Performance Evaluation of Wireless Network Coding under Practical Settings

Ihsan A. Qazi * Pratik Gandhi†
*Department of Computer Science
†School of Information Sciences
University of Pittsburgh, Pittsburgh, PA 15260
{*ihsan, psg11*}*@cs.pitt.edu*

***Abstract*— *Network coding* is a new research area that is likely to have interesting applications in practical networking systems. With network coding, intermediate nodes may send out packets that are linear combinations of previously received information. There are two key benefits of this approach: potential throughput improvements and a high degree of robustness. Traditionally, network coding has been employed in the domain of multicast and broadcast networks. Recently, it has found applications in peer-to-peer and wireless networks. However, the bulk of work on network coding is of theoretical nature and there exists very little experimental work that quantifies the efficacy of this approach in practical environments.**

In this paper, we evaluate the performance of network coding in a wireless network using test-bed experiments. We use a three node chain topology, where each node is equipped with a 802.11 card. Our results show an average throughput gain of 1.2 with network coding. Our insights reveal that the performance of network coding relies heavily on the presence of bi-directional traffic. If the difference in the upload and download traffic loads is negligible, large number of coding oppurtunities may arise, which results in a significant decline in the average queue size and the packet loss rate. We believe that with carefully designed topologies the gains from network coding could be even more and are likely to be significant enough to motivate deployment in APs.

I. Introduction

Wireless networks have become indispensable; they provide the means for mobility, city-wide Internet connectivity, distributed sensing, and outdoor computing. However, current wireless networks support transmission rates which are at least an order of magnitude smaller than the capacity typically available in wired networks. Furthermore, current wireless implementations suffer from throughput limitations and do not scale to large, dense networks.

Network coding is a potential way of increasing the throughput of wireless networks whereby an intermediate node mixes packets from various senders and sends them in a single transmission. This increases the information content of a packet and thus may increase the throughput of the network. In order to get a feel of the idea behind network coding, consider the scenario in Figure. 1, where *Alice* and *Bob* want to exchange a pair of packets via a router. In current approaches, *Alice* sends her packet to the router, which forwards it to *Bob*, and *Bob* sends his packet to the router, which forwards it to *Alice*. This process requires 4 transmissions. Now consider a network coding approach (See Figure 2). *Alice* and *Bob* send their respective packets to the router, which XORs the two packets and broadcasts the XOR-ed version. *Alice* and *Bob* can obtain each others packet by XOR-ing again with their own packet. This process takes 3 transmissions instead of 4. Saved transmissions can be used to send *new* data, increasing the wireless throughput.

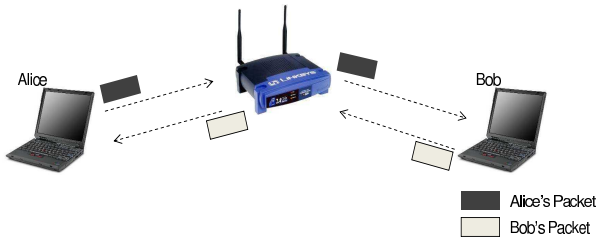

Fig. 1. Scenario without network coding

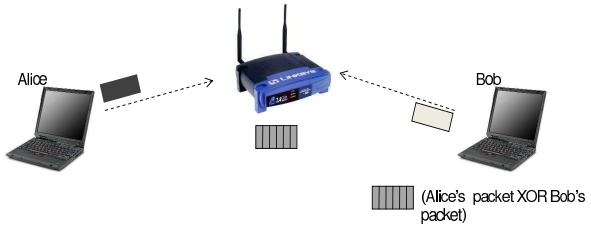

Fig. 2. Scenario with network coding

II. Primer on Network Coding

In network coding, we allow an intermediate node to combine a number of packets it has received or created into one or several outgoing packets. Assume that each packet consists of L bits. When the packets to be combined do not have the same size, the shorter ones are padded with trailing $0s$. We can interpret s consecutive bits of a packet as a symbol over the field F_{2^s}, with each packet consisting of a vector of L/s symbols. With linear network coding, outgoing packets are linear combinations of the original packets, where addition and multiplication are performed over the field F_{2^s}. The reason for

choosing a linear framework is that the algorithms for coding and decoding are well understood [1]..

III. Where can Network Coding be used?

In the following, we list a number of applications of network coding and discuss how its gain could be realized.

