Πανεπιστήμιο Θεσσαλίας
Πολυτεχνική Σχολή
Τμήμα Μηχανικών Ηλεκτρονικών Υπολογιστών,
Τηλεπικοινωνιών & Δικτύων

Κωδικοποίηση Δικτύου σε Ασύρματο Ad-Hoc Περιβάλλον

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Δελήμπασης Δημοσθένης

<u>Επιβλέποντες Καθηγητές</u>

Τασιούλας Λέανδρος

Πάσχος Γεώργιος

Βόλος, Σεπτέμβριος 2011

ΕΥΧΑΡΙΣΤΙΕΣ

Αρχικά, θα ήθελα να ευχαριστήσω θερμά τους καθηγητές κ. Γεώργιο Πάσχο και κ. Λέανδρο Τασιούλα για την αμέριστη συμπαράσταση, τις πολύτιμες συμβουλές και την καθοδήγησή τους. Επιπλέον να ευχαριστήσω τους συναδέλφους στο εργαστήριο τηλεπικοινωνιών του τμήματος, για την άμεση υποστήριξη και βοήθειά τους.

Τέλος ένα πολύ μεγάλο ευχαριστώ χρωστάω στην οικογένεια μου που ήταν πάντα εκεί για να με στηρίζει σε όλες τις αποφάσεις μου και να με εμψυχώνει με κάθε τρόπο.

ΕΙΣΑΓΩΓΗ

Οι τεχνολογικές εξελίξεις στον τομέα των ασυρμάτων δικτύων αλλάζουν θεμελιωδώς τον τρόπο με τον οποίο οι συσκευές επικοινωνούν μεταξύ τους. Οι σύγχρονες ασύρματες συσκευές μπορούν να σχηματίζουν δίκτυα, να αλλάζουν δυναμικά τις συνδέσεις μεταξύ τους ανάλογα με τις συνθήκες και να συνεργάζονται για να μεταφέρουν την πληροφορία αποδοτικά. Σε αυτά τα δίκτυα, η αύξηση της ρυθμαπόδοσης (throughput) αλλά και η διατήρηση της ευρωστία τους είναι σημαντικά κριτήρια καλής λειτουργίας.

Η κωδικοποίηση δικτύου (Network Coding) είναι μια τεχνική με την οποία επιτυγχάνεται αξιοσημείωτη αύξηση της ρυθμαπόδοσης. Στα παραδοσιακά δίκτυα η ροή πληροφορίας ακολουθεί την ίδια αρχή με τη τη ροή υγρού μέσα σε σωλήνες. Δηλαδή οι ενδιάμεσοι κόμβοι προωθούν την πληροφορία που φτάνει σε αυτούς, στους επόμενους κόμβους. Η τεχνική της κωδικοποίησης δικτύου αλλάζει αυτή τη θεώρηση και έτσι πλέον οι ενδιάμεσοι κόμβοι μπορούν να συνδυάζουν διαφορετικές (ανεξάρτητες) ροές πληροφορίας μεταξύ τους και να προωθούν πλέον τη συνδυασμένη αυτή ροή. Ο κόμβος στον οποίο προορίζονταν η αρχική ροή πληροφορίας, μπορεί να την επανακτήσει αυτούσια. Με αυτό τον τρόπο μειώνεται ο αριθμός τον μεταδόσεων και έτσι υπάρχει κέρδος στο throughput σε σχέση με την απλή περίπτωση.

Σκοπός της παρούσας διπλωματικής εργασίας είναι να μελετήσει την πρακτική υλοποίηση του NC σε πραγματικό ασύρματο δικτυακό περιβάλλον. Το σημαντικότερο αποτέλεσμα αυτής της εργασίας είναι η υλοποίηση ενός βέλτιστου αλγορίθμου δρομολόγησης για κωδικοποίηση δικτύου. Η υλοποίησή αυτή έγινε σε πραγματικές συσκευές δικτύου (ασύρματους κόμβους) της πλατφόρμας δικτυακών πειραμάτων (testbed) του τμήματος. Στα πλαίσια της ενασχόλησης αυτής κατασκευάστηκε ένας μηχανισμός παροχής ανάδρασης στον κεντρικό κόμβο (router) από τους πλευρικούς κόμβους. Ο κεντρικός χρονοπρογραμματιστής (scheduler) που λειτουργεί στο router συνδυάζει πακέτα προς αποστολή με μερική μόνο γνώση (πιθανοτικά) για το αν οι δέκτες μπορούν πράγματι να τα αποκωδικοποιήσουν. Στη συνέχεια και εφόσον η αποκωδικοποίηση απέτυχε, ο μηχανισμός παρέχει την απαραίτητη γνώση ανάδρασης στον χρονοπρογραμματιστή ώστε να παρθούν οι βέλτιστες αποφάσεις. Τελικά, η διαδικασία αυτή οδηγεί στην επίτευξη της μέγιστης ρυθμαπόδοσης σε περιβάλλον πιθανοτικής γνώσης (για την ικανότητα αποκωδικοποίησης). Η σημαντικότητα του επιτεύγματος αυτού, έγκειται στο γεγονός ότι η ντετερμινιστική γνώση της δυνατότητας αποκωδικοποίησης είναι συχνά αδύνατη (υπολογιστικά ασύμφορη). Εν αντιθέσει ο υλοποιημένος αλγόριθμος πετυχαίνει μια σημαντική βελτίωση στην απόδοση του συστήματος (το throughput του συστήματος ταυτίζεται μεγάλο βαθμό με αυτό που είχε προβλεφθεί θεωρητικά) ενώ παράλληλα ο υπολογιστικός φόρτος διατηρείται σε χαμηλά επίπεδα.

Το κύριο σώμα της εργασίας ακολουθεί ως παράρτημα.

Contents

1 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** (NC) is a new research area (a field of information theory and coding theory) 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. NC can be used to attain the maximum possible information flow in a network. 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. Coding in packet networks can be classified into two types: intra-session coding (where coding is restricted to packets belonging to the same session or connection) and inter-session coding (where this restriction is lifted and coding is allowed among packets belonging to possibly different sessions). The former, which is also referred to as superposition coding, has been extensively studied. It is well-known that intra-session coding improves the throughput of lossless multicast sessions and of lossy sessions (unicast or multicast). It is also known, however, that intra-session coding is suboptimal [1] and inter-session coding is necessary to achieve optimal throughput in general. In the present diploma thesis we will deal with practical inter-session Network Coding in wireless mesh networks. In particular we will present two cases of practical inter-session NC implementations (COPE and NCRAWL) and we will extend the later, by implementing an optimal throughput scheduling algorithm.

2 Fundamentals of Wireless Network Coding Schemes

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. Linear combination is not concatenation: if we linearly combine packets of length L, the resulting encoded packet also has size L. In contrast to concatenation, each encoded packet contains only a fraction of the information contained in original packets. One can think of linear Network Coding as a form of information spreading. The procedure of linear NC is as follows:

- **Encoding**
  Assume that a number of original packets $M^1, ..., M^n$ are generated by one or several sources. In linear Network Coding, each packet through the network is associated with a sequence of coefficients $g_1, ..., g_n$ in F_{2^s} and is equal to $X = \sum_{i=1}^{n} g_i M^i$. The summation has to occur for every symbol position, i.e., $X_k = \sum_{i=1}^{n} g_i M_k^i$, where M_k and X_k is the kth symbol of M and X respectively. For simplicity, we assume that a packet contains both the coefficients $g = (g_1, ..., g_n)$, called encoding vector, and the encoded data $X = \sum_{i=1}^{n} g_i M_i$, called information vector. The encoding vector is used by recipients to decode the data. For example, the encoding vector $e_i = (0, ..., 0, 1, 0, ...0)$, where the 1 is at the ith position, means that the information vector is equal to M_i (i.e., is not encoded). Encoding can be performed recursively, namely, to already encoded packets. Consider a node that has received and stored a set $(g^1, X^1), ..., (g^m, X^m)$ of encoded packets, where g^j [resp. X^j] is the encoding [resp. information] vector of the jth packet. This node may generate a new encoded packet (g', X') by picking a set of coefficients $h_1, ..., h_m$ and computing the linear combination $X' = \sum_{j=1}^{m} h_j X^j$ the corresponding encoding vector g' is not simply equal to h, since the coefficients are with respect to the original packets $M^1, ..., M^n$; in contrast, straightforward algebra shows that it is given by $g'_i = \sum_{j=1}^{m} h_j g_i^j$. This operation may be repeated at several nodes in the network.

- **Decoding**
  Assume a node has received the set $(g^1, X^1), ..., (g^m, Xm)$. In order to retrieve the original packets, it needs to solve the system $X^j = \sum_{i=1}^{n} g_i^j M^i$ (where the unknowns are the M^i). This is a linear system with m equations and n unknowns. We need $m \geq n$ to have a chance of recovering all data, i.e. the number of received packets needs to be at least as large as the number of original packets. Conversely, the condition $m \geq n$ is not sufficient, as some of the combinations might be linearly dependent.

As one can see above, the general linear NC demands elaborate calculations, thus because we are interested in practical NC schemes we will focus on a limited form of Network Coding which only uses XOR to combine packets. All transmissions are broadcasted 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 encode multiple packets for different neighbors and send them in a single transmission, if each neighbor already has the remaining information to decode the packet. A simple scenario without NC (a) and using NC (b) is illustrated using the Alice–Relay-Bob topology shown in Figure 1. In case (a) four transmissions take place but in (b) only three.
In this point we introduce some usefull terminology, shown in Figure 2.

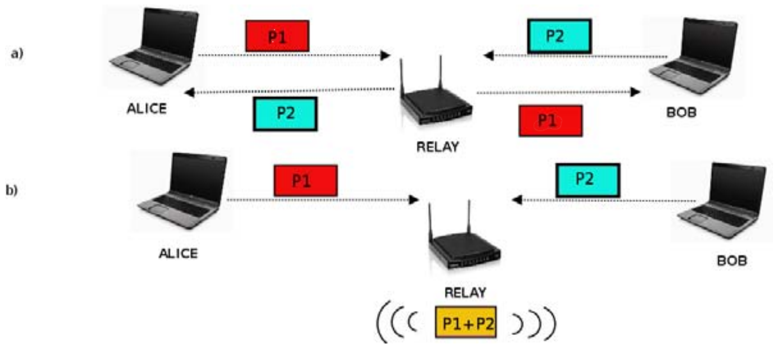
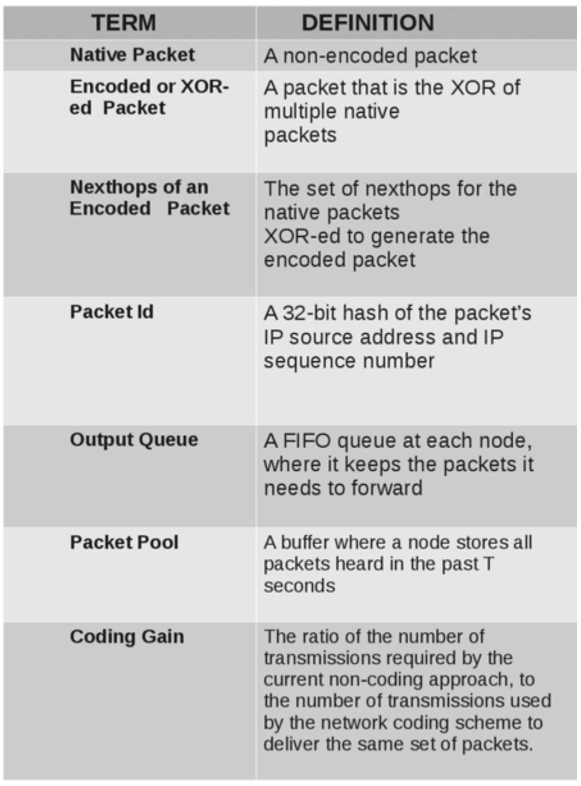


Figure 1: Alice-Relay-Bob topology.



TERM	DEFINITION
Native Packet	A non-encoded packet
Encoded or XOR-ed Packet	A packet that is the XOR of multiple native packets
Nexthops of an Encoded Packet	The set of nexthops for the native packets XOR-ed to generate the encoded packet
Packet Id	A 32-bit hash of the packet's IP source address and IP sequence number
Output Queue	A FIFO queue at each node, where it keeps the packets it needs to forward
Packet Pool	A buffer where a node stores all packets heard in the past T seconds
Coding Gain	The ratio of the number of transmissions required by the current non-coding approach, to the number of transmissions used by the network coding scheme to deliver the same set of packets.

Figure 2: Basic Practical NC Terms.

2.1 Practical Wireless Network Coding

To enable a practical application of NC to multi-hop wireless networks, one needs to address the following issues:

- **Network Coding for unicast applications**: though most of the theoretical results in NC are for multicast, the vast majority of Internet traffic is unicast. An application of NC to the wireless environment has to address multiple unicast flows, if it has any chance of being used. In particular, with multicast, all receivers want all packets. Thus intermediate nodes can encode any packets together, without worrying about decoding which will happen eventually at the destinations. In contrast, in unicast, packets from multiple flows may get encoded together at some intermediate node, but later their paths may diverge, at which point they need to be decoded. If not, unneeded data will be forwarded to area where there is no interested receiver, wasting much bandwidth.

- **Coping with bursty traffic and dynamic environments**: prior theoretical work on Network Coding shows that if the senders, the receivers, and the traffic demands are known a priori, it is possible to run a distributed optimization to find the optimal coding strategy [6]. In reality, users start transmitting immediately without allowing time for route optimization to converge. Further, the traffic is usually bursty and the set of senders and receivers keeps changing over time.

- **Broadcast with collision avoidance**: in wireless environments, NC relies on the broadcast nature of the medium to deliver a single encoded packet to multiple receivers. However, in contrast to unicast, 802.11 broadcast has no collision detection or avoidance mechanism. As a result, broadcast works badly in congested environments where the collision probability is high. However, these are the exact environments that benefit from Network Coding and its ability to send more information for less bandwidth. One may change the MAC layer completely, but for the short term it may be more desirable to make network coding work with 802.11 as this allows for a practical implementation using off-the-shelf hardware/drivers.

- **Low complexity encoding and decoding**: traditional Network Coding uses operations over large finite fields. Decoding operations have quadratic complexity, which becomes too slow for high throughput applications. Further encoding operations are also complicated since they involve multiplications in large finite fields. This makes their use in high throughput applications questionable. Encoding/decoding algorithms should have linear complexity for practical implementation.

Wireless is a broadcast medium, creating many opportunities for nodes to overhear packets when they are equipped with omni-directional antennas. 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. 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. The process of listening packets 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 (router may forward packets without coding if it will arise no gain doing so).

