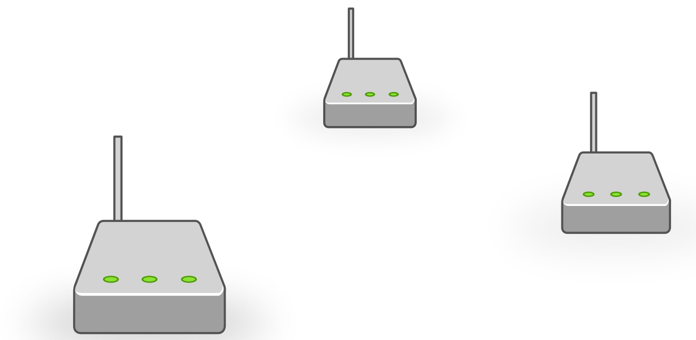
Inter-Flow Network Coding
for
Wireless Mesh Networks



Group 11gr1002

Martin Hundebøll
Jeppe Ledet-Pedersen

Master Thesis in Networks and Distributed Systems
Aalborg University
Spring 2011

AALBORG UNIVERSITET

The Faculty of Engineering and Science
Department of Electronic Systems
Frederik Bajers Vej 7
Phone: +45 96 35 86 00
http://es.aau.dk

Title:
Inter-Flow Network Coding for Wireless Mesh Networks

Project period:
February 1st - May 31st, 2011

Project group:
11gr1002

Group members:
Martin Hundebøll
Jeppe Ledet-Pedersen

Supervisor:
Professor Frank H.P. Fitzek

Number of copies: 4
Number of pages: 65
Appended documents:
(2 appendices, 1 CD-ROM)
Total number of pages: 71
Finished: June 2011

Abstract:

This report documents the development and implementation of the CATWOMAN (Coding Applied To Wireless On Mobile Ad-hoc Networks) scheme for inter-flow network coding in wireless mesh networks.
Networks that employ network coding differ from conventional store-and-forward networks, by allowing intermediate nodes to combine packets from independent flows.
CATWOMAN builds on the B.A.T.M.A.N Adv. mesh routing protocol. The scheme exploits the topology information from the routing layer to automatically identify coding opportunities in the network. The network coding scheme is implemented in the B.A.T.M.A.N Adv. Linux kernel module, and can be used without modification to device drivers or higher layer protocols.
The protocol is tested in three different topologies, that are configured in a test network with five nodes. The coding scheme shows up to 62% increase in maximum achievable throughput for bidirectional UDP flows. The tests reveal an unequal allocation of transmission slots, when the nodes have different link qualities, with a preference for the node with the strongest link. Tests with TCP shows only moderate or no gain for unidirectional traffic. Bidirectional TCP flows show asymmetric behavior, depending on link quality to the nodes. The performance evaluation of CATWOMAN shows promising first results and indicate that transparent network coding can be a beneficial improvement to real wireless mesh networks.

Table of Contents

Table of Contents **iii**

Preface **v**

1 Introduction **1**
1.1 Wireless Mesh Networks . . . 1
1.2 Routing in Wireless Mesh Networks . . . 2
1.3 Network Coding . . . 3
1.4 IEEE 802.11 Wireless Networks . . . 6

2 Project Description **11**
2.1 Routing Protocol . . . 11
2.2 Contributions of this Project . . . 11
2.3 Requirements . . . 12
2.4 Delimitations . . . 12

3 Protocol Design **13**
3.1 The B.A.T.M.A.N Adv. Routing Protocol . . . 13
3.2 Coding Opportunity Discovery . . . 17
3.3 Receiver Selection . . . 18
3.4 Coding Decisions . . . 19
3.5 Coding and Decoding Packets . . . 22
3.6 Protocol Packets . . . 24

4 Implementation **25**
4.1 Structure of B.A.T.M.A.N Adv. . . . 25
4.2 B.A.T.M.A.N Adv. Implementation . . . 27
4.3 CATWOMAN Implementation . . . 29
4.4 Data Structures . . . 34
4.5 Buffering . . . 37

5 Test Environment **41**
5.1 Experimental Setup . . . 41
5.2 UDP Performance Tests . . . 42
5.3 TCP Performance Tests . . . 44

6 Test Results **45**
6.1 Alice-and-Bob Topology . . . 45
6.2 X Topology . . . 51
6.3 Crossed Alice-and-Bob Topology . . . 56
6.4 TCP Performance . . . 59

6.5 Evaluation Summary 61

7 Conclusion **63**
7.1 Further Work 64

A CATWOMAN Configuration Tutorial **65**

B Attached CD **69**

Bibliography **71**

Preface

To evaluate the performance gain achieved by applying network coding to wireless mesh networks, this project designs and implements the CATWOMAN protocol. The protocol is based on the B.A.T.M.A.N Adv. open-source mesh routing protocol, which provides the structure and topology information used in CATWOMAN.

The project is conducted in the spring of 2011 and is the Master thesis of the authors. It is conducted in accordance with the study programme of Networks and Distributed Systems at the Section of Networking and Security, Department of Electronic Systems at Aalborg University.

This report is the documentation of the design, implementation, and evaluation of CATWOMAN. It briefly describes the technologies used in the project, before diving into the design of the mesh and network coding protocols, and how the two are implemented and integrated. The protocol testing is described and the results from the performance evaluation are given and discussed in detail. The project is concluded and proposals for further enhancements are presented.

Motivation

The choice of project is based on the topics of protocol design, network coding, and experimental evaluation. During our master programme, we have both worked with all three topics and this project gives further experience with these. In addition, the B.A.T.M.A.N Adv. kernel module has allowed us to dig into the exciting world of Linux kernel programming.

The fields of network coding and wireless mesh networks are evolving rapidly, and are clear-cut subjects for applicable research.

Acknowledgements

While conducting the project, feedback has been received from several persons and we would like to thank them all. Specifically, the following are thanked for competent help and feedback with both the protocol work and the documentation.

Morten V. Pedersen and Janus Heide for their helpful informal discussions and guidance with the theory of network coding and protocol design. The team behind B.A.T.M.A.N Adv., especially Marek Lindner, Antonio Quartulli, Linus Lüssing, and Simon Wunderlich, for their help with the

B.A.T.M.A.N Adv. and feedback on our work with implementing CATWOMAN. Also, a thanks goes to Open-Mesh for sponsoring ten mesh routers for the test environment.

Notes and References

The report makes use of cross references. Figures and tables are numbered and these numbers are used as references in the text. In some cases, the size of a figure or table makes it necessary to place it on the following page.

When using external sources, information about the source is given in the bibliography on page 71. The source information is referenced by a text and year encapsulated in braces. A reference to the example source is [Example08b].

Author Signatures

Martin Hundebøll   Jeppe Ledet-Pedersen

Chapter 1

Introduction

Since the first demonstration of wireless data communication in 1970[Abr70], devices ranging from embedded microprocessors to automobiles and trains now utilize the technology of wireless communications.

One widespread standard for wireless communication is the IEEE 802.11 specification[IEE99], which supports both managed infrastructure setups, where the network is deployed with static access points, and ad-hoc setups, where each station participates independently in the network.

This chapter describes the principles of communications and routing in these ad-hoc networks and introduces the idea of exploiting the overhearing of communication in wireless networks to improve throughput. It also describes some of the widely deployed routing protocols and the differences among them.

1.1 Wireless Mesh Networks

In wireless networks with managed infrastructure, routing information is often provided by a central entity, e.g. an access point. In ad-hoc mode, there is no central entity to provide access to other parts of the network and route between nodes. In this case, mesh networking is used to gather and distribute the information needed to provide this access and routing.

Mesh networks consist of mesh routers, which provide access to the existing infrastructure, and mesh clients, which both provides multi-hop connectivity to the mesh routers and utilizes the connectivity provided by other mesh clients. The difference of managed and mesh networks is illustrated in Figure 1.1.

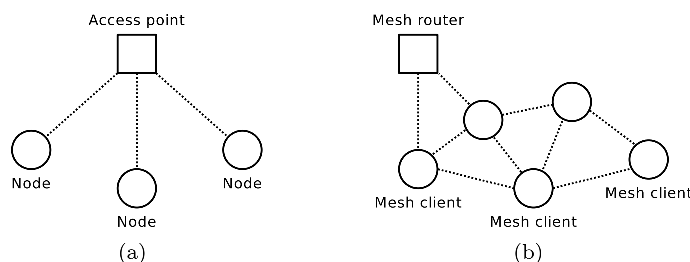


Figure 1.1: Wireless links in managed infrastructure mode (a) and in ad-hoc mode with mesh routing (b). In (a) access to other networks and nodes is provided by a central entity and in (b), the access is provided by both other nodes and mesh routers.

Wireless mesh networking is a promising next-generation technology for self-organising, mobile, and flexible networking, e.g. preliminary communication in disasters, inexpensive coverage in buildings and areas without existing cabling, extension and collaboration for cellular networks, etc. The primary advantage of mesh networks is the ability to quickly setup cheap and large networks where no existing infrastructure is available.

There exist several projects developing routing protocols for wireless mesh networks and an amendment for the 802.11 standard is in draft[HDM+10]. The protocol is however not widely deployed and other more used protocols are developed concurrently by communities throughout the world. In Europe, communities use and develop mesh networks to provide free and neutral internet access. Some of the biggest communities are Freifunk in Germany with more than 100 independent networks and Guifi.Net in Catalonia with a network that consists of more than 12,000 active nodes.

1.2 Routing in Wireless Mesh Networks

In mesh networks, where one or more nodes are out of range of other nodes, routing at intermediate nodes is needed to facilitate communication between distant nodes. Several mesh routing protocols exists and this section describes some of the most used ones and how they differ.

Mesh routing protocols often operate in a distributed manner, where each node in the network gathers information about its surrounding nodes and distributes this to other nodes in the network. Each node then makes routing decisions based on the information received from other nodes. Protocols are classified as being either proactive or reactive. Proactive protocols broadcast messages to announce presence and build routes and reactive protocols request routes on demand.

Mesh networks suffers from architectural issues that routing protocols must take into consideration. Some of these issues are:

- The shared communication medium limits routing performance.
- The protocol introduce overhead to discover routes and nodes
- Roaming is required to handle mobile nodes.
- Connections are temporarily lost when a routing node disappears.

Different protocols approach the issues with different methods, but common to all is that they try to minimize overhead and maximize robustness and connectivity.

1.2.1 Routing Methods

Routing protocols are divided into two types: *Reactive* protocols and *proactive* protocols. If the gathering of information and calculation of routes take place at the point where a route to a node is needed, the protocol is reactive. If the information about the network is distributed and routes are calculated whenever a change in the network occurs, the protocol is considered proactive.

The majority of mesh routing protocols use one of two methods to gather the information used when making routing decisions: Link state routing and distance vectors.

In protocols using link state routing, nodes floods the network with a map of its connected neighbours. Every node in the network uses these informations to calculate a complete map of the network, containing the shortest logical path to every node, which is selected when routing packets. The topology calculation is performed whenever a change in the network occurs and can be a complex task, causing some protocols to divide the network into a hierarchy, so that only a part of the network is calculated.

When using distance vector routing, every node in the network knows only the direction (or next hop) to which a packet should be routed. The next hop is chosen based on the lowest hop count and best connectivity towards a node. Distance vector routing does not compute a complete map of the network and therefore requires less resources at the price of less knowledge about the network. Therefore the routing decisions can not always be optimal.

The distributed routing must handle loops in the network, which occurs (in the simple case) if two nodes on a path both has the other node as the next hop towards the destination. In this case, packets are looping between the two nodes, which is known as routing loops.

1.2.2 The Optimized Link State Routing Protocol

Optimized Link State Routing (OSLR) is a widely spread mesh routing protocol that is deployed in many mesh routers running an embedded Linux. It is implemented as a daemon working on the IP layer, is proactive, and uses link state routing. The protocol is documented in RFC 3626, but early implementations exposed issues with the standard and extensions has since been added.

In the first implementations, the computation of topology maps was slow and often invalidated by changes. Routes were rapidly changing and looping, making the network hardly usable. To counter these issues, an extension was made to include the quality of links in routing decisions by using the Expected Transmission Count (ETX) as a metric.[DABM03] This was a major improvement, but did not solve the issues of routing loops. By sending topology control (TC) messages more often, the frequency of routing loops occurrence was decreased. The overhead introduced by the more frequent TC messages was limited by only broadcasting more often to nodes one, two, and three hops away, which showed to be sufficient. The adjustment of TC transmissions was named Link Quality Fish Eye mechanism and made OSLR usable, although routing loops still occasionally turned up.[Aic06]

1.2.3 The Ad hoc On-Demand Distance Vector Routing Protocol

Ad hoc On-demand Distance Vector (AODV) Routing is a reactive distance vector protocol developed partly by Nokia. The protocol works by sending a broadcast requesting routes towards the destination of a pending packet and selecting the best route returned. This mechanism minimizes the overhead traffic when the network is idle. (As opposed to proactive protocols, where control messages are flooded periodically.) The protocol is especially useful in sensor networks, where low power consumption is a requirement during silent periods.

1.2.4 The Better Approach To Mobile Ad-hoc Networking Protocol

The Better Approach To Mobile Ad-hoc Networking (B.A.T.M.A.N) protocol was initiated in the spring of 2006 as an alternative to OLSR and uses proactive distance vector routing. During the first three versions of the protocol, several features were added to the protocol: Asynchronous link awareness, multiple interfaces support, and gateway connectivity. Similar to OSLR, the protocol is implemented as a user space daemon that routes on the IP layer.

1.2.5 The B.A.T.M.A.N Adv. Routing Protocol

Due to performance issues caused by running the B.A.T.M.A.N. routing daemon in user space, the development of B.A.T.M.A.N Adv. was initiated in 2007. Besides running in kernel space and thus avoid expensive copying of packet to and from user space, the new protocol routes on the link layer.

B.A.T.M.A.N Adv. encapsulates all traffic from entry to exit and thereby acts as a virtual switch between all nodes in the network. This adds support for other network protocols than IPv4 and also makes other features easier to integrate. Today B.A.T.M.A.N Adv. supports bridging and roaming of non-mesh clients.

In march 2011, B.A.T.M.A.N Adv. was added to the Linux kernel mainline tree and has now deprecated the user land daemon.

1.3 Network Coding

Network Coding changes the way that nodes relay packets in a multi-hop network: Instead of simple store-and-forward, packets can be combined and forwarded. By applying network coding,

the upper bound for a flow in a multicast scenario, as described in the *max-flow min-cut theorem*, can be reached.[ACLY00]

The concept of network coding is illustrated in Figure 1.2, which illustrates the butterfly network. The network consists of a single source s, two sinks t_1, t_2, and four forwarding nodes $1, 2, 3, 4$. The source transmits two packets (b_1 and b_2) to the sinks. In Figure 1.2(a) node 3 must select one packet to forward and thus the deselected packet is delayed. When applying network coding, as illustrated in Figure 1.2(b), node 3 can combine the two packets, which are then extracted at the sinks and thus none of the packets are delayed. The combining of packets is performed with the XOR operation, which can be reverted by XORing the combined packet with one of the original packets: $(1 \otimes 2) \otimes 2 = 1$.

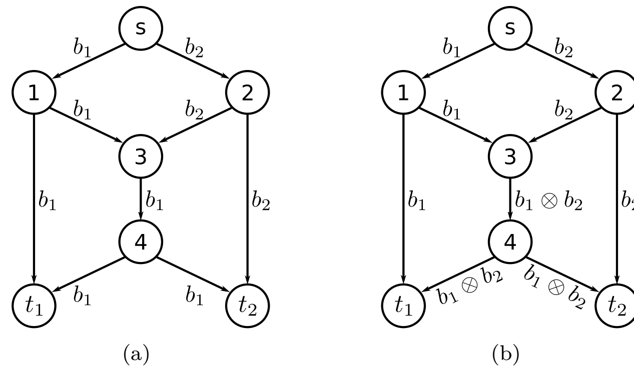


Figure 1.2: The butterfly network, in which store-and-forward (a) and network coding (b) is applied. The source, Node S, transmits two packets: One towards Node t_1 and one towards Node t_2. With network coding, one transmission can be avoided by combining the two packets at Node 4.

Network coding that combines multiple packets from one source is known as *intra-flow coding*, whereas network coding that combines packets from two or more separate flows is know as *inter-flow coding*. In the case illustrated in Figure 1.2, b_1 and b_2 are both from the same node, so the network coding performed at node 4 is intra-flow coding.

1.3.1 Network Coding in Wireless Mesh Networks

Throughput in wireless mesh networks can be increased by applying inter-flow network coding when forwarding packets. The idea is to exploit the shared medium to overhear packets and use these for decoding of network coded packets.

A simple example of network coding in wireless mesh networks is the scenario illustrated in Figure 1.3, where one node forwards packets between two other nodes. If Node A and Node B stores their transmitted packets, these can be used to decode the network coded packet sent by node Node R and thus one transmit is saved by node Node R compared to store-and-forward routing.

In [KRH$^+$06] the COPE protocol is developed to evaluate the performance of network coding in wireless mesh networks. The gain achieved by applying network coding in COPE is analysed for four topologies: Alice-and-Bob, X, Cross, and Wheel. The four topologies are illustrated in Figure 1.4.

1.3.2 Analytic Coding Gains

The gain obtained by coding packets is defined as the ratio of required transmissions *without* network coding to the required transmissions *with* network coding. In the Alice-and-Bob topology from Figure 1.4(a), one transmission is saved, so the coding gain is $4/3 = 1.33$.

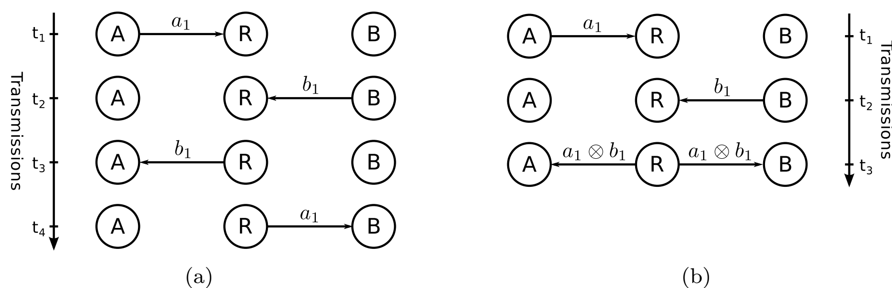


Figure 1.3: The Alice-and-Bob scenario, where node R relays packet a_1 from A to B and packet b_1 from B to A without network coding (a) and with network coding (b).

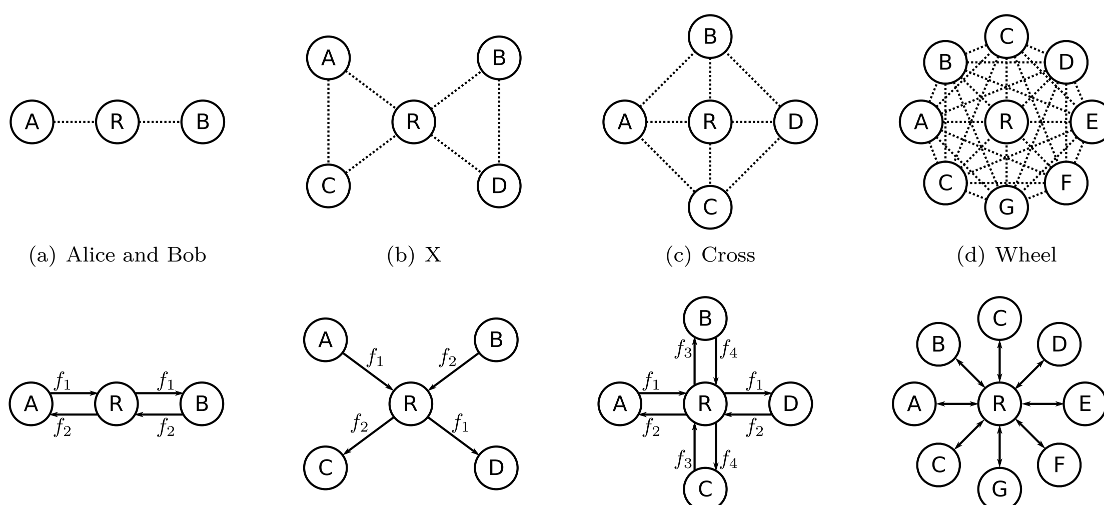


Figure 1.4: Wireless links (dotted lines) and flows (solid lines) in the four topologies analysed in COPE. In (a), one bidirectional flow intersect at Node R. In (b), two unidirectional flows intersect at Node R. In (c), two bidirectional flows intersect at Node R. In (d), multiple flows intersect at Node R.

The Alice-and-Bob topology is similar to the X topology in Figure 1.4(b), except that the receiving nodes use overheard packets to decode network coded packets from the relay. The potential coding gain is thus the same for the X topology, but coding should happen more often, as the bidirectional flows in Alice-and-Bob is assumed to be more rare than the unidirectional flows in the X topology.

In the Cross topology in Figure 1.4(c), the vertical nodes overhear bidirectional flow from the horizontal nodes and vice versa. In this topology, where four flows intersect in the relay node, four packets can be combined and the relay thus saves three transmissions. This leads to a potential coding gain of $^8/_5 = 1.6$.

In the fourth and rather unlikely Wheel topology in Figure 1.4(d), the outer nodes overhear all other nodes except the opposite. In this case, the relay node can code n packets in one transmission and the coding gain goes towards towards 2 as the number of outer nodes, n, increases: $^{2n}/_{n+1}$, The nominator is the number transmissions needed without network coding (one transmission each outer node and one forward to each outer node.) and the denominator is the number of transmissions with network coding. (One transmission from each outer node and a single combined transmission from the relay.)

In each scenario the coded packets are received by more than one node. Since multiple destinations is not supported by the physical layer, the nodes receive and process every received packet. In the last three scenarios (X, Cross, and Wheel topologies), overheard packets must be saved and used in later decoding. This requirement thus causes additional processing and storing of packets, which is the drawback of network coding in wireless mesh networks.

1.3.3 Experimental Coding Gains

The observed coding gain with COPE in the first three of the four illustrated topologies with UDP flows are 1.7, 1.65, and 3.5 for Alice-and-Bob, X, and Cross respectively. This is above the analytic gains and is explained by a side benefit obtained when coding: MAC gain. The MAC layer (further described in Section 1.4.2) distributes the access to the medium equally between the nodes.

Without network coding in the Alice-and-Bob and X topologies, $1/2$ of the capacity is needed by the relay node to forward the packets received from Alice and Bob. When the network is congested, only $1/3$ of the capacity is available, causing $1/2 - 1/3 = 1/6$ of the packets to be dropped by the relay. With network coding, the relay needs only one third of the capacity and the throughput is thus higher. In the Cross topology, the relay node receives only $1/5$ of the capacity and $1/2 - 1/5 = 3/10$ of the packets are dropped without network coding.

1.4 IEEE 802.11 Wireless Networks

This section gives a brief introduction to the IEEE 802.11 set of standards. Focus is limited to the features used in ad-hoc networks without a centralized infrastructure, and the section will therefore not touch on topics such as access points, distribution systems and security extensions.

IEEE 802.11 is a collection of standards for wireless local area networks issued by the Institute of Electrical and Electronics Engineers (IEEE). The original standard was published in 1997, and has since been extended with a large number of amendments to allow for higher speeds, improved Quality of Service and stronger security. The remainder of this section will focus on 802.11b networks, as used on the routers in our test environment.

