COM SYS Communication & Distributed Systems

RWTH AACHEN UNIVERSITY

Robust Link Selection and Channel Assignment for Centrally Managed Wireless Mesh Networks

Bachelor Thesis

Konstantin Manna

RWTH Aachen University, Germany

Chair of Communication and Distributed Systems

Advisors:

Dipl.-Inform.	Christoph Wollgarten
Dipl.-Inform.	Jó Ágila Bitsch Link
Prof. Dr.-Ing.	Klaus Wehrle
Prof. Dr.	Bernhard Rumpe

Registration date:	2014-05-20
Submission date:	2014-09-22

I hereby affirm that I composed this work independently and used no other than the specified sources and tools and that I marked all quotes as such.

Hiermit versichere ich, dass ich die Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, den 22. September 2014

Acknowledgments

I would like to thank Christoph Wollgarten at LANCOM Systems for enabling and supervising this thesis, who despite his huge workload always had a few minutes to spare to help me with my problems and create the AutoWDS status tool, which was much appreciated. I also want to thank Jó Ágila Bitsch for supervising this thesis at COMSYS and his great feedback on this work during several meetings. Furthermore I want to express my gratitude to Paul Smith who had some tips and templates for writing this thesis. Last but not least I thank Prof. Wehrle and Prof. Rumpe for giving me the chance to write this thesis.

Abstract

Using a wireless backbone in a Wireless Distribution System (WDS) can be tricky as performance decreases with increasing size due to interference, especially if channels and network topology are not selected carefully beforehand. Additionally network dissociations may occur easily if crucial links fail as redundancy is neglected.

Therefore we present an algorithm and its implementation which addresses this problem by finding a network topology and channel assignment that minimizes interference and thus allows a deployment to increase its throughput performance by utilizing more bandwidth in the local spectrum. Our evaluation results show an increase in throughput performance of up to 9 times or more compared to a baseline scenario where an optimization has not taken place and only one channel for the whole network is used. Furthermore our solution also provides a robust network topology which tackles the issue of network partition for single link failures by using survival paths.

We achieve this gain in performance by utilizing multi-radio accesspoints and enhancing the Dijkstra, Jarník, Prim (DJP)-algorithm for graphs by a scoring system combined with a greedy channel assignment.

Contents

1

Introduction

The rise of smartphones and other devices that are capable of using IEEE 802.11 Wireless Local Area Network (WLAN) bring along applications that exhaust any way to reconnect to the internet in order to down an up-load personal data, advertisement or media with a high footprint in storage. YouTube, WhatsApp, Facebook and similar apps move high quality images and videos to and from end-users smartphones. Since the number of smartphones in the field are constantly on the rise, this creates an additional strain for the backbone and access-links. As users prefer to use WLANs instead of mobile internet-connections due to their costs and limitations with respect to available bandwidth, a WLAN infrastructure like a WDS needs careful planning. Especially use cases like camping sites or large scale deployments (widespread rural areas) with a lot of users put high requirements on the WLAN backbone. In this paper we consider LANCOM *AutoWDS*, a WDS which implements an easily deployable WDS for a set of APs in order to create a wireless infrastructure for a specific area.

1.1 Motivation

AutoWDS in its current form does not scale well as throughput performance and an increasing number of connectivity failures quickly bring the system to its limits. Our goal is to optimize this system by boosting throughput performance and decreasing the number of occurring simple link failures. We want to achieve this by utilizing multiple radios, channels and redundant links (survival paths) for establishing connections among the APs, as using only a single channel places artificial limits on the achievable bandwidth. This means also we only deal with the physical connectivity (link layer) and are independent of a routing system, which could be placed on top of our solution. The problem we face doing this and that our work will elaborate on is that a good selection of links and channels is not trivial even for a moderately sized scenario. To tackle this challenge we use centrally run greedy algorithms to give us a better solution to our problem for a static environment.

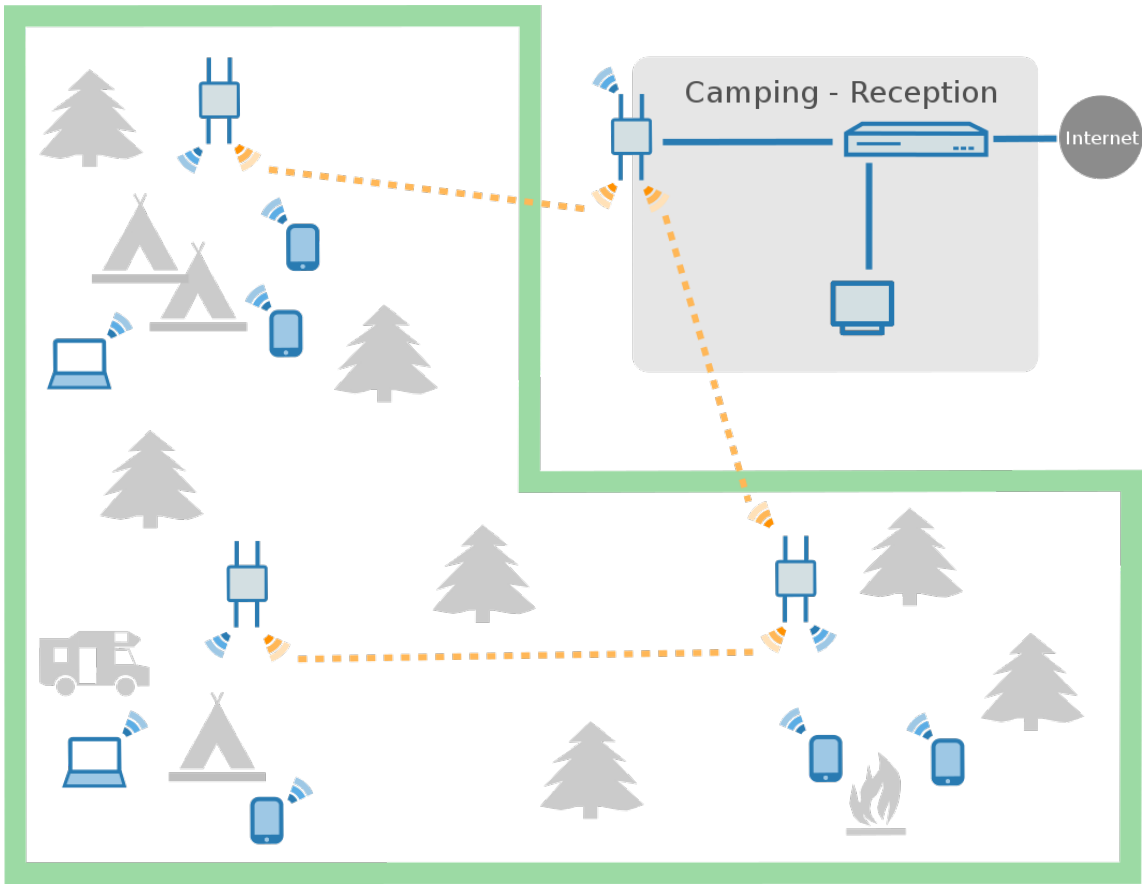


Figure 1.1 WDS deployment at a camping site. Typical scenario for using a WDS system instead of wired APs.

1.2 Structure of Thesis

At first we will recall terms and concepts used in this work followed by taking a look at the requirements and restrictions a possible solution would have to deal with. In Chapter 4, we will then review related work on this field and outline why those approaches have not been pursued. Subsequently we will describe our algorithms for choosing a topology, creating backup links and the successive channel assignment, ensued by the implementation of the aforementioned. We will also give an example on how to run the algorithms yourself for a simple scenario. In Chapter 7 the procedure is evaluated as a whole in practice and compared to the basic version of *AutoWDS*. Finally we conclude our findings by portraying open issues and identifying promising approaches to pursue.

2

Background

This chapter provides background information on wireless networks and recalls the graph-theoretic basics we used in this work. At first we will take a look at APs followed by illustrating the bigger picture on how they work within a network. Then, we examine the way APs communicate through channels, how those communications are affected by interference and how they access the shared medium. Finally, we will outline the implementation of the WDS we are optimizing. In the second part, we explain spanning trees on graphs and depict graph attributes like edge- and vertex-connectivity. Subsequently, we will give a detailed description of how we map a real world Accesspoint (AP)-infrastructure-setup to network graphs in order to compute a solution and conclude this part by discussing the relevance of the problem COLORING for our approach.

2.1 Wireless Networks

Wireless networks are computer networks which use radio-waves for communication. They are used in scenarios where using wires is not possible or cumbersome like systems where devices are mobile or subject to drastic link-quality changes. There are different scales where wireless networks are utilized:

- Smaller systems like a Personal Area Network (PAN) where a smartphone uses Bluetooth to connect to headphones. The typical range lies within a couple of meters.
- Medium sized systems like a WLAN with laptops or smartphones connecting to an infrastructure through a wireless AP. The typical range is up to 100 meters.
- Large scale systems like television broadcasting with Digital Video Broadcasting – Terrestrial (DVB-T) through satellites with ranges of up to 35,400 kilometers.

We will focus on medium sized systems in our work.

2.1.1 Accesspoints

A wireless AP is a device which allows WLAN-clients to connect to its network. Typically it serves as an entry point to a network-infrastructure and ultimately the internet like a common network switch, but APs can also be used in different ways. In the following we will describe the modes we use APs in:

Infrastructure-mode: Using this mode, the clients connect themselves to the APs in order to get access to the network behind the APs, like fileservers or the internet. To do so one or more APs announce their services through small broadcasted packets called beacons, which include a Service Set Identifier (SSID), the name/identifier of the wireless network. Those are used to differentiate multiple wireless networks from each other if used in the same area. SSIDs, if received, are then used by the clients to establish a link to the APs.

Point-to-Point mode: This mode is defined by a static link between exactly two interfaces in contrast to infrastructure- and ad-hoc- mode, where this affiliation is not determined. It allows an AP to create a dedicated link to another AP using the same mode. This link would consequently be shared only by these two APs, additionally the transmitted data is encrypted. This mode is useful in scenarios where complex connection-constellations can not be achieved by client-infrastructure-mode combinations solely. Alternatively, one could also use the ad-hoc mode, which was not available, as it was not implemented yet in LANCOM devices.

Client-mode: Here the AP behave the same as ordinary clients like laptops or smartphones. They search for wireless networks announced by APs in infrastructure mode and connect themselves to these. Often this mode is used to enable devices which lack 802.11-hardware to access a WLAN through a wired link to those APs.

Figure 2.1 Indoor AP (left) and an outdoor version deployed (right) [21].

2.1.2 Channel

A WLAN channel as specified by the IEEE 802.11 family uses a specific frequency-range in the Ultra High Frequency (UHF) or Super High Frequency (SHF) radio-scope in order to modulate data digitally on carrier waves. Doing this allows transmitting data from a sender station to a receiver station and creates a network-link between them. WLAN uses two license-free frequency bands at 2.4 GHz and additionally bands around 5 GHz. As we can see in Figure 2.2, channels have overlapping regions, which can lead to conflicts. Two independent pairs of radios in the same area can communicate almost free of interference, if they use different, non-overlapping channels. Newer APs may use channels with 40 MHz bandwidth in order to further increase throughput with the drawback of creating more overlapping frequency-regions and therefore possibly creating sources of interference for other channels close-by. The overlapping regions are a problem especially in the 2.4 GHz band. If wider versions of channels in the 5 GHz band are used, also those are affected, although not as bad as in the 2.4 GHz, where the channels overlap even at 20 MHz width.

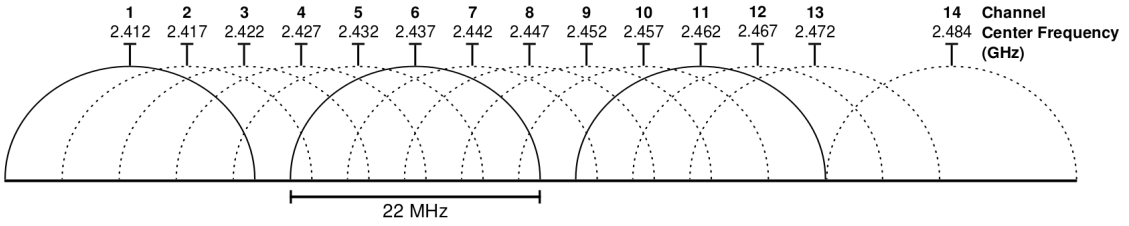


Figure 2.2 Mapping of channels to frequencies for the 2.4GHz band [22].

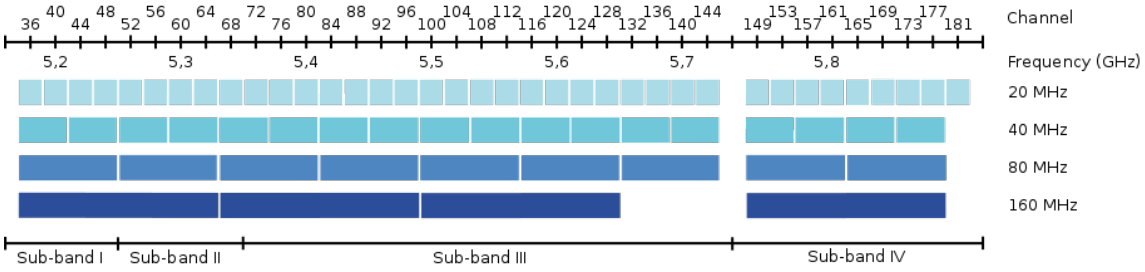


Figure 2.3 Channels in the 5 GHz band with different channel width.

2.1.2.1 CSMA / CA

Carrier Sense Multiple Access / Collision Avoidance (CSMA/CA) is a network protocol that describes how to access a shared medium, like a shared WLAN channel. It behaves as described by Janssen [17]:

> CSMA works on the principle that only one device can transmit signals on the network, otherwise a collision will occur resulting in the loss of data packets or frames. CSMA works when a device needs to initiate or transfer data over the network. Before transferring, each CSMA must

check or listen to the network for any other transmissions that may be in progress. If it senses a transmission, the device will wait for it to end. Once the transmission is completed, the waiting device can transmit its data/signals. However, if multiple devices access it simultaneously and a collision occurs, they both have to wait for a specific time before reinitiating the transmission process.

For a sizeable AP-deployment the accumulated backoff-waiting times increase as more data is transmitted and backoff-timers are triggered more often. This effect also quickly degrades network performance, especially in a mono-channel setup.

Another interesting, but undesired effect in accessing the medium despite following the CSMA algorithm is the hidden-station-problem. It describes a scenario where an AP B is in range of A and C, but A and C do not see each other (see Figure 2.4). Then, if the situation occurs that the two outer APs simultaneously want to send data to the AP in the middle, as they can not notice the other AP sending data, their packets collide at B. Therefore they effectively transmit less or nothing at all, since B is not able to correctly decode the packets. The effect is similar to two people talking to a third person at the same time. To alleviate the problem the RTS/CTS extension was introduced, which introduces small Request To Send (RTS)-requests and Clear To Send (CTS)-announcements. This is basically the station/person that is talked to announcing to whom it is going to listen for the next period of time. Note that the RTS/CTS not completely solves this problem, but lessens its impact.

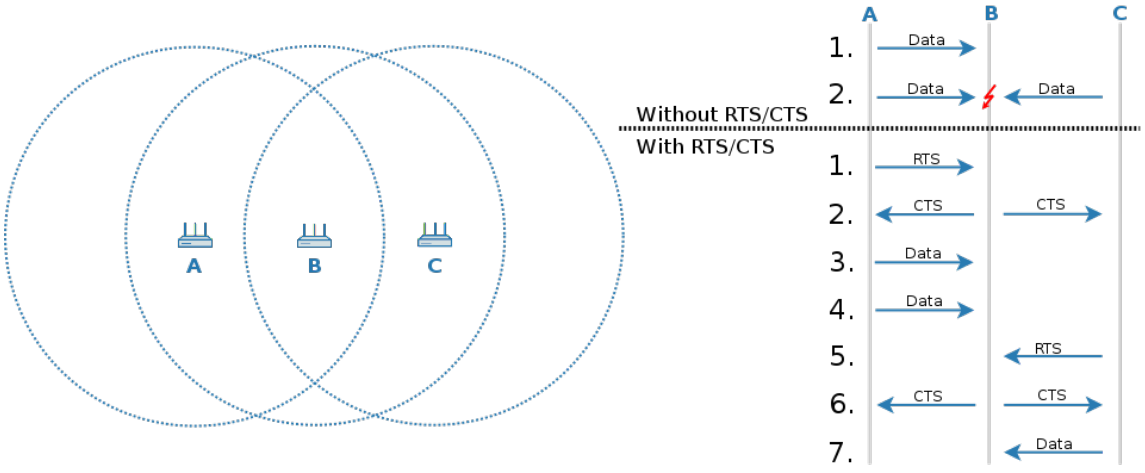


Figure 2.4 Hidden-Station Problem and the RTS/CTS solution. A sends data to B in step 1. Because of the range-limitation, C does not recognize the transmission of A to B. If B then wants to also transmit data to B, it checks if the medium is free, which it falsely assumes and then starts to transmit its data. This leads to collisions at B in step 2 and the data received will be corrupt. With RTS/CTS: First A asks B if it may send data to it, which B allows by sending the OK to every station listening (A and C). Then A sends its data to B while C has to wait until B's time slice expires (a CTS defines the amount of time how long a station will listen to another, so there is no need for constant polling). Eventually C succeeds in getting its CTS from A and is able to also transmit its data to B without collisions.

2.1.2.2 Interference

Wireless interference occurs if two senders at the same time transmit data on the same carrier frequency. If so the waves may interfere and be wrongly detected by the receiving stations. This leads to changed bits in the derived data and may result in turn to corrupt messages if more bits are flipped than the forward error correction algorithms can detect. Such a malformed packet may not even be recognized as such, if the preamble[1] is already corrupt. This consequently renders the rest of the packet as unusable noise. If a certain threshold of flipped bits is reached after the preamble, calculating the checksum on those packets will then show that there have been errors while transmitting the data. This will make again the receiving station drop this packet (destructive interference). As the receiving station will not acknowledge the receipt of the packet, the sender will assume the packet was not transmitted correctly and therefore try to send the same packet again. If this process occurs often, throughput performance will suffer as it takes more time to send the same amount of data. This kind of interference is called cooperative and occurs if two radios claim the same channel/medium (co-channel/adjacent-channel interference) if CSMA fails, like in the hidden-station or exposed-station problem. Whereas there is also uncooperative interference from devices like microwaves or cordless phones which do not access the medium purposefully, but rather emit noise by accident on the same frequency. Nevertheless, the effect is the same.

