CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

# Energy Efficient Control of Microwave Networks

Master's thesis in Computer science and engineering

Adam Frithiofson, Marcel Vacante

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2022

Master's thesis 2022

# Energy Efficient Control of Microwave Networks

Adam Frithiofson
Marcel Vacante

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2022

Energy Efficient Control of Microwave Networks

Adam Frithiofson, Marcel Vacante



Supervisor: Elad Schiller, Department of Computer Science and Engineering
Advisor: Anders Kvist, Ericsson
Examiner: Romaric Duvignau, Department of Computer Science and Engineering

Master's Thesis 2022
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2022

Energy Efficient Control of Microwave Networks

Adam Frithiofson, Marcel Vacante
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

# Abstract

Wireless microwave radio links have become a core component in modern communication infrastructure, providing access in places where fiber-optic cables are infeasible. High capacity links are commonly realized through bonding of multiple independent, parallel radio links, but utilization for these very rarely exceeds 50 %. This means that half of the radio link hardware is consuming power without providing any useful capacity. To combat this, we propose a software controller which can dynamically control power to radio hardware, adapting capacity to demand by powering off radios at periods of low utilization in order to reduce overall energy consumption. Results from computer simulations and measurements in a physical setup show that it is possible to save up to 50 % of energy consumed by radio hardware in these links with very small, if any, impact on users.

Keywords: microwave network, radio link, dynamic power management, energy-aware networks, green communication, energy efficiency

# Acknowledgements

We would like to thank our supervisor at Chalmers, Elad Schiller, for his continuous support and constructive feedback throughout the project. In addition, we would like to express gratitude towards everyone at Ericsson who have helped us at any stage in the project. In particular, we extend our special thanks to Anders Kvist and Martin Sjödin, who ensured we were always on the right track, that we had access to all the necessary equipment and data, as well as always being available for discussions about design or any obstacles we faced.

Adam Frithiofson and Marcel Vacante, Gothenburg, June 2022

# Contents

# List of Figures

# List of Tables

# 1

# Introduction

Since the rise of the internet and mobile networks, it is not just a possibility to communicate with anyone and anything at any time; the ability to do so is almost taken for granted. These communication technologies have become core infrastructure that society depends on and expectations on availability are high. Underground (or undersea) optical fibers are the favored carriers of information thanks to their high capacity, range and resistance to interference. However, their underground placement make them difficult and expensive to install. An alternative to these fibers is to create point-to-point microwave radio links that carry information directly through the air. In contrast to fibers, which require a single fiber cable to run all the way between endpoints, microwave radio links only require hardware to be installed at each endpoint. This enables bringing communication infrastructure to places that would otherwise be hard to justify economically. The use of such radio links is prevalent in construction of mobile networks as they allow base stations to be installed in places where fiber communication is infeasible, thus improving mobile networks coverage.

Unfortunately, microwave radio links are much less power efficient than optical fibers. Baliga *et al.* compare the energy consumption of fiber and wireless access networks and show that the wireless solution consumes tenfold the power for the same level of throughput [1]. This inefficiency not only has a negative impact on operators' electricity costs, but is also less sustainable from an environmental perspective. For these reasons, this thesis aims to investigate a means of reducing the energy consumption in microwave networks.

One such possibility for reduced energy consumption has been identified in microwave networks of Ericsson's customers. Utilization statistics have revealed that their radio links are often underutilized, as the capacity of these radio links is generally dimensioned to cope with peak demand with large margins. This leads to link utilization being significantly lower than actual capacity in many cases. Importantly, a significant number of these radio links are realized through aggregation of multiple parallel radio links, each one consisting of a dedicated pair of radio transceivers. This presents an opportunity to improve energy efficiency by turning off supply power to superfluous radios during periods of low utilization, thus reducing power consumption while only sacrificing unused capacity.

The goal of our work is to implement the aforementioned idea of dynamically turning off underutilized radios, as well as demonstrate that these energy savings can be realized without a significant impact on quality of service.

## 1.1 Aims and challenges

Our work aims to create a system for realizing these potential energy savings in microwave networks in the form of a software controller, and to demonstrate the viability of this scheme. To achieve this, there are several challenges which need to be considered. The most critical challenge is ensuring that the energy savings do not adversely impact the quality of service by causing capacity to be insufficient. This is difficult due to radio units having a significant startup time (8-10 seconds for the Ericsson MINI-LINK 6363 radios we use), and therefore spikes in demand can not immediately be met with an increase in capacity. Another challenge is that of fault handling, where unexpected failures in either a radio link or controller should be handled gracefully and never cause prolonged service outages. There is also the concern of potential hardware degradation, as power cycling radio units often may cause thermal stress and shorten their life expectancy. The final challenge is ensuring high scalability of our system. The system needs to work well in a network of any size, as the microwave networks where it is intended for use may consist of thousands of nodes and radio links.

## 1.2 Approach

We design and implement the aforementioned software controller as a platform, separating the performance-critical decision making into easily interchangeable decision-making algorithms that we call *policies*. The controller platform provides fault tolerance and scalability properties, whereas the performance in terms of energy savings and impact on quality of service is dictated by the implementation of a specific policy.

We implement two policies for use with our platform and evaluate their performance using a simulator that we create. The simulator simulates a policy's actions when to subjected to a given utilization pattern and outputs an estimate of saved energy as well as any impact on quality of service. The utilization patterns we feed the simulator with are gathered from radio links in a live microwave network, allowing us to estimate performance as if the system was deployed in that live network. We also evaluate performance in a physical testbed using our controller in order to compare with and validate the simulated results. Aside from performance evaluation, we evaluate the controller platform's robustness through fault injection experiments.

## 1.3 Related work

Dynamically adapting capacity to demand in order to save energy is not an original concept. Benini *et al.* surveys the field of Dynamic Power Management (DPM) and its

associated challenges, which primarily revolve around ensuring that capacity is always sufficient for current demand. Their survey focuses on systems with sub-second power state transition times and a centralized power manager [2]. Our work extends this to handle transition times in the order of 10 seconds and uses a distributed power manager (which we refer to as the *controller*) with fault-tolerant properties.

In the context of networking, there are some existing implementations of DPM. One of the most widespread examples is Adaptive Link Rate (ALR) for Ethernet, which lowers link speed when utilization is low in order to reduce energy consumption. In the case of ALR, changing the link speed typically takes less than 1 millisecond, which is unlikely to be noticed by users [3]. In our case, turning on radios takes upwards of 10 seconds and could have a significant impact on users.

Addis *et al.* propose a scheme in which they utilize a Software Defined Networking (SDN) controller to minimize energy consumption by shutting down underutilized nodes. However, such a scheme requires redundant paths to reroute traffic through [4], which are rarely present in microwave networks.

An example of DPM specifically in microwave radio links is Traffic-Aware Power Save (TAPS), a feature in Ericsson MINI-LINK radio link products which dynamically adjusts the strength of radios' transmitted signal to match the level of traffic. It never shuts off radios entirely [5], and is therefore only able to save around 4 watts per radio (for our model, MINI-LINK 6363). The scheme we propose does completely turn radios off, saving 17 watts per radio. While the energy savings of TAPS are comparatively low, it can adjust capacity very rapidly and does not come with any significant risk of user impact, unlike our scheme.

This work was inspired by recent developments in the area of self-stabilization, specifically, a control plane for software defined networks (SDNs) [6], [7], and edge computing [8]. These fault-tolerant solutions were based on advanced fault models, such as asynchronous message-passing systems with fairness assumptions [9]–[11], asynchronous crash-tolerant systems [12]–[14], and asynchronous Byzantine-tolerant systems [15]–[17]. Our work follows implicit synchrony assumptions and does not aim at analytically dealing (using formal proofs) with asynchronous fault models or the presence of malicious (Byzantine) behaviour. Nevertheless, we hope that the proposed solutions can serve as a starting point for more advanced fault-tolerant solutions via rigorous analysis.

## 1.4 Our contribution

We propose critical components for energy-aware microwave networks. Specifically, we design and implement the first, to the best of our knowledge, software-based controller that allows existing systems to become energy-aware. Our controller will continuously monitor link utilization and control power supply to radio units, turning them off when their capacity is not required. This will provide a significant reduction in energy consumption of many microwave radio links.

In order to realize the proposed solutions, we have developed a novel, distributed controller platform for reliable and dynamic management of supply power to radio units. The controller makes use of redundancy and automatic discovery and correction of faults in order to ensure reliable operation, which is of utmost importance for microwave network operators. We also provide two implementations of decision-making algorithms (also known as policies) to be utilized by the controller platform, which determine when it is efficient to turn off radios with minimal impact to users.

Through extensive evaluation that uses both hardware-in-the-loop testbed and simulation approaches, we show that our controller effectively handles the challenges posed by the scheme. The implementation was shown to be reliable in extended operation and throughout various fault injection experiments in the testbed system. Simulations on 112 real-world microwave radio links show an average power reduction of 34 watts and an average traffic impact of 30 milliseconds per day.

We expect microwave product manufacturers to use our work to provide more energy-efficient products, which will lower the energy consumption of microwave networks that use these products. This will reduce the carbon footprint and operating costs of microwave networks, making them more sustainable both environmentally and economically.

## 1.5 Organization

Microwave networks and their constituent components are described in Chapter 2, in addition to the challenges in making microwave networks energy-aware. Our implementation of an energy-aware controller for microwave networks is described in Chapter 3. In Chapter 4, we describe the decision making algorithms (policies) that the controller uses to determine when to enter or exit energy saving modes. How we evaluate the controller and the results of the evaluation are presented in Chapter 5 and Chapter 6, respectively. Finally, we provide conclusions of our work in Chapter 7.

# 2

# Background

This chapter introduces the fundamentals of microwave networks, their components and how these interact, the emerging concept of energy aware microwave networks, and its associated challenges.

## 2.1 Microwave networks

A *microwave network* consists of *nodes* that serve the same basic purpose as switches in traditional Ethernet switched networks. However, a distinguishing feature is the capability of utilizing wireless transceivers, more commonly referred to as *radio units* or just *radios*, allowing nodes to communicate wirelessly. A wireless connection between two nodes is referred to as a *radio link*, and is established by configuring a matching number of radios on both nodes.

The simplest example of such a radio link is one with two radios; one present at each node. The more interesting case, which our work is centered around, is when several radios are present at each node, creating multiple parallel, but independent, communication channels. These communication channels are commonly referred to as *carriers* and their capacity is typically combined into a single logical link using *Radio Link Bonding* (RLB). RLB provides both an increase in capacity as well as a level of failure resistance as the radio link can remain functional, albeit at reduced capacity, despite the complete failure of one or more redundant carriers. A two carrier bonded radio link is illustrated in Figure 2.1.

## 2.2 Radio link utilization

Radio link utilization, henceforth referred to simply as utilization, is a measure of how much of a radio link's maximum capacity is used at a given time. Utilization is heavily dependent on the users of a specific radio link, and as a result no two links have the exact same utilization pattern. However, there are some clear trends which the vast majority of all radio links seem to follow, the most important of which is a low level of utilization a majority of the time. This is illustrated in Figure 2.2, which depicts a utilization pattern captured in a live network for a real two-carrier bonded radio link. The utilization for this link rarely exceeds 50 %, which is very

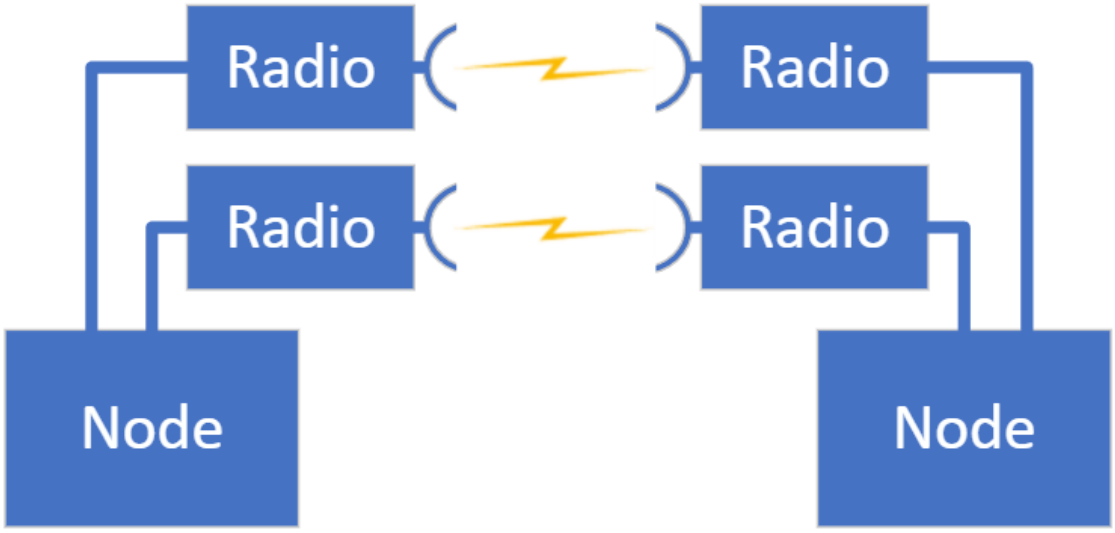


**Figure 2.1:** Illustration of a bonded radio link consisting of two carriers

common for radio links in our data set. As such, there is a great opportunity to save energy by keeping one of the two carriers powered off the majority of the time, only powering it on during the brief moments where the extra capacity is needed.

