Support of TCP in Wireless Mesh with Unstable Packet Forwarding Capacity

Chen Zhang, Yuanzhu Chen, and Cheng Li

Wireless Networking and Mobile Computing Laboratory
Memorial University of Newfoundland
St. John's, NL, A1B 3X5, Canada

***Abstract*—Opportunistic forwarding and network coding utilize the broadcasting nature of wireless transmission and fluctuation of link quality for enhanced performance in multi-hop wireless networks. However, TCP is not well supported because of the way they operate. The frequent occurrences of dropped packets and out-of-order arrival of them in opportunistic forwarding and decoding delay in network coding overthrow TCP's congestion control. We propose a mechanism, dubbed TCPFender, for TCP to function over the network layer that uses opportunistic data forwarding and network coding, to properly conduct congestion control in TCP and provide reliable data transport. Our experiment shows that TCPFender achieves significantly higher throughput compared to TCP over IP in a simulated wireless mesh.**

I. Introduction

A multi-hop wireless network, also known as wireless mesh, is a wireless data communication network, where nodes are situated beyond radio transmission ranges such that intermediate nodes are required to forward data [1]. They originated from military communication and disaster relief scenario but are nowadays finding civilian applications. Compared to single-hop wireless networks, such as Wi-Fi and cellular networks, multi-hop wireless networks can be deployed rapidly because they do not usually require infrastructure support[2]. However, to have such capabilities, important issues must be worked out, such as finding a path from one node to another efficiently, maintaining reliable wireless links, protecting nodes from network attacks, conserving power consumption, support applications of different characteristics, and so on.

Link quality variation has traditionally been treated as an adversarial factor in wireless networks, where their effect must be concealed from upper-layer protocols using strong codes or automatic retransmission. However, recent innovative data transfer methods utilize such a characteristic explicitly to achieve network performance that would not be possible in wireline networks [3]. In particular, ExOR [4] is a seminal effort in utilizing the quality fluctuation of wireless links. It involves an interplay between the network and link layers so that packet forwarding can be scheduled cooperatively on a per-packet, per-transmission basis. Another important feature of wireless links is broadcasting and it is natural to support network coding by mixing multiple data flows at a common intermediate node [5] [6] [7] [8]. For example, COPE [9] is a framework to combine and code data flows through such joint nodes to achieve higher throughput than IP forwarding in IEEE 802.11 wireless mesh networks. Interestingly, these ideas can be realized at the same time for another level of performance improvement. In particular, MORE [10] uses random linear network coding at the source and intermediate nodes and allows them to forward the coded packets opportunistically. It is able to ensure that nodes hearing the same transmission do not forward the same packets, so it does not need a special scheduler and achieve a better spatial reuse.

In general, opportunistic data forwarding and wireless network coding are proven to be effective in transporting UDP data streams. However, they are inherently unsuitable for supporting TCP. In such forwarding mechanisms, the frequent occurrences of dropped packets or out-of-order arrival of them overthrow TCP's congestion control, which has precise interpretations of the timing and presence of the feedback coming from the network layer. Unfortunately, opportunistic data forwarding does not attempt to forward packets in the same order as they are injected in the network so many packets will arrive in a different order. Similarly, in network coding, because both encoding and decoding introduce delay at the network layer, along with possible scenarios of not being able to decode some packets, TCP will also see many duplicate ACK segments and frequent timeouts. In fact, we know that the performance of TCP suffers from transient link quality fades and frequent collisions in wireless mesh networks despite of MAC layer retransmissions [11] [12].

In this article, we propose a solution for TCP function over the network layer that uses opportunistic data forwarding and network coding, dubbed TCPFender. By replacing part of the TCP's control feedback loop with a new mechanism, TCP is able to provide reliable data transport and congestion control as it is meant to. We tested TCPFender using computer simulation of a wireless mesh network, and observed a 80% throughput gain over TCP over IP forwarding.

The rest of this paper is organized as follows. In Section II we briefly describe related work. Section III is the description of our TCPFender and Section IV reports experimental results. We conclude this article with some discussion and outlook in Section V.

