Efficient Cooperative Relaying in Wireless Multi-Hop Networks with Commodity WiFi Hardware

Tobias Volkhausen, Kornelius Dridger, Hermann Lichte, Holger Karl

Efficient Cooperative Relaying in Wireless Multi-Hop Networks with Commodity WiFi Hardware

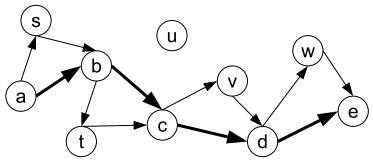
Tobias Volkhausen, Kornelius Dridger, Hermann S. Lichte, Holger Karl
University of Paderborn, Germany
{tobias.volkhausen|kdridger|hermann.lichte|holger.karl}@upb.de

***Abstract*—In wireless multi-hop networks, cooperative relaying exploits temporal and spatial diversity by additionally transmitting via a relay node and then combining direct and relay transmissions at the receiver. While such relaying improves packet error rates, it also costs an extra transmission. Along multi-hop paths, there often are nodes that can be a relay for multiple consecutive transmissions. This allows such a relay to transmit only once rather than on each individual hop along the routing path. We show by simulations that cooperative relaying is more efficient in multi-hop networks than in isolated three-node configurations. We compare our simulation to experimental results by implementing our multi-hop cooperation protocol on commodity Wi-Fi hardware. The appeal of using Wi-Fi is its wide availability. However, today's devices were not designed with cooperation protocols in mind. We explain the necessary steps to implement wireless cooperation protocols, using our multi-hop cooperation protocol as an example.**

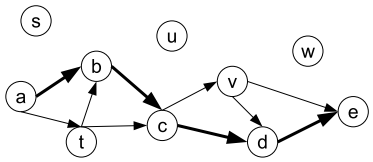
I. Introduction

In scenarios with slow fading channels (e.g., indoor office environments), link-by-link retransmissions may be inefficient as the channel might not improve during the retransmission. One solution is to exploit cooperation diversity and retransmit from another node in the source's vicinity. This node provides an independent channel to the destination, which can be used for the second transmission. Finally, the two transmissions are combined at the destination, providing a diversity gain and overall reduced error rate. Such use of a relay and message combining is also known as *cooperative relaying*.

For a multi-hop transmission, cooperative relaying with a suitable relay can be applied for each hop. Fig. 1(a) shows this idea. Along the path from source a via the intermediate nodes b, c, and d to the final destination e, the relays s, t, v, and w establish a Cooperative Triangle (CTR) for every hop. The relay selection in Fig. 1(a) is only one option. There are better relay choices along the path from a to e. Notice how node t is not only in the vicinity of a and b but also in the vicinity of the receiver c of the subsequent hop. If the Medium Access Control (MAC) protocol had chosen node t (see Fig. 1(b)) for assisting the transmission $a \rightarrow b$, its retransmission would have also benefited the following hop $b \rightarrow c$. Thus, the *single* retransmission of redundancy can improve the success probabilities of *two* ordinary transmissions. So instead of only looking at cooperation triangles, it is beneficial to consider larger node configurations. Two triangles next to each other, joined along a common side, look like a diamond. Therefore, the configurations *a-b-c-t* and *c-v-d-e* are called diamond configurations. The diamond configuration frequently occurs in multi-hop networks (see [1] for occurrence probability of diamonds for randomly placed nodes). The protocol in Fig. 1(b), which takes advantage of a shared relay, is called *two-for-one* cooperation in the diamond configuration.



(a) No routing information available: hop-wise cooperative relaying over *many individual* cooperative triangles (CTR).



(b) Routing-enhanced cooperative relaying: *Fewer transmissions* with same effect.

Fig. 1. Cooperative relaying in wireless multi-hop networks mitigates fading errors and improves message delivery rates by repeating messages at relays. The number of repeated messages can be decreased with knowledge of MAC information.