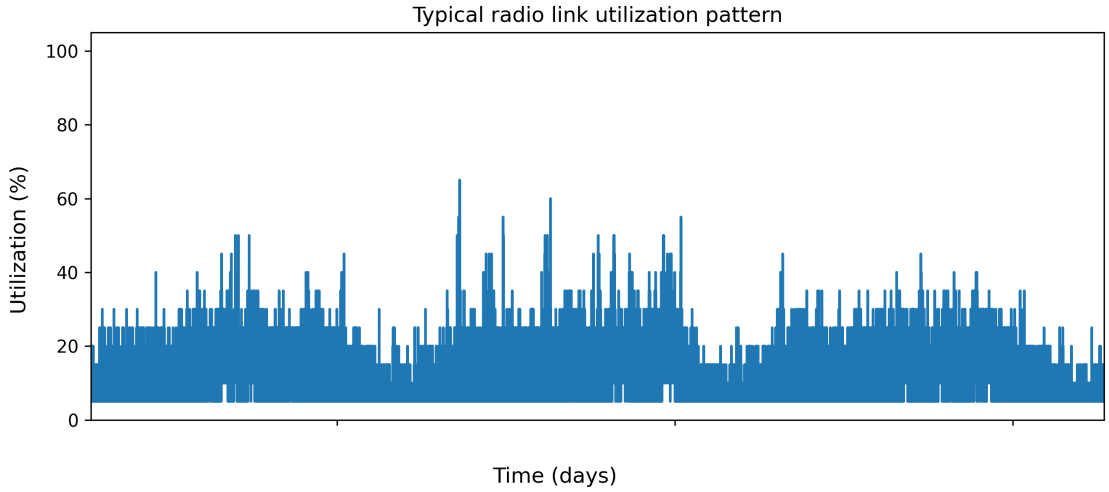


**Figure 2.2:** A plot of radio link utilization over time from a typical radio link in a live microwave network

There is also a clear periodicity to utilization patterns in most radio links, where utilization is higher during daytime, and lower during nighttime. Additionally, utilization one day tends to be similar to that of the previous day. Both of these trends can be easily seen in Figure 2.2. While not depicted here, weekends tend to have different utilization to weekdays, resulting in week-to-week regularities being even stronger than day-to-day regularities. These regularities can be exploited in order to make predictions about future utilization.

## 2.3 Energy-aware microwave networks

The concept of *energy-aware microwave networks* refers to microwave networks that adapt themselves to current circumstances in order to reduce their energy consumption. Our proposed method for introducing energy awareness to microwave networks is to power off radio units in bonded radio links when their capacity is not utilized, which is generally a majority of the time.

However, while operating in a state of reduced capacity is beneficial for energy consumption, it is not without problems. There are three main challenges with employing such a scheme: (1) ensuring capacity is always sufficient, (2) handling of faults and (3) avoiding hardware wear.

In regards to the first challenge (1), consider the event of a sudden increase in traffic exceeding current capacity while operating in a power saving state. Additional carriers would need to be powered on to meet the sudden demand, but performing such an operation is not instantaneous. The radios we use (Ericsson MINI-LINK 6363) take between 8 and 10 seconds to be fully operable after being powered on, and as a result, powering off radios at inopportune times would cause current demand to exceed capacity. When demand exceeds available capacity, a radio link is said to be *congested*.

Congestion has a significant impact on *quality of service* (QoS), which is highly undesirable by network operators. In case of congestion, nodes will start buffering data that cannot be transmitted, delaying its delivery, and discard data when buffers become full. Congestion can be mitigated by employing traffic prioritization and ensuring that only low priority traffic is dropped or delayed, while letting more critical high priority traffic flow undisturbed. Another mitigating factor comes from a majority of internet traffic being TCP-based [18]. With TCP, actual utilization will to some degree adapt to the reduced capacity. This is due to TCP's loss recovery mechanism of reducing a sender's transmission rate upon detecting congestion [19]. This will reduce the severity of the congestion and the amount of buffered or discarded traffic, although the quality of service will still be impeded since the actual throughput the users see is reduced.

Another problem with ensuring sufficient capacity (1) is that a radio link's capacity is not always constant. Environmental circumstances such as rain can degrade signal quality and cause capacity to be temporarily reduced. In such a case, it may be necessary to have multiple carriers active to handle a utilization level for which a single carrier would normally be sufficient.

In regards to the second challenge of fault handling (2), employing the energy saving scheme introduces new fault scenarios that need to be considered. Reducing the number of redundant radios active at any given moment makes the system more vulnerable to hardware failures and transient faults. In a two carrier configuration with no power saving, one carrier could fail and still allow the link to be operable with reduced capacity. As long as link utilization is below the capacity of this one

carrier, there would be little to no impact on QoS. Compare this to the case of being in the power saving state, i.e. having only one out of two carriers powered on. Should this carrier fail, the impact on traffic will be major as no other carriers are available to take its role immediately. Due to the aforementioned start-up time of radios, this effectively translates to at least 8-10 seconds of total service outage in the best case. If there is no mechanism to automatically recover a radio link in such a situation, the resulting outage may last until the link is restored manually by an on-site operator.

For the third and final challenge (3), there are possible problems relating to hardware degradation when power cycling hardware frequently. The main concern is that a more varied power consumption will cause hardware temperature to vary as well. This can in turn cause mechanical stress (due to thermal expansion and contraction) or condensation, both of which can negatively impact the longevity of hardware.

# 3

# Controller design

In order to control the power state of radio links, we introduce a *controller* entity to the system, similarly to how a controller in software-defined networking (SDN) manipulates the configuration of the network over which it has authority [20]. The controller will execute some *policy* that defines which power state is suitable in a given situation, and apply this state by turning on or off power to radios. The primary purpose of the controller is to reduce the energy consumption of the network, although it should do so with minimal impact for users. The following paragraphs highlight the key design criteria for the controller.

**Congestion minimization** If demand exceeds capacity, congestion will occur and adversely impact users of the network, which is highly undesirable by network operators. Therefore, the controller should minimize the amount of congestion introduced to the largest extent possible.

**Minimization of power state changes** Since frequent power state changes may have a negative impact on radio hardware longevity, the controller should avoid turning on and off radios excessively often.

**High scalability** Microwave networks may consist of thousands of nodes, and as such the controller must be highly scalable. A controller which scales poorly would be infeasible to deploy in a real network.

**Fault handling** The introduction of the controller to the system should not worsen fault handling or operation compared to if it were absent. For example, the controller should never cause a link to become stuck in a low-capacity state, even in case of a fault in the controller. The controller should be able to identify problems on its own and apply countermeasures to recover, gracefully handling unexpected faults, also known as being *self-healing* [21].

## 3.1 Architecture

The architecture we have opted to use for our controller is a distributed system, in which a controller instance is placed at the location of each node. Each controller

consists of multiple workers, one for each radio link at its local node. This means that each radio link is under the management of two workers, one at each endpoint. The two workers can communicate with both nodes and each other via a control plane that relies on their shared radio link, as opposed to a dedicated communication channel. This is also known as an *in-band control plane*. However, because the control plane is in-band, this communication will be disturbed if the radio link were to go down. In case of such a network partition, a worker would only be able to reach its local node and not the remote node or worker.

The selected architecture allows us to consider each radio link as an independent entity, meaning controllers only need to communicate with their immediate neighbours. As such, this scheme will scale to arbitrarily large networks, as long as the controller is dimensioned according to the maximum number of possible neighbours. In addition, this scheme provides a level of redundancy due to each radio link being managed by two independent workers on different controllers, where the failure of one worker will not result in a radio link getting perpetually stuck in a power saving state.

While a distributed controller has major advantages, it also introduces a new challenge. With two workers managing a single radio link, there is the possibility of disagreement, which would require some synchronization mechanism to prevent or resolve. We have opted to solve this by introducing a master-slave relationship between the two workers. The master is responsible for the decision-making and the slave simply follows what the master decides. In order to determine which worker should be master and which should be slave, there needs to be some distinguishing factor between the candidates [22, p. 329]. For this, we use the IP address of the controller, where the worker on the controller with the lower IP address will act as the master worker.

Our implementation will run on dedicated hardware, separate from the nodes, with each controller having a physical connection to its local node. However, the same architecture can be used for an implementation inside the nodes themselves if desired. Figure 3.1 illustrates our architecture in a network with two radio links between three nodes.

## 3.2 Worker operation

The workers have different behaviour depending on whether they are a master or slave. The slave worker begins by powering on all local and remote radios in order to reach a safe and well-defined state. During startup, the worker will send commands to both nodes in order to ensure all radios are turned on in case the remote worker has failed and is unable to manage its local radios. After synchronization, each worker will only control the state of its local node's radios. When all radios are on, the slave sends a message to the master, asking it to synchronize, which the master will do by restarting and following its own startup procedure. Once the master has acknowledged the slave's request, the slave will enter its main loop and wait for the master to send a message containing a new power state. When the slave receives

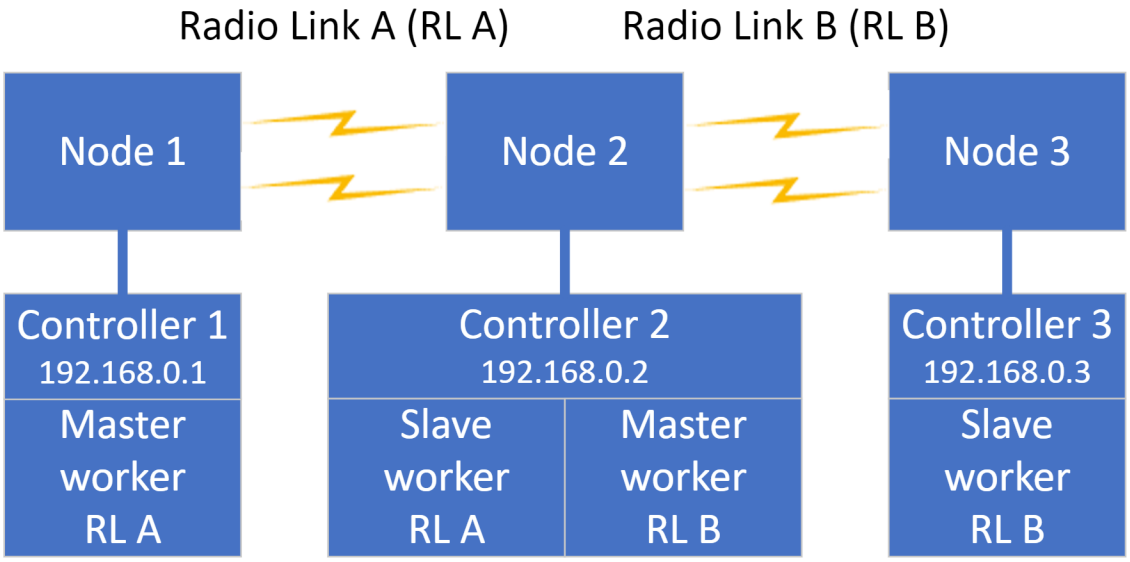


**Figure 3.1:** Distributed controller architecture with three interconnected nodes

such a message, it will apply the new state to its local node, turning on or off radios accordingly.

The master worker performs the same initial step as the slave, powering on all local and remote radios. It then sends this state (all radios on) to the slave worker to synchronize the state of the slave with its own, subsequently awaiting an acknowledgement. Afterwards, the master verifies that all carriers are up and running to ensure that the radio link is operating normally and has not, for example, suffered a hardware failure in one of the carriers. Once this has been verified, the master proceeds to its main loop, in which it starts executing the power saving scheme. Here, the master worker uses a policy (described in detail in Chapter 4) to determine how much capacity is desired at any given moment. It then calculates how many carriers are necessary to provide the desired capacity, taking into account that the current capacity provided by a carrier is affected by environmental circumstances such as rain. When the master worker deems a change to the power state to be desirable, it will communicate this to the slave and await an acknowledgement, after which both workers apply this new state to their respective node by turning on or off radios.

## 3.3 Fault handling

Since reliability and fault handling is critical, the controller is designed to be self-healing by making use of the *restart on failure* principle to return to a correct state whenever a problem is encountered. It has been widely observed that restarts often solve the problem of anomalies and unexpected behaviour [23], [24]. As such, designing a system around this principle together with a robust startup procedure provides a single course of action that can be used to recover from a wide variety of faults. This includes any fault that causes the system to crash and restart, even ones that were not explicitly considered during development.

For failure detection, we primarily make use of heartbeats between the workers. Every second, the worker will send a heartbeat request to the remote worker, after which it expects a reply within a second. If a response is not received within this time frame, there is a problem with either the remote worker or the communication over the radio link. Since all faults are handled by restarting, the worker simply needs to restart and rely on the startup procedure to recover. Most other failures do not need any explicit detection or handling. For example, if we do not succeed to turn on a radio due to failed hardware, an exception will be raised, causing the worker to restart, after which the startup procedure is expected to handle this situation as gracefully as possible.

Something that does require explicit handling is self-checks for desynchronization between actual radio states and the internal state of the worker. The workers will continuously verify that all carriers it thinks should be up are actually providing capacity. If a state of desynchronization persists for longer than 60 seconds (which would happen if a radio is down, but the controller thinks it is up), the controller will restart itself to recover.

## 3.4 Redundancy

It is not acceptable that an unexpected controller failure leads to a radio link being stuck in a reduced capacity state, as this could lead to congestion. To prevent this, we ensure a certain level of redundancy in the controller. Specifically, we require that both the master and slave workers are alive in order to enter or maintain any power saving state. Should a worker fail, it will be detected by loss of heartbeats, and the remaining worker will restart and bring the radio link to full capacity. The remaining worker will then attempt to resynchronize with the failed worker, failing and restarting repeatedly with the link remaining at full capacity. Only when the failed worker has recovered will synchronization succeed, redundancy be restored and power saving can be resumed.

With this level of redundancy, one simultaneous worker failure per radio link is tolerated. However, a simultaneous failure of both workers may still leave a link in an undesired power state until either worker has recovered and can assume control over the radio link's power state.