A coding decision (taken by the router) must be made as soon as possible, a fact that requires that the process of encoding must be time efficient. Throughput gain of NC depends on the existence of coding opportunities, which themselves depend on the traffic patterns. Factors affecting traffic patterns include: 1) Number of flows in the network 2) Kind of traffic (TCP or UDP) 3) Interference and noise 4) Ratio of upload and download traffic rates etc 5) Topology . More specifically:

1. **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 is increased beyond a certain threshold, the high 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 NC.

2. **Kind of traffic**
   Applications running over TCP would have different of gains from NC 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. Concerning UDP traffic, on the other hand, it does not exercise congestion control and thus more coding opportunities arise, resulting in higher throughput improvements.

3. **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.

4. **Ratio of upload and download traffic rates**
   Coding opportunities arise when packets from two or more different nodes traverse an intermediate node. When there is only unidirectional 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:

   - Information content produced due to coding is reduced

- 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

5. **Topology**
   The capacity of general NC for unicast traffic is still an open question for arbitrary graphs [4]. Figure 3 shows some simple topologies.

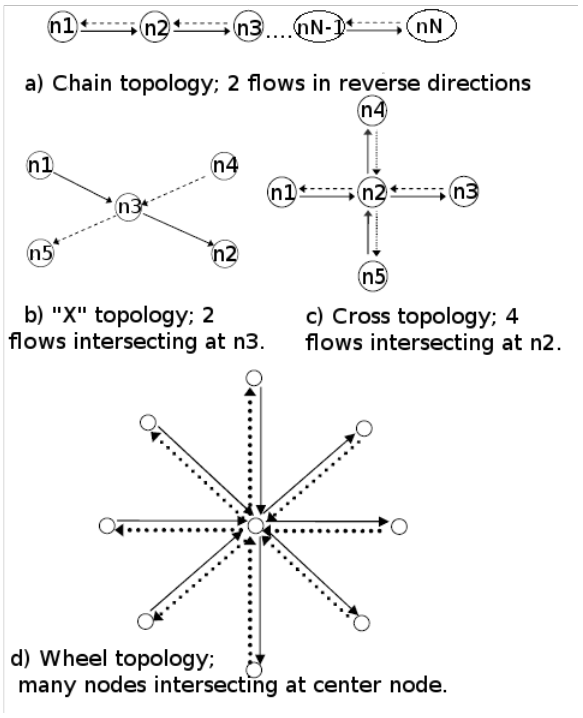

n1
n2
n3
nN-1
nN
a) Chain topology; 2 flows in reverse directions
n1
n4
n3
n5
n2
b) "X" topology; 2 flows intersecting at n3.
n4
n1
n2
n3
n5
c) Cross topology; 4 flows intersecting at n2.
d) Wheel topology; many nodes intersecting at center node.

Figure 3: Various NC topologies.

For the chain topology shown in Figure 3(a) 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. Finally the alice-relay-bob scenario depicted in Figure 1 has the same gain as the “X” topology. It should be noted that when opportunistic is employed the coding gains may increase as shown by [3].

2.2 Description of COPE Scheme

COPE [3] is a NC architecture for wireless mesh networks. It inserts a coding layer between the IP and MAC layers, which detects coding opportunities and exploits them to forward multiple packets in a single transmission. COPE incorporates three main techniques:
(a) Opportunistic Listening: wireless is a broadcast medium, creating many opportunities for nodes to overhear packets when they are equipped

with omni-directional antennae. COPE sets the nodes in promiscuous mode, makes them snoop on all communications over the wireless medium and store the overheard packets for a limited period T (the default is T = 0.5s). In addition, each node broadcasts reception reports to tell its neighbors which packets it has stored. Reception reports are sent by annotating the data packets the node transmits. A node that has no data packets to transmit periodically sends the reception reports in special control packets.
(b) Opportunistic Coding: the key question is what packets to code together to maximize throughput. A node may have multiple options, but it should aim to maximize the number of native packets delivered in a single transmission, while ensuring that each intended nexthop has enough information to decode its native packet. The above is best illustrated with an example. In Figure 4(a), node B has 4 packets in its output queue p1 , p2 , p3 , and p4 . Its neighbors have overheard some of these packets. The table in Fig 4(b) shows the nexthop of each packet in B's queue. When the MAC permits B to transmit, B takes packet p1 from the head of the queue. Assuming that B knows which packets each neighbor has, it has a few coding options as shown in Figure 4(c). It could send $p1 \oplus p2$. Since node C has p1 in store, it could XOR p1 with $p1 \oplus p2$ to obtain the native packet sent to it (i.e p2). However, node A does not have p2 , and so cannot decode the XOR-ed packet. Thus, sending $p1 \oplus p2$ would be a bad coding decision for B, because only one neighbor can benefit from this transmission. The second option in Figure 4(c) shows a better coding decision for B. Sending

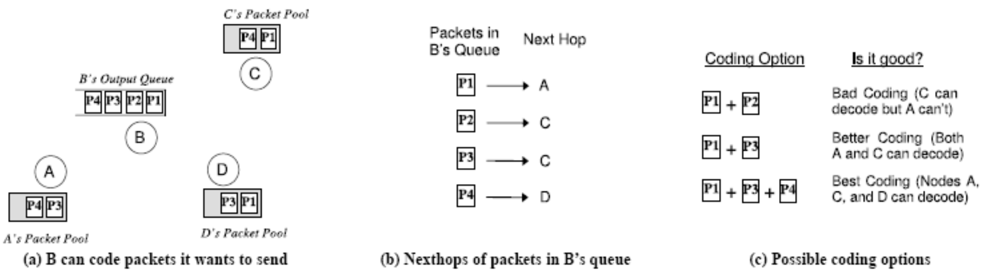

C's Packet Pool
P4 P1
C
B's Output Queue
P4 P3 P2 P1
B
A
D
P4 P3
P3 P1
A's Packet Pool
D's Packet Pool
(a) B can code packets it wants to send
Packets in B's Queue
Next Hop
P1 → A
P2 → C
P3 → C
P4 → D
(b) Nexthops of packets in B's queue
Coding Option
Is it good?
P1 + P2
Bad Coding (C can decode but A can't)
P1 + P3
Better Coding (Both A and C can decode)
P1 + P3 + P4
Best Coding (Nodes A, C, and D can decode)
(c) Possible coding options

Figure 4: Opportunistic Coding in action.

$p1 \oplus p3$ would allow both neighbors C and A to decode and obtain their intended packets from a single transmission. Yet the best coding decision for B would be to send $p1 \oplus p3 \oplus p4$, which would allow all three neighbors to receive their respective packets all at once. The above example emphasizes an important coding issue. Packets from multiple unicast flows may get encoded together at some intermediate hop. But their paths may diverge at the nexthop, at which point they need to be decoded. If not, unneeded data will be forwarded to areas where there is no interested receiver, wasting much capacity. The coding algorithm should ensure that all nexthops of an encoded packet can decode their corresponding native packets. This can be achieved using the following simple rule:

To transmit n packets, $p_1, \ldots, p_n$ to n next-hops, $r_1, \ldots, r_n$, a node can XOR the n packets together only if each next-hop r_i has all $n-1$ packets

p_j for $j = i$.

This rule ensures that each nexthop can decode the XOR-ed version to extract its native packet. Whenever a node has a chance to transmit a packet, it chooses the largest n that satisfies the above rule to maximize the benefit of coding.
(c) Learning Neighbor State: how does a node know what packets its neighbors have? As explained earlier, each node announces to its neighbors the packets it stores in reception reports. However, at times of severe congestion, reception reports may get lost in collisions, while at times of light traffic, they may arrive too late, after the node has already made a suboptimal coding decision. Therefore, a node cannot rely solely on reception reports. So there is also adopted an alternative aproach. Wireless routing protocols compute the delivery probability between every pair of nodes and use it to identify good paths. For e.g., the ETX metric [5] periodically computes the delivery probabilities and assigns each link a weight equal to 1/(delivery probability). These weights are broadcasted to all nodes in the network and used by a link-state routing protocol to compute shortest paths. In the absence of deterministic information, COPE estimates the probability that a particular neighbor has a packet as the delivery probability of the link between the packet's previous hop and the neighbor. Occasionally, a node may make an incorrect guess, which causes the coded packet to be undecodable at some nexthop. In this case, the relevant native packet is retransmitted, potentially encoded with a new set of native packets.

2.3 Packet Encoding Scheduling Mechanisms

The problem of scheduling refers to -after the relay node has determined which packets (belonging to different flows) should be encoded (mixed) together- which one of these mixed packets should be transmitted at each time slot. There exist two different approaches towards solving this problem: the relay has deterministic information about all key owners or the relay only has statistical overhearing information and utilizes feedback from the transmissions. As far as it concerns the statistical case two main algorithms that exist are:

1) Max Weight Algorithm
A control action consists of selecting a number of queues to serve (encode their packets together) at a single decision instance. The reward of each control is the sum of queue length times the average service rate for each queue. The Average Service Rate (ASR) is the expected number of successfully serviced packets times the transmission rate for which the encoded packet can be received by all intended receivers. This rate is the minimum of the reception rates of all intended receivers. Let $\mu_i(C)$ be the the ASR of flow i when control C is selected. Let also Q_i be the backlog of flow's i queue. In Max Weight Algorithm the selected control C* is the one that maximazes the product (let us call it control weight) of ASR and the flow queue backlog i.e C*:{$\mu_i(C^*)$ * Q_i > between the other control's weights }

2) Fixed Threshold Policy

In FTP the mixed packet combination is assigned a decoding probability P_D and if this number exceeds a fixed threshold, let say $P_D > G$ then this combination is considered meaningfull and it is known that the intented receivers could decode the packet with at least probability G. This procedure has been adopted by the COPE scheme. A variation of this policy is the $\delta - FTP$ algorithm in which instead of calculating average service rates, incoming packets are marked with information about decoding opportunities. In order to do so, overhearing probabilities q_i, j are compared with a fixed threshold $\delta \in [0, 1]$ and set to 1 if they exceed the threshold or zero otherwise.

2.4 Description of an Optimal Scheduling Algorithm

Because of the fact that the algorithms mentioned above are suboptimal concerning throughput, in this section we will present a throughput optimal algorithm suggested by the authors of [7]. For simplicity reasons we will present the algorithm for the case of two flows (see Figure 5; flow 1 is the red one, flow 2 is the blue one). Generally speaking this algorithm creates a

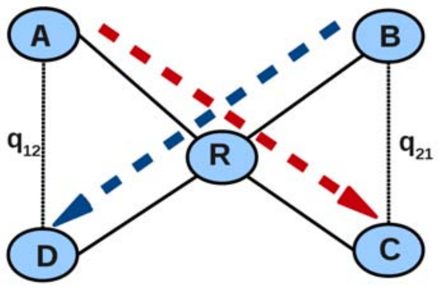


Figure 5: An example topology with two asymmetric flows ("X" topology).

virtual subnetwork for each flow, so that each node of this network represents the possible states of a packet: good (implying that the packet is owned by the other node) , bad (the other node does not own the packet), unknown (we do not know if the other node owns the packet). Packets with the same properties are grouped together (i.e they are enqueued in the same queue). For each flow we have three queues. For flow 1 there are $Unknown_1$, $Good_1$ and Bad_1. For flow 2 there are $Unknown_2$, $Good_2$ and Bad_2. The above categorization is done by the router node based on the feedback it gets, after sending an encoded (XOR-ed) packet to the nodes. An illustrative example is shown in Figures 6 and 7 .

The edges of the virtual network (that consists of the subnetworks of each flow) represent the state transitions of a packet and are assosiated with a transition probability (weight). The virtual subnetwork for the first flow is depicted in Figure 8. Each node (i.e queue) of the virtual network is associated with a queue backlog (for instance node n_u^1 has a backlog x_u^1). As we can see, when a packet of flow 1 enters the system (received by the router

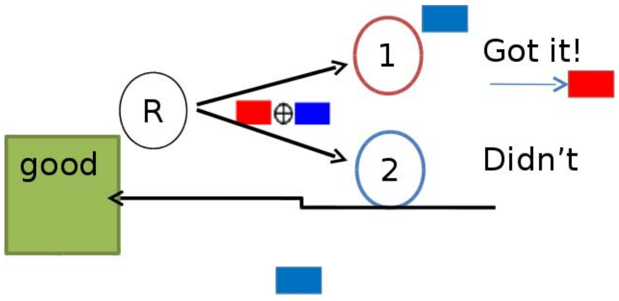


Figure 6: Blue packet is enqueued in $Good_2$.

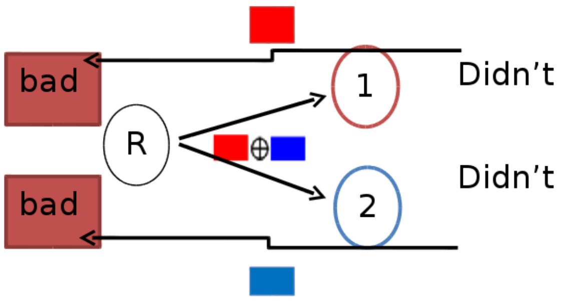


Figure 7: Red and blue packet are enqued in their respective bad queues.