Wireless networks that comply with IEEE 802.11b operate in the unlicensed ISM (Industrial, Scientific and Medical) frequency band on 2.4 GHz and delivers speeds from 1 to 11 Mbit/s depending on channel conditions and load. The frequency spectrum is divided in 14 overlapping channels of 22 MHz, each spaced 5 MHz apart, as shown in Figure 1.5. Multiple logical networks can coexist on the same channel, but have to share the available bandwidth by multiplexing their transmissions. Since the channel allocations overlap, networks are often placed on channels 1, 6 and 11 to avoid interfering with multiple networks. Use of the ISM band is unlicensed, and 802.11 network nodes can thus be subject to disturbing interference from other transmitters such as Bluetooth and ZigBee devices.

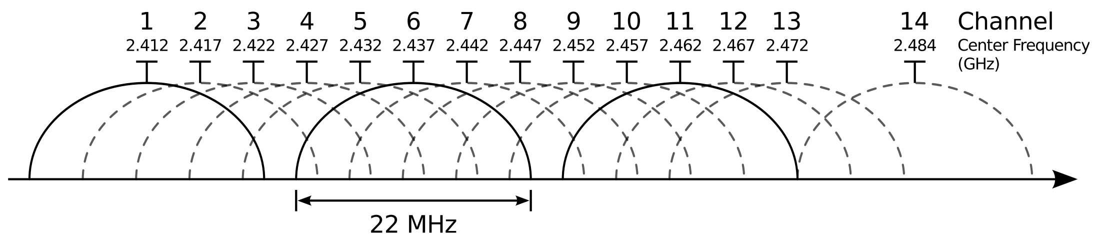


Figure 1.5: Frequencies of the 14 IEEE 802.11 channels in the 2.4 GHz ISM S-band. The channels overlap, so most networks are placed on channel 1, 6 and 11. The use of channel 13 and 14 is restricted in some countries.

Networks can either operate in infrastructure mode where clients exchange data via an associated access point, or in ad-hoc mode where clients communicate directly with each other. A

collection of network clients operation on the same frequency is referred to as a Basic Service Set (BSS), and networks without a centralized infrastructure are called an Independent Basic Service Sets (IBSS). Each BSS is identified with a 48-bit BSSID (Basic Service Set ID).

1.4.1 Frame Format

IEEE 802.11 share a number of features with regular 802.3 Ethernet, including 48-bit source and destination addresses. Consequently, upper-layer protocols such as the Address Resolution Protocol (ARP) can still be used without modifications. The generic frame format used by IBSS nodes in 802.11b networks is shown in Figure 1.6. The PLCP (Physical Layer Convergence Procedure) preamble and header are used for receiver synchronization and contain the modulation and bitrate configuration of the following fields. The MAC (Medium Access Control) header specifies, among other, the sender and receiver addresses, and the network BSSID that is used to separate logical networks residing on the same channel. The MAC header is explained in more detail in the following subsection. Encapsulation of higher layer protocols is handled by the LLC (Logical Link Control) header, which contains a type field to identify the encapsulated protocol in the data field. Finally, a four byte Frame Check Sequence (FCS) is appended to protect against random bit errors. The PLCP preamble is transmitted at a fixed rate of 1 Mbit/s, and the PLCP and LLC headers are always transmitted at a fixed rate of 2 Mbit/s. The data can be sent with either 1, 2, 5.5 or 11 Mbit/s.

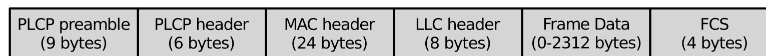


Figure 1.6: Generic 802.11 frame structure.

Frames can be divided in three main types: management, data, and control. Management frames contains subtypes for nodes to associate and deassociate with networks, and frames for probing for the availability of an already known network. The beacon frame that announces the presence and configuration of a network, are also part of the set of management frames.

Figure 1.7 shows the format of data frames in ad-hoc networks. The layout of data frames in infrastructure networks is slightly different, and contains an additional address to allow roaming between different access points. The Frame Control field is used to identify the frame as a data frame, and to inform receiving nodes about various properties of the frame. Frames that are retransmitted are e.g. marked by setting the Retry bit in the Frame Control field. The Duration/ID field is used to defer access to the medium for other nodes, by updating their Network Allocation Vector as explained in the following paragraph. Source and destination addresses serve the same purpose as in 802.3 Ethernet, while the BSSID identify the associated BSS of the sending node. Broadcast and multicast data frames, i.e. frames destined for multiple nodes, will have the broadcast or a multicast Ethernet address in the destination field. Unicast data frames are subject to retransmissions, and therefore contain a sequence field so receiving nodes can discard duplicate frames.

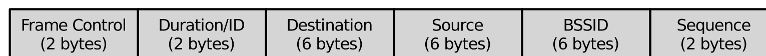


Figure 1.7: 802.11b IBSS Data Frame MAC Header.

Control frames includes data frame acknowledgements, Request To Send (RTS) and Clear To Send (CTS) used for virtual carrier sensing, and various frames to assist nodes in power save mode. The purpose of RTS/CTS frames is explained in the following paragraph.

1.4.2 Medium Access Control

Similar to 802.3 Ethernet, IEEE 802.11b is based on a Carrier Sense Multiple Access (CSMA) scheme to share the physical medium between multiple clients. However, implementing the CSMA with Collision Detection (CSMA/CD) from 802.3, requires nodes to be capable of transmitting and receiving simultaneously, which would increase the complexity and cost of the radio frontend. Instead, 802.11 networks employ a Collision Avoidance (CSMA/CA) access scheme. All ad-hoc network, and many infrastructure networks, use the Distributed Coordination Function (DCF) to divide medium access between contending nodes. Other, more advanced Coordination Functions exists that allows centralized coordination or fine-grained frame priorities.

An 802.11b unicast transmission sequence following the DCF, is illustrated in Figure 1.8. Frame transmission is considered an atomic operation, and the entire sequence of frame exchange must succeed for the transmission to be successful. The sending node starts by transmitting a Request To Send (RTS) frame, containing source and destination addresses and the duration of the entire transmission sequence. The receiving node confirms the transmission by sending a Clear To Send (CTS) frame with its own address and another duration field. The duration fields serves to update the Network Allocation Vector (NAV) of all nodes that overhears the RTS/CTS transmissions. All nodes count down their NAV timer periodically, and defer transmission as long as the NAV is non-zero. The RTS/CTS handshake is an optional feature, and it is often disabled in infrastructure network due to the increased overhead in frame transmissions. Mesh networks though, can benefit from the added protection against hidden terminals, that can cause packet collisions due to uncoordinated transmissions.

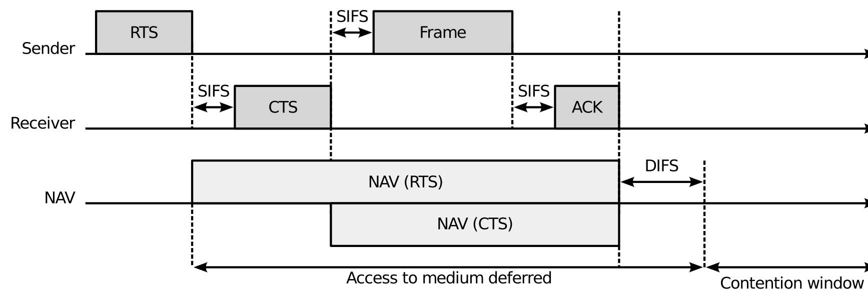


Figure 1.8: Transmission of frame with RTS/CTS handshake and positive acknowledgement [Gas05].

The RTS and CTS frames are separated by a Short Interframe Space (SIFS) of 10 μs. After receving a valid CTS, the transmitting node waits an additional SIFS before transmitting the data frame. 802.11 compliant networks use positive frame acknowledgements, and the receiving node is therefore required to send an acknowledgement frame if the data was received without errors. A new transmission sequence can proceed after the nodes have waited at least one DCF Interframe Space (DIFS) of 50 μs. If the node fails to receive a CTS or ACK frame, it must restart the transmission sequence by sending a new RTS. The number of automatic MAC layer retransmissions continue up to a driver specific limit, usually in the range of 7-16 retransmissions. Only unicast transmissions follow the pattern in Figure 1.8. Broadcast and multicast traffic is transmitted without an RTS/CTS handshake and without acknowledgements. This type of traffic has multiple, possibly unknown, destinations and it is thus not possible to request transmission acceptance or acknowledgement of successful reception. For the same reasons, broadcast traffic is transmitted at a fixed rate, as the transmitting node cannot perform automatic rate adaptation due to the lack of acknowledgements.

Before initiating a transmission sequence, the transmitting node must assess the availability of the channel by listening for ongoing traffic. If the channel is free, the node can start transmitting after waiting for one DIFS, to allow nodes waiting for a SIFS to interrupt. If the channel is not free, the node waits for one DIFS, selects a random slot of length 20 μs in the contention window

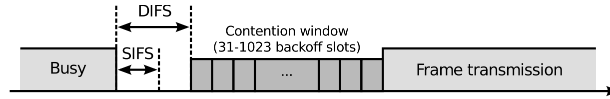


Figure 1.9: 802.11 Contention window [Gas05]. The window initially consists of 31 slots. It is doubled for each congestion event to a maximum of 1023 slots.

and waits for the selected transmission slot. Figure 1.9 illustrates the situation. If no other node starts a transmission before the selected slot comes up, the node transmits immediately. If another node starts transmitting in an earlier contention slot, the node defers its transmission and waits for the newly increased NAV to reach zero. A node that is interrupted while waiting for a contention slot, is allowed to restart its countdown at the value reached before being interrupted. This gives some degree of fairness between nodes, and prevents a highly active node from preventing a less active access to the medium. The number of slots in the contention window starts at 31 and is exponetially increased for every error to a maximum of 1023 slots.

The maximum theoretical throughput for an uncongested 11 Mbit/s 802.11b network with no packet loss and RTS/CTS enabled can now be calculated. The total waiting time due to interframe spacing is:

$$\underbrace{10\mu s}_{\text{RTS SIFS}} + \underbrace{10\mu s}_{\text{CTS SIFS}} + \underbrace{50\mu s}_{\text{Frame DIFS}} + \underbrace{10\mu s}_{\text{ACK SIFS}} = 80\mu s \tag{1.1}$$

Since the network is assumed to be uncongested, there is no contention period and transmission can start immediately after the DIFS. The RTS frame is 20 bytes, and both the CTS and ACK frames are 14 bytes. They are all transmitted at 2 Mbit/s with a 9 byte synchronization preamble and 6 bytes of PLCP header. The preamble is transmitted at 1 Mbit/s, and the PLCP header at 2 Mbit/s. This gives a transmission time of:

$$\underbrace{\frac{9\text{bytes}}{1\text{Mbit/s}} + \frac{6+20\text{bytes}}{2\text{Mbit/s}}}_{\text{CTS Frame}} + \underbrace{\frac{9\text{bytes}}{1\text{Mbit/s}} + \frac{6+14\text{bytes}}{2\text{Mbit/s}}}_{\text{RTS Frame}} + \underbrace{\frac{9\text{bytes}}{1\text{Mbit/s}} + \frac{6+14\text{bytes}}{2\text{Mbit/s}}}_{\text{ACK Frame}} = 457.8\mu s \tag{1.2}$$

The data frame consists of maximum 2312 bytes of data, a 24 bytes MAC header, an 8 bytes LLC header, and a 4 byte FCS all transmitted at 11 Mbit/s. The 6 byte PLCP header is transmitted at 2 Mbit/s with a 9 bit synchronization sequence sent with 1 Mbit/s. The transmission duration is therefore:

$$\underbrace{\frac{9\text{bytes}}{1\text{Mbit/s}} + \frac{6\text{bytes}}{2\text{Mbit/s}}}_{\text{PLCP Header}} + \underbrace{\frac{2312+24+8+4\text{bytes}}{11\text{Mbit/s}}}_{\text{Data and FCS}} = 1720.1\mu s \tag{1.3}$$

Dividing the transmitted 2312 bytes of data with with the total transmission time, gives a maximum achievable throughput of:

$$\frac{2312\text{bytes}}{80\mu s + 457.8\mu s + 1720.1\mu s} = 7.82\text{Mbit/s} \tag{1.4}$$

For 1500 byte packets, the highest achievable throughput is 6.76 Mbit/s. The calculations above are calculated with the *short* frame format that is used by most 802.11b devices. An older, *long* frame format is also specified in the IEEE 802.11b standard and could be used by devices on the network. Long frames use a 24 byte PLCP header and synchronization sequence transmitted with 1 Mbit/s, and will thus have a lower theoretical maximum achievable throughput. The total throughput is to be shared by all nodes on the network, so the experienced throughput on a network with multiple wireless devices should be lower.

Chapter 2

Project Description

Wireless mesh networks constitute a favourable platform for network coding. The capacity of such networks are limited by a subset of notes that route between distant parts of the network. With mesh nodes equipped with omnidirectional antennas, overhearing transmissions between other nodes is possible. The primary goal of this project is to implement transparent network coding to exploit the potential benefit from network coding in wireless mesh networks.

2.1 Routing Protocol

The implementation is based on the B.A.T.M.A.N Adv. routing protocol, and benefits from the existing topology information that is already present in B.A.T.M.A.N Adv. By integrating network coding directly in the routing protocol, coding opportunities can be identified without adding an additional layer for topology discovery. This simplifies the network stack, and avoids additional overhead traffic in the network.

B.A.T.M.A.N Adv. is used because all packets are processed by the kernel space implementation, which simplifies the implementation of network coding. The three other protocols, B.A.T.M.A.N, OSLR, and AODV, are implemented as user space daemons, that operate by altering the kernel routing table and are separated from the packet flows. Without direct access to both the topology information and the packets, the implementation would depend on multiple parts of the system and be more complex.

B.A.T.M.A.N Adv. operates solely on the link layer, and presents a virtual interface to applications. This makes the mesh network transparent to upper layer protocols and applications. By adopting network coding into B.A.T.M.A.N Adv., it inherently becomes protocol agnostic and should present a throughput gain without modifications to applications.

2.2 Contributions of this Project

The project focuses on developing a simple implementation of network coding and experimental evaluation in simplified network topologies with real devices. The implementation is generic and functions in arbitrary topologies without additional configuration and can be turned on and off in a running network. The evaluation is carried out for several aspects of wireless networking and concerns topics as throughput for both UDP and TCP, CPU usage, and delays.

In accordance with the B.A.T.M.A.N Adv. terminology, the project is dubbed CATWOMAN (Coding Applied To Wireless On Mobile Ad-hoc Networks) and the developed product is contributed to the B.A.T.M.A.N Adv. project. By making the implementation a part of the B.A.T.M.A.N Adv. routing protocol, network coding becomes publicly available and be integrated as a standard part of the Linux kernel. Other projects about network coding and mesh networks can extend, modify, and improve the implementation to achieve even greater gains.

2.3 Requirements

The protocol and implementation is designed with the following functional requirements in mind:

Protocol Agnostic CATWOMAN should be transparent to upper layer protocols and not require any modifications or information from higher layer protocols or applications. The network coding layer should e.g. not care if IPv4 or IPv6 is used on the network.

Device Driver Independent CATWOMAN must not depend on specific features in or modifications to wireless device drivers. Not all drivers export detailed statistics from the physical layer, so being dependent on this information would limit CATWOMAN to a subset of devices.

Topology Adaptive CATWOMAN should be deployable in arbitrary network topologies, and should automatically identify coding opportunities without manual configuration from the network administrator. B.A.T.M.A.N Adv. is designed for mesh networks with volatile topologies. CATWOMAN should thus be capable of adapting to changes in the topology without user intervention.

2.4 Delimitations

CATWOMAN is designed to be useful in arbitrary network topologies. However, to keep the original design simple, all network nodes are assumed to be static without mobility. Furthermore, all nodes should have wireless links and be equipped with omnidirectional antennas without asymmetric links.

The evaluation of CATWOMAN is limited to the simple Alice-and-Bob and X topologies, in order to keep the test cases simple and to maintain an understanding of the behavior of the network. For the same reason, CATWOMAN is evaluated with a simple traffic pattern with equal transmission rates from all nodes.

Chapter 3

Protocol Design

To implement network coding in a wireless mesh network, B.A.T.M.A.N Adv. is chosen as the routing protocol into which the network coding functionality is integrated. In order to successfully discover coding opportunities and exploit these with network coding, the B.A.T.M.A.N Adv. protocol must be understood and utilized. This chapter describes the mechanisms relevant to network coding and how these are used in the coding protocol. Furthermore, the coding and decoding of packets is defined.

3.1 The B.A.T.M.A.N Adv. Routing Protocol

The B.A.T.M.A.N Adv. routing protocol described in Section 1.2.5 is used to setup the mesh network and to gather information about nodes in the network, which is needed to discover possible coding opportunities. This section describes the routing protocol and the information it offers.

3.1.1 Packet Types

B.A.T.M.A.N Adv. has six different packet types, which are used for network discovery, different types of data transport and a network visualization tool. Each type is briefly described here.

Batman Packet The batman packet, which are also named Originator Messages (OGMs), is the primary packet of the B.A.T.M.A.N Adv. protocol. It is used to discover nodes and routes in the network and each node in the network creates and broadcasts an OGM at a fixed interval chosen by the node itself. As illustrated in Figure 3.1, neighbor nodes updates their routing table and rebroadcasts batman packets, so that distant nodes also learn about the originating node.

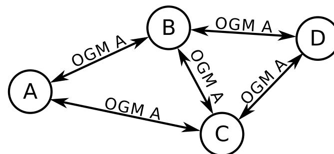


Figure 3.1: Node A broadcats batman packets (OGMs) to its neighbors, who update their routing table and rebroadcast the packet to their neighbors.

OGMs serves two main purposes: 1) Announce the presence of a node and the possible next-hops towards the announced node, 2) measure the quality of the possible routes towards the

announced node. Both are described later in this section. Batman packets contain information about the originator node and the forwarding node:

- Originator address - identifying the node who created the batman packet.
- Sequence number - used for link quality measurement and detection of duplicates.
- Transmit Quality (TQ) - describes the link quality of the total route to the originator.
- Address of the previous sender - used to discard OGMs already broadcasted by the receiving node.
- Time To Live (TTL) - limits the number of nodes an OGM can traverse.
- Host Neighbour Announcement (HNA) count - describes the number of non-batman capable nodes that can be reached through the originator. Informations about each HNA is appended to the OGM-message.
- Gateway Flags - used if the node offers a connection to another network.

Internet Control Message Packet The internet control message packets (ICMPs) are used to support a subset of the features provided by the IP-version of ICMP. As the B.A.T.M.A.N Adv. routing protocol operates at the link layer, nodes in network cannot always be reached by their IP addresses, and thus B.A.T.M.A.N Adv. supports ping requests, replies and failures.

Unicast Packet Unicast packets encapsulates unicast data from the higher layers. A unicast packets contains, in addition to the payload, the destination address and a TTL field.

Fragmented Unicast Packet As B.A.T.M.A.N Adv. encapsulates payload, the length of unicast packets can exceed the MTU of the link layer and packets must thus be fragmented and aggregated at the destination. Fragmented unicast packets carry parts of a payload packet and have a sequence field needed to aggregate the original unicast packet, an originator field to identify the sequence and flags to signal the end of a sequence.

Broadcast packet To support broadcasts to all nodes in the network, and not only the in-range neighbors of a node, broadcast packets are forwarded. To avoid duplicates, broadcast packets are equipped with a sequence number, an originator address, and a TTL.

Visualization packets B.A.T.M.A.N Adv. supports a graphical visualization of the network. To obtain information needed to do this, a server can be started on one ore more nodes in the network. A server announces its presence through visualization packets and each node reports information back to the server.

3.1.2 Node Discovery

As described above, all nodes in a B.A.T.M.A.N Adv. network periodically transmit OGMs to all other nodes in the network. When a node receives an OGM, it performs the following steps:

1. Check if the originator of the OGM is the node itself. In this case, the sender is a direct neighbor and the routing table is updated accordingly.
2. Check if the previous sender is the node itself. If so, the OGM has already been processed by this node and is dropped.
3. The originator of the OGM is determined. If it does not exist in the routing table, it is created.

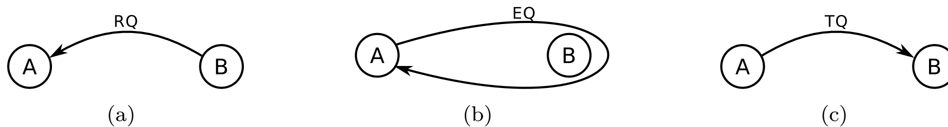
4. The ranking of the originator of the OGM is updated as described in Section 3.1.3.

5. The TQ and TTL fields in the OGM are updated and the OGM is broadcasted again.

Also, checks are made to avoid routing loops and duplicate OGMs.

3.1.3 Link Quality Estimation

To estimate the link quality towards an originator, the Transmit Quality (TQ) in an OGM is changed during its way through the network. The TQ describes the probability of a packet reaching its destination and is calculated as described here. The value is stored in an eight bit value between 0 and 255, which gives higher granularity without increasing the size of the packet.



(a) (b) (c)

Figure 3.2: The three link qualities used in B.A.T.M.A.N Adv. seen from the point of view of Node A. The Receive Quality (RQ) in (a) is the probability of receiving packets from Node B to Node A. The Echo Quality in (b) is the probability of packets from Node A being received by Node B and then by Node A. The Transmit Quality (TQ) is the probability of a packet from Node A being received at Node B.

Receive Quality In the scenario illustrated in Figure 3.2(a), Node B transmits OGMs that are received at Node A, who calculates the Receive Quality (RQ) from Node B. This is done using a sliding window of size N (defaults to 128), where the sequence numbers of the last N OGMs are remembered. The receive quality is then calculated as the percentage of received OGMs.

Echo Quality Whenever Node A broadcasts an OGM, Node B rebroadcasts the OGM, which then reaches Node A again. As with RQ, a sliding window is used to calculate the echo quality (EQ).

Transmit Quality The transmit quality is the probability of a packet sent from Node A to be received correctly at Node B. Since the EQ is the probability of a packet being delivered correctly *both* from Node A to Node B and back from Node B to Node A, the TQ can be derived using RQ and EQ:

$$EQ = RQ \cdot TQ$$
$$TQ = \frac{EQ}{RQ}$$

3.1.4 Propagation of Transmit Quality

To inform other nodes in the network about the TQ towards Node A, the TQ value is added to the OGM. At each node traversed by the OGM, the global TQ towards Node A is updated with the local TQ towards the latest sender of the OGM as illustrated in Figure 3.3.

If OGMs from Node A reaches another node through two or more distinct paths, the receiving node calculates the new global TQ for both paths and chooses the highest result when forwarding the OGM. For every node an OGM traverses, a hop penalty is applied, which favours short paths over longer ones.

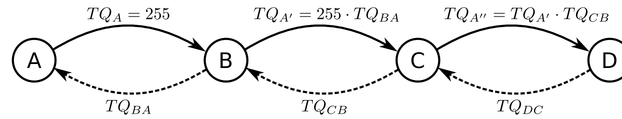


Figure 3.3: The TQ value in the OGM is updated for every node it traverses. Each node has a local TQ to the previous node and scales the global TQ towards Node A with this.

3.1.5 Route Selection

While some other routing protocols calculates the entire network topology when choosing routes, B.A.T.M.A.N Adv. keeps only information about which next-hop is the best towards a node. When a packet is received at the next-hop, this again chooses the best next-hop towards the destination.