# 4

# Decision-making algorithms (policies)

While the controller is a platform that is responsible for fault-handling and communicating with nodes to power on and off carriers, it does not make decisions regarding *when* to power on and off carriers. Instead, this responsibility is delegated to an interchangeable algorithm which we refer to as a *policy*. The policy monitors current utilization and outputs the percentage of total maximum capacity that is currently desired. An output of 50 % indicates that the number of active carriers should be enough to provide 50 % of the maximum link capacity, and assuming two identical carriers this would mean that one of them should be powered on. Note that the policy is not responsible for exactly which radios should be powered on, as this the responsibility of the controller.

We investigate two different policies for our energy saving scheme: a reactive policy which only considers recent utilization, and a reactive and proactive hybrid policy which also tries to anticipate the future based on utilization data from the previous week.

## 4.1 Reactive policy

One of the simplest policies is one that only considers instantaneous utilization and adjusts capacity accordingly. However, without any modifications, this simple policy has two major flaws. Firstly, it will not attempt to increase capacity until the current capacity is exceeded, likely causing a large impact on QoS, and secondly it will be very unstable when the current utilization fluctuates near the threshold for turning on and off a radio, causing frequent power state changes.

To solve the first problem, a *capacity margin* can be applied to the threshold of when to turn on and off a carrier. For example, if the current utilization is 45 %, we might want the capacity of the radio link to be at least 55 %, i.e. a 10 % capacity margin. As such, in a typical two carrier setup, an increase in utilization beyond 40 % would cause the second carrier to be powered on. With this improvement, slow gradual increases in utilization would be handled gracefully as carriers will be turned on well before there is risk of congestion. Figure 4.1 illustrates how the

capacity margin works to increase radio link capacity without congestion during a slow gradual increase in traffic load.

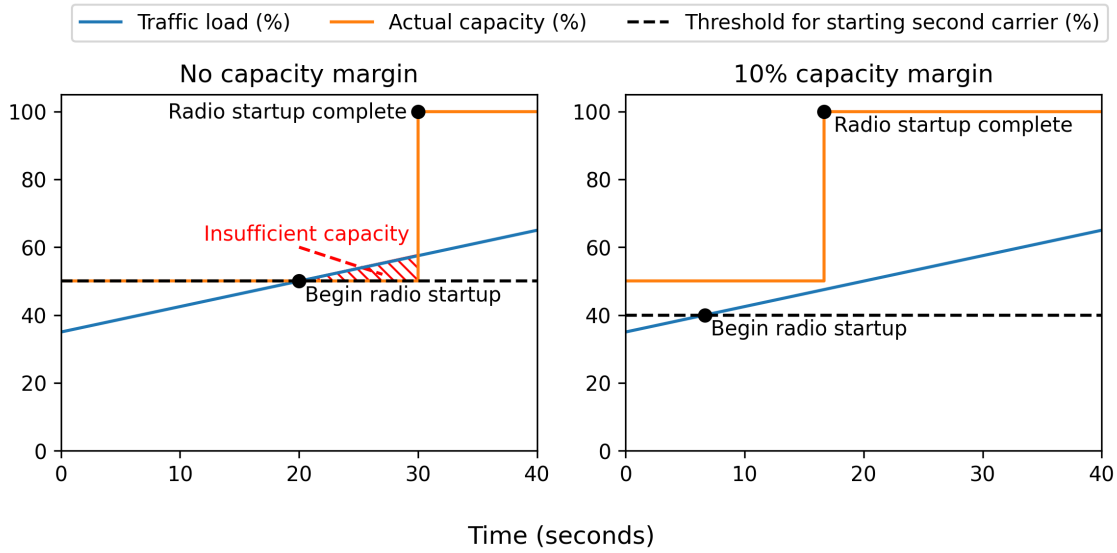


**Figure 4.1:** Illustration of how the capacity margin helps prevent congestion during slow gradual increases. Note how there is a period of insufficient capacity in the left plot while the radio is starting, whereas capacity is always sufficient in the right plot thanks to the capacity margin reducing the effective threshold for turning on the second carrier from 50 % to 40 %.

Unfortunately, slow gradual increases are not representative of all utilization patterns. Utilization of radio links can fluctuate rapidly, which becomes a problem when the utilization is close to the threshold at which additional carriers will be powered on. This would lead to carriers being turned off, only to be turned on again a short while later, likely introducing congestion during the startup and shortening the lifespan of the hardware. Therefore, we introduce a time hysteresis to the policy, requiring that a radio stays powered on for a minimum length of time. In practice this is realized by the policy outputting a desired capacity equal to the highest utilization peak observed during a sliding window, plus any capacity margin.

This improved policy is what we will refer to as our *reactive policy*, and is illustrated in Algorithm 1 with pseudo-code. The 10 % capacity margin and the 2 hour time window were selected based on what we believe to be reasonable values in most cases. As fine-tuning of the policies is not a focus area of this project, we do not quantitatively evaluate the effects of different values. However, it is expected that increasing these values will decrease the risk of congestion further and reduce the number of power state changes, but at the cost of increased energy consumption. This is due to radios being turned on at a lower utilization level, as well as being kept turned on for longer. Both of these values can easily be adjusted if desired.

**Algorithm 1** Pseudo-code for the reactive policy implementation

```
CAPACITY_MARGIN := 10 percent                          ▷ Adjustable value
SLIDING_WINDOW_SIZE := 2 hours                         ▷ Adjustable value
slidingWindowUtilization := new SlidingWindow(SLIDING_WINDOW_SIZE)
function ReactivePolicy(currentUtilization)
    slidingWindowUtilization.update(currentUtilization)
    return CAPACITY_MARGIN + max(slidingWindowUtilization)
end function
```

## 4.2 Hybrid policy (reactive and proactive)

A potential improvement to the reactive policy detailed above is to not only consider the most recent utilization, but also historical utilization. A prominent trend in real-world data is that utilization patterns tend to be very regular, where utilization around a specific time one week is very similar to utilization around that same time the previous week (or weeks). The same sort of regularity also exists between days, although weekends tend to be different as can be seen in interval B in Figure 5.2.

Our *hybrid policy* implementation is an extension of the reactive policy that attempts to predict the future by looking at the past. Aside from considering the recent past in a *lookbehind* window (identical to the sliding window in the reactive policy), it also considers what it expects to happen in a future *lookahead* window, where the future is represented by the same time interval the previous week. When the prediction is accurate, the policy correctly anticipates future demand and can increase the capacity well in advance, avoiding potential congestion even if demand increases suddenly. The difference between what the hybrid and reactive policy considers when making decisions is shown in Figure 4.2.

Algorithm 2 shows pseudo-code for the hybrid policy. The values for capacity margin and lookbehind were retained from the reactive policy and the new lookahead value was selected to be 1 hour based on what we believe to be reasonable. Increasing the lookahead is expected to have the same effect as increasing the lookbehind or capacity margin, resulting in more conservative energy savings as radios are kept powered on for longer. All three values can be adjusted if desired.

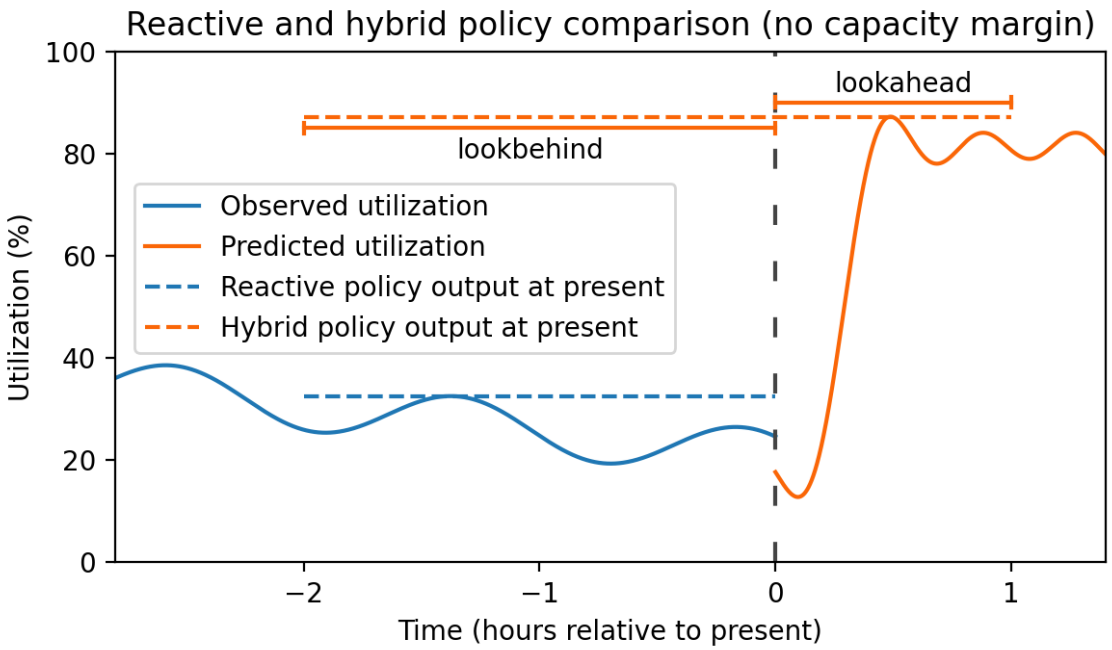


**Figure 4.2:** Comparison of policy output for the reactive and hybrid policy, with no capacity margin for the purpose of illustration. The reactive policy only considers the highest peak of the last two hours, whereas the hybrid policy also considers a predicted peak in the next hour based on data from the previous week.

**Algorithm 2** Pseudo-code for the hybrid policy implementation

```
CAPACITY_MARGIN := 10 percent                    ▷ Adjustable value
LOOKBEHIND := 2 hours                            ▷ Adjustable value
LOOKAHEAD := 1 hour                              ▷ Adjustable value
history := new History()
function HybridPolicy(currentUtilization)
    now := getCurrentTime()
    history.add(now, currentUtilization)
    return CAPACITY_MARGIN + max(
        history[now - LOOKBEHIND, now],
        history[now - ONE_WEEK, now - ONE_WEEK + LOOKAHEAD],
    )
end function
```

# 5

# Evaluation

The evaluation of the proposed energy savings scheme is divided into two separate parts: controller and policy evaluation. The controller implementation is evaluated in a hardware setup, whereas the policies are primarily evaluated using a purpose-built simulator that can simulate months' worth of traffic data on hundreds of radio links in a short span of time. Some policy evaluation is performed in real-time in the hardware setup as well, to compare with simulated results and provide confidence in the validity of the simulator.

## 5.1 Data set

In order to replicate real-world scenarios and run meaningful computer simulations, some form of traffic data is needed to simulate network traffic. While synthetic such data could be obtained through traffic modelling, we have opted to use utilization statistics that were gathered from a live microwave network in order to obtain more realistic traffic data. Ericsson has supplied utilization statistics from the microwave network of one of its customers. This data set contains 2.3 million 15-minute interval summaries, constituting about 4 months' worth of utilization data for 192 radio links. Out of these, 119 are bonded links consisting of two carriers and the remaining links are not bonded. There are no radio links with more than two carriers present in the data set.

### 5.1.1 Format

The format and available values for the 15-minute interval summaries are as follows:

1. Timestamp
2. Unique radio link identifier
3. Amount of time spent in 5 % utilization buckets

An example of one entry in this data set is shown in Table 5.1

| Field | Value |
|---|---|
| Timestamp | 2021-08-23 14:00 |
| Radio link identifier | 1155 |
| 0 % to 5 % | 308 s |
| 5 % to 10 % | 547 s |
| 10 % to 15 % | 45 s |
| 15 % to 20 % | 0 s |
| ... | 0 s |
| 95 % to 100 % | 0 s |

**Table 5.1:** A sample of utilization statistics for one 15-minute interval

### 5.1.2 Limitations

The data set has some limitations in regards to resolution, the most significant being that there is no way of telling what the utilization was at a specific point in time. For the example data in Table 5.1, we know that there are a total of 45 seconds where utilization was between 10 % and 15 %, but we do not know if these were all consecutive or if they were spread out inside the interval.

Another limitation is the low resolution of 5 % buckets. For a radio link with a capacity of 1 Gbit/s, it is not possible to tell the difference between a utilization of 51 Mbit/s and 99 Mbit/s since both fall in the 5 % to 10 % utilization bucket.

Lastly, even though the data set contains several months' worth of data, it is not entirely contiguous due to gaps every couple of weeks. We do not attempt to splice or interpolate these gaps, but instead treat each contiguous interval separately.

### 5.1.3 Data enhancement

In order to perform meaningful computer simulations and simulate scenarios on real hardware, we need realistic, high-resolution utilization data. Ideally, we would replay packet captures from live radio links for maximum authenticity, but since we do not have access to any such packet captures we will instead enhance our data set to provide second-by-second data.

We construct our enhanced data set by expanding each 15-minute interval summary, where we know the utilization for all 900 seconds but not the order. As a best-effort approximation, we randomize the order of these seconds within the 15-minute interval to obtain second-by-second data. The utilization value for each second is rounded to the upper end of the utilization bucket, so a second in the 0 % to 5 % bucket will be treated like a second of 5 % utilization. To illustrate this process, enhancing a 5-second interval with 3 seconds in the 0 % to 5 % bucket and 2 seconds in the 5 % to 10 % bucket, the resulting second-by-second interval could be [5, 10, 5, 5, 10] or any permutation thereof. A more extensive sample of such enhanced data is shown in Figure 5.1, with its original interval summaries shown in Table 5.2.