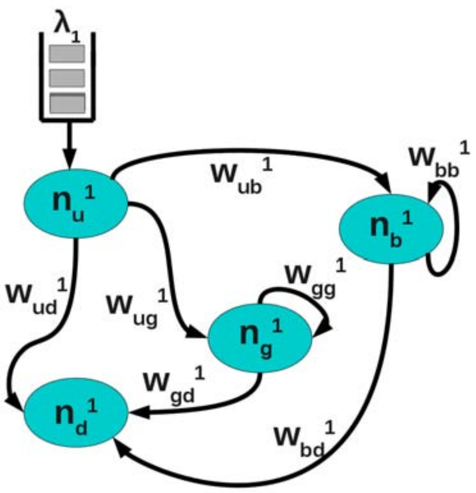
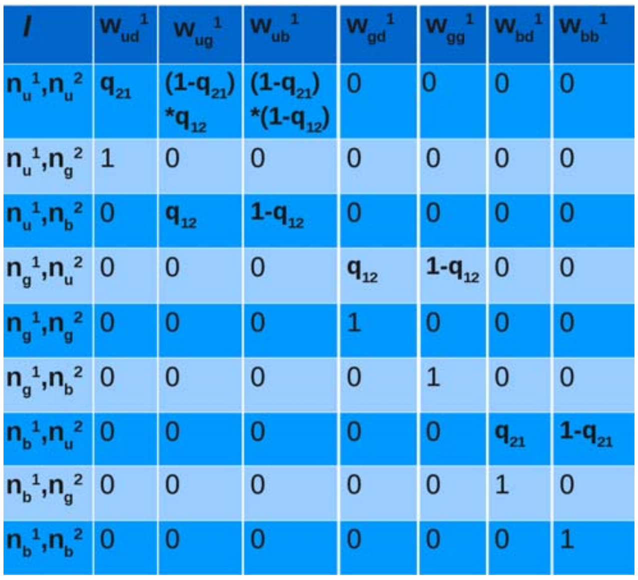


Figure 8: The subnetwork for flow 1.

node), we have no information about it so it is enqueued in the unknown queue. After this packet is encoded with another packet and gets transmitted, it might be characterized for example as good (and so be enqueued in $Good_1$) with probability w^1_{ug}. With probability w^1_{ub} it could be characterized as bad or with probability w^1_{ud} it could be successfully decoded and so leave the system i.e the backlogs for the destination node d of each subnetwork are always zero (x^1_d and x^2_d =0). Easily we can understand how a packet moves in its virtual subnetwork (it will be routed inside it until reaching

node d). In the same fashion, a packet of flow 2 will move likewise in its respective subnetwork. As far as it concerns scheduling: at each time slot, the router selects a control which corresponds to activating either:
a) one node from the virtual network (single control) or
b) two nodes from two different subnetworks (control pair), excluding destination nodes.
Once a control is taken, in case a) the first packet of the respective queue is transmitted or in case b) the two packets (the first packet of each corresponding queue) are encoded together and this encoded packet gets transmitted. In the table of Figure 9 we can see the transition probabilities (weights),



/	w_{ud}^1	w_{ug}^1	w_{ub}^1	w_{gd}^1	w_{gg}^1	w_{bd}^1	w_{bb}^1
n_u^1,n_u^2	q_{21}	$(1-q_{21})*q_{12}$	$(1-q_{21})*(1-q_{12})$	0	0	0	0
n_u^1,n_g^2	1	0	0	0	0	0	0
n_u^1,n_b^2	0	q_{12}	$1-q_{12}$	0	0	0	0
n_g^1,n_u^2	0	0	0	q_{12}	$1-q_{12}$	0	0
n_g^1,n_g^2	0	0	0	1	0	0	0
n_g^1,n_b^2	0	0	0	0	1	0	0
n_b^1,n_u^2	0	0	0	0	0	q_{21}	$1-q_{21}$
n_b^1,n_g^2	0	0	0	0	0	1	0
n_b^1,n_b^2	0	0	0	0	0	0	1

Figure 9: Transition probabilities for subnetwork of flow 1, for every control pair.

depending on which control pair is taken by the router, where q_{12} is the probability that the receiver of flow 2 has overheard a packet of flow 1, while q_{21} that the receiver of flow 1 has overheard a packet of flow 2. The respective table for subnetwork of flow 2 can be determined by exchanging indexes 1 and 2.

The question now arises is which would be an optimal throughput scheduling policy. The following algorithm comes to answer this question.
Let us first define some usefull notions:

- If a control I involves transmission from a single node j, located at subnetwork i then

$$\mu_k^i(I) = \begin{cases} r_i^{down} & if\ k = j \\ 0 & otherwise \end{cases}$$

- If the control involves an XOR packet from nodes j_1, j_2 located at subnetworks i_1, i_2 respectively, then

$$\mu_k^i(I) = \begin{cases} \min\{r_{i_1}^{down}, r_{i_2}^{down}\} & if\ k = j_1\ or\ k = j_2 \\ 0 & otherwise \end{cases}$$

- $\mathcal{I}$ is the set of all controls and I^* is the optimal control.

Optimal Algorithm
At each decision slot:

1) For each single control I=$\{n_c^i\}$ form the cost $Z(I) = x_c^i \mu_{n_c^i}(I)$

2) For each control pair I=$\{n_{c_1}^i, n_{c_2}^j\}$

   - form the weights
     $z_i(I) = \max\{x_{c_1}^i - \sum_{k \in N^{n_{c_1}^i}} w(n_{c_1}^i, k)(I) x_k^i\ ,\ 0\ \}$ and
     $z_j(I) = \max\{x_{c_2}^j - \sum_{k \in N^{n_{c_2}^j}} w(n_{c_2}^j, k)(I) x_k^j\ ,\ 0\ \}$
   - and the the cost $Z(I) = z_i(I)\mu_{n_{c_1}^i}(I) + z_j(I)\mu_{n_{c_2}^j}(I)$

3) select $I^* = \arg\max_{I \in \mathcal{I}}\{Z(I)\}$

The throughput optimality of the above algorithm, comes from the fact that it can be proven to have maximal stability region.

3 Network Coding architecture for Wireless Adaptive Links (NCRAWL)

NCRAWL is a lightweight framework for implementing and testing coding algorithms on real networks. The lightweight nature of NCRAWL is in terms of memory and CPU utilization. Some significant characteristics of this framework are

- it is a modular system because it consists of modules used for implementing vital operations such as encoding, decoding and scheduling. This modules have been designed effectively allowing low overhead operations

- it combines coding algorithms and scheduling

- it uses solely stochastic information

- it is channel aware in sense that routers are aware of all the potential NC opportunities that can take place within their neighborhoods at all times, as well as the maximum transmission rate of the encoded packets that allows for decoding. They are also able to determine which packets to code together in order to apply their NC policy.

3.1 NCRAWL Motivation

In cases where different transmission rates are employed to combat interference, NC may not always provide the favorable performance. Rate control is a traditional, effective means of coping with interference in wireless networks: the transmitter adapts the bit rate as per the quality of the link with the receiver. For example SampleRate selects the bit rate with the highest observed link throughput; the lower the throughput, the lower the bit rate. With regards to NC this is translated to transmitting encoded packets at the lowest bit rate that can be supported by all intended receivers, in order for any of them to be able to receive those encoded packets. As an example, consider the scenario in Figure 10, where nodes (1) and (3) wish to exchange packets (A) and (B) through the relay node (2), and f(a, b) is the highest-throughput bit rate on a link $a \to b$.

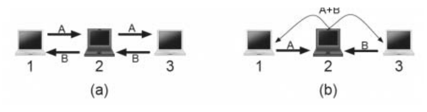


Figure 10: Simple chain topology example.

As mentioned above, while four transmissions are traditionally required, with NC the packet exchange can take place in three transmissions, since node (2) is transmitting an XOR combination ($A \oplus B$) of the packets to nodes (1) and (3). However, even though the same amount of information is exchanged with fewer transmissions, coding will not lead to throughput improvement, when: f(2,1) « f(2,3). In this case, the relay node (2) will have to transmit the packet ($A \oplus B$) at rate f(2,1); otherwise node (1) will not be able to decipher the packet. The use of this rate can be suboptimal for the link (2) $\to$ (3). f(1,2) « f(3,2), while f(1,2) « f(2,1). In this scenario, node (2) will have to wait for a long time until the reception of packet (A) from node (1), since the rate f(1,2) is very low. This will also delay the transmission of packet ($A \oplus B$); hence the throughput on link (2) $\to$ (1) will not be significantly increased.

3.2 A Brief Design Description

The design of NCRAWL involves the maintenance and update of local information with regards to the set of neighbors and their corresponding link qualities. Let us initially assume that all N nodes belonging to the network are participating in Network Coding. The following information is maintained for any node $k \in N$:

- The set U(k, r) of all direct neighbors of k for every transmission rate $r \in R$, where R is the set of all available transmission (bit) rates. With this, node k will know the set of neighbors that can potentially overhear packets transmitted by k at a certain bit rate r.
- The set U (m, r) of all direct neighbors of a node m, where $m \in U(k, r)$. This will provide node k with the set of nodes that can overhear the packet transmissions of node m at a certain bit rate r.

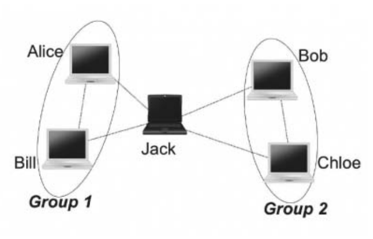


Figure 11: An "X" topology example.

Figure 11 shows an "X" topology example. Here, Jack is a node that can serve as a relay and can perform Network Coding. Jack has categorized his 4 neighbors into 2 groups. A group corresponds to the maximum transmission rate that can be supported by all nodes that belong to the group. Alice and Bill belong to group 1. They can communicate with each other and with Jack, using the maximum-throughput transmission rate r1, at any link direction. (Rate r1 is the highest-throughput bit rate that can be supported on any of the links that belong in group 1). Similarly for group 2, Bob, Chloe and Jack can exchange data using a rate r2, which is the maximum transmission rate that can be sustained by all links belonging in group 2. Note here the following: Nodes belonging to group 1 cannot communicate directly with nodes of group 2 at any transmission rate (i.e., Alice and Bob are not neighbors). In order for Alice to send a packet to Bob, she has to go through a relay node, such as Jack. It is possible that r1 = r2, since Alice and Bob may have similar link qualities with Jack. Again here, Jack will correspond Alice and Bob in two different groups, since Alice and Bob cannot sustain a direct link. Jack, Alice and Bill may have a common neighbor (e.g. John), with whom they can (mutually) communicate at a rate r3 < r1. In such a case, Jack will correspond Alice, Bill and John in a group 3, different than group 1. This organization of the local topological information enables Jack to make decisions with regards to transmitting encoded packets at appropriate rates, as we discuss below. Facilitating NC the formation of groups allows Jack to make efficient coding decisions in real time. Let us assume that Alice wants to send a packet to Bob, while Bob wants to send a packet to Alice, and that Jack is the relay, as in the above figure. Jack's locally maintained view of the topology suggests that Alice and Bob belong to groups 1 and 2 respectively. As soon as Jack receives both Alice's and Bob's packets, he performs an encoding operation (XOR) and transmits the encoded packet to both of them. If $r1 < r2$, the encoded packet is transmitted at rate r1, else at rate r2. Note that the encoded packet is not a broadcast packet (broadcast 802.11 packets are transmitted at the basic rate). It is a unicast packet, addressed to either Bob or Alice. Since both recipients are in monitor mode, they are both able to receive the encoded packet. If the reception of the encoded packet fails (e.g. a collision occurs), Jack will transmit the individual packets; we discuss this scenario below. Nodes perform the above operation opportunistically, and this can facilitate the exchange of information between groups. As an example, consider now the scenario where Alice wants to send a packet to Bob, and Chloe wants to send a packet to Bill. Since both packets have to be routed through Jack, he can perform the same operation as previously, and transmit an encoded packet that he constructs from the mixture of the two individual packets. Notice here that both Alice and Bill belong to the same group. With this, Jack implicitly assumes that Alice's packet can be overheard by Bill, since: (a) Alice is Bill's neighbor, and (b) the Alice-Jack link supports the same maximum rate as the Alice-Bill link. Since Bill can overhear Alice's packet, he will have the required information to decode the encoded packet transmitted by Jack, in order to get Chloe's packet. Clearly this requires from Bill to correctly decipher Alice's original packet. This packet, although transmitted to Jack successfully, may collide at Bill's antenna. In such a case, Bill is not going to be able to decode Jack's subsequent encoded packet. To address this situation, Bill acknowledges the packets that he has successfully decoded. If Bill does not send a positive acknowledgment for Chloe's packet, Jack will unicast the latter to Bill.

3.3 NCRAWL System Implementation Description

The big picture: the main NCRAWL system is a Click network packet processor that includes the SRCR routing protocol implementation for mesh networks [8]. There have been included two additional processing stages: the NCRAWL decoder and the NCRAWL encoder. These stages have been developed as individual Click elements, and have been placed before the beginning and after the end of the SRCR processing flow, respectively, as depicted in Figure 12. The main components of NCRAWL are described bellow:

a) packet decoder: the main tasks of the decoder module are the following:

- To use the available (from overhearing or ownership) key packets in order to decode the received encoded packet.
- To schedule the transmission of Layer-3 acknowledgements for the cor-

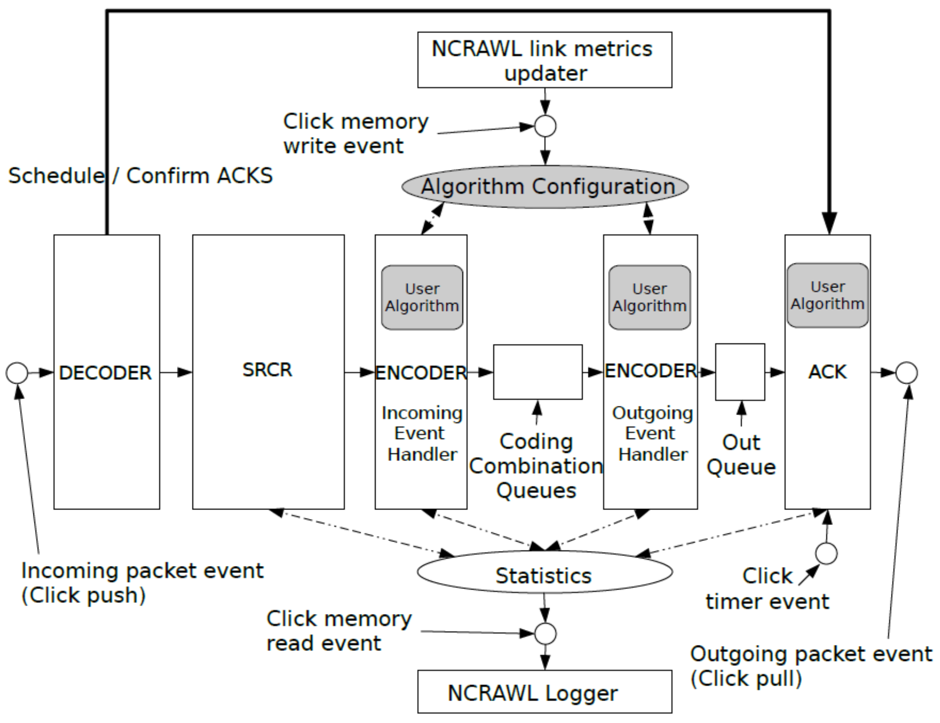


Figure 12: NCRAWL architecture.

rectly retrieved native packets, derived by the decoding operation.

- To determine any potential pending acknowledgements, as well as to verify any received acknowledgements.
- To tag and store all the correctly overheard data packets as potential keys; as discussed above, these will be potentially used in the near future for decoding received encoded packets. Moreover the key repository is used for packet resending in case a respective acknowledgement never arrives. The decoder resides at the packet receiving side of the system and is invoked by the respective packet arrival event.

b) NC packet encoder: the NCRAWL encoder element resides at the sending side of the system and is more complicated, since it maintains and manages the processor packet queues. A part of the element handles incoming packet events, another part deals with outgoing packet events and there is also code that gets invoked by timer as well as read and write Click configuration events (Figure 12). It is this element that exports the framework API which can be used to develop network coding algorithms. Specifically, the main assigned tasks for this module are the following:

- To process and place incoming native packets (keys) into particular available maintained queues. The system supports a plurality of queueing operations, which can be configured as per the requirements of the NC algorithm under development.