It has been shown through simulation that two-for-one cooperation is an improvement over conventional relaying [2]. However, it is still unclear whether it can be efficiently implemented on today's commodity hardware or if it requires completely new radio hardware. Experiments with Software-Defined Radios (SDRs) have shown that sophisticated combining algorithms like Maximal Ratio Combining (MRC) are not necessary in indoor environments where the wireless channel typically does not change significantly during a single packet transmission [3]; indeed Packet Selection Combining (PSC) gives comparable results. This is encouraging, since we would not have been able to do sophisticated, MRC-like, combining on commodity hardware, because low level physical layer functions and signal processing cannot be modified. Instead, it is sufficient to only modify the driver software.

978-1-61284-824-2/2012 - Copyright is with IFIP

We use Wi-Fi devices and an open-source driver. The driver modifications are not always straight forward and there are some challenges to conform to IEEE 802.11 specifications. There are several questions that need answers. What header fields can be used to store extra addresses, such as the relay address? How to disable automatic sequence number assignment, so that relay and source packet have the same number? How to avoid retransmission from the source and instead wait for the relay to send its data? How to send an ACK to the source, even though the relay sent the correct data. How to receive packets from source and relay alike and why promiscuous mode is not a good idea. How to make all these extra transmissions transparent for the higher application layer? We explain in detail how to implement general wireless cooperation and two-for-one cooperation.

The rest of the paper is organized as follows. Sec. II summarizes related work. Sec. III explains the two-for-one cooperation protocol for multi-hop networks and shows performance bounds by simulation. Sec. IV describes how to implement two-for-one cooperation with the readily available *ath5k/mac80211* device drivers. Sec. V describes an experimental setup for performance measurements and studies the data rate at the receiver with slow fading. Simulation results for the same traffic model complement our experimental results. Finally, Sec. VI discusses the implications of our findings.

II. Related work

Opportunistic relaying [4] and opportunistic routing [5] are similar approaches that also combat fading effects. At the MAC sub-layer, opportunistic relaying chooses the best relay before it transmits. Both opportunistic approaches require precise Channel State Information (CSI) to decide at the transmitters [6].

Our approach does not require precise CSI at the transmitters since it does not try to choose the best relay beforehand. Instead, it provides the destination with two copies for combining. Although our approach does not require CSI at the transmitters, a relay must know whether the receivers of the first and second hop are its neighbors. Unlike precise CSI, this information is easier to obtain and needs to be less frequently updated.

Opportunistic routing moves this *a priori* selection of the best relay to the routing layer [5]. It does not specify a particular next hop as the receiver. Instead, it exploits multi-user diversity by retransmitting from the node closest to the destination that successfully received the initial message. Again, this requires CSI to determine the best node. Unlike opportunistic routing, our approach can be implemented (see Sec. V) solely at the physical layer and the MAC sub-layer as long as the routing protocol provides information about the first and second hop receivers. In this case, the routing protocol does not need to be replaced and two-for-one cooperation can be implemented more efficiently.

Wireless cooperation has been implemented on different commodity hardware in [7]. While the results also show that it is possible, the hardware is very different from ours. They use sensor node sized SDR and explain that cross layer considerations are necessary, while having full access to even the lower layers. On the other hand, our work is done on Wi-Fi hardware, which is commonly used in many devices today. At the same time, WiFi hardware prevents us from modifying lower layers, which are typically implemented on chip.

Implementing cooperative relaying by modifying an open-source Linux wireless driver has been done by Korakis et al. [8]. Their proposed CoopMAC protocol exploits the multi-rate capability of the IEEE 802.11 standard. CoopMAC prefers a relay transmission over direct transmission if the achievable data rate via the alternative path using the relay is higher than direct transmission. Their work shows that it is possible to implement cooperative relaying on cheap commodity hardware in principle. However, it does not answer the question what gains can be achieved in wireless *multi-hop* networks.

In previous preliminary work [9] we investigated if implementing cooperation on commodity Wi-Fi hardware was at all possible. Due to encouraging first results, we set out to investigate further in our current work.