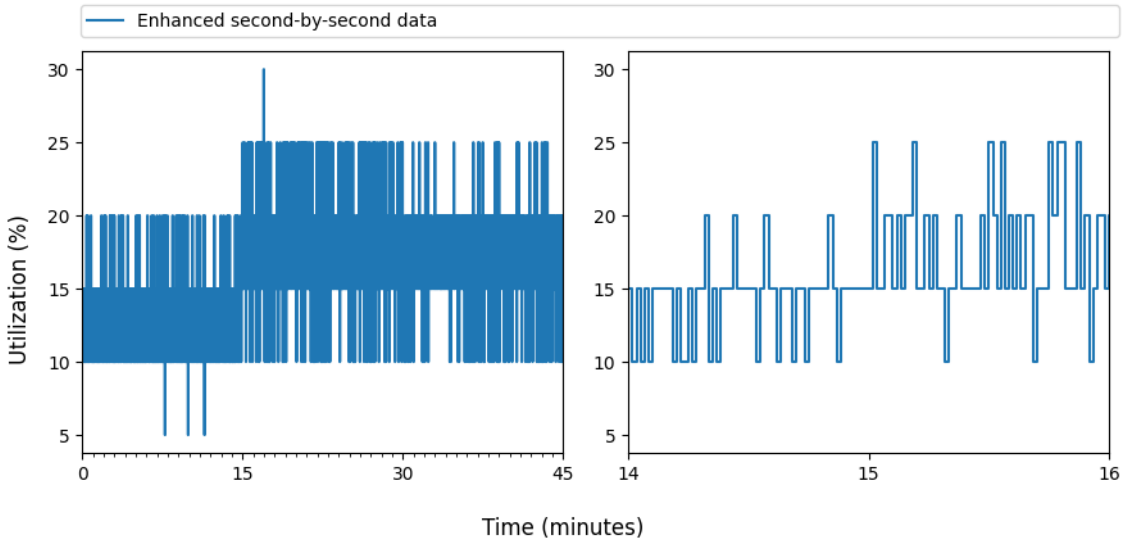


**Figure 5.1:** Sample of enhanced 15-minute intervals at two different time scales. The original 15-minute intervals summaries are described in Table 5.2.

| Bucket | 0-15 minutes | 15-30 minutes | 30-45 minutes |
|---|---|---|---|
| 0 % to 5 % | 3 s | 0 s | 0 s |
| 5 % to 10 % | 330 s | 63 s | 70 s |
| 10 % to 15 % | 509 s | 508 s | 575 s |
| 15 % to 20 % | 58 s | 225 s | 232 s |
| 20 % to 25 % | 0 s | 103 s | 23 s |
| 25 % to 30 % | 0 s | 1 s | 0 s |
| ... | 0 s | 0 s | 0 s |
| 95 % to 100 % | 0 s | 0 s | 0 s |

**Table 5.2:** The original 15-minute interval summaries from which the enhanced data in Figure 5.1 was created

This enhancement is not entirely reflective of real-world utilization patterns. It is quite likely that adjacent seconds in the real world have quite similar utilization, which a random order does not preserve. This will cause the data to appear more bursty compared to authentic data, and therefore likely be more difficult for a policy to handle. Therefore, a policy that performs well on this enhanced data should perform equally well or better when deployed on a live radio link.

An overlooked improvement to this enhancement process would be to also randomize the utilization within a bucket rather than just rounding it up. This would mean that a second in the 5 % to 10 % could end up as any value in this range, for example 7.213 %, and not always be rounded to 10 %. This would likely be even more difficult for a policy to handle, given that it introduces more variability to the data. As such, it would yield a better worst-case estimation of a policy's performance.

### 5.1.4 Cherry-picked data for detailed analysis

Analyzing policy behavior in detail on the entire data set is not feasible due to the large size of the data set. We therefore only do more detailed analysis on a subset of four intervals that were cherry-picked to be representative of the most common patterns in the data set, in addition to consisting of long continuous intervals.

- **Interval A** is the most common case, where utilization slowly increases and decreases throughout the day with a regular pattern. The vast majority of two carrier bonded radio links in our data set follows this pattern, albeit with varying amplitude.
- **Interval B** is a more extreme case where utilization rapidly increases and decreases once a day except for weekends with a very regular pattern.
- **Interval C** is from a link with very high and variable utilization, but still with some day-to-day and week-to-week regularities.
- **Interval D** is more irregular with no obvious repeating pattern.

Figure 5.2 illustrates these four intervals.

## 5.2 Controller evaluation

The purpose of the controller evaluation is to provide confidence in the stability of the controller design and implementation, both during normal operation and recovery after the occurrence of faults. As such, the controller evaluation considers both a week-long evaluation and fault injection experiments in a hardware testbed.

### 5.2.1 Hardware testbed

In order to evaluate the controller, we use a testbed consisting of microwave radio link equipment, general-purpose computers where we execute the controller, and an Ethernet traffic instrument. An overview of the physical hardware setup is shown in Figure 5.3.

**Radio link equipment - MINI-LINK** For radio link equipment, we use Ericsson MINI-LINK products. Our system consists of two MINI-LINK 6600 nodes that are connected by two pairs of 15 GHz MINI-LINK 6363 radios, thus providing two parallel, but independent radio links. The radio links are combined into one logical link through the use of Radio Link Bonding (RLB).

In our setup, radios are not communicating wirelessly using antennas as this is infeasible in our lab environment. Instead, radio pairs are connected via coaxial cables that carry the radio signal from one radio to the other. In a wireless outdoor setup, signal quality can be affected by environmental circumstances such as weather, causing the capacity of the radio link to vary over time. This is not the case when

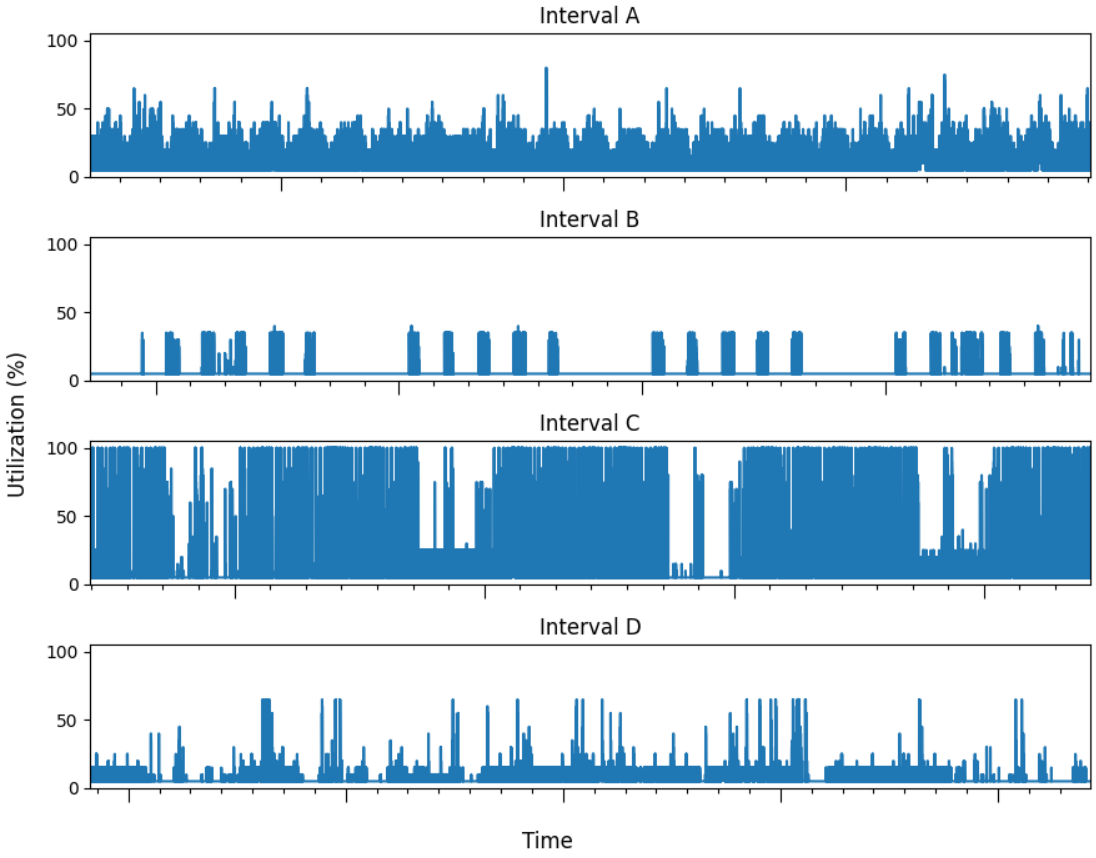


**Figure 5.2:** Utilization data for the cherry-picked intervals. Major ticks mark the beginning of a new calendar week and minor ticks the beginning of a new calendar day.

using a coaxial cable, but it is possible to emulate this behavior by making use of a feature in the MINI-LINK node which allows the capacity of a carrier to be artificially reduced.

The MINI-LINK node provides, through internal debug interfaces, a means of controlling the power supply to each individual radio unit. This is what use to power off radios. The node also provides utilization statistics and information about the current capacity provided by each carrier, which the controller makes use of to determine which radios to turn off.

The software version we are using for the MINI-LINK nodes is an internal development build of the generally available 1.17.1 software. A development build is necessary as the ability to control the power supply to radio units is not supported functionality in any officially released software to date.

**Controller hardware - Raspberry Pi** For the purpose of executing our controller software, we utilize two Raspberry Pi 3B+ single board computers running Raspberry Pi OS version 11 (bullseye). Each Raspberry Pi is directly connected to its adjacent MINI-LINK node via an Ethernet link, with no direct connection to the opposite

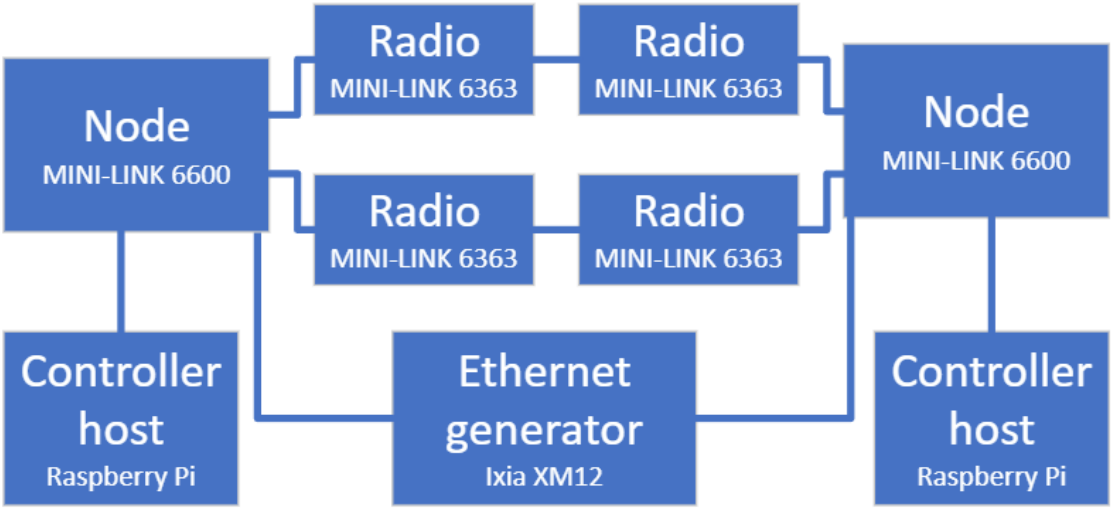


**Figure 5.3:** Overview of the physical testbed setup

node or controller. That is, the only way for one Raspberry Pi to communicate with its remote counterpart is via the radio link.

**Traffic instrument - Ixia** In order to test and evaluate the controllers' ability to regulate the system according to different levels of traffic, a traffic generator is required. For this purpose we use an Ixia XM12, capable of outputting 10 Gbit/s per port, which is connected to each MINI-LINK node via a 10 Gbit/s Ethernet link. Apart from generating traffic streams, the Ixia also provides measurements of performance statistics such as lost or delayed Ethernet frames for these streams, which is necessary for the evaluation.

**Energy measurements** For energy measurements in our hardware setup, we use an energy meter that is built into the MINI-LINK 6600 power supply. This provides a measurement of energy consumption for the entire node and not for individual radio units, but it should still be sufficient for our use case, which is primarily be validating estimations from computer simulations.

Since we introduce additional hardware in the form of Raspberry Pis to the system, we also measure their power consumption with a power meter. However, the result of this measurement is not used to judge the success of the controller, as further development of the concept will most likely not make use of any additional hardware. Instead, it will be integrated into the nodes' software, eliminating the need for additional hardware and making the power overhead negligible.

### 5.2.2 Week-long evaluation

The purpose of the week-long evaluation is to validate the stability and behavior of the controllers when operating for prolonged periods of time. In addition, we compare the results to those from a simulated run with the same policy and week-long traffic pattern in order to verify correctness of the simulator. For this evaluation,

the reactive policy is used since the focus of this evaluation is on correctness of the controller rather than policy performance, and the hybrid policy adds unnecessary complexity.

To perform this evaluation, the system is allowed to run without interruption for one week. During this time, the traffic instrument pushes traffic over the radio link according to the utilization data of interval A described in Section 5.1.4. This interval was chosen because its utilization is both periodic and high enough to allow for several interesting policy decisions to be made.

We consider the following criteria in the week-long evaluation: energy consumption of the nodes, impact on QoS, and how often the radio units are turned on or off. Energy consumption is measured by the nodes' built-in energy meter. QoS impact is measured in lost Ethernet frames and the maximum frame delay observed, where these metrics are provided by the traffic instrument. The number of times radios are turned on or off is recorded in logs of the controller. Finally, there should not be any unexpected behavior such as controller restarts during this week-long experiment.