A. P2P Networks

Arguably, the most widely known application using network coding is Avalanche [2]. Generally, in a peer-to-peer content distribution network, a server splits a large file into a number of blocks. Peer nodes try to retrieve the original file by downloading blocks from the server but also distributing downloaded blocks among them. To this end, peers maintain connections to a limited number of neighboring peers (randomly selected among the set of peers) with which they exchange blocks. In Avalanche, the blocks sent out by the server are random linear combinations of all original blocks. Similarly, peers send out random linear combinations of all the blocks available to them. A node can either determine how many innovative blocks it can transmit to a neighbor by comparing its own and the neighbor's matrix of decoding coefficients, or it can simply transmit coded blocks until the neighbor receives the first non-innovative block. The node then stops transmitting to this neighbor until it receives further innovative blocks from other nodes. Coding coefficients are transmitted together with the blocks, but since blocks usually have a size of hundreds of kilobytes, this overhead is negligible.

Network coding helps in 1) It minimizes download times 2) Due to the diversity of the coded blocks, a network coding based solution is much more robust in case the server leaves early (before all peers have finished their download) or in the face of high churn rates (where nodes only join for a short period of time or leave immediately after finishing their download) [1].

B. Wireless Networks

Bidirectional traffic in a wireless network: Residential wireless mesh networks: Even a limited form of network coding which only uses *xor* to combine packets may significantly improve network performance in wireless mesh networks. All transmissions are broadcast and are overheard by the neighbors. Packets are annotated with summary information about all other packets a node already heard. This way, information about which nodes hold which packets is distributed within the neighborhood. A node can *xor* multiple packets for different neighbors and send them in a single transmission, if each neighbor already has the remaining information to decode the packet.

Many-to-many broadcast: Network-wide broadcast is used for a number of purposes in ad-hoc networks (e.g., route discovery) and can be implemented much more efficiently with network coding. Already a simple distributed algorithm for random network coding reduces the number of transmission by a factor of 2 or more, leading to significant energy savings. In such a setting, a larger transmit power directly translates into a reduction in the number of required transmissions, which allows for interesting energy trade-offs [1].

IV. Benefits of Network Coding

A. Robustness and Adaptability

A compelling benefit of network coding is in terms of robustness and adaptability. Intuitively, we can think that network coding, similarly to traditional coding, takes information packets and produces encoded packets, where each encoded packet is equally important. Provided we receive a sufficient number of encoded packets, no matter which, we are able to decode. The new twist that network coding brings, is that the linear combining is performed opportunistically over the network, not only at the source node, and thus it is well suited for the (typical) cases where nodes only have incomplete information about the global network state.

B. Throughput Gains

A primary result that sparked the interest in network coding is that it can increase the capacity of a network for multicast flows. More specifically, consider a network that can be represented as a directed graph (typically, this is a wired network). The vertices of the graph correspond to terminals, and the edges of the graph corresponds to channels. Assume that we have M sources, each sending information at some given rate, and N receivers. All receivers are interested in receiving all sources.

In other words, when the N receivers share the network resources, each of them can receive the maximum rate it could hope to receive, even if it were using all the network resources by itself. Thus, network coding can help to better share the available network resources. Network coding may offer throughput benefits not only for multicast flows, but also for other traffic patterns, such as unicast.

V. Practical Wireless Network Coding

The design of a practical wireless system employing network coding must answer some of the following question:

- At which layer should network coding me implemented?
- How should a coding layer be designed?

A. At which layer should coding be implemented?

Wireless is a broadcast medium, creating many opportunities for nodes to overhear packets when they are equipped with omni-directional antennae. However, broadcast exchanges in 802.11 networks are unreliable, which may introduce high packet loss rate especially when the load is high. A mechanism must be chosen to circumvent this problem. A general approach is to use *pseudo-broadcast* whereby reliability is implemented at the coding layer. Unicast packets are still overheard by the nodes (since they are set to be in promiscuous mode) and sent to the coding layer. If the coding layer is just above the MAC layer, the delay of moving packets higher in the protocol stack can be saved and thus implementing reliability at this layer would incur lower cost.

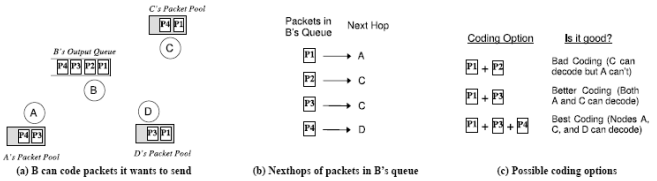

Fig. 3. An example of Opportunistic Coding

B. How should a coding layer be implemented?

When a wireless router (or a node in an Ad hoc network) receives a number of packets for forwarding, then it must answer the following questions:

- Which packets should to encoded?
- How many packets should be encoded?
- When should it encode packets? How long should the router wait before making a coding decision?