III. Simulation

For the simulation, we assume the nodes are all known, the transmission time slots are fixed, and ideal signaling, meaning that it is always error-free and does not cause overhead. These abstractions make the simulations more manageable, but still provide fairly realistic results. This can be seen in Sec. V, where we compare the simulation to real hardware results.

In order to capture the full benefit of wireless cooperation, such as robustness against rapidly changing channels, we decide to model our data transmission on a fairly low level. Whenever possible, we use parameters from the IEEE 802.11a standard [10].

The source node generates 1024 byte long packets from random bits and encodes them using a rate $1/2$ convolutional code. The coder output is punctured and modulated using BPSK, 16-QAM, or 64-QAM, according to desired data rate. The resulting modulation symbols are transmitted over a Rayleigh-fading channel with symbol-wise changing channel coefficients. For the autocorrelation of the fading channel, we choose the "Jakes-like" method with the "land mobile" autocorrelation function (Table 2.1 in [11]). The fading channel is parametrized by a maximum Doppler shift based on the maximum velocity of 1 m/s and a symbol time of 4 μs. This widely used model is suitable for mobile indoor and urban scenarios with many stationary, uniformly distributed scatterers.

The receiver adds Additive White Gaussian Noise (AWGN), parametrized by a given mean Signal-to-Noise Ratio (SNR). The received faded and noisy signal is equalized, using perfect CSI, and demodulated. The resulting bits are depunctured and decoded, using the Viterbi algorithm. Finally, correctness of the decoded message is verified using Cyclic Redundancy Check (CRC). The receivers use PSC to combine messages received over separate channels.

We compare the following protocols:

Direct transmission – At sufficiently high SNR it always achieves the highest goodput per link. It is therefore an important comparison.

Non-Cooperative Relaying (NCR) – Relaying without combining. Two small hops require less energy than one big one (non-linear path loss). Additionally, the packet is regenerated at the relay, which may reduce errors. The implementation, using existing MAC protocols, is straightforward.

Cooperative Triangle (CTR) – Relaying with combining at the relay and destination. In absence of routing information, a MAC protocol can only apply the CTR twice, namely first for the transmission $a \rightarrow b$ with relay c and second for the transmission $b \rightarrow d$, once again using relay c.

Weak Full Diamond (WFD) – In presence of routing information, the relay c can efficiently cooperate by transmitting only *once* and exploiting its transmission at both b and d. One less transmission.

Fig. 2 shows how much a higher diversity order improves the Packet Error Rate (PER) for increasing SNR in the diamond configuration. The superior performance of CTR

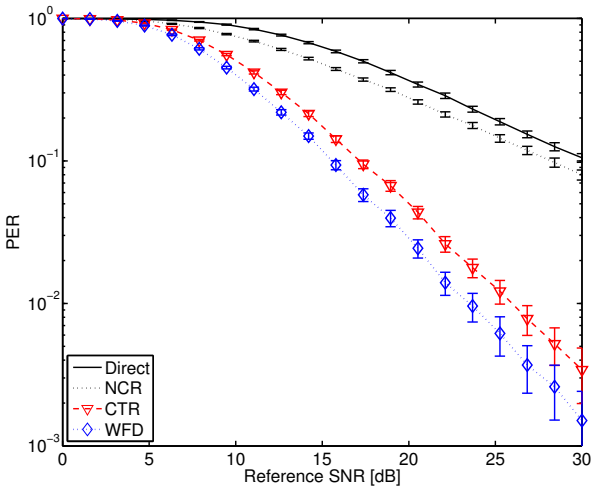


Fig. 2. Packet error rate of direct transmission, non-cooperative relaying, and cooperative relaying without and with routing information in presence of Rayleigh fading; all approaches use ideal signaling.

and WFD results from a higher diversity order. Notice the difference between CTR and WFD. At first glance these two protocols should show similar PER performance and merely differ in the number of transmissions they require. However, there is a subtle difference. In the *a-b-c-t* diamond of Fig. 1(b), if relay t does not receive correctly in the first phase, both protocols do the same. If t receives successfully, WFD will always be able to cooperate in the second phase and combine messages from t and b. In CTR, on the other hand, the additional $b \rightarrow t$ has to be successful first, in order for c to receive from both t and b. Fig. 3 shows the data rate at the receiver over SNR. At high SNR, the extra transmissions for cooperation do not pay off. With enough transmission power available, nodes should always prefer direct transmission if