### 5.2.3 Fault injection experiments

In order to evaluate the controller's fault handling, we perform multiple experiments that provoke the entire system in various ways and observe how it recovers. As these experiments are only focused on the correctness and stability of the controller and not policy decisions, we use a simple and predictable dual threshold policy that controls the power state directly based on current utilization. It turns off a carrier once instantaneous utilization is below 40 %, and turns it on again above 45 %.

The specific fault scenarios that are injected into the system and the expected outcomes are described in Table 5.3 through Table 5.11.

**Table 5.3:** Fault handling in case of hardware failure in only active carrier

| | |
|---|---|
| **Scenario** | The only active carrier suffers a complete failure. Three different causes are simulated: antenna failure, radio hardware failure and failure of the node's modem expansion card that is used to drive the radio.<br>**Antenna failure** is simulated by physically disconnecting the coaxial cable carrying the radio signal between the radios.<br>**Radio failure** is simulated by physically disconnecting the cable between the radio and the node.<br>**Modem failure** is simulated by physically removing the modem plug-in card from its slot in the node. |
| **Expected outcome** | The radio link is broken, causing communication to be down between the controllers. Controllers detect this by lost heartbeats and restore power to the other carrier(s), bringing the radio link back up. The radio link should not be completely down for more than 15 seconds. Until the failed carrier is restored (through manual action) and the radio link is completely healthy again, the controllers should not attempt to perform any power saving. |

**Table 5.4:** Fault handling in case a controller loses network connectivity

| | |
|---|---|
| **Scenario** | The Ethernet link between a controller and its node fails. This is simulated by physically disconnecting the Ethernet cable from the Raspberry Pi. |
| **Expected outcome** | The remaining controller detects the fault by lost heartbeats. Since redundancy is now lost, it should bring all carriers up and keep them in that state. If the Ethernet link is restored, the controllers should synchronize and normal operation should be resumed. |

**Table 5.5:** Fault handling in case a node stops responding with utilization data

| | |
|---|---|
| **Scenario** | The node connected to the master controller stops responding to requests for current utilization and carrier capacity. This is simulated by blocking these request with a firewall. |
| **Expected outcome** | The master controller crashes and restarts. Then, as part of the regular startup procedure, it restores power to all radios. When the node starts responding normally to requests, normal operation should resume. |

**Table 5.6:** Recovery after node crash

| | |
|---|---|
| **Scenario** | One of the nodes crashes and needs to restart to recover. This is simulated by removing and restoring the supply power to the node. |
| **Expected outcome** | The radio link is completely down until the node has restarted. When the node is back online after the restart, the controllers should synchronize and resume normal operation. |

**Table 5.7:** Recovery after host computer crash

| | |
|---|---|
| **Scenario** | One of the Raspberry Pis crashes and needs to restart to recover. This is simulated by removing and restoring the supply power to the Raspberry Pi. |
| **Expected outcome** | The remaining controller detects the crash by lost heartbeats. Since redundancy is now lost, it should bring all carriers up until the crashed controller has restarted, at which point normal operation should be resumed. |

**Table 5.8:** Recovery from partial or complete controller crash

| | |
|---|---|
| **Scenario** | The controller software suffers a fault that causes a worker thread, policy thread, heartbeat thread or the main thread to stop unexpectedly. This is simulated by manually stopping one of these threads. |
| **Expected outcome** | The controller continuously performs self-checks to ensure all threads are alive. Depending on which threads fail, either the worker restarts itself, the main thread recreates the worker or if the entire controller program stops, it is restarted by the Raspberry Pi's operating system as the controller is registered as a system service with a directive to always restart if it is not running. |

**Table 5.9:** Recovery from radio state desynchronization where a radio is actually off but the controller thinks it is on

| | |
|---|---|
| **Scenario** | Turning on a radio silently fails and the state in the controller (on) and the actual state of the hardware (off) is desynchronized. This is simulated by turning off a radio without informing the controller while running a fixed traffic level which is high enough for the second carrier to be needed. |
| **Expected outcome** | The capacity for that carrier should be detected as 0 by the master controller. If this incorrect state persists for 60 seconds, the master should fail a self-check, then crash and restart, after which it powers on all carriers (which should succeed unless there is also a real problem with the radio hardware) and thereafter resumes normal operation. |

**Table 5.10:** Recovery from radio state desynchronization where a radio is actually on but the controller thinks it is off

| | |
|---|---|
| **Scenario** | Turning off a radio silently fails and the state in the controller (off) and the actual state of the hardware (on) is desynchronized. This is simulated by turning on a radio without informing the controller. |
| **Expected outcome** | The next time this radio is turned on by the policy, the state becomes synchronized again. Since this state does not impact QoS more than normal operation it is not considered problematic if this state persists for extended periods. Energy savings are not optimal as long as this state persists though. |

**Table 5.11:** Handling of reduced capacity in active carrier

| | |
|---|---|
| **Scenario** | While running a fixed traffic load low enough for only one carrier to be active, gradually reduce the active carrier's capacity in order to simulate environmental circumstances such as heavy rain. The capacity is reduced until the single carrier is no longer sufficient according to the policy. |
| **Expected outcome** | The master detects the reduced capacity of the active carrier, determines it to be insufficient, and powers on enough carriers to meet the policy's demands. Since we don't simulate any capacity reduction for the second carrier, it alone is sufficient for the low traffic load, hence the originally active carrier should be powered off. None of this should cause any delayed or lost frames. |

## 5.3 Policy evaluation

In order to evaluate the policies' performance, we make use of a simulator that can provide results much faster than real-time evaluations on physical hardware. This provides estimates of the energy savings and QoS impact if the policies were used to control radio links in a large microwave network. As there are no interactions between radio links that affect the policy decisions, there is no need to simulate the entire network simultaneously. Instead, each radio link is simulated independently from the others.

### 5.3.1 The simulator

We have written a program that can simulate the actions of a policy when subjected to a specific traffic pattern and measure performance in terms of energy savings, QoS impact and number of times radios are turned on or off. The number of simulated carriers is configurable, and for simplicity, the simulator assumes that the capacity provided by all carriers is identical. This is not necessarily the case in the real world and is not something the actual controller implementation assumes.

The simulator works on a second-by-second basis since this is the resolution of the (enhanced) data we have available. For each second in the traffic pattern, the current utilization is given to the policy, which uses that information, along with its internal state (such as historical utilization) to determine how much radio link capacity is desired at that time. The simulator turns on and off radios depending on policy output, taking into consideration that a radio will not provide any capacity until 8-10 seconds after it has been turned on, but consumes power for this period regardless. In our simulations, we conservatively assume the radio startup time to always be 10 seconds. The data used to feed the simulator is the artificially enhanced real-world data as described in Section 5.1.3.

The simulator records the total number of radio power state changes, where powering on two pairs of radios simultaneously counts as two changes. It also records the relative energy consumption of radios compared to the baseline of always having all carriers powered on. Lastly, the simulator records QoS impact, for which we have defined our own metric: *congestion intervals*. A congestion interval, much like the name implies, represents an interval where congestion occurred, and is measured in seconds. At the end of a simulation, the simulator outputs a list of all intervals, and their duration, for which the simulated policy and utilization pattern resulted in congestion.

In order to estimate behavior during congestion, the simulator emulates the behavior of the nodes' buffers when utilization exceeds available radio link capacity. Data that could not be transmitted in the current second is put into a buffer and sent the next time there is capacity to spare. The simplest case where buffered data from one second can be completely flushed during the next second is illustrated in Figure 5.4. We consider this to be a 2 second long congestion interval. Figure 5.5 illustrates the length of congestion intervals in some more complicated cases where the buffer is filled for multiple consecutive seconds and/or can not be entirely emptied in a single second. Due to the 1 second resolution of the simulator, shorter congestion intervals than 2 seconds (one second where the buffer is filled and one where it is emptied) can not be detected.

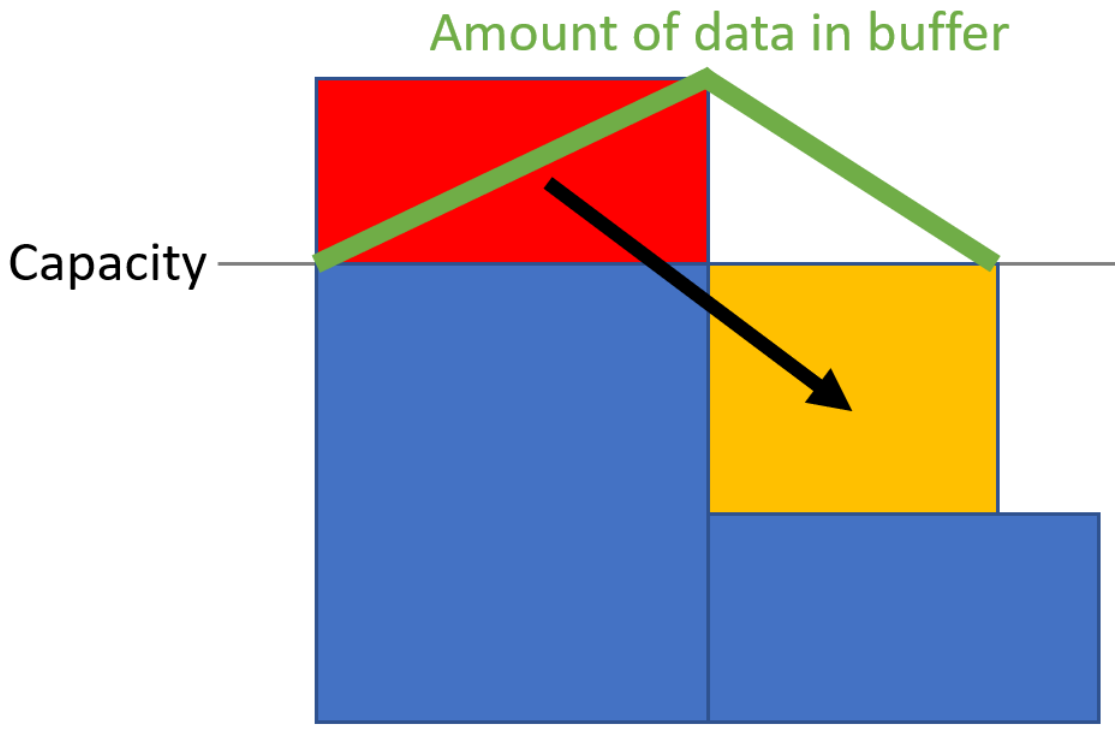


**Figure 5.4:** Buffer growth and shrinkage during congestion

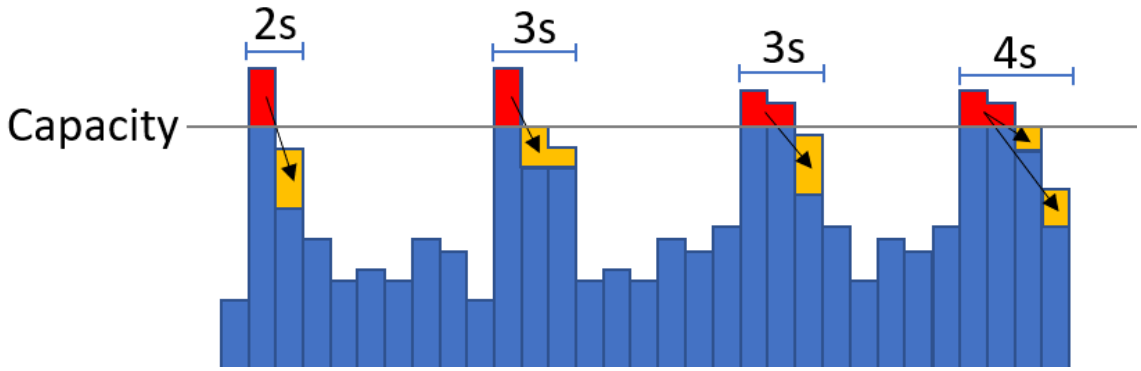


**Figure 5.5:** Congestion interval examples

### 5.3.2 Experiments

Two main sets of experiments are conducted in the simulator, where the first consists of simulations on the four cherry-picked intervals that were presented in Section 5.1.4. As these intervals contain some of the most common and challenging utilization patterns, they are used to provide a detailed picture of the policies' behaviour. In order to clearly show the controller behavior and impact of decisions, we simulate these links as if they consisted of four bonded carriers, regardless of their real-world configuration.

The second set of experiments is to evaluate the policies on the entire data set, not just on the cherry-picked selection. The results of this are presented as aggregate statistics about how the power saving scheme would perform in a large-scale deployment. For this experiment, each link is simulated according to its real-world configuration, disregarding single carrier links as they offer no potential for energy saving.

All simulation experiments are performed with three different policies: the reactive and hybrid policies as described in Chapter 4, plus a clairvoyant *Oracle* policy that can see into the future and adjust capacity for optimal power savings without ever causing congestion. Although completely unrealistic, this is useful as a point of comparison, representing the theoretical upper bound on energy savings without any impact on QoS. To make the comparison fair and prevent excessively frequent state changes by the Oracle policy, we require it to keep radios turned on for a minimum of 2 hours, just like the other two policies.

The following metrics are considered when evaluating policy performance: energy savings, QoS impact (in terms of congestion intervals), and number of power state changes made per day, as described in Section 5.3.1.

# 6

# Results

This chapter contains results of the evaluations and experiments described in Chapter 5.

## 6.1 Controller week-long evaluation results

The week-long evaluation as described in Section 5.2.2 was run for 7 days and 2 hours, with no traffic loss or unusual frame delay and a total of 26 radio power state changes. Table 6.1 shows average power measurements from the week-long evaluation and baseline values when not using our power savings scheme.