- To identify and combine packets together, towards forming encoded packets. The selection of the appropriate packet set follows the directions of the NC algorithm under consideration, supported by NCRAWL.
- To generate any potentially expected acknowledgement tokens for each one of the packets of an encoded combination.
- To piggyback any acknowledgements (through the use of scheduled, upcoming data transmissions) that have been scheduled by the decoder element, regarding outgoing native data packets.

c) Up-to-date topological information maintainer: the link metrics updater is responsible for collecting information about the network neighborhood as well as corresponding link transmission rates and PDR values from all nodes. The information gathered is passed to the rest of the system via the Click memory write event mechanism. Furthermore, the NCRAWL code that configures the encoding combination policies is invoked to react as needed. Gathering link quality information: The NCRAWL updater relies on the existing SRCR protocol component, which maintains link connectivity information and performs periodic measurements on all links. SRCR sends probe packets at all rates to determine the PDR for each link and chooses the highest rate that performs well. PDR information is then used by SRCR to calculate the ETX or ETT metric [9] [10], which provides information about entire routing paths, not just direct links. This information is kept in the SRCR link table, which can be read from other Click components. The SRCR period can be set as desired (the default value been used is 3 sec). Managing neighbor information: Based on the information of the SRCR link table, the link updater maintains its own so-called Neighbor Table (NT), which includes information for its neighbors. Initially, the NT is empty. The updater periodically reads the SRCR link table and updates NT as needed. The NT contents are updated whenever (i) a new neighbor appears, or (ii) an existing neighbor disappears, or (iii) a certain link quality changes. In such cases, the NCRAWL updater broadcasts a packet with the new NT contents and sets a timer. When such a NT packet is received (overheard), the updater replies by broadcasting its NT, provided it has not done so recently. The reply suppression threshold is set equal to the SRCR period. The NT packets are used by the NCRAWL updater to maintain the so- called Received NT Table (RNTT). This table complements the NT, holding information about the link quality as experienced/measured by the neighbor nodes themselves rather than by the local node. When an NT packet arrives, the corresponding RNTT entry is updated respectively. Packets from nodes that are not in the NT are ignored; a node must be "officially" reported as a direct neighbor by SRCR in order to be considered by NCRAWL.
Feeding NC algorithms with updated topological information: Each time the updater modifies the contents of the NT or RNTT (i.e., each time it proactively sends or receives a NT packet which leads to an update of the RNTT) a timer is set. When the timer expires, the new link qualities are passed to the main NCRAWL system, where they will potentially drive adaptive NC

decisions, based on the NC designer's needs. This timeout is (generously) set to 1 sec, providing ample time for any NT reply packets to arrive.
Keeping overheads low: The NCRAWL updater employs its own threads of execution to perform these information maintenance tasks (the current implementation uses 2 threads), but these remain suspended most of the time, making this component quite unintrusive in terms of CPU occupancy. Moreover, only a small fraction of the wireless bandwidth is typically used to collect the required link quality information from the neighboring nodes. Finally, the "reactiveness" of the updater is a function of the SRCR period. If a smaller period is used, link changes can be tracked faster (and more accurately) but the processing and communication overhead will increase too. Having said that, the implementation of the NCRAWL updater is not optimized for a very rapidly changing environment as this is not the focus of this work.
d) NCRAWL logger: the read events are used by another application, the NCRAWL logger, which gathers various statistics that are generated online by both encoder and decoder elements.
e) NCRAWL acknowledgement sender: NCRAWL acknowledges individual (native) data packets, but also groups packet acknowledgements per encoded combination. This way, if the same encoded packet has been successfully decoded at one recipient but failed at another, the sender can figure out which of the undelivered packets can be reused in encoded combinations, based on whether they have been logged successfully as keys by fellow recipient nodes. We should note here that NCRAWL provides this support; however, it expects that the user algorithm will make the final scheduling decisions. NCRAWL uses by default a user defined timeout threshold to resend packets that have not been acknowledged. Note also that a timer-expire event triggers the transmission of acknowledgements in separate packets when there is not enough outgoing traffic to piggyback them (Figure 12). NCRAWL also reschedules packets from the key repository for which acknowledgements have not arrived. Utilizing resources effectively: Efficient resource utilization was a main concern during the design of the various NCRAWL subsystems. More specifically, the repository that stores copies of packets uses a FIFO queue as the main indexing mechanism and can host up to a user defined quantity. After the storage limit is reached, the oldest packet is removed in order for a new one to get stored. The same packets are also indexed in a hash table based on their network-wide unique identifiers that we have previously discussed. The hashtable is used to quickly retrieve packets either as keys for decoding, or for resending them in case an expected acknowledgement token expires. The same indexing approach has been used for the acknowledgements and expected acknowledgement tokens as well.

3.4 NCRAWL VS COPE

The main differences between NCRAWL and COPE are mentioned below

- **Low CPU and memory utilization operations:** NCRAWL has been optimized at each stage of network coding operations, leading to a lightweight generic framework in terms of CPU and memory utilization. The result is a framework that provides the theoretical predicted throughput gain while at the same time it requires much less CPU processing, compared to COPE. The compreretive results are ilustratede in Figure 13. where NCRAWL's algorithm-1 is a Max-Weight

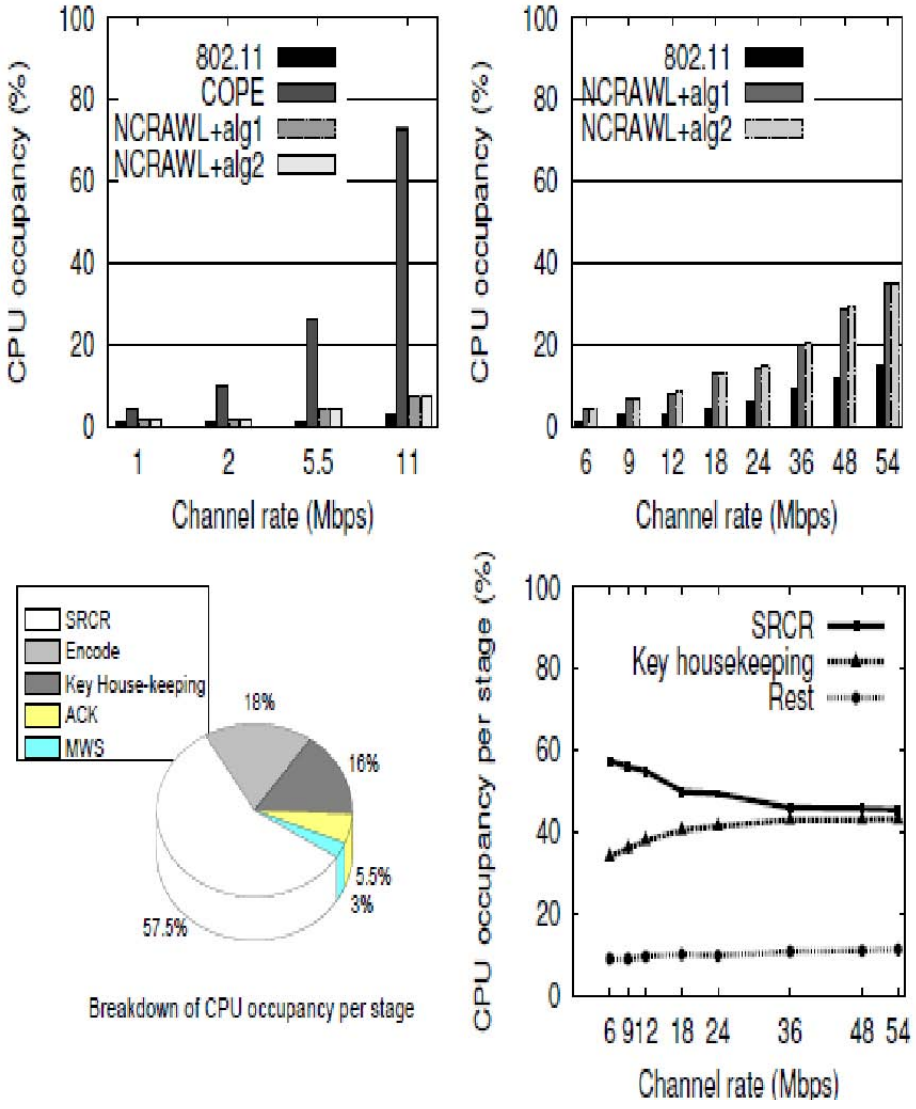


Figure 13: CPU comparative results.

algorithm like the one described in subsection 2.3 and algorithm-2 is a Myopic algorithm with feedback where bad packets are sent directly to MAC layer for transmission without coding while good packets are sent to the corresponding queue at the good state.

- **Avoidance of long packet headers:** with NCRAWL, nodes do not need to explicitly state (in future packet headers) which packets have been received and by whom. With COPE, each packet includes an additional COPE header, which contains this information, in order for neighbor nodes to know if their neighbors can perform decoding operations. In contrast, NCRAWL depends on the grouping mechanism which directly provides such information. Hence, theoretically Jack can directly predict that Bill and Chloe have sniffed Alice's and Bob's packets respectively. A possible (potentially minor) drawback with this design is that Alice's packet may have collided at Bill's antenna (with another packet from someone else); in this case, Bill would be unable to decode Jack's encoded packet. Since nodes acknowledge all the packets that they successfully decode, NCRAWL resolves this issue by having Jack to specifically unicast the packet to Bill. Note here that the COPE header is of variable length; the more the packets sniffed, the larger the header becomes for future data packets. However, this can create extremely long packet headers, thereby increasing the transmission overhead and reducing the throughput. More than that, this problem will be exacerbated at high bit rates, where thousands of packets are sniffed per second. Finally the processing overhead of the sniffed packets will also be a major issue: the sniffer has to filter all these packets, store them and add their IDs into future packet headers. This is a quite time-consuming task and is expected to tremendously degrade the performance of the device at high bit rates.

- **Ability to employ NC dynamically:** as explained above, in certain scenarios coding does not offer additional benefits. NCRAWL is able to perform coding selectively for certain link qualities and between specific groups, by simply observing the state of the links in the different groups. However, the COPE implementation cannot make such decisions online. With COPE, coding is always performed, and the extra COPE header is always used with data packets.

- **Aggressiveness in encoding:** NCRAWL is more aggressive in exploiting encoding opportunities. While COPE prefers encoding packets of similar lengths, NCRAWL encodes all kinds of packets that reach Layer-3 (e.g. data packets, TCP acknowledgments, etc). Depending on the implementation, searching for packets of appropriate length may incur even more processing overhead. (COPE considers 2 different virtual queues per node: one for small packets and one for large ones). On the probability of packet reception: NCRAWL is heavily dependent on the grouping mechanism. Jack can directly expect that with a high probability nodes belonging to the same group have sniffed the same packet. In contrast COPE incorporate's the IDs of the sniffed packets into the COPE header.

- **Predestination of the encoded packet:** the encoded packets are destined to the node with the poorest link (in the groups of interest). Consider for example group 1 in the above figure, and assume that all links in group 1 use the 6 Mbps transmission rate. Even though the same rate is used for all those links (an inherent design concept of NCRAWL), each link may have a different quality in terms of packet delivery ratio (PDR), and thus the expected transmission count (ETX) and expected transmission time (ETT) metric values may differ among the 1-hop links in the group. With NCRAWL, whenever Jack transmits an encoded packet to groups 1 and 2, he addresses the packet to the node X with the highest ETT value on the link $Jack \Leftrightarrow X$, among all the nodes that will sniff the packet. With this, if Jack receives an 802.11 ACK from X (perhaps after a potential number of MAC retransmissions of his encoded packet), he is quite confident that all the other nodes (in the respective groups) will have managed to successfully overhear his encoded packet.

4 Experimentation in Wireless Testbeds

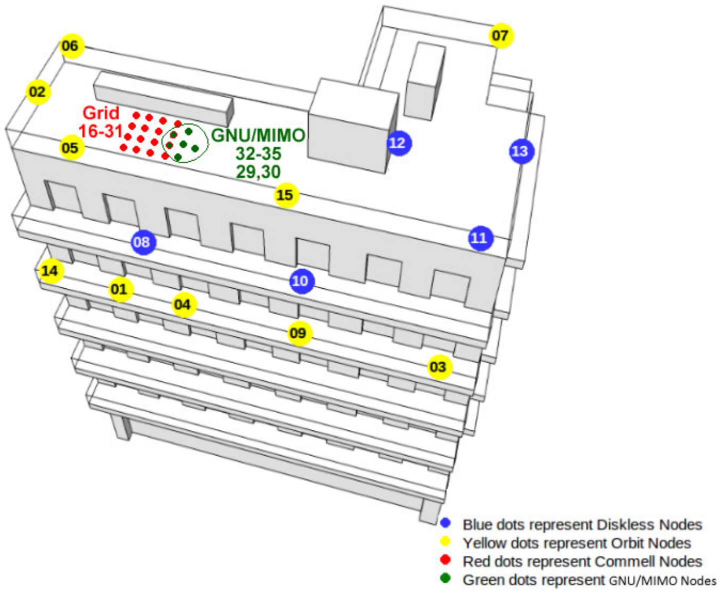
A testbed is a platform for experimentation of large development projects. A wireless network testbed consists of the hardware and software needed, in order to give researchers a wide range of environments in which to develop, debug, and evaluate their wireless network systems and protocols, in realistic conditions.

4.1 NITOS Testbed

NITOS (Network Implementation Testbed using Open Source code) is a testbed created by the Network Implementation Testbed Laboratory of the Computer and Communication Engineering Department at University of Thessaly, in collaboration with the Centre for Research and Technology Hellas (CERTH). The NITOS platform is open to any researchers who would like to test their protocols in a real-time wireless network. They are given the opportunity to implement their protocols and study their behavior in a custom tailor-made environment. The testbed's capabilities are constantly extended. Up to now it consist of 10 Orbit nodes, 5 diskless nodes and 20 Commell nodes. The topology placement of the nodes can be seen in Figure 14. Analytical details about the specifications of the nodes are explained in the next section.