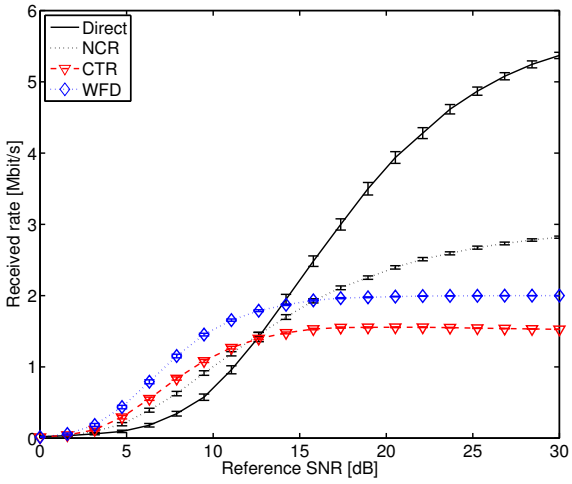


Fig. 3. Received rate of direct transmission, non-cooperative relaying, and cooperative relaying without and with routing information in presence of Rayleigh fading; all approaches use ideal signaling.

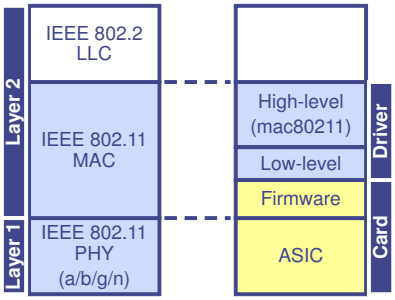


Fig. 4. Architecture of Linux wireless driver

they demand high received data rate. In practice, however, resources are often limited. For example, to deploy a limited number of nodes over an area as large as possible without losing connectivity, inter-node distances need to be as large as possible and hence the network operates at low SNR. In this case, cooperative relaying achieves higher received data rates than direct transmission or NCR. Among the cooperation protocols, WFD achieves higher throughput than CTR, because it requires one less transmission for similar or better PER. To obtain this improvement, the MAC protocol only needs access to routing information.

IV. Integrating Two-for-one-cooperation into the ath5k/mac80211 driver

Fig. 4 shows the architecture of the *ath5k* Linux wireless driver that we modified. The *ath5k* consists of a hardware-specific driver that is tailored to the specific manufacturer's chipset, e.g., the Atheros AR2414 used for our experiments, and a hardware-independent driver that implements most of the IEEE 802.11 MAC automaton and management, which works for all chipsets. Since most parts of the MAC protocol are implemented in the *mac80211* driver, the protocol can easily be modified to address an additional relay node and

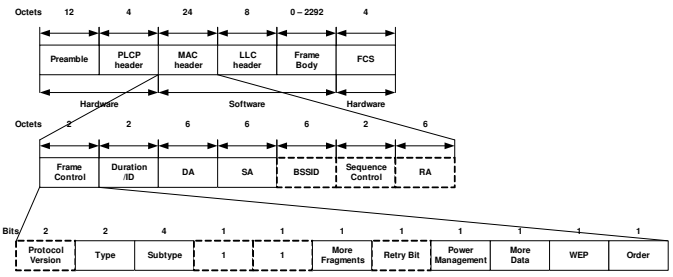


Fig. 5. Structure of a complete IEEE 802.11 MAC frame. Dashed lines indicate fields that needed to be modified for the implementation of two-for-one cooperation in the *ath5k* driver.

to retransmit overheard packets if the relay address matches. Unfortunately, some (time-critical) MAC functions cannot be changed, as they reside in proprietary firmware that is closely tied to the underlying Application-Specific Integrated Circuit (ASIC) that encapsulates the physical layer. Fig. 4 shows a graphical presentation of the different layers. For this reason, it is impossible to modify Request-to-Send (RTS), Clear-to-Send (CTS), and Acknowledgment (ACK). Specifically, the ACK is automatically generated and always sent if the intended receiver correctly receives a data frame. We choose to implement a simplified version of the two-for-one cooperation protocol without modified handshakes as the generation of RTS, CTS, and ACK frames is not implemented in software and hence cannot be modified.