We observed an average power reduction of 27.0 W compared to when no power saving was used. Assuming power consumption of both carriers is identical, the power consumption of all radios when they are active is 69.2 W ($299.2 - 264.6 = 34.6$ W per carrier). This equates to a 39.0 % reduction of total radio power, or in other words a relative radio power consumption of 61.0 %.

| State | Master-adjacent node | Slave-adjacent node | Total |
|---|---|---|---|
| No power saving | 121.8 W | 177.4 W | 299.2 W |
| One carrier forced off | 104.6 W | 160.0 W | 264.6 W |
| Our power saving scheme | 108.2 W | 164.0 W | 272.2 W |

**Table 6.1:** Average power measurements of the physical setup for one and two active carriers and the week-long execution with our dynamic power saving scheme. The reason for the large difference in absolute power consumption is that the two nodes are different models with different base power consumption.

Figure 6.1 shows the controller behavior during the week-long evaluation, in addition to that of a simulation performed on the same interval. Some minor differences can be found between the real-world and simulated results, which were caused by the traffic generator not being able to perfectly reproduce the desired pattern. As a result, the traffic level in the simulator was overall slightly higher than in the physical setup. This is what caused radios to be turned on two times in the simulator where they were kept off in the physical setup. With this taken into consideration, the measurements and simulated results are extremely similar, which provides confidence in both the correctness of the physical implementation and the simulator.

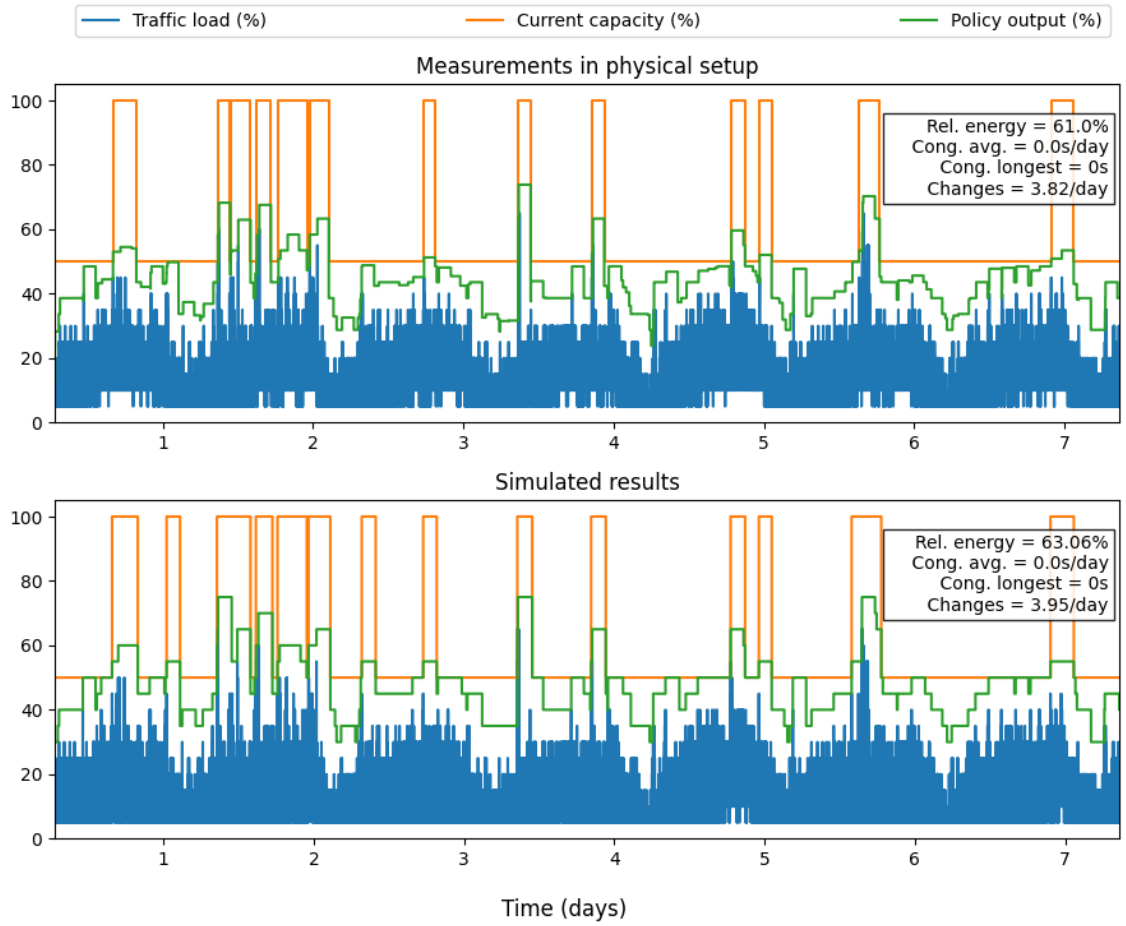


**Figure 6.1:** Measurements from the week-long evaluation and a simulated result of the same interval for comparison

The only unexpected event that was observed during the evaluation was a crash by the slave controller, after which it performed a restart. The reason was that a radio took slightly longer to start than normally, overstepping a 10 second timeout added to detect potential radio failures. Since startup time of radios is out of the controller's control, we do not consider this to be a failure of the controller. In fact, it demonstrates the resilience of the implementation to unexpected faults. After 3 seconds from the occurrence of the fault, the radio was successfully turned on and after an additional 3 seconds, the master and slave were synchronized and resumed normal execution. By simply increasing the timeout by a few seconds, this crash could have been avoided altogether. There were no other anomalies in the controller behavior and as such, we consider the week-long evaluation to have been successful.

## 6.2 Controller fault injection experiment results

All fault injection experiments described in Section 5.2.3 fulfilled the expected outcome. The controller was able to handle disturbances in the radio signal transmission, hardware failures in different components (radios, nodes or controllers), desynchro-

nized state between hardware and controller, and adapt to simulated environmental circumstances that affected the capacity of carriers. Each experiment was executed once for each relevant hardware unit. For example, the controller crash fault injection experiment was performed once for the master controller and once for the slave controller. The full details of each experiment can be found in Appendix A.

## 6.3 Simulation results on cherry-picked intervals

Simulations were performed on the cherry-picked intervals as described in Section 5.3.2, with Table 6.2 containing the resulting performance statistics. The full simulator output for each of these intervals can be found in Appendix B, which contain the same statistics, but also include a visualization of the associated traffic pattern, policy output and radio link capacity over time.

On the very regular and predictable intervals A (Figure B.1) and B (Figure B.2), the performance of the reactive and hybrid policy are very similar. Reactive achieves marginally better energy efficiency and fewer power state transitions, but at the cost of slightly more congestion. The same trade-off applies to the irregular interval D (Figure B.4), although with significantly worse absolute performance due to the unpredictable nature of the utilization pattern. However, the most significant difference in performance between the policies is observed for the extremely variable interval C (Figure B.3).

| Interval | Policy | Rel. energy | Cong. avg. | Cong. longest | Changes |
|---|---|---|---|---|---|
| A | Reactive | 56.99 % | 1.01 s/day | 2 s | 5.46/day |
| | Hybrid | 59.62 % | 0.9 s/day | 2 s | 7.25/day |
| | Oracle | 49.24 % | 0 s/day | 0 s | 4.78/day |
| B | Reactive | 32.08 % | 1.61 s/day | 3 s | 1.93/day |
| | Hybrid | 33.11 % | 0.69 s/day | 3 s | 2.2/day |
| | Oracle | 31.84 % | 0 s/day | 0 s | 2.02/day |
| C | Reactive | 88.13 % | 9.28 s/day | 14 s | 8.09/day |
| | Hybrid | 91.91 % | 0.29 s/day | 2 s | 2.38/day |
| | Oracle | 83.88 % | 0 s/day | 0 s | 11.13/day |
| D | Reactive | 36.49 % | 3.77 s/day | 5 s | 7.14/day |
| | Hybrid | 41.55 % | 2.86 s/day | 5 s | 11.54/day |
| | Oracle | 31.86 % | 0 s/day | 0 s | 5.08/day |

**Table 6.2:** Simulated performance statistics for the reactive and hybrid policy, as well as an oracle policy, on the four intervals A, B, C and D

For interval C, the hybrid seems to be the clear winner with 0.3 s/day congestion compared to 9.3 s/day for the reactive. This difference is partially due to the ability of the hybrid policy to anticipate future demand, but most of the difference is in fact due to the hybrid model having a 3 hour effective window size compared to only 2 hours for the reactive model. When increasing the window of the reactive policy to an equal size of 3 hours, the difference largely disappeared, as can be seen

in Table 6.3. While the hybrid policy offers slightly better quality of service, it does so at the cost of more state changes and higher energy consumption, in addition to increased implementation complexity. As such, for all four of these cherry-picked intervals, it does not appear that predicting the future using the simple strategy of our hybrid policy is of significant benefit.

| | Reactive (2 h) | Reactive (3 h) | Hybrid |
|---|---|---|---|
| Relative energy usage | 88.13 % | 89.98 % | 91.91 % |
| Avg. changes per day | 8.09 | 1.52 | 2.38 |
| Avg. congestion per day | 9.28 s | 0.86 s | 0.29 s |
| Longest congestion interval | 14 s | 3 s | 2 s |

**Table 6.3:** Comparison of statistics from simulations on interval C with a 3 hour window size for the reactive policy

Figure 6.2 clearly shows the reason behind the difference in performance between the reactive and the hybrid policy on interval B. Note how the hybrid policy is able to power on radios (i.e. increase the current capacity) before a sudden increase in traffic, thanks to historical utilization data from the prior week. This reduces the risk of congestion, but also causes radios be powered on despite not being needed, as can be seen in the latter half of the interval. Naturally, this results in unnecessary state changes and increased energy consumption for the hybrid policy.

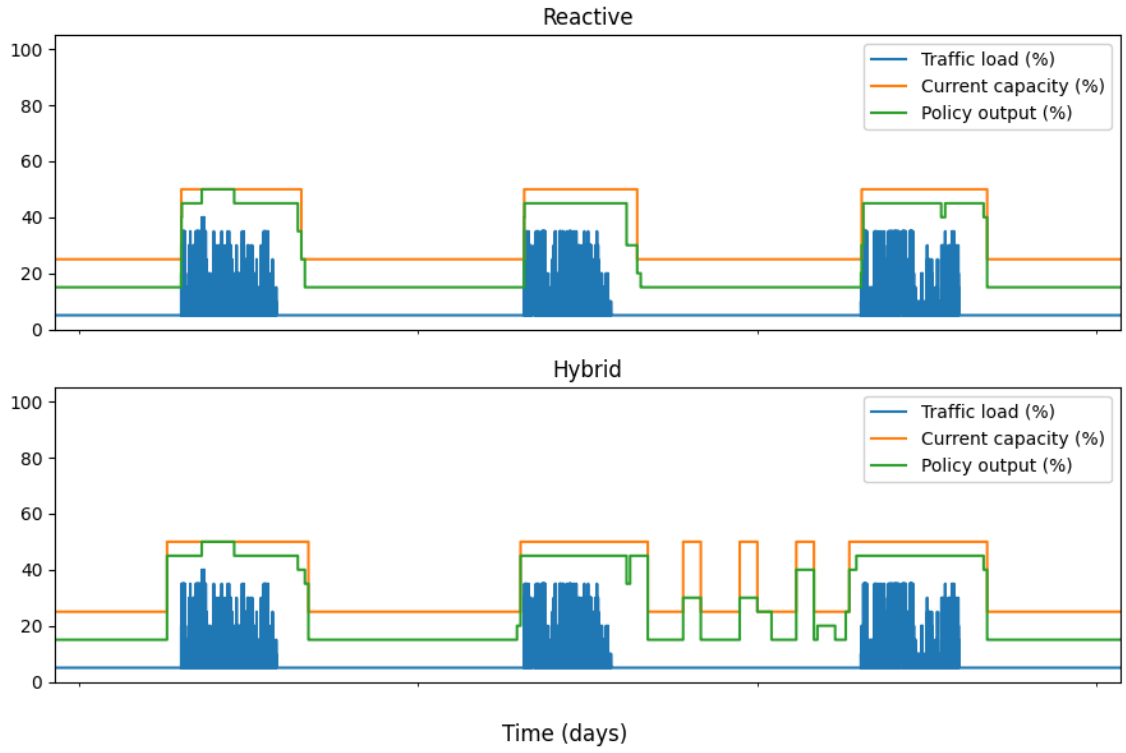


**Figure 6.2:** A close-up of the first three days of simulation on interval B, illustrating the difference in policy behavior

## 6.4 Simulation results on all links

Simulations were performed as described in Section 5.3.2 on 112 out of the 119 bonded radio links in the data set. The remaining 7 were excluded as they did not have any continuous utilization intervals longer than 8 days. Table 6.4 shows aggregated statistics from these simulations and Table 6.5 shows the worst recorded value from any link with regards to number of changes per day, congestion and energy savings.

The cherry-picked intervals B, C and D are from single carrier links in the real world and were therefore not included in this round of simulations. Additional simulations were performed where the real world configuration was disregarded and all 192 links were assumed to be either 2 or 4 bonded carriers. Results from these simulations are shown in Appendix C.

| | Reactive | Hybrid | Oracle |
|---|---|---|---|
| Relative energy usage | 50.54 % | 50.83 % | 50.12 % |
| Avg. changes per day | 0.22 | 0.39 | 0.05 |
| Avg. congestion per day | 0.033 s | 0.029 s | 0 s |
| Longest congestion interval | 3 s | 3 s | 0 s |

**Table 6.4:** Aggregate statistics of simulations when using different policies. Only data from radio links whose real world configuration has two carriers were simulated.

| | Reactive | Hybrid | Oracle |
|---|---|---|---|
| Worst relative energy usage | 58.52 % | 62.42 % | 52.07 % |
| Worst avg. changes per day | 5.12 | 8.59 | 1.81 |
| Worst avg. congestion per day | 3.23 s | 3.23 s | 0 s |

**Table 6.5:** Worst recorded simulation statistics for any link when using different policies. Only data from radio links whose real world configuration has two carriers were simulated.