II. Related Work

ExOR [4] utilizes link quality variation of wireless channels at the link layer and obtains the opportunistic forwarding gains. It is an explorative cross-layer opportunistic data forwarding mechanism, which proposes a forwarding schedule on nodes to reduce duplicate transmissions. However, the strict

schedule reduces the possibilities of spatial reuse so that nodes can not forward packets at the same time even though the wireless medium is available. Therefore, the channel capacity will be under-utilized.

MORE [10] propose a solution to the above problem by incorporating random network coding into opportunistic data forwarding. Based on a credit mechanism using link loss, it restricts the redundancy for each received packet. MORE divides data packets from upper layer into several batches and generates coded packets with a random linear combination of each batch. A receiver can decode original packets after it receives enough independent coded packets. However, the problem with MORE is that it would introduce a long decoding delay which may increase the possibilities of timeout in TCP, which will have a negative impact on TCP performance.

Furthermore, in mesh networks, the principal problem of TCP lies in performing congestion control in case of losses that are not induced by network congestion [12] [13]. Wireless networks suffer from several types of losses which are not related to congestion, for example, physical layer link loss. TCP is unable to distinguish losses caused by link quality variation or network congestion. When a packet is detected to be lost, TCP slows down the sending rate by adjusting its congestion window, it greatly reduces the TCP throughput.

In [14], Sundararajan *et al* proposes a new protocol called TCP/NC, it uses a sliding window approach for coding operations and modifies the TCP acknowledgment schemes by acknowledging the degrees of freedom, this scheme is defined as Seen concept. When a coded packet has been seen instead of a original packet been received, the receiver would generate an acknowledgment. Such ACKs enable a TCP-compatible sliding-window approach to network coding. However, opportunistic forwarding is not considered in their work, and the link quality variation is not explored to support network coding and improve the network performance. To address this problem, in this paper, we will propose an adaption module at both TCP sender and receiver sides to support TCP in a wireless mesh network where the network layer uses opportunistic data forwarding and network coding.

III. Design of TCPFender

A. TCPFender Overview

The objective of TCPFender is to allow TCP to function in a wireless mesh network that uses opportunistic data forwarding and network coding at its network layer. The essence is for the TCP sender to maintain a large congestion window even when there are a large number of dropped packets or out-of-order delivery. We intend to keep TCP intact for the principle of software modularization. Furthermore, the underlying link layer should be the stock IEEE 802.11, which only provides standard unreliable broadcast or reliable unitcast (best effort with a limited number of retransmits).

The task is challenging because TCP has its own interpretation of the arrival (or absence) of the ACK segments and their timing. For the sender to be able to open up its congestion window, it needs to continue to see ACK coming in from the network. The dilemma is that when packets are dropped or arrive out of order, the TCP receiver cannot signal the sender to proceed with the expected ACK segment.

TCPFender addresses this issue by allowing the receiving side to provide positive feedback early on when innovative coded packets are received, i.e. suggesting that more information has come through the network although it cannot be decoded yet for the time being. Furthermore, it triggers fast recovery when the receiving side acknowledges the arrival of packets belonging to a later batch, in which case the sending side will resend dropped packets of the previous batch. On the other hand, it is able to differentiate duplicate ACK segments caused by network congestion from those caused by opportunistic data forwarding.

B. Design Detail

TCPFender is inspired by MORE to utilize the opportunistic forwarding and network coding to better support TCP. The way we designed it is based on a batch-oriented wireless network coding operation similar to MORE. TCPFender has adaption modules under both the TCP sender and receiver. These modules are used to interpret observation of the network layer phenomena the way that is understandable by TCP. It forwards data in a similar batch-operated fashion as MORE. On the sender side, after receiving packets from the transport layer, packets are grouped into batches, where all packets in the same batch carry encoding vectors of the same basis. After the receiver decodes all packets of a given batch, the decoded (original) packets are delivered to up to the TCP receiver. These are done by the sending module and receiving module at the source and destination nodes. Both modules are part of the network layer that sits between TCP and the packet forwarding functions. Both end nodes and intermediate nodes run an enhanced version of MORE.

1) Modifications to MORE: The opportunistic data forwarding and network coding here is implemented based on MORE. As we recall, MORE is a batch-oriented operation, where all data pushed down by the transport layer sender are grouped into batches. Each batch has a fixed number β ($\beta = 50$ in our implementation) of segments of equal length (with possible padding). When the source has accumulated a batch worth data, they are coded with random linear network coding, tagged with the encoding vectors used, and transmitted to downstream nodes. Here, a downstream node is any node in the network closer to the destination. Any downstream node can recode and forward packets when receives sufficient number of them. MORE uses a transmission credit to balance the number of packets that a node receives and forwards. When the batch has been received and decoded by the destination, a batch acknowledgement is sent back to the source for it to start the next batch.