We modify the hardware-specific (*ath5k*) and the hardware-independent (*mac80211*) drivers. The source code and manual can be found at http://www.sourceforge.net/projects/coopwifi/. Whenever the operating system has a packet to transmit, it will encapsulate it in a *socket buffer* and send it to the *mac80211* driver. The socket buffer is an internal data structure of the network subsystem of the Linux kernel that represents a single packet [12]. The *mac80211* driver receives a link-layer packet with an 802.3 header that needs to be transformed into an 802.11 packet by the driver. Our first modification is done at the 802.11 header by changing the default protocol version to a reserved version number, i.e., from 00 to 01, to distinguish conventional 802.11 packets from our new packets. This and other modifications are illustrated in Fig. 5. Normally, the hardware must drop all frames with a protocol version other than the default, but this can be easily disabled. Furthermore, we use the fourth address field in the 802.11 header to store the *relay address*. To do so, we pretend to use the wireless distribution system by setting both From DS and To DS fields to one. Initially, the relay address field is set to the broadcast address (all ones) because information about relays is gathered only at the hardware-specific *ath5k* driver for efficiency. Upon reception of a packet, *ath5k* learns about other nodes in its vicinity. By discarding unwanted packets without passing them to *mac80211* we avoid filling up the reception buffer unnecessarily. Thus, only *ath5k* implements the so-called *relay table*, which is similar to the CoopTable of CoopMAC [8].

After converting the 802.3 header to an 802.11 header for the packet to be transmitted, the packet passes through the remaining *mac80211* just as non-modified packets would. Usually, the hardware automatically assigns the sequence number to outgoing packets. Since the relayed packets need to have the same sequence number as the original packet, we do not want the hardware to assign new sequence numbers to relayed packets. Unfortunately, it is impossible to disable the sequence number assignment for specific frames only, so we disabled the automatic assignment and use a software implementation in *ath5k* instead. This makes it possible to skip relayed frames by checking the protocol version and enables the relay to retransmit overheard packets without altering the packet. It also allows us to detect duplicates at the destination.

For two-for-one cooperation, we do not want corrupt packets to be retransmitted by the source. Therefore, we configure the retry counter to transmit packets only once. For retransmission, our relay table lists suitable relays for each known destination. Although this is not needed in a static four-node setup, it is required in a dynamic environment where users may join and leave the network at any time. If the relay table does not contain a suitable relay for the intended destination, we use the broadcast address as the relay address. Then, every node that receives the packet except the destination can set the relay address to its own MAC address and retransmit the packet. If the source of the packet overhears the relayed packet, it associates the relay address with the destination address using the relay table for later transmissions. We do not modify the source address to leave the hardware ACKs intact. Upon correct reception of a data frame, the hardware immediately acknowledges the frame by transmitting an ACK to the source. If the destination only receives the relayed packet, it still must send the ACK to the source and not to the relay. Since we cannot modify addresses, we use another reserved protocol version for relayed packets, i.e., 10. This helps us to distinguish at the destination whether the packet originated from the source (protocol version is 01) or the relay (protocol version is 10).

For the relay to overhear packets, two approaches are possible. First, the driver can be configured to work in *promisciuous mode*. In this case, the hardware does not filter any packets based on the destination address but passes all packets to the driver. In this case, we can use the Basic Service Set Identifier (BSSID) field to store the address of the second hop that needs to be obtained from the routing protocol. Unfortunately, enabling promisciuous mode results in a large overhead, due to the numerous packets from interfering users of different networks, which need to be processed by the driver. For efficiency, a second approach can be used instead. The hardware does not filter packets that are sent to the broadcast address. Thus, by storing the destination address in the BSSID field that, in the original implementation, contains the network's name, we can use the broadcast address as destination address instead. Upon reception, we restore the original destination address from the BSSID field. Since the hardware does not allow us to use arbitrary MAC headers, we

must store the second hop address as the first six bytes of the payload, shifting the payload accordingly. For the experiments that follow, we avoid promisciuous mode for performance reasons.

