Document Properties

Document Number:	D 3.3
Document Title:	**Final Report on Case Studies and Analysis for Dynamic Radio and Backhaul Configuration Mechanisms**
Document Editor:	Sébastien Auroux (UPB)
Authors:	Arash Asadi (IMDEA), Carlos Donato (UC3M), Martin Dräxler (UPB), Vincenzo Mancuso (IMDEA), Arianna Morelli (INCS), Vincenzo Sciancalepore (IMDEA), Pablo Serrano (UC3M), Fabio Toninelli (INCS), Christian Vitale (IMDEA)
Target Dissemination Level:	PU
Status of the Document:	Submitted to EC
Version:	1.0

Production Properties:

Reviewers:	Antonio de la Oliva (UC3M) Engin Zeydan (AVEA)

Document History:

Revision	Date	Issued by	Description
1.0	2015-04-30	UPB	First EC submission

Disclaimer:

Abstract:

This deliverable presents the final report on case studies and analysis for dynamic radio and backhaul reconfiguration mechanisms. Corresponding to the last deliverable of WP3, we focus on presenting the final results of the research activities performed within this work package of CROWD. The document first focuses on network discovery and monitoring, dynamic controller placement and dynamic backhaul reconfiguration, before reporting on the case studies on regional control of LTE and WiFi Direct relay by coordinating centralised ABSF and D2D clustering strategies, as well as infrastructure on demand for 802.11 networks.

Keywords:

interfaces, dynamic radio configuration, backhaul reconfiguration, controller placement, regional control, power cycling

Contents

List of Figures

List of Tables

List of Project Partners

Name	Acronym	Country
Intecs S.p.A. (*coordinator*)	INCS	Italy
Alcatel-Lucent Bell Labs France	ALBLF	France
Avea Iletism Hizmetleri A.S.	AVEA	Turkey
Fundacion IMDEA Networks	IMDEA	Spain
National Instruments	SIG	Germany
Universidad Carlos III de Madrid	UC3M	Spain
Universität Paderborn	UPB	Germany

List of Acronyms

3GPP	Third Generation Partnership Project
ABSF	Almost Blank Sub-Frame
AP	Access Point
BS	Base Station
BSB	Base Station Blanking
BSS	Basic Service Set
CBS	Coordinated Base Station Set
CLC	CROWD Local Controller
CN	Correspondent Node
CoMP	Coordinated Multi-Point
CPS	Coupled Processors System
CRC	CROWD Regional Controller
CROWD	Connectivity management for eneRgy Optimised Wireless Dense networks
D2D	Device-to-Device
DRONEE	Dual-radio opportunistic networking for energy efficiency
eNB	Evolved NodeB
FCPF	Flow processing-aware Controller Placement Framework
FlexFCPF	Flexible Flow processing-aware Controller Placement Framework
HTTP	Hypertext Transfer Protocol
ICIC	Inter-Cell Interference Coordination
ICI	Inter-Cell Interference
IoD	Infrastructure on Demand
JSON	JavaScript Object Notation
LTE	Long Term Evolution
LLDP	Link Layer Discovery Protocol
MAC	Medium Access Control

MCS	Modulation and Coding Scheme
MIQCP	Mixed Integer Quadratically Constrained Program
MN	Mobile Node
NB	North-Bound
OF	OpenFlow
PDU	Power Distribution Unit
RAN	Radio Access Network
SDN	Software Defined Network
SINR	Signal to Interference plus Noise Ratio
SNMP	Simple Network Management Protocol
TCP	Transmission Control Protocol
TLS	Transport Layer Security
UE	User Equipment
VNF	Virtual Network Functions
WiFi	Wireless Fidelity
WP3	Dynamic Radio and Backhaul Configuration Mechanisms

Executive summary

The purpose of this document is to present the final report on case studies and analysis for dynamic radio and backhaul reconfiguration mechanisms conducted within Dynamic Radio and Backhaul Configuration Mechanisms (WP3) of Connectivity management for eneRgy Optimised Wireless Dense networks (CROWD). Since the final specifications of functions and interfaces for dynamic radio and backhaul configuration mechanisms have already been reported in D3.2, this deliverable focuses on presenting the final results of the research activities carried out in WP3. Given that much of the research ideas presented in this document have been previously discussed in D3.1 and D3.2, we have opted for a compact presentation of the results, having first a brief overview of the research topics with further detailed explanation included in the document as appendix. A brief explanation of each of the section composing this document can be found in the following:

- Network Discovery Mechanism: This chapter describes the finalised version of the CROWD network discovery mechanism that allows the CROWD controllers to identify CROWD compliant network elements.
- Dynamic Controller Placement: This section continues and finalises the work on the dynamic controller placement that has been introduced in D3.2. We provide an improved optimisation model and a heuristic framework, which enables a fast and flexible placement of the CROWD controllers.
- Dynamic Backhaul Reconfiguration: This work constitutes and extension of the backhaul reconfiguration work presented in D3.2 which enables the use of the CROWD backhaul configuration approach in dense wireless access networks with hotspots of users.
- Control of LTE and WiFi direct relay by coordinating centralised ABSF and D2D clustering strategies: This chapter finalises the case study on the control of LTE and WiFi direct relay by reporting how to evaluate analytically the capacity region of WiFi devices in dense wireless access networks.
- Infrastructure on Demand for 802.11 WiFi networks: This work extends and finalises the case study on Infrastructure on Demand for 802.11 WiFi networks by providing evaluation results of the Infrastructure on Demand (IoD) testbed introduced in D3.2.

Key contributions

The main technical contributions of this deliverable can be summarised as follows:

- Description of Connectivity management for eneRgy Optimised Wireless Dense networks (CROWD) network discovery mechanism for the CRC
- Final optimisation model, heuristic algorithm and evaluation for the dynamic controller placement
- Improved heuristic algorithm and evaluation for the dynamic backhaul reconfiguration
- Final results of the study of regional control of LTE and WiFi direct relay by coordinating centralised ABSF and D2D clustering strategies developed in WP2
- Final results of the study of Infrastructure on Demand for 802.11 WiFi

The key contributions to the research and standardisation communities resulting from the work performed within the context of this deliverable are the following:

- Paper on "Efficient Flow Processing-aware Controller Placement in Future Wireless Networks" at IEEE WCNC 2015 [1]
- Paper on "Dynamic Network Reconfiguration in Wireless DenseNets with the CROWD SDN Architecture" at EuCNC 2015 [2]
- Paper on "Tackling the increased density of 5G networks; the CROWD approach" at 5GArch 2015 (VTC 2015) [3]
- Paper on "Dynamic Backhaul Network Configuration in SDN-based Cloud RANs" (Preprint) [4]
- Paper on "Modeling the Impact of Start-Up Times on the Performance of Resource-on-Demand Schemes in 802.11 WLANs" at SustainIT 2015 [5]
- Paper on "An OpenFlow Architecture for Energy Aware Traffic Engineering in Mobile Networks" at IEEE Network (accepted for publication) [6]

1 Introduction

This deliverable constitutes the last deliverable of Dynamic Radio and Backhaul Configuration Mechanisms (WP3) of CROWD. As reported in the Executive Summary, the final specifications of functions and interfaces for dynamic radio and backhaul configuration mechanisms have already been reported in D3.2. Therefore, this deliverable focuses on the final results of the research activities carried out within WP3, which concludes the work presented in the previous deliverables, D3.1 [7] and D3.2 [8].

As reported in previous deliverables, wireless traffic demand is growing exponentially and this growing demand can only be satisfied by increasing the density of points of access and combining different wireless technologies. This approach is already pursued by network operators. The CROWD architecture tackles the issues imposed by this densification of wireless access networks.

In this document, we assess the research activities within CROWD on the mechanisms for configuring radio access and backhaul networks, in terms of energy efficiency, coverage, and capacity, which operate at medium to long time scales. The document is structured in two parts, document body and appendix. While the document body includes brief explanations of the research work done, the appendix provides additional details on for individual research items.

The document body is again divided into two parts. In the first part we focus on the network discovery mechanism, the dynamic controller placement and the dynamic backhaul reconfiguration. We introduce and evaluate optimisation models and heuristic algorithms for both controller placement and backhaul reconfiguration and provide substantial evaluation results. In the second part we focus on extended results of the case studies of applications that can be executed with the CROWD Regional Controller (CRC). We show results for control of LTE and WiFi Direct relay by coordinating centralised ABSF and D2D clustering strategies developed in WP2, as well as testbed evalution results for the case study for Infrastructure on Demand for 802.11 WiFi networks.

2 Research activities

In this section we present the research work performed in the area of Dynamic Radio and Backhaul Configuration Mechanisms (WP3) during the last 10 months of work in the CROWD project. It is split into two parts: In the first part (Section 2.1) we present the results related to controller placement and backhaul reconfiguration (Tasks 3.1, 3.2 and 3.4) and in the second part (Section 2.2) we present the results from the case studies from Task 3.3.

2.1 Controller Placement & Backaul Reconfiguration

In this section we present results related to controller placement and backhaul reconfiguration (Tasks 3.1, 3.2 and 3.4).

2.1.1 Network Discovery Mechanism

According to the CROWD architecture (D1.3) we identified a district which consist in base stations, LTE and WiFi AP, and interconnecting backhaul links reconfigurable via OpenFlow (OF) where the CROWD Local Controller (CLC) works and enforces the decision in a short time scale, and the region as a logical area including several district and where the CRC operates in medium-long time scale. The optimisation of the network is performed by applications that run on top of the controllers. Applications take decisions that have to be enforced on the underlying networks. More detailed knowledge of the network the applications have, more the optimisation can be performed. The network discovery mechanism aims to provide specific information and characteristic of network elements which populate the districts in the reference scenarios.

The CLC and the CRC manage the Network discovery mechanism. The CLC receives information from the district, Wireless Fidelity (WiFi) and/or Long Term Evolution (LTE) nodes and/or from the backhaul: periodically the controller requires to the underlying networks the updating of the network elements considering the status of the nodes, i.e. if the network elements of Radio Access Network (RAN) or the backhaul elements are switched on or off, the behaviour of the nodes, the traffic and the connection or the users they served.

All the information are collected and used for statistic analysis. Moreover the CLC provides them to the higher layers, CRC and applications that work in short-term timescale, by means the North-Bound (NB) interfaces described in [9]. Figure 2.1 illustrates the communication flow.

The CLC receives information by means the specific interfaces at the southbound level as described in D3.2 [8]:

- The OpenFlow switch network uses Link Layer Discovery Protocol (LLDP)
- The LTE network is managed through a proprietary protocol
- The WiFi access points are managed by means the Simple Network Management Protocol (SNMP) protocol.

The proprietary protocol used for LTE networks has been developed in JavaScript Object Notation (JSON) format, the main messages exchanged for the network discovery scope are reported in Section A.1.

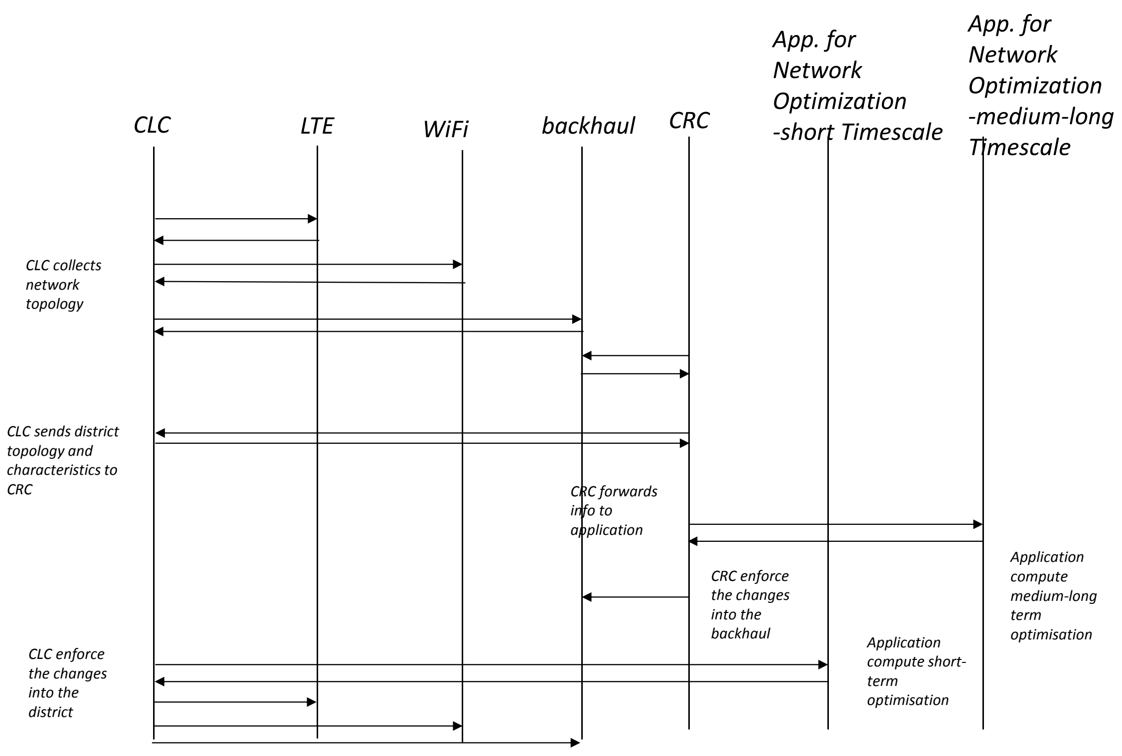
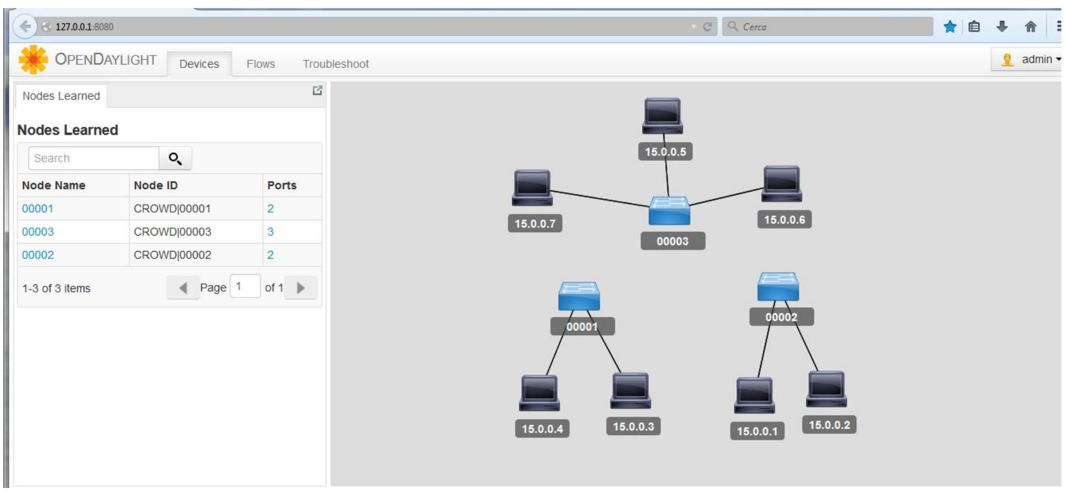

CLC
LTE
WiFi
backhaul
CRC
App. for Network Optimization -short Timescale
App. for Network Optimization -medium-long Timescale
CLC collects network topology
CLC sends district topology and characteristics to CRC
CRC forwards info to application
CRC enforce the changes into the backhaul
Application compute medium-long term optimisation
Application compute short-term optimisation
CLC enforce the changes into the district

Figure 2.1: Flow diagram network discovery

The specific network element information collected are related to the load, the bandwidth measurements and the properties of the node. Statistics about the average load and traffic are calculated by the CLC.

As a result, the controller can identify the CROWD compliant element, WiFi and LTE and indicate them properly. Figure 2.2 shows the outcome of the network discovery mechanism.



Nodes Learned

Node Name	Node ID	Ports
00001	CROWD|00001	2
00003	CROWD|00003	3
00002	CROWD|00002	2

1-3 of 3 items Page 1 of 1

Figure 2.2: CLC outcome network discovery

2.1.2 Dynamic Controller Placement

Modern cellular networks deployed by mobile network operators are generally very dense and heterogeneous to cope with the consistently growing traffic demands in mobile networks [10, 11]. These networks impose several challenges, such as a large amount of signalling overhead, massive inter-cell interference or high energy consumption. Hence, much recent work has been dealing with efficient control architectures for such wireless access networks and Software-Defined Networking (Software Defined Network (SDN)) with distributed *controllers* is often found to be a promising approach for this task [12, 13, 14].

But in addition, future wireless access networks as envisioned by CROWD need a vast range of network mechanisms with various purposes. These mechanisms range from network applications meant to optimise the usage of resources, e.g. Coordinated Multi-Point (CoMP) transmission, over coordinating network mechanisms for Inter-Cell Interference Coordination (ICIC) to virtualised applications such as Virtual Network Functions (VNF) to flexibly add network functions. All of these mechanisms induce several *data flows* in the network and thus there is considerable data flow processing work to be done in future wireless networks besides the mere control tasks associated with SDN controllers. Because of this, CROWD offers controller devices that are able to perform both conventional SDN network control and data flow processing to efficiently manage future wireless networks.

In Deliverable D3.2 [8] we have already introduced our Flow processing-aware Controller Placement Framework (FCPF) approach that places the CROWD controller architecture into a cellular network while meeting the requirements on latency, data rate and processing capacity imposed by both data flow processing and network control. We have further shown that this problem is NP-hard. However, the optimisation model shown was way too slow for a real-world usage and suffered from an overly complicated assignment of data flows to controllers.

As a result, we have improved our optimisation model and we have developed a fast heuristic algorithm to solve the flow processing-aware controller placement problem with a reasonable runtime. However, finding a good placement for such controllers for a given time only solves the problem temporarily. As the network load of a mobile network can change rapidly, the performance of a static one-time placement will generally become worse over time. Especially in modern dense and crowded networks, a large number of data flows appear and expire every second. The obvious solution might be to compute a new controller placement from scratch when the performance of the current placement decreases. But this approach has a lot of downsides. First of all, a new controller placement is expected to result in a lot of different control assignments compared to the previous placement, causing much reconfiguration overhead. In addition, even with a fast heuristic algorithm, the runtime for a complete new placement is likely too high for frequently changing network states. Hence, we have extended our heuristic algorithm to perform flexible controller reassignment based on the *current* controller placement.