We have two important revisions to MORE. First, when processing a given batch, the source does not need to wait till the last packet from the TCP sender before transmitting coded packets. That is, if k packets ($k < 50$) have been sent down by TCP at a point of time, a random linear combination of these k packets is created and transmitted. Initially, the coded packets only include information of the first few TCP segments of the batch, but will include more towards the end of the batch. The reason for this "early release" behavior is for the TCP receiving side to be able to provide feedback for the sender to open up the congestion window. Second, we use a deeper pipelining than MORE, where we allow multiple batches to flow in the

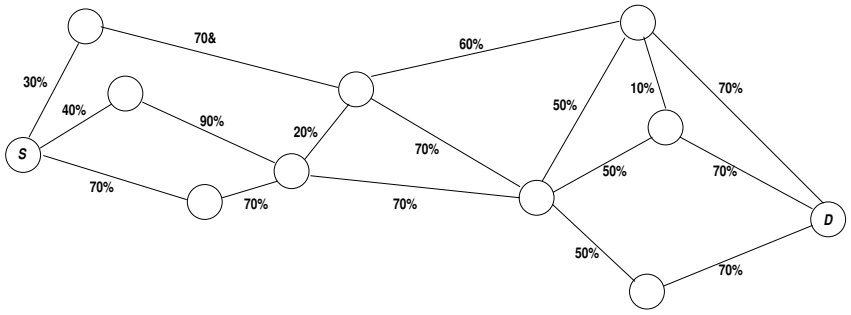

Fig. 1: Topology of simulation with physical layer link loss rates indicated.

network at the same time. To do that, the sending side does not need to wait for the batch acknowledgement before proceeding with the next batch. In this case, packets of a batch is labeled with a batch index for differentiation. The purpose of this is for TCP to have a stable, large congestion window size rather than having to reset it to 1 for each new batch. The cost of such pipelining is that all nodes must now maintain packets for multiple batches.

2) Receiver module: The receiver module is deployed at network layer on the receiver side. Its basic function is to generate ACKs and detect congestion in the network. It expects packets in the order of increasing batch index. For example, when it is expecting the bth batch, it implies that it has successfully received packets of the previous $b-1$ batches and delivered them up to the TCP receiver. In this case, it is only interested in and buffers packets of the bth batch or later.

Due to the network layer's characteristics, the destination node may receive packets of any batch. Suppose that the receiver is expecting the bth batch, and that the rank of the decoding matrix of this batch is r. In this case, the receiver has "almost" received $\beta \times (b-1)+r$ packets of the TCP flow, where $\beta \times (b-1)$ packets have been decoded and pushed up the TCP receiver, and r packets are still in the decoding matrix. When it receives a coded packet of the b'th batch, if $b' < b$ the packet is discarded. Otherwise, this packet is inserted to the corresponding decoding matrix. Such an insertion can increase r by 1 if $b' = b$ and the received packet is innovative. In either case, it generates an ACK of sequence number $\beta \times (b-1)+r$, which is sent over IP back to the source node. One exception is that, if $r = \beta$ (i.e. decoding matrix become full rank), the ACK sequence number is $\beta \times (\hat{b}-1)+\hat{r}$, where $\hat{b}$ is the next batch that is not full and $\hat{r}$ is its rank. At this point, the receiver moves on to the $\hat{b}$th batch. This mechanism ensures that the receiver can send multiple duplicate ACKs for the sender to detect congestion and start fast recovery. It also guarantees the reliable transmission at the end of the transmission of each batch.

3) Sender module: The sender's has two responsibilities. First, it buffers original uncoded packets in batches that have not been acknowledged. This purpose is that, when TCP pushes down a previously sent packet due to a loss event, it can still mix it with other packets of the same batch. Second, it needs to discern a case of seemingly duplicate ACKs that is not in fact caused by network congestion.