Out of the 112 links, 93 never actually reached high enough utilization for any of the policies to deem a second radio necessary at any point. This indicates that overprovisioning of radio links is very widespread, which means that implementing a power saving scheme such as the one we suggest can in fact significantly reduce energy consumption in real world microwave networks. Assuming radios consume the same power as in our physical testbed (34.6 W per carrier), 50.54 % relative energy usage would equate to a yearly energy saving of 33.6 MW h for these 112 links.

Notably, neither of our two policies were able to provide power saving benefits completely without introducing congestion. This would most likely be the case with any policy. If a link should at some random point go from 0 % to 100 % in the span of a single second, only two policies would be able to prevent congestion: a clairvoyant policy (such as our Oracle policy), and a dummy policy that always maintains all carriers powered on. As such, some minor QoS impact will likely have to be tolerated no matter how good the policy.

## 6.5 Energy overhead of controller hardware

While executing our controller, the power consumption of the two Raspberry Pis in our testbed was measured to be approximately 4 watts (2 watts each). Given the energy savings presented previously, this means that it is possible to save energy with our scheme despite introducing additional hardware that consumes energy. As previously stated though, we expect continuations of this project to execute the controller on preexisting hardware, thus largely eliminating the 4 watt overhead introduced by our additional hardware and realizing the full potential of the energy savings.

# 7

# Conclusion and future work

In this project, we have proposed, implemented and evaluated a method of making microwave networks energy-aware. This method aims to reduce energy consumption by turning off power to superfluous radio units in bonded radio links during periods of low utilization. One of the main challenges in doing so is ensuring that radios are always on when their capacity is required, as insufficient capacity would lead to a reduced quality of service. This is further complicated by the startup time of radio units being close to 10 seconds, meaning they need to be turned on well in advance.

We have developed a platform for realizing the proposed energy-saving method in the form of a controller software, similar to a software-defined networking controller. This software continually monitors radio links and controls power to their radio units according to how much capacity a decision-making algorithm (policy) deems necessary. In order to ensure graceful fault handling and reliable operation, the controller is designed with a distributed system architecture, which allows for fault tolerance through redundancy. In addition, the controller is designed to be self-healing and make use of the *restart on failure* principle, allowing it to recover from a large variety of fault scenarios. The distributed architecture also allows each radio link in a network to be treated independently from all other radio links, making the controller extremely scalable.

Evaluation of the controller platform took place in a physical testbed setup with Ericsson MINI-LINK radio link equipment. In this setup, we successfully demonstrated stability through both fault injection experiments and an uninterrupted week-long evaluation with real-world traffic.

We have created two policies that can be used together with our platform: a reactive policy that only considers recent utilization, and a hybrid policy which extends the reactive by using historical data to predict future utilization. Note that the design of the controller decouples policy implementation from the controller platform, meaning any policy implementation could be used together with our controller.

For evaluation of policies, we provide a simulator that can simulate policy behavior and performance, outputting estimates of energy savings and quality of service impact for a given utilization pattern. This tool enables the development and evaluation of policies without the need for deployment on live hardware, in addition to being

much faster than any real-time evaluation.

The simulator was used to conduct large-scale performance evaluations on a data set containing four months of utilization data for 112 bonded radio links. These showed an average yearly reduction in energy consumption of 300 kW h per radio link, with a minor quality of service impact of 30 milliseconds per day on average. Our two proposed policies performed similarly, but the reactive offered slightly better energy savings and fewer power state transitions, whereas the hybrid offered slightly lower impact on quality of service.

While outside the scope of our project, we recommend any continuation to integrate the controller into the embedded software of the nodes, eliminating the need for additional hardware acting as controllers. Our proposed architecture does not assume any additional hardware to be present, and can therefore be reused in an embedded implementation. This would remove the power overhead from using dedicated hardware, further improving energy efficiency, as well as make it possible to deploy the controller in microwave networks without visiting sites to install additional hardware.

Having proved the viability of the concept, we also note the possibility of further research into more advanced policies. By employing machine learning or a time series forecasting framework such as *Prophet* [25], which can account for more complex trends than our policies, it is likely possible to improve both quality of service impact and energy savings. To this end, the platform and simulator we have developed can be used as a foundation.

# Bibliography


[1] J. Baliga, R. Ayre, K. Hinton, and R. S. Tucker, "Energy consumption in wired and wireless access networks," *IEEE Communications Magazine*, vol. 49, no. 6, pp. 70–77, Jun. 2011, ISSN: 1558-1896. DOI: 10.1109/MCOM.2011.5783987.

[2] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 3, pp. 299–316, Jun. 2000, ISSN: 1557-9999. DOI: 10.1109/92.845896.

[3] C. Gunaratne, K. Christensen, B. Nordman, and S. Suen, "Reducing the Energy Consumption of Ethernet with Adaptive Link Rate (ALR)," *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 448–461, 2008, ISSN: 0018-9340. DOI: 10.1109/TC.2007.70836.

[4] B. Addis, A. Capone, G. Carello, L. G. Gianoli, and B. Sansò, "Energy Management Through Optimized Routing and Device Powering for Greener Communication Networks," *IEEE/ACM Transactions on Networking*, vol. 22, no. 1, pp. 313–325, Feb. 2014, ISSN: 1558-2566. DOI: 10.1109/TNET.2013.2249667.

[5] *Customer Product Information for MINI-LINK 6600 1.17 (M21.Q2)*, EN/LZN 712 0501/2 R1H, Ericsson AB, 2021.

[6] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid, "A Self-Organizing Distributed and In-Band SDN Control Plane," in *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, Jun. 2017, pp. 2656–2657. DOI: 10.1109/ICDCS.2017.328.

[7] M. Canini, I. Salem, L. Schiff, E. M. Schiller, and S. Schmid, "Renaissance: A Self-Stabilizing Distributed SDN Control Plane," in *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, Jul. 2018, pp. 233–243. DOI: 10.1109/ICDCS.2018.00032.

[8] Z. Georgiou, C. Georgiou, G. Pallis, E. M. Schiller, and D. Trihinas, "A Self-stabilizing Control Plane for Fog Ecosystems," in *2020 IEEE/ACM 13th International Conference on Utility and Cloud Computing (UCC)*, Dec. 2020, pp. 13–22. DOI: 10.1109/UCC48980.2020.00021.

[9] S. Dolev, T. Petig, and E. M. Schiller, "Brief announcement: Robust and private distributed shared atomic memory in message passing networks," in *PODC*, ACM, 2015, pp. 311–313.

[10] S. Dolev, C. Georgiou, I. Marcoullis, and E. M. Schiller, "Practically-self-stabilizing virtual synchrony," *J. Comput. Syst. Sci.*, vol. 96, pp. 50–73, 2018.

[11] S. Dolev, C. Georgiou, I. Marcoullis, and E. M. Schiller, "Self-stabilizing reconfiguration," in *NETYS*, ser. Lecture Notes in Computer Science, vol. 10299, 2017, pp. 51–68.

[12] O. Lundström, M. Raynal, and E. M. Schiller, "Self-stabilizing indulgent zero-degrading binary consensus," in *ICDCN*, ACM, 2021, pp. 106–115.

[13] O. Lundström, M. Raynal, and E. M. Schiller, "Self-stabilizing multivalued consensus in asynchronous crash-prone systems," in *EDCC*, IEEE, 2021, pp. 111–118.

[14] C. Georgiou, O. Lundström, and E. M. Schiller, "Self-stabilizing snapshot objects for asynchronous failure-prone networked systems," in *NETYS*, ser. Lecture Notes in Computer Science, vol. 11704, Springer, 2019, pp. 113–130.

[15] C. Georgiou, I. Marcoullis, M. Raynal, and E. M. Schiller, "Loosely-self-stabilizing byzantine-tolerant binary consensus for signature-free message-passing systems," in *NETYS*, ser. Lecture Notes in Computer Science, vol. 12754, Springer, 2021, pp. 36–53.

[16] S. Dolev, O. Liba, and E. M. Schiller, "Self-stabilizing byzantine resilient topology discovery and message delivery - (extended abstract)," in *NETYS*, ser. Lecture Notes in Computer Science, vol. 7853, Springer, 2013, pp. 42–57.

[17] S. Dolev, C. Georgiou, I. Marcoullis, and E. M. Schiller, "Self-stabilizing byzantine tolerant replicated state machine based on failure detectors," in *CSCML*, ser. Lecture Notes in Computer Science, vol. 10879, Springer, 2018, pp. 84–100.

[18] D. Lee, B. E. Carpenter, and N. Brownlee, "Observations of UDP to TCP Ratio and Port Numbers," in *2010 Fifth International Conference on Internet Monitoring and Protection*, May 2010, pp. 99–104. DOI: 10.1109/ICIMP.2010.20.

[19] E. Blanton, V. Paxson, and M. Allman, "TCP Congestion Control," Internet Engineering Task Force, Request for Comments RFC 5681, Sep. 2009. DOI: 10.17487/RFC5681.

[20] W. Xia, Y. Wen, C. H. Foh, D. Niyato, and H. Xie, "A Survey on Software-Defined Networking," *IEEE Communications Surveys Tutorials*, vol. 17, no. 1, pp. 27–51, 2015, ISSN: 1553-877X. DOI: 10.1109/COMST.2014.2330903.

[21] D. Ghosh, R. Sharman, H. Raghav Rao, and S. Upadhyaya, "Self-healing systems — survey and synthesis," *Decision Support Systems*, vol. 42, no. 4, pp. 2164–2185, Jan. 2007, ISSN: 01679236. DOI: 10.1016/j.dss.2006.06.011.

[22] M. van Steen and A. S. Tanenbaum, *Distributed Systems*, Third edition, Version 3.01. Maarten van Steen, 2017, ISBN: 978-1-5430-5738-6.

[23] G. Candea, S. Kawamoto, Y. Fujiki, G. Friedman, and A. Fox, "Microreboot – A Technique for Cheap Recovery," 2004. DOI: 10.48550/ARXIV.CS/0406005.

[24] G. Candea and A. Fox, "Crash-Only software," in *9th Workshop on Hot Topics in Operating Systems (HotOS IX)*, Lihue, HI: USENIX Association, May 2003. [Online]. Available: https://www.usenix.org/conference/hotos-ix/crash-only-software.

[25] S. J. Taylor and B. Letham, "Forecasting at scale," PeerJ Preprints, Preprint, Sep. 2017. DOI: 10.7287/peerj.preprints.3190v2.

# Appendix A

# Controller fault injection experiment results

This appendix contains the full results from the fault injection experiments described in Section 5.2.3. For a description of each scenario and the expected outcomes, see the tables in Section 5.2.3.

## A.1 Fault handling in case of hardware failure in only active carrier

See Table 5.3 for description of the scenario and expected outcome.

1. Traffic flowing at 30 % total radio link capacity. Second carrier turned off as a result.
2. Disrupt first (active) carrier by disconnecting radio signal cable, cable between node and radio or the modem plug in card.
3. Radio link goes fully down. No traffic is getting through.
4. Controller restarts due to lost heartbeats and attempts to power on all carriers.
5. After a short while, the second carrier is up and providing capacity.
6. Duration of total frame loss and maximum frame delay for frames that were not lost are recorded from Ethernet instrument. Results are presented in Table A.1.
7. Controllers are not able to start normal execution due to the first carrier being down.
8. Restore cable/plug in card of first carrier.
9. Controller successfully powers on first carrier.
10. First carrier comes up and provides capacity.

11. Controllers complete the startup/synchronization and continues to normal operation.

12. Second carrier is turned off by the controller.

| Injected fault | Total outage | Maximum frame delay |
|---|---|---|
| Radio signal cable disconnect | 8.74 s | 133 ms |
| Radio disconnect from node (master side) | 8.57 s | 143 ms |
| Radio disconnect from node (slave side) | 9.03 s | 131 ms |
| Modem disconnect from node (master side) | 9.35 s | 157 ms |
| Modem disconnect from node (slave side) | 9.48 s | 132 ms |

**Table A.1:** Measurements on Ethernet traffic when injecting faults in the only active carrier

## A.2 Fault handling in case controller loses network connectivity

See Table 5.4 for description of the scenario and expected outcome.

1. Traffic flowing at 30 % total radio link capacity. Second carrier turned off as a result.

2. The Ethernet cable between the master controller and its corresponding node is unplugged.

3. Both controllers detect lost heartbeats and restart.

4. Both controllers attempt to power on all carriers, where the master repeatedly fails and restarts, and the slave succeeds after its first attempt. The second carrier is powered on and operational after a short period of time.

5. The slave controller maintains all carriers powered on, as the controllers do not enter energy savings without having been able to synchronize.

6. When the Ethernet cable is reconnected, the controllers synchronize after a short period of time. The controllers then apply the original energy savings state by powering off the second carrier.

7. No frame loss or unexpected frame delay was observed during the entire procedure.

The experiment was repeated for the slave controller, where the exact same behaviour was observed, but with the mentions of slave and master reversed.

## A.3 Fault handling in case the node stops responding with utilization data

See Table 5.5 for description of the scenario and expected outcome.

1. Traffic flowing at 30 % total radio link capacity. Second carrier turned off as a result.
2. Add a firewall rule that blocks the requests for current utilization from the master controller to the node.
3. The master controller crashes due to the lack of response to the blocked request.
4. Second carrier is turned on during the master startup.
5. After completing the startup, the master attempts to resume normal operation, but immediately crashes again before making any power saving decision due to the still-blocked requests. Both carriers stay active while the master is in a restart loop.
6. Remove the previously added firewall rule.
7. Master stops crashing. Normal operation is resumed and the second carrier is turned off by the controller.
8. No frame loss or unexpected frame delay was observed during the entire procedure.