The first two decisions depend on what packets have been heard by the nodes in the neighbourhood of the router. Encoding a given packet with $n-1$ other packets requires that each next-hop node must have $n-1$ packets in order for the packet to be decodable. So the first task is to devise a way of getting this information from the neighbouring nodes. This is achieved via reception reports that are periodically sent by the nodes in the neighbourhood, these reports are generally annotated with data packets. The process of listening packets and $reports$ from nodes is called *Opportunistic Listening*. *Opportunistic Coding* refers to the process of making the most efficient decision possible. Since many coding possibilities may exist at any time, the router, however, must make the best coding decision. Figure 3 shows an example of *Opportunistic Coding* in action [3]. Node B has a number of coding choices that it can take. Although coding $P1$ and $P3$ would result in a gain but it is not the best coding decision possible. In fact B can encode $P1$, $P3$ and $P4$ together which would result in an even higher gain. This decision essentially translates into the following rule:

> To transmit n packets, $p_1, ..., p_n$ to n next-hops, $r_1, ..., r_n$, a node can XOR the n packets together only if each next-hop r_i has all $n-1$ packets p_j for $j \neq i$

A router cannot wait for long to encode packets because router queues may grow indefinitely in the meantime, therefore, a coding decision must be made as soon as possible which also requires that the process of encoding be time efficient.

VI. Factors affecting network coding gains

Throughput gain of network coding depends on the existence of coding opportunities, which themselves depend on the traffic patterns. Factors affecting traffic patterns includes 1) Number of flows in the network 2) Kind of traffic (TCP or UDP) 3) Topology 4) Interference and noise 5) Ratio of upload and download traffic rates etc. In this section we will give some intuition behind the effect of each of these factors.

A. Number of flows in the network

When the number of flows in the network is increased, it is highly likely that more coding opportunities would arise. This in effect would result in higher throughput. However, when the number of flows are increased beyond a certain threshold, higher load leads to contention which may result in a higher packet loss rate. Since these packets would also contain reception reports, the loss of which would prevent the intermediate node in making the best possible coding decision. This would impact the overall performance gain due to network coding.

B. Kind of traffic

Applications running over TCP would have different of gains from network coding as opposed to applications that run over UDP. This happens because of TCP's sensitivity to packet loss and reordering. Packet losses and packet reordering forces TCP sender to go into fast retransmit and timeouts which causes them to cut their congestion window sizes into half. This results in a significant reduction in the offered load and thus impacts the number of coding opportunities that may arise. UDP traffic, on the other hand, does not exercise congestion control[1] and thus the offered load does not vary significantly. This causes more coding opportunities and results in higher throughput improvements.

C. Topology

The capacity of general network coding for unicast traffic is still an open question for arbitrary graphs [4].

Figure 4 shows some simple topologies for which the theoretical coding limits are known. For the chain topology shown in Figure 4(a), [3] showed that the gain tends to 2 as the number of intermediate hops increase. The "X' topology has a maximum theoretical gain of 1.33; "Cross" topology has 1.6 whereas the "Wheel" topology has a maximum gain of 2. It should be noted when opportunistic is employed the coding gains may increases as shown by [3].

D. Interference and Noise

The throughput gain of coding depends considerably on the level of interference and noise in the wireless network under consideration. It may be the case that users associated with an AP which uses network coding has many other neighbouring APs which interfere with it. This increases the likelihood of packet losses and that of reception reports.

E. Amount of upload and download traffic

Coding opportunities arise when packets from two or more different nodes traverse an intermediate node. When there is only uni-directional traffic, the coding opportunities arise only between data and acknowledgment packets in the opposite directions. Since ACKs are typically much smaller than data packets, it results in the padding of ACKs with $0s$. Therefore, 1) Information content produced due to coding is reduced

[1] Applications on top of UDP may exercise congestion control

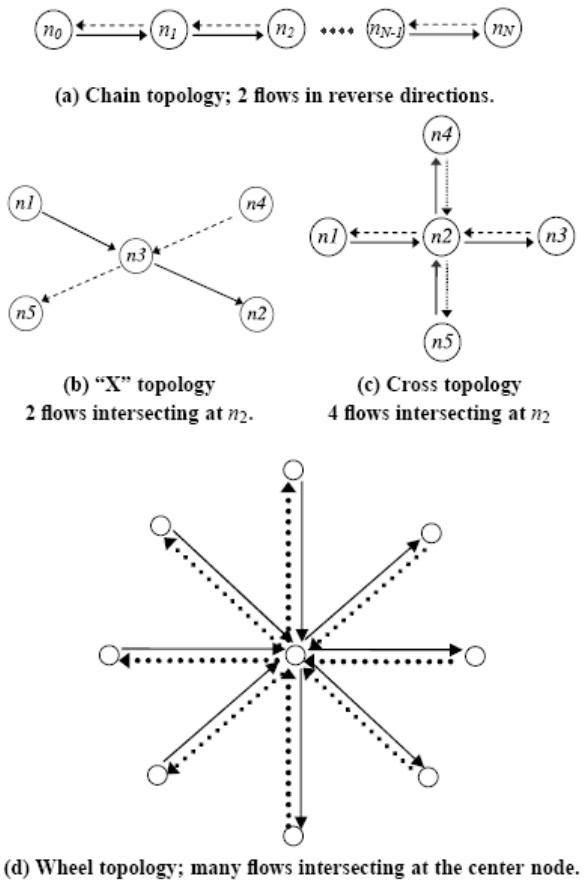

Fig. 4. Simple topologies for understanding the gains of network coding

2) The number of coding opportunities are also reduced. When the upload and the download traffic rates are large and comparable, more coding opportunities arise which results in a significant throughput gain.

