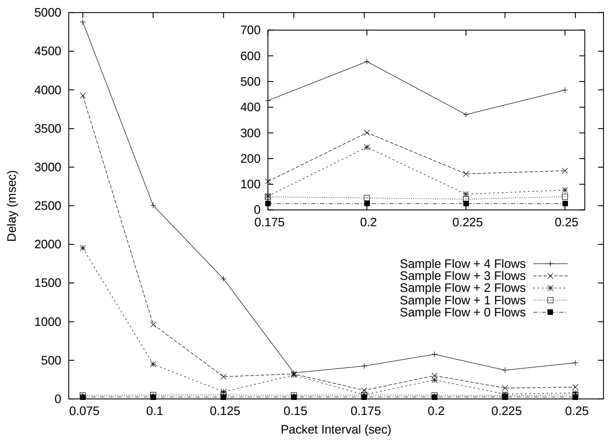

Fig. 10. Average delay when ARF is used; AODV is the routing protocol. To better show the differences when different flows are added in the network, the figure presents a zoom of the curves when the traffic is not heavy.

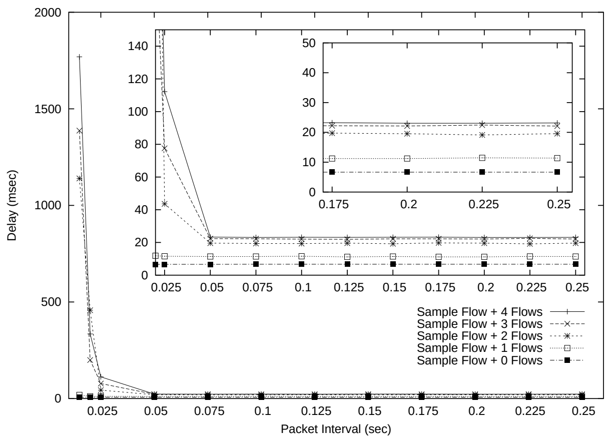

Fig. 11. Average delay when rate-aware routing approach is used. Since the behavior of the delay is very stable, we put a double zoom in the figure: one to highlight the *saturation points* and one to highlight the difference between delays when other flows are in the network.

values. We say that for each curve there is a *saturation point*. The lower the number of flows present in the network, the lower the packet interval time that *saturate* the network. Note, that also in the case of a single flow beside the sample flow, or the sample flow alone, there is a saturation point, but in both cases it falls out of the range of packet interval time showed in fig. 10.

Saturation occurs because the number of packets generated becomes far higher then the capacity of the network. Thus output queues fill up, while the MAC layer spends a lot of time in retransmissions due to collisions and contention of the channel. Since we used UDP traffic coupled with drop tail output queues, below the saturation point the delay may reach values of several seconds. In the case of rate-aware routing approach, the delay has lower values, up to 20 times lower than ARF, and a more stable behavior. The saturation points, in this case, have values lower than 0.05 $msec$ when 2 to 4 flows are present in the network. The other cases have a saturation point that is out of the range of fig. 11.

The reason of this large gain of the rate aware approach can be found in [13], where Lundgren *et al.* first observed the phenomena of *gray-zones*. Nodes in a gray-zone can be sensed by `HELLO` messages, which are sent at low rates, but cannot reliably exchange data traffic at higher rates. An example is $Sink_4$ in the 5 nodes topology. As the density of the network increases, normal multi-hop routing protocols, which are based on the shortest path metric in a hop sense, are more likely to choose nodes that are in a gray-zone. This is exactly what happens in our random topology, where, in order to relay packets, AODV chooses more easily nodes that are in a gray-zone. ARF and shortest hop path routing protocol have a very bad interaction, thus high delay values also when the network is under relatively light traffic condition. The rate-aware approach instead, selectively chose next-hops that are not in a gray-zone, thus able to exchange data at high rates. Due to space constrains, we do not show results for the ONOE algorithm. Nevertheless, ONOE shows only a more stable behavior, compared to ARF, since the gray-zones phenomenon limits its performances as well.