In this section we first present the final version of our FCPF problem statement in Section 2.1.2.1. Next, we give a short overview of the FCPF implementations in Section 2.1.2.2, which are described in detail in Section A.2. Last but not least, we provide various evaluation results for our flow processing-aware controller placement approaches in Section 2.1.2.3.

2.1.2.1 Final FCPF problem statement

As input for FCPF, we consider a graph $G = (V, E)$ with nodes V, e.g. LTE Evolved NodeB (eNB)s, WiFi Access Point (AP)s and switches, and unidirectional edges E, representing the backhaul links of the network. The nodes that fulfil the hardware requirements for serving as a controller are denoted by $C \subseteq V$. Every *potential controller* $c \in C$ has a processing power of $p_{\text{own}}(c)$ FLOPS and each link $(u, v) \in E$ has a maximum data rate of $b_{\text{cap}}(u, v)$ bit/s and a latency of $l_{\text{cap}}(u, v)$ seconds.

A *valid solution* requires a complete control structure, i.e. each node $v \in V$ is required to be controlled by at least one CLC (a node can be controlled by more than one CLC if needed for optimal network performance) and each CLC is required to be coordinated by exactly one CRC. Being the CLC of a node or the CRC of a CLC requires a processing capacity of p_{CLC} or p_{CRC} FLOPS, respectively. Also, the routing path between a node and its CLC needs to have at most a round trip latency of l_{CLC} s and a minimum data rate of b_{CLC} bit/s. Similarly, the round trip latency l_{CRC} s and the data rate b_{CRC} bit/s are required for the routing path of a CLC to its CRC. Our model determines these routing paths to ensure that these constraints are fulfilled. We use the binary variables $f_{c,d,u,v}$ to express that $(u, v) \in E$ is used on the routing path from node $d \in V$ to CLC $c \in C$ and the binary variables $g_{c,d,u,v}$ to describe the routing paths between a CLC and its CRC.

By way of illustration, Figure 2.3 shows a typical controller placement scenario and a possible control structure.

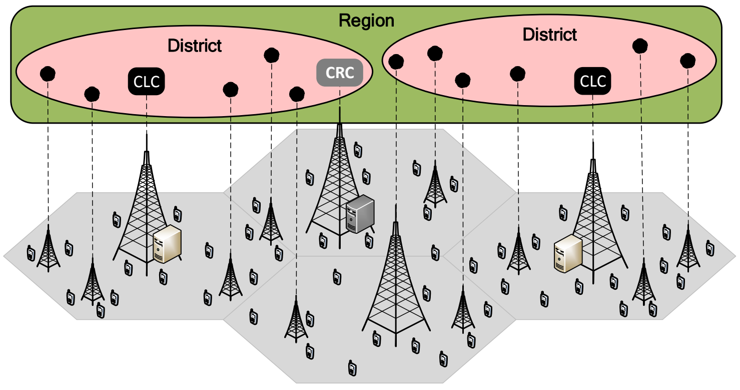


Figure 2.3: Typical controller placement scenario

Furthermore, we have a set F of data flows. Each flow is passing through certain nodes from V, denoted by the connection matrix $W \in \{0,1\}^{|F|\times|C|}$, defining the position of a data flow in the network. Every flow $x \in F$ requires a processing power of $p_{\text{flow}}(x)$ from a CLC. Further, the routing paths between the CLC and the nodes the flow is passing through need to provide a maximum round trip latency $l_{\text{flow}}(x)$ s and a data rate of $b_{\text{flow}}(x)$ bit/s. We have decided to consider the full round trip for all data flows, so that this view incorporates both upload and download traffic.

It is the main objective of FCPF to maximise the amount of data flows $x \in F$ that are *satisfied* by one CLC. By our definition, a CLC $c \in C$ *satisfies* a data flow x if and only if c controls all nodes v with $W_{x,v} = 1$ and the routing paths from all these nodes to c have sufficient resources to provide a data rate of $b_{\text{flow}}(x)$ and the demanded round trip latency $l_{\text{flow}}(x)$ taking into account the required processing capacity of $p_{\text{flow}}(x)$ FLOPS for processing x at c. The scheduler for data processing at a CLC is assumed to use equal share between all controlled nodes and all satisfied data flows. To guarantee that no constraint for network control and data flow satisfaction is violated, FCPF determines the necessary routing paths and verifies that all conditions are met at any time.

The main objective is to create a *valid solution*, i.e. a complete control structure with a maximum amount of data flows being satisfied by a CLC. The idea behind this is as follows: While we would of course like to see all data flows satisfied at any time, this might not always be possible because of a combination of too many data flows and too little network resources. So in this case, we see an incomplete control structure, i.e. not all nodes are correctly controlled, as more critical for the network than a couple of yet not processed data flows.

2.1.2.2 FCPF implementations

We have developed both an optimisation model and a heuristic algorithm to solve the FCPF problem. The former is a Mixed Integer Quadratically Constrained Program (MIQCP) and is presented in Section A.2.1. Our heuristic algorithm, Flexible Flow processing-aware Controller Placement Framework (FlexFCPF), is based on a multi-layer greedy approach and is described in Section A.2.2.

2.1.2.3 Evaluation

Our evaluation consists of three parts: at first we evaluate our optimisation model (MIQCP) against FlexFCPF's initial controller placement, which is only feasible in very small scenarios. Then, we provide results for the performance of the initial controller placement in larger, real-world size scenarios. Last but not least we evaluate FlexFCPF's flexible reassignment abilities using a dynamic network simulation.

All evaluations are executed on Intel Xeon X5650 CPUs running at 2.67 GHz. The optimisation problem has been implemented using the Pyomo package for optimisation modelling in Python [15] and is solved with Gurobi 5.6.2 [16] running in single-threaded mode. FlexFCPF has been implemented using Python. All plots contain confidence intervals at a 95% level unless they are too small and covered by the plot markers.

Evaluation scenario For all instances of this evaluation the nodes are placed on a regular grid with a mean inter-Base Station (BS) distance of $\bar{s} = 1000\,\mathrm{m}$ and are then shifted in both x and y direction using normally distributed random variables with zero mean and standard deviation $\frac{\bar{s}}{8}$. For the mesh topologies, two nodes are connected by a backhaul link if their distance is less than or equal to $1.5 \cdot \bar{s}$, which has consistently produced fully connected but not unrealistically dense topologies. In case of a tree topology, we first define the most central node of the network as connected and then recursively connect the unconnected node being closest to the network's center to its closest connected and more central node.

Each node becomes a potential controller with a probability of P_C with processing power $p_{\text{own}} =$ 200 GFLOPS. All links are assigned the same fixed capacity of 2.5 Gbit/s and the latency for each link is determined by its length multiplied by 1.45 and divided by the speed of light, assuming an optical backhaul network.

To assign data flows to nodes, we assume mobile user equipment as origin and/or destination of the data flows for this evaluation scenario. The GreenTouch connectivity model [17] is used and each flow is connected to up to three nodes with the best connectivity until a certain Signal to Interference plus Noise Ratio (SINR) threshold is reached. A flow belongs to one out of three types of data flows, as shown in Table 2.1, whose data has been extrapolated from [11]. At last, the processing capacity requested by a data flow f is determined by

$$p_{\text{flow}}(f) = 4 \cdot b_{\text{flow}}(f) \cdot \sum_{v \in V} W_{f,v} \, \frac{\text{FLOPS}}{\text{bit/s}}.$$

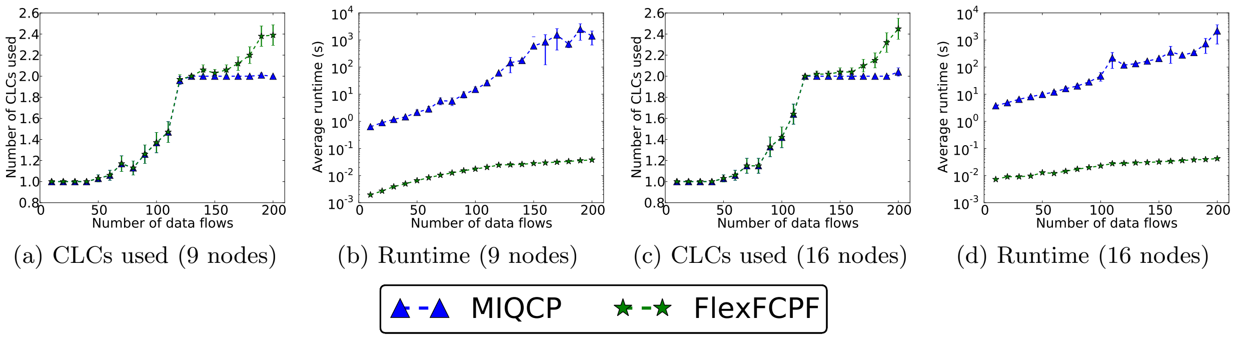
Finally, the CLC and CRC parameters from Table A.1 are chosen as follows: $b_{\text{CLC}} = b_{\text{CRC}} = 10$ Kbit/s, $l_{\text{CLC}} = 1$ ms, $l_{\text{CRC}} = 10$ ms, $p_{\text{CLC}} = p_{\text{CRC}} = 100$ KFLOPS.

MIQCP vs. FlexFCPF To compare the results of MIQCP and FlexFCPF, we have generated instances with 9 and with 16 nodes with $P_C = 1.0$, mesh topology and multiples of 10 data flows. These scenarios are big enough to see the important effects and small enough to run in reasonable time.

Table 2.1: Data flow types

type	probability	b_{flow}	l_{flow}
voice	0.5	500 Kbit/s	5 ms
video	0.4	1 to 4 Mbit/s	10 ms
data	0.1	1 to 20 Mbit/s	50 ms

In this first evaluation, all instances have been solved validly, only one CRC was used consistently, and all data flows have been satisfied. As can be seen in Figure 2.4a and Figure 2.4c, FlexFCPF performs just as well as MIQCP for fewer flows, before loosing a bit of ground around 100 data flows. Also, the number of CLCs used practically does not depend on the number of nodes in the network, as the resources needed for the control aspect are very small compared to the resources needed for the data flow processing. Meanwhile, Figure 2.4b and Figure 2.4d show that FlexFCPF operates about four orders of magnitude faster than MIQCP.



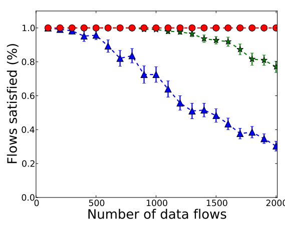
(a) CLCs used (9 nodes) (b) Runtime (9 nodes) (c) CLCs used (16 nodes) (d) Runtime (16 nodes)

Figure 2.4: Evaluation: MIQCP vs. FlexFCPF

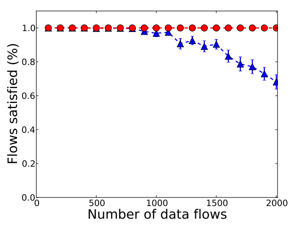
FlexFCPF in Large Scenarios To evaluate FlexFCPF for larger scenarios, we have generated instances with 25, 50 and 100 nodes, $P_C \in \{0.25, 0.5, 1.0\}$ and multiples of 100 data flows.

Again, all instances have obtained a valid solution and consistently only one CRC was used for each instance. However, not all flows could always be satisfied, especially for $P_C = 0.25$, as can be seen in Figures 2.5a to 2.5c. Next, Figures 2.5d to 2.5f show again that the number of CLCs used only slightly depends on the number of nodes in the network but more on the number of data flows, at least as long as sufficient potential controllers are available. Last but not least, Figures 2.5g to 2.5i give an overview on the runtime for our larger scenarios. As can be seen in Figures 2.5g and 2.5h, FlexFCPF struggles hard to satisfy as many data flows as possible in the scenarios where the resources are too sparse to satisfy every data flow. In addition, as revealed by Figure 2.5i, the runtime of FlexFCPF also depends on the amount of potential controllers in the network, which results in a larger solution space to be considered.

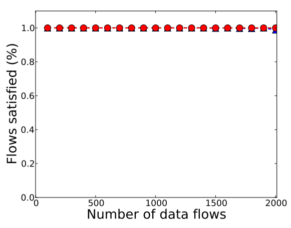
Flexible reassignment For this evaluation we have generated four networks and simulated the network operation over the course of 48 simulated hours using FlexFCPF. The generated networks have either 25 or 100 nodes and backhaul links with mesh or tree topology and are illustrated in Section A.2.3. When generating the nodes, each of them becomes a potential controller with a probability of $P_C = 0.6$.



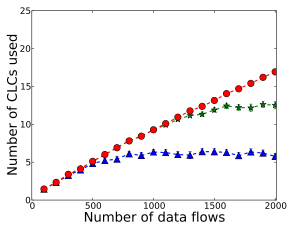
(a) Flows sat. (25 nodes)



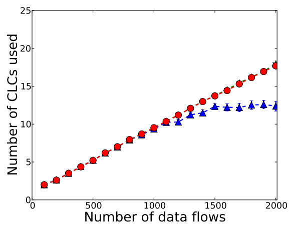
(b) Flows sat. (50 nodes)



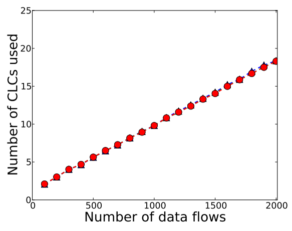
(c) Flows sat. (100 nodes)



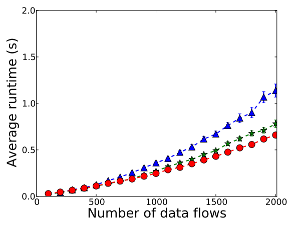
(d) CLCs used (25 nodes)



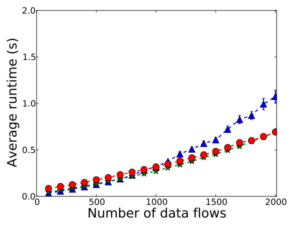
(e) CLCs used (50 nodes)



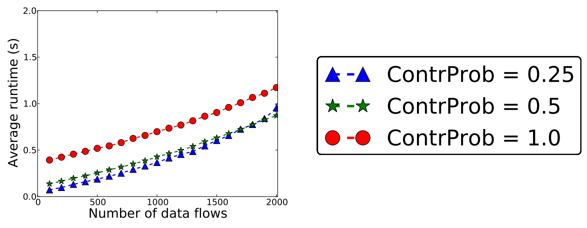
(f) CLCs used (100 nodes)



(g) Runtime (25 nodes)



(h) Runtime (50 nodes)



(i) Runtime (100 nodes)

Figure 2.5: Evaluation of the FCPF optimisation model

Incoming data flows are generated using a nonstationary Poisson process with $\lambda = |V| \cdot \text{loadlevel}(t)$, simulated using the thinning method [18]. Our daily load curve loadlevel(t), with t being the current time in seconds, is derived from [19] and shown in Figure 2.6. The duration of a flow is determined using an exponential variable with parameter 0.02 resulting in an expected flow duration of 50 seconds and thus an *approximated* expected number of data flows in the network of $50 \cdot |V| \cdot \text{loadlevel}(t)$ at time t.

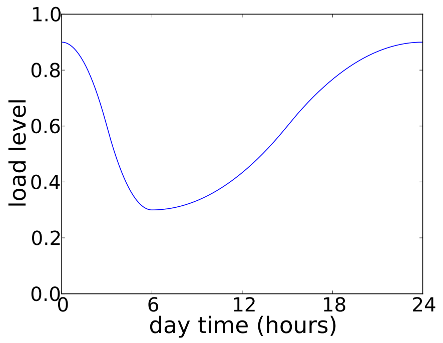
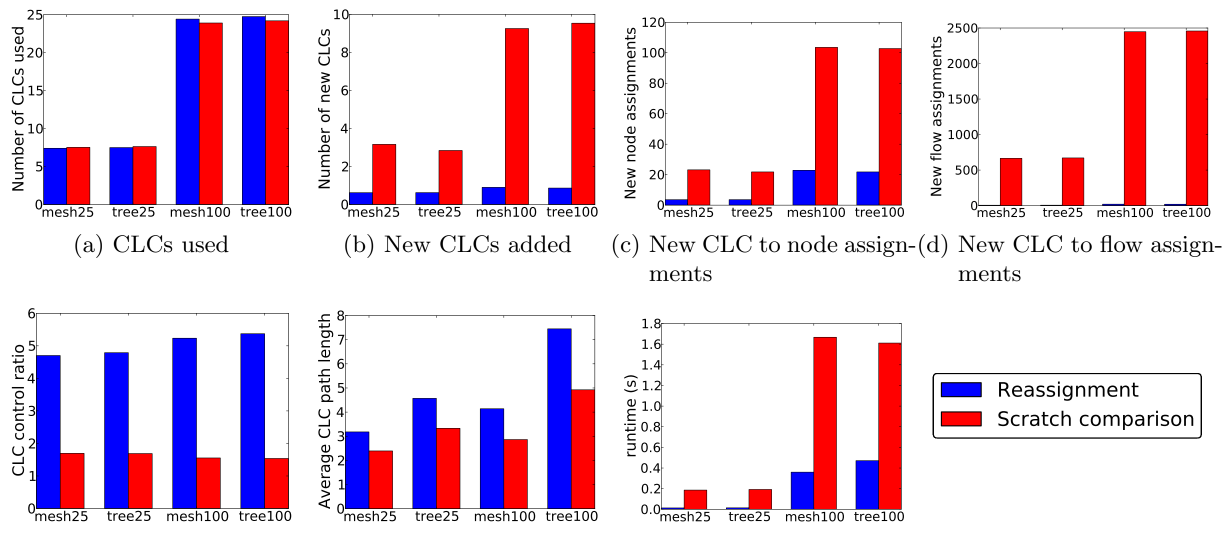


Figure 2.6: Evaluation: daily load curve

All simulations were launched at $t = -3600$ seconds to start the 48 hour network monitoring at $t = 0$ in a running state, skipping an initial transient. To generate the data for our evaluation, we have extracted the data from FlexFCPF each time the set of CLCs is modified in the course of reassignment. At these points, we further perform for comparison an initial controller placement on an empty copy of the current network.