This latter point is a phenomenon caused by opportunistic forwarding. Specifically, when the network links are of high quality at a certain point, many extra coded packets may arrive at the destination, causing the receiving module to send multiple ACKs of the same sequence number. In this case, such duplicate ACKs are not a signal for network congestion, and should be treated differently by the sending module. These two cases of duplicate ACKs can actually be differentiated by tagging the ACKs with the sequence numbers of the data segments triggering them. Note that these ACKs are for the functioning of the sending and receiving modules, and need to be processed before handing up to the TCP sender.

IV. Performance evaluation

In this section, we investigate the performance of TCPFender by running computer simulation using Network Simulator ns-2 (version 2-25). The topology of the simulation is depicted in Fig. 1, where the Physical Layer link loss rates are also indicated. The source and destination nodes are at the opposite ends of the network, one FTP application delivers long files from the source to the destination. Source node emits packets continuously till the end of simulation, and the simulation last for 100 seconds. All the wireless links have a bandwidth of 1Mbps and the buffer size on the interfaces is set to 100 packets. To compensate for the link loss, we use the hop-to-hop redundancy factor for each packet in a loss link. The redundancy factor is calculated based on loss rate and proposed by MORE [10]. However, if we directly use the loss rate of link at the Physical Layer, it will not include the effect of packet collisions. Instead, we conducted some preliminary tests to measure link loss rate of both the Physical and Link layers, which is naturally higher but closer to the real rate. The redundancy factors of the links are thus set according to these revised rates. We compare our protocol against TCP/IP in such a lossy wireless mesh network.