For our purposes we generally define interference for wireless connections as any kind of disruption in communication, that includes both cooperative and non-cooperative interference.

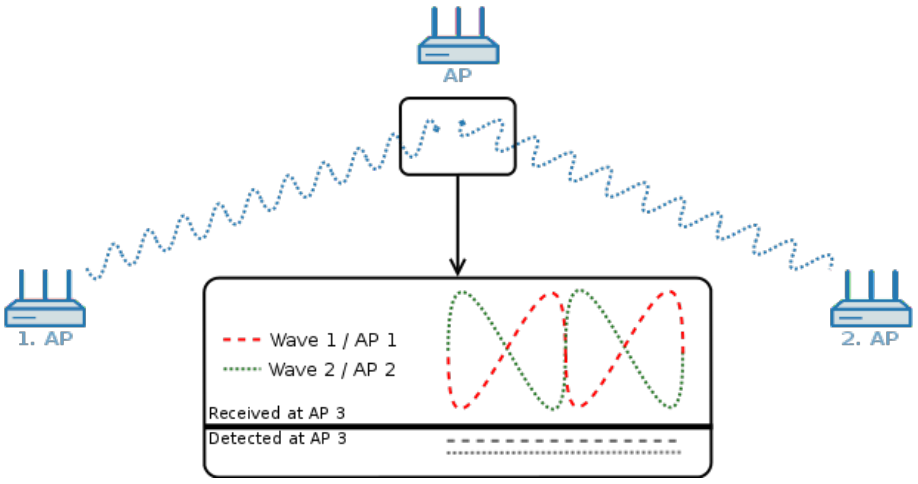


Figure 2.5 Using the same or close-by carrier frequencies, waves may interfere and result in falsely detected bits and render transmitted data corrupt.

Despite solutions like RTS/CTS and others that mitigate the hidden station problem (and alleviate the cooperative interference), there are still some situations where collisions and therefore interference occurs. Additionally at a certain signal-strength level a station, while doing its carrier sensing, may not be able to differentiate noise from another stations transmission. As Padhye et al.[26] mention, interference is the key cause of performance degradation in wireless networks. Hence, our idea is to increase throughput performance by reducing interference as much as possible.

[1]The preamble is a certain signal to indicate an incoming data stream.

2.1.3 Wireless Distribution System (WDS)

A WDS is defined by the DD-WRT team [10] as a system, that

> creates a wireless backbone link between multiple access points that are part of the same wireless network. This allows a wireless network to be expanded using multiple access points without the need for a wired backbone to link them, as is traditionally required. The WDS-enabled AP can accept wireless clients (e.g. wireless laptop users) just as traditional APs would.

They also point out, that WDS is currently not a certified standard of the Institute of Electrical and Electronics Engineers (IEEE) and every vendor implements it in a different way. For an example see Figure 1.1. The idea behind this system is to make increasing the range of an already pre-configured WDS easy by just adding another AP somewhere in range of the other APs and powering it on. Setting up the rest of the infrastructure and configuration from then on is supposed to happen automatically. This system allows us for example to quickly bring a network infrastructure to areas where no existing infrastructures (existing power- or network-cabling) can be utilized for this purpose, without the burden of creating such a costly infrastructure.

2.1.3.1 WLC

The purpose of a WLC is to centrally manage and configure multiple APs automatically - effectively ease the job of the administrator responsible for a wireless infrastructure. They are usually located in central networking-cabinets next to other network core components like switches and servers. Managing such a WDS can also be done through another central entity somewhere else in the network, for example by electing an AP for this job, which is directly connected to the wired Local Area Network (LAN), as it is done by DD-WRT [10]. Taking it one step further and eliminating the central logic at all would then result in a distributed logic on each of the APs, which is however more complex. We will only deal with the centrally, managed scenario and ignore distributed approaches.

Figure 2.6 A LANCOM WLAN Controller [21]

2.1.3.2 WLC Aided Configuration

For a common WLAN infrastructure deployment all APs are connected to the wired backbone and use their radios solely in infrastructure mode to offer clients an entry point into the network. The procedure of establishing such a network can be

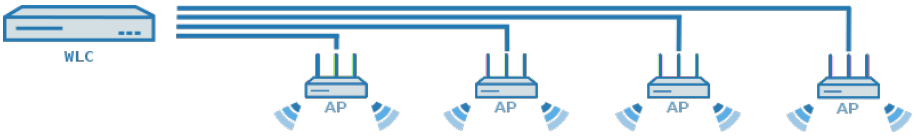


Figure 2.7 Common AP deployment where APs are connected to backbone via wire.

described as follows:

1. APs search for a WLC on the wired network by Internet Protocol (IP)-broadcast.
2. After the authentication of the AP the AP establishes a secure (Datagram Transport Layer Security (DTLS)) channel to the WLC through a Control And Provisioning of Wireless Access Points (CAPWAP)-Layer.
3. AP receives a configuration from the WLC through the secure channel and reconfigures itself accordingly.

2.1.3.3 Automatic WDS

AutoWDS is an extension to the WLC-aided configuration by also allowing such a configuration procedure over wireless links instead of solely wired connections.

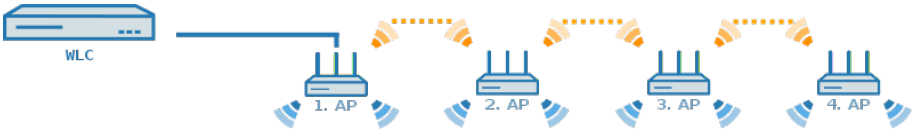


Figure 2.8 *AutoWDS* configuration setup. Only one AP is connected to the backbone via wire. Other APs use their radios to connect to APs that are connected to the backbone via wire (first AP) or relay-APs (2nd AP).

The procedure is enhanced accordingly:

1. The first AP finds a WLC and proceeds as before. Additionally it announces its connection to the WLC over its radios to other APs.
2. The second AP does not find a WLC on its LAN-interface and extends its search on the radios. There it receives the announcement of the first AP. Consequently it connects itself to the first AP by client-infrastructure-mode and offers the same network on its other radios.
3. The second AP establishes a secure connection (Wi-Fi Protected Access 2 (WPA2)) to the first AP and continues its search for a WLC over this link, where it finds the WLC and also receives a configuration (CAPWAP / DTLS) and proceeds as AP one.
4. Gradually all the APs connect each other to the backbone, receive a configuration and spread the WLAN network.

2.2 Graph-theoretic Basics

In the following we will recall the definitions of graphs, directed / weighted edges and paths in a graph.

- A graph in graph theory is a set of nodes and edges between these nodes.
- A directed graph / digraph has directions on its edges defined.
- A weighted graph has weights attached to its edges.
- A graph is called connected if every pair of nodes is connected. Two nodes A,B are connected if there exists a path from A to B. A path is a sequence of edges where each edge has to fit to the next, like the game of domino.

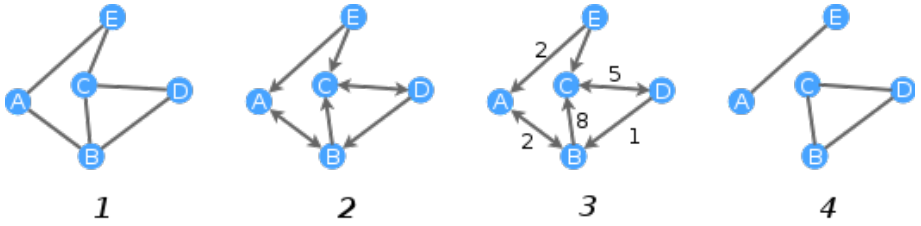


Figure 2.9 Example graphs - 1: undirected 2: directed 3:weighted, directed 4: disconnected

We choose to use graph theory to represent our networks structures for two reasons:

- We are able to easily express all the relevant parts of a network directly in a graph without any modifications to graph theory, like APs and modules by nodes, links between APs with edges and signal to noise ratios with weightedness of edges.
- Graph theory is a well researched field and a lot of problems in it can be solved efficiently and intuitively as we will see later with the DJP algorithm.

2.2.1 k-Vertex-connectivity, k-Edge-connectivity

A graph is called *k-vertex-connected* if there exists a set of k vertices whose removal disconnects parts of the graph. A graph is called *k-edge-connected* if the graph is still connected after the removal of k edges.

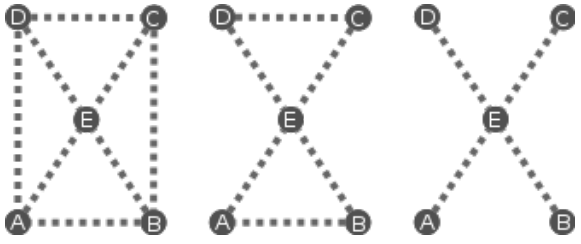


Figure 2.10 Graph-connectivity attributes from left to right: 2-vertex and 2-edge, 0-vertex and 1-edge, 0-vertex and 0-edge

2.2.2 Minimum Spanning Tree

A spanning tree is a subgraph for a given graph which contains all nodes of the original graph, but only the necessary and edge-weight minimal subset of edges which are needed to connect each node to the component. In order to find a minimum spanning tree for a non-negative weighted graph we use the DJP algorithm [6, 28]. It works as described by the following:

1. Initialize the minimum tree with a single, randomly chosen vertex from the given graph.
2. For all edges originating the set of nodes in the minimum tree to nodes which have not been visited yet, select the edge with the lowest score and add it and its nodes to the minimum tree.
3. Repeat step two as long as not all vertices of the given graph are visited.

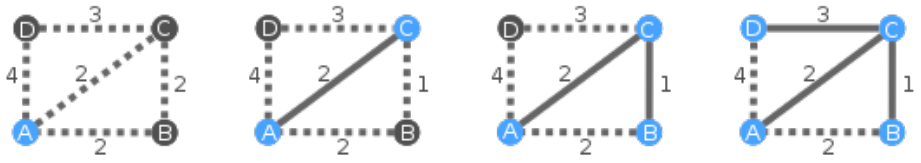


Figure 2.11 DJP algorithm calculating a minimal spanning tree.

2.2.3 Significance of COLORING

Other pursuits of this problem including [20, 29, 30, 33] mention a kinship to the problem COLORING on graphs. On the other hand, Raniwala et al. [30] also mention:

> At first glance, this problem appears to be a graph-coloring problem. However, standard graph-coloring algorithms cannot really capture the specification and constraints of the channel assignment problem. A node-multi-coloring formulation fails to capture [...] communicating nodes need[ing] a common color. On the other hand, an edge-coloring formulation fails to capture the [...] constraint where no more than q (number of NICs per node) colors can be incident to a node. While a constraint edge-coloring might be able to roughly model the remaining constraints, it is incapable of satisfying the constraint of limited channel capacity.

In addition to the modeling problems a potential result of such a COLORING solution still would not make a point on the topology to use. COLORING works on given network graphs and therefore topologies, but we also have to decide how to create and select the network topology. As a consequence, this formal problem or solutions to it are not considered in our work.

2.2.4 Mapping Network to Graph

For our purposes we will use the following mapping from devices and modules to a undirected, weighted graph. Each AP and each module of our APs is represented by a node.

For each AP we add edges with a special attribute ”artificial” to each module of its corresponding AP and describe those edges as artificial or device-module-edges. Those edges also have their Signal-to-Noise Ratio (SNR) value set to the maximum value possible.

If two modules are within receive range of each other, we add an edge between the corresponding module nodes with the average signal-to-noise ratio as the edge-weight. We call those edges module-module connections or real connections. Since the average of the two SNR values might not describe all scenarios well enough (in a scenario where SNR values differ broadly), we easily can adjust the edge-weight to the minimum or maximum of both values.

Furthermore we ignore one-sided discoveries, i.e. one module receives some beacons of the other module, but not the other way round.

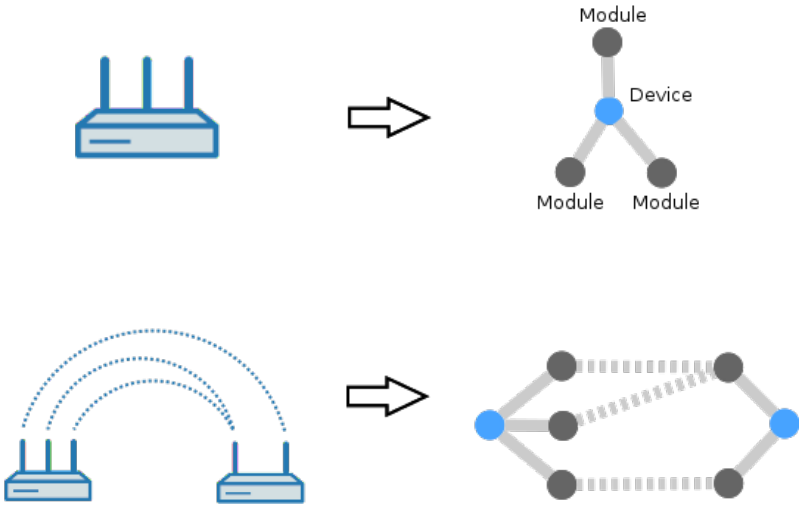


Figure 2.12 Graph representation of one (upper) and two connected APs (lower).

3

Requirement Analysis

We will refer to *AutoWDS* in its current form as *AutoWDS basic* and the improved system, with the features of this requirements analysis realized, is called *AutoWDS extended*. Up to now *AutoWDS basic* automatically uses just one or a random set of channels for its backbone wireless connections. Additionally the topology it creates often reassembles a tree-like structure, since the APs connect themselves to the first AP which comes into range and additionally there are no fallbacks, if a link fails. This results not only in severe bottlenecks for throughput, but also to prolonged downtimes, if a link happens to fail. Since after a disconnection it takes some time for the APs to automatically reconnect to the network, it nevertheless results in an unnecessarily long network-downtime for the attached clients. Especially clients that depend on an uplink connection severely suffer from this lack of alternative / backup links. The following requirements were therefore expressed to tackle these shortcomings.

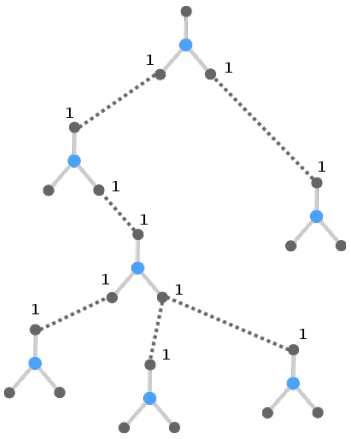


Figure 3.1 *AutoWDS basic* network topology and channel assignment

3.1 Increase Throughput

Since the systems, which are connected to the APs, consume more data every day and with an increasing storage-footprint of new media like streaming High Definition (HD) videos and generally downloading files of up to gigabytes, a wireless system is often solely defined by its capability of moving quantities of data over the air to the clients. This greed for throughput is amplified by an increasing number of devices that take part in radio communication, thus it is also the key metric for quality in *AutoWDS*. Furthermore not a single stream of data should be granted exclusive access to the radio, but multiple streams in parallel (gateway to clients) and also across the network (clients to clients), as the common mesh / gateway / internet scenario does exist, but not exclusively (gateway to one client). Some use cases do only require local communication within the network.

3.2 Reduce End-to-end Link Failures

AutoWDS basic is due to its tree-like structure rather susceptible to network partition. As this may be just an inconvenience to users of devices like cellphones or laptops who are running non critical programs, keeping up a connection to certain parts of the network in an industrial environment gains importance. For example where heavy machines depend on their uplink connections in order to continue proper operation. Hence *AutoWDS extended* has to provide the possibility to create and optionally use redundant connections in the network topology. The failing scenario is defined as one link breaks while others are still usable. A resulting network topology should have the possibility of having the 2-edge-connected attribute, as it may no be needed in all cases.

3.3 Utilize Variable Number of Radios

AutoWDS extended is currently planned to operate with only up to about 100 or less APs with a variable amount of radio-modules per AP. Current versions of APs are equipped with one up to three radios each. It is supposed to work in heterogenous environments with different numbers of radio-modules per AP. As a consequence the available radio modules should be utilized to achieve gains in overall throughput instead of finding an optimal solution which just uses one radio, as this solution is still inferior compared to a solution that utilizes multiple radios. Switching channels in short intervals to serve more than one channel is explicitly not desired as this leads to packet loss and latency for those channels the radio currently not services.

3.4 Use Variable Channel-sets for Assignment

AutoWDS extended will be used in variable environments and hence may face diverse restrictions with respect to the channels that are allowed to be used. Thus *AutoWDS extended* must accept a list of channels only of which it may choose from for channel assignment.

3.5 Restrictions

The new solution has also to be able to operate under the following restrictions:

- Centralized Computation and Configuration
- Static Environment
- Data Link Layer (Layer 2) Usage
- Economic Constraints

3.5.1 Centralized Computation and Configuration

AutoWDS extended must also be able to compute its solution on a central entity in contrast to a distributed fashion. Especially the APs should not be used to compute such a solution as no additional tasks are to be assigned to them. Although not currently planned, a potential algorithm may be included in the WLC in the future.

Reconfiguration of APs and collection of data from the AP-network will only be able through a central WLC. That means the potential network-topology and channel assignment solution has to be set on the WLC and may not be broadcasted or otherwise distributed by or through the APs initially. The reason for such a focus on centralized system is to provide the administrator the opportunity to intervene and set manual wireless connections and other options, which is not as easily feasible in a distributed setup. Additionally the management of certificates for security purposes is built up hierarchically and executed by the central WLC. Especially since the support for doing things in a distributed fashion in the APs and WLCs (like ad-hoc-mode) is missing and would have to be implemented, makes distributed approaches undesirable.

3.5.2 Static Environment

AutoWDS basic and extended is and will be designed for a static environment of APs. That means there will be no highly frequent changes in network topology or in link-quality. Although this does not render APs immobile, changes if any are expected to be slow and gradual.

3.5.3 Data Link Layer (Layer 2) Usage

The solution has to enable communication on Layer 2, as some use cases do not use Layer 3 (IP) for their communication. Note that also a tunnel-mechanism like ingress / egress in Multiprotocol Label Switching (MPLS)[1] [9] is not preferred as this mechanism is currently not implemented in our target platform LANCOM Operating System (LCOS).

[1] In MPLS ethernet frames are encapsulated in a Layer 3 packet for transportation and then later unpacked when delivering to the destination.

3.5.4 Economic Constraints

Neither the central entity nor the APs are limited regarding power consumption. Both will be permanently connected to the power grid and there is no power saving mode which could affect the design or computation. *AutoWDS extended* has to be able to find a solution on demand and in a short timeframe.