A routing scenario is illustrated in Figure 3.4, where Node F transmits a packet to Node A. The only information Node F has about routes to Node A, is two global TQs: One for the path through Node D and another for the path through Node E. In this case the path through Node D is the best, and the packet is transmitted to Node D, who knows only the global TQs for routing through Node B, Node C, and Node E. Whichever node it choses has to make the same choice. In this case, Node C is chosen, who then choses the direct path to Node A.

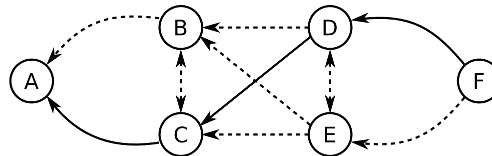


Figure 3.4: When transmitting a packet from Node F to Node A, each node selects the best next-hop based on global TQs for each path (solid lines).

3.1.6 Host Neighbour Announcements

Section 3.1.1 mentions that OGMs includes announcements about neighbors reachable through the originator. As these neighbors knows nothing about the B.A.T.M.A.N Adv. protocol, the connected originator must inform the network about its neighbors, which is done by adding a list of the addresses of the connected neighbors to each OGM transmitted by the originator. Other nodes in the network then keeps a list of host neighbors in the network and their connected originator.

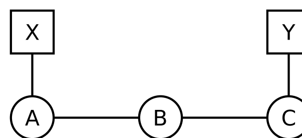


Figure 3.5: Nodes can provide access to hosts that are not directly connected to the mesh network. This is done by adding Host Neighbour Announcements to the tail of the originator messages.

A simple scenario is illustrated in Figure 3.5, where Node A is connected to Neighbour X and Node C is connected to Neighbour Y. When Node A receives a packet from Neighbour X, that is directed at Neighbour Y, it knows that the packet must be transmitted to Node C.

3.2 Coding Opportunity Discovery

To exploit the coding opportunities described in Section 1.3, information about the network topology is needed. Figure 3.6 illustrates a scenario, where a possible coding opportunity exists: Since Node B and Node D have overheard Packet 1 and Packet 2, respectively, the two packets can be combined and forwarded by Node R as one.

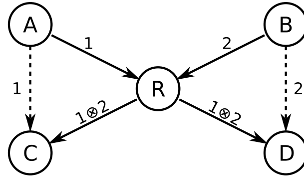


Figure 3.6: To discover coding opportunities, Node R must know that Node C overhears packets from Node A and Node D overhears packets from Node B.

To discover this coding opportunity, the TTL field of OGMs is exploited. The relay node in Figure 3.6 (Node R) discovers the two overhearing links by comparing the TTL of identical OGMs as illustrated in Figure 3.7. The first OGM from Node A is received by Node R, who stores the TTL value. When the OGM is rebroadcasted by Node C, it is received by Node R, who compares the TTL of the OGM with the stored one. If the received TTL is one less than the stored, Node R knows that Node C can hear packets from Node A.

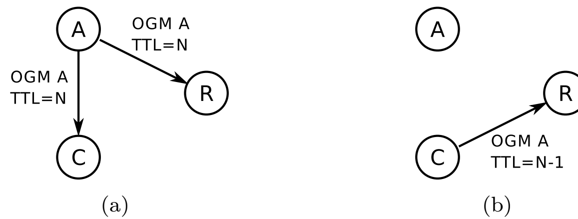


Figure 3.7: To discover a possible coding opportunity, Node R remembers the TTL of the OGM received directly from Node A (a). If the same OGM is received from Node C with TTL decreased by one (b), Node R knows that there is a link from Node A to Node C.

In the scenario illustrated in Figure 3.6, the relay has now discovered that Node C hears packets from Node A and that Node D hears packets from Node B. As packets arrive at Node R, the following steps are taken to find if network coding is possible:

1. When Packet 1 arrives, Node R traverses coding neighbors of Node D, who is the destination of Packet 1. In this case, Node B is the only node.
2. With Node B as the *destination side* coding node, Node R traverses the list of coding neighbors of Node A, who is the source of Packet 1, and finds Node C.
3. Node R searches for packets from Node B to Node C and finds none, as Packet 2 has not yet arrived.
4. Packet 1 is buffered for a configured period. (E.g. 10 milliseconds)
5. When Packet 2 arrives, Node R traverses coding neighbors of Node C and finds Node A.
6. Node R then traverses the coding neighbors of Node B and finds Node D.

7. Node R searches for packets from Node A to Node D and finds Packet 1. (Given that Packet 2 arrives within the configured hold time.)

8. Packet 1 is combined with Packet 2 and transmitted.

This approach is designed for the X-topology described in Section 1.3 but is easily adopted to the Alice-and-Bob topology. This is done by considering nodes as coding neighbors to themselves. This is illustrated in Figure 3.8, where the coding neighbors from Figure 3.6 are joined into two single nodes.

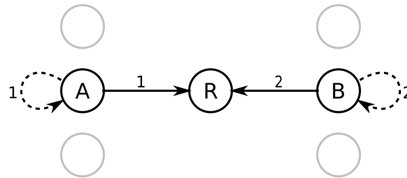


Figure 3.8: To support the Alice-and-Bob topology, neighbour nodes are considered as coding neighbors to themselves.

When following the steps described above for the scenario in Figure 3.8, the two nodes are found as coding neighbors to themselves and a coding opportunity is discovered.

3.3 Receiver Selection

Coded packets are destined for multiple recepients. As explained in Section 1.4, the 802.11 MAC only allows a single destination in unicast packets. It is therefore necessary to adapt a method of addressing multiple destination nodes with a single packet. One option is to broadcast all frames, and add the destination addresses in the packet data. This approach has a number of major drawbacks. As there is no single destination, an RTS/CTS sequence can not be carried out, since there is no node to send the Clear To Send message. This could be an issue in a network with hidden nodes. The lack of a single receiver also means that MAC layer retransmissions are infeasible, since it is not unambiguous who should send the acknowledgement. Finally, broadcast traffic has no automatic rate adaptation and is therefore generally transmitted at a lower speed since the link quality to the receivers are unknown.

A better options is to set the wireless nodes in promiscuous mode, enabling them to receive packets addressed to other nodes. Frames should then be sent with the address of one of the destination nodes in the MAC header, and the remaining addresses in the data field. Compared to broadcast transmissions, this approach has a number of advantages. The destination node can perform the RTS/CTS handshake, and must acknowledge the frame. Any retransmissions due to missing acknowledgements from the addressed node, are also usable for the additional receivers. The rate adaptation algorithm can also be used for the link between the transmitting and receiving node, although the link to the other receivers could be so poor that the selected rate will result in packet loss.

Using broadcast frames or enabling promiscuous mode, both have the drawback that all packets require processing in software. All frames must thus pass through the Linux kernel networking layers, and can not be filtered by neither the network device nor the device driver.

Which node should then be set as the destination in the MAC header? The B.A.T.M.A.N Adv. TQ value, as discussed in Section 3.1.3, can be used as an estimate of the link quality to each receiving node. Always choosing the node with the lowest TQ value would favor reliability for all nodes over total throughput. Contrary, total throughput is prioritized over reliability if the node with the highest link quality is chosen. If the nodes have almost equal TQ values, such hard limits could end up giving preference to a single node, even if the channel conditions are nearly similar. Instead, CATWOMAN uses a random node, but weighted with the TQ values such that

the weaker node will have a higher probability of being in the MAC header. If node A has a TQ of 100 and node B a TQ of 50, node B should be the destination node twice as often as A.

Consider two nodes, A and B, with link error probabilities ε_A and ε_B. The probability of setting A as the MAC header destination is then defined to be:

$$\alpha = \frac{\varepsilon_A}{\varepsilon_A + \varepsilon_B} \tag{3.1}$$

Similarly, the probability of choosing B as the destination is:

$$\beta = \frac{\varepsilon_B}{\varepsilon_A + \varepsilon_B} = 1 - \alpha \tag{3.2}$$

Clearly, $\alpha + \beta$ should be equal to 1, as exactly one of the nodes must be the destination node.

3.4 Coding Decisions

Coding should only be performed if there is an expected gain in overall throughput. This section derives an estimate of the increase in packet loss introduced by the network coding, and uses it to decide on a set of rules for when the relay node should code packets. The decision rules are not used in the CATWOMAN implementation, but should be included in a later version. All transmissions are assumed to have the same packet size, with static link qualities and no rapid changes in error probabilities. All links are configured to the same transmission speed and none of the links are asymmetric.

3.4.1 Alice and Bob

Figure 3.9 shows the Alice-and-Bob topology with link error probabilities ε_{AR} and ε_{BR}. The link error probability is the probability that a single frame transmission fails. The relay node is configured to retransmit packets up to ρ times. For the MadWifi driver used in the test setup, ρ is equal to 11, but other drivers use up to 16 retransmissions.

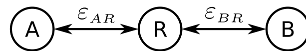


Figure 3.9: Coding node link qualities. The links qualities between Node R and the two other nodes, Node A and Node B, are denoted ε_{AR} and ε_{BR}, respectively. They describe the probability of a failing transmission.

Without Network Coding

Without network coding the packets are sent individually to each node, and the receiving node is always inserted as the MAC destination. Lost packets will therefore be retransmitted up to ρ times by the MAC layer and only depends on the error probability of the individual link. Let $P(E_A)$ be the probability that Alice loses the transmission from R. The packet error probability between R and A is then:

$$P(E_A) = \varepsilon_{AR}^{\rho} \tag{3.3}$$

Similarly, the probability of B failing to receive a packet from R without coding is:

$$P(E_B) = \varepsilon_{BR}^{\rho} \tag{3.4}$$

With Network Coding

When network coding is enabled, the probability of losing a packet depends both on the link to Alice and to Bob. Assume that Bob is set as the MAC layer destination for the transmission. The probability that Alice will lose the packet in exactly one transmission is ε_{AR} multiplied with the probability that Bob received the packet without errors in one transmission, $(1 - \varepsilon_{BR})$. If Bob did not receive the packet, it will be retransmitted and Alice will get a another chance to receive it. Hence, the probability that Alice will lose the packet in i attempts, is $\varepsilon_{AR}^{i} \cdot \varepsilon_{BR}^{i-1} \cdot (1 - \varepsilon_{BR})$. The total probability that Alice will lose the packet when it is transmitted to Bob, can be found by summing these probabilities for up to ρ retransmissions, and adding the probability that both Alice and Bob loses the packet, $(\varepsilon_{AR} \cdot \varepsilon_{BR})^{\rho}$. With α and β being the probabilities that respectively Alice or Bob is the MAC destination, the total probability of Alice losing a packet in the network coded case is:

$$P(E_A)' = \beta \cdot \left(\varepsilon_{AR}^{\rho} \varepsilon_{BR}^{\rho} + \sum_{i=1}^{\rho} (1 - \varepsilon_{BR}) \cdot \varepsilon_{BR}^{i-1} \cdot \varepsilon_{AR}^{i} \right) + \alpha \cdot \varepsilon_{AR}^{\rho} \tag{3.5}$$

The corresponding probability that Bob loses a transmission is then:

$$P(E_B)' = \alpha \cdot \left(\varepsilon_{BR}^{\rho} \varepsilon_{AR}^{\rho} + \sum_{i=1}^{\rho} (1 - \varepsilon_{AR}) \cdot \varepsilon_{AR}^{i-1} \cdot \varepsilon_{BR}^{i} \right) + \beta \cdot \varepsilon_{BR}^{\rho} \tag{3.6}$$

Figure 3.10 shows the packet error probability for Alice as a function of the link quality to Alice and Bob. As expected, the packet error probability increases as the Alice's link quality decreases. However, as shown on the plot, it is also dependent on the link quality to Bob. Figure 3.11 shows the total increase in packet loss for Alice when network coding is enabled. Note that the probability scale is different for the two plots. Since the network coding effectively allows two packets to be sent in one transmission, the packets should be coded if the increase in packet loss is less than 50 %. As seen on the figure this is always the case, and network coding should be used independent of the link qualities in the Alice and Bob scenario.

3.4.2 X Topology

In the X-topology, the decoding node must have overheard the decoding packet for the transmission to be successful. Consider node C in Figure 3.12, with network coding enabled. The node must have received both the transmission from R, which can be addressed to either D or C, and have overheard the transmission from A. The failure probability for node C thus depend on all of $\varepsilon_{AC}, \varepsilon_{AR}, \varepsilon_{RC}$ and ε_{RD}.

Without Network Coding

Without network coding, the failure probability depends solely on the link quality between node R and node C. As with the Alice and Bob topology, the probability of packet error after up to ρ retransmissions is:

$$P(E_C) = \varepsilon_{RC}^{\rho} \tag{3.7}$$

With Network Coding

If network coding is used in the network, node C will be unable to decode the packet if it either loses the packet from R, or fails to overhear the transmission from A. Let the packet loss probability from A to C be $P(E_{AC})'$ and the packet loss probability from R to C be $P(E_{RC})'$. The probability that C will fail to decode the received packet then is 1 minus the probability that both transmissions succeed:

$$P(E_C)' = 1 - (1 - P(E_{AC})') \cdot (1 - P(E_{RC})') \tag{3.8}$$

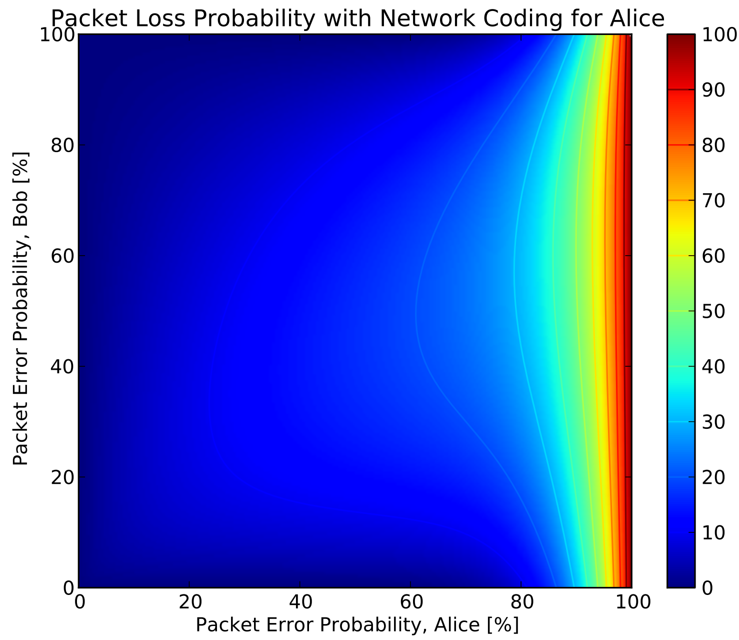


Figure 3.10: Packet loss probability for Alice with network coding. ρ is equal to 11. The colorbar shows the corresponding packet error probability in percent.

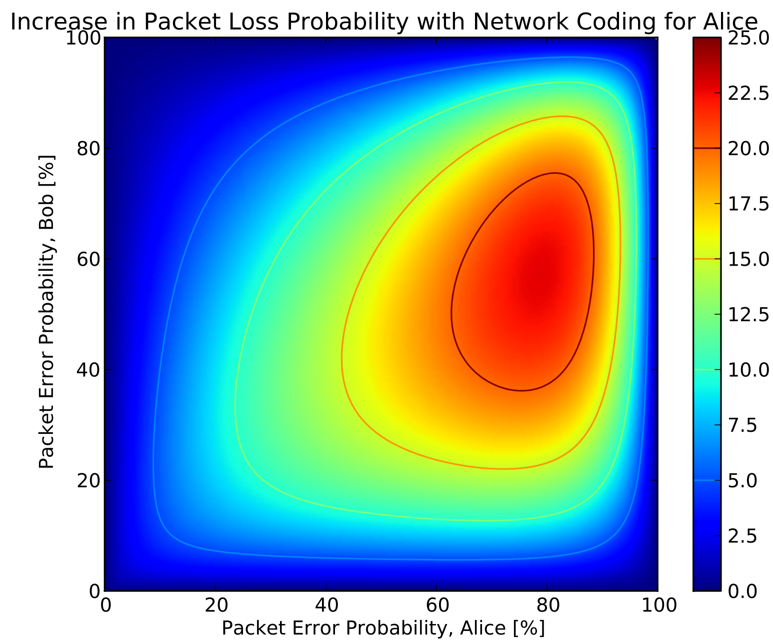


Figure 3.11: Increase in packet loss probability for Alice with network coding enabled compared to uncoded case. ρ is equal to 11. The colorbar shows the corresponding packet error probability in percent.

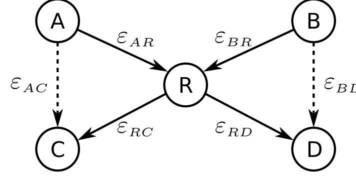


Figure 3.12: Coding node link qualities with X topology. On each side, three link qualities influence the probability of successful coding/decoding sequence: The quality of the overhearing links, the quality of links toward the relay node, and the quality of the links from the relay node.

The transmission from node A to node R will always have R set as the destination node. Node C is thus required to overhear at least one of the transmissions from A to R. The probability that A fails to receive this transmission is:

$$P(E_{AC})' = (\varepsilon_{AR} \cdot \varepsilon_{AC})^{\rho} + \sum_{i=1}^{\rho} (1 - \varepsilon_{AR}) \cdot \varepsilon_{AR}^{i-1} \cdot \varepsilon_{AC}^{i} \tag{3.9}$$

Both C and D candidate to be the destination node for the transmission from R. Let α be the probability that D is set as the MAC destination, and β be the corresponding probability that C is chosen. Following the same argumentation as in the Alice and Bob topology, the probability that C will not receive the transmission from R is:

$$P(E_{RC})' = \alpha \cdot \left((\varepsilon_{RC} \cdot \varepsilon_{RD})^{\rho} + \sum_{i=1}^{\rho} (1 - \varepsilon_{RD}) \cdot \varepsilon_{RD}^{i-1} \cdot \varepsilon_{RC}^{i} \right) + \beta \cdot \varepsilon_{RC}^{\rho} \tag{3.10}$$

As in the Alice and Bob scenario, network coding should only be used if the increase in packet loss is less than 50 %. The relay node can thus calculate an estimate of $P(E_C)'$ from the B.A.T.M.A.N Adv. TQ values, and use this as a coding decision parameter. Figure 3.13 plots the packet loss probability for Charlie as a function of the link qualities from A to Charlie and the Relay. ε_{RC} and ε_{RD} are both set to 0.2. When the link to the relay is perfect ($\varepsilon_{AR} = 0$), Charlie will only have one chance of overhearing the transmission, and the packet loss probability is equal to ε_{AC}. As the link error probability to the relay is increased, Charlie will get more chances of overhearing the transmission and the packet error probability is decreased.

3.5 Coding and Decoding Packets

When a coding opportunity is found and the relay node has decided to code two packets together, the following steps are taken before transmitting the packet:

1. The packet with the longest payload length is selected as destination buffer. This is done to avoid zero padding the smaller packet in order to include the larger.

2. As the MAC layer handles retransmissions for one destination only, the TQ values of the two destinations are weighted with a random number and the lowest is selected as MAC header destination. The other destination, which is more likely to succeed in one of the (re)transmissions, is added to the packet header.

3. The shortest data length is stored in the packet header, as the destination for the shorter packets cannot know the original length.

4. Information needed to select decoding packets and restoring the coded packets is stored in the packet header, as described in detail in Section 3.6.

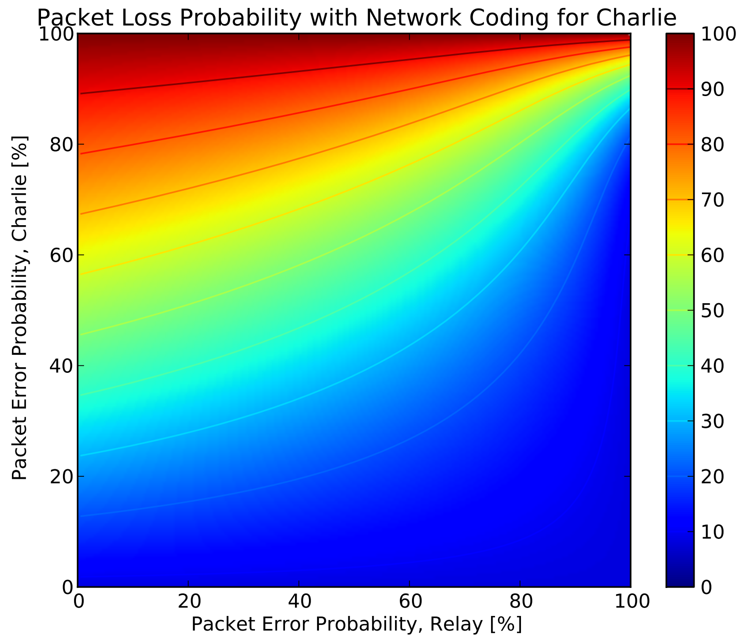


Figure 3.13: Packet loss probability for Charlie in X topology, with network coding enabled. The number of retransmissions is set to 11, and ε_{RC} and ε_{RD} are both 0.2. The colorbar shows the corresponding packet error probability in percent.

5. The source buffer is XORed into the destination buffer. If the two packets have non-equal length, the data not XORed with the shortest packet is left unchanged.

When the coded packet is transmitted and received by one or more destinations, both the destination in the MAC header and the packet header is checked to verify that the packet is destined for the receiving node. When a match is found, the packet is passed for decoding, where the corresponding decoding packet is searched and the packet is decoded. The steps taken to decode a coded packet are:

1. The packet header contains two packet IDs; one for each included packet. Selection is based on the location of the destination address for the receiving node. If the node finds its address in the MAC header, the second packet ID is selected when searching for a decoding packet. If the address is in the packet header, the first packet ID is selected.

2. If no decoding packet is found, the coded packet is dropped. This can happen, if the node failed to overhear the decoding packet or if the decoding packet times out before the coded packet arrives. The latter can be the case, if the network is congested and large driver buffers delay the coded packet.

3. To decode the received packet, the decoding packet is XORed into the coded packet. Only the number of bytes given in the length field of the packet header is XORed and if this is shorter than that of the decoding packet, the receiving node removes the tail of the decoded buffer to restore the original packet.

4. When the packet is decoded, the information from the coding packet header is used to restore the original unicast packet header, and the packet is processed as a normal unicast packet.

3.6 Protocol Packets

As described in Section 3.1, B.A.T.M.A.N Adv. uses several types of packets. To support network coding, a new packet type with the fields needed to decode and restore packets is introduced. An additional field is added to the unicast packet header to identify each packet, which is used when searching for decoding packets.

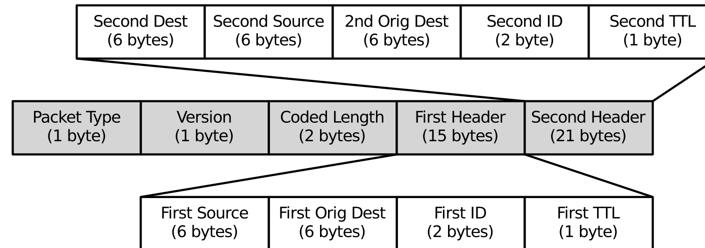


Figure 3.14: The coding packet header, which has the fields needed to restore the two coded packets. This includes the additional destination, the fields from the two unicast packet headers, and the length of the shorter of the two packets.

Figure 3.14 illustrates the coding packet header, which consists of the B.A.T.M.A.N Adv. type and version, the length of the coded data and two groups of fields: One describing the first included packet and one that describes the second.

The description of both included packets consists of the original sources, packet IDs, and TTLs. The sources are, together with the given destinations, needed to identify the decoding packets, since packet IDs are only unique for a specific source-destination-pair. The description of the second included packet also holds the additional destination, which does not fit into the MAC header.