F. MAC and Coding layer interactions

How should the coding layer be implemented? The answer to this question question depends on the kind of MAC layer under consideration. For our analysis, we consider the 802.11 MAC protocol since our implementation also uses the same data link layer technology. In 802.11 MAC protocol, unicast exchanges are reliable, however, broadcast is unreliable. Neither any ACKs are sent for broadcast packets nor does a node back-off in the face of high contention. This implies that encoded packets and reception reports are very likely to get lost. This would significantly affect the performance of network coding. Furthermore, TCP traffic would be adversely affected. Therefore, for practical purposes it seems that the coding gains may not be fully realized if normal broadcast is used. The COPE implementation employs *pseudo-broadcast*. In pseudo-broadcast, the encoded packet is sent as a unicast message to one of the nodes. However, since the LAN cards are configured to be in promiscuous mode, they overhear every packet. Since a unicast packet is destined for a single node, the only destination would issue an ACK at the MAC layer. However, other nodes will strip the MAC header[2] but would not invoke the reliability measures of the MAC layer since that would have been the case only if the destination address was for the concerned node. Therefore, reliability for such packets is implemented at the coding layer which resides between the network and data link layer. In this case, each native packet is ACKed and in case of loses, a retransmitted packet may also be encoded.

VII. Experimental Setup

A. Testbed Characteristics

We used the implementation (called COPE) provided by Katti et al. [3] for our real testbed experiments. Nodes in the testbed run Red Hat Linux. COPE is implemented using the Click toolkit[3]. The implementation runs as a user space daemon, and sends and receives raw 802.11 frames from the wireless device using a libpcap-like interface. The implementation exports a network interface to the user that can be treated like any other network device (e.g., eth0). Applications interact with the daemon as they would with a standard network device provided by the Linux kernel. No modifications to the applications are therefore necessary. The implementation is agnostic to upper and lower layer protocols, and can be used by various protocols including UDP and TCP.

The testbed nodes runs the Srcr implementation, a state-of-the-art routing protocol for wireless mesh networks. The protocol uses Dijkstra's shortest path algorithm on a database of link weights based on the ETT metric. Each node in the testbed is a PC equipped with an 802.11 wireless card attached to an omni-directional antenna.

B. Traffic Model

We used a utility program called udpgen [5] to generate UDP traffic for our experiments.

C. Topology

We used the Alice and Bob topology shown in Figure 1 and Figure 2. Due to the shortage of resources, we were restricted to only three nodes in our experiments.

D. Metrics

Our evaluation considers many metrics. The diversity of these metrics allow us to have a deeper insight into the problem of network coding. The metrics that we take into account are as follows:

- *Network Throughput*: It is the sum of the bytes received by all flows in either direction in a given experiment.
- *Throughput Gain*: The ratio of the measured network throughputs with and without wireless network coding
- *Average Queue Size*: This is the average size of the output queue at the coding layer of the bottleneck router.

[2]Under normal circumstances, the MAC layer drops those packets which are not destined for it, unless it is not a broadcast packet