4

Related Work

The following four approaches are all solutions that are graph-based and target a centralized and static channel assignment. Those are not the only ones on this matter, but the approaches currently in use and most relevant contestants to solve our problem. Note that there are also numerous solutions on distributed and/or dynamic or semi-dynamic environments, which we will not cover here (See Chapter 3). Except the BFS-CA, the prevalent solutions make excessive use of the conflict graph, which according to Si et al. [31] has difficulties to model the varying number of radios equipped on APs. Also, most of the works solely focus on assigning channels to an existing topology and do not touch the network topology at all, what seems like a handicap as topology control is an essential part in dealing with wireless interference as we will see later.

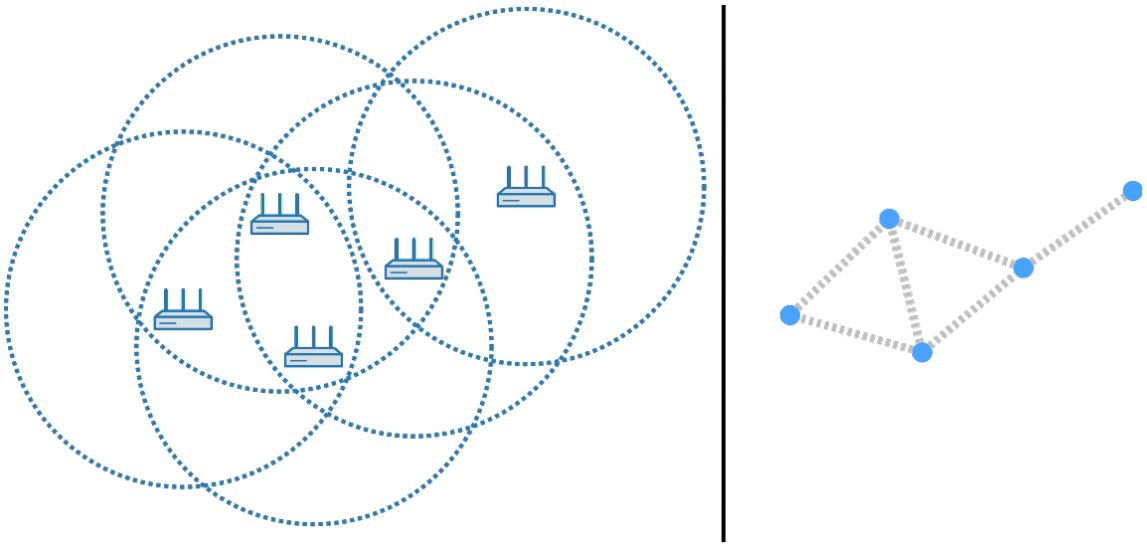


Figure 4.1 Creation of an unit disk graph from the receive range map of APs

4.1 CLICA

The aim of CLICA [23] is to minimize interference conflicts for a given unit disk graph preserving the connectivity. That means it takes the given network topology-graph as granted and tries to minimize the overall interference by resolving conflicts as much as possible. It takes the following parameters as input:

- Unit disk graph
- Number of radios at each node and total number of channels available
- Interference conflicts in form of a conflict graph

CLICA's mode of operation is described by Si et al. [31] as follows:

- Randomly assign a node v the highest priority, then assign other nodes priorities decreasing in the order obtained by depth.
- While traversing the nodes in the decreasing order of their priorities obtained above, assign channels to the incident links of these nodes. The operation of assigning a channel to a link includes assigning this channel to both, a radio at this node and a radio at the neighbor node. Then, the priorities of unvisited nodes are adjusted according to their degree of flexibility, which is the number of channels that a node can choose from without breaking the connectivity preservation. Essentially, the nodes with a lower degree of flexibility will have their priorities increased so that they are visited earlier in the later steps.
- When picking a channel in the above step, a node v_1 picks a channel for its incident link (v_1, v_2) in a greedy manner: A locally optimal choice is made by selecting the channel that minimizes the maximum link-conflict-weight among all links that can interfere with link (v_1, v_2). After a channel is assigned to a link the conflict graph is updated to reflect the new link conflict weights.

The reason why we decided not to use this algorithm is that CLICA only tries to minimize interference by assigning channels the best way possible for a given unit disk graph, which is basically each possible connection. For a highly connected network topology like in Figure 5.1 this would lead to suboptimal results as it does not restrict itself to necessary connections and therefore possesses a lot of potential for interference. The restriction to use only the best and absolutely necessary links is a vital part in order to further decrease interference as much as possible for such a topology.

4.2 INSTC

As pointed out by Si et al. [31], INSTC [35] is similar to CLICA [23] with a some alterations, which is why we will not go into further detail and rather focus on its differences. They introduce Link Co-channel Interference (LCI), which for a link represents the number of links which interfere with this link and serves as a measure of interference. Additionally, they accept k as an input parameter which results in a k-connected graph as outcome to make the topology resilient to node failures. Although this feature would come in handy for our survival path requirement, it does not match our failing scenario of a single link at a time instead of a whole node outage. Using a k-(node)-connected graph instead of an k-edge-connected graph increases the number of edges that have to be utilized (since one failing node involves multiple failing edges). Consequently the resulting network topology has a higher grade of connectivity than it needs to have and therefore conversely affects overall throughput since a higher node connectivity leads to fewer usable different channels. Additional to the weaknesses of CLICA, they also require the number of radio modules to be identical on each AP, which is a deal-breaker for us as we have to be able to fulfill requirement 3.3 (Utilize Variable Number of Radios). The LCI measure introduced here served as a basis for our edgescore calculation in Formula 5.1.

4.3 BFS-CA

BFS-CA [29] extends the common conflict graph with modules, resulting in a Multi-Radio Conflict Graph (MCG), which more closely reassembles the real world setup. This makes it easier for them to deal with the different numbers of radios available on each device. They let the APs (or mesh routers in their case) sniff the network on regular intervals and determine a ranking for the channels. Those rankings are then sent to their central entity, the Channel Assignment Server (CAS). This CAS in turn derives the MCG and assigns each node in this graph a channel in Breadth First Search (BFS)-manner by considering the received rankings of the APs. In order not to partition the network by introducing a Channel Assignment (CA), which would disconnect some links by setting certain radios to different channels, they also use one radio on each AP on an overall-common channel. This common channel is then used for initial setup, management frames and as a backup if other links break in order to keep the graph connectivity attribute.

Although BFS-CA, as others, is still anxious about topology alterations, since at a first glance it might introduce too severe problems like, additional hops for packets, increased interference footprint and higher susceptibility to errors due to longer travel times of packets. Nevertheless it is the algorithm which we were inspired by the most and therefore have some ideas in common. The interesting features we reused and refined are:

- Idea of a ranking system for each possible channel
- Central computation on a CAS, which reassembles our WLC.
- Taking foreign sources of interference into consideration as well.

Yet, we decided against its implementation for our purposes for the following reasons:

- Using all possible channels in the network topology creates to much interference for networks with more traffic. A selection process on this underlying topology is essential to our minds as selecting a few but high quality links leading to a planned and controlled topology will create a lot less interference than a network topology where all possible links are used for communications. This is especially the case if not enough channels are available for assignment, since these can be more effectively designed (See Chapter 5.2).
- Using one radio on each AP for a common overall-channel seemed to us as a misspending of radio modules. Radio modules on APs are scarce and should be utilized efficiently. Furthermore one overall-channel which is utilized by each AP does not scale well and is basically already done in *AutoWDS basic*.
- The ranking process, suggested to run on the APs, creates load on the APs. We want to use the APs merely as sensors and not as decision-makers, since this puts additional load on the APs and brings problems/complexity with synchronisation and acquiring the overall view on the network.
- The ranking algorithm itself could be improved. The solution averages values too early in the process and uses a random pick not as a last resort - providing supoptimal results.
- BFS's order of assigning channels puts one node (gateway-node) above all others during channel assignment. In some of our use cases there is no gateway-node available (See Section 3.1) and we need a fair distribution of channels, which is not feasible with this solution [31].

4.4 CTA

Subramanian et al. [33] use a modified Tabu-algorithm [15] in two steps to find a CA for the conflict graph. It works by starting with a random CA and, like an evolutionary algorithm, iteratively changing small details and selecting the best temporal solution to continue for the next iteration. An iteration is here further subdivided into generating a number of random neighboring assignments, where only the color / channel of one vertex is changed. From this set they select the assignment with the lowest interference. The tabu-list, which describes the forbidden options which are not allowed to be chosen for improvement, contains a list of vertex-colorings that already have been used. This procedure is then executed as long as there have been improvements in the last i_{max} rounds. If for i_{max} rounds no improvement can be found, the algorithm proceeds with step two. As the first step may give channel assignings which are not valid (more channels assigned than a node has radios), the second step merges channel assignments until the solution is valid again. Merging channels obviously increases the interference.

This solution is also topology preserving and a deal-breaker for our implementation (See Section 4.3). Additionally the evolutionary attribute does not provide controlled, but more heuristic results and we need an algorithm with more control over its internals.

5

Algorithmic Design

Even for a simple scenario like in Figure 5.1, where we have a lot of possible links between the modules, it is not obvious how to choose a low-interference network topology from this graph. A shortsighted solution may be trying to use every possible link, since this would decrease overall travel times for each packet and therefore reducing network workload and possible interferences as a packet is not longer in the network than it needs to be. The problem in doing so is the small number of channels we can assign to these links, even if we allow using multiple channels. We can only assign one channel to a module. Even trying to start out small and assigning just one channel to one module would consequently force us also to assign the same channel to all connected modules in order to maintain connectivity as two modules can only communicate with each other if they use the same channel (See upper half in Figure 5.2).

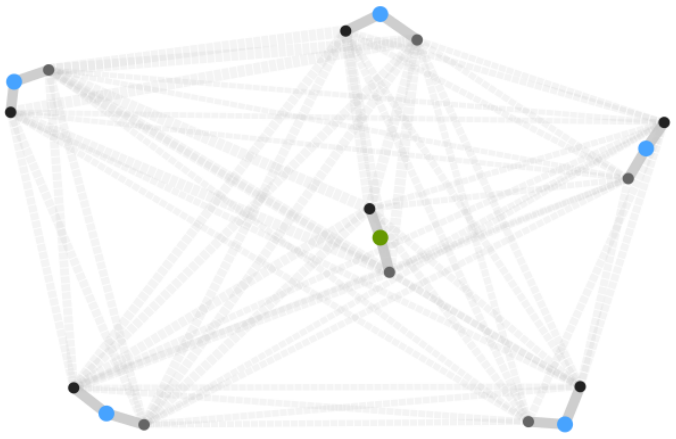


Figure 5.1 A network topology graph for a simple scenario with 6 APs with two modules each. The green node indicates a connection to the wired network.

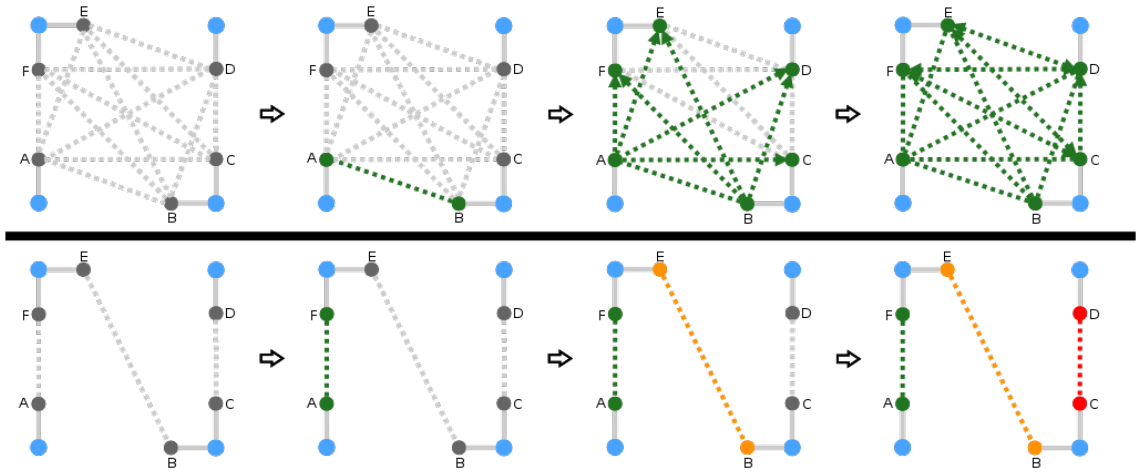


Figure 5.2 Coloring a highly connected network (upper) may result in fewer channels with high interference being used compared to a reduced network topology (lower) where multiple interference-free channels can be used. Especially the dependent modules, which also have to receive the same channel in order to maintain connectivity, are the reason for single channel usage here: Wanting to only color edge A, B you also have to assign the same channel to nodes C, D, E, F and therefore its edges.

On the other hand if we choose a small subset of links in this graph(like a Minimal Spanning Tree (MST)), we might be able to assign more channels, leading up to no interference at all (See Figure 5.2). As a drawback this could create bottlenecks, single points of failures and leading to packets staying longer in the communication network, since they can not take the shortest path. This is why we later need to add further links (survival paths).

A consequence of the mono-channel setup in a dense topology is a radical throughput performance loss as a transmission from A to B would prohibit any other communication due to the utilization of the same channel. Even if links (A, F), (E, B) and (D, C) would have a low link quality and packets can not travel directly to their destination a simultaneous transmission allows a higher throughput in the same time. Naturally only the second solution is favorable for a bigger scenario, because more APs would have to wait to put their data on the spectrum. Especially adding redundant links to a small subset of links in order to fulfill the reduced end-to-end link failure requirement is not straight-forward. Each newly added link increases overall throughput capacity, but at the same time may create co-channel interference or worse, decrease the number of overall assignable channels.

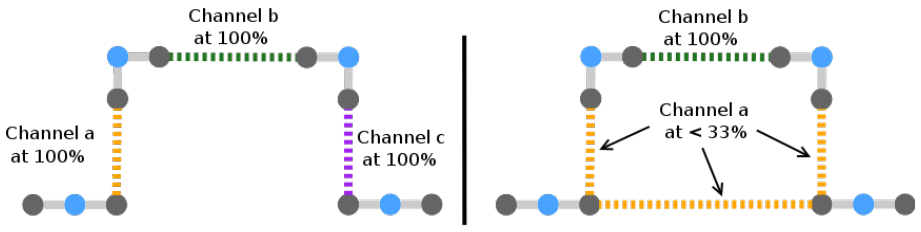


Figure 5.3 Additional links do not necessarily increase throughput performance rather on the contrary may decrease the number of non-interfering channels used (3 versus 2). Percentage depict link quality and therefore flow capacity.

In the following two sections we will present an algorithm creating network topologies, seeking to find the best compromise, by first creating the topology and then assigning channels on the resulting topology as basis.

5.1 Topology Creation

Creating the topology is also split into two parts. First, creating a MST, respecting a certain provided formula, which scores edges depending on their expected throughput. Second, adding redundancies in form of backup or parallel links to the MST. To make it more vivid we added an example after the algorithmic description.

5.1.1 Minimal Spanning Tree Topology

In order to select the edges, which we want to use for creating connections between the APs, we use a derivation of the DJP-algorithm [6, 28]. The reason we chose a MST algorithm over other techniques like all-pairs shortest path is, that a MST gives us the best essential edges in a graph. We can then extend these essential set of edges with further redundant edges if we have or want to (survival paths). Adding more than the essential edges immediately plays in favor of interference, which we want to avoid by all means and so have to do so carefully.

From the set of all nodes which are connected by edges, we start by selecting an arbitrary initial node and mark it as visited. All edges from this set of visited nodes to unvisited nodes are called productive edges. We keep those productive edges in a list sorted by their scores. The score of an edge is determined by the following formula:

$$ES = \frac{b}{(i+1) * (c+1)} \tag{5.1}$$

where ES is the score of this edge, b is the expected bandwidth, i is the number of interfering modules and c is the connected count of the corresponding nodes. Dividing b by i and c describes how the theoretically available bandwidth would have to be shared among other interfering radios. This characteristic results in a lower ES for edges where the assumed throughput is lower due to interference by other radios. The following three subsections will explain the formula in more detail.

For each round we determine from this list the edge with the highest score. If two links have the same ES and different SNRs, we pick the edge with the higher SNR as this edge has to share its channel among fewer participants and therefore results in higher throughput with fewer interference to the network. As a last resort if there is also a tie in SNR values, we pick one at random. We then mark the new node as visited and add also the new links to the list. Finally we have to update the scores for each affected link and continue with the next round until all nodes are marked visited. The outcome is a minimal spanning tree for this graph with a custom evaluation function.

5.1.1.1 Interfering Modules

Since the channel assignment has not taken place yet, we do not know which modules are actually interfering with this link if we would use it. However we do know to which other modules this module already has connections to and since those have to use the same channel in order to communicate with each other, we can derive an estimate as lower bound for the number of interfering modules for this link (A, B) by counting the following modules: Total interfering modules is equal to the number of visited nodes which we can reach by one hop over a module-module connection from node A and B. Those modules definitely interfere with our current connection, since they:

- are in range with at least one node of this connection (one hop distance)
- have to use the same channel (communication over module-module link)
- do actually interfere, because we already decided to use this connection (visited node)

The value is incremented by one, because if this connection would be used, itself again acts as a source for interference and it nicely solves the division by zero problem. With an increasing count in interfering modules, the value of the link decreases and vice versa. This reflects perfectly the concept of a shared medium.

5.1.1.2 Expected Bandwidth

The expected bandwidth describes the available bandwidth we assume to get for this link depending on the SNR. For a given SNR we can estimate the maximal possible throughput since effective bitrate and therefore throughput is amongst others dependent on the SNR and modulation which is continuously adjusted by the network driver for the radio module during operation.[1] The bandwidth was chosen as the numerator, since it is effectively shared by the radio modules within range.

As we can take from Figure 5.4, a higher SNR-value results in more bandwidth available to use and share and works in favor of this edge. Note that the mapping of SNR to achievable throughput may have to be adjusted for different hardware / devices / drivers. In general the behaviour will stay the same, as a higher SNR will necessarily result in a better modulation, which also increases throughput.

Another factor of influence would be the number of erroneous bytes that have been transmitted over this link. This is however not used for determining the expectet bandwidth as we do not know what those values are going to look like before actually using this link to transmit data.

[1] The rate adaptation algorithm which is used in LANCOM devices is Minstrel [32].

