Multipath TCP with Network Coding for Wireless Mesh Networks

Steluta Gheorghiu Alberto Lopez Toledo Pablo Rodriguez
Telefonica Research
{steluta, alopezt, pablorr}@tid.es

***Abstract*—In wireless multihop networks, techniques such as multipath, local retransmissions and network coding have been successfully used to increase throughput and reduce losses. However, while these techniques improve forwarding performance for UDP, they often introduce side effects such as packet reordering and delay that heavily affect TCP traffic.**

In this paper we introduce *CoMP*, a network coding multipath forwarding scheme that improves the reliability and the performance of TCP sessions in wireless mesh networks. *CoMP* exploits the wireless mesh path diversity using network coding, performs congestion control and uses a credit-based method to control the rate at which linear combinations are transmitted. *CoMP* uses a simple algorithm to estimate losses and to send redundant linear combinations in order to maintain the decoding delay at a minimum and to prevent TCP timeouts and retransmissions.

We evaluate *CoMP* through extensive simulations and compare it to state-of-the-art protocols. We show that *CoMP* not only achieves a higher throughput, but also is more efficient than existing protocols, making TCP sessions feasible for wireless mesh networks even under heavy losses.

***Index Terms*—network coding, wireless, TCP, multipath**

I. Introduction

Wireless mesh networks (WMN) are emerging as the next-generation systems to connect communities. As these networks proliferate, it is expected that users will start demanding more versatile applications such as video streaming and VoIP. Unfortunately, the performance of TCP degrades very fast in WMN [4]. When losses occur in a wireless environment, TCP considers them as being caused by congestion, and triggers its congestion avoidance mechanism which reduces the sending rate to adapt to the new network conditions. Still, the actual capacity of the network has not changed, and hence the network becomes underutilized.

A typical approach to address the problem of improving TCP performance in WMN is to exploit the path diversity in the mesh. One example is *Horizon* [7], which also implements a mechanism to signal congestion to TCP, but it does not offer any reliability and every loss is recovered through retransmissions at TCP layer.

However, multiple paths pose further challenges. Consider the simple scenario from Fig. 1, where the source S can communicate with destination D using 3 neighbors, A, B and C. Assume that S sends three packets, P_1, P_2, P_3, and the three relays receive the packets as illustrated in the figure. In this situation, the nodes need to *coordinate* and *schedule the packets* transmissions so that the destination receives all of them without duplicates.

A. Lopez Toledo is supported by the Institució Catalana de Recerca i Estudis Avançats (ICREA). Telefonica Research participates in Torres Quevedo subprogram (MICINN), cofinanced by the European Social Fund, for Researchers recruitment. Part of this work was carried out with assistance of financial support from the European Community under grant FP7-INFSO-ICT-215252 (N-Crave Project). This work has been partially funded by project TSI2007-66869-C02-01 from Spanish Science and Technology Ministery (MICINN).

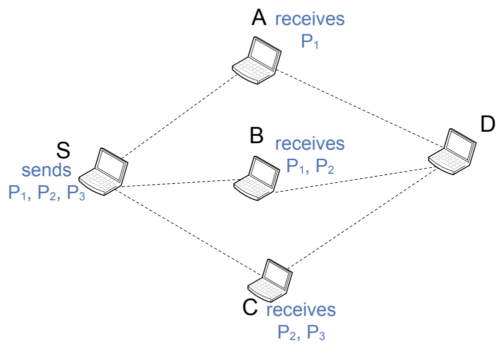


Fig. 1. Multipath example. The source S can reach the destination D through three neighbors, A, B and C.

Network coding [3] can effectively be employed to solve the relay coordination and the packet scheduling problems. In the above example, each of the relays can use network coding to simply forward a linear combination of the received packets. Once the destination D receives the three linear combinations, it solves a linear system and recovers the original packets.