(a) CLCs used (b) New CLCs added (c) New CLC to node assignments (d) New CLC to flow assignments

(e) Average CLC control ratio (f) Average CLC path length (g) runtime

Figure 2.7: Evaluation: FlexFCPF simulation vs. initial controller placement

Throughout all simulations, FlexFCPF has solved all networks with only one CRC, satisfying all data flows apart from a small exception posed by the 100 node tree topology where a few times a single flow could not be satisfied and more than one CRC had to be used for a short period of time during high load periods. We determined that this originates from the very sparse backhaul tree topology compared to the quite large network size.

Figure 2.7 summarises the most important data that we extracted from our simulation runs. First of all, Figure 2.7a displays the average number of CLCs used. We observe that the flexible reassignment is fully competitive with the from-scratch comparison considering the number of used CLCs, even though it is technically disadvantaged as it has to build up on an already existing controller placement.

Next, Figures 2.7b to 2.7d give an impression of the reconfiguration overhead caused by the newly calculated assignments. Figure 2.7b illustrates the average amount of newly added CLCs compared to the set of CLCs of the previous assignment. Similarly, Figure 2.7c and Figure 2.7d depict the average number of new CLC-to-node and CLC-to-flow assignments. As can be seen, the from-scratch comparison case is significantly outperformed by the flexible reassignment approach for all these metrics and as a result, the flexible reassignment saves an immense amount of reconfiguration overhead caused by establishing new assignments in the network.

Figure 2.7e and Figure 2.7f show a small downside of the flexible reassignment. Figure 2.7e illustrates the average *CLC control ratio* in the networks, i.e. the average amount of CLCs a node is assigned to. We recognise that the approach of finding a CLC for a specific flow pursued by the flexible reassignment (as described in Section A.2.2.2) leads to nodes being assigned to multiple CLCs, as generally not always the same CLCs are going to have available resources to satisfy data flows incoming at the same node. On the other hand, the global view of the initial controller placement generally provides a small CLC control ratio as a CLC is assigned to as many nearby nodes and its passing through data flows as possible. The average CLC path length data shown in Figure 2.7f signifies a natural continuation of this characteristic. Clearly, we have to accept longer routing paths with the flexible reassignment when no nearby CLC is able to satisfy an incoming data flow. In total, this means that the flexible reassignment has to devote more network resources for the network control compared to the from-scratch initial placement, taking away resources for processing data flows in theory. However, as we have seen earlier, the flexible reassignment still performs equally well for data flow satisfaction and for the number of CLCs used. We can explain this by observing that the resources required for the network control are significantly smaller than the resources required by the data processing.

Finally, Figure 2.7g gives an overview of the algorithm runtimes. Again, we see the flexible reassignment is clearly superior compared to the initial controller placement. Especially in crowded networks, we see a small runtime as a significant advantage.

2.1.2.4 Observations

In Deliverable [8] we have introduced the idea of flow processing-aware controller placement. Since, we have improved that approach and we have presented an efficient heuristic algorithm that solves the NP-hard flow processing-aware controller placement with a runtime that makes it suitable for real-world usage while providing close to optimal results. We have further extended it with the ability of performing flexible reassignment during network operation. As our evaluation shows, the flexible reassignment performs equally well compared to a new controller placement from scratch while saving a significant amount of reconfiguration overhead and runtime.

2.1.3 Dynamic Backhaul Reconfiguration

To handle the increasing wireless data rate demands of users in 5G mobile networks [11], it is not only important to efficiently utilise the available wireless resources to provide high data rates but also to coordinate wireless transmissions to reduce harmful interference. Such coordination mechanisms require a set of BSs that implement a coordination scheme to serve one or a group of User Equipments (UEs). We call this set of BSs a *Coordinated Base Station Set (CBS)*. Hence, as the coordination mechanisms are implemented in the wireless domain, the selection of BSs for the CBSs has to be decided based on the characteristics of the wireless channel [20, 21]. Implementing coordination among BSs is only possible if capacity and low-latency connections are provided by the backhaul network that interconnects the BSs. Thus, the constraints on the backhaul network have to be considered in addition to the wireless characteristics when implementing BS coordination mechanisms [22, 23, 24].

In Deliverable D3.2 [8], we have shown how dynamic backhaul reconfiguration can increase the feasibility of base station coordination in networks with limited backhaul network resources, but we did not take the new challenges from very dense wireless access networks into account. Thus, here we extend our dynamic backhaul reconfiguration approach to handle hotspot areas with a high density of users, i.e. a high density of desired CBS. We also show that this extension is beneficial for scenarios without hotspots and how it dynamically adapts to the density of hotspots.

2.1.3.1 Hotspot Algorithm

To deploy our algorithm in dense scenarios with hotspots of CBSs, we extend the heuristic algorithm from Deliverable D3.2 [8] by a CBS prioritisation mechanism. This mechanism is explained in detail in this section, together with a brief explanation of the whole algorithm in Appendix A.3.

The extension divides the desired CBS into two sets $\mathrm{W}_{\mathrm{hotspot}}$ and $\mathrm{W}_{\mathrm{normal}}$ before the other steps of the algorithm are executed. The calculations in this extension use two threshold values t_v, which determines how strict the filtering of hotspot vertices should be, and t_h, which determines how strict the filtering of hotspot CBS should be. The numbers for the threshold values depend on the scenario and can be determined using a parameter study.

The algorithm first calculates for each vertex v from the backhaul graph in how many CBS the vertex is present as:

$$h_v = \mathbb{1}_{v \in W_i}$$

$\mathbb{1}_X$ denotes an indicator variable with value 1 if condition X is true and value 0 otherwise. Each vertex v is considered a hotspot vertex if

$$h_v > |W| \cdot t_v$$

and is added to the set of hotspot vertices V_h. Now each CBS W_i is added to $\mathrm{W}_{\text{hotspot}}$ if

$$\frac{\sum_{v \in W_i} \mathbb{1}_{v \in V_h}}{|W_i|} \geq t_h$$

otherwise it is added to $\mathrm{W}_{\text{normal}}$.

Now the steps of the original algorithm for backhaul resource allocation have to be executed for each CBS.

2.1.3.2 Evaluation

In this section we present simulation results that show how our new algorithm performs compared to the algorithm from our previous work [8].

Hotspot Simulation Scenario A fixed number of BSs are placed on a regular grid, with a mean inter-BS distance of $\bar{s} = 1000\,\text{m}$ (urban scenario [25]), and are then shifted in both x and y direction according to two independent, normally distributed random variables with zero mean and standard deviation $\frac{\bar{s}}{8}$.

The backhaul network is created as a mesh topology where two BSs are connected by a link if their distance is less than or equal to $1.5 \cdot \bar{s}$. This value produces a partially connected mesh network; smaller or larger values result in too sparse or too dense topologies, which are not realistic. All links in the backhaul network are assigned the same set of available wavelengths $K = 4$ and each wavelength is assigned the same fixed capacity of 2.5 Gb/s. The latency for each link is determined by the distance multiplied by 1.45 divided by the speed of light as we are modeling an optical backhaul network.

In order to create a hotspot of CBSs, a x,y coordinate is selected uniformly as the hotspot center. Now a fraction h of all CBSs is placed around the hotspot center based on a normal distribution with the hotspot coordinate as the mean and standard deviation $\frac{\bar{s}}{4}$ as the desired hotspot CBSs. All other desired CBSs are generated by placing then uniformly at random on the plane covered by the placed BSs. The BSs which are considered as the CBS are all BSs located inside a circle around the coordinates of the CBS with a given radius. We determine this radius by multiplying the mean inter-BS distance with a factor $r = 1.5$, which results in 5 BSs per CBS on average.

The capacity demand d for each BS in the CBS is set to the same value and is either 0.625 Gb/s, 1.25 Gb/s or 2.5 Gb/s. This implies that at a demand of 2.5 Gb/s one complete wavelength is required to connect a BS to the controller.

To determine the threshold values for the CBS prioritisation t_v and t_h, we performed a parameter study and identified $t_v = 0.1$ and $t_h = 0.9$ as the best values to maximise the number of feasible CBSs in this scenario.

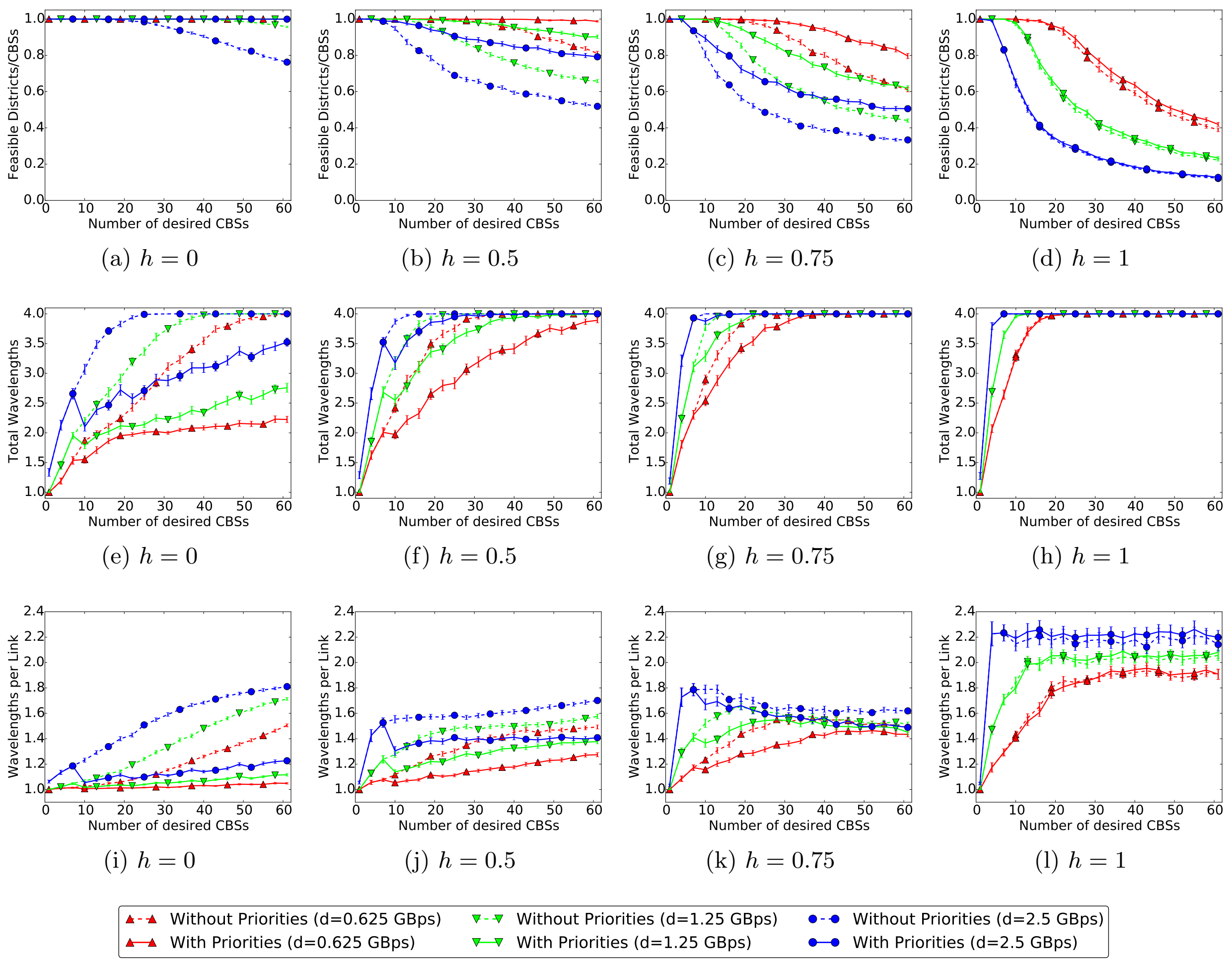


Figure 2.8: CBS prioritisation simulation

Hotspot Simulation Results The results for the simulation are shown in Figure 2.8, where we investigate the influence of different hotspot CBS fractions h. For $h = 0$ no hotspot CBSs exist and for $h = 1$ all CBSs are hotspot CBSs. Solid lines are our new algorithm with the CBS prioritisation step, dashed lines the algorithm from our previous work as a reference.

In Figures 2.8a to 2.8d we show the resulting feasibility of the CBS, i.e. the fraction of desired CBSs that were successfully established. Even for a scenario with no hotspots ($h = 0$) the CBS prioritisation step increases the CBS feasibility to 100% for all capacity demands and all numbers of desired CBSs. In the scenarios with $h = 0.5$ and $h = 0.75$, the feasibility drops as the number of desired CBSs increases. Still the CBS prioritisation step increases the CBS feasibility by over 20%. In the scenario with all desired CBSs in the hotspot ($h = 1$), there is no significant improvement from prioritising CBSs, which is the expected outcome.

Figures 2.8e to 2.8h show the results for the overall number of used wavelengths and Figures 2.8i to 2.8l show the results for the number of used wavelengths per link. In the scenario with no hotspot CBSs ($h = 0$) the CBS prioritisation significantly improves the efficient use of wavelengths, as both the total number of used wavelengths and the number of wavelengths per link are significantly lower when using the CBS prioritisation. This effect is also evident in the scenarios with $h = 0.5$ and $h = 0.75$. For $h = 1$ there is again no significant difference.

Non-Hotspot Scenario Extended Results Because of the indicated improved performance of the algorithm in a non-hotspot scenario, we investigated this scenarios further and performed the same evaluation as in Deliverable D3.2 [8] where we evaluated the performance of the heuristic algorithm in large scenarios. Again we a mesh topology scenario with 36 BSs and show results for 2 (solid lines) and 4 (dashed lines) available wavelengths per link. In the previous results [8] we had shown how the heuristic algorithm without CBS prioritisation efficiently assigns wavelengths in the *reconfigurable case* with 4 available wavelengths compared to the *baseline case* with 2 available wavelengths. Here we evaluate the influence of the added CBS prioritisation.

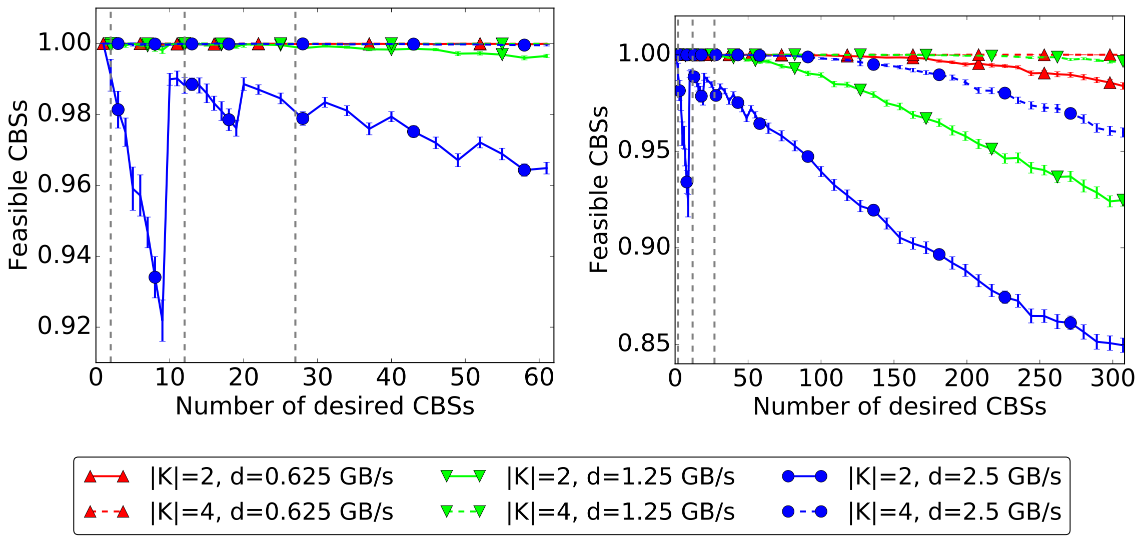


Figure 2.9: Feasible CBSs

Figure 2.9 shows the new results for the number of feasible CBSs: in the left plot zoomed in to 1 to 61 desired CBSs as in the previous evaluation and in the right plot for the full range of up to 300 desired CBSs. There are two major differences to the previous evaluation: First, especially when looking at a range of 1 to 61 desired CBSs, the lines are less smooth with a notable jump around 9 desired CBSs, especially for $d = 2.5$. This effect is a result of the CBS prioritisation step because the prioritisation of CBSs always happens with respect to the total set of CBSs. With a small set of total CBSs without explicit hotspot CBSs the algorithm does not always prioritise CBSs in the best way and the CBS prioritisation provides no advantage yet. Second, and more importantly, the number of feasible CBSs is significantly improved compared to the previous evaluation without CBS prioritisation. Without CBS prioritisation the number of feasible CBSs started to decrease below 1 for $d = 0.625$ around 60 desired CBSs. With CBS prioritisation, this decrease occurs at around 300 desired CBSs. For $d = 1.25$ and $d = 2.5$ there is a similar improvement and CBS prioritisation improves the number of feasible CBSs by a factor of 5.

The plots again contain vertical dashed lines indicating tipping points where the algorithm starts to assign more than 2 wavelengths if there are $|K| = 4$ wavelengths available per link.