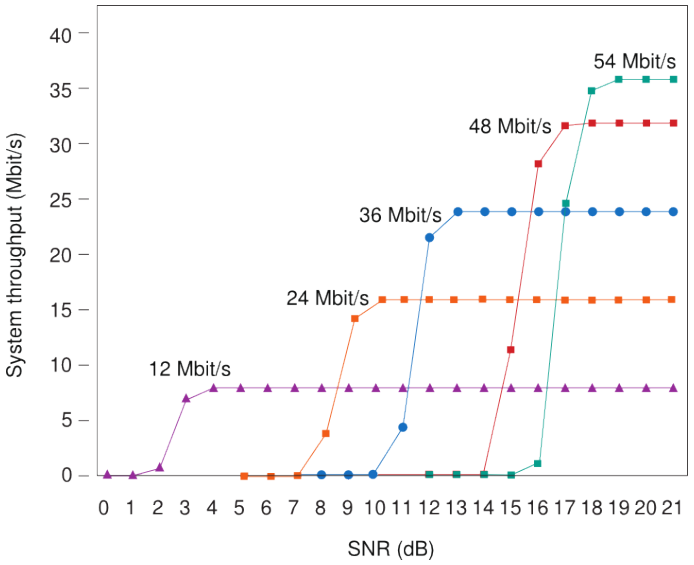


Figure 5.4 Expected throughput SNR-values by Otsuki et al. [24]. We assume to get the best possible throughput for a given SNR, since modulation is automatically adapted by the APs with increasing SNR.

5.1.1.3 Connected Count

The number of nodes we can reach from module A and module B by just using module-module edges (For example nodes A, B, C in Figure 5.13). A higher connected count diminishes the importance of this edge, since this value controls how many channels we can use later for the overall graph. If this value had not been taken into consideration for calculating the score, the algorithm would rather create long chains of connected modules. Those links in this chain would admittedly have the best SNR values, but since they have to share the same channel, it would result in lowering the overall throughput. Granting this value a higher impact in the formula, like for example in

$$ES = \frac{b}{(i+1) * (c+1)^2} \tag{5.2}$$

would have the effect of overemphasizing the module connectedness and lead to poor choices in links with respect to SNR - lowering overall throughput again. The simple, linear impact in the formula has shown to achieve the best tradeoff between those two extremes.

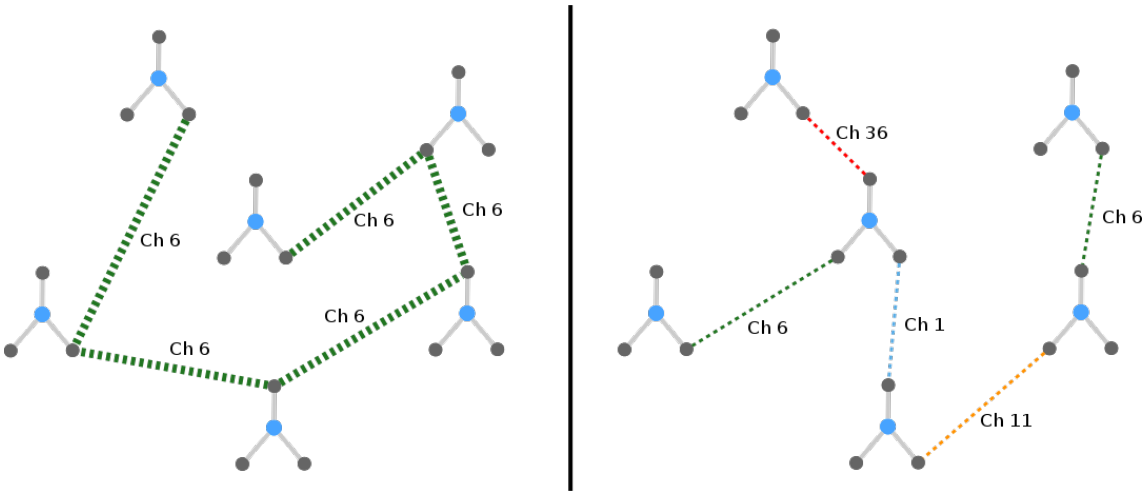


Figure 5.5 Impact of the connected count on the formula. Resulting topology in case the connected count is underemphasized (left), leading to long chains of possibly high quality links, but only one usable channel, which has to be shared among all modules. Topology in case of an overemphasized connected count (right). As the algorithm avoids reutilizing already used modules, links with low quality are preferred, leading a lot of exclusive channels with lower link quality. Thicker edges illustrate better link quality.

5.1.1.4 Example MST Creation

Let's take a look at a simplified example, where we assume b=SNR. Also note that we immediately expand module-device edges to keep the example short. Normally the algorithm would also expand each of those edges step by step. However due to their maximum weight these have precedence over all other edges, so that we merely skip the tedious images.

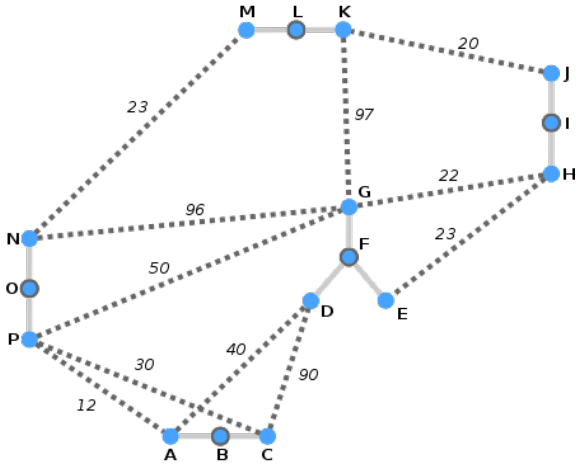
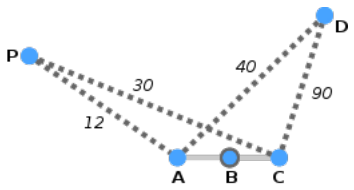


Figure 5.6 Example input topology - Edge-weights representing the SNR

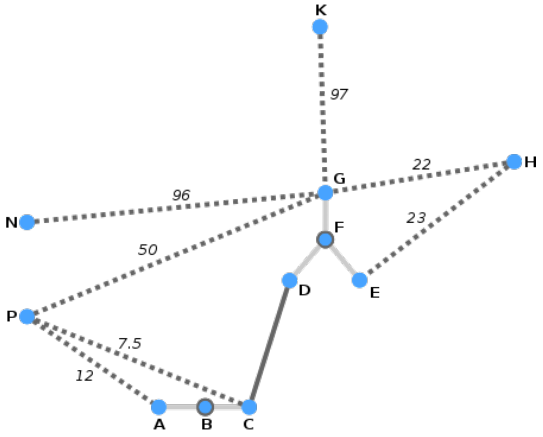
Figure 5.6 represents the underlying network topology with all possible links for a scenario with five APs where all APs are equipped with two radios except F, which has three.



Edge	*b*	*i*	*c*	*ES*
(A,D)	40	0	0	40
(C,D)	**90**	**0**	**0**	**90**
(A,P)	12	0	0	12
(C,D)	30	0	0	30

Figure 5.7 First round - Picking (C, D) with highest score

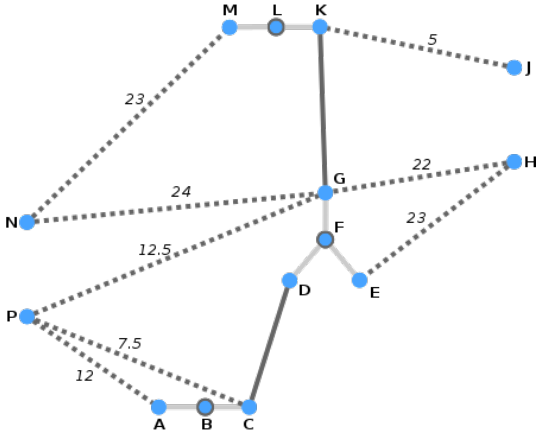
We start off by selecting a random node (B). Actually the first steps here would have been expanding the fake edges (B, A) and (B, C), but as noted above we skip those. From this set of nodes we then consider all edges to newly, up to now undiscovered nodes (listed in the right table in Figure 5.7). From this list of edges we calculate the *ES* on each edge, which results in the selection of edge (C, D). Note, from now on the values on the edges represent the *ES* instead of *b*.



Edge	*b*	*i*	*c*	*ES*
(A,P)	12	0	0	12
(C,P)	30	1	1	7.5
(G,P)	50	0	0	50
(G,N)	96	0	0	96
(G,K)	**97**	**0**	**0**	**97**
(G,H)	22	0	0	22
(E,H)	23	0	0	23

Figure 5.8 Second round - Score for edge (C,P) decreased - Picking edge (G, K)

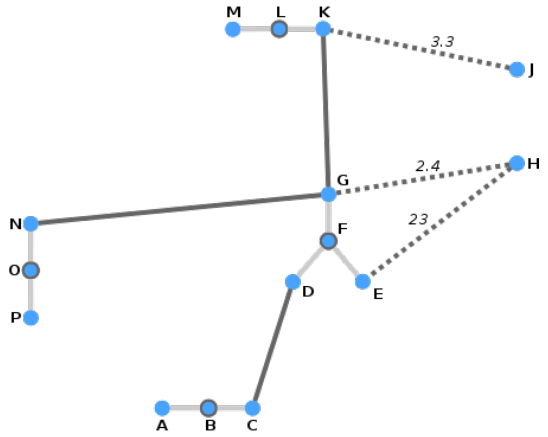
After the expansion of the fake edges for AP F, we proceed by refreshing all the edgescores, portraying edge (G, K) as the best edge (See Figure 5.8). Note how the score for edge (C, P) was decreased, since node C is already used for another edge.



Edge	*b*	*i*	*c*	*ES*
(A,P)	12	0	0	12
(C,P)	30	1	1	7.5
(G,P)	50	1	1	12.5
(G,N)	**96**	**1**	**1**	**24**
(G,H)	22	1	1	5.5
(E,H)	23	0	0	23
(K,J)	20	1	1	5
(M,N)	23	0	0	23

Figure 5.9 Third round - Best edge is (G,N)

All edges connected to G have been recalculated as G was used in the last step. Edge (G, N) is nevertheless the edge which promises according to the formula the most throughput, even more than the up to now unused edge (E, H).



Edge	*b*	*i*	*c*	*ES*
(E,H)	**23**	**0**	**0**	**23**
(G,H)	22	2	2	2.4
(K,J)	20	1	2	3.3

Figure 5.10 Fourth and final round - Best edge to connect the last nodes is (E,H)

Edgescores of the remaining links rapidly decrease as using these would definitely cause interference. This is the first step the connected count c has a greater impact in calculating the *ES* as the path N, G, K has length two. Using edge (G, H) would increase this distance to three, which would be unfavourable for the coloring process (all edges connected to G would have to use the same channel). Luckily we can utilize (E, H) which does not necessarily interfere with other links (it may interfere after the coloring process, if not enough colors are available).

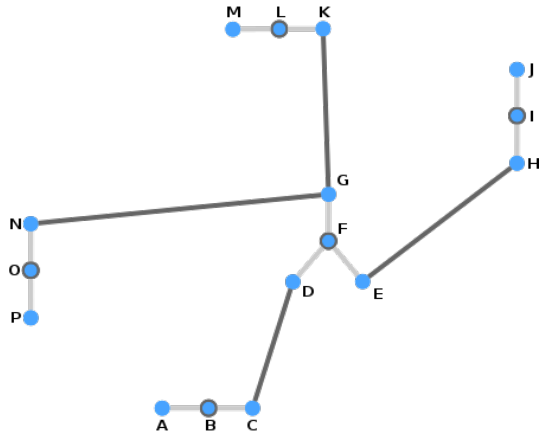


Figure 5.11 Resulting MST

5.1.2 Survival Paths

A spanning tree is particularly susceptible to graph partition by just removing one link so we have to add redundancies in form of supplementary connections. However those redundant connections have to be carefully chosen in order not to negatively impact the successive channel assignment and therefore to push interference. Note that this is an optional feature, since for some use cases a spanning tree topology is enough or a topology with redundant links is not easily implemented with the existing code or hardware.

We accomplish this by iterating over all the edges of the spanning tree and simulate each connection failing. We then check if there still exists a path from node A to node B of this failing connection. Only if this failing connection cuts the graph in half and there is no path to the other side, we start looking for a backup route in the following fashion: First we separate the graph into two groups: group A with all nodes and links reachable from node A and group B with the same for node B. We create a list with unused edges which connect the two groups and calculate the scores on them and pick again the edge with the highest score. This edge is the best edge to reconnect the two parts (according to Formula 5.1) and is called the survival edge for this failing scenario. Nevertheless we have to be aware that it might not be feasible to reconnect those parts if the underlying structure does not permit it. For example the link between nodes E and F in Figure 5.12 might be the only connection possible and failing it would cause network disruption. But if this link fails there is nothing we can do. The result is a robust network topology, which despite the added interfering links, still yields a high overall throughput with redundant paths.

The survival path attribute for a graph can also be expressed in the following formal way:

> For each edge (a, b) of the calculated spanning tree graph G', find a path from a to b without traversing (a, b), in a way that the sum of the Edgescores of the path is maximal.

Formally defined as:

$$G = (V, E) \quad \forall edge_{a,b} \exists path_{a,b} | edge_{a,b} \notin path_{a,b} \quad a,b \in V, edge_{a,b} \in E$$

Finding the optimal edge:

$$max(edgescore(path_{a,b})) | edgescore(path_{a,b})) = \sum_{e \in path_{a,b}} edgescore(e)$$

Using survival path actively and additionally to the MST links, instead of a fallback, would require a shortest path routing that deals with circles in the graph.

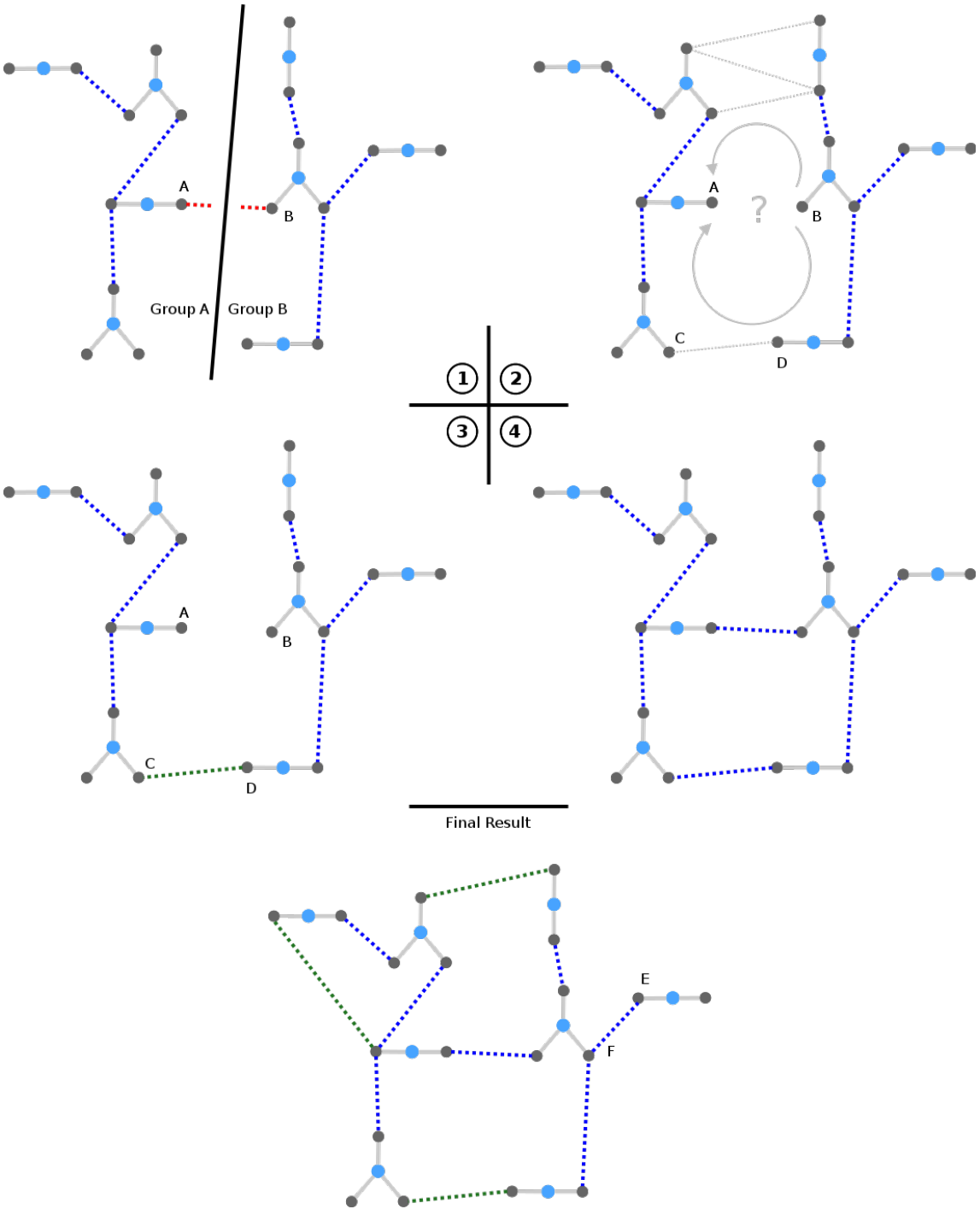


Figure 5.12 Finding survival paths. Simulation of connection between node A and node B failing, creating two groups. Finding a survival path for node A and B. Eventually all connections have a survival path. No redundant link possible for failing connection (E, F).

Channel	Interference Count
36	4
11	**1**
1	**1**

Channel	Interference Count
36	**4**
11	5
1	5

Table 5.1 Channel-list without (left) and with (right) foreign influence

5.2 Channel Assignment Algorithm

For a given network topology graph and a set of channels to choose from, we can now assign channels to the module-nodes or if you like the links between those. Therefore we iterate over all module-module edges and assign channels to the adjacent module-nodes if they do not already have one assigned. The channel for an edge is chosen by the following pattern for each channel-group: Select all the modules which are connected by module-module edges. This set of nodes is called the channel-group. For this channel group we create a list of channels and corresponding interference occurrences. Among these we pick those channels from the set of all possible channels, which have been used the least in this channel-list. If there is a tie in usage we can additionally respect foreign networks by taking the occurrences of foreign radio modules also into account. Since we do not know how much traffic the foreign channels carry and therefore how much they utilize the channels, we can only weight them equally compared to our own channel-usage instead of a more fine-grained subdivision. As a further tie-breaker we pick those channels which have been used the least in general. At last we resort to just selecting one channel at random, but this should rarely occur. Especially respecting foreign networks allows use to evade heavily used bands and channels like for example 1 and 11 in the 2.4 GHz band which are used by devices out of our control.