Typical network coding techniques divide traffic in *generations* [6] and a coded packet is a linear combination of the packets that belong to the same group, e.g. *MORE* [5]. This approach introduces a coding delay, because the source can send coded packets only after it has all the packets from a group, and a receiver can decode native packets only after it receives a number of independent linear combinations. Furthermore, this delay in the coding operations has a negative impact on the TCP performance in wireless environments.

In order to avoid the coding delay, [8] presents *online coding*, where the source mixes the packets from the TCP congestion window and the receiver acknowledges every innovative linear combination it receives, even if it cannot decode a native packet immediately. This approach gives a new interpretation of the ACKs, and allows for a TCP-sliding window scheme for coding. However, this scheme is single path and the coding operations are performed end-to-end. For the rest of the paper we refer to this approach as *TCP-ARQ*.

With these considerations in mind, we introduce *CoMP*, a network coding multipath protocol for WMN that extends current state-of-the-art and combines network coding with multipath routing to increase throughput for TCP sessions in

WMN. *CoMP* is situated below TCP/IP in the protocol stack. The main features of our protocol are as follows:

- a multipath online network coding scheme that exploits the path diversity in a mesh network, and eliminates the delay problems introduced by coding (section II-A);
- an algorithm to estimate the loss probability in the network and to adjust online the rate of sending linear combinations (section II-B);
- a hybrid credit-based algorithm, where nodes compute distributedly the rate of generating linear combinations (section II-C);
- a congestion control mechanism based on back-pressure (section II-D).

We extensively evaluate *CoMP* using the *ns-2.33* network simulator [1], and compare it to state-of-the-art protocols: *Horizon* [7], *MORE* [5], and *TCP-ARQ* [8]. The results show that *CoMP* not only achieves a higher throughput, but it is also more efficient, due to exploiting path diversity and re-encoding packets on a hop-by-hop basis.

The paper is organized as follows. In section II we present our protocol with a detailed description of its features. Section III provides the results of our simulations and a comparison to *Horizon*, *MORE*, and *TCP-ARQ*. Section IV concludes with some final remarks.

II. Protocol Description

In this section we describe the main aspects of *CoMP*.

A. Multipath online coding

For our multipath coding scheme, we use a similar approach as the one in [8], but with some essential differences. First, *TCP-ARQ* uses only one path to forward the packets from sender to receiver, while *CoMP* takes advantage of the path diversity characteristic of WMN and forwards packets along multiple paths. Second, in *TCP-ARQ* relays only forward the packets received from the upstream nodes and the coding is performed end-to-end. In our scheme, intermediate nodes re-encode packets received from different upstream neighbors, on a hop-by-hop basis. In the next sections we discuss in more detail the mechanisms that we use to enable online network coding for multipath schemes.

B. Online loss estimation and rate adaptation

The rate at which nodes need to forward linear combinations to recover from losses remains an open research question. If the rate is too low, due to the unreliability of the wireless medium, receivers may not be able to decode. On the other hand, if the rate is too high, nodes may receive redundant linear combinations that consume network resources, but are not innovative. Moreover, since the wireless channel is variable in time, the rate of sending coded packets needs to be updated frequently enough to take into account current channel conditions. With these considerations in mind, we introduce an algorithm to estimate the loss between any two nodes and to distributedly compute the rate at which each node should forward linear combinations.

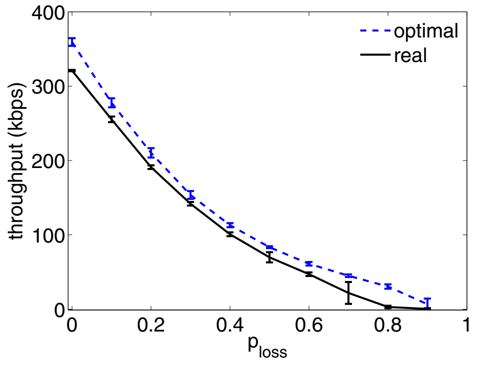


Fig. 2. Evaluation of the loss estimation algorithm for the topology in Fig. 1.