The receiver path is far more complex than the transmission path within both drivers. We must make sure that the relay only retransmits packets when the relay address in the packet matches its MAC address or if the relay address is set to the broadcast address (i.e., the source does not know a suitable relay yet). According to Selection Decode-and-Forward (SDF), we do not retransmit corrupt packets indicated either by a failed CRC or corrupt Physical Layer Convergence Procedure (PLCP) header. These packets are dropped right away. We use the existing code in *ath5k* to pass on the packet (again contained in a socket buffer) from *ath5k* to *mac80211*. The *ath5k* uses the retry bit to signal the *mac80211* whether to further pass on the packet to the operating system (retry bit unset) or whether to put it into the existing transmission queue for relaying (relay bit set). Right before we can put the packet into the queue, we need to make sure that *ath5k* will accept the packet as an outgoing packet. This is needed because the packet does not pass through the normal transmission path and therefore lacks required information, e.g., the transmission rate. Currently, we use a static transmission rate that can be set over the Linux *proc* interface (a configuration facility integrated into the file system that allows user applications to configure parameters of device drivers). Since the hardware automatically computes and appends the CRC to any frame prepared for transmission, the relay's *ath5k* must remove the CRC of the overheard packet. Normally, the CRC is removed later on the receive path within *mac80211*.

Since the destination can now receive packets from source and relay, we need to drop duplicates. Like the relay, the destination does not accept corrupt packets. Whenever the destination receives a packet correctly, i.e., both PLCP header and CRC are correct, we check the protocol version to identify the packet's origin. If the packet originates from the source, we accept the packet and continue after having stored the packet's sequence number and the time of its arrival. Thus, whenever we receive a packet from a relay, we compare its sequence number and time of arrival with the stored values to decide whether the corresponding packet from the source has already arrived correctly. In which case, the relay's packet can be dropped to avoid duplicates.

In our experiment, which is detailed and discussed in Sec. V, we want to show which gains are actually possible with two-for-one cooperation using commodity hardware. Therefore, we statically select nodes, so that first and second hop are always known. This way, our results are not tainted by relay selection effects.

V. Experimental Measurements with Commodity Hardware

Fig. 6 shows the experimental setup in the lab. Due to the lack of official documentation for the Atheros AR2414 chipset and the experimental *ath5k*, we were not able to reliably set and determine the actual transmission power. Therefore, in order to get a meaningful estimate, we used a spectrum analyzer with a modified Wireless Local Area Network (WLAN) card for calibration. To our surprise, changing transmission power setting did not seem to have any effect. We cannot say if this is because of a driver issue or a hardware limitation.