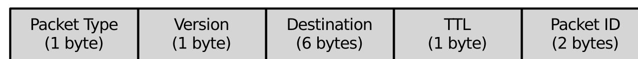


Figure 3.15: The unicast packet header to which an additional field is added (decoding id). The field allows overhearing nodes to identify the packet needed to decode a packet.

All packets transmitted by a node is tagged with a incrementing decoding ID. The ID is a two byte field added to the header as illustrated in Figure 3.15, which is incremented by one for each packet transmitted by the node.

Chapter 4

Implementation

The protocol designed in Chapter 3 is implemented as an integrated part of the B.A.T.M.A.N Adv. kernel module. This chapter describes how B.A.T.M.A.N Adv. is structured when configured for a mesh network and how this is implemented in the kernel module. The network coding protocol is integrated by adding hooks to the existing code and is also described in this chapter. Finally, packet buffers are needed to discover coding opportunities and decode received packets. The structure of these buffers and how they are searched is described in detail.

Network coding is implemented for unicast packets only and does not support fragmented unicast packets. Because of this, the description does not cover how these packet types are handled by B.A.T.M.A.N Adv.

4.1 Structure of B.A.T.M.A.N Adv.

B.A.T.M.A.N Adv. is a pure layer 2 routing protocol and is implemented as a module to the Linux kernel, where it became part of the kernel main tree in March 2011. It is integrated as a layer between the network interface drivers and a virtual network interface, which is used by applications to communicate through the mesh network.

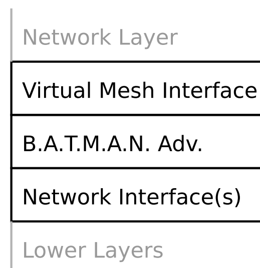


Figure 4.1: The structure of a B.A.T.M.A.N Adv. setup on a single node. Packets are delivered to the module by either the network interface or the virtual mesh interface. The module checks the originator destination of the packet and either forwards the packet to the next-hop or delivers the packets to the upper layer.

As illustrated in Figure 4.1, upper layer interfaces with a virtual interface and needs to know nothing about B.A.T.M.A.N Adv. or the mesh network. This structure makes the entire mesh network act like a virtual switching device that carries packets from one virtual interface to another.

4.1.1 Sending Packets

To keep track of nodes and host neighbors in the network, every B.A.T.M.A.N Adv. node maintains three tables:

Originators - Information about all B.A.T.M.A.N Adv. nodes in the network. The originator table stores information about each node in the mesh network and is used when routing both local and forwarded packets.

Transtable_global - Host neighbors in the network and the connected originator endpoints. The global translation table is searched whenever a local packet is transmitted, to find the originator with a connection to the destination of the packet.

Transtable_local - Host neighbors connected to the node itself. The local translation table is used when broadcasting OGMs, where the entries of the table are appended to the OGM to inform the network of the host neighbors.

The virtual mesh interface is the local entry and exit point to the mesh network. When a packet is transmitted through the interface, it is passed to B.A.T.M.A.N Adv. as a complete MAC frame with the address of the virtual interface as source address. The destination is either a virtual interface on another node, a host neighbour, or a group address (multicast or broadcast). The steps taken before actually transmitting the packet is illustrated in Figure 4.2.

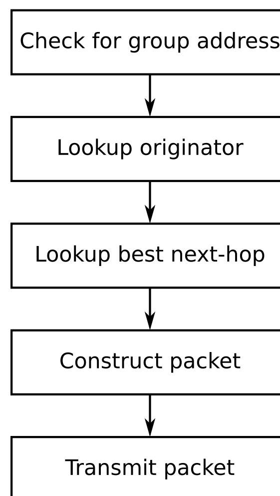


Figure 4.2: The steps taken when transmitting a packet from a local application to the mesh network. The originator providing a connection to the destination is looked up and the best next-hop towards this is found. Then the batman packet is constructed with the original packet and the found information, before it is handed to the device driver for physical transmission.

To route the packet, B.A.T.M.A.N Adv. looks up the destination address in the global translation table and finds the B.A.T.M.A.N Adv. node who is connected to the destination of the packet. The best next-hop towards the found originator is then looked up in the originator table and a B.A.T.M.A.N Adv. packet of the right type is constructed with the original packet and the found information.

Packets that are send through the virtual interface (be that either from an upper layer, an application, or a node on a bridged network) are distinguished from packets that are forwarded in the mesh network. Throughout this chapter, packets send through the virtual interface are denoted *local packets*.

4.1.2 Forwarding Packets

Packets from the mesh network is received by nodes through a configured network device and is processed by B.A.T.M.A.N Adv. if the Ethernet type matches the B.A.T.M.A.N Adv. type (0x4305). If the originator field of the packet does not match the address of the node, the packet is forwarded with the steps illustrated in Figure 4.3.

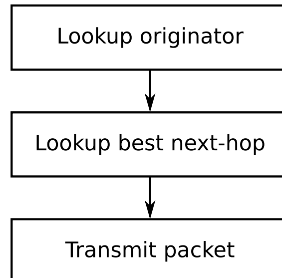


Figure 4.3: The steps taken when forwarding a packet in the mesh network. The originator destination of the packet is looked up and the best next-hop towards this is found. Then the packet is transmitted to the next-hop.

The approach is similar to when transmitting local packets, except the received packet is already constructed with the originator. To forward the packet, the best next-hop towards the originator is looked up and the packet is transmitted.

4.1.3 Receiving Packets

When a batman packet reaches the originator it is directed to, the batman header is stripped from the packet and it is delivered to the virtual interface. The virtual interface may be bridged, in which case the packet can be relayed to a host neighbour as described in Section 3.1.6.

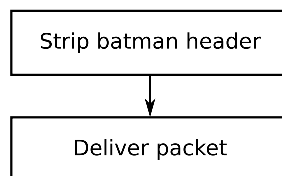


Figure 4.4: The steps taken when receiving a packet in the mesh network. The encapsulating header is stripped and the packet is delivered to the virtual interface.

4.2 B.A.T.M.A.N Adv. Implementation

When loading the B.A.T.M.A.N Adv. kernel module and adding a network interface to its configuration, the module registers itself to receive packets with the B.A.T.M.A.N Adv. Ethernet type and a virtual interface is created. This section described the function involved when sending, receiving, and forwarding packets. The structure of the major files that contains the involved functions is illustrated in Figure 4.5.

The virtual network interface is created and used by functions in *soft-interface.c*. On transmission, packets are passed from *soft-interface.c* to *unicast.c*, where the packet header is constructed and passed to *send.c* for transmission.

In *hard-interface.c*, the network device connected to the mesh network is handled. Here, packets arrive and are passed to functions in *routing.c* for further processing. If the packet is to

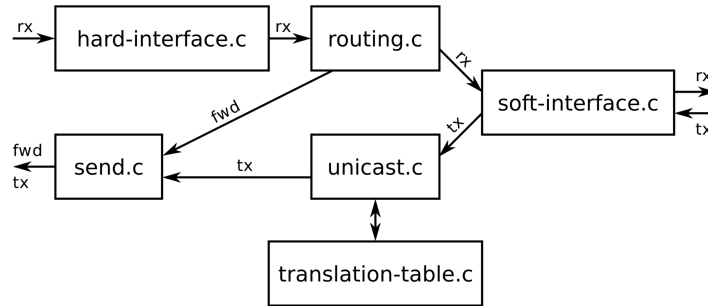


Figure 4.5: Files involved when transmitting, forwarding, and receiving packets. Packets that arrive from the network is delivered by the Linux kernel to a receive function in *hard-interface.c* and packets are transmitted to the network device by a function in *send.c*. Packets from and to the node are processed in *soft-interface.c*.

be forwarded, it is passed to *send.c*. If it is directed to the node itself or connected host neighbors, it is passed to *soft-interface.c*.

The following sections describe these packet flows in detail, which is relevant when implementing and integrating the network coding as described in Section 4.3.

4.2.1 Processing Originator Messages

OGMs are the heart of B.A.T.M.A.N Adv., as these provide the information about the mesh network topology. When an OGM is received by a node, several functions are called to update the topology information used when routing packets. Figure 4.6 illustrates the major functions called upon arrival.

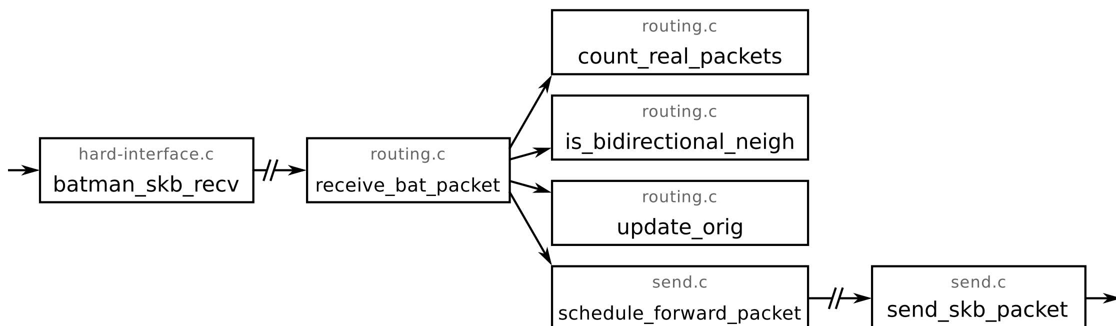


Figure 4.6: Major functions called when receiving an originator message. The primary processing is done in *receive_bat_packet()*, where several functions are called to update the topology information. The OGM is then scheduled for rebroadcasting in *schedule_forward_packet()*, where TQ values are used to select which OGMs should be forwarded.

The function *batman_skb_recv()* is the entry point of all packets with the B.A.T.M.A.N Adv. Ethernet type. The function switches on the batman packet type and passes the incoming packet to the appropriate function according to the type. An OGM is of type 0x01 and before it is passed to *receive_bat_packet()*, it is checked for fragmentation and aggregated if necessary. Fragmentation occurs if the physical layer supports an MTU less than the size of the OGM, which can include multiple HNAs and thus be of arbitrary size.

The main processing is done in *receive_bat_packet()*, where all relevant information about the originator of the received OGM is updated. The first call to *count_real_packets()* updates the

sliding window that keeps track of how many of the latest N OGMs that are received correctly.

The call to *is_bidirectional_neigh()* checks if the received OGM is a echo from a neighbouring node, i.e. the OGM originates from the node itself. If so, the node from which the OGM is received is a bidirectional neighbour and the TQ value towards this neighbour is calculated and updated as described in Section 3.1.

In *update_orig()* the next-hop towards the originator of the OGM is reconsidered. If the node from which the OGM is received already is the best next-hop, nothing further is done. Otherwise, the new TQ value is compared to the current best next-hop, which is replaced if the TQ is better.

Finally, the received OGM is scheduled for rebroadcast. The function *schedule_forward_packet()* queues the OGM and at a given interval, the queued OGMs are traversed. If the queue holds multiple OGMs from the same originator, the one with the best TQ value is broadcasted.

4.2.2 Transmitting Packets

When packets are transmitted by the upper layer, the Linux kernel delivers the packet to a transmit function, which was registered during initialization. In B.A.T.M.A.N Adv. this is *interface_tx()*. The function checks the source address of the packet (which is passed as a complete MAC frame), and adds it to the local translation table if not found. As broadcast packets are handled differently, the destination address is checked for group addresses and processed by the appropriate functions if this is the case. If the destination is a unicast address, the packet is passed to *unicast_send_skb()*.

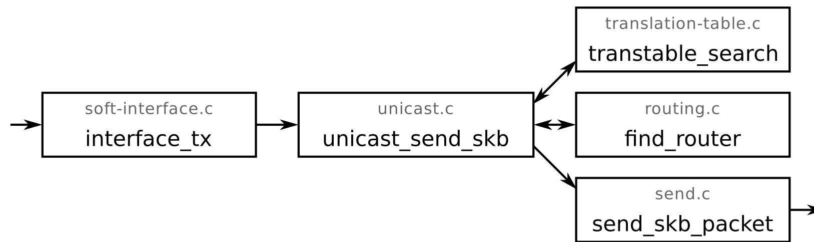


Figure 4.7: Major functions called when transmitting a packet. Packets arrives from the upper layer to *interface_tx()* and are transmitted, after processing, in *send_skb_packet()*.

As illustrated in Figure 4.7, *unicast_send_skb()* searches the global translation table for the originator, with a connection to the destination of the packet. Then, the best next-hop is looked up by calling *find_router()* and unicast packet header is constructed. This is passed along with the address of the next-hop to *send_skb_packet()*, where the new MAC header is constructed and the packet is passed to the network device driver.

4.2.3 Receiving and Forwarding Packets

Unicast packets are received by the same function as OGMs, *batman_skb_recv()*, where the batman packet type is used to determine which function should process the packet.

In *recv_unicast_packet()* the originator address in the batman header is checked. If the packet is not directed to the receiving node, it is passed to *route_unicast_packet()*, where the best next-hop towards the originator is looked up and the packet is transmitted by *send_skb_packet()*. If the originator address is that of the node itself, the packet is passed to *interface_rx()*, where the batman header is stripped before the packet is handed to the upper layer.

4.3 CATWOMAN Implementation

The implementation of network coding in B.A.T.M.A.N Adv. aims at altering the existing code as little as possible. To achieve this, hooks are added to the relevant functions so that the coding and decoding functions can be kept in separate files, which is illustrated in Figure 4.9.

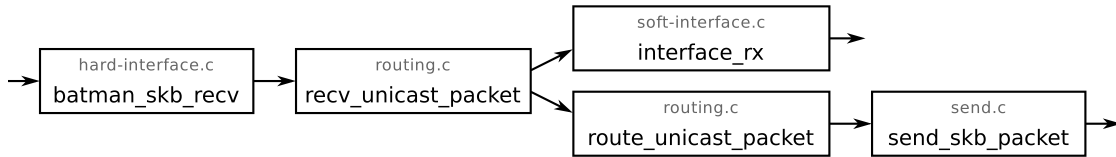


Figure 4.8: Major functions called when a packet is received for either forwarding or delivering. Packets arrive to *batman_skb_recv()* and are, after processing, either delivered to the virtual interface by *interface_rx()* or transmitted in *send_skb_packet()*.

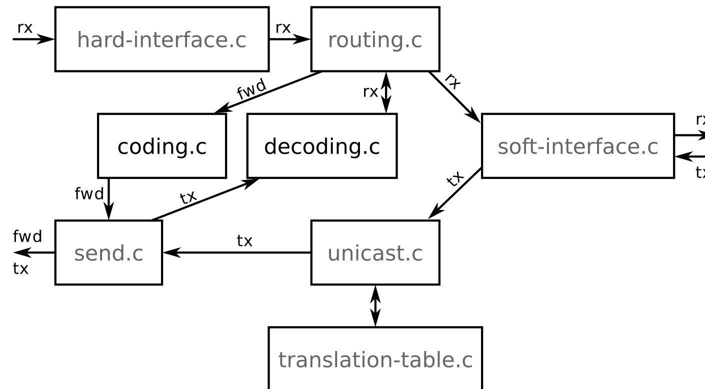


Figure 4.9: The files where network coding is hooked into the existing code. When packets are transmitted, they are stored in *decoding.c* for use in later decoding. When packets are forwarded, they are processed in *coding.c* to find coding opportunities.

The discovery of possible coding neighbours is implemented as a hook in *routing.c*, so that whenever an OGM is received, it is checked according to the description in Section 3.2. To be able to decode packets, transmitted local packets are added to a decoding pool by a hook in *send.c*. Another hook in *routing.c* adds packets that are overheard from other nodes to the same decoding packet pool. When a packet is to be forwarded, it is processed in *coding.c*, where a coding packet pool is searched for coding opportunities.

4.3.1 Discovering Coding Neighbours

For each entry in the originator table, two lists of coding neighbours are maintained: Ingoing coding neighbours and outgoing coding neighbours. The former is a list of coding neighbours that the originator can hear and the latter is a list of coding neighbours that can hear the originator. Having two lists for each node makes searching simpler, which is described in Section 4.4.

When an OGM is received, it is parsed by *receive_bat_packet()*. To update the lists of coding neighbours, a call to *is_coding_neighbour()* is added as illustrated in Figure 4.10.

The existing implementation of B.A.T.M.A.N Adv. keeps the needed information to check if an originator is a coding neighbour to the originator of the received OGM, which is utilized as described in Pseudo Code 1. If the function returns true, the two nodes are added to each others lists of coding neighbours.

The above description of how coding neighbours are discovered only covers the X topology. To support the Alice-and-Bob topology, another check is carried out in *receive_bat_packet()*: When the OGM is received directly from the originator itself, the node itself is added to its list of coding neighbours. In practice, this is equivalent to having two coding neighbours with a 100% probability of overhearing each other packets.

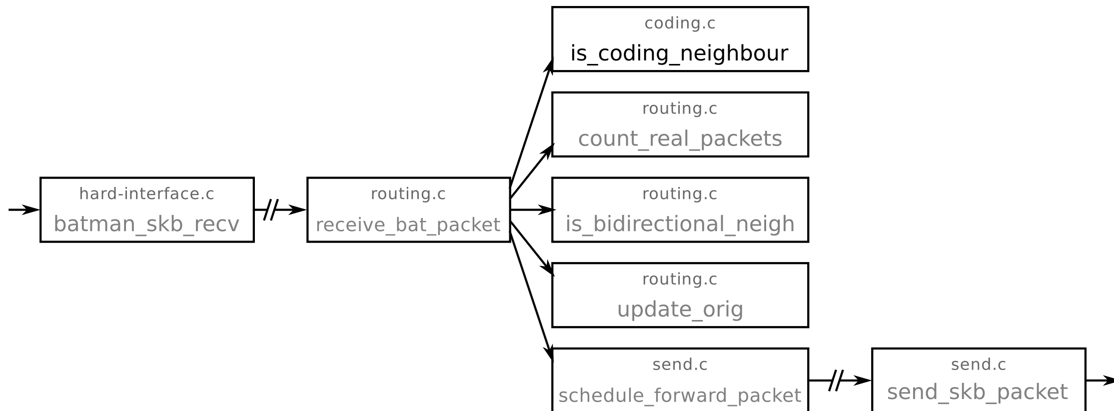


Figure 4.10: The call to *is_coding_neighbour()* is added to *receive_bat_packet()* to update the list of coding neighbours for the originators of each received OGM.

Pseudo Code 1 All received OGMs are checked in *is_coding_neighbour()* to find coding neighbours. The function checks the conditions described in Section 3.2 and returns true (1) if all conditions are met.

```
1:  procedure CheckNeighbour(ogm, neighbour)
2:      ▷ Get the originator of the OGM
3:      orig ← OgmToOrig(ogm)

4:      ▷ Check that the OGM directly from this originator has been received.
5:      if LastSeqNo(orig) ≠ SeqNo(ogm) then
6:          return 0
7:      end if

8:      ▷ Check that this OGM has traversed one node more than the previous.
9:      if LastTTL(orig) ≠ TTL(ogm)+1 then
10:         return 0
11:     end if

12:     ▷ Check that this OGM has traversed only one node.
13:     if orig ≠ PrevSender(ogm) then
14:         return 0
15:     end if

16:     ▷ A coding opportunity is found.
17:     return 1
18: end procedure
```

4.3.2 Coding Packets

Whenever a packet is forwarded, it must be checked if that packet can be coded with another before transmission. To do this, a hook is added to *route_unicast_packet()* in *send.c*. Figure 4.11 illsutrates how the function calls *send_coding_packet()*, which searches the pool of coding packets and calls *code_packets()* if a coding opportunity is found. The procedure used when searching is described in detail in Section 4.4. If no opportunity is found, the packet is added to the pool of coding packets, so that it is included in the next search for a coding opportunity. The pool is traversed periodically and packets that has been hold back for a given time (defaults to 10 milliseconds), are removed from the pool and transmitted.

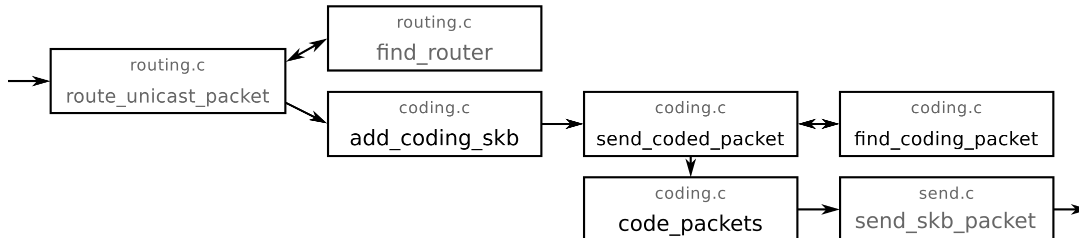


Figure 4.11: When packets are forwarded, a pool of coding packets is searched to find a coding opportunity. If a packet is found, the two packets are coded together and transmitted. Otherwise, the packet is added to the pool for the next search(es).

The procedure used when coding two packets together is listed in Pseudo Code 2 and consists of four blocks:

1. Select the largest packet as destination buffer and the smallest as source buffer.
2. Code the two packets together into the destination buffer.
3. Select one destination address for the MAC header and one for the batman header.
4. Create the new packet header and transmit the packet.

The largest packet is used as destination buffer to avoid zero padding the smallest packet. The length of data that is coded together is written in the batman header and is used when decoding the packets. The two packets are coding in a simple for-loop, where each byte in the source packet is XORed with the corresponding byte in the destination packet.

As described in Section 3.3, only one destination can be placed in the MAC header and the other thus have to be stored in the batman header. The batman header is constructed so that each of the receiving nodes can identify the correct packet by the whether their address is placed in the MAC header or the batman header.

The MAC header is constructed in *send_skb_packet()*, so the address for the MAC header is passed to this function.

4.3.3 Decoding Packets

When a coded packet is received, the packet needed to decode is searched in a pool of decoding packets. As illustrated in Figure 4.12, packets are added to the pool whenever a packet is overheard or a local packet is transmitted. At a fixed interval, the pool is traversed and timed out packets are purged.

To decode the packet, the decoding packets are searched and, if a match is found, used to decode the packet as illustrated in Figure 4.13. A coded packet is passed to *recv_coded_packet()*, where *find_decoded_packet()* is called to search the decoding pool as described in Section 4.4. If no decoding packet is found, the coded packet is dropped.

Pseudo Code 2 Code two packets together and transmit the result. The procedure selects a packet as destination buffer, code the two packets, creates a header for the coded packet, and transmits the packet.

```
 1: procedure CodePackets(packet_1, packet_2)
 2:     ▷ Select the largest packet as destination buffer.
 3:     if Length(packet_1) ≥ Length(packet_2) then
 4:         packet_to ← packet_1
 5:         packet_from ← packet_2
 6:     else
 7:         packet_to ← packet_2
 8:         packet_from ← packet_1
 9:     end if

10:     ▷ Code packet_from into packet_to
11:     length ← Length(packet_from)
12:     packet_to ← XorPackets(packet_to, packet_from, length)

13:     ▷ Lowest Random weighted TQ gets into the MAC header.
14:     tq_1 ← TQ(packet_1),  tq_2 ← TQ(packet_2)
15:     if RandWeight(tq_1) ≤ RandWeight(tq_2) then
16:         mac ← Dest(packet_1)
17:         first ← packet_1
18:         second ← packet_2
19:     else
20:         mac ← Dest(packet_2)
21:         first ← packet_2
22:         second ← packet_1
23:     end if

24:     ▷ Construct batman packet and transmit it with mac as destination.
25:     packet_coded ← AddHeader(packet_to, first, second)
26:     SendPacket(packet_coded, mac)
27: end procedure
```

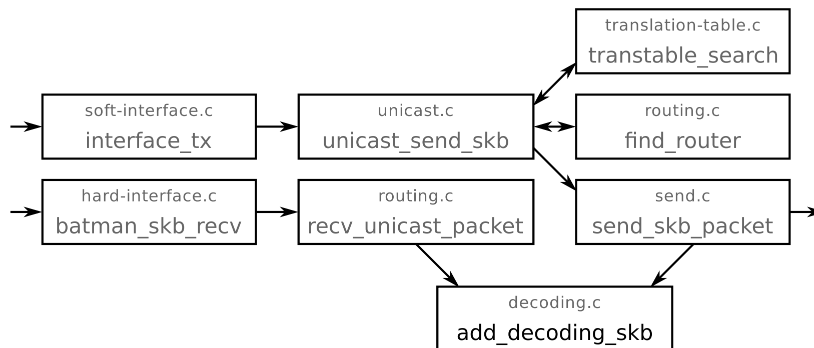


Figure 4.12: Packets that are overheard but not directed to the node and packets transmitted by the node are added to a pool of decoding packets, which is searched when a coded packet is received.