In order to estimate the loss, each node N_k keeps the following information:

- p_i: a list of the *ids* of the packets sent to each of its downstream neighbors, N_i;
- r_i: the total number of packets that each of its downstream neighbors N_i reports it had received from N_k;
- s_{kl}: the *id* of the last packet that node N_k received from each of its upstream neighbors, N_l;
- t_{kl}: the total number of packets that N_k received from each of its upstream neighbors, N_l.

Node N_k updates the p_i list whenever it sends a packet to downstream node N_i. s_{kl} and t_{kl} are updated with each reception of a new packet from upstream neighbor N_l. Note that the update is done even if the packet is not innovative.

Due to the broadcast nature of the wireless medium, a node may overhear the transmissions of its downstream neighbor, and we take advantage of this feature as follows. Whenever N_k has to send a packet, it piggybacks s_{kl} and t_{kl} in the header for each of its upstream nodes, N_l. Next, its upstream neighbors can pick up the packet and find out the information they need to estimate the loss. Since the destination does not forward any linear combination, the previous approach cannot be used. However, the destination needs to send TCP ACKs to the sender, so it piggybacks s_{kl} and t_{kl} on these packets.

Once a node N_k receives feedback from one of its next hops, say N_i, it updates the information for the corresponding hop and computes the loss towards that hop, according to Alg. 1. If the node determines that a packet has been lost, it sends a redundant linear combination and adjusts the sending rate taking into account the total estimated loss. The total loss that N_k sees is given by: $loss_k = \prod_{j=1}^{n_h} loss_{kj}$, where n_h represents the number of downstream nodes of relay N_k.

We evaluate our loss estimation algorithm for the topology in Fig. 1, where two relays forward packets to destination. For the *optimal* curve, the nodes send packets at the optimal redundancy rate, which is computed based on the given loss probability. For the *real* case when the nodes use Alg. 1, the throughput is very close to the optimal one, see Fig. 2. Note that our loss estimation algorithm is accurate, and it does not incur any cost, since it piggybacks the needed information on the packets sent in the network.

C. Hybrid credit-based scheme

When multiple paths are used in parallel, one important decision is how much of the traffic needs to be sent through

Algorithm 1 Loss estimation. Node N_k uses the feedback received from N_i to estimate the loss towards N_i, $loss_{ki}$

```
1: if node $N_k$ receives $s_{ik}$ and $t_{ik}$ from downstream neighbor $N_i$ then
2:     count no. pkts that $N_k$ sent to $N_i$, $w_s = |C|$, where $C = \{p_i(j)/p_i(j) \leq s_{ik}, j = 1 : size(p_i)\}$
3:     compute no. pkts that $N_i$ received from $N_k$, $w_r \leftarrow t_{ik} - r_i$
4:     $loss_{ki} = \frac{w_s - w_r}{w_s}$
5:     $r_i \leftarrow t_{ik}$
6:     erase $p_i(j)$ for which $p_i(j) \leq s_{ik}$
7: end if
```

each path. For the example in Fig. 1, if each node A, B and C generates a linear combination every time it receives a packet from the source, it results in an explosion of the rate, with the destination receiving several packets that are not innovative.

To address this issue, [5] and [6] associate a credit with each packet sent in the network. The credits are created by the source, transfered to the downstream nodes and consumed at the destination. For *CoMP*, we use a hybrid credit-based scheme which has two parts. First, credits are generated at source and transfered to the forwarders, as the schemes mentioned above do. Second, credits can be generated by intermediate nodes as well whenever they detect a loss, which differentiates our approach from the existing ones. This second part enables each node to compute locally and distributedly the rate that it needs to send to recover the losses.

D. Congestion control

For multipath schemes in WMN, where multiple flows can traverse the same set of links, one must ensure fairness and balance the load among the available paths. To this end, we introduce a congestion control mechanism to select a forwarder for a packet, based on the *backpressure* methodology [9], which allows a node to send packets to a neighbor as long as the neighbor is less congested. The level of congestion is given by the size of the queue, i.e. the more packets are queueing at a node, the more congested the node is. For the congestion control mechanism, each node N_k keeps the following information for every flow that it forwards:

- an output queue, output_buffer$_i, i = 1 : n_f$, where n_f is the number of flows that node N_k forwards;
- the queue sizes of its next hops for each flow, $q_{ij}, i = 1 : n_f, j = 1 : n_{hi}$, where n_{hi} is the number of next hops that N_k has for flow i.