[3]Click is a software router

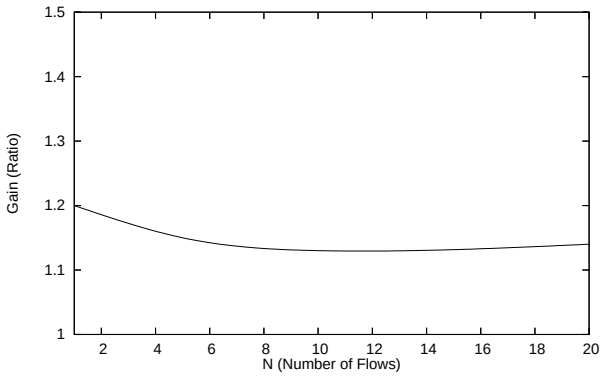

Fig. 5. Throughput gain as function of the number of flows

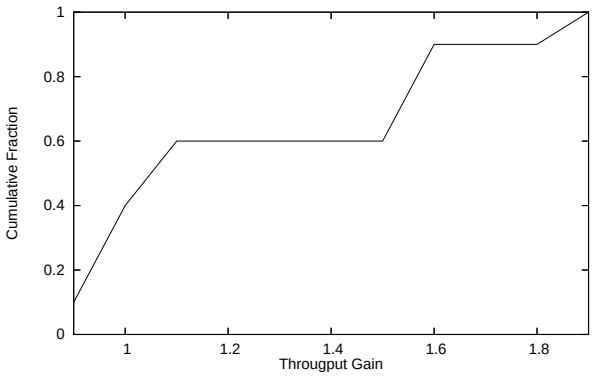

Fig. 6. CDF of the throughput gains in the Alice-Bob Topology

- *Difference between the upload and download traffic rates*: This is the difference in the amount of bytes received in either directions. This metric is important because the gains are very much tied to the degree of asymmetry in traffic in either direction.
- *Number of packets dropped*: This refers to the number of packets dropped at the output queue of the router.

VIII. Performance Evaluation

Our test-bed comprised of three wireless nodes that were arranged in a chain topology. Each node, was placed at equal distance (roughly two meters) from the router. The experiments were performed in a room where few APs [4] were expected to interfere with the transmissions in our network. However, since all experiments were in the same environment, the effects of such interference were not significant even though such interference limited the maximum possible throughput of our network. All experiments were run for 90 seconds and each data point represents the average of 10 runs. The data packet size was 1500 bytes.

A. Throughput Gain with and without Network Coding

The *throughput gain* is defined as:

$$Gain = \frac{T_w}{T_{wo}} \quad (1)$$

where T_w and T_{wo} are the throughputs with and without network coding, respectively.

Figure 5 shows the throughput gain as a function of the number of flows in the network. It can be observed from the figure that the gain stays almost constant at 1.2 across a range of flows. For a 3 node, chain topology, the maximum gain *without opportunistic coding* is 1.33 whereas *with opportunistic coding* it tends to 2 [3]. The average case performance depicted in Figure 5 shows that the gain is very close to the theoretical limit of 1.33. Some loss in gain occurs because of the overhead incurred due to the coding of packets. It should be noted that for some individual runs, we observed gains as high as 1.85, which matches with our intuition of the gains when opportunistic coding is used[5]. This can be seen in Figure 6 which shows the cumulative fraction of the number of flows as a function of the throughput gain. Although the average throughput gain did lie around 1.2 but the gain for some runs was close to 2 and for others it was close to 1. The runs resulting in a gain of value slightly less than 1[6] was used) may have occurred due to higher packet loss rate, resulting in a decline of throughput due to retransmissions of native packets[7]. Our intuition indicates that running experiments for a longer period of time is likely to remove these outliers (which were not significant) since the overhead due to retransmissions would be amortized in that case.

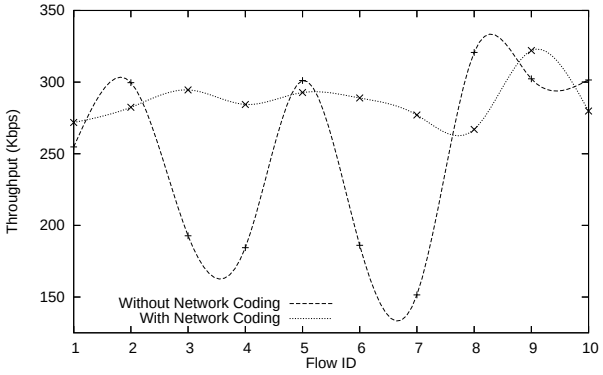

Fig. 7. Throughput variations with and without wireless network coding

We also noticed a considerable variation in the throughput when network coding wasn't used. This maybe due to the fluctuations in the interference in our experiment location. Network Coding on the other hand is able to mask such fluctuations and thus results in a more sustained throughput as shown in Figure 7.

[4]These access points are deployed by the Department of Computer Science at the University of Pittsburgh

[5]The implementation that we used for our experiments makes use of *opportunistic coding*