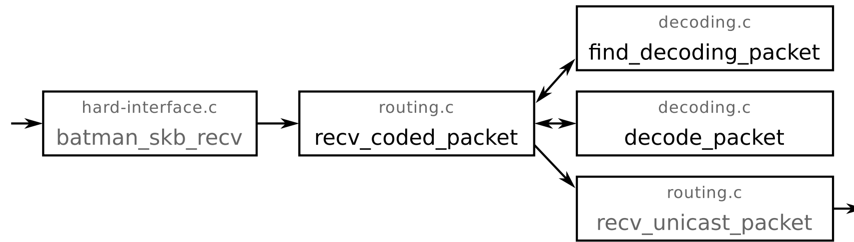


Figure 4.13: To decode a received packet, the matching packet is found in the decoding pool and *decode_packet()* is called. The decoded packet is then passed to *recv_unicast_packet()* as if it where received as a non-coded unicast packet.

When a matching decoding packet is found, *decode_packet()* is called. The procedure used to decode the packet is listed in Pseudo Code 3 and consists of three blocks:

1. The data is decoded by XORing the found decoding packet into the coded packet.
2. The length of the decoded packet is decreased if the smallest of the two coded packets is to the node.
3. The original unicast packet is reconstructed with the correct header.

The decoding in the first block is done in the same way as when packets are coded together: Each byte in the decoding packet is XORed into the coded packet. When the packet has been decoded, the function checks if the decoded data should be resized. This is done by comparing the length of the decoding packet with the length of the coded data; if the decoding packet is longer, the decoded packet is the smaller of the two combined packets and the data should be resized. In the third block, the correct information for the reconstructed batman unicast header is found by checking the location of the MAC address for the receiving node. If the address of the receiving node is located in the coded packet header, the destination address in the MAC header is changed so that the upper layer does not drop the packet.

4.4 Data Structures

The process of routing packets must be fast and non-complex, as the time required to forward one packet cannot exceed the time it takes for a packet to arrive. This requirement is inherited by the process of finding, coding, and decoding packets, so the data structures and searching procedures are thus a crucial part of the implementation.

This section describes how lists of packets and the information about coding neighbors are structured and used when searching for coding opportunities and decoding packets.

4.4.1 Coding Neighbours

Figure 4.14 illustrates how a relay node discovers two coding neighbors as described in Section 3.2. When a coding neighbour is discovered, two lists are updated at Node R:

- The list of ingoing coding neighbors for the node who just forwarded the OGM.
- The list of outgoing coding neighbors for the originator of the OGM.

The illustrated X topology requires overhearing of packets, and thus the Alice-and-Bob topology is preferred when coding packets. This priority is implemented by adding nodes to the head of their own lists of coding neighbors and always traversing these lists from the head.

References to the two lists are added to the originator structure, which is used by the existing B.A.T.M.A.N Adv. implementation to hold information about each originator in the mesh

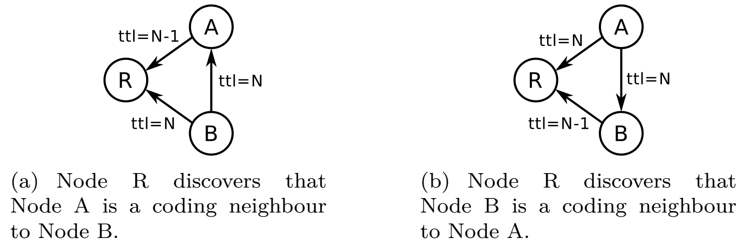
Pseudo Code 3 Decode a coded packet with the decoding packet. The procedure decodes the coded data, restores the original header, and returns the decoded packet.

```
1:  procedure DecodePacket(packet_coded, packet_decoding)
2:      ▷ Decode packet
3:      length ← CodedLength(packet_coded)
4:      packet_out ← XorPackets(packet_coded, packet_decoding, length)

5:      ▷ Resize packet_out if the shortest one if for us.
6:      if Length(packet_decoding) < length then
7:          packet_out ← Resize(packet_out, length)
8:      end if

9:      ▷ Restore batman unicast header from the correct part of the coded header.
10:     mac ← MyMac()
11:     if MacDest(packet_coded) = mac then
12:         packet_unicast ← UnicastFromFirst(packet_decoded, packet_coded)
13:     else
14:         packet_unicast ← UnicastFromSecond(packet_decoded, packet_coded)
15:         ▷ Set the MAC destination so upper layers will accept the packet.
16:         ResetMacDest(packet_out, mac)
17:     end if

18:     return packet_out
19: end procedure
```



(a) Node R discovers that Node A is a coding neighbour to Node B.

(b) Node R discovers that Node B is a coding neighbour to Node A.

Figure 4.14: In (a) the relay adds Node A to the list of outgoing coding neighbors of Node B and adds Node B to the list of ingoing coding neighbors of Node A, In (b), Node A is added as ingoing to Node B and Node B is added as outgoing to Node A.

network. The originator structures are stored in a hash table as illustrated in Figure 4.15, where MAC addresses of the originators are used as keys. In Section 4.4.3 a description is given of how the two lists are used when searching for coding opportunities.

4.4.2 Packet Pools

Overheard and transmitted packets are stored for use when decoding, and forwarding packets are stored for later coding. The packets are stored in a structure where packets from a specific source-destination pair can be located quickly. The structure uses the same hash table implementation as B.A.T.M.A.N Adv. uses to store originator structures, but the keys are made of a source-destination pairs, where each byte of the reversed destination address is XORed with the corresponding byte from the source address. The generation of a source-destination key is listed in Pseudo Code 4. The element identified by a source-destination pair is denoted a *coding path*.

The generated coding path keys are not guaranteed to be unique, so multiple coding paths can be stored in one bin. Each coding path is a structure with the two addresses identifying the

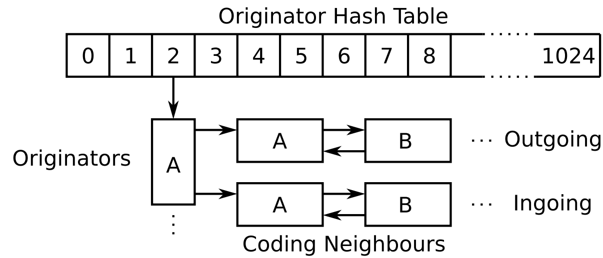


Figure 4.15: Structure of the list of coding neighbors. In this case, Node A can hear packets from itself and from Node B. Also, Node B can hear Node A.

Pseudo Code 4 The procedure used to generate keys for the packet pool hash tables. Each byte from the reversed destination address is XORed with the corresponding byte from the source address.

1: **procedure** SOURCEDESTINATIONKEY(*source*, *destination*)
2: ▷ XOR each byte of the addresses together
3: $l \leftarrow$ ETHADDRLENGTH()
4: **for** $i \leftarrow 0, l$ **do**
5: $key_i \leftarrow source_i \otimes destination_{l-i-1}$
6: **end for**

7: **return** *key*
8: **end procedure**

path and a reference to a list of packets. The lists is a doubly linked list with a tail pointer, so that packets can be added and removed in a FIFO manner. An example of a coding path with a list of packets is illustrated in Figure 4.16.

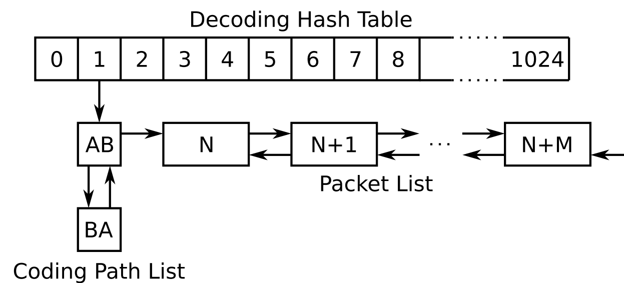


Figure 4.16: Structure of the two pools of packets: Coding packets and decoding packets. The keys used to index the hash table are a pair of source and destination addresses XORed together. Each bin in the table contains a list of *coding paths*, where the addresses of the next-hop and prev-hop are stored for uniqueness. Each coding path points at a list of packets stored from this path.

4.4.3 Searching for Coding Opportunities

The procedure used to search for coding opportunities is listed in Pseudo Code 5. The procedure is run for every unicast packet that is routed by a node. It looks up all nodes that the destination node of the packet can hear and for each of these, it looks up all nodes that can hear the source node of the packet. If a packet is found for one of these coding paths, a coding opportunity is found and the search is stopped.

Pseudo Code 5 The procedure used to search for a coding opportunity. For each combination of ingoing and outgoing coding neighbors, the packet pools are searched for coding opportunities.

```
 1: procedure FindCodingPacket(packet)
 2:     dst ← Dst(packet)
 3:     src ← Src(packet)

 4:     ▷ Search ingoing coding neighbors of the destination.
 5:     for dst_ingoing ← IngoingNeighbours(dst) do
 6:         ▷ Search outgoing coding neighbors of the source.
 7:         for src_outgoing ← OutgoingNeighbours(src) do
 8:             ▷ Get the coding path of this pair of coding neighbors.
 9:             key ← SourceDestinationKey(dst_ingoing, src_outgoing)
10:             path ← GetCodingPath(key)

11:             ▷ If a packet is stored for this coding path, an opportunity is found.
12:             if HasPacket(path) then
13:                 return PopPacket(path)
14:             end if
15:         end for
16:     end for

17:     ▷ No coding opportunity found for packet.
18:     return 0
19: end procedure
```

The complexity of the searching procedure is rather high, but is acceptable as the number of coding neighbors is limited to the amount of nodes in range. The implementation could be extended, so that the quality of the overhearing link must meet a given criteria, before a node is added to a list of coding neighbors. This would decrease the number of nodes in the two lists of coding neighbors and thus the run time of the search.

4.5 Buffering

The packet pools described in Section 4.4 introduces an additional worst-case delay to packets being routed. The implementation has a timed worker function, that is called every 10 milliseconds. The function traverses the stored packets in the coding packet pool and transmit timed out packets uncoded. The timeout can be configured and is denoted the *hold time*.

The worst case scenario is illustrated in Figure 4.17, where the second packet arrives at the relay node shortly after the first packet was forwarded. In this case, both packets are kept in the packet pool and are not forwarded until the timeout is reached and the worker function is called.

Figure 4.17: The worst case scenario with regard to delay. In (a), Packet 1 is kept in the coding packet pool before being forwarded to Node B. In (b), Packet 2 arrives at the relay node shortly after Packet 1 was transmitted and is also kept in the coding packet pool before being forwarded.

In the best case scenario, network coding decreases the delay by forwarding Packet 2 in the same slot as Packet 1 and thereby virtually advancing Packet 2 in the transmit queue. The

best case is thus when Packet 2 arrives at the relay node shortly after Packet 1 as illustrated in Figure 4.18.

Figure 4.18: The best case scenario with regard to delay. In (a), Packet 2 arrives at the relay node just after Packet 1. In (b), the relay can transmit both packets in one slot.

The increased delay is thus expected when the arrival rate at the relay is below the point, where packets arrive within the configured hold time. The decreased delays are expected when the arrival rate is above that point.

4.5.1 Network Device Transmit Queue

With respect to throughput, network coding only gives an advantage when the network is congested. If there is free capacity in the network, packets may as well be sent individually instead of being coded together. In a congested network, packets are queued in the device transmit queue, which can be used as a coding packet pool. This approach would enable network coding only when the network is congested and avoid the delay introduced by holding packets as described above. While implementing the network coding, the structure of queues in the network stack turned out to be a hurdle as described below, why packets are stored in the custom packet pool instead.

Figure 4.19 shows a simplified illustration of the networking stack. When a user space application sends a chunk of data, it passes through the network protocol layers before being copied to the transmit queue of the outgoing interface. This interface delivers one packet at a time to the network interface driver, which instructs the network device to transmit the packet. A simple solution would therefore be to search for coding opportunities directly in the device transmit queue; when transmitting the packet in the front of the queue, traverse the queue and code the packet with the first coding opportunity found.

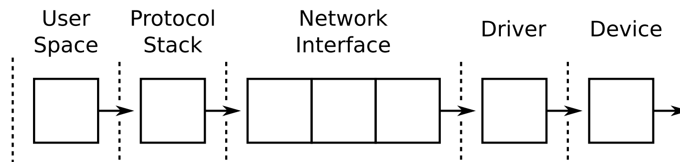


Figure 4.19: Simplified transmit queue. An application transmits data, which is processed by the transport and network layers, queued in the network device interface and transmitted by the driver.

Unfortunately, this approach is not possible due to the actual buffer organization of the network stack. Figure 4.20 depicts a more realistic illustration of the buffers in the networking layers. From the user space application, the packet is copied through the network protocol layers and routed to a network interface as in the simplified case. In the router configuration used in the test setup (described in Chapter 5), the packet is routed through a series of virtual interfaces. The packet is first delivered to a Linux Ethernet Bridge, `br0`, that joins the wired `eth0` interface and the `bat0` mesh interface with a single IP address and Ethernet collision domain. The `bat0` interface is itself a virtual interface that forwards packets to the `mesh0` interface, provided by the MadWifi wireless driver. The driver allows multiple Virtual Access Point (VAP) interfaces to be created, e.g. to create several networks with different BSSIDs or access restrictions. The `mesh0` thus forwards packets to the `wifi0`, which is the actual physical interface.

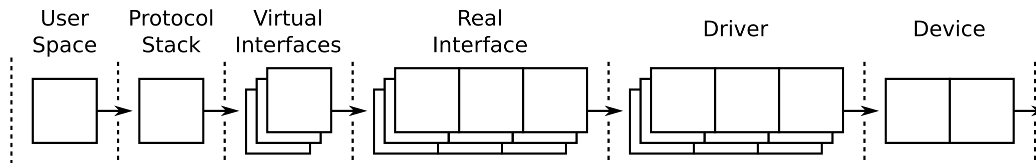


Figure 4.20: Real transmit queue. When a packet is delivered to the network device interface, it is queued in multiple virtual interface, the network interface, the driver, and the hardware device.

Only non-virtual interfaces, i.e. `wifi0`, have actual transmit queues. By default, the queue size on MadWifi devices is configured to 195 packets. These packets are divided over multiple prioritized queues, that are governed by a Queueing Discipline. (By default, there is one queue for each IP Type of Service.) The Queueing Discipline selects the first packet to be transmitted, and passes this on to the device driver by calling the associated `hard_start_xmit()` function. Most network device drivers, including the MadWifi driver used in the test environment, maintain a Direct Memory Access (DMA) ring buffer in the driver. The MadWifi driver has a prioritized 200 packet ring buffer. The copying from the network device to the ring buffer is much faster than the actual frame transmission, which result in the packets queueing up in the device buffer rather than the network transmission queue. The ring buffer is strictly internal to the device driver, and can not be accessed from the network device. Finally, a number of buffer elements may be available in the device itself.

The network coding must be carried out in the driver for the `bat0` interface, which only has a reference to the associated `mesh0`. Due to the organization of the VAP devices, it is possible to obtain a pointer to the `wifi0` interface by searching the Linux device tree, but this is a very non-portable solution and will most likely be infeasible with other device drivers. Furthermore, the driver ring buffer is still inaccessible, which means that packets will not be enqueued in the transmission queue unless the 200 packet ring buffer fill up. The size of the driver ring-buffer is non-configurable for most network drivers.

Using very large ring buffers in the device driver is associated with a number of other drawbacks, such as very long delays when the buffer is full, and failure to provide TCP congestion information in a timely manner. The issue is not confined to the MadWifi driver, and both the carl9170usb and iwl3945 drivers used in the development of CATWOMAN suffers from the same issue. Jim Gettys has coined the term *Bufferbloat* and initiated the `www.bufferbloat.net` site to create awareness of the problem, and to advocate for improved buffer management to mitigate the problem. One proposals is to use dynamic scaling of the queue size, in order to achieve a target delay [LLM11].

Chapter 5

Test Environment

To evaluate the performance of the network coding protocol, tests are carried out in a real wireless network. Tests are conducted for both UDP and TCP as described in this chapter. The description includes the testbed and the tools that are used to carry out the tests.

5.1 Experimental Setup

The test platform consists of five OM1P routers from Open-Mesh[Ope11], and four laptops for traffic generation. The OM1P router is based on an Atheros AR2315 Wireless System-on-a-Chip, with a 200 MHz MIPS processor and 32 MB of RAM. All the routers are configured with the open source firmware OpenWRT version 10.03.1-RC5 with the 2.6.30.10 Linux kernel. The B.A.T.M.A.N Adv. is compatibility version 12 and is patched with the CATWOMAN implementation described in Chapter 4. The used OpenWRT source code, a compiled image for the OM1P, the B.A.T.M.A.N Adv. source, and the CATWOMAN source are available on the CD attached in Appendix B.

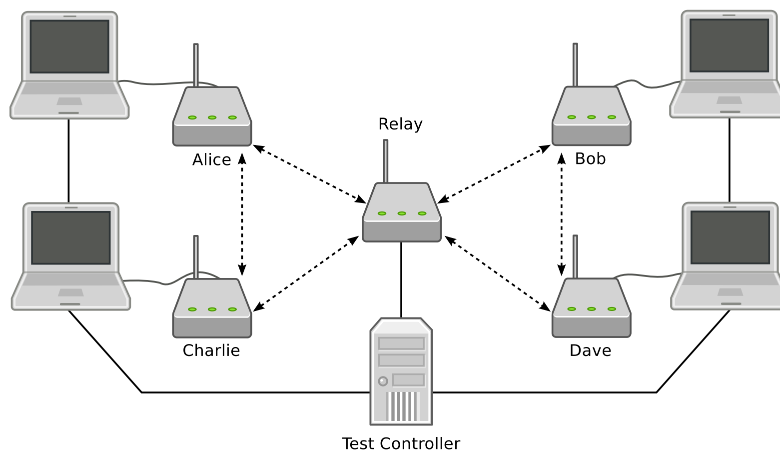


Figure 5.1: Nodes and Computers in the testbed. Five nodes form the X topology and four computers are connected to the outer nodes. The computers and the relay node are connected to a test controller, where the tests are coordinated.

Figure 5.1 shows the basic test network topology. The OM1P routers are referred to as *nodes* and the laptops as *slaves*. All nodes except the Relay are connected to laptops for traffic

generation and throughput monitoring, and each laptop is connected to a central test-server for test coordination and data collection.

Since the test network contains a number of hidden node scenarios (Alice and Charlie should not overhear transmissions from Bob and Dave, and opposite), all nodes are configured to use the RTS/CTS mechanism as specified in Section 1.4. Most wireless devices implement some kind of rate adaptation algorithm to automatically select an appropriate data rate based on channel conditions. This is especially important for mobile nodes that needs to adapt to fast changing channel conditions. In order to avoid the tests being biased due to varying rates, rate adaptation is disabled, and all nodes are set to a fixed speed of 11 Mbit/s. The nodes are all configured to transmit with 10 dBm.

The nodes are placed in a university building with a large open space atrium. Every node is placed to have a direct line of sight to the relay node, as shown in Figure 5.2. As the channel used for testing is shared with other networks in the building, all tests are carried out at night, to have a minimum of interfering traffic from surrounding wireless devices. The channel has a constant minimum load of approximately 200 kbit/s due to beacons from wireless access points, but this should have a minimal influence when the network is congested. The maximum rate between two nodes has been measured to 5.4 Mbit/s, and should be considered the channel maximum when using 802.11b devices.

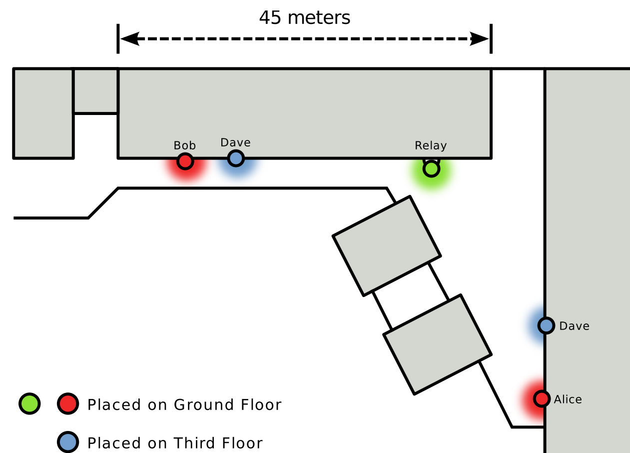


Figure 5.2: Node placements in the test setup. Alice, Bob and the Relay node are placed on the ground floor, while Charlie and Dave are placed on the third floor. All nodes are positioned to have a clear line of sight to the Relay node, but far enough to avoid direct routes from one side to another.

5.2 UDP Performance Tests

The test execution is handled from a central server with the `catbench.py` tool, which creates a test report that can subsequently be plotted with `catplot.py`. The source code for both programs is available on the attached CD. All tests are performed with and without CATWOMAN enabled, by sweeping multiple times over a set of offered loads with a fixed step size. Each test is run for 30 seconds, where traffic is generated with the `iperf` utility. `iperf` transmits equally spaced 1498 byte[1] UDP packets with increasing target data rates. For each target data rate, `catbench.py` measures the following properties for the slaves:

- Throughput is measured using `iperf` in UDP mode. The throughput is calculated from the number of packets received and the transmission speed. If e.g. 50% of the packets are

[1] 1470 bytes of random data, 8 byte UDP header, 20 byte IP header

received by the destination slave when sending with a target rate of 1 Mbit/s, the throughput is 500 kbit/s.

- Average delay, measured by sending ICMP echo requests, in intervals of one second, to the receiving node using the `ping` utility and reading the average reply time.

The nodes are monitored from the test server via the slaves. For the nodes, the following properties are measured:

- The number of forwarded and coded packets are exported via `debugfs`, and measured by `catbench.py`. This number is only useful for the tests where CATWOMAN is enabled, since the node should forward all packets if it is disabled.
- The number of packets in the hold queue. The size of the queue is sampled every second, and the average size is calculated after each test.
- CPU load of the node, measured by reading the number of scheduler ticks allocated for the idle process from `/proc/stat`.
- The number of transmitted and received frames, and the number of RTS and data frame retransmissions.
- The number of decoding failures. Ideally, this is zero for Alice and Bob since the nodes always overhears their own transmissions.