Whenever the node has the opportunity to transmit a packet in the wireless medium, it first selects the flow for which to forward a packet, as the one for which it queues the highest number of packets (see Alg. 2). After the flow is selected, N_k chooses the forwarder to be the node with the shortest output queue, say N_h, as long as the backpressure condition holds.

Next, N_k removes the first packet from the *output_buffer*$_f$ and it adds the packet id in p_h, the list of packets sent to N_h. It adds in the header of the packet the size of its own queue, q_{fk} for backpressure computation and for each of its upstream neighbors N_l, the *id* of the last received packet, s_{kl}, and the total number of received packets, t_{kl}, for loss estimation. Finally, N_k sends the packet.

Algorithm 2 Flow scheduling and forwarder selection. When sending a packet, node N_k selects a flow f and chooses the forwarder for that flow. Next, it updates the state information and it sends the first packet from the output_buffer$_f$.

```
1: select the flow $f$ such that:
2: $f = \arg\max_i size(\text{output\_buffer}_i), i = 1 : n_f$
3: select forwarder: $N_h = \arg\min_j q_{fj}, j = 1 : n_{hf}$
4: remove $P_0$ from output_buffer$_f$
5: add $id(P_0)$ in $p_h$ list
6: add $q_{fk}$ and $(s_{kl}, t_{kl})$ for each upstream neighbor $N_l$, in the header of $P_0$
7: send $P_0$
```

E. System Design

For each flow f, every node keeps a coding_buffer$_f$ and an output_buffer$_f$. The coding_buffer$_f$ contains packets that the node uses to generate linear combinations. The source stores here native, uncoded packets, while the other nodes store linear combinations. The output_buffer$_f$ contains coded packets that will be sent in the network. Packets are removed from the output_buffer$_f$ whenever the node has the possibility to transmit a packet. In this case, a node first runs Alg. 2 to select a flow f and a forwarder N_h. After that, the node removes the first packet from the output_buffer$_f$, it updates the statistics for node N_h and it sends the packet.

Sender. When a native packet of flow f is produced, the source stores it in the coding_buffer$_f$. Note that source performs *online coding*, which means that it generates a new linear combination every time it receives a new packet. To generate a coded packet, the source mixes all the packets from the coding_buffer$_f$ and stores the resulting linear combination in the output_buffer$_f$ (see Alg. 3). When the source receives a TCP ACK for flow f, it removes the oldest packet from the coding_buffer$_f$, thus sliding the coding window.

Algorithm 3 Online coding at source. If a data packet arrives, the source stores it and generates a linear combination. m represents the size of the coding_buffer$_f$. The coefficients c_i are randomly chosen from $GF(2^{16})$. If it receives a TCP ACK, the source removes the oldest packet from the coding_buffer$_f$.

```
1: receive packet $P_k$ of flow $f$
2: if $P_k$ is DATA packet then
3:     store packet in the coding_buffer$_f$
4:     generate $LC_k = \sum_{i=1}^{m} c_i P_i$
5:     store $LC_k$ in the output_buffer$_f$
6: else
7:     $P_k$ is TCP ACK
8:     remove $P_0$ from coding_buffer$_f$
9: end if
```

Relays. When receiving a coded packet LC_r from flow f, an intermediate node N_k first removes from the coding_buffer$_f$ those linear combinations that contain native packets older than those mixed in LC_r. Then, he updates the state for the upstream neighbor N_l who sent LC_r. The updated information is the *id* of the last received packet s_{kl}, and the total number of received packets, t_{kl}. Next, if the coded packet is innovative, N_k stores it in the coding_buffer$_f$, otherwise N_k drops it (see Alg. 4). A relay generates and forwards a linear combination when it receives a credit. Note that a linear combination of coded packets is also a linear combination of the original packets.