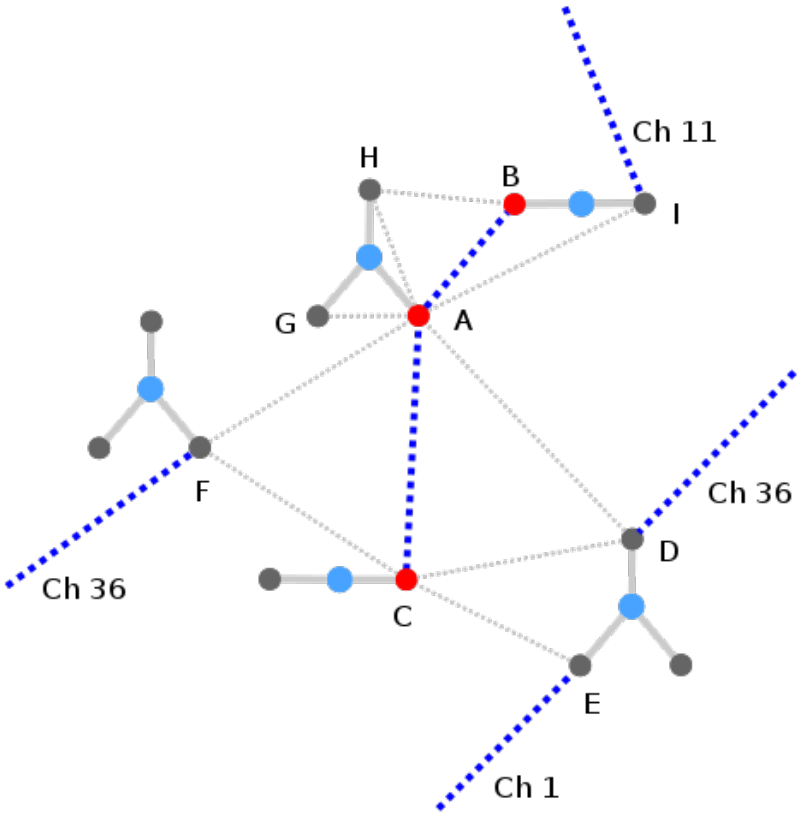


Figure 5.13 Channelset A, B, C selected for coloring.

Since A, B and C are in the receive range of D, E, F, G, H, I they are possible sources of interference if we use the same channel as those. Therefore we survey how often every channel interferes, where one module can also interfere multiple times. For example does module F which already has channel 36 assigned interfere 4 times with the channel-set A,B,C ((A, F), (C, F), (C, D), (A, D)). If we only could choose from channel 1,11 and 36 we would then decide to use channel 1 or 11 to assign to the channel-set depending on how often we overall used those already. If however we want to take also foreign networks into consideration and lets say modules A and C would both detect two other WLAN networks at channel 11 and two at channel 1, then we would add another four for channel 11 and channel 1 in the channel-list, leading to Channel 36 as the best choice. Note modules G and H are ignored for the counting process since they do not have links or channels assigned.

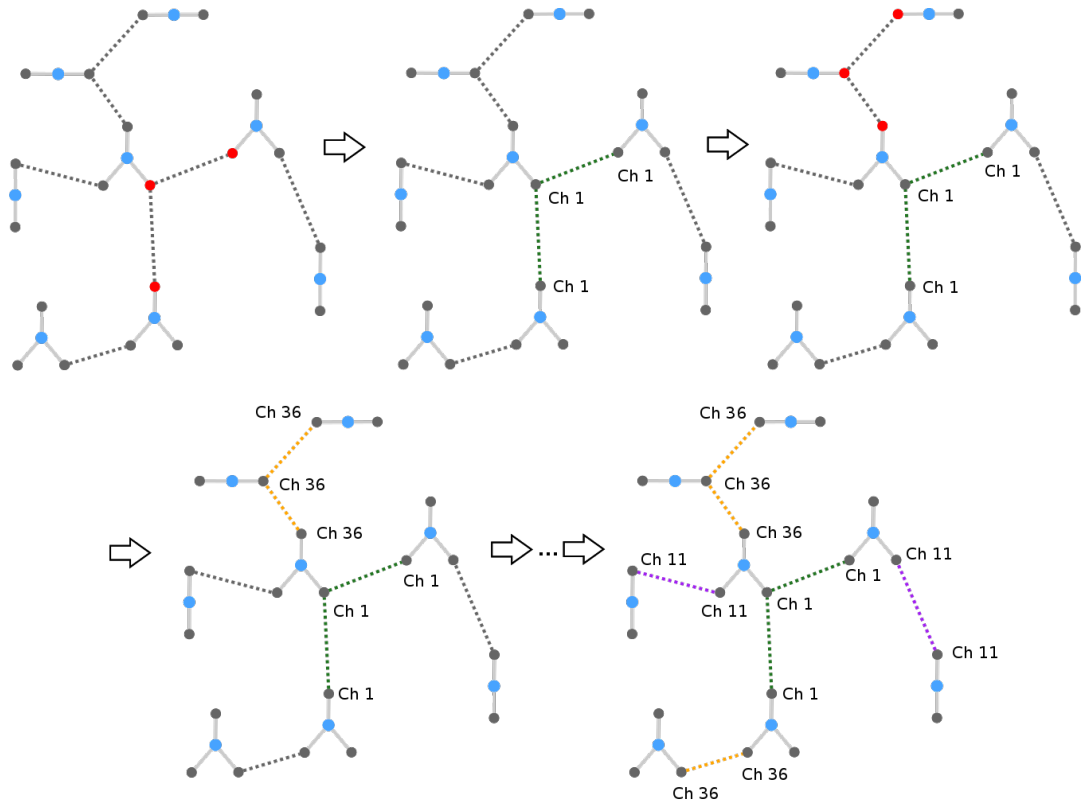


Figure 5.14 General overview on the channel assignment process.

Figure 5.14 shows an example-assignment for a network with 8 APs with 2 and 3 modules equipped and available channels [1, 11, 36] without foreign influence. Each graph depicts the status after each of the assignments. Red marked nodes represent a channel-group. Currently there is no special order in which the channel-sets are colored.

6

Implementation

We implemented our optimization as an addition to *AutoWDS basic*, after it successfully created a stable network tree topology (See Chapter 2.1.3.3). The additional steps in the discovery process of nearby radios are:

1. The APs enable the background scan. This means the APs cycle through all possible channels for a few milliseconds and listen for other broadcasted wireless networks and store the recognized mac addresses and signal strength in a corresponding table. Note the APs regularly switch to the background-scan-mode to discover other APs.
2. APs send scan results continuously to the central WLC.
3. When all the scan results are in, we read the scan-results table from the WLC to our client station and parse the data to a networkX graph.
4. On the client station we compute an optimized network topology for a given set of input parameters set by the administrator.
5. Write back the data from the client station to the WLC to the configure-topology-table.
6. The WLC derives configurations in Type-Lenght-Value (TLV)-format (CAPWAP) and sends these to the APs.
7. The APs receive the configurations and reconfigure themselves to establish the new network links.
8. After a while all the APs solely use the newly configured connections and announce the network to other stations and clients.

Later eventually steps three to five will be also incorporated in the WLC to make the process optimization more convenient.

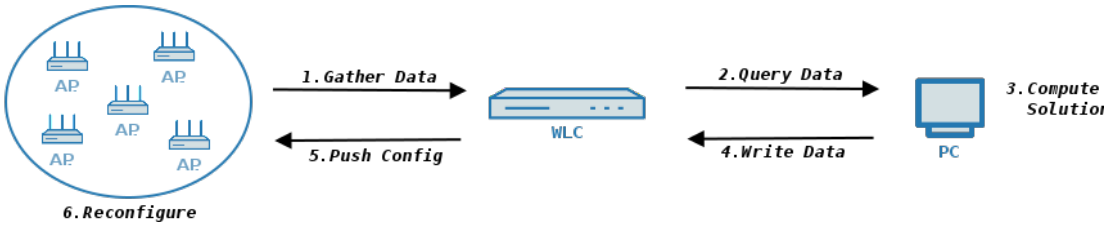


Figure 6.1 General flow of information in *AutoWDS extended*

6.1 Language of Choice

The language we used for the implementation is Python 2.7. Although we also could have implemented the extension in c/c++ since the basic version was already written in this language, python does have a certain key advantages, we did not want to miss and want to list here.

Rapid prototyping: Since our algorithms can easily work on a given set of data independently and the interfaces to get the data from are well defined and accessible, using a lightweight language like python had the benefit of not having to embed our new code into the existing environment. Although it still can be implemented in the native language C/C++ for our scenario, this will take more time and effort, which was not the focus of this work.

Existing libraries: The NetworkX library in python made it particularly easy for us to map real world scenarios to graph abstractions as we did not have to create those structures ourselves. Although there are some libraries that provide graph abstractions in C/C++ we could not have as easily employed them due to their copyright restrictions and the closed source virtue of our target system.

Faster Debugging: Especially the initial phase of the project was error-prone and needed a lot of debugging. Secluding our algorithm from LCOS brought the advantage of not having to undergo the lengthy process of compiling a whole new LCOS firmware, every time we made a fix in our code. Additionally not having to flash the firmware then into the WLC was a great time saver as keeping the WLC running and consequently the APs connected, allowed us to independently modify our algorithms without having to wait until the network was back up again. Not to mention the live debugging capabilities of python, which was really helpful and made debugging easy.

Existing APIs: Fortunately also the code for interaction with the interfaces (Simple Network Management Protocol (SNMP) / Secure SHell (SSH) / Telnet) of the WLC was written in Python, so we could easily integrate it into ours and had no problem gathering the 'seen'-data.

Maintainability: As python has the reputation of being an easy to learn language, this was another benefit and played absolutely in our favor with the goal of keeping the code simple, lightweight and reusable for someone else.

6.2 Dependencies

As already mentioned we used NetworkX [14] to model our graphs and work with its elements. NetworkX's rich features like "has_path" made things especially easy in computing the survival paths. Another library we used is python collections [12]. This library implements specialized container data types additionally to those shipped with standard python. It's 'Counter' structure is used to create the channel-lists and retrieve the most or least used elements. We extended this library for our convenience since it did only provide the functionality for getting the most common elements from its lists and not the least common, which we needed[1]. The only closed source library we used is the herein before mentioned library to query the WLC for the data of the APs. Nevertheless due to its separation into modules, gathering the input-data can be carried out by other libraries or custom scripts.

6.3 Structure of the Code

We split the code into two modules. One is handling the gathering of input data, followed by parsing it and finally dealing with its output data. This includes sending/receiving to and from the WLC, transforming data to and from networkX graphs, checking configurations for validity and other facility work, like exporting data structures for debugging or visualization to a JavaScript Object Notation (JSON) file. The other module exclusively works on and with graphs and therefore is the main and interesting one to solve the theoretical problem.

6.3.1 Interfacing Outer World and Managing Data

The name of the python module for this job is *wlc_com*. Its responsibilities include the following:

Receiving the necessary data: For this purpose we use the LANCOM internal source code-python library 'testcore', which creates a SSH / Telnet connection to the WLC and parses its table data to local data structures. The format we find on the WLC is depicted in Figure 6.1. This table is filled on a regular interval by the APs. Here each line represents an AP (identified by its device mac address) and its radio-module (identified by its module mac address) receiving the beacon of another radio module (seen module mac address) and a signal strength indicator in form of SNR. Basically this shows us which radio module sees which other radio module and how good.

[1]The request for including this method was already issued at 01/2013 http://bugs.python.org/issue16994

Device Mac Address	Module Mac Address	Seen Module Mac Address	SNR
ece55574a4d5	ece555ffd61e	ece555ffd5bf	56
ece55574a4d5	ece555ffd6cc	ece555ffd5b9	48
ece55574a4a5	ece555ffd667	ece555ffd5d6	33
...	...	...	...

Table 6.1 Table representation of a mesh network topology on a LANCOM WLC

Transform the data to a networkX graph: The table data from above can then be molded into a directed, weighted networkX graph. Where we create a node for each mac address and module mac address and attach a boolean attribute to it to tell them apart. In parallel we add two types of edges:

- Fake edges between module-nodes and device-nodes with an attribute "SNR" which contains the maximum possible value for this attribute.
- Real edges for each line in the table above, where the two nodes are the module mac address pairs. Note that one row in the table represents just a directed edge from one module to the other. We only respect two-sided connections, since one-sided edges are due to their poor quality not worth considering and would further complicate finding a solution. As we have two SNR-values for a single undirected edge when merging two directed edges, we are at liberty to chose. Currently we are using the average of both values, but implemented an option to use the larger or the smaller value if favoured. Especially the lower SNR value might better represent a connection for more pessimistic scenarios.

Conducting a validity check on the result: Before generating the configuration and sending the result to the WLC we have the option to perform sanity checks on the outcome. Among others we test for the following:

- Is each device connected to the network, or are there disconnected components?
- Do modules which are supposed to establish a connection use the same channel and band?
- Are only channels used which have been specified as input?
- Are all devices only connected to their modules over module-device edges?

Send results back to the WLC: If we were able to compute a solution and all checks passed, we again establish a connection to the WLC and write back a network topology and channel assignment configuration. This configuration will then be enforced on the APs by the managing WLC. This is done by configuring each module to use only the assigned channel and set entries in the network topology table. Depending on other parameters, the APs will then after a predefined time frame receive their specific configurations and act on those. APs do not immediately switch to the new configuration as this could cut branches of the network too early - possibly leaving APs unconfigured and unable to rejoin the network (See Figure 3.1).

6.3.2 Solving the Theoretical Problem

The module for this job is named *tcca* and handles finding the solution based on graphs. With a given undirected networkX graph we can compute a solution in three phases analogous to the algorithmic design. First we create a minimal spanning tree, followed by computing the survival paths and finally assigning channels to the resulting network topology.

6.3.2.1 Creating the Minimal Spanning Tree

In the first phase we take the networkX graph as input, either randomly created or received from the *wlc_com*-module, and call the "calculate_st"-function. This function starts by randomly selecting a start-node and putting this node into the visited-set. At the same time it also creates a counter-list called edge-list with all the edges to neighbors of the initial nodes the calculated edge score as their corresponding values.

After this initialization it enters the main loop by removing all the elements in the edge-list, which do not lead to new, up to now unseen nodes. We call those nodes unproductive. To escape the loop at a certain point, it is then checked if the edge-list is now empty to leave the loop, or if there still are edges leading to new nodes. The most of the time the loop spends with the following:

- Selecting the edge with the highest score from the edge-list.
- Expanding the edge. This includes adding it to the MST, adding the appropriate nodes to the visited-set and adding all productive edges with their edge-scores to the edge-list.
- Deleting the used edges to speed up updating the edge-scores.

In order to calculate the score of an edge this main function makes use of another function "calculate_score_for_edge". Here basically all the necessary values for the scoring-formula (See Formula 5.1) are gathered and the edge-score is calculated.

After successful execution of the loop, a minimal spanning tree in form of a NetworkX graph is returned to the caller.

6.3.2.2 Calculating the Survival paths

With the MST graph as basis we are now able to add redundant paths. As you might have already guessed the function's name for this job is "calculate_survival_links". To simulate each edge failing, this function iterates over all the edges in the MST and removes these edges from the MST. This sub-MST is then used to check if it already contains an alternative path from and to the nodes of the failing edge. Luckily NetworkX already provides such a test with "has_path". Only if this test fails we continue by creating our two groups A and B for each side of the split graph. The most costly part is then iterating again over all edges in the underlying graph and checking if this edge would reunite the halves of the graph and determining their scores. Finally we pick the best edge from the counter-list and add it together with its failing edge back to the MST.

6.3.2.3 Coloring the Edges

For the last phase of assigning channels to the edges/modules we use the "calculate_ca"-function. This one uses either the MST graph or the survival-graph as input additionally to the necessary underlying network graph and a list of allowed-channels. To start we initialize the overall-channel-counter with zeros which tracks overall channel usage in case there are ties. We iterate again over all edges in the graph and skip over all edges, that are not module-module edges or are already colored. For each uncolored edge we then determine the channel-group by running a BFS-search over all connected modules. As required by the design we now need to estimate the local interferences. This is done by the function "count_local_interference" which returns two counters-lists: internal- and external- interference. Those contain channels with a number of interference clashes as their values, which means how often the use of a certain channel would interfere with other modules in the area of the channel-group. Naturally we continue by choosing the channel which has the lowest value in this counter, as this channel or channels would interfere the least with its environment. If the outcome is not unique, we enter the tie-breaker section by extending the choosing process to the overall-channel-counter and at last resort to picking one channel at random from the final candidates. Finally the designated channel is assigned to all the elements of the channel group. Just as the others, this function returns in the end a NetworkX graph, but now with channels assigned to the edges and modules of the allowed-channel-set.

6.4 Example

To give you an impression on how to work with the code, we illustrate the process of finding a solution by taking a look at a simplified example. See the code snippet in Table 6.2. Therefore we create a grid network with random edge weights (line 5 - 41). Note that such a deployment is without doubt suboptimal as the APs are placed too close to each other.

To get the initial MST from this graph we call the "calculate_st" function on it (line 43 / left side in Figure 6.3).

Adding the redundant paths works just as easily by using the "calculate_survival_links" function (line 45 / right side in Figure 6.3).

Finally we want to assign channels / colors to the edges by utilizing "calculate_ca" on the two graphs (lines 47, 49 and Figure 6.4).

```
import networkx as nx
import random
import tcca

graph = nx.Graph()

for i in range(9):
    i = str(i)
    graph.add_node(i, isModule=False)
    for j in range(2):
    j = str(j)
    module_name = i + "." + j
    graph.add_node(module_name, isModule=True)
    graph.add_edge(i, module_name)
    graph.edge[i][module_name]["snr"] = max_weight = 1000

seen_links = {0: [0, 1, 3, 4],
      1: [0, 1, 2, 3, 4, 5],
      2: [1, 2, 4, 5],
      3: [0, 1, 3, 4, 6, 7],
      4: [0, 1, 2, 3, 4, 5, 6, 7, 8],
      5: [1, 2, 4, 5, 7, 8],
      6: [3, 4, 6, 7],
      7: [3, 4, 5, 6, 7, 8],
      8: [4, 5, 7, 8]}

for node_a in seen_links:
    for node_b in seen_links[node_a]:
    for i in range(2):
        i = str(i)
        for j in range(2):
        j = str(j)
        node_a = str(node_a)
        node_b = str(node_b)
        moda = node_a + "." + i
        modb = node_b + "." + j

        if moda == modb:
            continue

        graph.add_edge(moda, modb, snr=random.randint(30, 96))

st_graph = tcca.calculate_st(graph)

survival_graph = tcca.calculate_survival_links(st_graph, graph)

ca_st_graph = tcca.calculate_ca(st_graph, graph, [1, 6, 11])

ca_survival_graph = tcca.calculate_ca(survival_graph, graph, [1, 6, 11])
```

Table 6.2 The python code for generating the example network graph and solutions. Note the tcca import, which is our library for topology creation and channel assignment.