4.2 Hardware Specifications

NITOS Server
A console server in which the experimenter connects, so that to have access to testbed resources (manage the reserved nodes, transfer files, run experiment scripts etc).
Development Server
A development server relies on NITOS testbed's intranet and its basic feature is to provide ease of use in source code development and compilation procedures. NITOS users can take advantage of that facility because Nitos-Dev has all the development versions of the packages that are used on the testbed node images, so the user does not have to download any extra software or check any version compatibilities. This is particularly convenient for wireless driver development because typically they have to get compiled



06
07
02
Grid 16-31
GNU/MIMO 32-35 29,30
05
12
13
15
08
11
14
01
10
04
09
03
● Blue dots represent Diskless Nodes
● Yellow dots represent Orbit Nodes
● Red dots represent Commell Nodes
● Green dots represent GNU/MIMO Nodes

Figure 14: NITOS nodes deployment.

against certain versions of the kernel source. The major motivation for building Nitos-Dev was to deal with file transfer latency. For example, click modular router's executable is typically 11 MB in size. Changing code, recompiling and transferring the new executable can be very frustrating, if the link quality with the testbed's front-end is not fast enough. **Orbit Node**
Orbit nodes are actually PC's equipped with :

- 1 GHz VIA C3 proccessor
- 512 MB of RAM
- 40 GB disk
- two ethernet ports
- two 802.11 a/b/g cards
- a Chassis Manager i.e a PCI connecting control board, that has been designed specifically for the VIA MB770 mainboard. This board is powered up by standby power, and has its own Ethernet NIC to receive power control commands.

Video Testbed
Fifteen of the wireless Orbit nodes are equipped with Logitech C120 USB web-cameras, capable of up to 30 fps video capture (640x480 pixels). This setup enables video-related research, such as real time services and QoS provision for video in wireless networks.
Mobility
NITOS testbed will offers three WiFi enabled nodes that are mobile. Two of them will be mounted on programmable robots, specifically i-Robots of

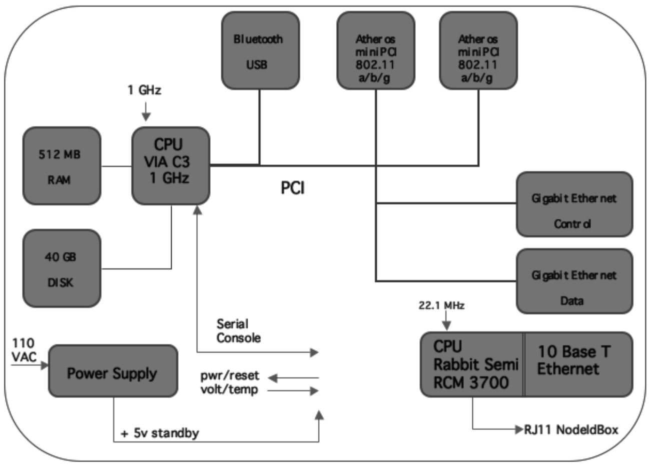


Figure 15: Orbit node schematic.

the Acroname Robotics company. The third follows a standard predefined path, as it is mounted on a DC-motor kit (Phidget LV motor kit) and moves on rails placed on the rooftop of one of the buildings.

Diskless Nodes

The basic characteristic of these nodes is the lack of a local disk. Furthemore, their price is significantly lower than the Orbit nodes due to their hardware specifications. Sometimes, we may refer to those diskless nodes as Low Cost Nodes (LCN). The LCN are [alix2.c2] system boards of [PC Engines company] equipped with:

- CPU: 500 MHz AMD Geode LX800
- DRAM: 256 MB DDR DRAM
- Storage: Compact Flash socket
- Power: DC jack or passive POE, min. 7V to max. 20V
- Three front panel LEDs, pushbutton
- Expansion: 2 mini PCI slots, LPC bus
- Connectivity: 2 Ethernet channels (Via VT6105M 10/100)
- I/O: DB9 serial port, dual USB port

Commell Nodes

NITLAB's custom made nodes that feature heterogeneous hardware are based on Commell LV-67B motherboards, with Intel Core 2 Duo P8400 2,26 GHz processors and 1 GB RAM. With 20 Commell Nodes, NITOS expanded its hardware facilitites and combines different technologies. NITOS Commell nodes are deployed in 4x4 GRID and the rest of them 4 + 2 (from the GRID) form a star topology. Those 6 Commell nodes (in star topology) are attached with GNU Radio boards and support MIMO features.

GNU radios

Six of the Commell nodes are connected with GNU Radio boards (software defined radios), specifically USRP1 boards with XCVR2450 daughterboards (2.4-2.5 GHz and 4.9 to 5.85 GHz Dual-band Transceiver). The GNU radio software allows the researcher to program a number of physical layer features (e.g. modulation), thereby allowing for dedicated PHY layer or cross-layer research. A GNU radio board connected with a Commel node is illustrated next.

MIMO support

WMIA-199N WLAN 802.11 n (ATHEROS 9160) wireless miniPCI cards as well as WMIR-200N Ralink RT2860+2850 mini PCI cards are used. The basic setup is to support two types of polarization enabling vertical and horizontal diversity. This is done with an external box that is attached with the Commel node's box and it can be rotated either horizantally or vertically. The external box hosts 3 antennas that are connected to the miniPCI cards via Pigtails.

Switch

A PoE switch [D-Link DES1228P] is used for wired connection between server and nodes.

The general NITOS architecture is summarized in Figure 15.

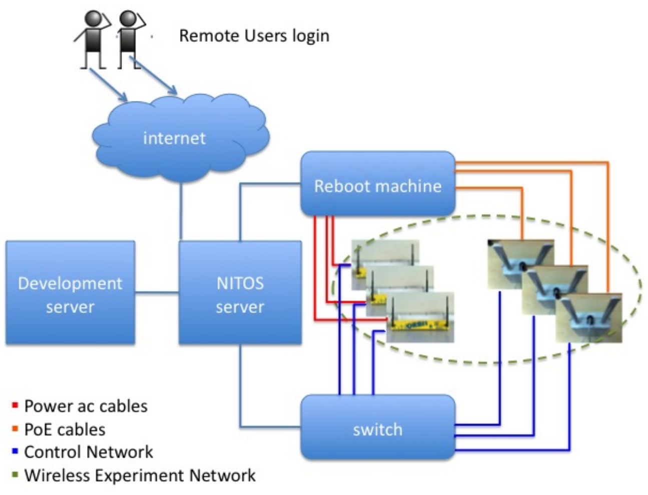


Figure 16: NITOS architecture diagram.

4.3 Software Specifications

NITOS testbed is based on a set of software tools for its operation. In order to book nodes and channels for a specific time period, one has to use the NITOS Scheduler web-application, specifically designed to facilitate the reservation procedure.

Being aware of the fact that an experimenter needs connectivity and link quality data before he selects the desired nodes for an experiment, there has been created a topology-connectivity tool, based on the TLQAP protocol. By means of this tool the user has access to up-to-date information regarding the quality of each direct wireless link in the testbed.

For setting up and controlling an experiment, as well as for retrieving measurements, it is recommended the use of the OMF software framework (cOntrol and Management Framework). Though one is free to use custom scripts in association with standard widely-used network tools (e.g. iperf), OMF offers a complete network experiment control solution and ultimately ease of use.

PXE and Frisbee is used for booting the nodes and loading the desired kernel images on them respectively. It is also used NFS for mounting the filesystem. This approach is adopted to avoid the delays and inconveniencies related to burning an entire filesystem on the nodes. The user can build his own image or use the images provided with every new account and located in the user's home directory.

5 Developing Network Coding Systems

In this section we will see things from a developer's perspective. See appendix at the end of the document for some fundamental concepts of computer networking.

5.1 Network Drivers

Driver is a program that allows operating system's kernel to communicate with various hardware devices such as the hard disk, sound cards, wireless network cards etc. Drivers translate the language of the operating system to an API (Application Programming Interface) that the hardware device can understand. When the operating system starts up, it tries locating various devices that are connected to a computer system. It then loads the appropriate driver into the kernel memory space and waits for an input from the user to communicate with the device, so that the device can perform the appropriate action (by calling the respective function). Concerning a wireless network driver, the basic functions are transmission, reception and rate adaptation.

First of all, the wireless driver has to be loaded into kernel. When the driver gets loaded, it has to allocate some memory space for itself to exist within the kernel. Thus the driver has to request for this space from the kernel. Then the device is registered with the kernel. By doing so, the kernel is informed about the existence of the particular device. Next step involves exporting non specific system calls that can be invoked by user level program through the kernel. Although user and kernel space are separated (a program in userspace cannot access kernel space memory) a device can be easily accessed by a file in the /dev directory of the linux kernel, which implies that one can use C code to talk to the device, just by writing to and reading from this file. Then the wifi driver calls the function open to initialize basic settings for transmission. This is generally represented as dev->open. This function gets executed when a user pushes up an interface. This is done by “ifconfig ath0 up” for an Atheros card’s interface. So kernel calls the open function which in turn turns on the hardware device. This function also performs operations such as allocating address space for the hardware, copying the MAC address and setting interrupt request (irq) numbers. Then it starts the netif_start_queue to initialize the network queue to accept packets from kernel.

Let see now what happens when a user sends a request to the wireless card driver (let say that he is requesting a web page using his browser). The request to deliver packets arrives from the upper layer (application layer). This packet is then fragmented into smaller chunks, each one of size specified by the fragment value. This is then encapsulated with TCP/IP information by the network layer. Kernel now appends this packet with Ethernet information, namely source, destination and packet type as obtained by the ARP table. Till this point, this packet is only an Ethernet packet that can be sent over by wire to a hub. The wifi driver, converts these Ethernet packets to WiFi packets as specified by the IEEE standards. All through this process, sk_buff is used by the kernel to keep track of the packet headers. This is the kernel buffer which stores information of each packet like its data, TCP HEADER, IP HEADER etc. This header is transferred from the upper layer through every layer in the OSI Model of the linux kernel. In each layer, it will be appended with that layer header and sent to the layer below it. Details about sk_buff’s structure can be found in file sk_buff.h of the kernel’s source. When a packet arrives to the kernel from the upper layers, kernel calls the function dev->hardstart_xmit. This function is often mapped with a device specific function in the driver. This function performs some sanity check such as checking the packet’s size, before converting it to an 802.11 one. Thus an Ethernet layer packet as it arrives to the kernel, is then modified to an 802.11 packet by removing the header of this packet and replacing it with a new header. With the new MAC header of 802.11 type, the packet is ready to be sent out. Normally what follows is a call to a device specific function, such as ath_tx_start() (in the case of Atheros drivers). This function is responsible for setting the transmission rates of the packets which are sent out. This function also decides on the QoS of the packet, based on the various flags set by the user to differentiate the packet. In the case of need for encryption of data packets, it is this function which is responsible for doing so. It also defines specific rates for management, control and data packets. It is generally a rule in wifi networks that management packets are transmitted at the basic rate. This is because of the fact that not all stations can listen to packets with high rates. With these procedures, packet is placed in the queue and a transmit interrupt is raised. In many cases a bottleneck may be caused by the device (for instance when having the combination of a slow device with a fast CPU). In this case if an extra packet arrives, it is stored in hardware queue. Linux kernel provides a sophisticated mechanism to be informed about slow hardware devices. Kernel provides driver with functions such as netif_stop_queue() and netif_wake_queue(). The first one is used to inform the kernel from the driver, to stop sending packets when packet buffer overflow occurs. The second is used to resume this interrupted operation. These two functions can be used alternatively to control the feed rate of packet come from kernel into the driver's buffer. Sometimes netif_tx_disable() is used instead of netif_stop_queue() in case of disabling packet transmission anywhere outside hardstart(). This method is particularly helpful when sending high stream video packets. It may also happen that because of a buggy device, a packet may fail to be transmitted. To make sure everything goes right, kernel initializes a watchdog timer that times out if the job was not done properly. This timer is similar to the timer used by ping to get a response back. The driver must have a function to handle this timer interrupt. Often these handlers have methods to report transmission failures to the upper layers.

As far as it concerns packet reception, this could be a little trickier than transmission, because of the fact that one needs to allocate memory space in the kernel to copy this information and send it to the upper layer. Reception of packets is generally interrupt driven, though there are cases when polling may be used. Often the hardware, when it receives a packet, checks to see if the packet is addressed to itself or if the packet is broadcast. For this cases, it raises an interrupt to the driver. It may be noted that, in promiscuous mode, device accepts all packets. Driver executes the appropriate interrupt handler routine, which calls the receive function. The receive function does the work of ripping the packet header and passing this packet to the upper layer. This function is also responsible for recording the statistics information about the packets received. Every packet entering this function goes through a series of check for validating the packet. Then the 802.11 stack input function removes the MAC header of the packet and substitutes it with an Ethernet header. This packet is further checked for a valid sequence number to identify if these packets are duplicate. Duplicate packets are discarded as these are nothing but mere retries of the same packet.

5.2 Click Modular Router

Click is a software architecture for building flexible and configurable routers. A Click router is assembled from packet processing modules called elements. Individual elements implement simple router functions like packet classification, queueing, scheduling, and interfacing with network devices. These elements are writtern in C++ programming language. Click naturally belongs to OSI Layer-3. It can also give hints to OSI Layer-2 (MAC) because it can cooperate with Layer-2 drivers such as Atheros Madwifi, but it cannot be used for radical Layer-2 implementations. A router's configuration is (conceptually) a directed graph with elements at the vertices; packets follow along the edges of the graph. Configurations are written in a declarative language that supports user-defined abstractions. The click configuration file describes connectivity between elements (an acyclic directed graph is formed). It also allows configuration arguments to be specified for the elements.

Typically there are three kinds of processing flows in Click:

- A flow that delivers packet to the next stage (a push flow)

- A flow that gets a packet from the next stage (a pull flow)

- A flow that does not handle packets (typically triggered by a timer)

The types of elements are:

- Push: Pushes packet to the next element. E.g. FromDevice();
- Pull: Pulls packet from the previous element E.g. ToDevice();
- Agnostic: May act as push or pull. E.g. Paint();

The most important properties of an element are:

- Element's class. Each element belongs to one element class. This specifies the code that should be executed when the element processes a packet, as well as the element's initialization procedure and data layout.
- Ports. An element can have any number of input and output ports. Every connection goes from an output port of one element to the input port of another. Different ports can have different semantics; for example, second output ports are often used to emit erroneous packets.
- Configuration string. The optional configuration string contains additional arguments that are passed to the element at router's initialization time. Many element classes use these arguments to set per-element state and fine-tune their behavior.
- Method interfaces. They are methods exported to other elements. For example, methods for transferring packets like push(), pull() etc. Elements communicate at run time through these interfaces, which can contain both methods and data.
- Handlers. They are methods exported to the user rather than to other elements in the router's configuration. They support simple, text based read/write semantics. Each element can easily install any number of handlers, which are access points for user interaction. They appear to the user as files in Linux's /proc directory; for example, a count handler belonging to an element named e would be accessible in the file /proc/click/e/count. One of e's methods is called when the user reads or writes this file. This lightweight mechanism is most appropriate for local modifications to an element, such as changing a maximum queue length. Handlers are also useful for exporting statistics and other element's pieces of information.