Algorithm 4 Coding at relays. The relay N_k receives a packet of flow f from neighbor N_l. m represents the size of the coding_buffer$_f$. The coefficients c_i are chosen from $GF(2^{16})$ at random.

```
1: receive coded packet $LC_r$ of flow $f$
2: update coding_buffer$_f$
3: $s_{kl} = id(LC_r)$; $t_{kl} \leftarrow t_{kl} + 1$
4: if $LC_r$ is innovative then
5:     store $LC_r$ in the coding_buffer$_f$
6:     if receive credit associated to $LC_r$ then
7:         generate $LC_j = \sum_{i=1}^{m} c_i LC_i$
8:         store $LC_j$ in output_buffer$_f$
9:     end if
10: else
11:     drop $LC_r$
12: end if
```

Receiver. When destination N_d receives a linear combination of flow f, it updates the *id* of the last received packet and the total number of received packets for the sender. Next, it stores the coded packet only if it is linear independent with all the packets received previously. Since the source is coding online, then the destination decodes a native packet on the fly, if possible, and passes it up to the TCP layer (see Alg. 5). If no native packet can be decoded, the destination acknowledges the reception of the current packet to the source.

III. Performance Evaluation

We evaluate our protocol through extensive simulations in *ns-2.33* [1] and compare it to improved versions of *Horizon* [7], *MORE* [5], and *TCP-ARQ* [8]. *Horizon* estimates the TCP window and signals congestion to the TCP sender whenever the estimated value reaches a certain threshold. In our simulations, we feed *Horizon* with the actual TCP window size, not an estimate. Also, *MORE* sends probe packets periodically to estimate the loss in the network and adjusts the rate of sending linear combinations accordingly. For our implementation of *MORE* we use our algorithm to estimate losses instead, eliminating the overhead of sending probes. For *TCP-ARQ*, only the source sends redundant linear combinations and the authors state that the redundancy factor R should be adjusted online. Since the authors do not give any indication on how to adjust R online, we use our loss estimation algorithm Alg.1.

A. Simulation Setup

We run the protocols over the Roofnet topology provided at [2]. The losses are given by the Shadowing propagation model and depend on the distances between the nodes. We choose randomly 100 pairs of (source, destination) nodes and compute the routing information in MATLAB using a modified

Algorithm 5 Decoding at destination. The destination N_d receives a linear combination.

```
1: receive coded packet $LC_k$ of flow $f$ from neighbor $N_l$
2: $s_{dl} = id(LC_k)$; $t_{dl} \leftarrow t_{dl} + 1$
3: if $LC_k$ is innovative then
4:     store $LC_k$ in the coding_buffer$_f$
5:     if a packet can be decoded then
6:         deliver the decoded packet to the TCP layer
7:     else
8:         send an ack to the sender
9:     end if
10: else
11:     drop packet $LC_k$
12: end if
```

version of the directed diffusion algorithm, reinforcing up to d paths [10]. By varying the number of reinforced paths d, we vary the number of downstream neighbors that a node can pick as next-hops for the packets. The average path length is of 2.5 hops.

We disable RTS/CTS for all experiments. In order to simulate heavy loss conditions, we also disable MAC layer retransmissions for all the experiments. The rest of the parameters have the default *ns-2.33* values.

We use the following metrics to evaluate our protocol:

- *throughput* measured at destination (decoded rate);
- *efficiency*, defined as the ratio of the goodput at destination to the total throughput transmitted in the network;
- *relative improvement* of *CoMP* over the other protocols, defined as the ratio of the throughput obtained with *CoMP* to the throughput obtained with the other protocols;
- *aggregate throughput* only for the case when multiple flows are running in parallel in the network. The aggregate throughput is computed as the sum of the individual throughput of all flows that run simultaneously.