Source Code 5.1 gives an example of a configuration file for `catbench.py` in the Alice and Bob topology. The slaves are configured with their IP addresses on the monitoring network (172.26.72/24) and on the test network (10.10.0/24). Each slave is then associated with a node for routing on the test network. Finally, a bidirectional flow through the relay node is created between Alice and Bob and the next-hop for each flow is configured. The next-hop is checked every second to ensure the correctness of the measurements. A test can now be run with:

```
% ./catbench.py --output=test.pickle --duration=30 --min=100 --max=3000 --step=100
```

```python
1  # Create Alice
2  alice = Slave("alice")
3  alice.set_ip("172.26.72.40", "10.10.0.9")
4  alice.set_node_ip("10.10.0.100", "00:72:cf:28:19:1a")
5  # Create Bob
6  bob = Slave("bob")
7  bob.set_ip("172.26.72.104", "10.10.0.8")
8  bob.set_node_ip("10.10.0.102", "00:72:cf:28:19:16")
9  # Create Relay
10 relay = Node("relay")
11 relay.set_ip("cwn1.personal.es.aau.dk", "00:72:cf:28:19:da")
12 # Connect flows
13 alice.set_flow(bob)
14 bob.set_flow(alice)
15 # Setup route checks
16 alice.add_route(bob.node, relay)
17 bob.add_route(alice.node, relay)
```

Python Source Code 5.1: `catbench.py` configuration for Alice-and-Bob topology. Each slave is created and their connected nodes are configured.

5.3 TCP Performance Tests

The performance of TCP flows on a CATWOMAN enabled network is briefly tested to investigate the effects on the TCP congestion avoidance algorithm. The tests are preliminary and the subject must be tested further to expose any unknown relations between network coding and the TCP congestion avoidance algorithms. They are carried out with the `catcp.py` tool and the results are parsed and plotted with `tcplot.py`, which are also available on the attached CD. Again, the measurements are done with `iperf` and each test is repeated 50 times with each of the four congestion control algorithms:

CUBIC The default congestion avoidance in Linux that backs off if packet loss is registered.

Vegas Congestion is detected by increased round trip times, so that random packet loss is not regarded as congestion.

Veno A combination of Vegas and the older Reno algorithms, where the former is used to detect *congestion states* in which case the recovery time is increased.

Westwood+ Uses low pass filtering of acknowledgement arrival rates to estimate channel capacity.

Two different tests are made in the Alice-and-Bob topology: Unidirectional flows and bidirectional flows. Both tests are run for each congestion avoidance algorithm with three different configurations of the relay node:

- Network coding enabled with MAC address selected by lowest TQ value.
- Network coding enabled with MAC address selceted by lowest TQ value weighted with a random value as described in Section 3.3
- Network coding disabled.

The range of conducted tests and configurations is listed in Table 5.1.

	CUBIC	Vegas	Veno	Westwood+
Unidirectional	TQ Only TQ Weighted No Coding	TQ Only TQ Weighted No Coding	TQ Only TQ Weighted No Coding	TQ Only TQ Weighted No Coding
Bidirectional	TQ Only TQ Weighted No Coding	TQ Only TQ Weighted No Coding	TQ Only TQ Weighted No Coding	TQ Only TQ Weighted No Coding

Table 5.1: Test matrix for TCP

Chapter 6

Test Results

This chapter presents the results of the performance evaluation of CATWOMAN. The primary outcome of the test is a verification of the expected throughput increase, when network coding is enabled in congested networks. The evaluation is carried out in the previously described test network. Three configurations of the test environment have been used with UDP flows; a simple Alice-and-Bob topology with bidirectional flows through a relay node, an X topology that relies on overhearing transmissions from other nodes, and a cross with two intersecting bidirectional flows through the relay. Finally, a test is included that addresses the performance of CATWOMAN with unidirectional and bidirectional TCP flows.

The tests are all run at night when the interfering traffic on the wireless channel is expected to be low. The results should however not be compared across tests, as the node placement and channel conditions have varied slightly between tests.

6.1 Alice-and-Bob Topology

The first test consists of a simple Alice-and-Bob topology with bidirectional UDP flows between the end nodes. Alice and Bob can not exchange packets directly, and all traffic must pass through the relay node. This setup is illustrated in Figure 6.1.

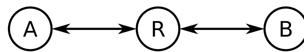


Figure 6.1: Alice-and-Bob test topology. Two flows intersects at the relay node: One from Alice to Bob, and one from Bob to Alice.

6.1.1 Expectations

The following reasoning is given by Fang Zhao in *Distributed Control of Coded Networks*[Zha10]. Let the offered loads from the nodes be BW_A and BW_B, and the traffic through the relay node be BW_R. The total channel capacity is 1. If the network does not use network coding, the relay node must transmit the same number of packets as Alice and Bob combined. Hence, the total throughput peaks when $BW_A = BW_B = 1/4$ and $BW_R = 1/2$. When the network is completely congested, the MAC layer allocates $1/3$ of the capacity to each node, due to the exponential backoff as described in Section 1.4.2. The total throughput of the network is thus limited by the relay node to $1/3$. In the intermediate period between the throughput peak and congestion, the throughput is limited by the relay to $BW_R = 1 - BW_A - BW_B$. Figure 6.2 shows the theoretical total throughput as a function of the offered load, with and without network coding.

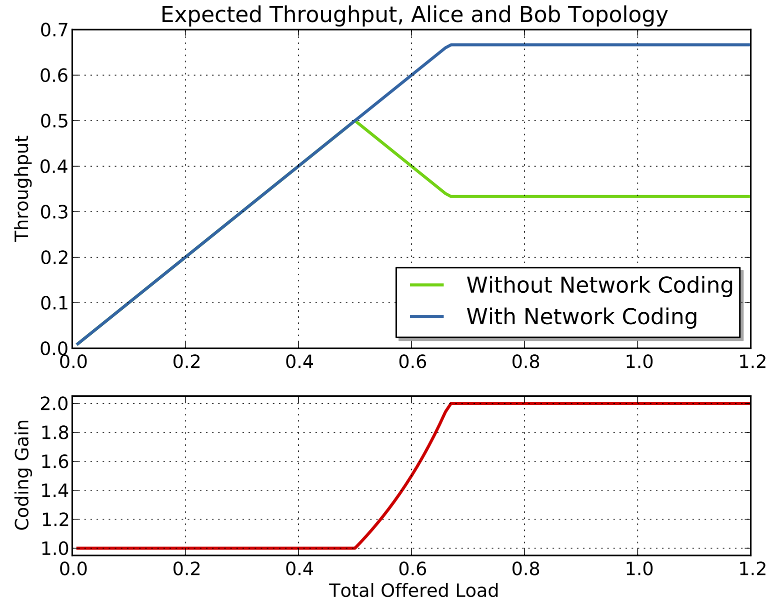


Figure 6.2: Expected throughput for offered load in Alice-and-Bob topology. Network coding should push the point of congestion and level out. Without network coding, the throughput drops when congestion occurs.

With network coding enabled, the total throughput peaks at $2/3$, since the required bandwidth for the relay matches the MAC layer allocation. Given that the offered loads from Alice and Bob are equal, this throughput is maintained independent of the total offered load. A theoretical coding gain of 2.0 is thus possible, partly because the MAC layer fairness limits the uncoded throughput. If Alice and Bob transmit with unequal rates, the coding gain is limited by the rate of the minimum flow, as coding requires packets from both flows.

6.1.2 Measurements

This section presents the measurement results from the Alice-and-Bob topology. Figure 6.3 shows the aggregated throughput vs. offered load from all nodes as measured in the test network. The shape of the throughput curve without network coding generally matches the model. The throughput should follow the offered load, and peak when the nodes transmit with half the capacity of the channel. For the 11 Mbit/s 802.11b network used in the test setup, the maximum achievable throughput is 5.4 Mbit/s, as measured in Section 5.1, and the throughput should thus peak when the offered load reaches 2.7 Mbit/s. When the 200 kbit/s of constant beacon traffic is taken into consideration, this seems to match the test results that peak at just below 2.5 Mbit/s. When the network is completely congested and the MAC fairness prohibits channel access for the relay node, the throughput should settle at $1/3$ of the maximum capacity. Again, this seems to be consistent with the measurements which settle at approximately 1.7 Mbit/s. The transition phase from the peak to the point of complete congestion is longer, which could be caused by uneven channel allocation as explained in the next paragraph.

With network coding enabled, the throughput is expected to follow the offered load and settle at a peak of 3.6 Mbit/s, independent of the load. As seen on the plot, the throughput peaks at 3 Mbit/s, but then drops to a speed of approximately 2.7 Mbit/s. Consequently, the coding gain only reach 1.6 instead of 2.0 as predicted by the model. Figure 6.4 and 6.5 show the individual throughputs and coding gain for Alice and Bob. Both nodes follow the same general shape as the aggregated throughputs, but Bob achieves speeds lower than those of Alice in both the coded and uncoded case. In the band from 0 to 2.5 Mbit/s, when the network is not congested, the

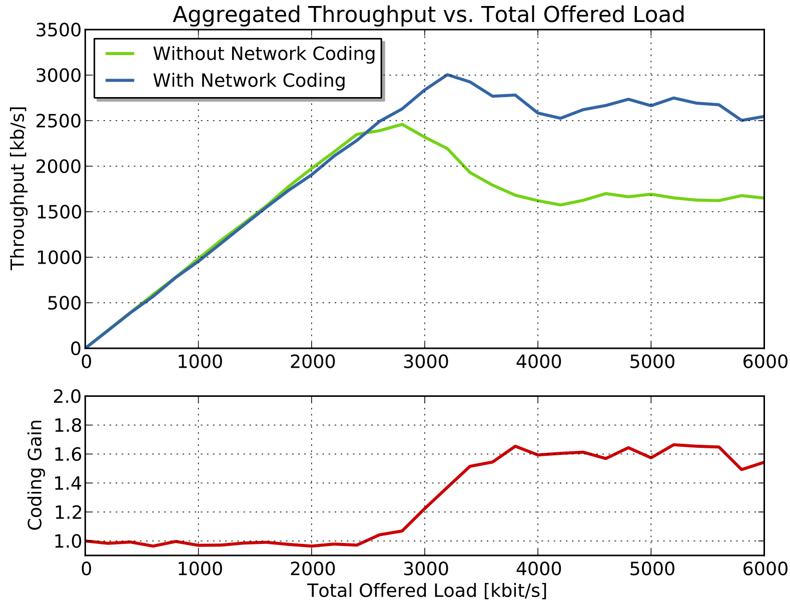


Figure 6.3: Measured aggregated throughput for total offered load in Alice-and-Bob topology. The coding gain is seen after the network becomes congested.

throughput for both nodes is slightly lower when coding. Due to the hold interval, the node waits for a packet to code with even though the network is not congested. This causes an increase in the packet loss, and a decrease in the throughput because of the coding.

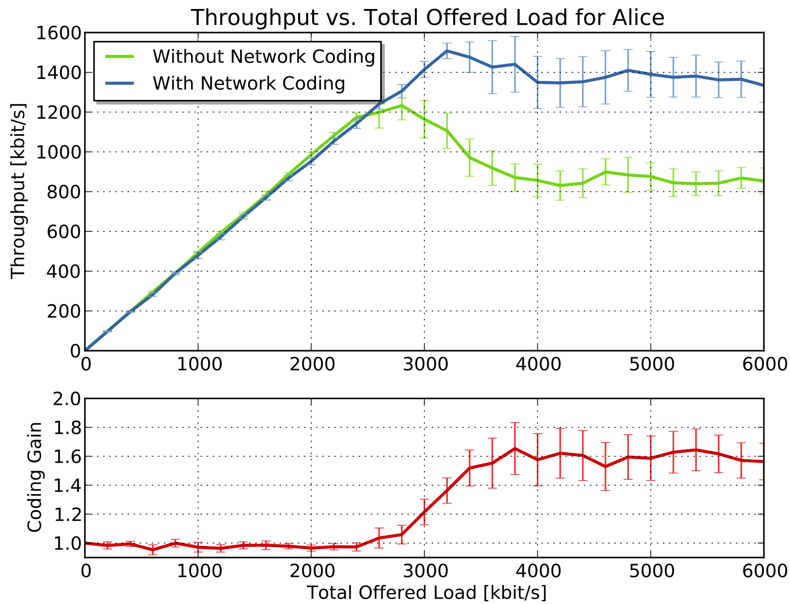


Figure 6.4: Measured throughput vs. total load for Alice. The error bars show the 95% confidence interval.

The relay node can only code packets with the minimum rate of the flows from Alice and Bob. If Alice transmits with 50 % of Bob's rate, only half of Bob's packets can be coded with a

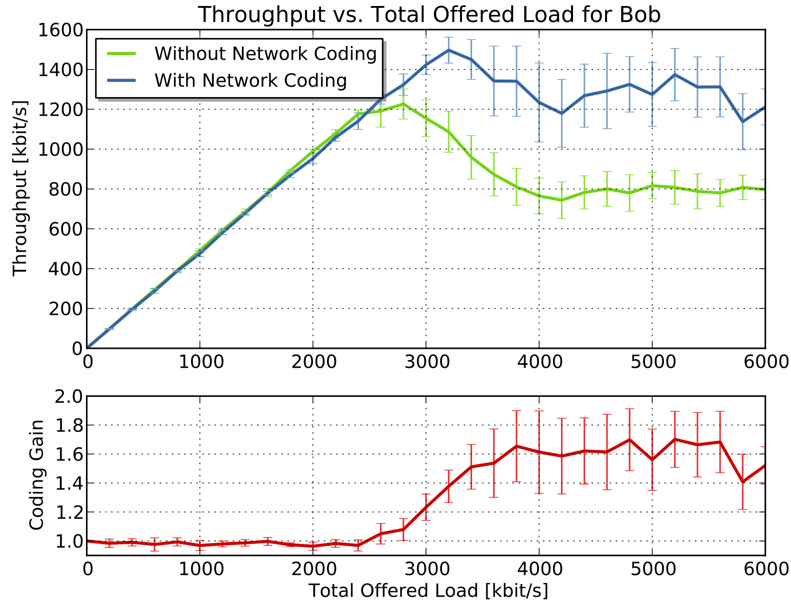


Figure 6.5: Measured throughput vs. total load for Bob. The errors bars shows the 95% confidence interval.

corresponding packet from Alice, and the remaining packets must be sent uncoded. The 802.11 MAC allocates 1/3 of the channel capacity to each node when the network is congested, and the relay can send the same *number* of packets even though the effective throughput is doubled for coded packets. The coding gain as a function of access distribution for Alice, is plotted in Figure 6.6. As expected, the coding gain peaks at 2.0 when the nodes both receive 50 % of the link, since every packet can be coded with a packet from the opposite flow, thereby doubling the effective throughput. If there is a difference in the transmission rate of the nodes of 20 %, the maximum achievable coding gain will drop to 2/3, because the relay node must send 20 % of the packets without coding, and is still only allocated one third of the bandwidth. Furthermore, when there is contention for the channel, the overall capacity is lowered because of frame retransmissions, and because the inter-frame spacing is increased due to exponential backoff timers.

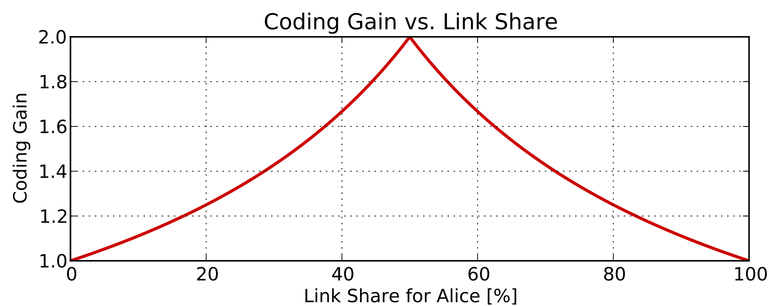


Figure 6.6: Coding gain as a function of link share for Alice. The coding gain decreases as link share becomes unequal.

Figure 6.7 and 6.8 show the number of RTS and data frame retransmissions for Alice and Bob. RTS frames are retransmitted if a node fails to get a matching CTS in the virtual CSMA scheme. Similarly, the data frames are retransmitted if no acknowledgement is received from the destination node. Congestion occurs at a lower rate when network coding is not enabled, and the

number of retransmissions of both RTS and data frames increase earlier. Bob have more frame retransmissions in both the coded and uncoded case, which explains the lower throughput because not all the packets from Alice can be coded. In the congested period, Bob has approximately 15 to 25 % more retransmissions than Alice, and consequently lower throughput, which indicate that the MAC does not divide the medium access completely fair between the two nodes if their link qualities differ.

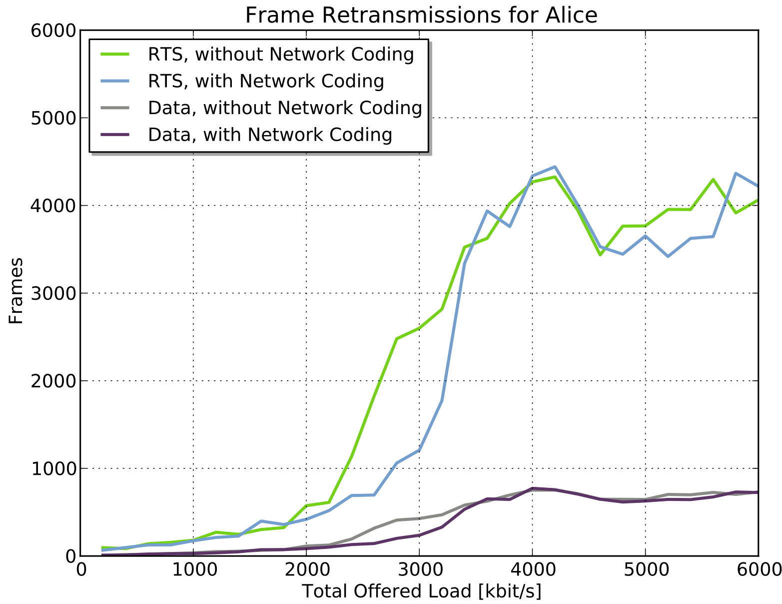


Figure 6.7: Frame transmission errors for Alice in the Alice-and-Bob topology. The increase in retransmissions for both RTS and data frames, is delayed when using network coding.

The plot in Figure 6.9 shows the number of forwarded and coded packets by the relay node. The numbers have been scaled, such that the number of packets required to achieve the same throughput without coding is equal to 1. The total number of packets is thus 0.5 when every packet is coded, since each coded packet carry the same data as two uncoded.

The offered loads are equal from both nodes, and the inter-packet period is expected to be the same. When the load is low and the period between packets is higher than the 10 ms hold time, a lost or retransmitted frame will result in the packet from the opposite flow to time out in the hold queue and be forwarded. When the load is increased, almost every packet can be coded with a packet with the opposite flow and the total number of packets drop to 0.5 of the corresponding non-coded case. If a packet is lost, a new one will most likely arrive before the packet in the hold queue times out. When the total offered load exceeds approximately 3 Mbit/s, the network becomes congested. The nodes are allocated uneven shares of the channel as explained, and the number of coded packets decrease since there is a lack of packets to code with.

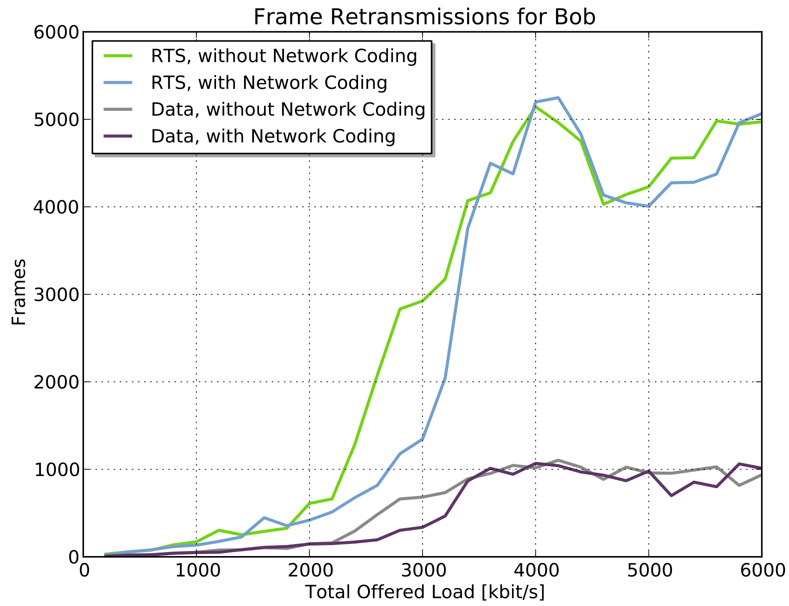


Figure 6.8: Frame transmission errors for Bob in the Alice-and-Bob topology. The increase in retransmissions for both RTS and data frames, is delayed when using network coding.

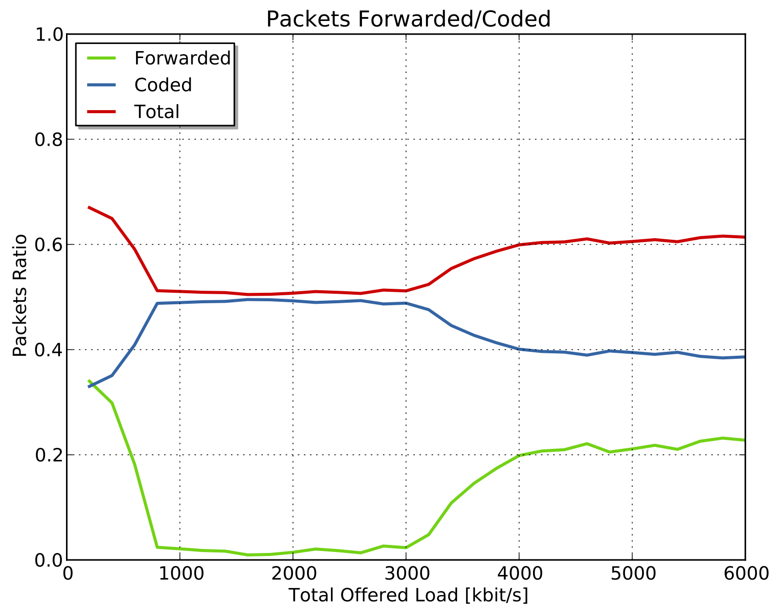


Figure 6.9: Number of forwarded and coded packets vs. throughput. The numbers have been normalized such that 1.0 is the number of packets transmitted if coding is was not used.

6.2 X Topology

This test addresses CATWOMAN performance with transmission overhearing. An X topology with four end-nodes and a relay is used. Alice and Bob each has a unidirectional flow to an opposing node, and all traffic must pass through the relay. The relay node can send a coded packet to Dave and Charlie, provided that they have both overheard the respective transmissions from Alice and Bob.

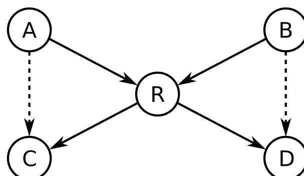


Figure 6.10: X test topology. Node A transmits packets to Node D through the relay node, and Node B transmits packets to Node C through the relay node. Node C overhears packets sent by Node A, and Node D overhears packets sent by Node B.

6.2.1 Expectations

Ideally, the theoretical throughput for the X topology is the same as for the Alice-and-Bob topology. However, this requires perfect overhearing from Alice to Charlie, and from Bob to Dave. As shown in Section 3.4, the packet loss not only depends on the link between the overhearing nodes, but also on the link between the sender and the relay. Due to the increased packet loss because of missing MAC layer retransmissions, the coding gain is expected to be lower than is the case with Alice and Bob.