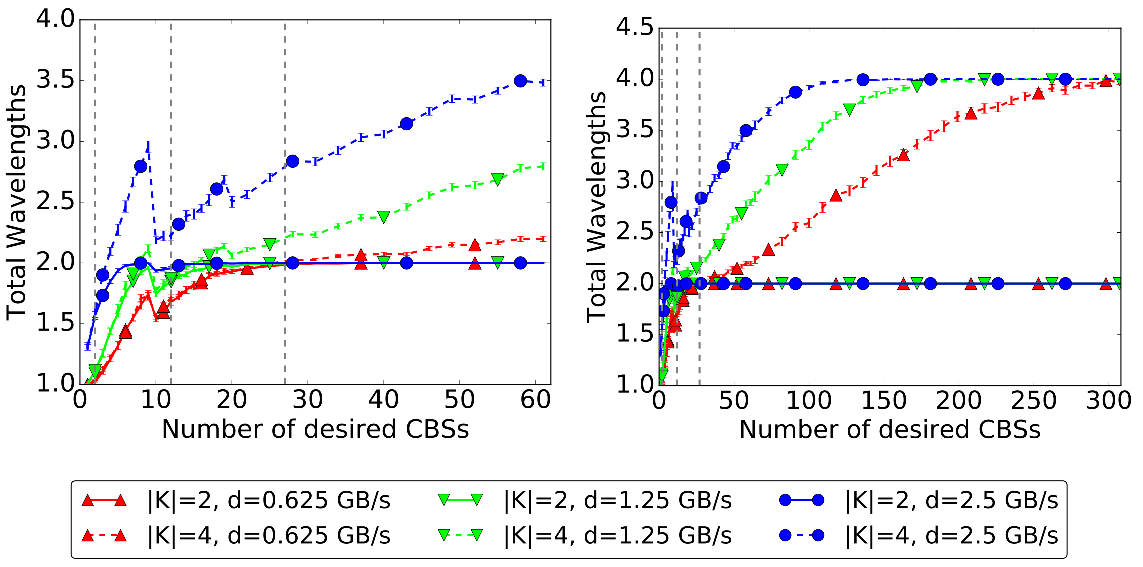


Figure 2.10: Total Wavelengths

The results for the used total wavelengths are shown in Figure 2.10 and the results for the used wavelengths per link in Figure 2.11. The tipping points (vertical dashed lines) also identified in the previous evaluation exist again at around 2, 12, 27 desired CBSs, respectively for $d = 2.5$, $d = 1.25$ and $d = 0.625$. At these tipping points the capacity with $|K| = 2$ available wavelengths is not sufficient to establish all desired CBSs and the algorithm starts to assign more than 2 wavelengths.

The notable jump at around 9 desired CBSs is also visible in the results here for all values of d and is again due to the CBS prioritisation not working advantageously with a small total set of CBSs. Apart from that, the results are consistent with the previous results: the algorithm does not assign additional available wavelengths until the capacity demands have increased so much that additional wavelengths are required to maintain the feasibility of BSs coordination, i.e., keep the number of feasible CBSs at 1. Again, after the tipping points wavelength are assigned gradually.

Figure 2.12 show the backhaul power consumption based on the power consumption model by Grobe et al. [26] as a comparison to the previous results in Deliverable D3.2 [8]. Again the reference power consumption of a full, static assignment of all wavelengths are indicated by horizontal, dashed lines, the lower one for $|K| = 2$ and the higher one for $|K| = 4$. The results for the backhaul power consumption are consistent with the other results: with CBS prioritisation the wavelengths are assigned more efficiently and thus the power consumption in the range of 1 to 61 desired CBSs is lower compared to the previous results without CBS prioritisation. For the full range of 1 to 300 desired CBSs, the slope and final value of the backhaul power consumption is similar to the previous results without CBS prioritisation. Thus, CBS prioritisation does not have a negative influence on the energy consumption of the approach for backhaul network reconfiguration.

2.1.3.3 Observations

Our simulation shows that our extension enables the use of our approach in *dense* wireless access networks with hotspots of users, and furthermore increases the feasibility of base station coordination in networks without hotspots. This is a significant improvement compared to our previous work [8].

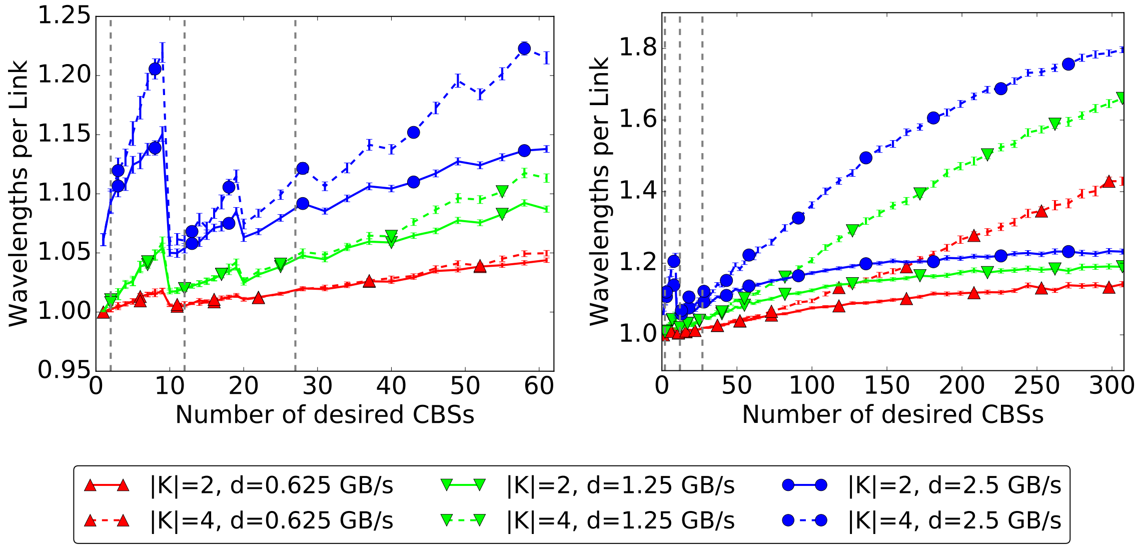


Figure 2.11: Wavelengths per Link

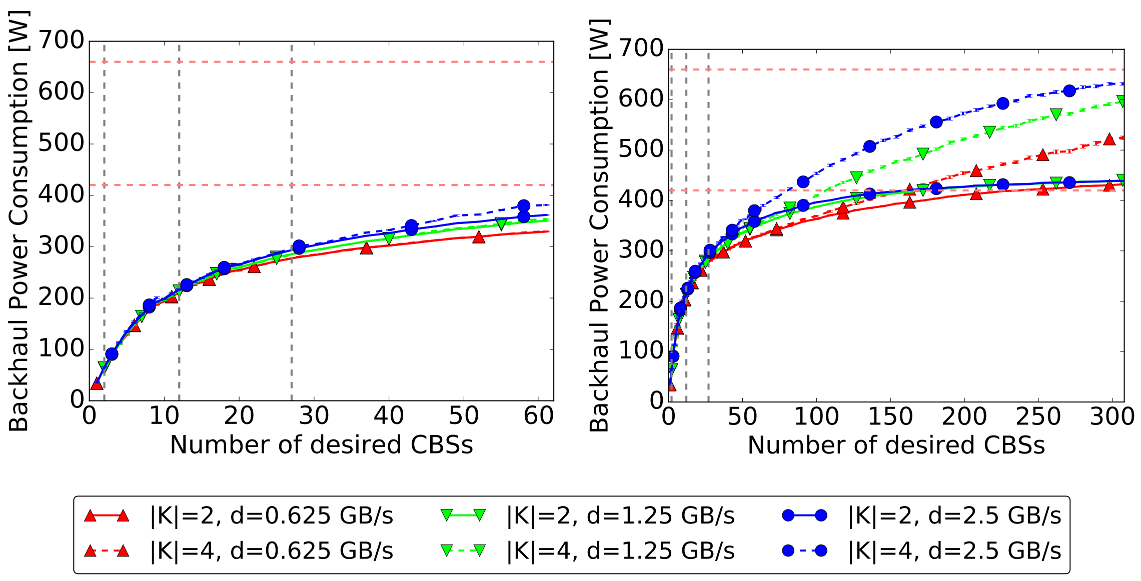


Figure 2.12: Backhaul Power Consumption

2.2 Case Studies

In this section we present the results from the case studies from Task 3.3.

2.2.1 Control of LTE and WiFi Direct relay by coordinating centralised ABSF and D2D clustering strategies

2.2.1.1 Background

In D3.2 [8], we have proposed the control of LTE cells and WiFi Direct relay to cope with the extremely densification of the wireless network in order to improve the system performance for both efficiency and user quality of experience. In particular, on the one hand we focused our research on inter-cell interference coordination ICIC solutions in order to limit the impact of the neighbouring cells while, on the other hand, we have considered the possibility to offload cellular traffic to WiFi technologies by means of opportunistic schemes. The compound effect was analysed and deeply studied, showing how independent approaches may be combined to boost system performance.

Here, we want to point out how all existing proposals, focusing on WiFi offloading techniques, do not take into account that the WiFi resources are shared and, thus, have very limited application. In fact, from this perspective, offloading to WiFi can experience serious congestion and become the real performance bottleneck of the network. Therefore, we show how to evaluate analytically the capacity region of WiFi devices in dense networks, providing some clue on how to jointly drive ICIC and offloading decisions in the cellular network.

2.2.1.2 LTE cells with WiFi Direct opportunistic relay

We assume a multi-cell LTE network with frequency reuse 1, with N users placed in the system. We analyse the downlink channel communication even though an extended approach can be applied on the uplink direction. Fig. 2.13 illustrates the network architecture, which is tailored for an efficient SDN-based approach to the management of network elements. We provide details of each component of our proposal showing how they are jointly analysed in our study.

ABSF control. From an SDN point-of-view, the CLC directly communicates through a Northbound interface with stand-alone applications, in charge of taking decisions for the local part of the network. Amongst different application, we focus on the *Almost Blank Sub-Frame (ABSF) Application.* Based on the current user channel information and the user traffic load, the ABSF application enables each of the eNBs to transmit on a particular set of subframes (defined as ABSF patterns). The algorithm behind the ABSF application is based on Base Station Blanking (BSB), defined in [27]. Every 200 LTE frames (namely a "decision period"), the CLC collects user and channel information and issues the correct ABSF pattern to each eNB through a Southbound interface (i.e., through a modified version of the X2 interface of LTE).

D2D control. The CLC also handles the cluster formation for Device-to-Device (D2D) clustering. The CLC collects the information regarding the channel quality and location of the UEs and uses the *Merge & Split* algorithm [28] to form clusters throughout the network. In the process of cluster formation, the CLC takes into account the distance of the cluster members from each other and the achievable throughput gain after clustering. Moreover, we assume that users are willing to activate D2D functionalities with a probability γ. In each cluster, the members can send/receive the cellular data through the cluster head, i.e., the node with the highest channel quality towards LTE eNB. Note that each cluster member can potentially become a cluster head once its LTE channel quality is the highest of its cluster. However, the cluster head selection decision is renewed only when the CLC issues a new ABSF pattern. Cluster members use WiFi Direct as the platform for intra-cluster communication. Since the eNB is aware of clustering decisions whenever a packet is destined to a cluster member, the eNB simply transmits the packet to its corresponding cluster

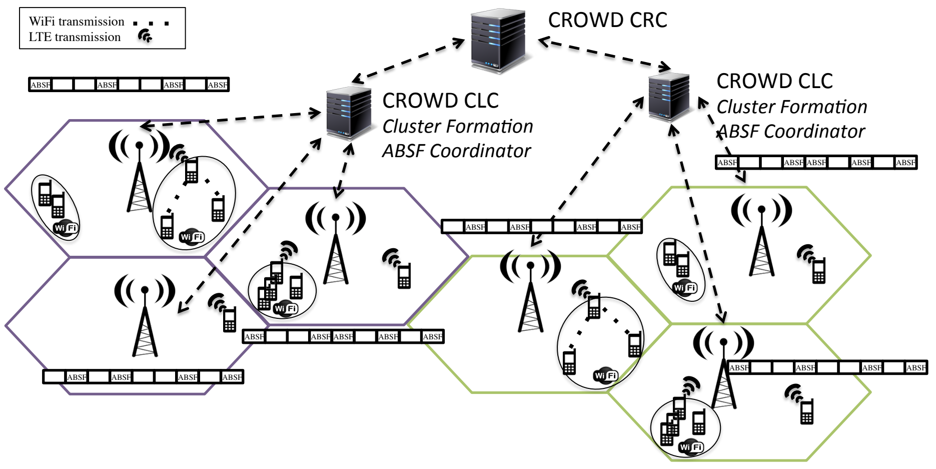


Figure 2.13: Network architecture.

head. This maximises the system throughput and improves the spectral efficiency by using higher Modulation and Coding Schemes (MCSs). In order to quantify the amount of traffic that each cluster head can relay in the outband bandwidth, for every decision period, the CLC allocates cluster-heads to non-overlapping WiFi channels. We assume that the scenario is dense, but also sufficiently small so that all cluster heads scheduled on the same channel contends for the same resources, affecting each other's performance.

D2D rates. Exploiting the knowledge of cluster head channel qualities, the CLC is able to evaluate the set of achievable rates by using the Coupled Processors System (CPS) modeling [29]. To make ABSF pattern decisions, the CLC does not take into account the demand of users or cluster heads. Instead, it considers the traffic load effectively relayed in each cluster by any selected cluster head. In this way, eNBs avoid wasting cellular resources for traffic that could not be relayed on time by cluster heads. Moreover, among all possible rates that could be selected (e.g., within the stability region of D2D transmissions, computed with the CPS model), the CLC selects the rates by optimising proportional fairness function, amongst per-user throughputs. The approach followed in here is the same as in [29].

2.2.1.3 WiFi Offloading: realistic evaluation

Due to fast emergence of wireless technologies and services, wireless networks have become very dense. This huge densification results in severe performance issues due to high interference and low efficiency in the utilisation of wireless resources and power. Most of the current research efforts address these issues in isolation which leads to ignoring the impact of other factors and taking simplifying assumptions. In contrast, in dense networks, inefficiencies are highly correlated and these solutions must be evaluated concurrently. However, the concurrent evaluation can become very challenging with realistic assumptions that take into account the capacity of different technologies (e.g., LTE and WiFi Direct) and their impact on each other's performance. Moreover, a deep analysis allows us to evaluate the impact of these solutions as one and determine whether the achievable gain justifies the resulting complexity or not.

A major problem in dense networks roots from the interference. On the one hand, in LTE the major source of interference is due to neighboring cell activity, a.k.a. Inter-Cell Interference (ICI). A common approach to mitigate ICIC is to apply a Third Generation Partnership Project (3GPP) recently standardised mechanism, called Almost Blank SubFrame (ABSF). ABSF coordinates different LTE eNBs by preventing their transmissions for a specified set of subframes (ABSF pattern), where only control signals are permitted. Several works focus their research on how to properly design ABSF patterns to limit ICI and boost the network performances, especially in high-density scenarios. On the other hand, the low spectral efficiency is an open issue that is commonly observed in the dense networks. However, researchers found the means to leverage the densification of today's wireless network to improve the spectral efficiency through a popular branch of cooperative communications, namely D2D communication. Some of proposed works suggest to form clusters among cellular users over a secondary interface (e.g., WiFi Direct or LTE Direct) and use the cluster member with the highest cellular channel quality for relaying traffic to the users with lower cellular channel quality. The results obtained from numerical and empirical simulations illustrate that the clustering proposal achieves significant throughput gain in comparison to the legacy cellular system when only a small set of users have D2D communication possibility.

While both ICI reduction schemes and spectral efficiency enhancement solutions have shown promising outcomes, they have not been evaluated in a real-world scenario concurrently. In order to give some insights on the joined effects of the aforementioned techniques, we propose to simultaneously evaluate the impact of two common used techniques, namely, the one described in [27] and D2D clustering [28].

Although the rate achievable with WiFi is higher than with cellular technologies of the same generation, WiFi transmissions mainly suffer from poor coordination of transmitters, exhibiting high performance degradation in very-dense scenarios. Therefore, here we evaluate the integration of different techniques, such as ABSF and D2D clustering, in a realistic set-up, with the intention to quantify the benefit of using *outband* (WiFi) transmissions to enhance spectral efficiency in cellular access networks (LTE). Therefore, we are compelled to say that WiFi will act as the bottleneck of cellular transmission in the future network design, and the impact of the fact that WiFi is a contention-based mechanism operating in unlicensed spectrum is often underestimated.

Several works investigated the set of achievable rates in WiFi networks when the system is not working in saturation condition, i.e., when the WiFi transmitters have infinite backlog and they always have a packet to transmit. Among the available approaches, in order to characterise the performances of the WiFi communication channel in any traffic condition, we leverage a new modelling technique by means of a queuing model, called CPS. In particular, parallel queues of backlogged traffic experience different serving rates that vary over time, depending on the activity of the other queues. As a result, the service rate of each of the queues in a CPS is strongly coupled with the activity of the other queues, similarly to the throughput achieved by WiFi users, which is strongly coupled to the number of stations attempting to access the channel. Therefore, the real advantage of such approach allows to easily impose a set of achievable rates for the WiFi transmitters, through basic rate limiting techniques, that optimise the use of the system resources given a particular utility function.

2.2.1.4 Numerical results

We prove the benefit of using simultaneously a D2D clustering technique and the BSB algorithm in a realistic setup with consideration of achievable WiFi capacity by means of numerical results. In order to quantify the actual gain achieved, we use numerical simulations on a particular dense scenario, and we compare the results attained against a set up that uses BSB or clustering in standalone fashion, or against a set up that does not exploit any of the mentioned approaches.