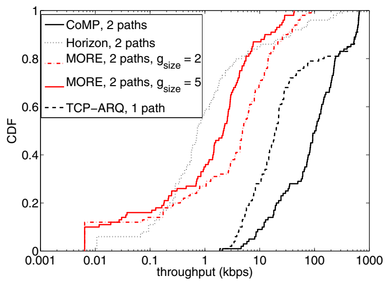
For all the figures, each point is the average of at least 10 runs such that the standard deviations for the different protocols do not overlap.

B. Overall Results

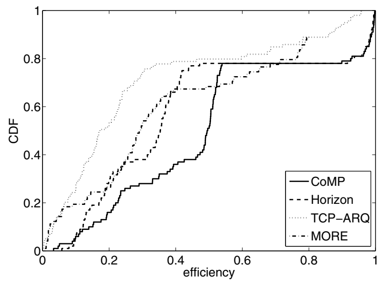
In this section we discuss the results for different scenarios.

1) Multiple paths: In this case, there is only one flow active in the network at a time. Note from Fig. 3(a) that *CoMP* achieves significantly higher throughput than the other protocols, and it can yield an improvement of 10x over *TCP-ARQ* and more for *Horizon* and *MORE* (see Fig. 3(c)). There are about 20% of flows in the network for which the source and the destination are one hop away, and for those flows *TCP-ARQ* has a similar performance to *CoMP*. If the paths are longer, then *TCP-ARQ* is more sensitive to losses. *Horizon* achieves a lower throughput because it relies on TCP retransmissions to recover every lost packet. *MORE*'s throughput decreases significantly if the generation size increases due to the additional delay introduced by coding.

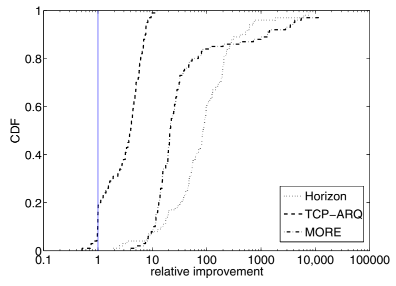
Regarding the efficiency, we compute it as the ratio of the decoded rate at destination to the total rate sent into the network, and therefore for a 2-hop long flow, the maximum efficiency can be equal to 0.5. *CoMP* generally outperforms



(a) Throughput

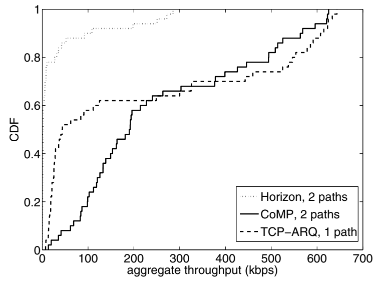


(b) Efficiency

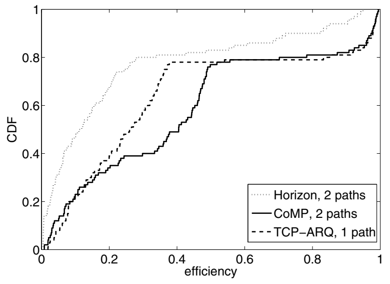


(c) Relative improvement of *CoMP* over the other protocols

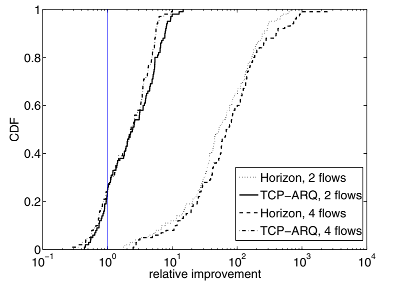
Fig. 3. Roofnet topology with only one flow active in the network. Nodes have at most two downstream neighbors (*i.e.* two reinforced paths, $d = 2$).



(a) Aggregate throughput for 2 flows



(b) Efficiency for 4 parallel flows



(c) Relative improvement of *CoMP* over the other protocols

Fig. 4. Roofnet topology with multiple flows running in parallel. Nodes have at most two downstream neighbors (*i.e.* two reinforced paths, $d = 2$).

the other protocols, except for the 20% flows that are one-hop long and *TCP-ARQ* is equally efficient (see Fig. 3(b)).