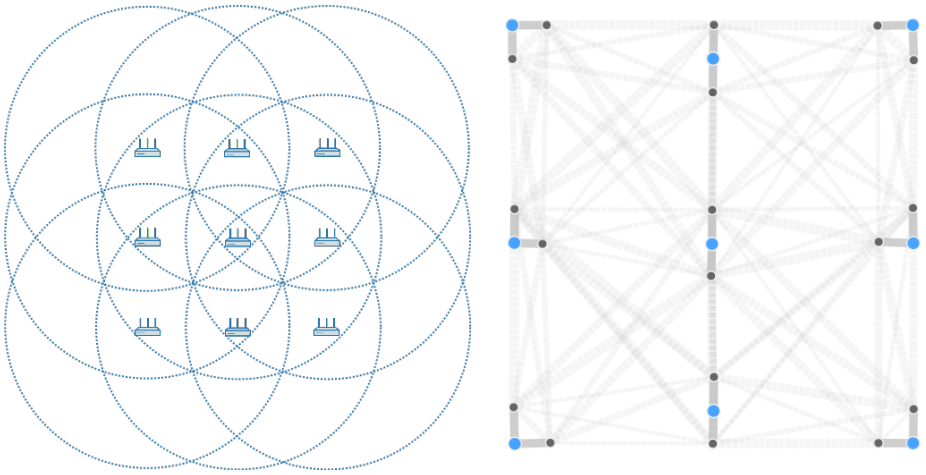


Figure 6.2 Grid layout with two radio modules per AP (left). Resulting network topology (right). Thicker edges indicate higher link-SNR.

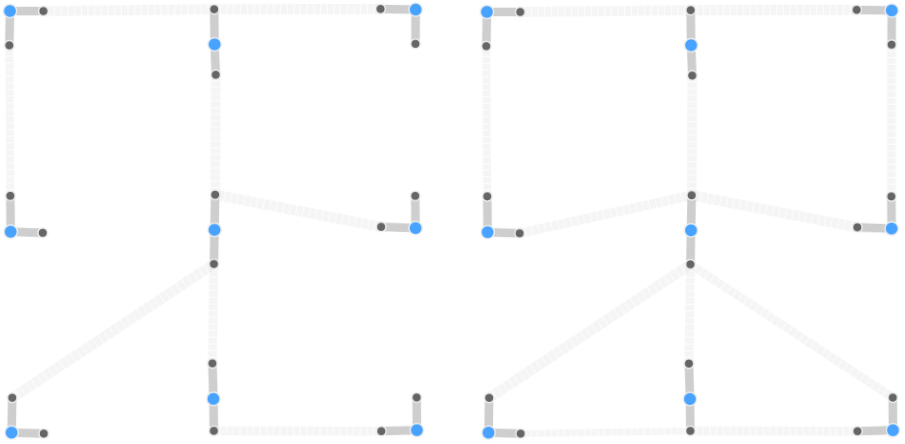


Figure 6.3 Spanning tree on the graph (left) and spanning tree with survival paths (right)

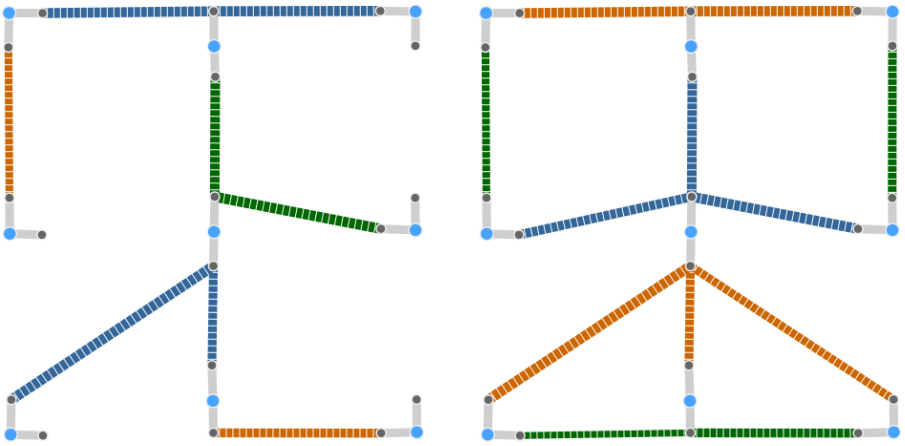


Figure 6.4 Spanning tree with channels assigned (left) and spanning tree with survival paths with channels assigned (right)

6.5 Problems and Aids

***AutoWDS* Status Tool:** To make things easier for debugging and also to actually see what is going on during the execution of the algorithms, we created a tool called *AutoWDS* status, which is based on D3.js, a JavaScript library for visualizing data [36]. Furthermore attached with a tool to extract data from the WLC, it is able to display current statuses and established links of a real world, deployed network.

It works by reading a graph represented as a JSON-file and displays it interactively in html with JavaScript. As it is basically a force directed graph, the nodes position themselves according to their SNR-edge-values. Additionally you can drag and position single nodes freely anywhere on the map for better singularization of entities. In the tcca module we also included an export function called "write_json(graph, filename)" which creates a json file to be read by the autowds status tool. This tool has also been used to create most of the images in this thesis.

Representation of Tables in LANCOM Devices: While writing the code which deals with the WLC and receives the data, we initially had difficulties extracting the data out of the tables represented in the WLC since they reside there in a somewhat obscure format and are spread over different tables. So that we effectively had to gather the data and join them so that we receive a nice table like Figure 6.1 with which we can work with. Nevertheless we were able to cope with it, although the code for it is somewhat longer and cumbersome.

Python Logging Module: Another great helper turned out to be the utilization of pythons logger module. We excessively made use of its Error / Warning / Info / Debug statements which made debugging a lot easier without being an obstacle if we did not need the output as we could just switch to a different level of logging and enjoy the silence.

Bitbucket: This was another great help in getting things organized. Especially git [19] as a version control system and the issue-tracking system one can find in Bitbucket [4] gave us great control and overview on the project without being a hassle to administrate. If you have or want to further work on this project we can only recommend getting access to our Bitbucket repository, because there you find additional notes on every issue and basically see why things went the way they did.

PyCharm: For our python coding we used the Integrated Development Environment (IDE) PyCharm [18] and were more than pleased with its speed and debugging possibilities. Its refactoring methods made changing the code easy and comfortable. Particularly the code analysis tool that comes with it made sticking to the Python Enhancement Proposal 8 (PEP8) [2] style-guideline almost fun and we hope it helps you to better understand our code.

[2] See http://legacy.python.org/dev/peps/pep-0008/

Proof of Concept: We knew from the beginning that a full and complete integration into the existing environment (LCOS / WLC) was not feasible in the scheduled time frame. This is why we aimed at a quick and dirty, proof of concept-like solution which could be easily done with python on a separate machine.

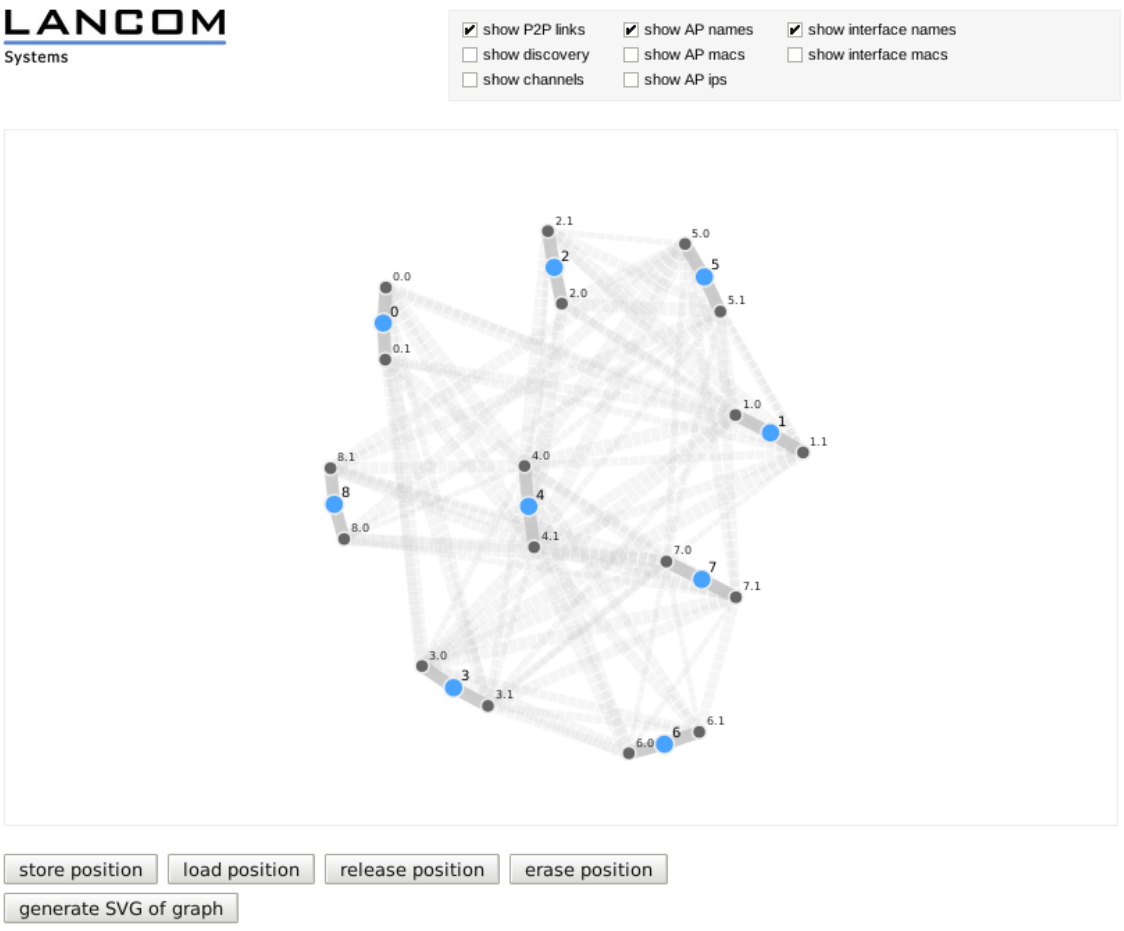


Figure 6.5 *AutoWDS* status web interface showing a network topology

7

Evaluation

We followed Pol, Koomen and Spillner's way of testing [27] by using their model and dividing the process of testing into the six phases planning, preparation, test-specification, execution and result analysis. Due to extent this work is aimed at, it was not feasible to implement all the systems of the related solutions on this field on a common environment to get a fair assessment for each work and to really estimate how each solution performs against the others. On the other hand such a comparison would have to be treated with caution since, each solution is designed with another metric/goal kept in mind. Si et al. [31] provide an overview on the current well-known solutions on CA and may serve for a further review and comparison. Therefore the following tests will focus on the comparison between *AutoWDS basic* and its extended, optimized version. Furthermore our tests were restricted to the evaluation of spanning tree topologies, because *AutoWDS* currently still uses a bridged network for its APs instead of a proper routing system, with the drawback of Spanning Tree Protocol (STP) being active.[1] This means even if we would try to run a network topology with survival paths in it on *AutoWDS*, STP would disable some links in order to eliminate circles in the topology.[2] As a consequence achievable throughput will be lower than it could be with survival paths.

[1] STP is a temporary measure since it does not scale well. Our survival paths are supposed to lay the ground for redundancy-systems in Layer 2 (like FabricPath [34]).

[2] Circles in a network topology are undesirable since they can lead to broadcast-storms see IEEE 802.1D

7.1 Planning

The goal of this test is to confirm the claim of an increase in overall throughput of *AutoWDS extended* compared to *AutoWDS basic*. Therefore we will evaluate the whole process by deploying an actual WDS with *AutoWDS* instead of just performing a code analysis or review. To discern performance gains we run *AutoWDS basic* and extended on the same networking setup and measure its achieving overall throughput. In order to create enough throughput we attach traffic generators to the APs to simulate real clients. These generators will then send data to all the other nodes in the network. To get the values of how much data was actually carried over the wireless network, we will query the APs for this information in a way that it does not affect the measurement, like querying too often for too much data would cause additional load on the APs and the network as a whole.

The scope of the test is kept to the same scale as portrayed in the requirements analysis. That means it will not exceed 100 APs with expected target size of about 30 devices. We would expect to recognise noticeable differences between the two systems, since interference was strongly avoided where possible and radio modules are more efficiently used than before.

We would expect a significant increase of data that has been able to transmit due to the usage of multiple collision domains. The increase should be the same one could expect by comparing a network hub, which has essentially just one collision domain, and a network switch, which has multiple domains and does not have to wait until the medium is free for sending again. The second source for an increased throughput would be the increased number of packets that could be received without errors due to a lessened interference.

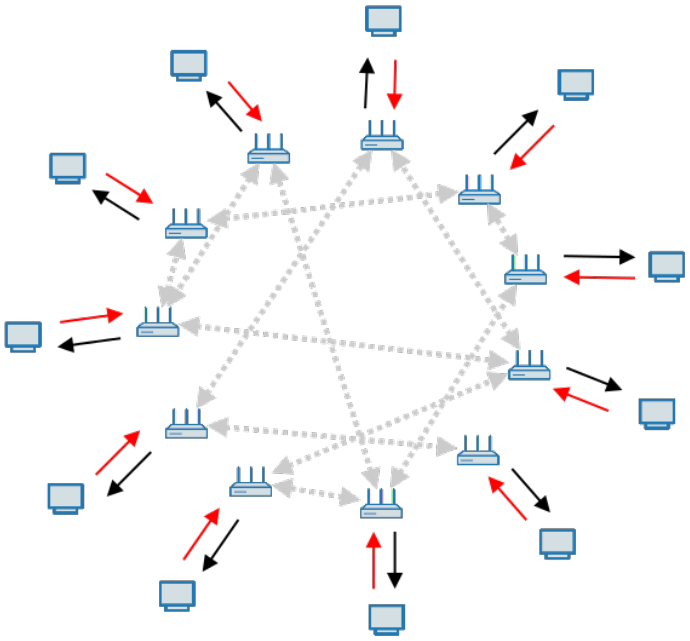


Figure 7.1 Concept of testbed with traffic-generating systems attached to APs by cable, sending broadcast traffic and saturating the network.

7.2 Preparation

The basic approach for measuring the throughput is to attach traffic generators to the APs and make those send as much broadcast traffic to all other stations which are part of this network and therefore saturate the throughput capabilities of the network. The idea behind this setup is to simulate real end-device-clients attached to the APs, where each client has its own ip address and therefore mac- address and table. Cabling the clients to the AP instead of using real mobile devices like smartphones was preferred as this gave us more control to keep out other impacting effects. Moreover, moving around 3 Devices per AP would have been a challenge, especially to do it in the same way for each test-run to keep circumstances reproducible.

Due to limitations in cabling for power and network and severe interference in certain parts of our building we could not use the full 30 APs as initially planned. Therefore we had to consider ourselves satisfied with just 12 APs.

7.2.1 Modifications for Practical Testing

Although we did not have to modify the code of our algorithms for the network, we had to modify the LCOS firmware to better fit the data we needed as input for the algorithms to our requirements. Therefore we took the latest stable firmware version 9.0 and added the following extension:

Addition of the Seen-Table: Although most of the data we needed did already exist in the current LCOS-firmware, it was spread over numerous places and retrieving the necessary dates was tedious and performance-costly. That is why we decided to gather everything necessary in one special table (See Figure 6.1).

Reduced Beacon Rate: Due to the close proximity of the APs in our testbed we had to reduce the amount of beacons which were sent by the APs to decrease their influence in the throughput performance test. The default rate with one beacon per 100ms was especially for the mono-channel setup too much as those beacons were received by all 12 other APs. The beacon rate for the test-setup was therefore increased to one beacon per 800ms.

Default Point-to-Point-Scanning Mode Deactivated: Per default the APs on a regular basis switch into a scanning mode, where they iterate over all available channels in milliseconds and listen for other radios. During this time frame the AP can not receive or transmit any data, which would have a drastic dramatic impact on the throughput-tests. This is why we deactivated the scanning mode for the duration of the throughput-test, after the network topology has reached a stable state.

7.2.2 Physical Structure

We deployed the APs at a typical office environment as depicted in Figure 7.2 and used only omnidirectional antennas for the APs. Additionally the APs were also connected by wire to their client systems on a central server in form of virtual machines.

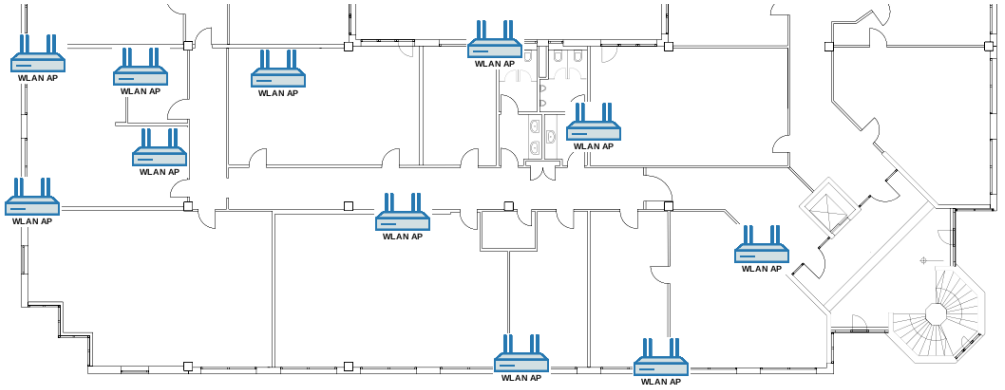


Figure 7.2 Physical arrangement of APs in a typical office complex spread over an area of roughly 15m x 40m

The hardware we used were mainly LANCOM devices and also 6 OpenBAT devices, although flashed with the same firmware.

List of used hardware:

- 3 x LANCOM L-322agn dual Wireless [21]
- 3 x LANCOM L-452agn dual Wireless
- 6 x Hirschmann OpenBAT-R
- 1 x LANCOM WLC-4100
- 2 x LANCOM GS-2352P

Note that the distance between the APs should have been increased even further to more closely resemble a real use case-deployment. Thus the actual interference would have gone up, since with a lower SNR the APs would not have been able to recognize each others transmissions as those and categorize it as interference instead of applying CSMA/CD, leading to more corrupt packets. We were able to simulate this problem with a reduced transmit-power to some degree, but even on the lowest setting most of the APs were still able to receive each others beacons well enough. This is another factor that will lower general throughput for all setups, which we will notice in the test results. Therefore the throughput-differences between the two systems in a real deployment will probably be higher, since *AutoWDS basic* will suffer worse from this effect compared to *AutoWDS extended*.