The evaluation scenario consists of a seven-cell network and, when clustering is in place, three different WiFi channels, i.e, three different collision domains. On the WiFi channels, the Medium Access Control (MAC) layer used is 802.11n on the 2.4 Ghz bandwidth. The demand of each of the users is fixed and equal to 1 Mb/s in any of the evaluation scenarios. We assume that users do not move and do not leave during the time window of the simulation. As a result, when formed, the cluster formations do not change over time, while the cluster head choices do change, according to the variation of the channel quality over time. When clustering is in place, the CLC decides the

Table 2.2: The parameters of the evaluation of D2D opportunistic outband relay schemes in LTE

Parameter	Value
Scenario	Circular area, radius 50 m
Cellular uplink bandwidth	20 MHz
eNB TX power	27 dBm
Thermal noise power	-174 dBm/Hz
Cluster head election time T	every 2 s
Fading, shadowing, pathloss	Reyleigh, 9 dB, UMa [30]
WiFi cluster head TX power	20 dBm
WiFi bandwidth	20 MHz

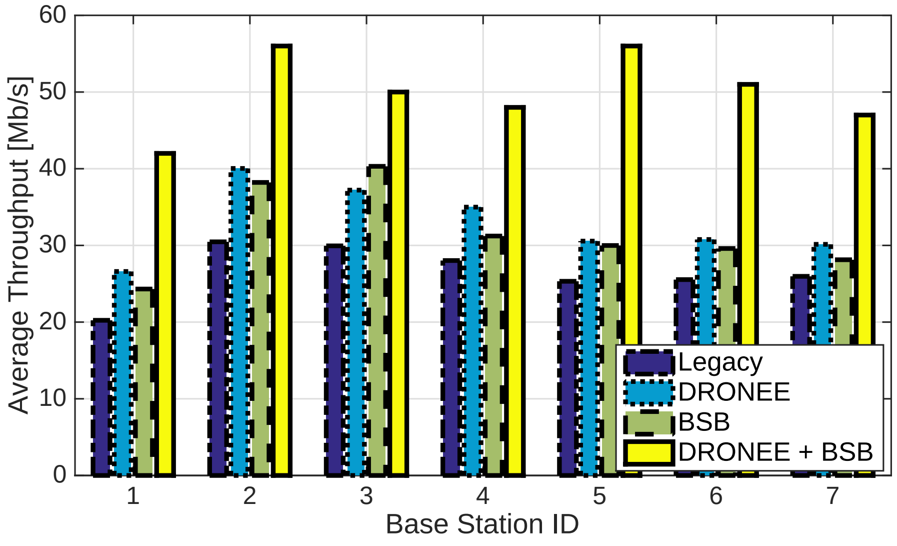


Figure 2.14: Average throughput comparison for different schemes in a network with 500 users and 7 base stations.

ABSF patterns considering the cluster head as a single user with an aggregate traffic load obtained as the sum of the demand of the cluster head itself (1 Mb/s) plus the total demand required by users in the cluster.

The simulation parameters are chosen based on the values suggested by the ITU-R guidelines for evaluating IMT-Advanced networks, see Table 2.2.

Fig. 2.14 shows the average throughput achieved by each of the 7 base stations regularly distributed over the simulated area, after adopting one of the 4 different schemes in a network with 500 users. For D2D-based clustering, base stations adopt a weighted round robin policy and assign resources to cluster heads with weights proportional to the cluster sizes. Interestingly, we can note that each of BSB or Dual-radio opportunistic networking for energy efficiency (DRONEE) provides a good level of gain in terms of throughput in comparison to conventional cellular networks. Those single improvements are due to efficient ICI management of BSB as well as high spectral efficiency of DRONEE, which exhibits a higher spectral efficiency when taken in isolation. However, we want to highlight the significant improvement shown in the case of DRONEE working in parallel with BSB. In that case, even though WiFi connections act as bottleneck due to real data rate limitations, BSB boosts the network capacity bringing up to 100% more throughput than legacy scheme while 75% of gain with respect to single enhancement approach. This still validates our preliminary findings, already reported in D3.2 [8], proving that a compound effect study of applying two independent enhancement approaches leads to a huge gain in term of spectral efficiency. In addition, we show in Fig. 2.15 the cumulative distribution function of the average data rates achieved by each user placed in the network (either clustering user and cluster head users). We can

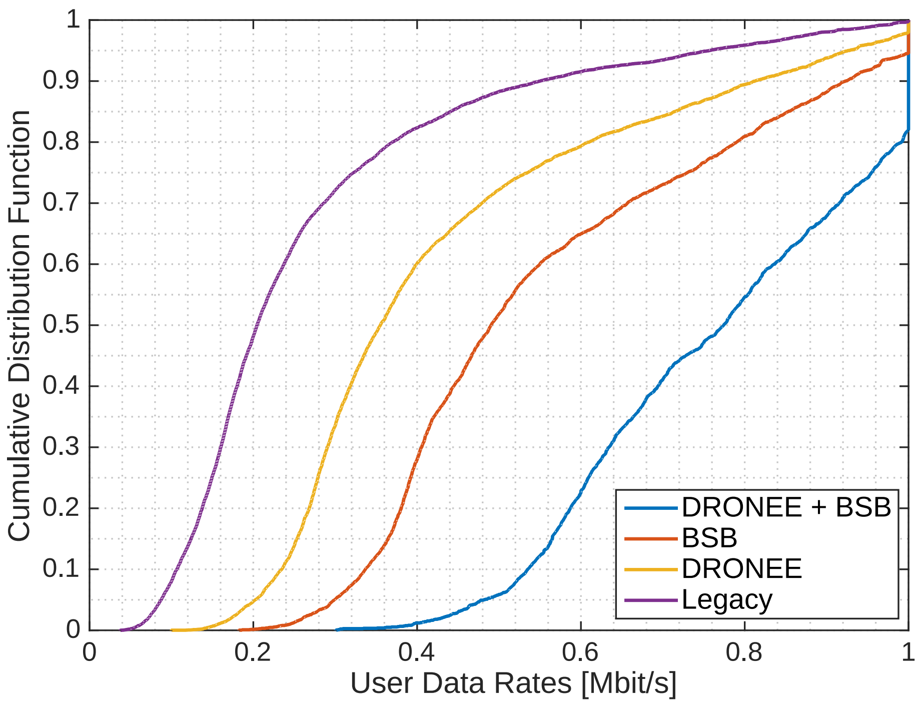


Figure 2.15: Cumulative distribution function of user data rates (both clustering users and cluster head users) for different schemes in a network with 500 users and 7 base stations.

easily see that applying a joint approach will also benefit the user fairness, as data rates are almost uniformly distributed based on the channel condition experienced by each of the interested users. As expected, the curves used for BSB and DRONEE exhibit lower data rates compared to the ones shown for the joint approach. However, the gain stemming from the coupled control of BSB and DRONEEis not equivalent to the sum of the gain provided by BSB and DRONEE clustering because both techniques rely on wireless channel diversity to improve the performance. Therefore, the diversity gain of clustering after applying ICIC via ABSF patterns with BSB is lower because channel quality of users increases due to lower interference (i.e., the opportunistic scheduling gain is lower).

2.2.2 Infrastructure on Demand for 802.11 networks

2.2.2.1 Background

In D3.1 [7], we analysed how the delays when activating the infrastructure (in particular, the AP of an 802.11 WLAN) have a significant time on the performance of an Infrastructure on Demand (IoD) scheme (we recently presented these results in [5]). In D3.2 [8], motivated by these findings, we further explored via experimental measurements the times required to vary between the different states a device may support (e.g., completely off, active, idling) and the corresponding power consumption. Given that the actual devices differ in both of these characteristics, i.e., time to vary between states and power consumption, the optimal point of operation of a network (based on a given criterion for the inherent performance vs. consumption trade-off) depends on the hardware considered, we also designed in D3.2 a Testbed for IoD experimentation. The operation of this testbed is based on OpenFlow [31], extended through the set of primitives and functionality that we describe next and presented in [6].

2.2.2.2 Node discovery and status

One first challenge is to maintain the list of nodes that compose the network, which in our case appear and disappear more frequently than in “traditional” (wired) networks, due to users’ mobility and wireless propagation. For the case of nodes connected to the network, i.e., APs or eNBs, they just have to establish a connection to the default OpenFlow transport protocol `6653` via TLS or TCP, and then perform the usual `OFPT_HELLO` exchange. For those nodes that can be (de)activated, we need to extend the database to announce this feature, which we do via the `reserved` field in the `OFPT_FEATURES_REPLY` message.

In WiFi networks there are at least three mechanisms to detect when a link towards a mobile terminal is available: (*i*) the `probe request` messages the mobile periodically sends on different channels to detect known or new APs, which can also trigger the presence of a new node that is duly reported to the technology dispatcher; (*ii*) passive scanning mechanisms; and (*iii*) the `Neighbour Report` message exchange from the recent 802.11k amendment, already supported by new devices such as e.g. iPhones, which is used by mobile terminals to learn about APs in the surroundings, and by APs to gather measurements about the quality of the channel towards other APs, which are then reported to the network.

Note that the usage of the links carrying data can be easily tracked with per-flow `counters`. By periodically polling these counters with read operations, the database can identify which links are becoming congested and which ones are being under-occupied and could support more traffic. This can then trigger the corresponding optimisation. We note that an alternative implementation could be based on the use of per-flow `meters`; indeed, by specifying `meter bands`, we can trigger an action when certain thresholds are passed.

2.2.2.3 Node (de)activation

Energy-efficient operation of a network requires the ability to power on and off resources as required. Accordingly, our testbed has been designed to provide such support. For the corresponding set of operations, we cannot rely on or extend the default OpenFlow primitives, and therefore we need to specify new primitives. To do this, we rely on the “experimenter” symmetric messages (`OFPT_EXPERIMENTER`), which are used for those nodes that upon registration announced that they supported deactivation, in addition to the time it takes to switch between states (so the the controller can preemptively activate resources) and the corresponding power consumption (to duly optimise energy consumption).

Following the above, we extend the OpenFlow set of primitives with a pair of commands, `switch_on` and `switch_off`, to power on and off (respectively) a given node. As we will describe in our testbed, these primitives can be used even when nodes do not support a sleep state, but are connected to e.g. a switched rack Power Distribution Unit (PDU), which can be remotely controlled via a network connection.

2.2.2.4 Network controlled handovers

Changing the point of attachment of a WiFi node while providing a seamless experience results is very challenging, given that in 802.11 (*i*) the mobile terminal typically selects its “best” point of attachment, and (*ii*) the signalling for carrier-grade operation is relatively complex. Still, thanks to the recent 802.11v and 802.11r amendments, this can be achieved with minor disruption of the service (as illustrated in our performance evaluation). Fig. 2.16 illustrates a simplified version of the signalling occurring on the access network. Next, we describe how these interactions support the changing of the point of attachment when triggered by the OpenFlow command.

Following Fig. 2.16, when a terminal powers on, it authenticates and associates with the best AP (i.e., AP1), and performs the `Neighbour Report` exchange to learn about the APs in the vicinity

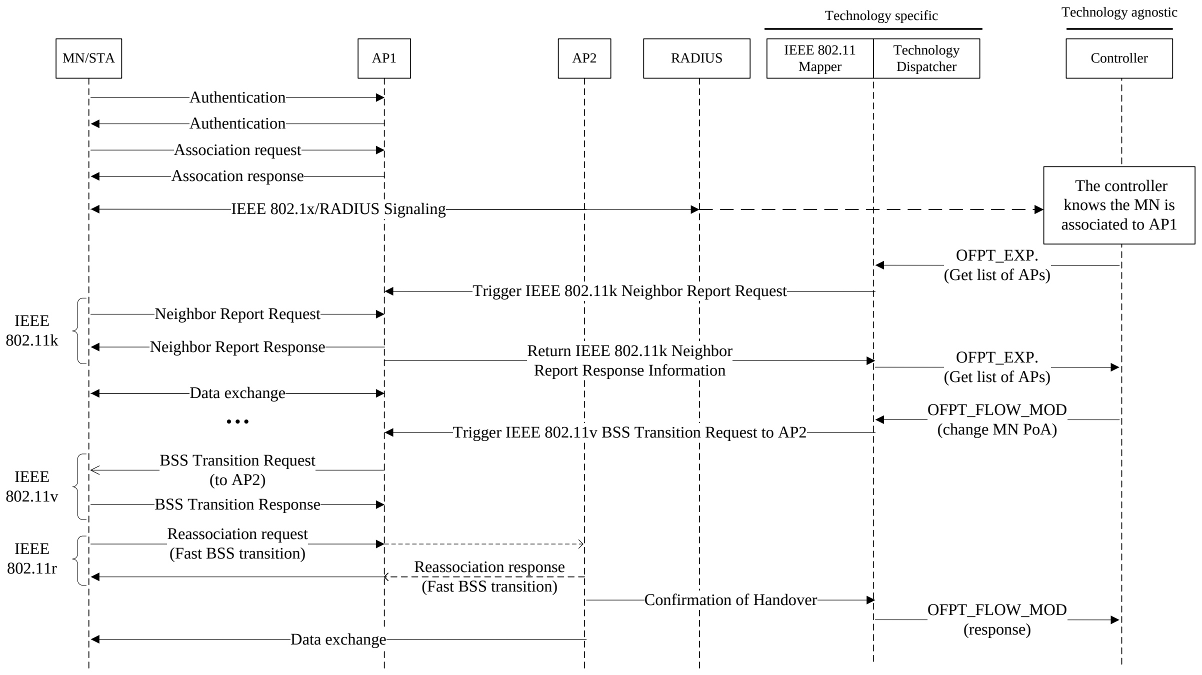

Technology specific
Technology agnostic
MN/STA
AP1
AP2
RADIUS
IEEE 802.11 Mapper
Technology Dispatcher
Controller
Authentication
Authentication
Association request
Assocation response
IEEE 802.1x/RADIUS Signaling
The controller knows the MN is associated to AP1
OFPT_EXP. (Get list of APs)
Trigger IEEE 802.11k Neighbor Report Request
IEEE 802.11k
Neighbor Report Request
Neighbor Report Response
Return IEEE 802.11k Neighbor Report Response Information
OFPT_EXP. (Get list of APs)
Data exchange
...
OFPT_FLOW_MOD (change MN PoA)
Trigger IEEE 802.11v BSS Transition Request to AP2
IEEE 802.11v
BSS Transition Request (to AP2)
BSS Transition Response
IEEE 802.11r
Reassociation request (Fast BSS transition)
Reassociation response (Fast BSS transition)
Confirmation of Handover
OFPT_FLOW_MOD (response)
Data exchange

Figure 2.16: Simplified handover operation.

belonging to the same network. During these operations, the APs that overhear the node activity report on the different links available to the mobile terminal, and update the database accordingly (among these, AP2 in the Figure). Assuming at some point that the optimiser decides that it is better if the Mobile Node (MN) associates with AP2, the controller issues a `OFPT_FLOW_MOD` primitive to change the (only) default forwarding entry of *MN*, from node *AP1* to node *AP2*. This primitive is processed by the 802.11 module with the controller, which issues a command to AP1 to trigger the 802.11v `BSS Transition Request` message, so the MN re-associates with AP2. Thanks to the use of 802.11r, this re-association is noticeably shorter than the original one. As we will see in the next section, the controller can duly issue other OpenFlow primitives to minimise the impact on performance.

One of the benefits of an OpenFlow-based architecture is that it supports the implementation of a variety of mobility protocols. For simplicity, we decided to implement the following sequence of commands whenever the controller decides to change the point of attachment of a MN: (1) activate at the switch *bicasting* of traffic between the Correspondent Node (CN) port and the ports the APs are connected to; (2) issue the 802.11v BSS Transition Request; (3) wait until the handover is completed, and inform the controller accordingly; and (4) stop the bicasting of traffic at the switch and re-configure the forwarding tables accordingly.

2.2.2.5 Testbed deployment

To validate the feasibility of our approach, we have prototyped our architecture in a small testbed that includes the technology specific modules for WiFi. The testbed, deployed in an office setting, is depicted in Fig. 2.17 and consists of one controller, an OpenFlow switch, two APs, two MNs, a CN to support traffic generation, and a switched PDU to support the (de)activation of APs via HTTP requests.

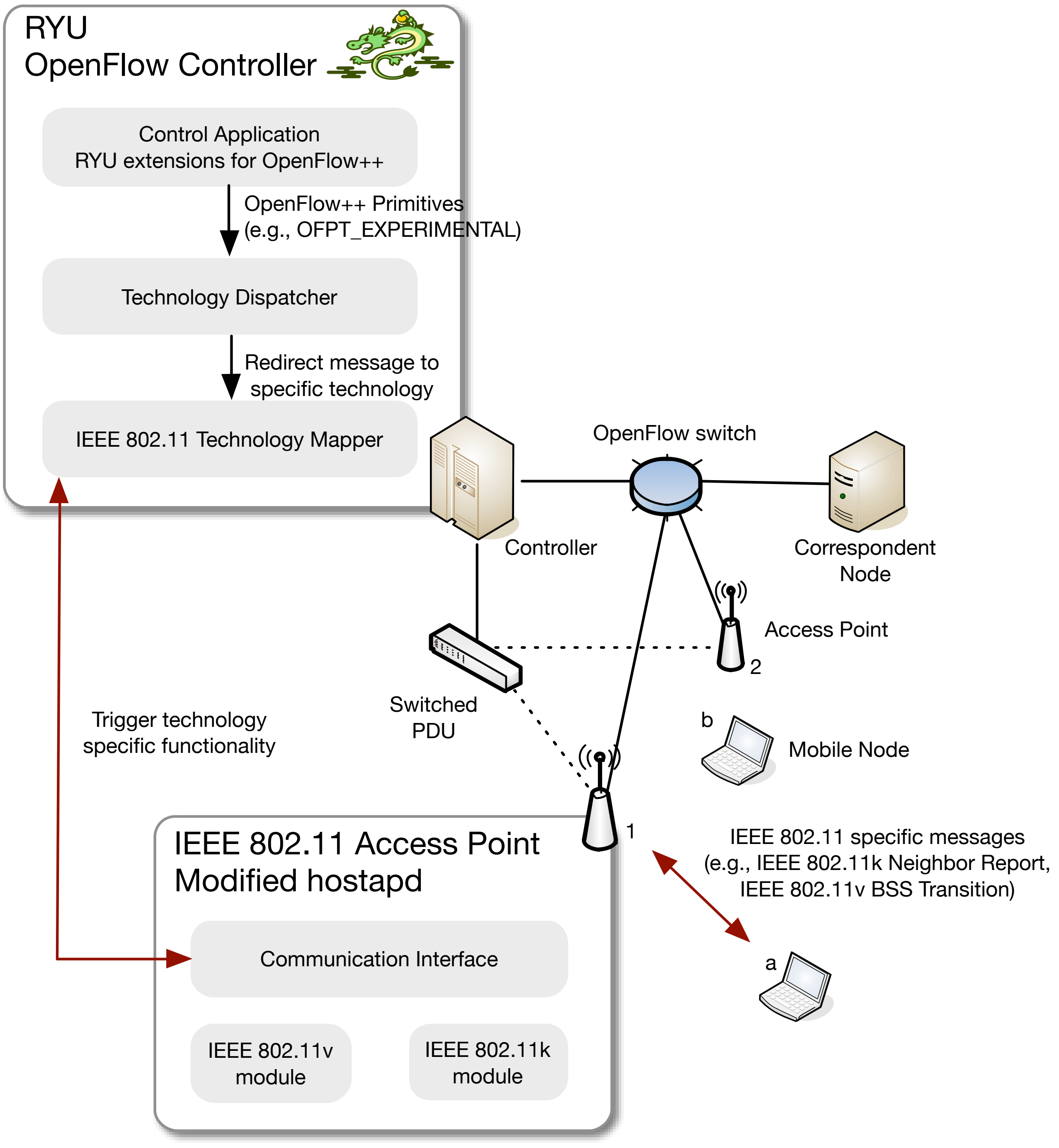


Figure 2.17: Deployed testbed and implemented modules and interfaces.