Figure 17 shows a sample element, Tee (2). 'Tee' is the element's class; a Tee copies each packet received on its single input port, sending one copy to each output port. Configuration strings are enclosed in parentheses; the '2' in 'Tee(2)' is interpreted by Tee as a request for two outputs. Method interfaces are not shown explicitly, as they are implied by the element's class. Figure 18 shows several elements connected into a simple router that counts incoming packets and then throws all of them away. Click supports two kinds of connections, push and pull. On a push connection, packets start at the source element and are passed downstream to the destination element. On a pull connection, on the other hand, the destination element

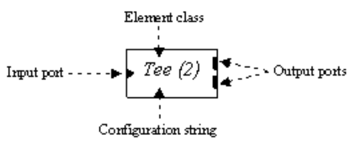


Figure 17: A sample element; Triangular ports are inputs and rectangular ports are outputs.

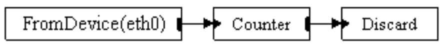


Figure 18: A router configuration that throws away all packets, after counting them.

initiates packet transfer: it asks the source element to return a packet, or a null pointer if no packet is available. The processing type of a connection whether it is push or pull is determined by the ports at its endpoints. Each port in a running router is either push or pull; In a push port we have event based packet flow, but in a pull port it can be used scheduling or polling. Connections between two push ports are push, and connections between two pull ports are pull. Connections between a push port and a pull port are illegal. They may also be created agnostic ports, which behave as push when connected to push ports and pull when connected to pull ports. In diagrams bellow, black ports are push and white ports are pull; agnostic ports are shown as push or pull ports with a double outline. In Figure 19 there are some illegal connections. The top configuration has four errors: (1) FromDevice's push output connects to ToDevice's pull input; (2) more than one connection to FromDevice's push output; (3) more than one connection to ToDevice's pull input; and (4) an agnostic element, counter, in a mixed push/pull context. The bottom configuration, which includes a Queue, is legal. In a properly configured router, the port colors on either end of each connection will match.

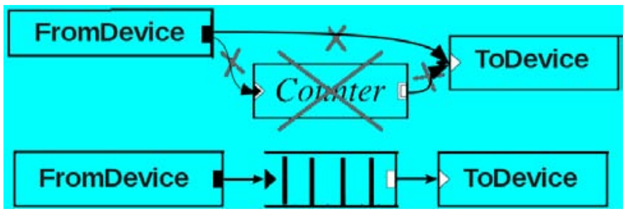


Figure 19: Some element connection violations.

As far as it concerns a packet, it consist of:

- char*
- Access with struct*
- Annotations i.e metadata to simplify processing carring information downstream, that are statically defined fields in a packet. We can have

IP header information, TCP header information, Paint annotations and user defined annotations.

Click elements do not have implicit queues on their input and output ports, because of the associated performance and complexity costs this would provoke. Instead, queues in Click are explicit objects, implemented by a separate Queue element. This gives the router's designer explicit control over an important router property, namely how packets are stored. It also enables valuable configurations that are difficult to arrange otherwise. For example, a single queue feeding multiple devices, or a queue feeding a traffic shaper on the way to a device. Explicit queues necessitate both push and pull connections. A Queue has a push input port and a pull output port; the input port responds to pushed packets by enqueueing them, and the output port responds to pull requests by dequeueing packets and returning them.

Here follows the description of some basic elements. The format is ElementName(configuration arguments) | Push, pull, or agnostic (specifies port types)| Port descriptions (packet types and numbers of ports)| Description.

- ARPQuerier(...) | Push | First input takes IP packets, second input takes ARP responses with Ethernet headers. Output emits ARP queries and IP-in-Ethernet packets. Uses ARP to find the Ethernet address corresponding to each input IP packet's destination IP address annotation; encapsulates the packet in an Ethernet header with that destination Ethernet address.

- ARPResponder(ip eth, ...) | Agnostic | Input takes ARP queries, output emits ARP responses | Responds to ARP queries for IP address ip with the static Ethernet address eth.

- CheckIPHeader(...) | Agnostic | Input takes IP packets. | Discards packets with invalid IP length, source address, or checksum fields; forwards valid packets unchanged.

- Classifier(...) | Push | Input takes any packet | Examines packet data according to a set of classifiers, one classifier per output port. Forwards packet to output port corresponding to the first classifier that matched. For example classifier: "12/0800" checks that the packet's data bytes 12 and 13 contain values 8 and 0, respectively.

- Discard | Push | Discards all input packets.

- FromDevice(devicename) | Push | No inputs | Sends packets to its single output as they arrive from a Linux device driver.

- FromLinux(devicename, ip/netmask) | Push | No inputs | Installs into Linux a fake Ethernet device devicename and a routing table entry that sends packets for ip/netmask to that fake device. The result is that packets generated at the router host and destined for ip/netmask are emitted on FromLinux's single output as they arrive from Linux.

- GetIPAddress(16) | Agnostic | Input takes IP packets | Copies the IP header's destination address field (offset 16 in the IP header) into the destination IP address annotation; forwards packets unchanged.

- IPClassifier(...) | Push | Input takes IP packets | Examines packet data according to a set of classifiers, one classifier per output port. Forwards packet to output port corresponding to the first classifier that matched. For example classifier: “ip src 1.0.0.1 and dst tcp port www” checks that the packet’s source IP address is 1.0.0.1, its IP protocol is 6 (TCP), and its destination TCP port is 80.

- Queue(n) | Push input, pull output | Input takes any packet | Stores packets in a FIFO queue; maximum queue capacity is n.

- ToDevice(device) | Pull | Input takes Ethernet packets; no outputs | Hands packets to a Linux device driver for transmission. Activates pull requests only when the device is ready.

- ToLinux | Push | Input takes Ethernet packets; Linux will ignore the Ethernet header except for the protocol field. No outputs | Hands input packets to Linux’s default network input software.

More elements and their detailed description can be found at
http://read.cs.ucla.edu/click/elements

The user can also write his own elements. All he has to do is to add his element’s class i.e two C++ source files (File.hh and File.cc), override some methods (like port_count(), processing(), initialize(), etc.) , export the element and compile the elements. Some Debugging aids, when programming in Click, that could be used are:
-to print messages with click_chatter()
-dmesg at the prompt
-read /var/log/messages
-ksymoops

There are three possible ways to run Click. It can be run as:
- a Kernel module. In this way it completely overrides Linux routing, requires root permissions but the highest speed can be achieved. If Click crashes then consequently kernel will also crash (leading to a system crash). So it is recommended for final systems or to present experimental results.
- a Userlevel program. In this way it runs as a daemon on a Linux system. On the one hand it is a bit slower than Kernel-level Click but on the other it is easier to install and debug (but still sufficiently fast). So it is recommended for development and prototyping. In order for User-level Click to receive packets from network and send packets to the system, elements FromDevice(athX) and ToDevice(athX) are used respectively. The argument (athX) is the network’s interface that Click is connected to. In addition, to send packets to the system and receive packets from the system, virtual TUN/TAP interfaces are used . In this case default route to

[1]TUN and TAP are virtual network kernel devices. They are network devices that are supported entirely in software, which is different from ordinary network devices that are backed up by hardware network adapters. TAP (as in network tap) simulates an Ethernet device and it operates with Layer-2 packets such as Ethernet frames. TUN (as in network

the tun/tap device has to be set so traffic passes through the Click route. Because of the fact that Linux routing is still working, ip addresses must be correct.

- a routing agent within the ns-2 simulator(nsclick). It may be a bit difficult to install, but we can and take advantage of the simulation's benefits (having multiple routers in one system, less hardware, and being shielded of exogenous factors like interference)

5.3 Roofnet

Roofnet is an experimental 802.11b/g mesh network developed at the Computer Science and Artificial Intelligence Laboratory of the Massachusetts Institute of Technology (MIT). Part of the research project at MIT includes link-level measurements of 802.11, finding high-throughput routes in the face of lossy links, link adaptation, and developing new protocols which take advantage of radio's unique properties (ExOR). The software developed for this project is available free as open source. Roofnet consists of about 50 nodes in apartments in Cambridge. Each node is in radio range of a subset of the other nodes and can communicate with the rest of the nodes via multi-hop forwarding. A few of the nodes act as gateways to the wired Internet. A primary feature of Roofnet's design is that it requires no configuration or planning, and is thus easy to deploy and expand. A new user can turn on a new node and start using it for Internet connectivity with no configuration beyond installing the hardware. The new user need not allocate an IP address, aim a directional antenna, or ask existing users to perform any special actions to add the new node. One consequence of an unplanned network is that each node can route packets through any of a large number of neighbors, but the radio link to each neighbor is typically of marginal quality; finding the best multi-hop routes through a rich mesh of marginal links turns out to be a challenge. We will now focus on the softaware running at each node. The nodes use the Click software router toolkit for route discovery and packet forwarding. All the routing software runs in the kernel; this gives presice control over packet queuing and scheduling (to give routing messages high priority), and allows tight integration between routing and the 802.11 driver (to get feedback about failed transmissions, and control transmit power level and bit-rate). Click greatly eases routing protocol development by allowing upgrades without reboots and co-existence of multiple routing and forwarding schemes in the same kernel. Each node also runs a Web server, a NAT, and a DHCP server on its wired Ethernet port. The DHCP server and NAT allow user's home computers to use the node as a router without any configuration. The Web server provides a simple

TUNnel) simulates a network layer device and it operates with Layer-3 packets such as IP packets. TAP is used to create a network bridge, while TUN is used with routing. Packets sent by an operating system via a TUN/TAP device are delivered to a user-space program that attaches itself to the device. A user-space program may also pass packets into a TUN/TAP device. In this case TUN/TAP device delivers (or "injects") these packets to the operating system's network stack (thus emulating their reception from an external source).

configuration interface (to turn on and off DHCP, and to set the IP address of the wired interface), a status monitor showing what routes are available and their current metrics and a means for rebooting the node. The software architecture is presented in Figure 20.

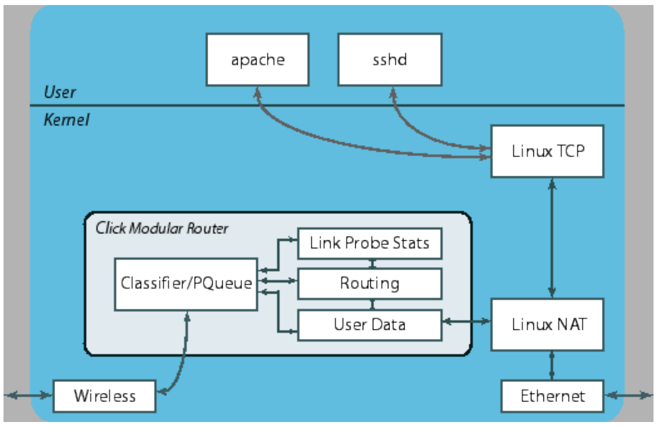


Figure 20: Roofnet software architecture.

Roofnet uses a routing protocol called **SrcRR**. The main goal of SrcRR is to find high-throughput routes. The key challenges it faces are the intermediate quality of most links, asymmetric link loss rates, frequent changes in link loss rates, and frequent losses of routing protocol packets due to interference from hidden terminals.

SrcRR's general design is inspired by DSR [11]. When a node n_0 needs to find a route to a destination n_d, it broadcasts a query for n_d. Each node n_i that hears a query forwards it, appending its own identifier to a source route in the packet. Each time n_d hears a query for itself, it sends a reply back to n_0 along the source route accumulated in the query. Node n_0 (and every node that sees the query or reply) adds all the links mentioned in the reply to a local link-state database, and uses Dijkstra's algorithm on that database to find the best route. When n_0 sends data packets to n_d, it includes that route (i.e. the sequence of node identifiers) in each packet as a source route. The primary way in which SrcRR differs from DSR is that SrcRR uses the ETX [9] metric to help it choose good routes. ETX continuously measures the loss rate in both directions between each node and its neighbors using periodic broadcasts. It assigns each link a metric that estimates the number of times a packet will have to be transmitted before it (and the corresponding 802.11 ACK) are successfully received; thus the best link metric is one. The ETX route metric is the sum of the link metrics; thus ETX penalizes both long routes and routes that include links with high forward or reverse loss rates. A node forwards a query if it has not seen the query before, or if the query's total route metric is better (lower) than the best instance of the query the node has yet seen. This increases the amount of query traffic, but decreases the algorithm's bias in favor of shortest hop count. Nodes also delay for a random period less than one second before forwarding a query to avoid contention. When a node forwards a query, it includes the link ETX metric to whatever node it heard the query from; nodes store these metrics in their link-state databases, and use them to compute the route metric to minimize with Dijkstra's algorithm. While a source node is sending data along a route, SrcRR uses the following techniques to discover if the route has broken or declined in quality. First, when a node forwards a data packet, it updates the packet's source route to contain the latest ETX metric from the preceding node; if the routes in the two directions are the same, this suffices to keep both ends aware of the current route quality. Second, if the 802.11 card indicates that ten packets in a row have failed to elicit an 802.11-level ACK, the node will send the current link metric to the source. Third, if a node is passing data in one direction but sees no data in the other (i.e. the route is asymmetric or broken), it will periodically send the current link metric to the source. Fourth, if a source node sees a new metric for a link it is using, it re-runs Dijkstra's algorithm to ensure it is using the best known route. Finally, if a source node notices that the route it is using has a current route ETX metric more than twice as high (half as good) as the best it has seen since the last query, it will flood a new query.

SrcRR is independent of IP, and operates at a lower layer. It uses 32-bit addresses; in the usual case in which it is carrying IP packets, SrcRR use IP addresses in its headers. A SrcRR node maintains a mapping from SrcRR 32-bit addresses to 48-bit 802.11 MAC addresses, derived implicitly from SrcRR query broadcasts.

6 Implementation of an Optimal Scheduling Algorithm

In this section it will be presented the implementetion details of the algorithm mentioned in section 3.4 . The algorithm was implemented as a part of this diploma thesis, in order to examine if the throughput region, presented by the authors of [7], can be achieved in practice, while in addition processing overheads will be kept low. The algorithm was built on the NCRAWL framework. In order to do that we extended the framework with some new modules of processing and modified some of the existing ones. In brief:

1. Four new queues added ($Good_1$, Bad_1, $Good_2$, Bad_2). The original NCRAWL's queues were transformed to act as unknown ones.