7.2.3 Logical Structure

We simulated the traffic generating systems with OpenVZ [25] on one server with a Virtual Local Area Network (VLAN) for each system to separate the traffic for each client. This was done so that we did not need to setup twelve different real hardware machines as those merely have to send and receive traffic. The VLAN infrastructure is transparent to the VZ containers and the APs. The host-machine on the server encapsulates all frames before sending it over the VLAN Trunk to the switch, which again decapsulates the frames before sending them to the APs.

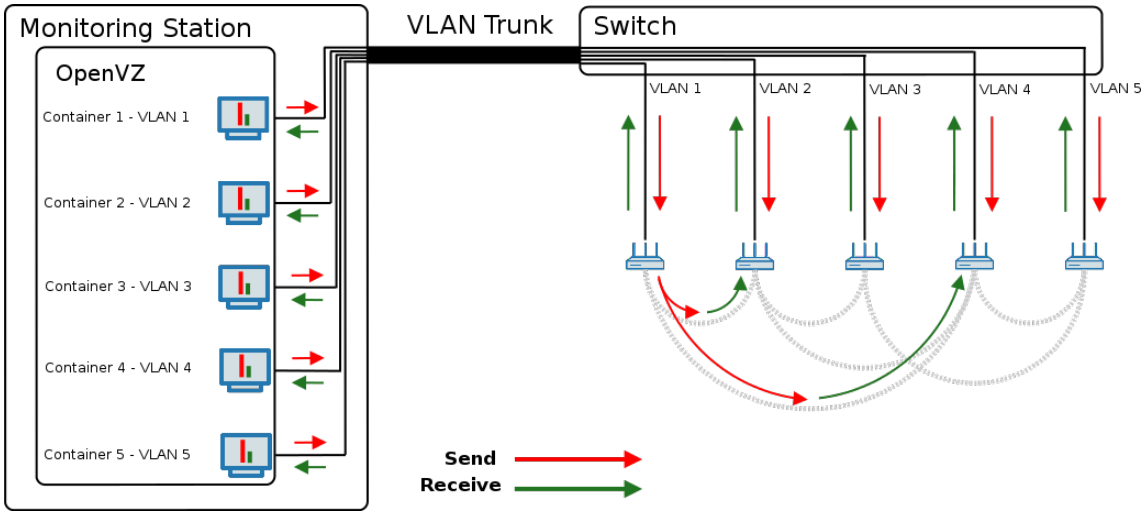


Figure 7.3 Test arrangement with OpenVZ and VLANs to simulate a busy network.

To generate the traffic we used Iperf [13] on the attached systems in User Datagram Protocol (UDP) mode with a data rate of 1Mbit per second for a total of 10 minutes. Since we could not directly send UDP frames to broadcast addresses with Iperf, we ran multiple instances of Iperf with different target ip addresses. Although 1Mbit for each stream may not seem much, however after multiplying it with the number of targets and the up- and downstreams the effective rate results in 1Mbit * 12 * 2 = 24Mbit/s for each AP. As we will see later on, this will turn out to be enough to entirely saturate the wireless network and is still within the capabilities of the used Gigabit Ethernet to get the data to and from the APs, as 24Mbit/s * 11 = 264Mbit/s does not exceed the 1000Mbit/s VLAN Trunk connection as possible bottleneck.

```
1 iperf -u -c 172.16.40.2* -t 600 -b 1M
```

Listing 7.1 Iperf mode of operation for generating the traffic.

7.3 Specification

Each test run describes a 10 minutes performance stress test for the network with throughput saturation. This should be enough time to notice any temporal effects and to get a picture of the lasting performance.

To keep environmental conditions for each of the *AutoWDS* systems as similar as possible we conducted the runs alternatively with respect to channel usage. So that a possible temporal interference in a certain band would affect both runs and not just favor a single system.

Actually to get more statistic significance we should have run this test-array more than once, but for each new setup there had to be invested a considerable amount of time in the pre-setup of *AutoWDS* (getting it up and running, despite bugs). So it was not feasible to run multiple test-arrays within the time frame that was scheduled to execute the whole evaluation.

Additionally we ran the test cases with two different antenna power-settings to simulate the hidden station problem. Therefore we first set the radios to full power, resulting in a high connectivity between the nodes and on a second run decreased the transmit power as much as possible to get a network with a smaller degree of interconnectivity.

Test run nr.	*AutoWDS* version	Channels used	Antenna power
1	basic	1	low
2	extended	1,6,11	low
3	basic	36	low
4	extended	36,40,44	low
5	basic	1	high
6	extended	1,6,11	high
7	basic	36	high
8	extended	36,40,44	high
9	extended	1,6,11,36,40,44	low
10	extended	1,6,11,36,40,44	high

Table 7.1 Test runs executed with different channelusage and varying antenna power.

7.4 Execution

The tests were executed mainly in the evening and on weekends in order to catch those short time frames where the radio bands are not as crowded as on ordinary workdays. The general procedure for a testrun was implemented as follows:

1. Power on the System (WLC + APs).
2. Wait until all APs are connected to the wireless backbone (See description of *AutoWDS* in Chapter 2.1.3.3).
3. For *AutoWDS basic* we continue with the next step, for the extended version also the following additional steps have to be conducted.
   (a) Wait until all the wireless measurements arrived at the WLC.
   (b) Run our algorithm, which reads data from the WLC, computes a solution and writes back the topology-to-do to the WLC.
   (c) Wait until the WLC reconfigured all the APs.
   (d) Wait until the APs all successfully reconnected through their new connections to the wireless backbone.
4. Start the scripts that iteratively query the APs for their status tables every seven seconds [3]
5. Start the receiving Iperf processes on the VZ machines.
6. Start the sending Iperf processes on the VZ machines.
7. Wait the predefined 10 minutes until the tests finish.
8. Archive the collected data.

[3] Seven seconds is roughly as long as it took the testcore framework to query the APs for all their table-data. Querying them more often would have had an impact on their performance.

During the testruns we made snapshots of the network topologies, where you can actually see the improvements.

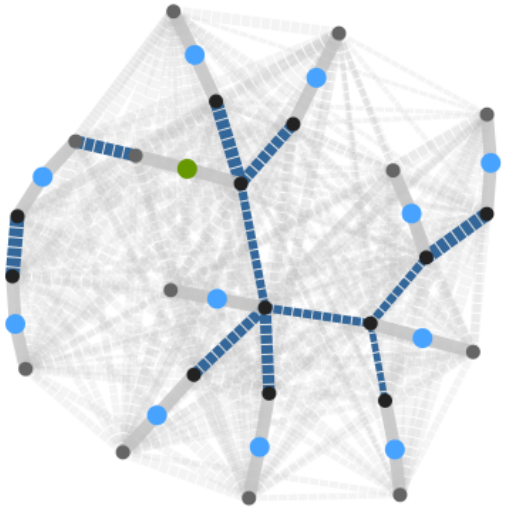


Figure 7.4 Topology of the test network with only one channel in use (*AutoWDS basic*).[4]

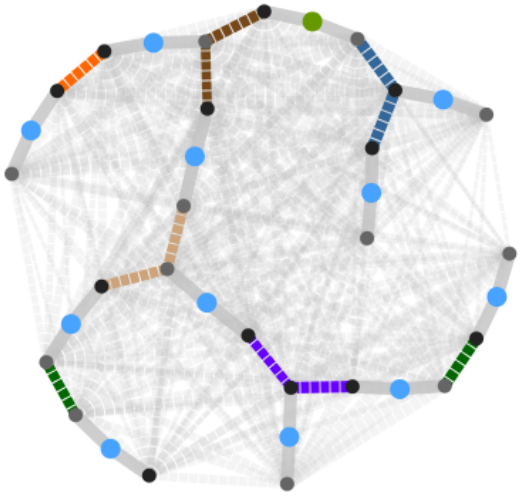


Figure 7.5 Topology of the test network with six channels in use (*AutoWDS extended*).

[4]Different edge-colors indicate different channels. Also edge thickness denotes the quality of a link, where thicker means higher quality.

After gathering and aggregating all the archived data from the testruns, we receive the following performance charts.

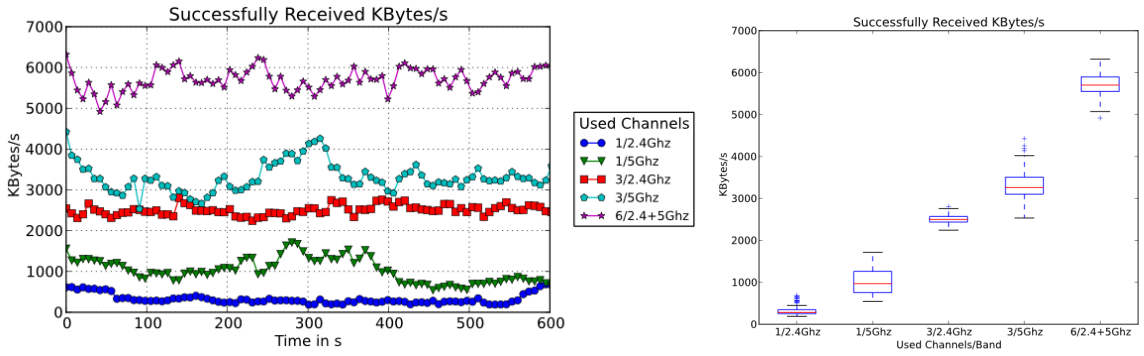


Figure 7.6 Received bytes with decreased antenna transmit power

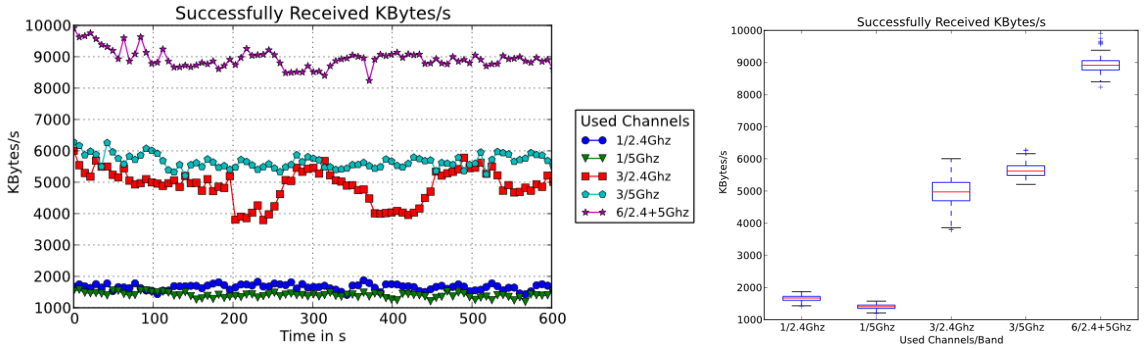


Figure 7.7 Received bytes with full transmit power

Analyzing the successfully received bytes for the low power (Figure 7.6) and high power (Figure 7.7) setup we recognize noticeable differences for each channel-assignng.

- The base scenarios where only one channel was utilized performe the worst within a range of 0.5 to 1.5Mbit/s on average per AP. Using three channels gives roughly three times the throughput (2.5 - 5Mbit/s for 2.4 GHz and 2.5 - 5Mbit/s for 5 GHz) compared to the baseline. Utilizing up to 6 Channels shows the best results with an effective throughput of 5.5 - 9Mbit/s. Note that the gain for the 6 channel setup is still linear as it provides roughly 6 times the throughput of the baseline.
- We further discern that the high transmit power setup outperforms the low power setup by a factor of two with an even more noticeable difference.
- With slight variations for the high-power 3/2.4 GHz setup the rates are also rather constant.
- Besides the high-power 1/5Ghz setup we realize 5 GHz's superiority over the 2.4 GHz band.

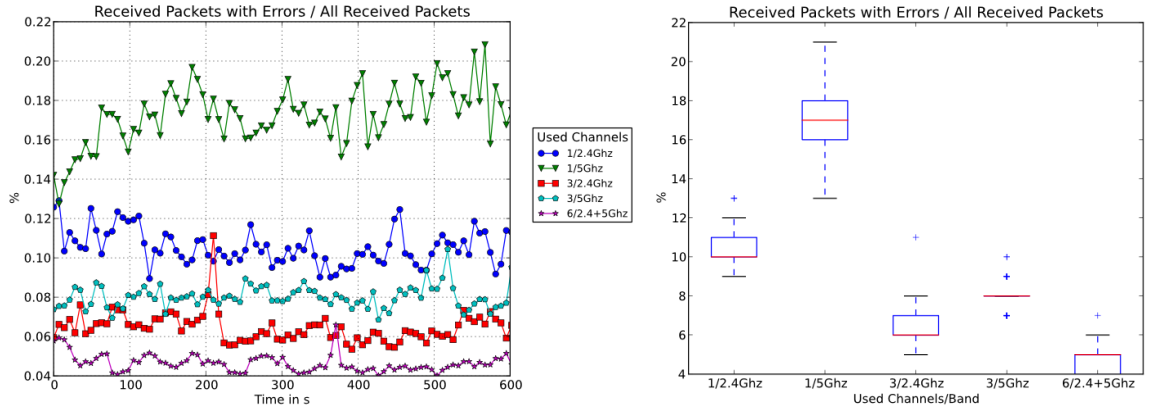


Figure 7.8 Ratio of successfully received packets to packets containing errors for reduced transmit power scenario

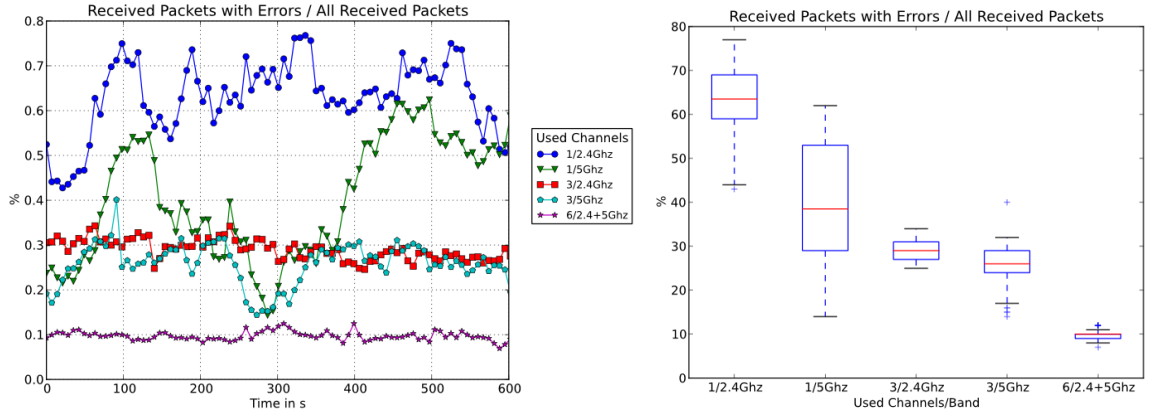


Figure 7.9 Ratio of successfully received packets to packets containing errors for full transmit power scenario

From the existing data we could also extract the number of errors over time. Deriving the rates and setting those in proportion to successfully and all transmitted packets, we can now take a look at the effective rate of corrupt packets and therefore bytes, as an error in a packet spoils the whole packet and all data in it.

- The base-scenario again performs subpar and shows the highest error rates with 10 to 65 percent depending on the transmit power. Using three channels, the error-rates shrink down to 6 to 30 percent. Adding another three channels seems to further reduce error rates to 5 to maximal 10 percent.
- As with the successfully transmitted bytes we note a greater difference when using different transmit power setups. Increasing the transmission power also increases the error-rates at least twice and up to 6 times (one channel / 2.4 GHz).
- In the low-power setup error-rates are rather constant compared to the high-power setup where we discern greater deviations especially for those tests in the 5 GHz band.

Note that different from Pol, Koomen and Spillner's recommendation [27] the test was not run by someone impartial, but also by us.

7.5 Analysis

The results match our expectations, despite all the limitations of our testbed. Using only a single channel in *AutoWDS basic* does not scale well, no matter how much transmit power is used and independent of the band. Not only is the baseline scenario incapable of transmitting much data, most of it is also corrupt. The results also affirm our estimate that the basic version transfers only a little more than control data.

While setting up the test environment we also noticed that this effect of medium overload is no result of the size of our network. Whereas two unimpeded APs were able to achieve a throughput of about 2Mbit in each direction, adding a third one would dramatically decrease overall throughput down to about 100kbit or less, growing worse with each newly added AP in receive-range. Even spreading the APs further apart would not promise better results since the inherent problem of reusing the same channel for the forwarding and receiving link restricts the forwarding capabilities drastically.

As the figures show, the outcome is dependent on the number of channels used. A setup with only three distinct channels allowed yields still better results than a mono-channel-setup, but is inferior to a setup where we can use even more channels as this gives us the possibility to use more collision domains which results in more available bandwidth and therefore increased throughput. In our example we used APs with only two radios, which limits us in selecting separate modules for different connections. This means that an AP which already established two connections over its two modules would have to share one of its modules in order to apply a new link to a foreign AP in order to maintain connectivity. In conclusion: Instead of just using a lot of channels or just using a lot of radios per AP, a combination of both promises the best results.

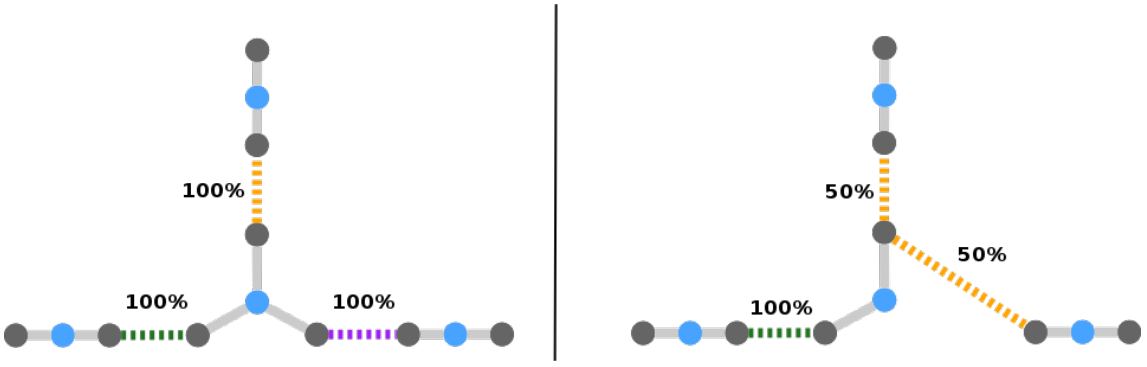


Figure 7.10 Link separation capability for an AP with three modules compared to an AP with two modules.