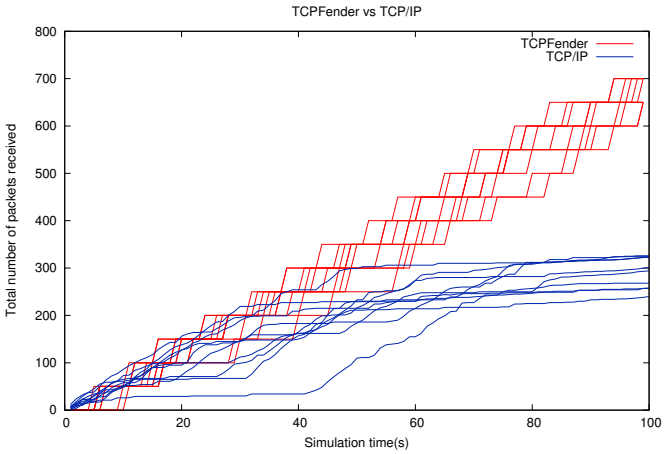

Fig. 2: Throughput of TCPFender vs. TCP/IP

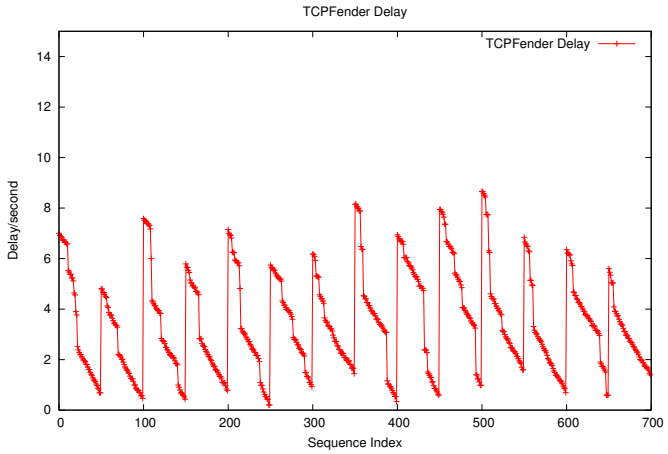

Fig. 4: End to end delay of TCPFender

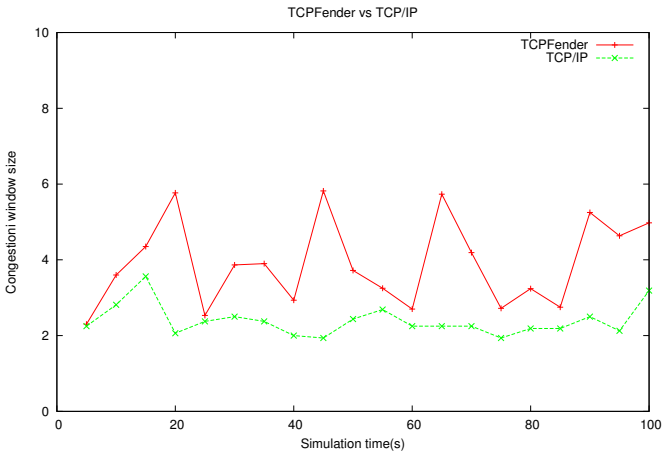

Fig. 3: Congestion window size for TCPFender and TCP/IP

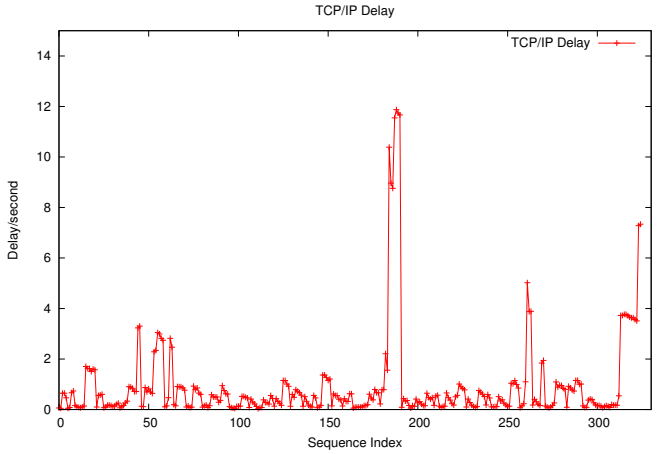

Fig. 5: End to end delay of TCP/IP

We first examine whether TCPFender can effectively help opportunistic forwarding and network coding to transport data reliably in the mesh network. We repeated the same scenario for 10 times with different random seeds for TCPFender and TCP over IP, respectively. The source node S and the destination node D are at the opposite ends of topology. In TCPFender, every other node has the opportunity to forward coded packets as they are all somewhere between the source and destination. All nodes operated in the 802.11 broadcast mode. In contrast for TCP/IP, we adopted the short-path routing to forward packets. Here, we used the standard ARQ mechanism of 802.11 to offset the effect of link loss.

We plotted the throughput of both approaches in Fig. 2, which shows, for each simulation run, how many packets have been received by the destination node's TCP module at a given point of the test. We can observe that there is a clear advantage of TCPFender over TCP/IP, and our data show that the gain is about 80%. Note that for the case of TCPFender, the number of packets received by the destination always increase by 50. This is when a batch is successfully received and decoded, at which point they are delivered up by the receiving module to TCP all at once. The performance difference shows the effectiveness of TCPFender in keeping the sender's congestion window open by the receiving module providing positive feedback whenever the rank of its decoding matrix increases.

To further verify this, we plotted the evolution of the congestion window size on the source node for one TCPFender and TCP/IP pair in Fig. 3. For TCP/IP, the packet loss keeps the congestion window is closed to approximately 2 even though the link layer ARQ attempts to transmit a packet up to 7 times. For TCPFender, because the receiver module would acknowledge the increase of the degree of freedom in the decoding matrix, the TCP sender is able to keep the window at around 4, effectively doubling the throughput.

Now that TCP can be supported by network coding and opportunistic data forwarding, we are also interested in the end to end packet delivery delay for both approaches, which are plotted in Fig. 4 and Fig. 5. In the figures, the x-axis is the TCP sequence number, and the y-axis is the delay in seconds. Notice that the delay for TCP/IP is a fraction of TCPFender

because the latter must wait till all packets can be decoded before handing them over to TCP. This is an inherent feature of batch-based network coding rather than that of TCPFender. In Fig. 4, the delay is in a saw-tooth form because packets at the beginning a batch always have longer delays than packets later in the batch.

V. Concluding Remarks

This work was motivated by the need of supporting TCP in a wireless mesh network where the network layer uses opportunistic data forwarding and network coding. For this to work, we designed an adaption module at the TCP sender and receiver sides, respectively. In order for the TCP sender to keep its congestion window open, the modules completes the control feedback loop of TCP in two ways. First, it releases ACK segments early when the receiver module sees an innovative packet. Second, it differentiates duplicate ACKs caused by network congestion from those caused by opportunistic data forwarding. From the simulation results, TCPFender has an approximately 80% throughput gain over TCP/IP even though it has additional overhead for TCPFender header.