6.2.2 Measurements

Figure 6.11 shows the aggregated throughput from Alice and Bob to Dave and Charlie. Similar to the Alice-and-Bob scenario, the throughput should peak at half the channel capacity for the uncoded case. The maximum achievable capacity was measured to 5.4 Mbit/s in Section 5.1, and the throughput should thus peak at 2.7 Mbit/s. As seen from the plot, the peak already occurs when the offered load reach 2.15 Mbit/s, which could be caused by retransmissions because of poor links between the nodes. Enabling network coding gives an increase in the throughput, and shifts the point of congestion to approximately 2.5 Mbit/s, compared to the 3.6 Mbit/s predicted by the model. It is noted that the coding gain therefore only reach 1.4, compared to 1.6 in the Alice-and-Bob topology, and 2.0 as predicted by the model.

Looking at the achieved throughput for the individual nodes shows a clear advantage to Bob when the network becomes congested. The throughput vs. offered load is shown in Figures 6.12 and 6.13. Without network coding, both nodes peak when their load is about 1.1 Mbit/s. However, as the loads are increased further, Bob maintains rates of around 1 Mbit/s whereas Alice drops to approximately 700 kbit/s. Almost the same trend is observed when network coding is enabled. Consequently, Bob achieves coding gains between 1.4 and 1.6, while Alice only sees gains of 1.2 to 1.4, leading to a joined gain of 1.4 as shown in the aggregated plot.

As was the case in the regular Alice-and-Bob topology, the coding gain is limited if the two nodes are not allocated equal shares of the transmission slots. The number of RTS and data frame retransmissions from Alice and Bob are shown in Figures 6.14 and 6.15. It is clear that Alice's throughput is severely affected by failures to get CTS frames from the relay and authorization to send the frame. Alice transmits four to five times as many transmission requests as Bob, and the throughput inequality becomes even more distinct than in the Alice-and-Bob case. The amount of retransmitted data frames is also higher for Alice, which indicates that the link between the relay and Alice could be poor in either direction.

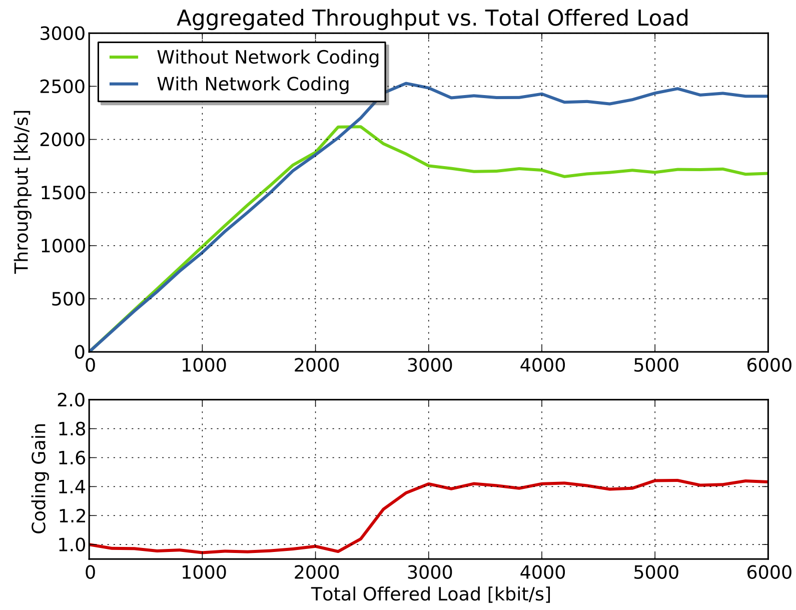


Figure 6.11: Measured aggregated throughput for total offered load in X topology. The coding gain is seen when the network becomes congested.

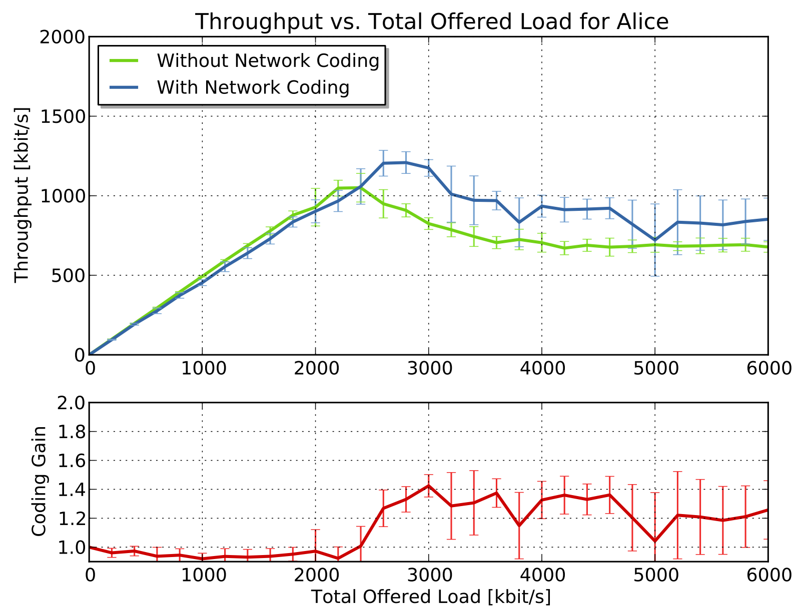


Figure 6.12: Measured throughput vs. total load for Alice. The error bars show the 95% confidence interval.

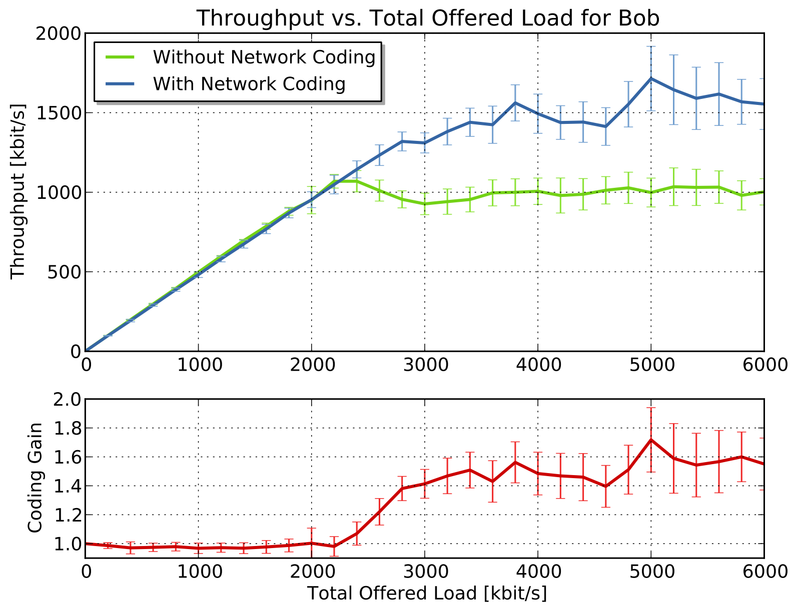


Figure 6.13: Measured throughput vs. total load for Bob. The errors bars shows the 95% confidence interval.

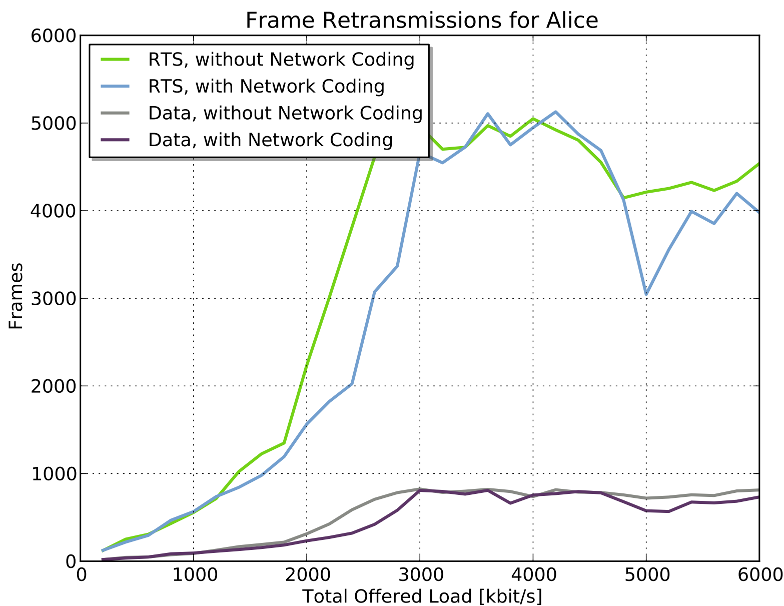


Figure 6.14: Frame transmission errors with and without network coding for Alice in the X topology.

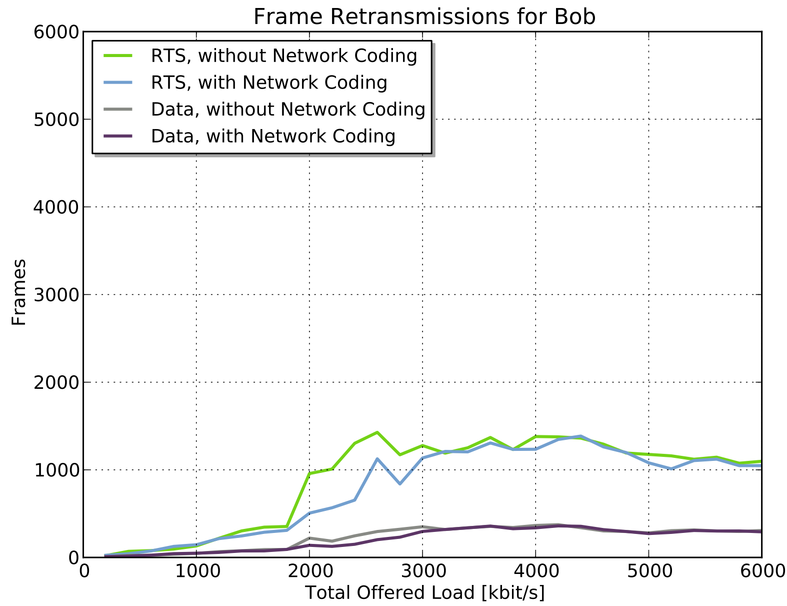


Figure 6.15: Frame transmission errors with and without network coding for Bob in the X topology.

The number of forwarded and coded packets in shown in Figure 6.16. Generally, the plot has the same shape as in the Alice-and-Bob topology; first a phase with low contention for the network and low coding, then a period where almost every packet is coded, and finally a phase with heavy contention and a decrease in the number of coded packets. Once again, the effects of unequal link allocations becomes apparent when the network is congested. In the phase of complete congestion, 20 % of packets were forwarded of in the Alice-and-Bob topology, compared to almost 40 % in the X.

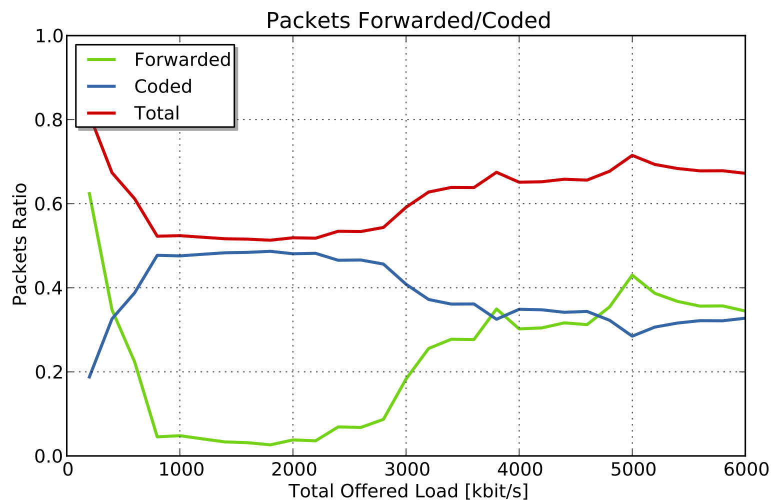


Figure 6.16: Number of forwarded and coded packets vs. throughput, normalized so that 1.0 is the number of packets transmitted if coding is not used.

Some of the decrease in throughput, compared to the Alice-and-Bob scenario, can be explained with Charlie and Dave failing to overhear the transmissions to the relay node. Figure 6.17 shows the amount of coded packets sent from the relay node, and the number of decoding failures on each of the destination nodes. As expected, the number of decoding failures increase as the number of coded packets increase. Charlie overhears transmissions from Alice, and has a notably higher number of decoding failures than Dave. This further supports that the link from Alice to other nodes is inferior.

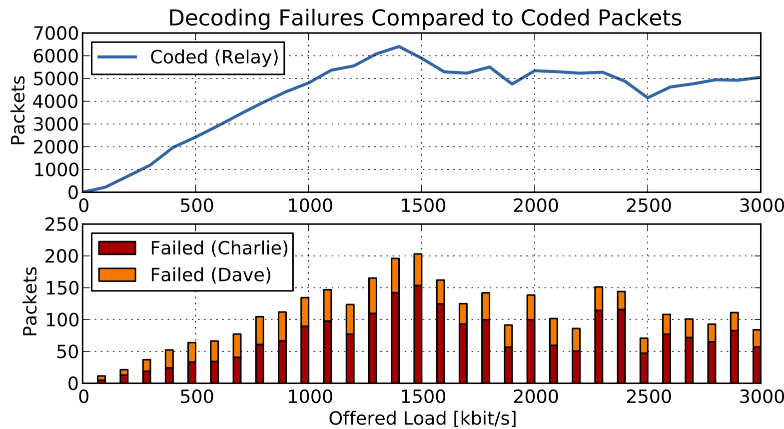


Figure 6.17: Number of packets coded by the Relay (top) compared to the number decoding failures at Charlie and Dave (bottom).

Figure 6.18 shows the average delay, measured from Alice to Bob and in the opposite direction. In the uncongested phase, the hold queue introduces up to 10 ms of delay for uncoded packets, which renders the coded case inferior. When the congestion sets in, network coding reduces the number of packets in the transmission queue yielding a lower delay.

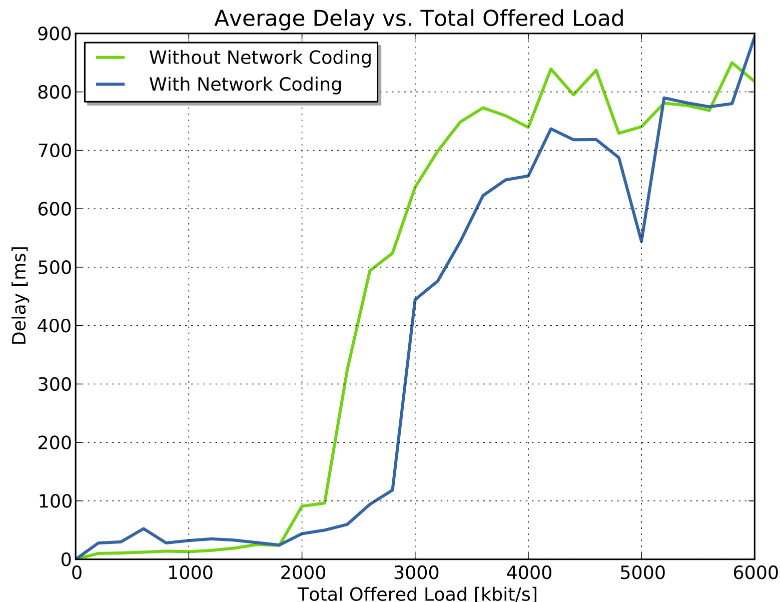


Figure 6.18: Average delay vs. offered load in X topology. With network coding, delay is increased before congestion occurs and decreased after.

6.3 Crossed Alice-and-Bob Topology

This test considers two bidirectional flows between Alice and Dave, and between Bob and Charlie. The topology and flows are depicted in Figure 6.19. Again, all packets must be routed through the relay node, and the total throughput is thus bounded by the capacity allocation for the relay. Note that the relay is configured to code packets for the Alice-and-Bob topology only and not the X topology.

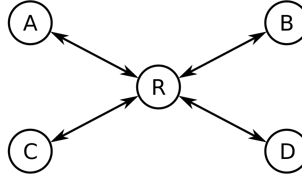


Figure 6.19: Crossed Alice and Bob test topology. Two bidirectional flows intersect at the relay node. One between Node A and Node D, and one between Node B and Node C.

6.3.1 Expectations

The expectation for the total throughput in a crossed Alice and Bob scenario follows the same reasoning as in the simple Alice and Bob topology. Without coding, the relay node must transmit the same amount of packets as the remaining nodes combined. The total throughput therefore peaks when $BW_A + BW_B + BW_C + BW_D = BW_R = 1/2$. When the network is completely congested, the MAC layer allocates $1/5$ of the total bandwidth to each node. As the total throughput is bounded by the capacity of the relay node, this settles at $1/5$. For the intermediate loads, where the network is partially congested, the total throughput is limited by the relay node to $BW_R = 1 - BW_A - BW_B - BW_C - BW_D$. The expected throughputs as a function of offered load, with and without coding, are illustrated in Figure 6.20. If each node could overhear transmissions from every other node, except for the one directly across the cross, the relay could code four packets together instead of two. This would increase the total throughput to $4/5$, and the coded throughput curve would resemble the plot in Figure 6.2.

With network coding enabled, the relay only needs to transmit half the packets and the total throughput will peak $2/3$ when $BW_A + BW_B + BW_C + BW_D = 2 \cdot BW_R = 2/5$. At the point where the network is completely congested, the MAC again allocates $1/5$ of the available bandwidth to each node. Due to the network coding, the total throughput will settle at twice the rate of the non-coded case. Again, the theoretical coding gain is 2.0.

6.3.2 Measurements

The test results show a coding gain that peaks at 1.6 before it drops to 1.4. The peak is caused by the later point of congestion achieved with network coding and is thus a result of both the coding gain and the MAC gain. The drop to a total gain of 1.4 is caused by having less benefit of the MAC gain, since the relay node requires $1/3$ of the total capacity, but only gets $1/5$ from the MAC.

The measured throughput peaks of 3495 kbit/s with network coding and 2780 kbit/s without and matches the expected throughputs, which are $5400 \cdot 2/3 = 3600$ kbit/s with network coding and $5400 \cdot 1/2 = 2700$ kbit/s without network coding. When the network is fully congested, the throughput is measured to be 1985 kbit/s with network coding and is below the expected throughput of $5400 \cdot 2/5 = 2160$ kbit/s. This, and the coding gain below the theoretical maximum, is caused by unequal capacity at the nodes. The inequality is illustrated in Figure 6.22, which show the same situation as with the Alice-and-Bob test in Section 6.1. The measured throughput

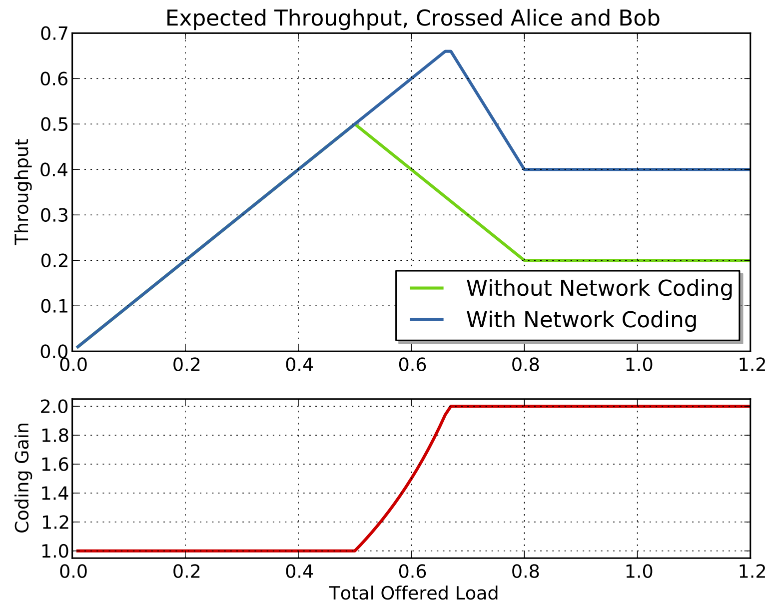


Figure 6.20: Expected throughput for offered load in crossed Alice and Bob topology.

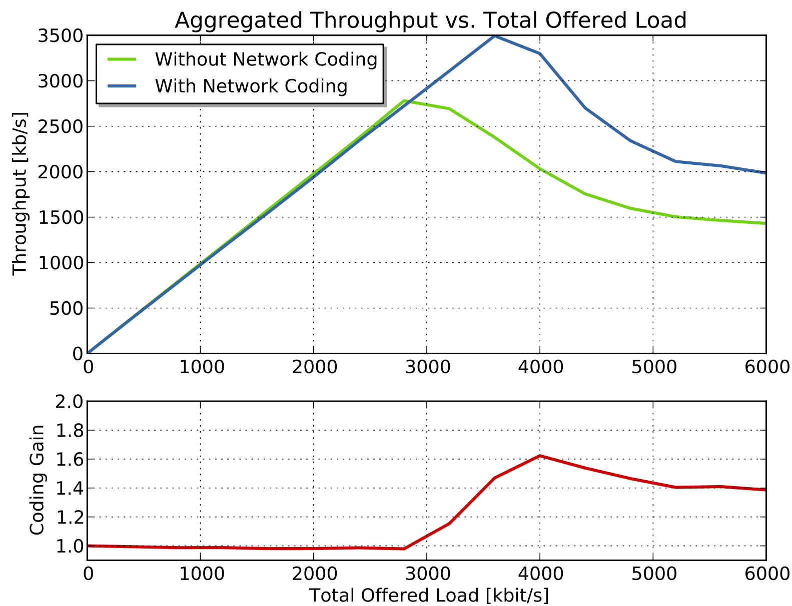


Figure 6.21: Measured aggregated throughput for total offered load in crossed Alice and Bob topology.

without network coding is above the expected throughput, but is not settled and should drop further.

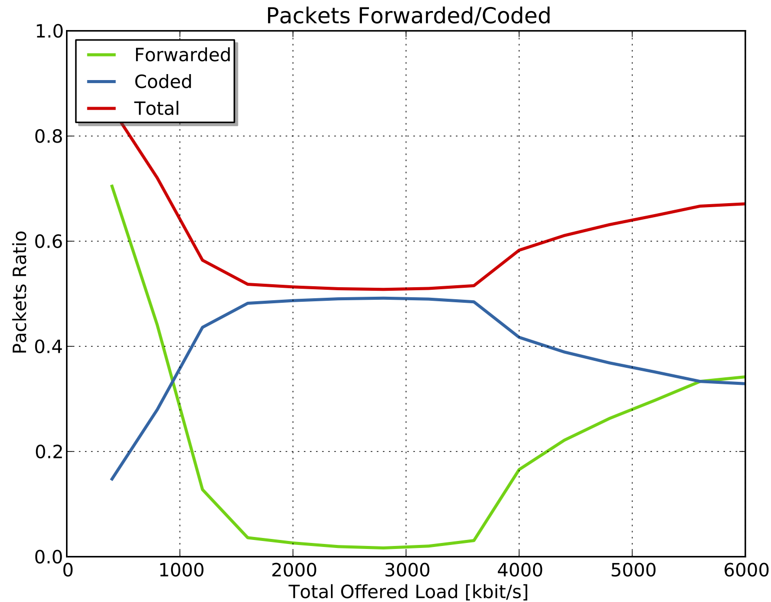


Figure 6.22: Number of forwarded and coded packets vs. throughput, normalized so that 1.0 is the number of packets transmitted if coding is was not used.