2. The acknowledgment procedure of NCRAWL was re-designed from scratch. In addition now the receivers can not only piggy-back acknowledgments, but they are able to send explicit acks (this is usefull in the case of asymmetric flow senarios where piggy-backing does not work)

3. A categorizing mechanism has been added. Its purpose is to insert a packet in the apropropriate queue (Unknown, Good, Bad) based on feedback the router gets via acknowledgements.

4. The scheduling procedure of NCRAWL has been modified towards taking into acount the proposed algorithm's metrics

In our case there exist six queues in total. In particular $Unknown_1$, $Good_1$, Bad_1 for the first flow and $Unknown_2$, $Good_2$, Bad_2 for the second. The main actions taking place, in order for the router to enqueue each packet (according to the feedback it gets) is that when the router sends an encoded packet, it enqueues an eapg entry in a FIFO queue (lets call it eapgFifo). Each eapg (encoded ack packet group) entry consists of the packet ids of the native packets (that have been encoded together) and a timestamp t_s (essentially it's the current time that the encoded packet was sent). It also inserts the packet ids of the two native packets in a hash table (acksToBeReceivedHash). Finally the two native packets are stored (in a structure called packetStorage) because they might need to be retrieved in the categorization procedure. When the receiver gets and decodes an encoded packet, it sends an acknowledgement back to the router (i.e it sends the packet id of the packet intended to the receiver). When the router receives an ack from a receiver it removes the packet id from the acksToBeReceivedHash. In addition, in the router's code a timer is triggered periodically (every 2 seconds), in order for the router to start the categorization procedure.

-Categorization Procedure: Let us call t_g the time that the timer was triggered. During this procedure, router reads the t_s of the first (oldest) entry (firstEapgEntry) in the eapgFifo. If t_s is at least 1 second older than t_g (i.e timestamp rule holds) then checks whether the packets of this eapg have been acked, by checking the presence (not acked) or absence (acked) of their ids in the acksToBeReceivedHash. The condition on the timestamp ensures that enough time has passed, so we can safely consider that a receiver had not been able to decode a packet. The categorization rules for the packets indicated in the firstEagpEntry are:

- if both packets have been acked
  -remove both packets from packetStorage
- if the first packet has been acked but the second has not
  -remove the first from packetStorage
  -enqueue the second in $Good_2$
- if the second packet has been acked but the first has not
  -remove second from packetStorage
  -enqueue the first in $Good_1$
- if both packets have not been acked
  -enqueue both packets in their respective bad queues (Bad_1 , Bad_2)

Then the firstEapgEntry is removed from eapgFifo and the same procedure is repeated for the next eapg entries, until for some entry the timestamp rule does not hold (so the categorization procedure terminates and will be repeated when the timer will be retriggered). The reason that a packet is removed from packetStorage is that, in essence, has left the system (reached its destination).

Now we will describe how the scheduling part of the algorithm is implemented. Interpreting the weights of step (2) of the algorithm, for the case of two flows they can be formulated as follows:

1. For $I = (n_u^1, n_u^2)$
   $z_1(I) = (x_u^1 - (1-q_{21})(q_{12}x_g^1 + (1-q_{12})x_b^1))^+$ and
   $z_2(I) = (x_u^2 - (1-q_{12})(q_{21}x_g^2 + (1-q_{21})x_b^2))^+$
2. For $I = (n_u^1, n_g^2)$
   $z_1(I) = x_u^1$ and $z_2(I) = q_{12}x_g^2$
3. For $I = (n_u^1, n_b^2)$
   $z_1(I) = (x_u^1 - q_{12}x_g^1 - (1-q_{12})x_b^1)^+$ and $z_2(I) = q_{12}x_b^2$
4. For $I = (n_g^1, n_u^2)$
   $z_1(I) = q_{21}x_g^1$ and $z_2(I) = x_u^2$
5. For $I = (n_g^1, n_g^2)$
   $z_1(I) = x_g^1$ and $z_2(I) = x_g^2$
6. For $I = (n_b^1, n_u^2)$
   $z_1(I) = q_{21}x_b^1$ and $z_2(I) = (x_u^2 - q_{21}x_g^2 - (1-q_{21})x_b^2)^+$

- For each of the above cases we compute cost $Z(I) = z_1(I)\mu_{n_{c_1}^1}(I) + z_2(I)\mu_{n_{c_2}^2}(I)$
- We have excluded controls $I = (n_g^1, n_b^2), I = (n_b^1, n_g^2)$ *and* $I = (n_b^1, n_b^2)$ because they are kind of meaningless.
- For the case of single controls, finding the cost is trivial.

Every time a change happens in a queue (by enqueueing a packet or dequeueing it in order to be sent out) the I^* is recomputed(updating the costs) so the most profitable (max cost) control combination is on the head of a list that includes the elements of $\mathcal{I}$ (all the controls). We will call this list controlsList.

The algorithm has been implemented in order to run on Nitos testbed. So, having the intention to avoid the occupation of many nodes (five in particular) of the testbed we came up with an alternative solution, involving only three nodes (alice-relay-bob topology). In this case the key needed by a node to decode an encoded packet is its own packet (the one that had been sent before). So nodes have to store (keep) their own packets. We simulate the overhearing probalities q_{12} and q_{21} by introducing a mechanism in which a node keeps a packet with a certain probability. In the figures below we can see the basic tasks of each node.

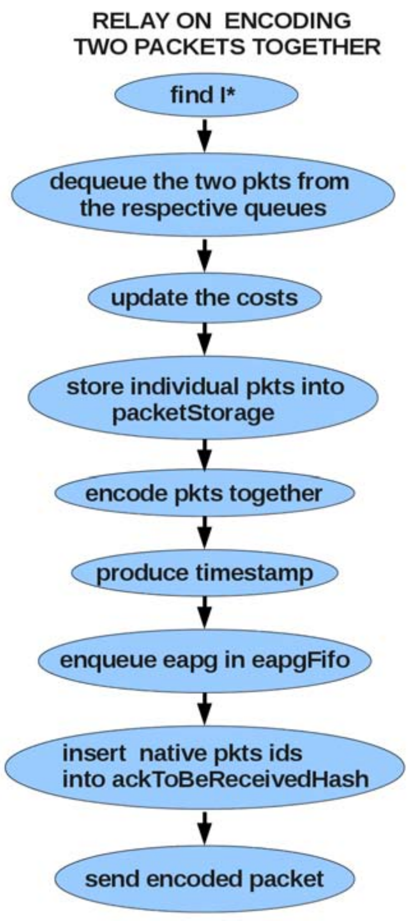


Figure 21: Router node when encoding two packets together.

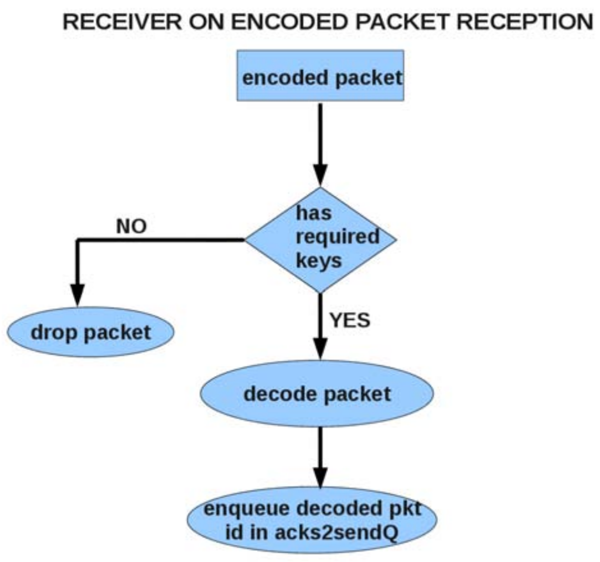

RECEIVER ON ENCODED PACKET RECEPTION
encoded packet
has required keys
NO
drop packet
YES
decode packet
enqueue decoded pkt id in acks2sendQ

Figure 22: Receiver actions when receives an encoded packet.

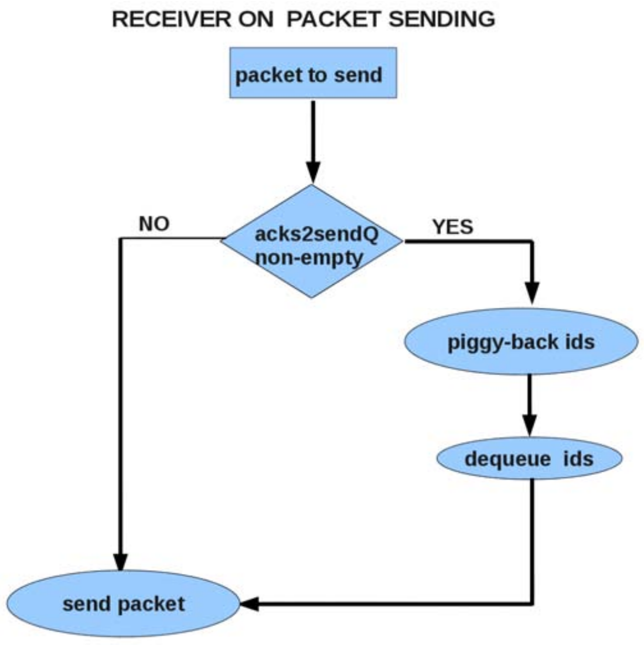


Figure 23: Receiver when sending a native packet.

The throughput region of our system, determined after several experiments for the case of 12 Mbps with $q_{12} = 0.5$ and $q_{21} = 0.8$ is depicted below:

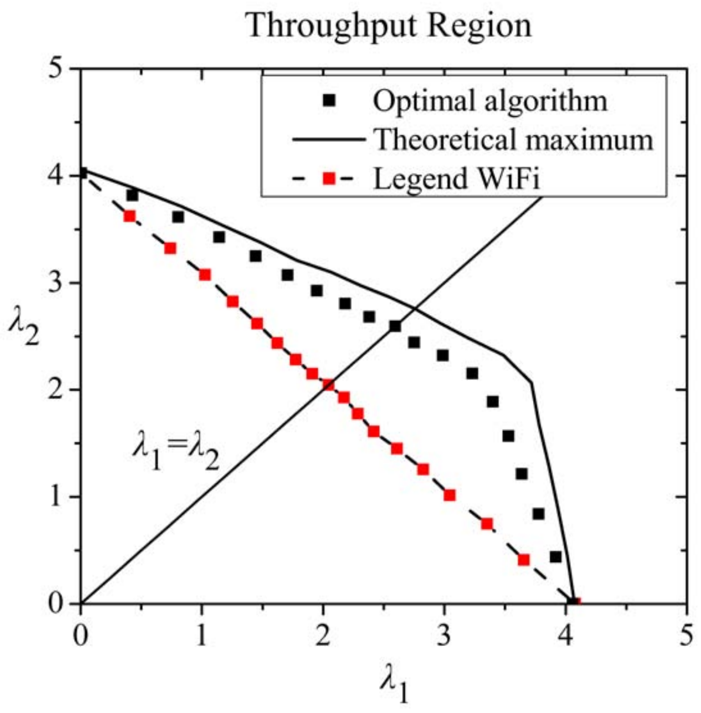


Figure 24: The throughput region for the case of 12Mbps links.

The above results were achieved, while the router (which has the most computational intensive tasks among the other nodes) utilized 8-10 % of its CPU (we are refering to an orbit node which is equiped with an 1 GHz VIA C3 proccessor).

References

[1] R. W. Yeung, “Multilevel diversity coding with distortion,” in IEEE Trans. Inform. Theory, vol. 41, no. 2, pp. 412–422, Mar. 1995.

[2] G. Bianchi, “Performance Analysis of the IEEE 802.11 Distributed Coordination Function”, in IEEE JSAC, Vol. 18, pp. 535-547, 2000.

[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] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A high-throughput path metric for multi-hop wireless routing", in ACM MobiCom ’03, San Diego, California, September 2003.

[6] D. S. Lun, N. Ratnakar, R. Koetter, M. Mdard,E. Ahmed, and H. Lee, "Achieving Minimum-Cost Multicast: A Decentralized Approach Based on Network Coding", in IEEE INFOCOM, 2005.

[7] Georgios S. Paschos, Leonidas Georgiadis and Leandros Tassiulas, "Optimal scheduling with pairwise XORing of packets under statistical overhearing information and feedback"

[8] MIT Roofnet. http://pdos.csail.mit.edu/roofnet

[9] D. S. J. De Couto, D. Aguayo, J. Bicket, and R. Morris, "A High Throughput Path Metric for MultiHop Wireless Routing", in ACM MOBICOM, 2003.

[10] R. Draves, J. Padhye, and B.Zill, "Routing in Multi-Radio, Multi-Hop Wireless Mesh Networks", in ACM MOBICOM, 2004.

[11] David B. Johnson, David A. Maltz, and Yih-Chun Hu, "The Dynamic Source Routing protocol for mobile ad hoc networks (DSR)", in IETF, 2003.

A Fundamentals of Computer Networking

A.1 OSI and TCP/IP Models

The Open Systems Interconnection model (OSI model) is a product of the Open Systems Interconnection effort at the International Organization for

Standardization. It is a prescription of characterizing and standardizing the functions of a communications system in terms of abstraction layers. Similar communication functions are grouped into logical layers. An instance of a layer provides services to its upper layer instances while receiving services from the layer below. The layers and the interaction between them can be seen in Figure 25. A short description of the purpuse of each layer is as follows:

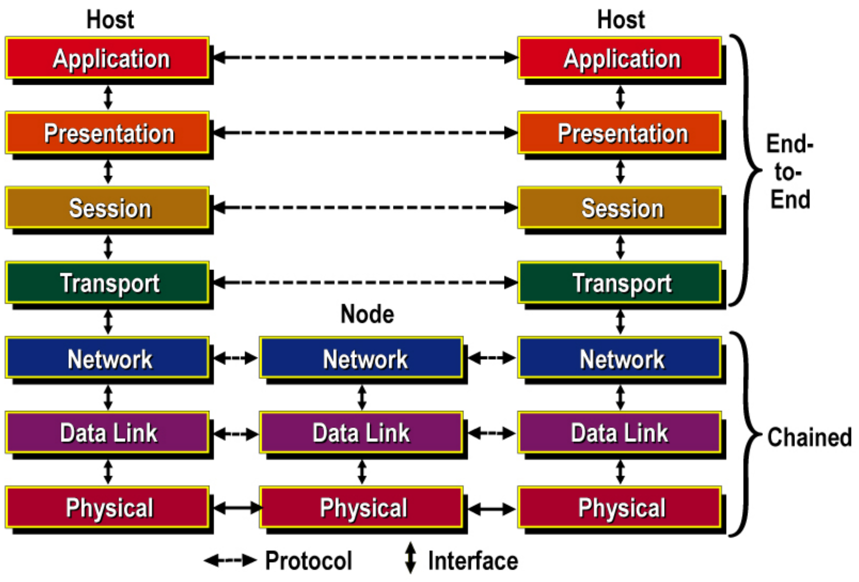


Figure 25: OSI model layers.