In fact, our adaptive modules are designed generally enough to not just support network coding and opportunistic data forwarding, but any packet forwarding technique that can cause many dropped packets or out-of-order arrivals. One example would be multi-path routing, where IP packets of the same data flow can follow different paths from the source to the destination [15]. By emulating how TCP receiver would signal the TCP sender, we are able to adapt TCP to function over such a network layer with unstable packet forwarding capacity without having to modify TCP itself.

TCPFender can be extended and further investigated in the following interesting ways. For example, to verify how multiple TCP flows interact with each other over a network coded, opportunistic forwarding network layer, or more generally any error-prone network layer. Alternatively if we look at the network layer, how we can code the duplex data of the same TCP flow, or how TCP flows with partially overlapped routes can be coded and forwarded opportunistically.

Acknowledgement

This work was supported in part by the Natural Sciences and Engineering Research Council (NSERC) of Canada (Discovery Grants 293264-12 and 327667-2010, and Strategic Project Grant STPGP 397491-10).

References

[1] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, *Mobile Ad Hoc Networking: The Cutting Edge Directions*, 2nd ed. Wiley-IEEE Press, March 2014.

[2] P. Larsson, "Selection diversity forwarding in a multihop packet radio network with fading channel and capture," *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 5, no. 4, pp. 47–54, October 2001.

[3] Y. Chen, J. Zhang, and I. Marsic, "Link-layer-and-above diversity in multi-hop wireless networks," *IEEE Communications Magazine*, vol. 47, no. 2, pp. 118–124, February 2009.

[4] S. Biswas and R. Morris, "ExOR: Opportunistic multi-hop routing for wireless networks," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*. ACM, August 2005, pp. 133–144.

[5] Z. Wang, Y. Chen, and C. Li, "Corman: A novel cooperative opportunistic routing scheme in mobile ad hoc networks," *IEEE journal on selected areas in communications*, vol. 30, no. 2, February 2012.

[6] R. Ahlswede, N. Cai, Shuo-Yen, R. Li, and R. W. Yeung, "Network information flow," *Information Theory*, vol. 46, no. 4, pp. 1204 – 1216, July 2000.

[7] P. A. Chou, Y. Wu, and K. Jain, "Practical network coding," in *Allerton Conference on Communication, Control, and Computing*, October 2003.

[8] C. Fragouli, J.-Y. L. Boudec, and J. Widmer, "Network coding: an instant primer," *ACM SIGCOMM Computer Communication Review*, vol. 36, no. 1, pp. 63–68, January 2006.

[9] S. Basagni, M. Conti, S. Giordano, and I. Stojmenovic, "XORs in the air: Practical wireless network coding," *Networking, IEEE/ACM Transactions*, vol. 16, no. 3, pp. 497 – 510, June 2008.

[10] S. Chachulski, M. Jennings, S. Katt, and D. Katabi, "Trading structure for randomness in wireless opportunistic routing," *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 12, pp. 169–180, October 2007.

[11] A. A. Hanbali, E. Altman, and P. Nain, "A survey of TCP over ad hoc networks," *Communications Surveys & Tutorials*, vol. 7, no. 3, pp. 22–36, March 2006.

[12] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz, "A comparison of mechanisms for improving tcp performance over wireless links," *Networking, IEEE/ACM Transactions*, vol. 5, no. 6, pp. 756 – 769, December 1997.

[13] H. Balakrishnan, S. Seshan, E. Amir, and R. H. Katz, "Improving tcp/ip performance over wireless networks," *Proceeding MobiCom Proceedings of the 1st annual international conference on Mobile computing and networking*, pp. 2–11, 1995.

[14] J. K. Sundararajan, D. Shah, M. Medard, S. Jakubczak, M. Mitzenmacher, and J. Barros, "Network coding meets TCP: Theory and implementation," *Proceedings of the IEEE*, vol. 99, no. 3, pp. 490 – 512, March 2011.

[15] S. Jain and S. R. Das, "Exploiting path diversity in the link layer in wireless ad hoc networks," in *Proceedings of the Sixth IEEE International Symposium on a World of Wireless Mobile and Multimedia Networks*. IEEE, June 2005, pp. 22 – 30.