Compared to the base scenario our link selection and channel assignment algorithm roughly doubles the throughput with every channel added. Of course this effect is limited by using a separate channel for each link and also by the number of modules for an AP. The former effectively assigns each link its own channel and the latter is a question of practical relevance as current APs are only shipped with two or a maximum of three radio modules. As our gathered data indicate, it is after all a notable improvement to utilize multiple channels and modules.

7.5.1 Reflection on the Requirements

In this section we will analyze by how far we were able to meet the imposed requirements and restrictions from the requirement analysis.

Increased Throughput: Our measurement results indicate that throughput could considerably be increased. How much is determined by the input parameters of the algorithms like channels that can be utilized and hardware environments like the number of radio modules available at the APs.

Reduced end-to-end Link Failures: The survival path property takes care of this requirement. That is for the given error-scenario of one failing connection at a time, there is always a backup connection over a different link available as far as the underlying topology permits. We were not able to test this feature with the given implementation, but at least in theory it fulfills the 2-edge-connectedness attribute in the resulting graph. Nevertheless we are confident that it also works out in practice. If multi-flow / routing support will be implemented for our hardware in the future, a decrease in node separation for failing links should be recognizable.

Multiple Radios Utilized: Our algorithm utilizes the radio-modules as far as they are relevant for creating links between APs with a high estimated throughput. Depending on the given infrastructure not all radio modules are necessarily used, since not all of them are needed in order to create a connected network topology. Moreover using all available links may not necessarily be beneficial, since if not enough radios are available, adding new links can force other radios on the same channel, which effectively leads to sharing a single channel between more radios instead of exclusive access (See Figure 5.3).

Having some radio modules to spare gives us the opportunity to use those exclusively for client connections and assigning them a channel which does not interfere with the wireless backbone.

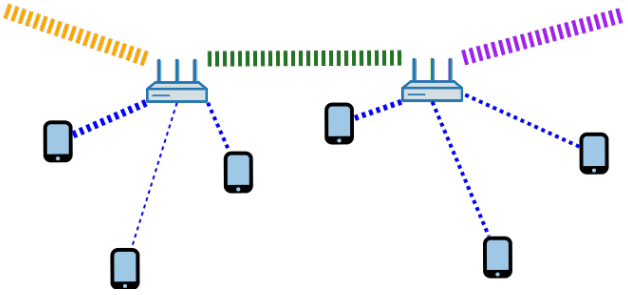


Figure 7.11 Unused modules can be used for separate client connections without interfering the WLAN backbone. This setup would require at least three radios on each AP.

Capability of Using Variable Channelsets: Our algorithm selects channels from a given input set of channels. Only those channels are then used to determine the channel assignment. It is even possible to specify only one channel, resulting in a topology optimized network graph only. Therefore this solution also fullfills this requirement.

Centralized Computation and Configuration: The requirement dictates that computing a solution should be possible on a centralized entity (servers / WLC). Our algorithm was implemented as a python script and can be run on a single system if given the right input data. Creating the configurations for the AP will still be conducted by the central WLC as depicted in Figure 6.1.

Static Environment: Also the second requirement is taken into consideration, since our algorithm is specially designed for a static environment where link quality and linkstate stay the same for a foreseeable time. If major changes in network connectivity or link quality occur, the chosen network topology and channel assignment may lead to temporal suboptimal results, therefore also a recomputation of the network topology would have to be triggered. Currently there are no automatic triggers implemented, so that an administrator would have to decide when to optimize the network (commonly after initial deployment), but our algorithm can be further extended for an automatic self-optimizing network.

Layer 2 Usage: APs still use bridged point to point connections after our optimization to span the wireless network backbone, therefore creating no obstacles for Layer 2 frames. The ethernet frames are just forwarded like in a switched environment and are not encapsulated in layer three packets. The survival paths pave the way for different routing systems, yet still to come. Eventual systems include FabicPath [34], MPLS [9] and Layer 3 routing only (breaking the Layer 2 connectivity).

Economic Constraints: Algorithm runtime mainly depends on the number of connections between the nodes. Calculating the initial spanning tree is done in BFS mode and therefore has quadratic runtime for the worst case, depending on how sparse the graph is. The following calculation of the survival paths has also quadratic runtime, since we iterate over all connections and for each connection in the worst case over all other connections to find the backup path. In a common deployment scenario the underlying network graph will be rather sparsly connected since the number of APs used and placement are carefully selected. Typically they are placed in a way that they span a network over an area as far as possible. Current deployments and deployments in near future won't exceed 100 APs. Even if fully meshed with a total of $\sum_{i=1}^{100} i = \frac{100(100+1)}{2} = 5050$ edges it won't pose a problem to the capabilities of a WLC or an administrators machine.

7.5.2 Reflection on Related Work

Comparing our solution to those of others would need a common ground. For example hardware requirements, number of modules and in general heterogenity / homgenity of the network are factor that could play in favor of some algorithms and therefore won't lead to fair results. Additionally different solutions were designed to achieve different goals like throughput, latency or weighted throughput networks. Certainly we could agree on this common deployment and compare each solution for random networks with respect to throughput, but this would require a full-scale simulation as hardware-resources are limited and too easily prone to real world difficulties. Although maybe a key aspect, we did not have enough time to simulate even our own solution to the full extent and therefore neither those of others. Nevertheless we would appreciate further work on this approach. In order to at least provide some kind of comparison we will take a look at the features of our work, which most of the related work did not show or provide.

Variable Nr. of Radios: Our algorithms support a variable number of radios on each device instead of a fixed numer like in INSTC [35]. This allows to optimize the wireless network even for heterogeneous environments where devices have a variable number of radios or just simply one.

Vendor Neutral: Although we used only LANCOM devices in our case, the solution is vendor neutral. As long as the devices provide the required data, any network with devices that can be configured (also manually) to use a specific link for a certain radio can profit from our optimization.

No MAC changes: Our solution does not require any Media Access Control (MAC)-layer changes and can be run on any commodity WLAN hardware.

No Forced Backup Links: Thanks to our modular approach, adding redundant links to the initial MST is optional. You are free to only optimize and use the MST if you are also limited by the STP, like we were in our evaluation.

Considers Foreign Interference: Admittedly we are not the first to do so (BFS-CA [29]), but our solution also considers foreign interfering WLANs which can have a great impact in performance if neglected.

Customizable Survival Pats: In order to create the redundant links (survival paths) we iterate over each link in the MST and simulate it failing. You could just as well iterate over the nodes in the MST to get a k-node-connected survival graph or whatever order you chose. This means you can add redundant links in decreasing order of optimality up to a fully meshed graph.

Customizable Ranking-Formula: Without great effort you can also adapt the ranking-formula and create a spanning tree which is optimal to some other metrics you need.

No Gateway/Special Node necessary: At last you do not need a special gateway node like in BFS-CA [29] to start our algorithm with. This is especially usefull in setups where all the node are equal or a uplink to the internet is not desired.

8

Conclusion

In this work we have presented a new algorithm, which optimizes overall throughput of a WDS by evading interference through an optimized network topology and utilizing multiple channels with multiple radios. Additionally it creates a more failure resistant network topology by adding redundant links to the infrastructure.

We implemented this algorithm in python to make this approach easy to use, customizable and comprehensible by providing examples, explanations for how and why we did things the way we did and documenting our code.

Our evaluation indicates a major improvement on the existing *AutoWDS basic* of at least up to 9 times the throughput in an admittedly rather small testbed. Yet, the optimization made the system really usable for wireless clients with high throughput demands.

Although this work does not break any new ground in scientific research, it solves the imposed problem well by reusing and extending existing ideas and algorithms to achieve its goal of considerably increasing throughput for a given WDS.

8.1 Limitations

The requirements analysis explicitly dictates the static attribute of APs and consequently their link quality, therefore our solution won't yield good results for more volatile setups where link quality or network topology change frequently as the topology is not continuously optimized. This would be the case if for example the APs are mobile or the environmental factors impacting link quality change rapidly and/or often.

Since also a key requirement, this solution works only in a centralized managed fashion and thereby can't be used for unmanaged more autonomous APs.

8.2 Future Work

Open issues which could not be addressed by us, but nevertheless would like to see implemented or tested are the following:

Further Evaluation: Our evaluation is more a proof of concept instead of an elaborate audit. As described we ran our throughput-tests only in our lab environment, which was also designed with other metrics and scenarios kept in mind. Therefore a more thorough and dedicated approach would include an increased hardware diversity, in general the full scale testing of up to 100 devices and longer test-periods with varying deployment parameters like transmit-power, configuration timings and different topologies. Especially simulating our solution with random graph layouts and different underlying topologies could provide opportunities to even further increase throughput. Also the feature of evading foreign used channels should be evaluated if the implementation of *AutoWDS* permits as this will be one of the factors which strongly affect performance.

Extensive Comparison to Related Work: As we only compared our work by listing features, a true comparison still has to be done if one would like to know the strength and weaknesses considering actual throughput-performance. As noted this will probably take some time as all the solutions would have to be implemented in presumably the same language. And then deciding on the environmental settings like interference by other devices and node connectivity will create new challenges.

Further Optimization of the Algorithm: Algorithm runtime is not in the focus of concern at the moment but could be in the future. As a result and in order to more closely integrate the solution into *AutoWDS extended*, porting the code to a native language like C/C++ could give some additional performance gains.

Learning Algorithm: In its current state, the optimization is triggered through the administrator or user during the initial setup or every now and then if the network appears slow. As this always just captures the status quo of the network, the optimization is susceptible to temporary interferences which could lead to suboptimal results in the long run and it would have to be rerun in order to keep up the best topology possible. To deal with the snapshot difficulty we had the idea of also looking at the history of a used channel and then decide if it would be worth it to continue to use a certain link and channel or switch to another one. This would allow us to better evaluate link-quality and provide us with a more fine grained view on the network as a whole and how it is affected by other influences over time. We could then act upon those changes automatically in a more timely manner leading to even better results.

Bibliography

[1] Adya, A., Bahl, P., Padhye, J., Wolman, A., and Zhou, L. A multi-radio unification protocol for ieee 802.11 wireless networks. In *Broadband Networks, 2004. BroadNets 2004. Proceedings. First International Conference on* (2004), IEEE, pp. 344–354.

[2] Airberry. Introduction to mesh networks, 2012.

[3] Akyildiz, I. F., Wang, X., and Wang, W. Wireless mesh networks: a survey. *Computer Networks 47*, 4 (2005), 445 – 487. http://www.sciencedirect.com/science/article/pii/S1389128604003457.

[4] Atlassian, I. U. Bitbucket, aug 2014. https://bitbucket.org/.

[5] Bethard, S. J. Python argparse library, Aug 2014. https://docs.python.org/2.7/library/argparse.html.

[6] Boruuvka, O. O jistém problému minimálním.

[7] Das, A. K., Alazemi, H. M., Vijayakumar, R., and Roy, S. Optimization models for fixed channel assignment in wireless mesh networks with multiple radios. In *SECON* (2005), pp. 463–474.

[8] Das, A. K., Vijayakumar, R., and Roy, S. Wlc30-4: static channel assignment in multi-radio multi-channel 802.11 wireless mesh networks: issues, metrics and algorithms. In *Global Telecommunications Conference, 2006. GLOBECOM'06. IEEE* (2006), IEEE, pp. 1–6.

[9] Davie, B., and Rekhter, Y. *MPLS: technology and applications*. Morgan Kaufmann Publishers Inc., 2000.

[10] Wds linked router network, aug 2014. http://www.dd-wrt.com/wiki/index.php.

[11] Dijkstra, E. W. A note on two problems in connexion with graphs. *Numerische mathematik 1*, 1 (1959), 269–271.

[12] Foundation, P. S. Python collections library @ONLINE, jun 2014. https://docs.python.org/2.7/library/collections.html.

[13] Gates, Warshavsky, T. F. D. Q. *Iperf v2.0.4: The TCP/UDP bandwidth measurement tool*. NLANR applications support, University of Illinois at Urbana-Champaign, Urbana, IL, USA, may 2014. http://iperf.sf.net.

[14] Hagberg, A. A., Schult, D. A., and Swart, P. J. Exploring network structure, dynamics, and function using NetworkX. In *Proceedings of the 7th Python in Science Conference (SciPy2008)* (Pasadena, CA USA, Aug 2008), pp. 11–15.

[15] Hertz, A., and de Werra, D. Using tabu search techniques for graph coloring. *Computing 39*, 4 (1987), 345–351.

[16] Hiertz, G., Denteneer, D., Max, S., Taori, R., Cardona, J., Berlemann, L., and Walke, B. Ieee 802.11s: The wlan mesh standard. *IEEE Wireless Communications* (Feb 2010), 104–111. http://www.comnets.rwth-aachen.de.

[17] Janssen, C. Carrier sense multiple access (csma), aug 2014. http://www.techopedia.com/definition/5649/carrier-sense-multiple-access-csma.

[18] JetBrains. Pycharm, aug 2014. Version 3.4.1. http://www.jetbrains.com/pycharm/.

[19] Junio C. Hamano, Shawn O. Pearce, L. T. Bitbucket, aug 2014. http://git-scm.com/.

[20] Katzela, I., and Naghshineh, M. Channel assignment schemes for cellular mobile telecommunication systems: A comprehensive survey. *Personal Communications, IEEE 3*, 3 (1996), 10–31.

[21] Lancom systems products, Aug 2014. http://www.lancom-systems.de/.

[22] Physical (phy) layer of lan/man interfaces. the ethernet and the wlan interface, aug 2014. http://alpha.tmit.bme.hu/meresek/lanfiz_eng.htm.

[23] Marina, M. K., Das, S. R., and Subramanian, A. P. A topology control approach for utilizing multiple channels in multi-radio wireless mesh networks. *Computer Networks 54*, 2 (2010), 241 – 256. Wireless Multi-Hop Networking for Infrastructure Access. http://www.sciencedirect.com/science/article/pii/S1389128609002217.

[24] Nobuaki Otsuki, Yusuke Asai, T. S. Research and development of wireless network coding to achieve high-efficiency multihop wireless systems. *NTT Technical Review 8*, 3 (2010).

[25] Openvz linux containers, jul 2014. http://openvz.org.

[26] Padhye, J., Agarwal, S., Padmanabhan, V. N., Qiu, L., Rao, A., and Zill, B. Estimation of link interference in static multi-hop wireless networks. In *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement* (2005), USENIX Association, pp. 28–28.

[27] Pol, M., Koomen, T., and Spillner, A. Management und optimierung des testprozesses. *Heidelberg: dpunkt. verlag* (2000).

[28] Prim, R. C. Shortest connection networks and some generalizations. *Bell system technical journal 36*, 6 (1957), 1389–1401.

[29] Ramachandran, K. N., Belding-Royer, E. M., Almeroth, K. C., and Buddhikot, M. M. Interference-aware channel assignment in multi-radio wireless mesh networks. In *INFOCOM* (2006), vol. 6, pp. 1–12.

[30] Raniwala, A., Gopalan, K., and Chiueh, T.-c. Centralized channel assignment and routing algorithms for multi-channel wireless mesh networks. *SIGMOBILE Mob. Comput. Commun. Rev. 8*, 2 (apr 2004), 50–65. http://doi.acm.org/10.1145/997122.997130.

[31] Si, W., Selvakennedy, S., and Zomaya, A. Y. An overview of channel assignment methods for multi-radio multi-channel wireless mesh networks. *Journal of Parallel and Distributed Computing 70*, 5 (2010), 505 – 524. http://www.sciencedirect.com/science/article/pii/S074373150900183X.

[32] Smithies, D. Minstrel, 2005. http://madwifi-project.org/browser/madwifi/trunk/ath_rate/minstrel/minstrel.txt.

[33] Subramanian, A. P., Gupta, H., Das, S. R., and Cao, J. Minimum interference channel assignment in multiradio wireless mesh networks. *Mobile Computing, IEEE Transactions on 7*, 12 (2008), 1459–1473.

[34] Systems, C. Scale data centers with cisco fabricpath. Tech. rep., Cisco Systems, Inc. USA, 2012.

[35] Tang, J., Xue, G., and Zhang, W. Interference-aware topology control and qos routing in multi-channel wireless mesh networks. In *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing* (New York, NY, USA, 2005), MobiHoc '05, ACM, pp. 68–77. http://doi.acm.org/10.1145/1062689.1062700.

[36] Tim Dwyer, T. J. D3js force-directed graph, Aug 2014. http://d3js.org/,http://bl.ocks.org/mbostock/4062045.

[37] Zhu, J., and Roy, S. 802.11 mesh networks with two-radio access points. In *Communications, 2005. ICC 2005. 2005 IEEE International Conference on* (2005), vol. 5, IEEE, pp. 3609–3615.

A

Appendix

A.1 List of Abbreviations

AP	Accesspoint
AutoWDS	Automatic WDS
BFS	Breadth First Search
BFS-CA	Breadth First Search Channel Assignment
CAA	Channel Allocation Algorithm
CA	Channel Assignment
CAPWAP	Control And Provisioning of Wireless Access Points
CAS	Channel Assignment Server
CCA	Clustered Channel Assignment
CLICA	Connected Low Interference Channel Assignment
CSMA/CA	Carrier Sense Multiple Access / Collision Avoidance
CTA	Centralized Tabu-based Algorithm
CTS	Clear To Send
DJP	Dijkstra, Jarník, Prim
DTLS	Datagram Transport Layer Security
DVB-T	Digital Video Broadcasting – Terrestrial
ESSID	Extended SSID
HD	High Definition
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers
INSTC	Minimum Interference Survivable Topology Control
IP	Internet Protocol
JSON	JavaScript Object Notation
LAN	Local Area Network
LCI	Link Co-channel Interference
LCOS	LANCOM Operating System
MAC	Media Access Control
MCG	Multi-Radio Conflict Graph

MPLS	Multiprotocol Label Switching
MST	Minimal Spanning Tree
MUP	Multi-radio Unification Protocol
PAN	Personal Area Network
PEP8	Python Enhancement Proposal 8
RTS/CTS	Request To Send / Clear To Send
RTS	Request To Send
SHF	Super High Frequency
SNMP	Simple Network Management Protocol
SNR	Signal-to-Noise Ratio
SSH	Secure SHell
SSID	Service Set Identifier
STP	Spanning Tree Protocol
Telnet	Telecommunication Network
TLV	Type-Lenght-Value
UDP	User Datagram Protocol
UHF	Ultra High Frequency
VLAN	Virtual Local Area Network
WDS	Wireless Distribution System
WLAN	Wireless Local Area Network
WLC	Wireless LAN Controller
WMN	Wireless Mesh Network
WPA2	Wi-Fi Protected Access 2