2) Multiple flows: In Fig. 4 we show the results for the case when multiple flows run in parallel in the network. Since *MORE*'s performance degrades very quickly as the generation size increases, we do not take it into account for this case.

Notice from Fig. 4(a) that for 60% of the flows, the aggregate throughput obtained when two sessions run concurrently in the network is higher with *CoMP* than with *TCP-ARQ*. This happens for performance-challenged flows, where paths are long and experience high losses. For these situations, using one extra path together with coding hop-by-hop instead of end-to-end increases throughput. On the other hand, for the case when paths are short and possibly physically overlapping, using one more path increases the interference, thus having a negative effect on the throughput obtained with *CoMP*. The effect of the interference can also be noticed from Fig. 4(c), where 20% of flows have a relative improvement less than 1, meaning the throughput is lower with *CoMP* than with *TCP-ARQ*. *Horizon* achieves a throughput close to 0 for almost 80% of the flows, due to the fact that it does not offer any reliability in an extremely volatile environment.

In terms of efficiency, *CoMP* outperforms the other protocols for 55% of the flows. For the 20% of one-hop flows, *TCP-ARQ* achieves equal efficiency, since multiple paths cannot be exploited. There are another 25% of flows for which *CoMP* is slightly less efficient than *TCP-ARQ*, because even if with *CoMP* the destination receives a higher throughput, there are also more packets sent into the network, lowering efficiency.

IV. Conclusions

In this paper we presented *CoMP*, a network coding multipath protocol that improves TCP performance in lossy environments by combining online coding and multipath techniques. *CoMP* uses an online algorithm to estimate current loss conditions in the network and to adjust the rate of sending coded packets. *CoMP* relies on a feedback mechanism for TCP, where destination acknowledges the independent linear combinations that it receives, instead of decoded packets. All these features allow a seamless interaction between *CoMP* and TCP. We show through extensive simulations the advantages of our protocol over the current state-of-the-art in terms of throughput and efficiency.

References

[1] The network simulator - ns-2. http://nsnam.isi.edu/nsnam/index.php/.
[2] The roofnet. http://pdos.csail.mit.edu/roofnet/doku.php.
[3] R. Ahlswede, N. Cai, S.-Y. R. Li, and R. W. Yeung. Network information flow. *IEEE Trans. on Information Theory*, 46(4):1204–1216, July 2000.
[4] H. Balakrishnan, V. N. Padmanabhan, S. Seshan, and R. H. Katz. A comparison of mechanisms for improving tcp performance over wireless links. *ACM/IEEE Transactions on Networking*, 1997.
[5] S. Chachulski, M. Jennings, S. Katti, and D. Katabi. Trading structure for randomness in wireless opportunistic routing. Technical report, MIT, Feb. 2007.
[6] C. Gkantsidis, W. Hu, P. Key, B. Radunovic, P. R. Rodriguez, and S. Gheorghiu. Multipath code casting for wireless mesh networks. In *Proceedings of CoNEXT*, 2007.
[7] B. Radunovic, C. Gkantsidis, D. Gunawardena, and P. Key. Horizon: Balancing tcp over multiple paths in wireless mesh networks. Technical report, Microsoft Research, 2008.
[8] J. K. Sundararajan, D. Shah, M. Medard, M. Mitzenmacher, and J. Barros. Network coding meets tcp. In *Infocom*, 2009.
[9] L. Tassiulas and A. Ephremides. Stability properties of constrained queueing systems and scheduling policies for maximum throughput in multihop radio networks. *IEEE Transactions on Automatic Control*, 37:1936–1948, 1992.
[10] A. L. Toledo and A. X. Wang. Efficient multipath in sensor networks using diffusion and network coding. In *Proceedings of the 40th Annual Conference on Information Sciences and Systems*, pages 87–92, 2006.