[6]which means that the throughput was In fact reduced when networking coding

[7]Native packets are un-encoded packets

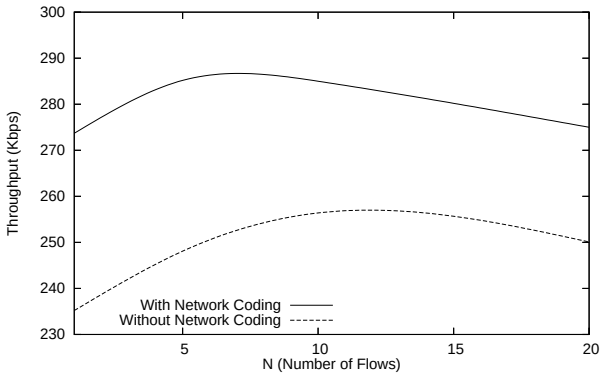

Fig. 8. Throughput as a function of the number of flows

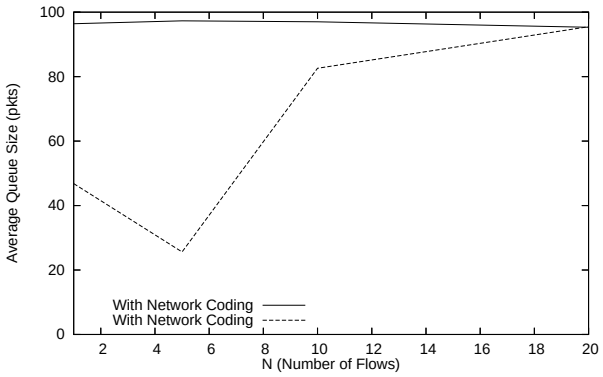

Fig. 9. Queue Size as a function of the number of flows

B. Throughput as a function of the number of flows

Figure 8 shows the throughput as a function of the number of flows in the system. It can be seen from the graph that the throughput gain remains always unchanged as the number of flows increased from 1 (in either direction) to 20 (in either direction). The difference remains at roughly 30Kbps. The overall throughput was below 350 Kbps. This happens because of two reasons:

- In our experiments, nodes were arranged such that there were no hidden terminals. and the cards were configured to send at a rate of 1Mbps.
- The 802.11 MAC roughly divides the rate evenly among the competing nodes which implies that each node should get around 333Kbps. This is what we observed in our experiments.

Furthermore, we did observe reasonable asymmetry in the amount of upload and download traffic. This unfairness occurred because of the comparative quality of the channels from the sources of the bottleneck which is generally referred to as the *capture effect*. For example, in our experiment we observed that at times the channel between Alice and router was better than the channel between Bob and router, which at times made bob unable to push the same amount of traffic as Alice.

C. Queue size with and without network coding

Figure 9 shows the average queue size at the bottleneck link router, both with and without network coding. When network coding is not used, the average queue remains almost full (the maximum queue size was set at 100 packets). This occurs because the router receives packets from both Alice and Bob each of whom are sending at an average rate of 333 Kbps. However, since the router is also only able to send at an average rate of 333Kbps, it results in the queues building up. This causes packet losses to occur and thus results in a considerably smaller throughput than with network coding.

Network coding, on the other hand, is able to derive considerable benefit from the *full queues* behaviour. This happens because larger queues help the router in coding larger number of packets together. Since, encoding large of packets together not only results in higher throughput but also results in the eviction of the native packets which considerably reduces the average queue size. This can be see in Figure 9 where the average queue size with 5 flows (in either direction, with a total of 10 flows in the network), resulted in a average queue size of 25 packets as compared to an average queue size of 100 packets without network coding.

However, as the number of flows are increases in the network, contention also increases which tends to increase the packet loss and thus the number of retransmissions. Although, retransmitted packets are also allowed to be encoded but the higher packet loss rate prevents the throughput from increasing even further with larger number of flows in the network. We believe that with implementation that caters for reliability at the MAC, one can achieve better results.

D. Queue size as a function of the upload and download traffic loads

Coding opportunities are significantly more when the upload and download traffic loads are comparable. Note that coding can only take place in the presence of bi-directional traffic. Figure 10 shows the average queue size as a function of the difference in the upload and download traffic loads. As the difference increases the average queue size increases. This happens because with large differences in the traffic in the two directions, the behaviour of the average queue size approaches that of without network coding since very few coding opportunities arise. On the other hand, when the difference is almost negligible, the average queue size is nearly zero. That is why no packet loss was observed for such cases (see Figure 11

E. Packet drop rate with and without network coding

Figure 12 shows the number of packet dropped as a function of the number of flows in the network. Up to 10 flows (in either direction), the packet drops are less when network coding is used. However, when the number of flows are increased even further, the drops with coding increase even higher than without network coding. This happens because the implementation of coding that we are using uses asynchronous hop by hop ACKs for each native packet that is transmitted in

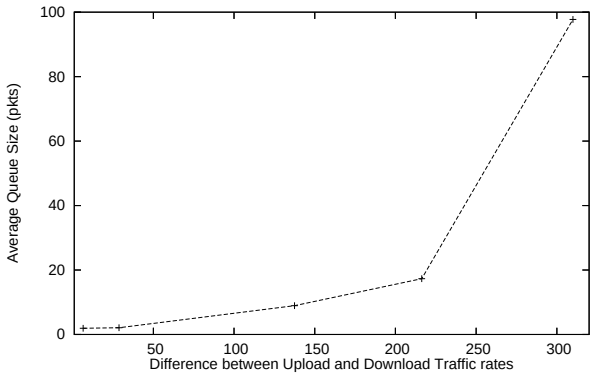

Fig. 10. Queue Size as a function of the difference between the upload and download traffic loads