The controller is a desktop machine running Linux and the `Ryu` SDN framework,[1] which we extend to support the proposed `OpenFlow++` API, and two Python libraries: one to map the `switch_on/off` commands to the switched PDU, and another one for the 802.11-specific mechanisms. The controller also runs a `MySQL` database[2] to keep the network status. Our controller uses a variation of the TE algorithm that we designed in [32], which is based on an integer programming formulation and minimises the number of nodes required to support a set of flows at a reduced computational cost. Other TE schemes could be supported, based on, e.g., flow management policies or traffic measurement requirements (see [33] for a recent survey).

The APs are small PCs running Linux and extended to support the required functionality as follows. We have modified the widely used `hostapd` demon[3] to set up a connection with the 802.11 module at the controller, so that the AP can report changes in the network conditions (which are updated to the database) and can set up new configurations. When the controller issues a change on the default forwarding path of a node, the 802.11 module translates this primitive to an 802.11k request to perform channel measurements (so that the mobile updates the list of available APs) and an 802.11v command to change the point of attachment. An overview of the developed software modules and interfaces is depicted in Fig. 2.17.

The CN is a regular desktop machine, while the MNs are small PCs, like the switch, which runs `OpenVSwitch`. Finally, we set up an additional desktop machine, not shown in the picture, running the `FreeRADIUS` server[4] to enable the use of WPA2-enterprise as required by WiFi Passpoint.

2.2.2.6 Results

To validate that the controller activates resources as traffic requirements vary, we perform the following experiment. We first generate a Transmission Control Protocol (TCP) flow between the CN and node a, which is associated with AP 1, while AP 2 is inactive. At $t = 20$ s, we generate another TCP flow between the CN and node b, which saturates the capacity of the link. Thanks to periodic measurements, the controller decides 3 s later that more resources are required, and switches on AP 2. Once this AP is available, it notifies the controller, which then immediately issues a handover trigger to node b that associates with the new AP at $t = 45$ s.

The results of this experiment are depicted in Fig. 2.18, where we plot the per-flow throughput (bottom) and the total throughput (top) as measured by the Wireshark tool. According to the figure, the first TCP flow achieves at first about 20 Mbps, but when the second flow appears, its throughput its reduced to 5 Mbps, while the former gets about 15 Mbps (with some variations due to contention). Once this has been moved to AP 2, the flow from node a gets again about 20 Mbps, while the flow from node b gets about 20 Mbps as well.

We also represent in Fig. 2.18 the estimated energy consumption of the infrastructure, following the model of [34]. As compared with the case of both APs on, our solution reduces energy consumption by 18%, an improvement that comes at the cost of an increased delay, mainly because the time it takes to detect a new flow and power-up the AP. Following the results reported in [35], we estimate that our framework could reduce energy consumption by 30–40% in a campus-wide deployment (note that our experiment corresponds to an "upper bound" in terms of performance reduction, as there is always wireless activity).

[1] `http://osrg.github.io/ryu/`
[2] `http://www.mysql.com`
[3] `http://w1.fi/hostapd/`
[4] `http://freeradius.org`

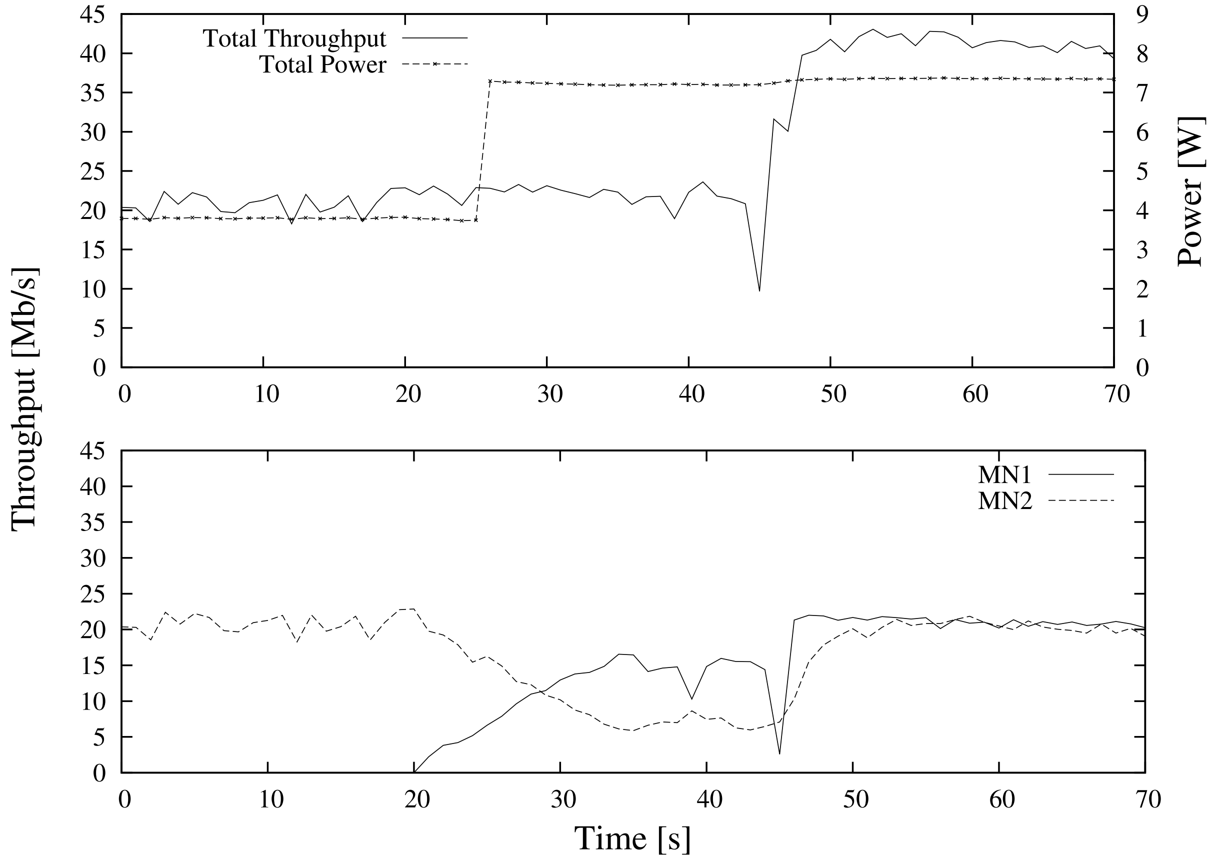


Figure 2.18: Validation of the proof of concept.

3 Conclusion

In this document we have provided the final results of the research activities that conclude the work of WP3 of CROWD. The research has resulted in promising results to achieve the objectives we have set for CROWD. We have presented a network discovery mechanism that allows the CROWD controllers to identify CROWD compliant elements in the network. Next, we have introduced a fast and flexible heuristic framework to handle the dynamic controller placement in CROWD and we have extended the CROWD backhaul reconfiguration approach to support the feasibility of the CROWD mechanisms, which operate at medium to long time scales. Last but not least, the extended results from our case studies confirm the performance of the CROWD medium to long time scale mechanisms.

As reported within the key contributions of this document, the most promising results have also already been published or are prepared for publication in the research community to strengthen the impact of CROWD.

A Appendix

A.1 Network Discovery Mechanism

As already said in Section 2.1.1, the proprietary protocol used for LTE networks has been developed in JSON format. The main messages exchanged for the network discovery scope are reported below:

This message is sent from eNB, into EmuLte, to CLC to notify itself. The parameters into message are:

```
   - "obj":     sender
   - "type":    Type of message
   - "name":    Name to identifier the device
   - "mac":     Mac address of the device
   - "state":   State of \ac{eNB} [0 = OFF; 1 = ON]
   - "pos":      eNB's coordinates position.

{
    "message":
    {
        "obj": "BS",
        "type": "create",
        "name": "bs-name",
        "mac": "00:00:00:00:00:FF",
        "state": "1",
        "absf" : "1",
        "pos":
        {
            "x": "60",
            "y": "60"
        }
    }
}
```

This message is sent from eNB to notify how many and which device the BS is serving actually. The parameters into message are:

```
- "obj":     sender
   - "type":    Type of message
   - "name":    Name to identifier the device
   - "state":   Bs state [0 = OFF; 1 = ON]
   - "pwr":     Power consumption in Watt
   - "ue-link": Array of UE. For each UE are the follow information:
                 - "name": It is the name of UE
                 - "ip":   It is the UE's ip address
                 - "mac":  It is the UE's mac address
                 - "posx" and "posy": They are the actual UE's position
```

```
{
    "message":
    {
        "obj": "BS",
        "type": "update",
        "name": "bs-name",
        "state": "1",
        "pwr": "12.345",
        "ue-link":
        [
            {
                "name": "1",
                "ip": "192.168.56.1",
                "mac": "00:00:00:00:00:01",
                "type": "update|del|add",
                "pos":
                 {
                     "x": "60",
                     "y": "64"
                 }
            },
            {
                "name": "2",
                "ip": "192.168.56.2",
                "mac": "00:00:00:00:00:02",
                "type": "update|del|add",
                 "pos":
                 {
                     "x": "67",
                     "y": "70"
                 }
            }
        ]
    }
}
```

This message is sent from the CLC to the eNB which is serving the target UE. In this message will be provide the UE and the target eNB that will serve same UE later. The parameters into message are:

```
   - "obj":          sender
   - "type":         Type of message
   - "bsDestName":   It is target BS's name
   - "ueIp":         It is target UE's IP
```

```
{
    "message":
    {
        "obj": "CLC",
        "type": "HO",
        "bsDestName": "bs-name",
        "ueIp": "IP"

    }
}
```

This message is sent from the CLC to the eNB. In this message will be provide to switch ON or switch OFF the eNB. The parameters into message are:

```
   - "obj":          sender
   - "type":          Type of message
   - "newState":      It can be [0 = OFF; 1 = ON]
```

```
{
    "message":
    {
        "obj": "CLC",
        "type": "changeState",
        "newState": "0",
    }
}
```

This message is sent from the BS to the CLC. In this message BS notify the state (ON/OFF). The parameters into message are:

```
   - "obj":           sender
   - "type":          Type of message
   - "state":         It can be [0 = OFF; 1 = ON]
{
    "message":
    {
        "obj": "BS",
        "type": "state",
        "name": "bs-name",
        "state": "1"
    }
}
```

This message is sent from one eNB to the CLC about measurament of a specific UE. The parameters into message are:

```
   - "obj":           sender
   - "type":          Type of message
   - "name":          It is BS's name
   - "pwr":           Power consumption in Watt
   - "state":         Bs state [0 = OFF; 1 = ON]
   - "ue-measures":       The follow information is an array of UE. For each UE are the fol
```

```
            "name":      Attached UE's name.
"ip":         Attached UE's ip.
            "mac":       Attached UE's mac.
"tpt-dl":     Downlink trhouhtput [Mbit/sec]
"tpt-ul":     Uplink trhouhtput [Mbit/sec]
            "CQI":       Channel Quality Indicator [0-15]
"Prx":        Power received
            "Prx-N":     Vector about Power received from neighbor
                 "bs-name":      Name of specific bs can be heard
                 "Prx":          Power received [dBm]
```

```
{
    "message":
    {
        "obj": "BS",
        "type": "update",
        "name": "bs-name",
        "state": "1",
        "pwr": "12.345",
        "ue-measures":
        [
            {
                "name": "1",
                "ip": "192.168.56.1",
                "mac": "00:00:00:00:00:01",
                "tpt-dl": "93.318",
                "tpt-ul": "93.318",
                "CQI":"7",
                "Prx":"65",
                "Prx-N":
                [
                 {
                 "bs-name":"2",
                 "Prx":"5"
                 },
                 {
                 "bs-name":"4",
                 "Prx":"70"
                 }
                ]
            },
            {
                "name": "3",
                "ip": "192.168.56.3",
                "mac": "00:00:00:00:00:03",
                "tpt-dl": "93.318",
                "tpt-ul": "93.318",
                "CQI":"15",
                "Prx":"70",
                "Prx-N":
                [
                 {
                 "bs-name":"2",
                 "Prx":"5"
                 },
                 {
                 "bs-name":"8",
                 "Prx":"30"
                 }
                ]
            }
        ]
    }
}
```

The protocol used for managing WiFi access point is standard SNMP. hereby we report the request performed to extract information.

```
prevRxOctets = snmpMgr.getNumericOid(".1.3.6.1.2.1.2.2.1.10." + wlanIfId);
              prevTxOctets = snmpMgr.getNumericOid(".1.3.6.1.2.1.2.2.1.16." + wlanIfId);
              prevRxDiscards = snmpMgr.getNumericOid(".1.3.6.1.2.1.2.2.1.13." + wlanIfId);
              prevTxDiscards = snmpMgr.getNumericOid(".1.3.6.1.2.1.2.2.1.19." + wlanIfId);
              prevRxErrors = snmpMgr.getNumericOid(".1.3.6.1.2.1.2.2.1.14." + wlanIfId);
              prevTxErrors = snmpMgr.getNumericOid(".1.3.6.1.2.1.2.2.1.20." + wlanIfId);
```

Regarding the OpenFlow (OF) protocol refer to www.openflow.org.

A.2 Dynamic Controller placement

A.2.1 Final FCPF optimisation model

Our final FCPF optimisation model is a Mixed Integer Quadratically Constrained Program (MIQCP). It takes the parameters listed in Table A.1 as inputs, which correspond to the above problem definition and uses the variables listed in Table A.2 to store the decisions about controller placement and data flow satisfaction.

Table A.1: MIQCP input parameters

V	set of nodes (WiFi APs, LTE BSs, routers/switches)
$C \subseteq V$	set of nodes that can serve as controller (CRC or CLC)
E	set of links with $E \subseteq V \times V$
F	set of flows passing through at least one node $v \in V$
W	matrix with $W_{x,v} = 1$ iff $x \in F$ passes through $v \in V$
$p_{\text{own}}(c)$	processing power at node $c \in C$
$b_{\text{cap}}(v, w)$	maximum data rate for link $(v, w) \in E$
$l_{\text{cap}}(v, w)$	latency of link $(v, w) \in E$
$b_{\text{flow}}(x)$	requested data rate for flow $x \in F$ to its CLC
$l_{\text{flow}}(x)$	required round trip latency for flow $x \in F$ to its CLC
$p_{\text{flow}}(x)$	processing capacity required for flow $x \in F$ at a CLC
$t_{\text{in}}(x)$	arrival time of data flow $x \in F$
$t_{\text{dur}}(x)$	duration of data flow $x \in F$
b_{CLC}	required data rate for a node to its CLC
b_{CRC}	required data for a CLC to its CRC
l_{CLC}	required round trip latency for a node to its CLC
l_{CRC}	required round trip latency for a CLC to its CRC
p_{CLC}	processing cap. required at a CLC for controlling a node
p_{CRC}	processing cap. required at a CRC for controlling a CLC

Table A.2: MIQCP variables

$CLC_{c,v} \in \{0,1\}$	determines whether $c \in C$ is a CLC for $v \in V$
$CRC_{c,d} \in \{0,1\}$	det. whether $c \in C$ is the CRC for CLC $d \in C$
$isCLC_c \in \{0,1\}$	determines whether $c \in C$ is a CLC
$isCRC_c \in \{0,1\}$	determines whether $c \in C$ is a CRC
$Sat_{c,x} \in \{0,1\}$	determines whether $c \in C$ satisfies $x \in F$
$isSat_x \in \{0,1\}$	determines whether $x \in F$ is satisfied by a CLC
$Proc_c \in \mathbb{Z}_+$	amount of units req. processing cap. from $c \in C$
$f_{c,u,v,w} \in \{0,1\}$	determines whether $(v, w) \in E$ is included in the path from CLC $c \in C$ to node $u \in V$
$g_{c,d,v,w} \in \{0,1\}$	determines whether $(v, w) \in E$ is included in the path from CRC $c \in C$ to CLC $d \in C$

The following constraints have to be ensured. Some constraints use a big-M constant denoted as $\mathcal{M}$.