VII. Conclusion

We presented in this paper a simulation case study concerning the improvements that a rate-aware routing protocol may achieve in terms of throughput and end-to-end delay in wireless mesh networks. Our case study first focused on a simple 5-nodes topology, in order to well understand the behavior of the ARF and the ONOE rate adaptation algorithms and a simple rate-aware routing approach. While the first two solutions are placed at MAC layer, the rate-aware routing solution is a cross-layer approach, since low level statistics are brought to the routing layer in order to make path construction. This first analysis has shown that using a rate-aware routing approach, may reduce, or at least not deteriorate the end-to-end delay, while giving small throughput gain.

Then we moved to a more general topology, fixing our observation on a single flow while varying the traffic condition of the network. Once again the rate-aware routing approach outperforms the other approaches. The bad performance of ARF and ONOE are found in the high latency to find right transmission rate at MAC level and the presence of the gray-zones phenomena. This is overcome by coupling rate adaptation and routing level, avoiding gray-zones and maintaining limited rate adaptation latency.

Returning to the question at the beginning of this paper: Can we reduce end-to-end delay by a rate-aware multi-hop solution? The answer is: probably.

Our simulations are not totally real, since NS-2 has a poor physical channel model, and reduced interference model. While in the case of IEEE 802.11 a/g multi-rate radio interface we can suppose to observe a similar behavior, with other type of interfaces things may go very differently. Further, by now we did not yet simulate TCP traffic. Other studies have shown that in mesh networks, TCP traffic behaves very differently from UDP traffic. Thus assuming the results shown in this work as also valid for TCP traffic would be a hazard.

Despite some limitations, our simulations offer an interesting insight in the behavior of multi-rate radio interfaces in the context of mesh networks. Moreover, they show that encapsulation and layer isolation may not be the best solution in mesh networks, while a cross-layer solution may give good improvements.

References

[1] IEEE, “Wireless lan medium access control (mac) and physical layer (phy) specifi cations,” *IEEE Standard 802.11*, June 1999.

[2] A. Kamerman and L. Monteban, “Wavelan ii: A high-performance wireless lan for the unlicensed band,” *Bell Labs Technical Journal*, 1997.

[3] G. Holland, N. Vaidya, and P. Bahl, “A rate-adaptive mac protocol for multi-hop wireless networks,” *Proceedings of ACM MOBICOM’01*, July 2001.

[4] Atheros Communication, “Atheros wireless lan 2.4/5-ghz 802.11a/b/g 108 turbo radio-on-a-chip wlan networking products and technology overview,” July 2004. [Online]. Available: http://www.atheros.com/pt/index.html

[5] Madwifi Project Information. [Online]. Available: http://sourceforge.net/projects/madwifi /

[6] FreeBSD: The Power to Serve. [Online]. Available: http://www.freebsd.org

[7] M. Lacage, M.H.Manshaei, and T. Turletti, “Ieee 802.11 rate adaptation: A practical approach,” *INRIA Research Report number 5208*, May 2004. [Online]. Available: http://www.inria.fr/rrrt/rr-5208.html

[8] S. Zhao, Z. Wu, A. Acharya, and D. Raychaudhuri, “Parma: A phy/mac aware routing metric for ad-hoc wireless networks with multi-rate radios,” *IBM Research Report*, Dec. 2004.

[9] Y. Seok, J. Park, and Y. Choi, “Multi-rate aware routing protocol for mobile ad hoc networks,” *Proceedings of IEEE VTC2003-spring*, Apr. 2003.

[10] S. Sheu, Y. Tsai, and J. Chen, “mr^2rp: The multi-rate and multi-range routing protocol for ieee 802.11 ad hoc wireless networks,” *Wireless Networks*, May 2003.

[11] B. Awerbuch, D. Holmer, and H. Rubens, “Effects of multi-rate in ad hoc wireless networks,” *John Hopkins University, Technical Report*, 2004.

[12] The Network Simulator NS-2. [Online]. Available: http://www.isi.edu/nsnam/ns/

[13] H. Lundgren, E. Nordström, and C. Tschudin, “Coping with communication gray zones in ieee 802.11b based ad hoc networks,” *Proceedings of 5th ACM International Workshop on Wireless Mobile Multimedia (WoWMoM’02)*, Sept. 2002.