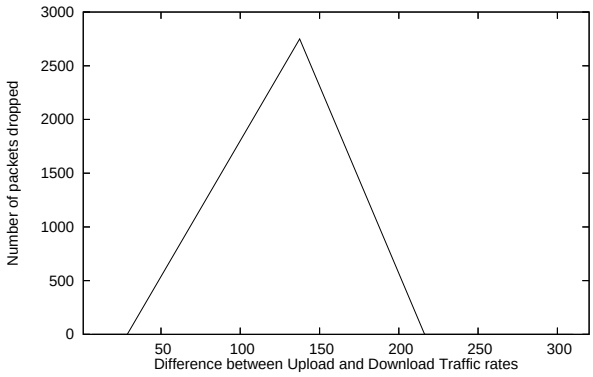

Fig. 11. Packet drops as a function of the difference between the upload and download traffic rates

the encoded packet. This results in a considerably higher ACK traffic than with network coding. This also tends to impact the total number of packet drops.

IX. Related Work

Research on network coding started with a pioneering paper by Ahlswede et al. [6], who showed that having the routers mix information in different messages allows the communication to achieve multicast capacity. This was soon followed by the work of Li et al. [7], who showed that, for multicast traffic, linear codes are sufficient to achieve the maximum capacity bounds. Koetter and Medard [8] presented polynomial time algorithms for encoding and decoding, and Ho et al [9]. extended these results to random codes. Lun et al. [10] studied network coding in the presence of omni-directional antennae and showed that the problem of minimizing the communication cost can be formulated as a linear program and solved in a distributed manner. All of this work is primarily theoretical and assumes multicast traffic. A few papers study specific unicast topologies showing that, for the studied scenario, network coding results in better throughput than pure forwarding [11], [3].

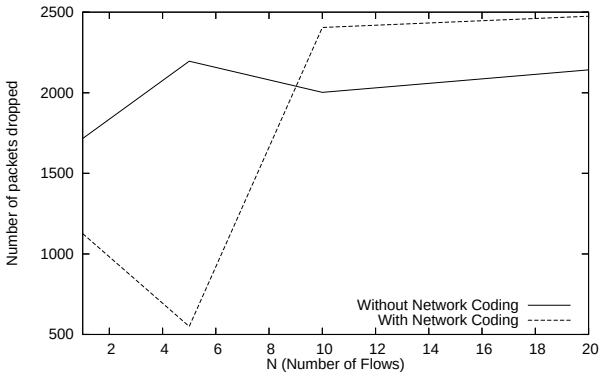

Fig. 12. Packet drops as a function of the number of flows

X. Future Work

In this work, we only experimented with a chain topology, comprising of 3 nodes. In the future, we would like to experiment with different topologies so as to better appreciate the relationship of coding gains with different network topologies. We would also like to dwell into theoretical study related to network coding, since many questions like "What is the capacity of general graphs for the case of unicast traffic when network coding is used?" are still unanswered [3].

XI. Conclusion

In this paper, we looked into the efficacy of network coding from a practical perspective. We used the COPE implementation [3] of network coding for our experiments. The throughout gain for a 3 node, chain topology was around 1.2, although we did see the gains from individual runs very close to 2. Our results also shows that the coding gains are closely tied with the ratio of the upload traffic to the download traffic As the difference increases, less coding opportunities arise, however, when the difference is negligible, more coding opportunities arise which also results in a dramatic decline in the queue size and packet drop rate. Our results show that network coding gains are significant enough to motivate a practical of deployment of the paradigm in APs.

References

[1] C. Fragouli, J.-Y. L. Boudec, and J. Widmer, "Network Coding: An Instant Primer," in *ACM SIGCOMM CCR*, 2006.
[2] C. Gkantsidis and P. Rodriguez, "Network coding for large scale content distribution," in *IEEE INFOCOM*, Mar 2005.
[3] S. Katti, H. Rahul, W. Hu, D. Katabi, M. Mdard, and J. Crowcroft, "XORs in the air practical wireless network coding," in *Proceedings of ACM SIGCOMM*, 2006.
[4] Z. Li and B. Li, "Network Coding in Undirected Networks," in *CISS*, 2004.
[5] "http://pdos.csail.mit.edu/click/ex/udpgen.html."
[6] R. Ahlswede, N. Cai, S. R. Li, and R. W. Yeung., "Network Information Flow," in *IEEE Transactions on Information Theory*, 2000.
[7] S. R. Li, R. W. Yeung, and N. Cai, "Linear network coding," in *IEEE Transactions on Information Theory*, 2003.
[8] R. Koetter and M. Mdard, "An algebraic approach to network coding," in *IEEE/ACM Transactions on Networking*, 2003.
[9] T. Ho and R. Koetter, "Online incremental network coding for multiple unicasts," in *DIMACS Working Group on Network Coding*, 2005.