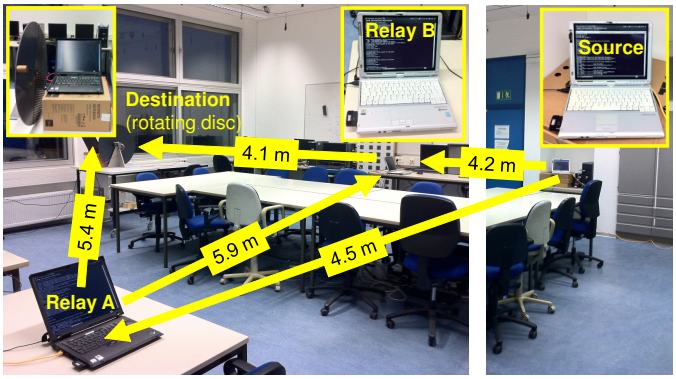


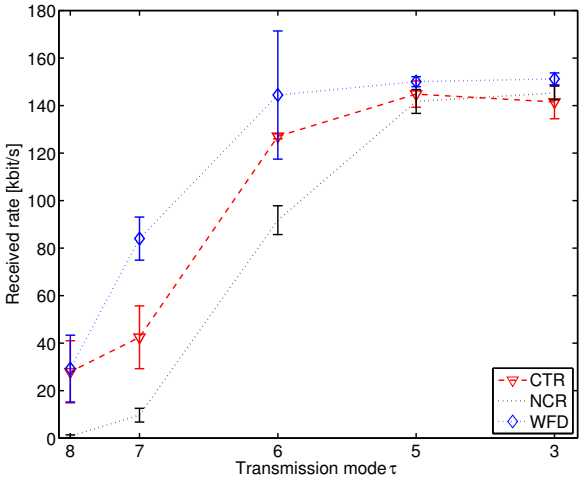
Fig. 6. Symmetric diamond consisting of four notebooks with Atheros WLAN cards and a rotating disc in front of the destination to emulate fading. Packets received on the direct path are ignored.

Since we cannot vary transmit power, we have to use a workaround to get varying signal quality at the receiver. By using a fixed packet generation rate, we can vary IEEE 802.11 TX modes. Higher modulation and code rates require better SNR at the receiver for successful decoding. Therefore, a higher transmission rate is comparable to a lower SNR. In the following, we will often use TX modes and SNR interchangeably, even though they are obviously not the same. We argue that they show similar performance results.

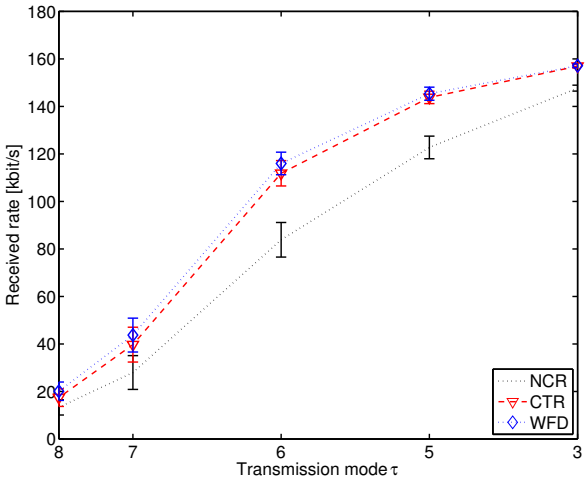
TABLE I
Transmission modes of the IEEE 802.11a/g physical layer [10]

Mode τ	Modulation	Coding rate R_c	Coded bits per OFDM	Data bits symbol	Data rate [Mbit/s]
1	BPSK	1/2	48	24	6
2	BPSK	3/4	48	36	9
3	QPSK	1/2	96	48	12
4	QPSK	3/4	96	72	18
5	16-QAM	1/2	192	96	24
6	16-QAM	3/4	192	144	36
7	64-QAM	2/3	288	192	48
8	64-QAM	3/4	288	216	54

We measure NCR, CTR, and WFD for the transmission modes 3, 5, 6, 7, and 8 (corresponding to transmission rates 12, 24, 36, 48, and 54 Mbit/s) as summarized in Table I by constantly transmitting 50.000 packets containing 200 bytes payload every 0.01 s, resulting in a data rate of 160 kbit/s as an example for voice-like traffic. Fig. 7(a) shows the measured data rate at the receiver for varying transmission modes (and hence varying modulation and code rate as shown in Table I). Due to this low rate traffic model, the network never operates in saturated conditions. Therefore, all approaches achieve the same data rate at the lowest transmission rate, corresponding



(a) Experimental results



(b) Simulation results

Fig. 7. Experimental and analytical results for received rate vs. transmission rate for the voice-like load model comparing NCR to the proposed MAC protocol with four-nodes setups. Confidence intervals shown for 95 % confidence level.

to the highest SNR. Both CTR and WFD outperform NCR due to their increased diversity order of two at all nodes. In the experimental results, WFD outperforms CTR. Interestingly, the simulation results depicted in Fig. 7(b) show CTR and WFD very close together. One explanation for the difference is that in reality the wireless channel is worse than modeled, so that using cooperation is even more beneficial. The WFD protocol cooperates more often and, therefore, benefits more.