All nodes need to be the end of a routing path (A.1) that is starting at a CLC (A.2). For all intermediate nodes on a routing path, the ingress and egress have to be balanced (A.3).

$$\sum_{(c,w)\in E} f_{c,d,c,w} = CLC_{c,d}, \quad \forall c \in C, d \in V, c \neq d \tag{A.1}$$

$$\sum_{(u,d)\in E} f_{c,d,u,d} = CLC_{c,d}, \quad \forall c \in C, d \in V, c \neq d \tag{A.2}$$

$$\sum_{(u,v)\in E} f_{c,d,u,v} = \sum_{(v,w)\in E} f_{c,d,v,w}, \quad \begin{array}{c}\forall c \in C,\ d, v \in V,\\ c \neq d \neq v\end{array} \tag{A.3}$$

Similar constraints are needed for the routing paths from each CLC to its CRC (A.4-A.6).

$$\sum_{(c,w)\in E} g_{c,d,c,w} = CRC_{c,d} \cdot isCLC_d, \quad \forall c, d \in C, c \neq d \tag{A.4}$$

$$\sum_{(v,d)\in E} g_{c,d,v,d} = CRC_{c,d} \cdot isCLC_d, \quad \forall c, d \in C, c \neq d \tag{A.5}$$

$$\sum_{(u,v)\in E} g_{c,d,u,v} = \sum_{(v,w)\in E} g_{c,d,v,w}, \quad \begin{array}{c}\forall c, d \in C,\ v \in V,\\ c \neq d \neq v\end{array} \tag{A.6}$$

Each node is required to be controlled by at least one CLC (A.7) and each CLC must be assigned to exactly one CRC (A.8).

$$\sum_{c\in C} CLC_{c,v} \geq 1, \quad \forall v \in V \tag{A.7}$$

$$\sum_{c\in C} CRC_{c,d} = isCLC_d, \quad \forall d \in C \tag{A.8}$$

Further, the CLC (A.9) and CRC (A.10) decision variables need to be set.

$$\mathcal{M} \cdot isCLC_c \geq \sum_{v\in V} CLC_{c,v}, \quad \forall c \in C \tag{A.9}$$

$$\mathcal{M} \cdot isCRC_c \geq \sum_{d\in C} CRC_{c,d}, \quad \forall c \in C \tag{A.10}$$

A CLC can only satisfy a flow if it controls all nodes a flow is connected to. If this is not the case, constraint (A.11) forces the corresponding decision variable to be zero.

$$\mathcal{M} \cdot (Sat_{c,x} - 1) \leq \sum_{v\in V} (CLC_{c,v} - 1) \cdot W_{x,v}, \quad \begin{array}{c}\forall c \in C,\\ x \in F\end{array} \tag{A.11}$$

A data flow needs to be satisfied by at most one CLC (A.12) and the corresponding decision variable needs to be set (A.13).

$$\sum_{c\in C} Sat_{c,x} \leq 1, \quad \forall x \in F \tag{A.12}$$

$$isSat_x = \sum_{c\in C} Sat_{c,x}, \quad \forall x \in F \tag{A.13}$$

Moreover, the data rate limits of each link (A.14) must not be exceeded.

$$\sum_{c\in C, d\in V} f_{c,d,v,w} \cdot \left(b_{\text{CLC}} + \sum_{x\in F} W_{x,d} \cdot Sat_{c,x} \cdot b_{\text{flow}}(x) \right) + \sum_{c,d\in C} g_{c,d,v,w} \cdot b_{\text{CRC}} \leq b_{\text{cap}}(v,w), \quad \forall (v,w) \in E \tag{A.14}$$

As a next step, the number of controlled nodes and satisfied flows is determined for each potential controller (A.15).

$$Proc_c = \sum_{x \in F} Sat_{c,x} + \sum_{d \in V} CLC_{c,d} + \sum_{d \in C} CRC_{c,d}, \forall c \in C \tag{A.15}$$

Now, we can make sure that the latency constraints for each flow (A.16), CLC (A.17) and CRC (A.18) are met by offering the same percentage of a controller's processing capacity to each unit served.

$$\begin{aligned} &\sum_{(v,w) \in E} f_{c,d,v,w} \cdot (l_{\text{cap}}(v,w) + l_{\text{cap}}(w,v)) + \frac{Proc_c \cdot p_{\text{flow}}(x)}{p_{\text{node}}(c)} && \text{(A.16)} \\ &\quad \leq l_{\text{flow}}(x) + \mathcal{M} \cdot (1 - Sat_{c,x}), \quad \forall c \in C, d \in V, x \in F, W_{x,d} = 1 \\ &\sum_{(v,w) \in E} f_{c,d,v,w} \cdot (l_{\text{cap}}(v,w) + l_{\text{cap}}(w,v)) + \frac{Proc_c \cdot p_{\text{CLC}}}{p_{\text{node}}(c)} && \text{(A.17)} \\ &\quad \leq l_{\text{CLC}} + \mathcal{M} \cdot (1 - CLC_{c,d}), \quad \forall c \in C, d \in V \\ &\sum_{(v,w) \in E} g_{c,d,v,w} \cdot (l_{\text{cap}}(v,w) + l_{\text{cap}}(w,v)) + \frac{Proc_c \cdot p_{\text{CRC}}}{p_{\text{node}}(c)} && \text{(A.18)} \\ &\quad \leq l_{\text{CRC}} + \mathcal{M} \cdot (1 - CRC_{c,d}), \quad \forall c, d \in C \end{aligned}$$

Last but not least, we need to cope with possible loop and corner cases (A.19-A.22).

$$f_{c,d,v,w} + f_{c,d,w,v} \leq 1, \quad \forall c, d \in V, (v,w) \in E \tag{A.19}$$

$$g_{c,d,v,w} + g_{c,d,w,v} \leq 1, \quad \forall c, d \in V, (v,w) \in E \tag{A.20}$$

$$CLC_{c,c} = isCLC_c, \quad \forall c \in C \tag{A.21}$$

$$CRC_{c,c} = isCRC_c \cdot isCLC_c, \quad \forall c \in C \tag{A.22}$$

The objective function (A.23) defines a lexicographic order in which maximising flow satisfaction weighs higher than minimising the number of controllers.

$$\text{minimise: } \sum_{c \in C} \big(isCRC_c + isCLC_c\big) - 3|C| \cdot \sum_{x \in F} isSat_x \tag{A.23}$$

A.2.2 FlexFCPF heuristic algorithm

In this section we elaborate on the implementation details of our heuristic algorithm FlexFCPF. First, we focus on the initial controller placement functionality of FlexFCPF for placing controllers from scratch before coming to the flexible reassignment abilities of FlexFCPF. All procedures use the input parameters from Table A.1.

A.2.2.1 Initial controller placement

When confronted with a network without an existing controller placement, FlexFCPF executes the CPGREEDY procedure shown in Algorithm 1. CPGREEDY successively adds CLCs to the network by calling FINDCLC and first focuses on having a CLC assigned to each network node. When all nodes are controlled by at least one CLC and if there are still potential controllers available, CPGREEDY switches its strategy and fully focuses on data flow satisfaction. The procedure ends if all data flows are satisfied, if no more controllers are available or if no new flows could be satisfied by adding an additional CLC, e.g. if all remaining data flows cannot be satisfied with the network's remaining resources. Right before terminating, the CLEANUPCLCCONTROLS procedure is executed, which removes all CLC-to-node assignments that were established during the controller placement

Algorithm 1 CPGREEDY()

```
option = "neighbors"
while $|uncontrolled\,nodes| > 0$ do
    FINDCLC(option)
    if |CLCs| = |C| then
        break // no more potential controllers available
    end if
end while
if |CLCs| < |C| then
    option = "flows"
    while $|unsatisfied\,flows| > 0$ do
        $lastsat = |satisfied\,flows|$
        FINDCLC(option)
        if |CLCs| = |C| or $lastsat = |satisfied\,flows|$ then
            break
        end if
    end while
end if
CLEANUPCLCCONTROLS()
```

and that eventually did not serve any purpose, i.e. the CLC does not satisfy any data flow passing through the node and the node is assigned to more than one CLC.

The FINDCLC procedure presented in Algorithm 2 determines which potential controller is going to be added as a CLC next, according to the option set by CPGREEDY. For this purpose, it acquires the best candidates via GETCLCCANDIDATES(*option*). But before adding the best candidate as a CLC right away, FINDCLC first tries to find a CRC for it using the FINDCRC procedure as a CLC that cannot be assigned to a CRC would violate the integrity of our two-layer controller architecture. If a CRC can be found, the candidate is confirmed as a new CLC and assigned to the CRC. Finally, Algorithm 5 is called which assigns nodes and data flows to the recently added CLC.

Algorithm 2 FINDCLC(*option*)

```
candidates = GETCLCCANDIDATES(option)
for $v$ in candidates do
    $c$ = FINDCRC($v$)
    if $c$ is not None then
        ADDNEWCLC($v$)
        break // next CLC determined
    end if
end for
```

The FINDCRC procedure from Algorithm 3 tries to assign a CRC to a given CLC v. First, it checks the already available CRCs, starting with the closest one. The CHECKCRCCONTROL procedure verifies if assigning a CRC to a node complies with all network constraints according to a given routing path. If an existing CRC can be assigned to v, the added CRC control is returned, otherwise a new CRC has to be added to the network using a similar strategy as before except now considering all potential controllers that are not yet CRCs. For the special case of the first CRC in the network, we have decided to seek the most centralised potential CRC in order to benefit all future CRC-to-CLC assignments.

Algorithm 3 FINDCRC(v)

```
$paths$ = {SHORTESTPATH(source=$c$, target=$v$) for $c$ in CRCs}
sort $paths$ by lengths
for $p$ in $paths$ do
   if CHECKCRCCONTROL($p$) then
      return ADDCRCCONTROL($c, v$)
   end if
end for
$paths$ = {SHORTESTPATH(source=$c$, target=$v$) for $c$ in C - CRCs} // a new CRC needs to be added
if |CRCs| = 0 then
   sort $paths$ by average length to all potential controllers
else
   sort $paths$ by lengths
end if
for $p$ in $paths$ do
   if CHECKCRCCONTROL($p$) then
      return ADDCRCCONTROL($c, v$)
   end if
end for
return None // failed finding a CRC for $v$
```

Algorithm 4 GETCLCCANDIDATES(*option*)

```
$candidates$ = C - CLCs
if $candidates$ - CRCs $\neq \emptyset$ then
   $candidates$ = $candidates$ - CRCs // avoid CRCs if possible
end if
if $option$ = "neighbors" then
   sort $candidates$ by highest amount of uncontrolled nodes in $\{v\} \cup$ neighbors($v$)
   if best value = 0 then // no node with uncontrolled neighbors
      $candidates$ = GETCLCCANDIDATES("isolated nodes")
   end if
else if $option$ = "isolated nodes" then
   sort $candidates$ by shortest distance to an uncontrolled node
else if $option$ = "flows" then
   sort $candidates$ by highest amount of unsatisfied flows passing through $v$
   if best value = 0 then // no node with unsatisfied flows
      $candidates$ = GETCLCCANDIDATES("isolated flows")
   end if
else if $option$ = "isolated flows" then
   sort $candidates$ by shortest distance to a node with an unsatisfied flow passing through
end if
```

The critical CLC candidate selection is done by the GETCLCCANDIDATES procedure which is shown in Algorithm 4. GETCLCCANDIDATES considers all potential controllers that are not yet used as a CLC. But if possible, the procedure also excludes any CRC to leave them with resources for controlling future CLCs. To determine the best CLC candidates, GETCLCCANDIDATES then uses the option *neighbors* or *flows*, depending on what has been specified by CPGREEDY. However, for corner cases GETCLCCANDIDATES is allowed to switch to one of the more specific options *isolated nodes* or *isolated flows* in case the metrics used by *neighbors* or *flows* do not provide a sufficient result for sorting the candidates.

As stated before, the ADDNEWCLC procedure (Algorithm 5) is responsible for assigning nodes and data flows to a new CLC v. To this end, it calculates the shortest paths from v to all other nodes in the network. Then, it prioritises the nodes to be assigned to the new CLC by sorting the paths, depending on whether a network already has a complete control structure or not. In the former case, ADDNEWCLC considers paths to uncontrolled nodes first, while in the latter case, the paths are sorted according to the amount of unsatisfied data flows passing through the target nodes.

Algorithm 5 ADDNEWCLC(v)

```
paths = {SHORTESTPATH(source=$v$, target=$i$) for $i$ in nodes}, potential flows = {}
if |uncontrolled nodes| > 0 then // no valid solution yet
  sort paths by paths to uncontrolled nodes first, path length next
else // valid solution already found, now focus on flow satisfaction
  sort paths by paths to nodes with unsatisfied flows first, path length next
end if
while (|paths| > 0 or |potential flows| > 0) and (|uncontrolled nodes| + |unsatisfied flows| > 0) do
  if |potential flows| > 0 and (|uncontrolled nodes| = 0 or |new nodes controlled| ≥VCRatio) then
    $f$ = GETMOSTDEMANDINGFLOW(potential flows)
    if CHECKFLOWSAT($v, f$) = True then
      ADDFLOWSAT($v, f$)
    end if
    potential flows.remove($f$)
  else
    $p$ = GETNEXTPATH(paths)
    if CHECKCLCCONTROL($p$) = True then
      ADDCLCCONTROL($p$)
      UPDATEPOTENTIALFLOWS($v$)
    else
      break // no more resources left at CLC $v$
    end if
  end if
end while
```

After the prioritisation is done, nodes are assigned to be controlled by v. To check that no model constraint is violated in the process, the CHECKCLCCONTROL procedure verifies if assigning a node to v complies with all network constraints each time before assigning it to v. When a node is assigned to v the UPDATEPOTENTIALFLOWS function is executed to update the list of the *potential flows* for v, i.e. the flows that are only passing through nodes that v already controls and thus can be satisfied by v if sufficient resources are available. However, we hold back before assigning potential flows to v. Assuming a network with many data flows, a CLC would probably have a lot of potential flows after controlling only a few nodes, run out of resources by satisfying them very quickly and hence a complete control structure would probably not be obtained. Hence, data flows may only be satisfied by a CLC once it controls at least VCRatio $= \frac{|V|}{|C|}$ nodes that had previously been uncontrolled or if there are no more uncontrolled nodes in the network.

Eventually, when the conditions are met so that v is allowed to satisfy data flows, we successively try to assign the most demanding, in terms of requested processing capacity, data flow to v. Again, all constraints are checked using the CHECKFLOWSAT procedure before confirming that v henceforth satisfies a certain data flow. In any case, the data flow is then removed from the list of potential flows for v, as it is either now satisfied by v or as it cannot be satisfied by v. ADDNEWCLC terminates when no more nodes and no more data flows can be assigned to v.

A.2.2.2 Flexible reassignment

After establishing a complete control structure for a network, we consider the following triggers for changing the network configuration:

1. Incoming data flows
2. Low load situations

Satisfying incoming data flows: Each time that a new data flow appears in the network, FlexFCPF at first tries to assign it to one of the network's CLCs by calling the BROWSECLCsFORFLOW procedure (Algorithm 6). Unlike the ADDNEWCLC procedure (Algorithm 5), where a CLC is assigned to as many data flows as possible from its nearby nodes, this procedure specifically tries to assign a certain data flow to the closest possible CLC. If BROWSECLCsFORFLOW fails to satisfy an incoming data flow, FlexFCPF launches the CPGREEDY procedure which will in this case result in executing FINDCLC($flows$) (see Algorithm 1) to add a new CLC that will satisfy the new data flow.

Algorithm 6 BROWSECLCsFORFLOW(f)

```
CLCs* = {CLCs controlling all nodes $f$ is passing through}
for $v$ in CLCs* do
  if CHECKFLOWSAT($v, f$) = True then
    return ADDFLOWSAT($v, f$)
  end if
end for
CLCs+ = CLCs - CLCs*
sort CLCs+ by average distance to all nodes $f$ is passing through
for $v$ in CLCs+ do
  paths = {SHORTESTPATH(source=$v$, target=$i$) if $v$ does not control $i$ and $f$ passes through $i$}
  for $p$ in paths do
    if CHECKCLCCONTROL($p$) = True then
      ADDCLCCONTROL($p$)
    else
      break // $v$ cannot satisfy $f$
    end if
  end for
  if all nodes assigned and CHECKFLOWSAT($v, f$) = True then
    return ADDFLOWSAT($v, f$)
  else // $v$ cannot satisfy $f$, remove useless control assignments
    for $p$ in paths do
      REMOVECLCCONTROL($p$)
    end for
  end if
end for
```

Low load detection: While it is obvious when it is necessary to add an additional CLC, answering the question when it is possible to remove a CLC while still satisfying all data flows is much harder. To do this, we define a notion of *load* for a CLC:

$$\text{CLCload}(c) = \max\left(\max_{f\in\text{Sat}(c)} \frac{\text{RTT}(f)}{l_{\text{flow}}(f)}, \max_{v\in\text{Contr}(c)} \frac{\text{RTT}(v)}{l_{\text{CLC}}}\right),$$

where Sat(c) is the set of data flows satisfied by c, Contr(c) is the set of nodes controlled by c and RTT($\cdot$) is the round trip time for a CLC-to-flow or CLC-to-node assignment. Recall: because of the assumed equal share scheduling for data processing (see Section 2.1.2.1), the round trip time increases for all nodes and flows assigned to a CLC as soon as a new node or flow is assigned to it. As FlexFCPF ensures that the latency constraints for an existing assignment are never violated, it holds that CLCload(c) $\in (0, 1]\ \forall\ c \in$ CLCs.

To detect low load situations, FlexFCPF consistently monitors the average CLC load in the network. If the average CLC load stays below a threshold value *LoadLimit* for more than 10 seconds, FlexFCPF executes the LowLoad procedure that is described in Algorithm 7.