- Application (Layer-7): provides services directly to user applications. Because of the potentially wide variety of applications, this layer must provide a wealth of services. Among these services are establishing privacy mechanisms, authenticating the intended communication partners and determining if adequate resources are present. Application layer functions typically include identifying communication partners, determining resource availability, and synchronizing communication. When identifying communication partners, the application layer determines the identity and availability of communication partners for an application with data to transmit. When determining resource availability, the application layer must decide whether sufficient network or the requested communication exist. Examples of Layer-7 include Telnet, FTP, NFS, NIS, etc. The data unit of this layer is a message or stream (both could be named just data).

- Presentation (Layer-6): performs data transformations to provide a common interface for user applications, including services such as reformating, compresing and data encryption. An example is the service for the conversion of an EBCDIC-coded text file to an ASCII-coded one.

- Session Layer (Layer-5): establishes, manages and ends user connections while also manages the interaction between systems. Services include such things as establishing communications as full or half duplex and grouping data. An example of a Session Layer Protocol is the OSI protocol suite Session Layer Protocol, also known as X.235 or ISO 8327. In case of a connection loss this protocol may try to recover the connection. If a connection is not used for a long period, the Session Layer Protocol may close it and re-open it. Session layer services are commonly used in application environments that make use of remote procedure calls (RPCs).

- Transport (Layer-4): insulates the three upper layers (5 through 7) from having to deal with the complexities of the layers below (1 through 3), by providing the functions necessary to guarantee a reliable network link. Among other functions this layer provides error recovery and flow control between the two end-points of the network connection. The main example protocols are TCP (Transmission Control Protocol), UDP (User Datagram Protocol) and RDP (Reliable Datagram Protocol). The data unit of this layer is a segment (TCP case) or Datagram (UDP case).

- Network (Layer-3): establishes, maintains and terminates network connections. Among other functions, standards define how routing and relaying are handled. Two examples of network layer protocols are IP (Internet Protocol) and ICMP (Internet Control Message Protocol). The data unit of this layer is an IP Datagram or Packet.

- Data Link or MAC (Layer-2): ensures the reliability of the physical link established at Layer-1. Standards define how data frames are recognized and provide flow control and error handling at the frame level. Protocol examples of Data link layer are Ethernet, Frame Relay, 802.11 wireless Lan, PPP(Point-to-Point Protocol), Token Ring, etc. The data unit of this layer is a Frame.

- Physical (Layer-1): controls transmission of the raw bitstream over the transmission medium. Standards for this layer define parameters such as the amoung of signal voltage swing, the duration of symbols (grouped bits) and so on. Physical layer protocols are the DSL (Digital Subscriber Line), ISDN (Integrated Services Digital Network), SONET (Synchronous optical networking), etc. The data unit of this layer is a bit.

OSI reference model came into existence way before TCP/IP model was created. Advance Research Project Agency (ARPA) created OSI reference model so that they can logically group the similarly working components of the network into various layers of the protocol. But after the advent of the Internet, there arose the need for a streamlined protocol suite, which would address the need of the ever growing Internet. So the Defense Advanced Research Project Agency (DARPA), decided to create TCP/IP protocol suite. This was going to address many, if not all the issues that had arisen with OSI reference model. Even though the concept is different from the OSI model, the layers in TCP/IP are nevertheless often compared with the OSI layering scheme in the following way: the TCP/IP protocol suite consists of four protocol layers that approximately correspond to the OSI Reference Model, as depicted in Figure 26. TCP/IP's Network Interface Layer de-

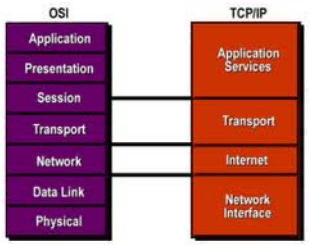


Figure 26: OSI vs TCP/IP.

fines the interface between the host system and the network medium. It is specific to the network implementation and roughly corresponds to the combined OSI Reference Model Physical Layer and Data Link Layer; some OSI Network Layer functionality could also be included. The Internet Layer defines communications between hosts on networks. It provides the path to link these networks into a single internetwork. Using IP and a number of adjunct protocols, the Internet Layer can efficiently route packets across the internetwork. It corresponds to the OSI Network Layer. The OSI Transport Layer provides reliable end-to-end communication between hosts, while the OSI Session Layer provides end-to-end communication between two communicating processes within the hosts. TCP provides a reliable, assured, connection-oriented service between two hosts, and UDP provides an unreliable, connectionless service between two hosts. TCP and UDP offer some Session Layer functionality since they provide addressing for higher layer applications. TCP/IP's Application Services Layer provides the end-user window into the network as well as useful functions for the user. It is functionally similar to OSI Layers 5–7 (i.e., Session, Presentation, Application) and offers such applications as e-mail, file transfer, remote terminal access, and access to the World Wide Web (WWW). For the purpuses of the present diploma thesis, we will adopt the "Five-layer Internet model" or "TCP/IP protocol suite" (James F. Kurose, Keith W. Ross, Computer Networking: A Top-Down Approach, 2008), visualized in Figure 27. In Figure 3 it is also described the encapsulation of a packet i.e as the packet moves down to lower layers, each layer(more specifically each layer of the Kernel's TCP/IP protocol stack) adds its respective header to the packet. Then the packet is pushed to the next layer until reaching the last one (Layer-1) and gets transmitted. In this Figure, H1 denotes a TCP or UDP header, H2 the Network header and H3 the Data Link header. The reverse procedure is called packet decapsulation and is taking place when a node receives a packet, until the data of the packet is delivered to the application.

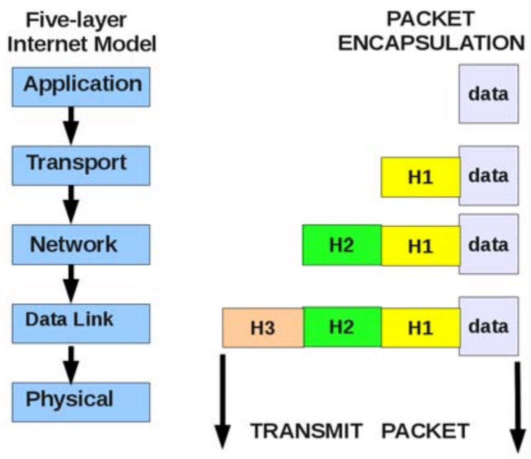


Figure 27: TCP/IP protocol suite model and packet encapsulation.

A.2 Wireless Networks

A Wireless network refers to any type of computer network that is not connected by cables of any kind. It is a method by telecommunications and enterprise (business) installations to avoid the costly process of introducing cables into a building, or as a connection between various equipment locations. Wireless telecommunications networks are generally implemented and administered using radio waves as their transmission medium. This implementation takes place at the physical level (layer) of the network structure.

A wireless ad hoc network is a decentralized type of wireless network. The network is ad hoc because it does not rely on a preexisting infrastructure, such as routers in wired networks or access points in managed (infrastructure) wireless networks. Instead, each node participates in routing by forwarding data for other nodes, and so the determination of which nodes forward data is made dynamically based on the network connectivity. In addition to the classic routing, ad hoc networks can use flooding for forwarding the data. The earliest wireless ad hoc networks were the "packet radio" networks (PRNETs) from the 1970s, sponsored by DARPA after the ALOHAnet project. The decentralized nature of wireless ad hoc networks makes them suitable for a variety of applications where central nodes can't be relied on, and may improve the scalability of wireless ad hoc networks compared to wireless managed networks, though theoretical and practical limits to the overall capacity of such networks have been identified. Minimal configuration and quick deployment make ad hoc networks suitable for emergency situations like natural disasters or military conflicts. The presence of a dynamic and adaptive routing protocols enable ad hoc networks to be formed quickly.

Wireless ad hoc networks can be further classified by their application:

- wireless mesh networks (WMN)
- mobile ad hoc networks (MANET)

- wireless sensor networks (WSN)

In this particular diploma thesis we will deal with wireless mesh networks. A WMN is a communications network made up of radio nodes organized in a mesh topology. Wireless mesh networks often consist of mesh clients, mesh routers and gateways. The mesh clients are often laptops, cell phones and other wireless devices while the mesh routers forward traffic to and from the gateways which may but need not connect to the Internet. The coverage area of the radio nodes working as a single network is sometimes called a mesh cloud. Access to this mesh cloud is dependent on the radio nodes working in harmony with each other to create a radio network. A mesh network is reliable and offers redundancy. When one node can no longer operate, the rest of the nodes can still communicate with each other, directly or through one or more intermediate nodes. Wireless mesh networks can be implemented with various wireless technology including 802.11. As mentioned above wireless mesh network can be seen as a special type of wireless ad hoc network. A wireless mesh network often has a more planned configuration, and may be deployed to provide dynamic and cost effective connectivity over a certain geographic area. The mesh routers may be mobile, and be moved according to specific demands arising in the network. Often the mesh routers are not limited in terms of resources compared to other nodes in the network and thus can be exploited to perform more resource intensive functions. In this way, the wireless mesh network differs from an ad-hoc network, since these nodes are often constrained by resources.

Wireless networks have gained increasing popularity because of their ability to allow the components of a system to stay connected and WMNs have emerged as a key technology for next-generation wireless networking. Mesh networks are self configuring, self managing, and self healing. When a mesh node powers up, it broadcasts and listens to identification messages from neighbor nodes and a network is thus self formed. Their dynamic reconfiguration ability ensures that failure of a particular link to a node does not lead to node isolation. Mesh networks can cover a wider geographical area without having to establish additional backhaul communication links, resulting in a cost effective technology. Hence WMNs have been accepted as a fast, low-cost, and easily extensible solution for providing network connectivity and coverage to distributed users in a wide area. The ease of maintenance, robustness, and reliability of these networks makes them suitable for varied applications.

As far as it concerns the implementation of wireless computer networks Wi-Fi or IEEE 802.11 is a set of standards for this purpose in the 2.4, 3.6 and 5 GHz frequency bands. The most popular are 802.11b and 802.11g protocols, which are amendments to the original standard. 802.11-1997 was the first wireless networking standard, but 802.11b was the first widely accepted one, followed by 802.11g and 802.11n. Security was originally purposefully weak due to export requirements of some governments, and was later enhanced via the 802.11i amendment after governmental and legislative changes. 802.11n is a new multi-streaming modulation technique. Other standards in the family (c–f, h, j) are service amendments and extensions or corrections to the previous specifications. 802.11b and 802.11g use the 2.4 GHz ISM band. Because of this choice of frequency band, 802.11b and g equipment may occasionally suffer interference from microwave ovens, cordless telephones and Bluetooth devices. 802.11b and 802.11g control their interference and susceptibility to interference by using direct sequence spread spectrum (DSSS) and orthogonal frequency division modulation (OFDM) signaling methods, respectively. Finally 802.11a uses the 5 GHz U-NII band, which, for much of the world, offers at least 23 non-overlapping channels rather than the 2.4 GHz ISM frequency band, where all channels overlap. Better or worse performance with higher or lower frequencies (channels) may be realized, depending on the environment. The knowledge of the operations of 802.11 medium access control protocol helps in the understanding of the time required for a packet transmission in a wireless ad hoc network. In 802.11 protocols, the fundamental channel access mechanism is based on the Distributed Coordination Function (DCF) mode [2]. It is a decentralized algorithm and does not require a single node to monitor or coordinate the channel access scheme. The two techniques employed by the DCF mode are the basic access mechanism and the RTS/CTS method. The basic access method involves the transmission of ACK packets from the destination node after the reception of the packet from the source node. In the case of RTS/CTS mechanism, the source node first sends the Request To Send (RTS) packet and waits for the Clear To Send (CTS) packet from the destination node. This is followed by the actual data transmission and the reception of the ACK packet from the destination. The random channel access in 802.11 networks is based on the Carrier Sense Multiple Access Collision Avoidance (CSMA/CA) scheme. When a data packet is ready to be sent, the protocol senses the channel for ongoing transmissions. If the channel is observed as free for a particular period of time called Distributed Inter Frame Size (DIFS), the DCF mode initializes the back-off counter and waits till the counter becomes zero before attempting transmission. The packet is transmitted when the counter reaches zero. Upon successful transmission, the next packet is chosen from the queue. The packet transmission may fail, if a collision is encountered with any other packets in the network and a back-off counter is chosen at random from a uniform distribution. A maximum of M transmissions are attempted before the packet is discarded. A simplyfied flow chart of the CSMA/CA is presented in Figure 28.

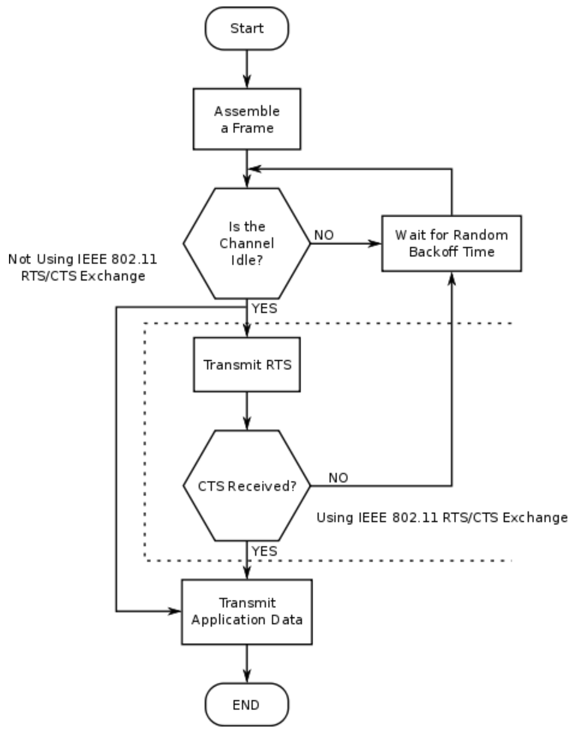

Start
Assemble a Frame
Is the Channel Idle?
NO
Wait for Random Backoff Time
Not Using IEEE 802.11 RTS/CTS Exchange
YES
Transmit RTS
CTS Received?
NO
Using IEEE 802.11 RTS/CTS Exchange
YES
Transmit Application Data
END

Figure 28: Simplyfied CSMA/CA procedure.