[10] D. S. Lun, M. Mdard, R. Koetter, and M. Effros, "Further results on coding for reliable communication over packet networks," in *IEEE International Symposium on Information Theory*, 2005.
[11] Z. Li and B. Li, "Network coding: The case for multiple unicast sessions," in *Allerton Conference on Communications*, 2004.

XII. Appendix

You will need Linux[8], a compatible **802.11a/b/g card**[9], the **Click modular router**, **Madwifi source code**[10] (madwifi-ng) and the **Roofnet source code** to build COPE. Click is a software router that runs on each node. Wireless LAN cards that are based on the Atheros chipset require a special driver called madwifi-ng (where ng is the 'next generation' version of this driver) which is available online for no cost. Roofnet provides the routing protocol which the click router runs. COPE uses the features provided by the device driver to configure the LAN card according to the needs of the experiments. Therefore, the first task is to build and install the driver.

A. Building and Installing Madwifi-ng

After downloading madwifi-ng, enter the madwifi-ng directory. Read INSTALL; you may need to install a kernel from sources. While you're at it, make sure your kernel has /dev/tun support. Build the drivers:

- make all
- make install

B. Installing Click and COPE

To build COPE, enter the click directory (we will call this directory CLICKDIR from now on) and run the following commands:

- ./configure −−enable-userlevel −−enable-wifi −−disable-linuxmodule −−enable-roofnet
- make all
- make install

You may need gcc-3.4 and g++-3.4. Load the newly built drivers for the wireless cards:

- modprobe ath_pci

C. Loading Roofnet Configuration

Finally, load the Roofnet configuration

- cd click/conf/wifi/
- ./gen_config_cope.pl −−dev ath0 | CLICKDIR/userlevel/click −

[8]We used the Enterprise version of Red Hat Linux

[9]The wireless card must have either a Atheros-based chipset or Intersil Prism 2.5-based chipset. Otherwise, roofnet and cope wouldn't work. We used the LINKSYS, dual-band wireless adapter for our experiments

[10]This is the driver for the wireless cards that are based on the Atheros chip-set. Remember to use the ng(next-generation) version of madwifi

D. How to see if it's working

At this point running /sbin/ifconfig should show some new tap interfaces. The srcr interface, which is configured as 5.X.X.X, uses multi-hop routing. The lowest 24-bits of both addresses are based on the wireless device's hardware address. To view statistics about broadcast probes receiving from nearby nodes run,

- CLICKDIR/conf/wifi/read_handler.pl srcr/es.bcast_stats

You can also view known routes using

- CLICKDIR/conf/wifi/read_handler.pl srcr/lt.routes

You may need apt-get install netcat to make read_handler work. COPE is not enabled by default. You can turn it on via

- CLICKDIR/conf/wifi/write_handler.pl srcr/scramble_q.enable_coding true

E. Experimental verification

The experiment is as follows. Alice is sending packets to Bob via the router in the middle and similarly Bob is sending packets to Alice via the router. Take 3 wireless nodes - Alice, Router and Bob and arrange them in a straight line such that Alice and Bob are equidistant from the router like a typical 3-node topology (say 20 metres, we can vary this later). Collect their IP addresses, we will call them respectively ALICEIP, ROUTERIP, BOBIP. The script[11] assumes the following format:

- python run_expt.py ALICEIP ROUTERIP BOBIP true,false

The 4 arguments for the script are the IP addresses of Alice, Router and Bob machines in that particular order and the last argument is whether you want to run the experiment with COPE enabled or disabled, if 'true' it is enabled and so on. So for example if the IP addresses are as follows

Name IP
Alice 130.49.223.182
Router 130.49.223.197
Bob 130.49.223.207

we would run

- python run_expt.py 128.30.2.21 128.30.2.22 128.30.2.23 false

which would execute the Alice Bob experiment with COPE disabled. The experiment will run for approximately 90 seconds, and at the end will report throughput numbers. Now if you run the experiment with COPE enabled you should see a significant increase in throughput compared to without COPE.

The gains will vary depending on your ambient traffic conditions, environment etc, but should be around 40-100% increase in overall throughput. Now if you move the nodes around, the gains will start varying. If you dont see any gains, the usual reason is that one of the nodes among Alice or Bob has a very strong link to the router, then the node will capture the whole route and obtain most of the throughput. This will minimize the coding opportunities and hence reduce the gain.

[11]Available in the click/conf/wifi directory

The other reason could be that Alice and Bob are hidden nodes to each other and their packets are colliding when they transmit simultaneously. This will significantly reduce the number of coding opportunities at the router and hence reduce the gain COPE provides.