The increased throughput when applying network coding comes at a price of increased processing requirements. Figure 6.23 shows the CPU utilization for the relay node, with and without network coding enabled. As the transmission speed is increased, the processing requirements also increase for both cases. The best point to see the effect of applying network coding, is in the range from 2.0 to 2.8 Mbit/s, when almost all packets are coded, and the network is uncongested both with and without coding. Throughout this range, the CPU utilization is approximately 5% higher when coding. The measurements from the other tests also show an increase in CPU usage of 5% or less.

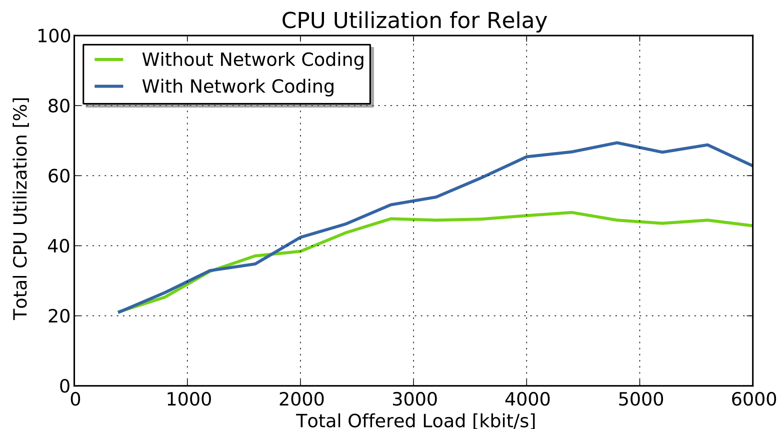


Figure 6.23: CPU utilization for the relay node. When all packets are coded, an increase in CPU utilization is seen. After the point of congestion, more packets are processed with network coding than without, causing higher CPU utilization.

6.4 TCP Performance

The network coding impact on TCP throughput is tested according to the description in Section 5.3, where multiple congestion avoidance algorithms are tested with uni- and bidirectional flows in the Alice-and-Bob topology. Since the primary focus is the gain achieved with network coding, the impact of TCP congestion avoidance algorithms are a secondary focus and these tests are considered to be preliminary.

Because of the increased packet loss that is caused by not having retransmission to the secondary destination of coded packets, the coding gain is expected to be less than with UDP. For the case with unidirectional flows, where TCP sends acknowledgements in the opposite direction, a minor coding gain is expected, as these can be coded together with data packets.

When using only the TQ value reported by B.A.T.M.A.N Adv. to select the destination for the MAC header, the flows are expected to be unequal, because one node experiences an increase in packet loss on average. By enabling the random weighting, the packet loss should be distributed more fair to the two nodes and result in more equal throughputs.

With regard to the congestion avoidance algorithms, the algorithms designed for lossy pipes are expected to perform better than CUBIC, which takes packet loss as a sign of congestion. The weighted TQ selection, where the MAC header destination with retransmission is randomly weighted before comparison, is expected to be more fair in cases where the TQ value is almost identical for both destination.

6.4.1 Unidirectional Flows

The results from unidirectional TCP flows are illustrated in Figure 6.24 and shows no coding gain for neither non-weighted or weighted TQ selection. There is also a clear drop when coding with Vegas as congestion avoidance algorithm. This is explained by the fact that Vegas consider increased round trip times as a sign of congestion. The limited amount of coding opportunities in the unidirectional flow leads to packets being held back in the coding packet pool, which gives an increased RTT. Except for Vegas, the coding gain for both weighted and non-weighted TQ selection is close to one.

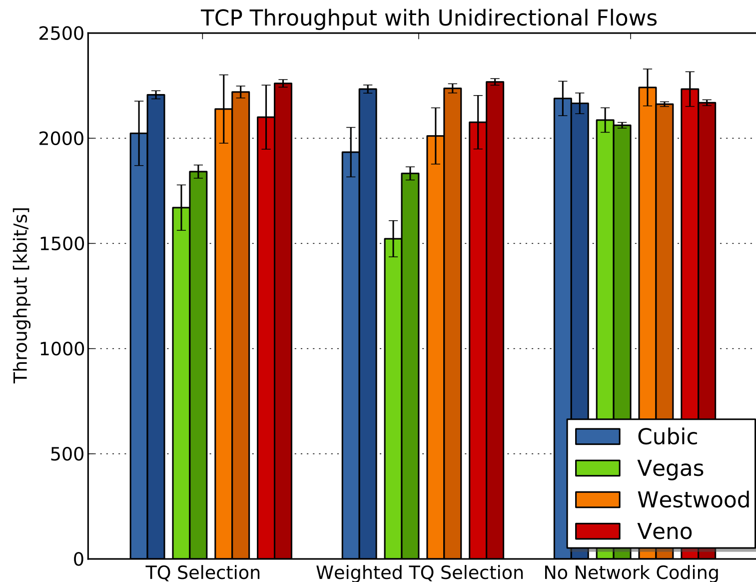


Figure 6.24: Throughput with unidirectional TCP flows for different congestion avoidance algorithms. Light bars are throughput for Alice and dark bars are for Bob.

6.4.2 Bidirectional Flows

The coding gains with bidirectional flows are not above 1.2, which is below the expected result. The random weighted TQ selection shows a reduction in fairness, which is also unexpected.

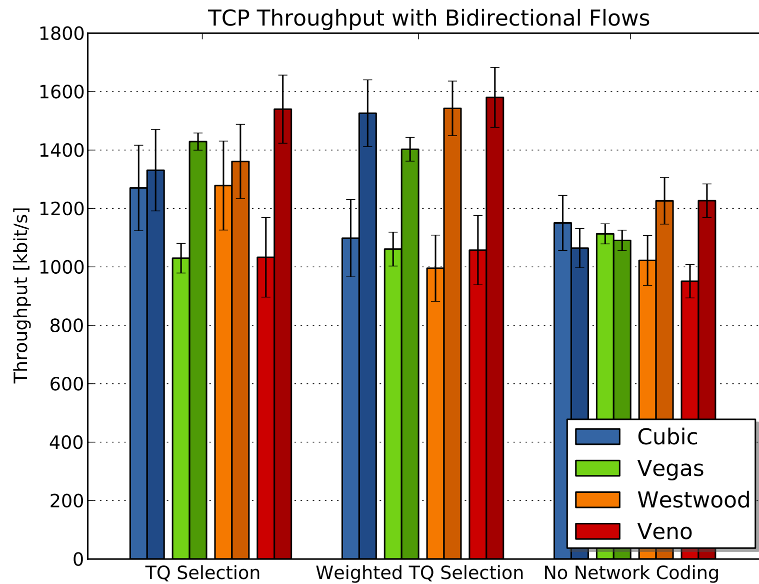


Figure 6.25: Throughput with bidirectional TCP flows for different congestion avoidance algorithms. Light bars are throughput for Alice and dark bars are for Bob.

Coding Gain

The unexpected low gain seen when coding is partly caused by the additional packet loss that is introduced by not having link layer acknowledgements on the secondary destination in a coded packet. For CUBIC and Westwood, this is interpreted as congestion and the back-off therefore prevent higher throughput.

With Vegas and Veno, the CATWOMAN hold time is suspected to prevent the increase in throughput, because only a subset of the the packets are coded. This increases jitter, which can be interpreted as congestion by the to algorithms.

Random TQ Selection When testing CUBIC and Westwood+, the use of random weighted TQ values is less fair than using plain TQ values when selecting the destination address for the MAC header. The difference in fairness is not seen for Vegas and Veno, which depend on RTT. This indicates that the difference is caused by unequal loss of acknowledgements between the flows.

6.5 Evaluation Summary

This section summarizes the test results from the previous performance evaluation. With CATWOMAN enabled, all UDP tests shows an increase in the total throughput when the network becomes congested. Coding gains of up to 1.6 are observed in both the Alice and Bob, and crossed Alice and Bob scenarios. The test results for the uncoded measurements seems to be consistent with the model from [Zha10], and a clear increase in throughput is observed due to a combination of the network coding and the MAC layer fairness.

All the tests shows an uneven allocation of transmissions for the end-nodes, with a clear preference for the strongest node. The weaker nodes thus has to send more transmission requests to gain access to the channel. This limits the number of packets from one flow, leading to a decline in the number of packets that can be coded. As more packets require forwarding, the total throughput drops. Especially the X topology suffers from the uneven MAC fairness, which limits the coding gain to 1.4. The increase in processor load because of the coding, is measured to be approximately five percent.

The TCP tests shows very asymmetric throughputs for the nodes. With unidirectional flow, either no change or a decrease in throughput is observed depending on the link quality and congestion control algorithm. For bidirectional flows, Bob sees a coding gain of 25 to 40 %, while Alice experiences a minor decrease in throughput. The measurements are included as a preliminary test, and require more investigation before a final conclusion on the effects of network coding can be made.

The aggregated and individual coding gains are listed in Table 6.1 for all the tests. The coding gains should be read as the minimum and maximum values measured in the congested period. The TCP results lists the coding gain with the weighted TQ selection.

Topology	Aggregated	Alice	Bob	Charlie	Dave
UDP, Alice and Bob	1.57-1.62	1.53-1.63	1.58-1.66	-	-
UDP, X	1.38-1.43	1.06-1.41	1.41-1.71	-	-
UDP, Crossed Alice and Bob	1.40-1.62	1.43-1.63	1.38-1.61	1.39-1.60	1.36-1.62
TCP, Unidirectional	0.81-0.99	0.73-0.93	0.89-1.05	-	-
TCP, Bidirection	1.12-1.21	0.95-0.97	1.26-1.43	-	-

Table 6.1: Coding gain summary. Charlie and Dave only transmits data in the crossed Alice and Bob topology, which is why no coding gains are listed for the other topologies. The TCP results show the coding gain with the weighted TQ selection.

Chapter 7

Conclusion

This work presents the CATWOMAN scheme for transparent network coding in wireless mesh networks. CATWOMAN is designed and implemented as an extension to the B.A.T.M.A.N Adv. routing protocol. Without adding additional management overhead, the scheme exploits the topology information from the routing layer to identify coding opportunities in the network.

The wireless medium is inherently shared, which allows overhearing of transmissions from other nodes. Network coding can benefit from this by combining overheard packets, and utilizing the information already present in the destination nodes. This provides a potential gain in throughput since the information from multiple packets can be combined into a single transmission and decoded by the receiving nodes.

The CATWOMAN implementation is evaluated in a simple test setup with five nodes in three different topologies. The network coding is performed on the link layer, and the tests are primarily focused on UDP traffic in order to measure the maximum achievable performance without influence from the transport layer. Measurements includes throughput, delay and ratio between coded and forwarded packets. A preliminary test of TCP performance is also conducted to get an initial idea of the impact from network coding on the congestion control algorithms.

The results from the performance evaluation shows an increase in achievable throughput up to 62% for intersecting UDP flows in the Alice and Bob topologies. The tests reveal an unequal allocation of transmission slots when the nodes have different link qualities, with a preference for the node with the strongest link. As the network coding requires packets from multiple crossing flows to obtain a gain, the performance increase is lower than predicted by the theoretical model. In the X topology, where transmission overhearing is required, network coding showed a maximum increase of 43%, again caused by the inequality of transmissions slots. The gain is further reduced because of packet loss for the overhearing nodes.

CATWOMAN inserts a hold queue where packets wait for up to 10 ms for another packet to be coded with. When the network utilization is low, the network coding thus introduces an additional delay. However, the network coding improves delay times in a congested network, because the transmit queue is emptied with double rate.

The test network uses completely equal traffic rates from the nodes, which is a best-case scenario for network coding. Simple topologies and traffic patterns are however beneficial in order to get an understanding of the synergies from the network coding and the fairness of the 802.11 MAC. Full scale tests are required to evaluate the performance of network in real network topologies and realistic traffic patterns.

The TCP test shows only moderate or no gains for unidirectional flows, and decreases fairness in some configurations. Further investigation is necessary to determine the cause of the decreased performance and to mitigate the issues.

The tests with network coding and the CATWOMAN protocol shows promising first results and indicate that network coding is beneficial in common use cases. Together with B.A.T.M.A.N

Adv., the CATWOMAN implementation form a solid base for further development and shows potential for the adaption of network coding in real wireless mesh networks.

7.1 Further Work

The issues with TCP performance should be addressed and the cause investigated in further detail. If the performance drop is indeed caused by the increased packet loss, a possible solution could be to send out one or more redundant packets with a combination of the last packets transmitted. If a node missed a single packet, it can thus be extracted with some probability. The repair packet rate should be varied depending on the link qualities to the destination nodes. Changing the algorithm for selection of the MAC destination node could also have a potential effect, and remove some of the asymmetry in the results.

The B.A.T.M.A.N Adv. community has proposed a new Neighborhood Discovery Protocol (NDP) to separate the link quality estimation and routing table updates in different packet types. NDP messages will be sent more frequently than originator messages, and contain information about direct neighbors of the source node and their link qualities. CATWOMAN's coding opportunity discovery and decision can directly benefit from this new protocol, as the use of TTL values from Originator Messages is no longer needed.

Energy consumption should also be addressed. Network coding requires the nodes to be in promiscuous mode, which causes increased processing overhead since all packets are filtered in software. Wireless routers often operate in promiscuous mode when bridging multiple networks, and the prevalence of this requirement should be investigated in further work.

Appendix A

CATWOMAN Configuration Tutorial

To setup a wireless mesh network with B.A.T.M.A.N Adv. and CATWOMAN, OpenWRT and the patched B.A.T.M.A.N Adv. code must checked out and built as described in this appendix.

Building the Patched B.A.T.M.A.N Adv. Module

To build the patched B.A.T.M.A.N Adv. module, check out the NC branch from github:

```
% git clone git://github.com/jledet/batman-adv-nc.git batman-adv
% cd batman-adv && git checkout nc
```

To use the module on the local machine, make sure that the kernel headers are installed and build the module with:

```
% make
```

Load the module and mesh it:

```
% sudo insmod ./batman-adv.ko
```

The network coding can be enabled and disabled by echoing 1 or 0 into the `sysfs`:

```
# echo 1 > /sys/devices/virtual/net/bat0/mesh/catwoman
```

Building OpenWRT

The CATWOMAN enabled OpenWRT source code consists of three parts: The OpenWRT source, a B.A.T.M.A.N Adv. package description and the B.A.T.M.A.N Adv. module source. The build tool is configured with relative paths, so make sure that the following commands are run from the same directory as the first command above. To get the CATWOMAN enabled OpenWRT source, check it out from github:

```
% git clone git://github.com/jledet/backfire-git.git backfire
% git clone git://github.com/jledet/openwrt-batman-feeds.git
% git clone git://git.open-mesh.org/batctl.git
```

A subset of the tools used in the router are pulled in from subversion:

```
% cd backfire && ./scripts/feeds update
```

Load the configuration for the OM1P Router and start building the image:

```
% cp .config-om1p .config
% make -j3
```

Go get a cup of coffee and play a set of table tennis, because building the image takes a while.

Flashing the Image

When the image is built, it is available in the `./bin/atheros/` folder. To upload it to the router, use the flash tool provided by Cloudtrax:

```
% cd ..  && svn checkout http://dev.cloudtrax.com/downloads/svn/ap51-flash-ng
```

The tool uses promiscuous mode to communicate with its built-in TCP stack. Before starting the tool, unplug the power from the router and connect the router to the ethernet interface of the computer. Then start the tool and plug in the power:

```
% cd ap51-flash-ng
% sudo ./ap51-flash eth0 ../backfire/bin/atheros/openwrt-atheros-combined.squashfs.img
```

Change `eth0` accordingly if the router is connected to another interface. The flashing is completed when the flashing tool prints:

```
Booting after successful flash
```

The first boot takes a while, because the image has to build the filesystem. Once booted, the router attempts to acquire an IP address with DHCP. If this fails, it falls back to a static IP address (`10.10.0.200/24`). One can now log in with SSH:

```
% ssh root@10.10.0.200
```

The default password is `gotham`.

Configuring the Testbed

The routers automatically configures the mesh network, so put up the desired topology (e.g. the Alice-and-Bob topology described in Section 5.1.) If the router is unable to acquire an IP address with DHCP, it configures a bridge with the wired and wireless interfaces. This makes the slaves able to communicate through the mesh network, so setup the slaves with IP addresses similar to the setup illustrated in Figure A.1.

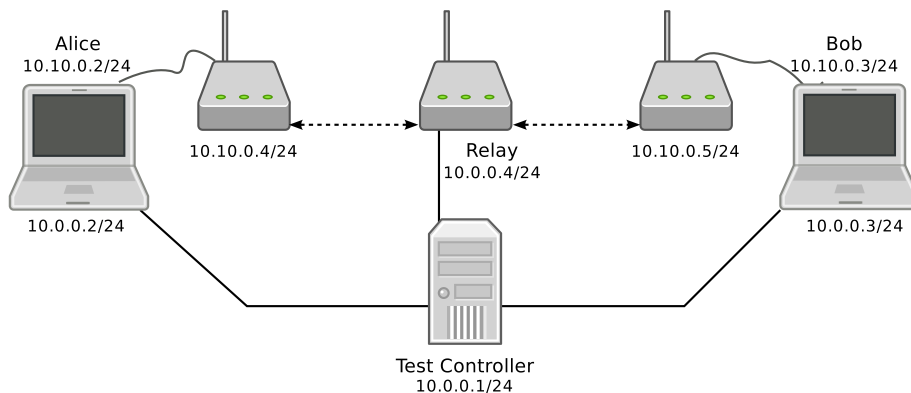


Figure A.1: Suggested configuration of the testbed.

The test controller uses SSH keys to automatically log in and run commands on the slaves, so create a user for this purpose on the slaves:

```
% sudo useradd catwoman
% sudo passwd catwoman
```

From the test controller, generate an SSH key and copy it to the slaves:
`% ssh-keygen && ssh-copy-id catwoman@<slave-address>`

Now verify that the test controller can access slaves without typing a password and the testbed should be ready for testing.

Configuring the Test Tool

The `catbench.py` and `catplot.py` benchmarking tools can be checked out from:
`% git clone git://github.com/jledet/catbench.git`

The `catbench.py` tool reads a configuration from either `ab.py` or `x.py`, which is chosen upon execution. In this case, the Alice-and-Bob topology is chosen, so configure it according to the testbed. In Python Source Code A.1 a configuration for the setup in Figure A.1 is listed.

```
1  # Create Alice
2  alice = Slave("alice")
3  alice.set_ip("10.0.0.2", "10.10.0.2")
4  alice.set_node_ip("10.10.0.4", "de:ad:be:ef:10:04")
5
6  # Create Bob
7  bob = Slave("bob")
8  bob.set_ip("10.0.0.3", "10.10.0.3")
9  bob.set_node_ip("10.10.0.5", "de:ad:be:ef:10:05")
10
11 # Create Relay
12 relay = Node("relay")
13 relay.set_ip("10.0.0.4", "de:ad:be:ef:10:01")
14
15 # Connect flows
16 alice.set_flow(bob)
17 bob.set_flow(alice)
18
19 # Setup route checks
20 alice.add_route(bob.node, relay)
21 bob.add_route(alice.node, relay)
```

Python Source Code A.1: `catbench.py` configuration for Alice-and-Bob testbed topology.

The test is now ready to run:
`% ./catbench.py --output=test.pickle --duration=30 --min=100 --max=3000 --step=100 --config=ab`

This command executes the test with the following parameters:

- `--output=test.pickle` The measured data is saved in `test.pickle`.
- `--duration=30` Each test with `iperf` is running for 30 seconds.
- `--min=100` The test starts at rate 100 kbps.
- `--max=3000` The test is stepped up to rate 3000 kbps.
- `--step=100` The test is performed for each 100 kpbs between the min and max parameters.

- `--config=ab` The testbed configuration is read from ab.py.

During the test, `catbench.py` prints each intermediate result and the estimated time left.

Plotting the results

The main tool used to plot the results of a test is `catplot.py`. When a test is completed, the results are plotted with the command:

```
% ./catplot.py --data=test.pickle --out=.
```

This parses the data in `test.pickle` and saves the plots in a subfolder of the current folder: `./test`. Also, a few default plots are showed on the screen. More plots can be showed on the screen by parsing parameters to `catplot.py`. These can be seen in the tail of the file. To plot the number of failed decodings, use the `failed.py` tool:

```
% ./failed.py test.pickle failed.pdf
```

To view the TQ values to and from the relay node, use the `tq.py` tool:

```
% ./tq.py test.pickle tq.pdf
```

TCP Tests

To run a test with TCP instead of UDP, the `catcp.py` tool is used. Since the test changes TCP congestion on the slaves, it must be allowed to do so without typing a password. This is allowed by adding the following line to `/etc/sudoers`:

```
root ALL=(ALL) ALL
```

Be aware that this is highly insecure and should be disabled after testing. The TCP tests can now be run with:

```
% ./catcp.py | tee tcp.txt
```

The results are then plotted with:

```
% ./tcplot.py tcp.txt tcp.pdf
```

Figure A.2: Two test slaves and one test node. One of the slaves is connected to the node on the picture and the other to a node above the table.

Appendix B

Attached CD

Contents:

- `./backfire/` - The OpenWRT source.
- `./backfire.img` - Binary image for the OM1P router.
- `./batman-adv/` - The B.A.T.M.A.N Adv. source code with the CATWOMAN implementation.
- `./batctl/` - The B.A.T.M.A.N Adv. configuration tool source code.
- `./catbench/` - The tools used to test CATWOMAN and plot the results.
- `./data/` - Measurements from the tests conducted in this report.
- `./openwrt-batman-feeds/` - The package descriptions of B.A.T.M.A.N Adv. for OpenWRT.
- `./master.pdf` - This report.

Bibliography

[Abr70] Norman Abramson. The aloha system: another alternative for computer communications. In *Proceedings of the November 17-19, 1970, fall joint computer conference*, AFIPS '70 (Fall), pages 281–285, New York, NY, USA, 1970. ACM.

[ACLY00] R. Ahlswede, Ning Cai, S.-Y.R. Li, and R.W. Yeung. Network information flow. *Information Theory, IEEE Transactions on*, 46(4):1204 –1216, July 2000.

[Aic06] Corinna 'Elektra' Aichel. The OSLR story, 2006. `http://www.open-mesh.org/wiki/open-mesh/The-olsr-stor`.

[DABM03] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, and Robert Morris. A high-throughput path metric for multi-hop wireless routing. In *Proceedings of the 9th ACM International Conference on Mobile Computing and Networking (MobiCom '03)*, San Diego, California, September 2003.

[Gas05] Matthew S. Gast. *802.11 Wireless Networks: The Definitive Guide*. O'Reilly Media, Inc., 2nd edition, 2005.

[HDM+10] G. Hiertz, D. Denteneer, S. Max, R. Taori, L. Cardona, J.and Berlemann, and B. Walke. Ieee 802.11s: The wlan mesh standard. *IEEE Wireless Communications*, pages 104–111, Feb 2010.

[IEE99] Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications. IEEE Standard 802.11, June 1999.

[KRH+06] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Médard, and Jon Crowcroft. Xors in the air: practical wireless network coding. *SIGCOMM Comput. Commun. Rev.*, 36:243–254, August 2006.

[LLM11] Tianji Li, Douglas Leith, and David Malone. Buffer sizing for 802.11 based networks. *IEEE/ACM Transactions on Networking*, 19(1):156 –169, February 2011.

[Ope11] Open-Mesh OM1P Professional Mini Router, 2011. `http://www.open-mesh.com/index.php/professional/professional-mini-router-eu-plugs.html`.

[Zha10] Fang Zhao. *Distributed Control of Coded Networks*. PhD thesis, Massachusetts Institute of Technology, 2010.