Unsurprising, both CTR and WFD outperform NCR due to their increased diversity order. It is important to note that WFD saves an entire relay's transmission compared to CTR. In conclusion, the experimental results show that two-for-one cooperation can be implemented on commodity hardware and that it outperforms conventional relaying protocols.

VI. Conclusion

Two-for-one cooperation is superior to conventional cooperation protocols, such as CTR. As long as routing layer information is available in a multi-hop situation, two-for-one cooperation reduces the total number of transmissions, while maintaining or even improving overall performance.

We implemented two-for-one cooperation on commodity Wi-Fi hardware, which today is readily available in a numerous devices. However, this hardware and the IEEE 802.11 standard were not designed for cooperation protocols. We described in detail how we overcame certain limitations and how we fitted the required extra information into unused header fields and partially into the payload. We further illustrated how to make the modifications transparent for the higher application layer. The result is a first version of our *cooperation enabled Wi-Fi driver* for different cooperation protocols.

Acknowledgment

This work is partially financed by the Federal Ministry of Education and Research (BMBF) under the project MIKOA (Verbund-Nr. V3AVS009).

References

[1] S. Valentin, H. S. Lichte, H. Karl, I. Aad, L. Loyola, and J. Widmer, "Opportunistic relaying vs. selective cooperation: analyzing the occurrence-conditioned outage capacity," in *Proc. 11th Int. Symp. Modeling, Anal. and Simulation of Wireless and Mobile Syst. (MSWiM)*, Oct. 2008, pp. 193–202.

[2] H. S. Lichte, S. Valentin, H. Karl, I. Aad, L. Loyola, and J. Widmer, "Design and evaluation of a routing-informed cooperative MAC protocol for ad hoc networks," in *Proc. 27th IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2008, pp. 1858–1866.

[3] S. Valentin, D. H. Woldegebreal, T. Volkhausen, and H. Karl, "Combining for cooperative WLANs - a reality check based on prototype measurements," in *Proc. IEEE Workshop on Cooperative Mobile Networks (CoCoNET2)*, Jun. 2009, pp. 1–5.

[4] A. Bletsas, A. Khisti, D. P. Reed, and A. Lippman, "A simple cooperative diversity method based on network path selection," vol. 24, no. 3, pp. 659–672, 2006.

[5] S. Biswas and R. Morris, "ExOR: opportunistic multi-hop routing for wireless networks," in *Proc. Conf. Applications, Technologies, Architectures, and Protocols for Comput. Commun.*, 2005, pp. 133–144.

[6] S. Valentin, "Cooperative relaying and its application – from analysis to prototypes," Ph.D. dissertation, Faculty for Electrical Engineering, Computer Science and Mathematics, University of Paderborn, Oct. 2009.

[7] A. Bletsas and A. Lippman, "Implementing cooperative diversity antenna arrays with commodity hardware," *IEEE Communications Magazine*, vol. 44, no. 12, pp. 33–40, Dec. 2006.

[8] T. Korakis, Z. Tao, S. R. Singh, P. Liu, and S. S. Panwar, "Implementation of a cooperative MAC protocol: performance and challenges in a real environment," *EURASIP Journal on Wireless Communications and Networking*, vol. 2009, pp. 1–19, 2009.

[9] K. Dridger, "Implementing cooperative diversity on IEEE 802.11 WLAN adapters – a feasibility study," Bachelor Thesis, Faculty for Electrical Engineering, Computer Science and Mathematics, University of Paderborn, Dec. 2009.

[10] *IEEE Std 802.11a-1999 – Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications – High-speed Physical Layer in the 5 GHz Band*, IEEE, 1999.

[11] M. K. Simon and M.-S. Alouini, *Digital Communications over Fading Channels*, 2nd ed. John Wiley & Sons, Inc., 2004.

[12] J. Corbet, A. Rubini, and G. Kroah-Hartman, *Linux Device Drivers*, 3rd ed., A. Oram, Ed. O'Reilly, 2005.