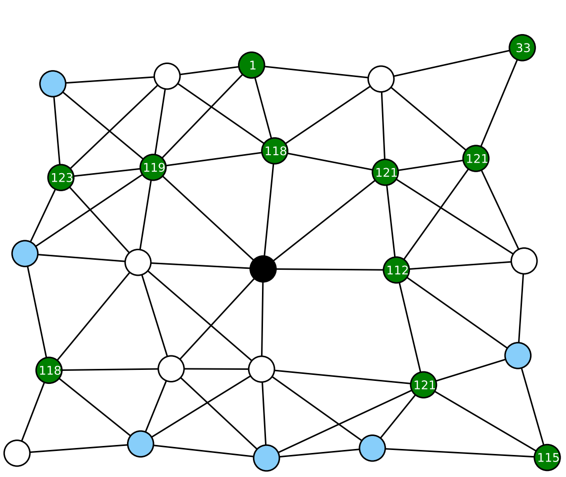
Algorithm 7 LowLoad()

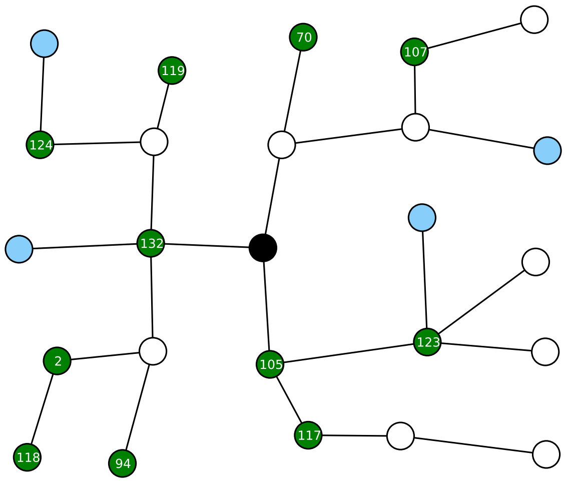
```
len = |CLCs|
load = ⌈sum(CLCload($c$) for $c$ in CLCs)⌉ // amount of definitely required CLCs
while |CLCs| > load do
    removeCLC(getCLCwithLeastLoad())
end while
if |CLCs| < len then
    browseCLCs()
    if $|uncontrolled\,nodes| > 0$ or $|unsatisfied\,flows| > 0$ then
        CPgreedy()
    end if
end if
if |CLCs| ≥ len then
    $LoadLimit = LoadLimit$ - 0.05
else
    $LoadLimit = 0.9$
end if
```

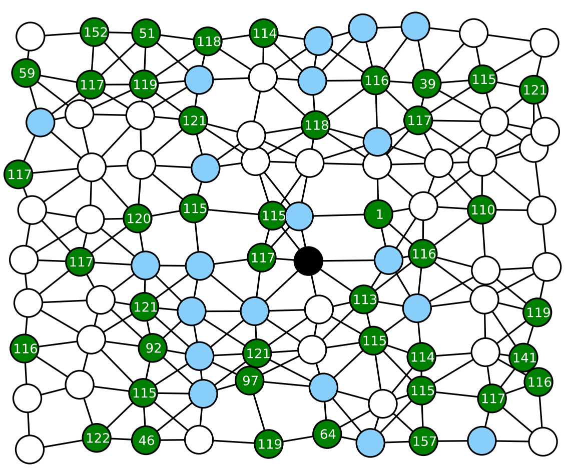
In short, LowLoad removes all not necessarily required CLCs and then runs the browseCLCs procedure, which browses through all current CLCs and tries to assign uncontrolled nodes and unsatisfied data flows to them. As this procedure works very similar to the addNewCLC procedure we skip a detailed representation. If there are still uncontrolled nodes or unsatisfied data flows browseCLCs has been executed, the CPgreedy procedure is launched to correct this as for the case of incoming data flows. If, in the end, there are not less CLCs than before, the low load correction is seen as a failure and *LoadLimit* is decreased by 0.05 for the next round. Otherwise, *LoadLimit* is set (back) to its initial value 0.9.

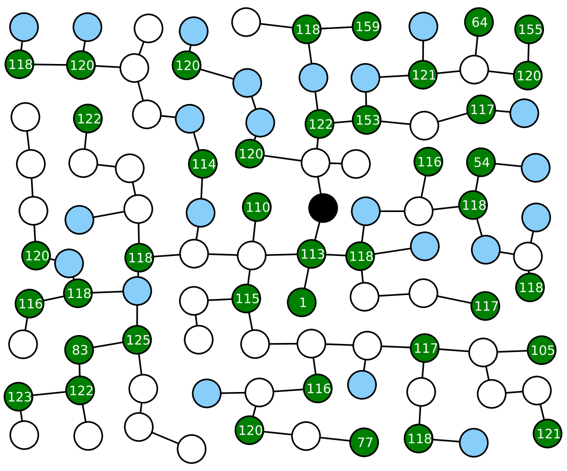
A.2.3 Simulation topologies

Figure A.1 displays our four test networks from the dynamic network simulation in Section 2.1.2.3 with the first extracted state after $t = 0$. The CLCs are labelled with the amount of satisfied data flows.


(a) Mesh topology (25 nodes)


(b) Tree topology (25 nodes)


(c) Mesh topology (100 nodes)


(d) Tree topology (100 nodes)

● CRCs ● CLCs ● potential controllers

Figure A.1: Evaluation: network topologies

A.3 Dynamic Backhaul Reconfiguration

The inputs for the heuristic algorithm, introduced in Section 2.1.3, are the backhaul network as an annotated graph $G = (V, E)$, a set of available wavelengths per link K and the desired CBS W with each $W_i \subset V$ together with their constraints on data rate and latency.

An overview of the heuristic is depicted in Figure A.2 and the individual steps are explained below.

```
CBS Prioritisation
▷ decide for each CBS if it belongs to a hotspot
▷ output: list of hotspot CBSs and normal CBSs
For each CBS in $W_{hotspot}$, then for each CBS in $W_{normal}$
    1. Maximum-Path BFS
       ▷ start modified BFS from each vertex
       ▷ output: BFS tree for each vertex
    2. Match CBS
       ▷ match BFS trees against CBS ▷ output: possible candidate BFS trees for CBS
    3. Back-Track BFS Trees
        a) Check constraints
           ▷ recheck constraints on candidate BFS trees
        b) Wavelength Assignment
           ▷ determine wavelengths for all candidate BFS trees
       ▷ output: possible candidate BFS trees and their wavelength assignment for CBS
    4. Match CBS
       ▷ match BFS trees against CBS again
       ▷ output: confirmed candidate BFS trees for CBS
    5. Find Best BFS Tree
       ▷ compare candidate BFS trees
       ▷ output: best BFS tree for CBS
```

Figure A.2: Hotspot BFS algorithm overview

CBS Prioritisation This new step of the algorithm divides the desired CBS into two sets $\mathrm{W}_{\mathrm{hotspot}}$ and $\mathrm{W}_{\mathrm{normal}}$. The calculations in this step use two threshold values t_v, which determines how strict the filtering of hotspot vertices should be, and t_h, which determines how strict the filtering of hotspot CBS should be. The numbers for the threshold values depend on the scenario and can be determined using a parameter study.

The algorithm first calculates for each vertex v from the backhaul graph in how many CBS the vertex is present as:

$$h_v = \mathbb{1}_{v \in W_i}$$

$\mathbb{1}_X$ denotes an indicator variable with value 1 if condition X is true and value 0 otherwise. Each vertex v is considered a hotspot vertex if

$$h_v > |W| \cdot t_v$$

and is added to the set of hotspot vertices V_h. Now each CBS W_i is added to $\mathrm{W}_{\mathrm{hotspot}}$ if

$$\frac{\sum_{v \in W_i} \mathbb{1}_{v \in V_h}}{|W_i|} \geq t_h$$

otherwise it is added to W$_{\text{normal}}$.

Now the actual algorithm for BBU/LC placement and backhaul resource allocation has to be executed for each CBS. The following steps are executed per CBS, starting with the CBSs from W$_{\text{hotspot}}$ and the CBSs from W$_{\text{normal}}$:

Maximum-Path BFS The heuristic performs a modified breadth-first search from each vertex of the graph as the start node. Whenever a new tree edge (u, v) is discovered, the constraints for the new edge are checked in the following way:

- Does the latency to the new vertex v via the whole path from the root via u and the new edge (u, v) not exceed the maximum round-trip latency?
- Are there any wavelengths on (u, v) with enough free capacity to connect v to the root vertex, if v is part of the CBS?

If these checks are not successful, the new vertex v is discarded, otherwise it is added to the tree. The result of this step is a set T of BFS trees, which only contain vertices that meet the constraints within the tree. This does not yet take into account reciprocal effects between multiple BSs in the CBS.

Match CBS The heuristic checks for every BFS tree T_i from step *2)*, whether it contains all BSs from the desired CBS. The result is a set of only these BFS trees T that match the desired CBS.

Back-Track BFS Trees In the constraint checking in step *1)* vertices were discarded based on the latency for the full paths to the start node and the link capacity for only single links and not the full paths. Here the capacity constraints are checked again taking into account the capacities on full paths.

If the constraints are not violated a wavelength assignment for all routing paths between the BSs from the CBS and the BFS tree root has to be determined. Because the required data structures for this step can easily be constructed in the back-tracking phase, the back-tracking and the wavelength assignment are executed in one step.

Details on the implementation of the wavelength assignment can be found in Deliverable D3.2 [8].

Match CBS After removing vertices in the previous step, the data on trees matching the CBS is not valid any more. Thus, step *2)* is repeated. If a tree still matches the CBS, the starting node is a valid candidate for a controller location for that CBS.

Find Best BFS Tree Finally, the best BFS tree from the remaining candidates has to be determined. The algorithm calculates three different costs per candidate tree and sums them up in a weighted sum n per tree

$$n = \sum (w_g \cdot n_g + w_a \cdot n_a + w_l \cdot n_l)$$

with

- n_g as the total number of wavelengths used in the tree
- n_a as the number of wavelengths that have to assigned additionally for using that tree
- n_l as the number of used links

and w_g, w_a and w_l as weight factors.

The algorithm then selects the tree with the lowest total cost, sets the root BS as the controller, stores the routing paths and updates the annotations on G for running the next iteration for the next CBS.

Bibliography

[1] Sébastien Auroux and Holger Karl. Efficient flow processing-aware controller placement in future wireless networks. *Proceedings of IEEE Wireless Communications and NetworkingConference (WCNC)*, 2015.

[2] Sébastien Auroux, Martin Dräxler, Arianna Morelli, and Vincenzo Mancuso. Dynamic network reconfiguration in wireless densenets with the crowd sdn architecture. In *European Conference on Networks and Communications (EuCNC) 2015*, 2015.

[3] M. I. Sanchez, A. Asadi, M. Dräxler, R. Gupta, V. Mancuso, A. Morelli, A. de la Oliva, and V. Sciancalepore. Tackling the increased density of 5G networks; the CROWD approach. In *IEEE 81st Vehicular Technology Conference: VTC2015-Spring, First International Workshop on 5G Architecture (5GArch 2015)*, 2015.

[4] M. Dräxler and H. Karl. Dynamic Backhaul Network Configuration in SDN-based Cloud RANs. *arXiv preprint arXiv:1503.03309*, 2015.

[5] Jorge Ortn, Pablo Serrano, and Carlos Donato. Modeling the Impact of Start-Up Times on the Performance of Resource-on-Demand Schemes in 802.11 WLANs. In *Proceedings of the 4th IFIP Conference on Sustainable Internet and ICT for Sustainability*, April 2015.

[6] Carlos Donato, Pablo Serrano, Antonio de la Oliva, Albert Banchs, and Carlos J. Bernardos. An OpenFlow Architecture for Energy Aware Traffic Engineering in Mobile Networks . *IEEE Network (accepted for publication).*

[7] Sébastien Auroux, Claudio Cicconetti, Martin Dräxler, Arianna Morelli, Laurent Roullet, Pablo Serrano, and Angelos Chatzipapas. Initial specification of Dynamic Radio and Backhaul configuration mechanisms. Project deliverable D3.1, CROWD, 2013.

[8] Arash Asadi, Sébastien Auroux, Claudio Bottai, Martin Dräxler, Philipp Dreimann, Arianna Morelli, Laurent Roullet, Pablo Serrano, and Fabio Toninelli. Final Specification of Functions and Interfaces of Dynamic Radio and Backhaul Configuration Mechanisms. Project deliverable D3.2, CROWD, 2014.

[9] Arianna Morelli, Claudio Bottai, Simone Mattiacci, Vincenzo Mancuso, Arash Asadi, Christian Vitale, Vincenzo Sciancalepore, Laurent Roullet, and Pablo Serrano. Final specification for the wireless enhancements functions and interfaces. Project deliverable D2.2, CROWD, 2014.

[10] Prakash Bhat, Satoshi Nagata, Luis Campoy, Ignacio Berberana, Thomas Derham, Guangyi Liu, Xiaodong Shen, Pingping Zong, and Jin Yang. LTE-Advanced: An operator perspective. *IEEE Communications Magazine*, pages 104–114, 2012.

[11] Cisco. Cisco visual networking index: Global mobile data traffic forecast update, 2013–2018. *Cisco Public Information*, 2014.

[12] Hassan Ali-Ahmad, Claudio Cicconetti, Antonio de la Oliva, Martin Dräxler, Rohit Gupta, Vincenzo Mancuso, Laurent Roullet, and Vincenzo Sciancalepore. CROWD: An SDN Approach for DenseNets. In *Proceedings of the 2nd European Workshop on Software Defined Networks (EWSDN)*, 2013.

[13] Hassan Ali-Ahmad, Claudio Cicconetti, Antonio de la Oliva, Vincenzo Mancuso, Malla Reddy Sama, Pierrick Seite, and Sivasothy Shanmugalingam. An SDN-based Network Architecture for Extremely Dense Wireless Networks. In *Proceedings of IEEE Software Defined Networks for Future Networks and Services (IEEE SDN4FNS)*, 2013.

[14] Advait Dixit, Fang Hao, Sarit Mukherjee, TV Lakshman, and Ramana Kompella. Towards an elastic distributed sdn controller. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 7–12. ACM, 2013.

[15] William E Hart. Python optimization modeling objects (pyomo). In *Operations Research and Cyber-Infrastructure*, pages 3–19. Springer, 2009.

[16] Gurobi Optimization. `http://www.gurobi.com/`.

[17] Oliver Blume, Anton Ambrosy, Michael Wilhelm, and Ulrich Barth. Energy efficiency of lte networks under traffic loads of 2020. In *Wireless Communication Systems (ISWCS 2013), Proceedings of the Tenth International Symposium on*, pages 1–5. VDE, 2013.

[18] Averill M. Law and W. David Kelton. *Simulation modeling and analysis*, volume 2. McGraw-Hill New York, 1991.

[19] Ying Zhang and Ake Årvidsson. Understanding the characteristics of cellular data traffic. *ACM SIGCOMM Computer Communication Review*, 42(4):461–466, 2012.

[20] Chenyang Yang, Shengqian Han, Xueying Hou, and A.F. Molisch. How do we design CoMP to achieve its promised potential? *Wireless Communications, IEEE*, 20(1):67–74, 2013.

[21] P. Baracca, F. Boccardi, and V. Braun. A dynamic joint clustering scheduling algorithm for downlink CoMP systems with limited CSI. In *Wireless Communication Systems (ISWCS), 2012 International Symposium on*, pages 830–834, August 2012.

[22] V. Jungnickel, K. Manolakis, S. Jaeckel, M. Lossow, P. Farkas, M. Schlosser, and V. Braun. Backhaul requirements for inter-site cooperation in heterogeneous LTE-Advanced networks. In *IEEE International Conference on Communications Workshops (ICC)*, 2013.

[23] T. Biermann, L. Scalia, C. Choi, H. Karl, and W. Kellerer. Backhaul Network Pre-Clustering in Cooperative Cellular Mobile Access Networks. In *Proc. IEEE World of Wireless Mobile and Multimedia Networks (WoWMoM)*, 2011.

[24] Thorsten Biermann, Luca Scalia, Changsoon Choi, Holger Karl, and Wolfgang Kellerer. CoMP clustering and backhaul limitations in cooperative cellular mobile access networks. *Pervasive and Mobile Computing*, 8(5):662 – 681, 2012.

[25] Oliver Blume, Anton Ambrosy, Michael Wilhelm, and Ulrich Barth. Energy Efficiency of LTE networks under traffic loads of 2020. In *Proceedings of the Tenth International Symposium on Wireless Communication Systems (ISWCS)*, 2013.

[26] Klaus Grobe, Markus Roppelt, Achim Autenrieth, Jörg-Peter Elbers, and Michael H. Eiselt. Cost and energy consumption analysis of advanced WDM-PONs. *IEEE Communications Magazine*, 49(2), 2011.

[27] Vincenzo Sciancalepore, Vincenzo Mancuso, Albert Banchs, Shmuel Zaks, and Antonio Capone. Interference Coordination Strategies for Content Update Dissemination in LTE-A. In *The 33rd Annual IEEE International Conference on Computer Communications (INFOCOM)*, 2014.

[28] Arash Asadi and Vincenzo Mancuso. Dronee: Dual-radio opportunistic networking for energy efficiency. *Computer Communications*, 2014.

[29] Christian Vitale, Gianluca Rizzo, and Vincenzo Mancuso. A coupled processors model for 802.11 ad hoc networks under non saturation. 2015.

[30] M Series. Guidelines for evaluation of radio interface technologies for imt-advanced. *Report ITU*, 2009.

[31] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. OpenFlow: Enabling Innovation in Campus Networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[32] Antonio De La Oliva, Albert Banchs, and Pablo Serrano. Throughput and energy-aware routing for 802.11 based mesh networks. *Comput. Commun.*, 35(12):1433–1446, July 2012.

[33] Ian F. Akyildiz, Ahyoung Lee, Pu Wang, Min Luo, and Wu Chou. A roadmap for traffic engineering in sdn-openflow networks. *Computer Networks*, 71(0):1 – 30, 2014.

[34] Pablo Serrano, Andres Garcia-Saavedra, Giuseppe Bianchi, Albert Banchs, and Arturo Azcorra. Per-frame energy consumption in 802.11 devices and its implication on modeling and design. *Networking, IEEE/ACM Transactions on*, 2014.

[35] Fatemeh Ganji, Lukasz Budzisz, Fikru Debele, Nanfang Li, Michela Meo, Marco Ricca, Yi Zhang, and Adam Wolisz. Greening campus WLANs: energy-relevant usage and mobility patterns. *Computer Networks, Special Issue on Green Communications*, 2014. accepted for publication.