## A.4 Recovery after node crash

See Table 5.6 for description of the scenario and expected outcome.

1. Traffic flowing at 30 % total radio link capacity. Second carrier turned off as a result.
2. The power cable is unplugged for the node adjacent to the master controller.
3. Both controllers detect lost heartbeats and restart.
4. Since the node (and therefore) radio link is down, the controllers cannot communicate and synchronize.
5. Reconnect the previously unplugged power cable.
6. When the crashed node has booted and the radio link is back up, the controllers successfully power on all carriers and synchronize.
7. Normal operation is resumed, and the second carrier is turned off by the

controller.

The experiment was repeated for the node adjacent to the slave controller, where the exact same behaviour was observed, but with the mentions of slave and master reversed.

## A.5 Recovery after host computer crash

See Table 5.7 for description of the scenario and expected outcome.

1. Traffic flowing at 30 % total radio link capacity. Second carrier turned off as a result.
2. Remove power cable to master Raspberry Pi.
3. Slave restarts due to loss of heartbeat.
4. Second carrier is turned on during the slave startup.
5. Slave does not enter normal operation since it can not reach the master controller.
6. Reconnect power cable to master Raspberry Pi.
7. When the master has started, the controllers synchronize and resume normal operation, turning off the second carrier.
8. No frame loss or unexpected frame delay was observed during the entire procedure.

The experiment was repeated for the slave controller, where the exact same behaviour was observed, but with the mentions of slave and master reversed.

## A.6 Recovery from partial or complete controller crash

See Table 5.8 for description of the scenario and expected outcome.

### A.6.1 Worker thread crash

1. Traffic flowing at 30 % total radio link capacity. Second carrier turned off as a result.
2. Signal the master worker thread to stop.
3. Slave restarts due to loss of heartbeat.

4. Second carrier is turned on during the slave startup.
5. A new master worker is created by the main thread after less than 30 s.
6. The controllers synchronize and resume normal operation, turning off the second carrier.
7. No frame loss or unexpected frame delay was observed during the entire procedure.

The experiment was repeated for the slave controller, where the exact same behaviour was observed, but with the mentions of slave and master reversed.

### A.6.2 Policy/heartbeat thread crash

1. Traffic flowing at 30 % total radio link capacity. Second carrier turned off as a result.
2. Signal the master's policy thread to stop.
3. Master fails a self-check and restarts.
4. Second carrier is turned on during the master startup.
5. The controllers synchronize and resume normal operation, turning off the second carrier.
6. No frame loss or unexpected frame delay was observed during the entire procedure.

The experiment was repeated for the heartbeat thread on both master and slave side. The same behavior was observed in all three cases, apart from which controller failed the self-check and restarted.

### A.6.3 Main thread crash

1. Traffic flowing at 30 % total radio link capacity. Second carrier turned off as a result.
2. Send SIGKILL to the master's main thread.
3. Slave restarts due to loss of heartbeat.
4. Second carrier is turned on during the slave startup.
5. A new main thread is created by systemd after around 30 s.
6. A new master worker is created by the main thread.

7. The controllers synchronize and resume normal operation, turning off the second carrier.

8. No frame loss or unexpected frame delay was observed during the entire procedure.

The experiment was repeated for the slave controller, where the exact same behaviour was observed, but with the mentions of slave and master reversed.

## A.7 Recovery from radio state desynchronization where a radio is actually off but the controller thinks it is on

See Table 5.9 for description of the scenario and expected outcome.

1. Traffic flowing at 66 % total radio link capacity, both carriers active.

2. The radio unit on the master side of the first carrier is manually powered off without informing the controllers.

3. Master detected the desynchronization and restarted after 75 s because a carrier that was supposed to be up was not providing any capacity for an extended period.

4. During startup, the master powers on all radios and synchronizes with the slave controller. Full capacity is restored after a short period of time.

5. The controllers enter normal operation with two carriers active, as per the traffic level.

6. Significant congestion occurred during the time at which the radio was powered off, as the traffic level exceeded available capacity. No exact measurements were recorded from the Ethernet instrument.

The experiment was repeated for the radio at the node adjacent to the slave controller, where the exact same behaviour was observed, but with the mentions of slave and master reversed.

## A.8 Recovery from radio state desynchronization where a radio is actually on but the controller thinks it is off

See Table 5.10 for description of the scenario and expected outcome.

1. Traffic flowing at 30 % total radio link capacity. Second carrier turned off as a

result.

2. The radio of the second carrier on the node adjacent to the master controller is powered on. In total 3 radios active, but still only 1 carrier established.

3. The system remains in said state until the traffic level is manually increased to 45 % at which point it activates the second carrier. (less than 50 % to avoid congestion while powering on the second carrier)

4. The actual state and the state in the controller are once again aligned.

5. When manually reducing the traffic level to below the threshold, the system returns to normal energy savings operation with everything is synchronized.

6. No frame loss or unexpected frame delay was observed during the entire procedure.

The experiment was repeated for the radio at the node adjacent to the slave controller, where the exact same behaviour was observed, but with the mentions of slave and master reversed.

## A.9 Handling of reduced capacity in active carrier

See Table 5.11 for description of the scenario and expected outcome.

1. Traffic flowing at 30 % total radio link capacity. Second carrier turned off as a result.

2. Reduced capacity of active carrier to 33 % of total link capacity.

3. Second carrier started in order to uphold the 10 % margin.

4. Once second carrier was up and provided capacity, the first carrier was taken down since the second carrier alone provided 50 % capacity.

5. No frame loss or unexpected frame delay was observed during the entire procedure.

# Appendix B

## Policy experiment results for cherry-picked links

This appendix contains the results from the simulations described in Section 5.3.2, which were performed using the cherry-picked link intervals described in Section 5.1.4. These results are analyzed in Section 6.3.

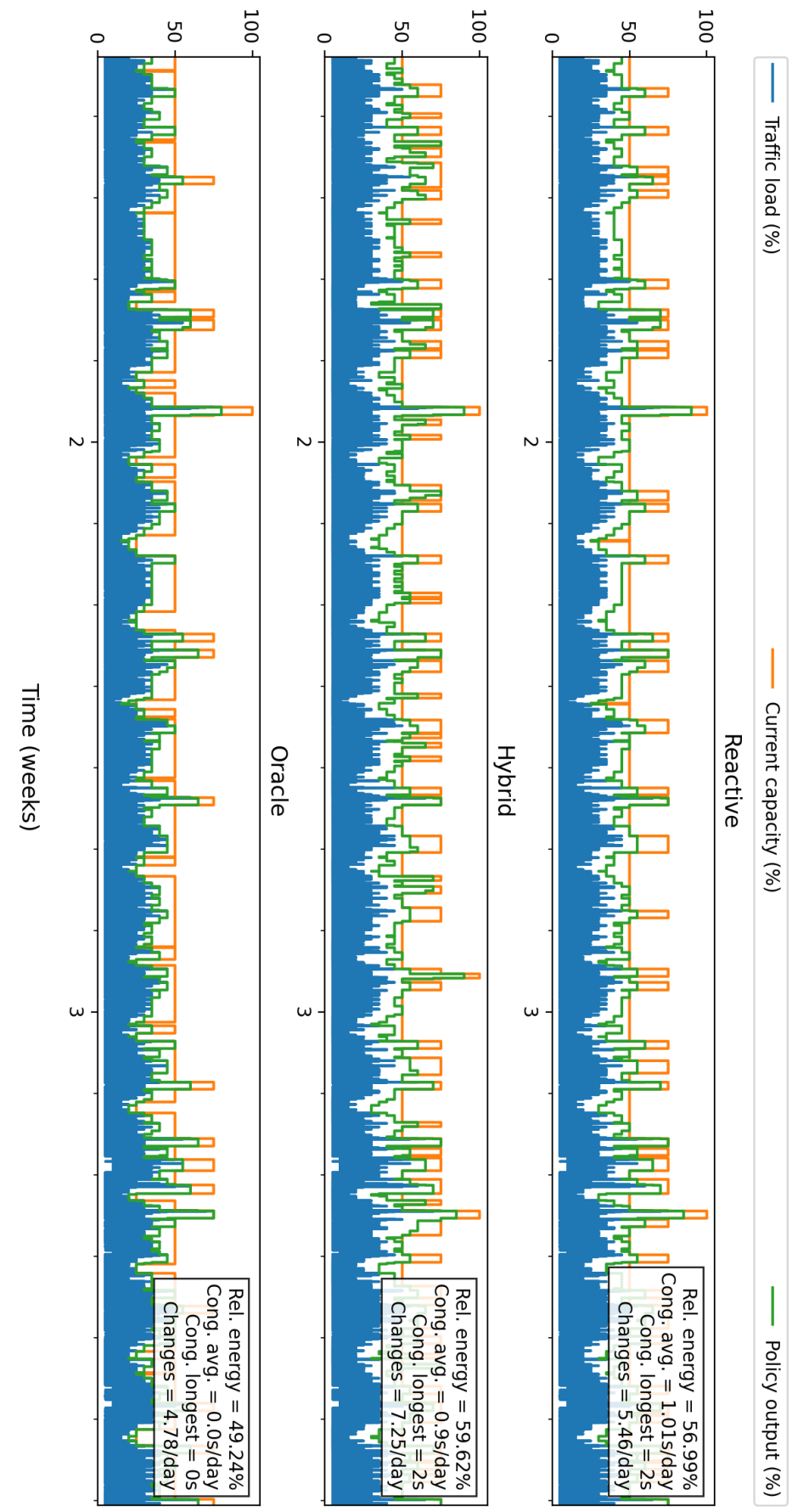


**Figure B.1:** Simulation results for interval A

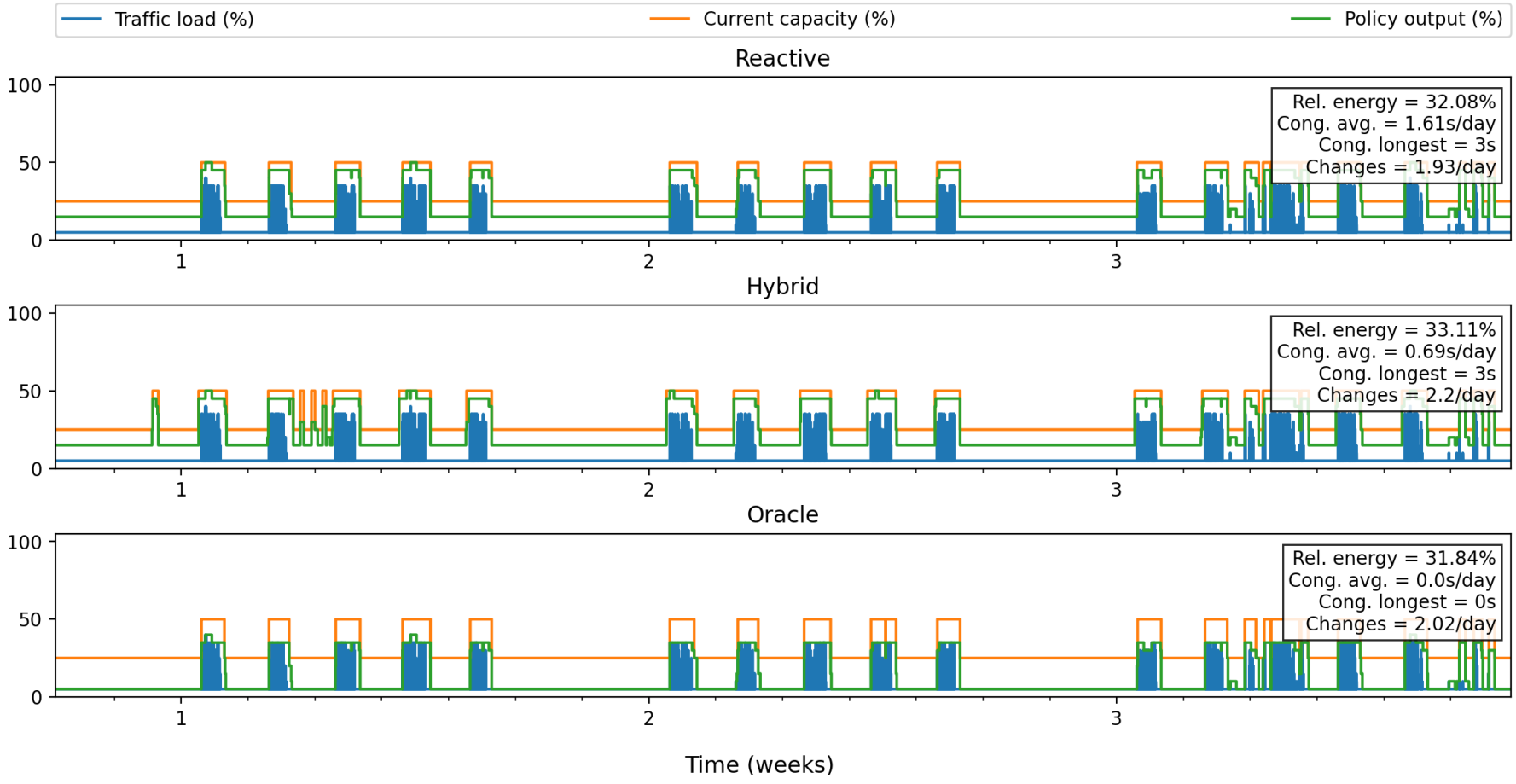


**Figure B.2:** Simulation results for interval B

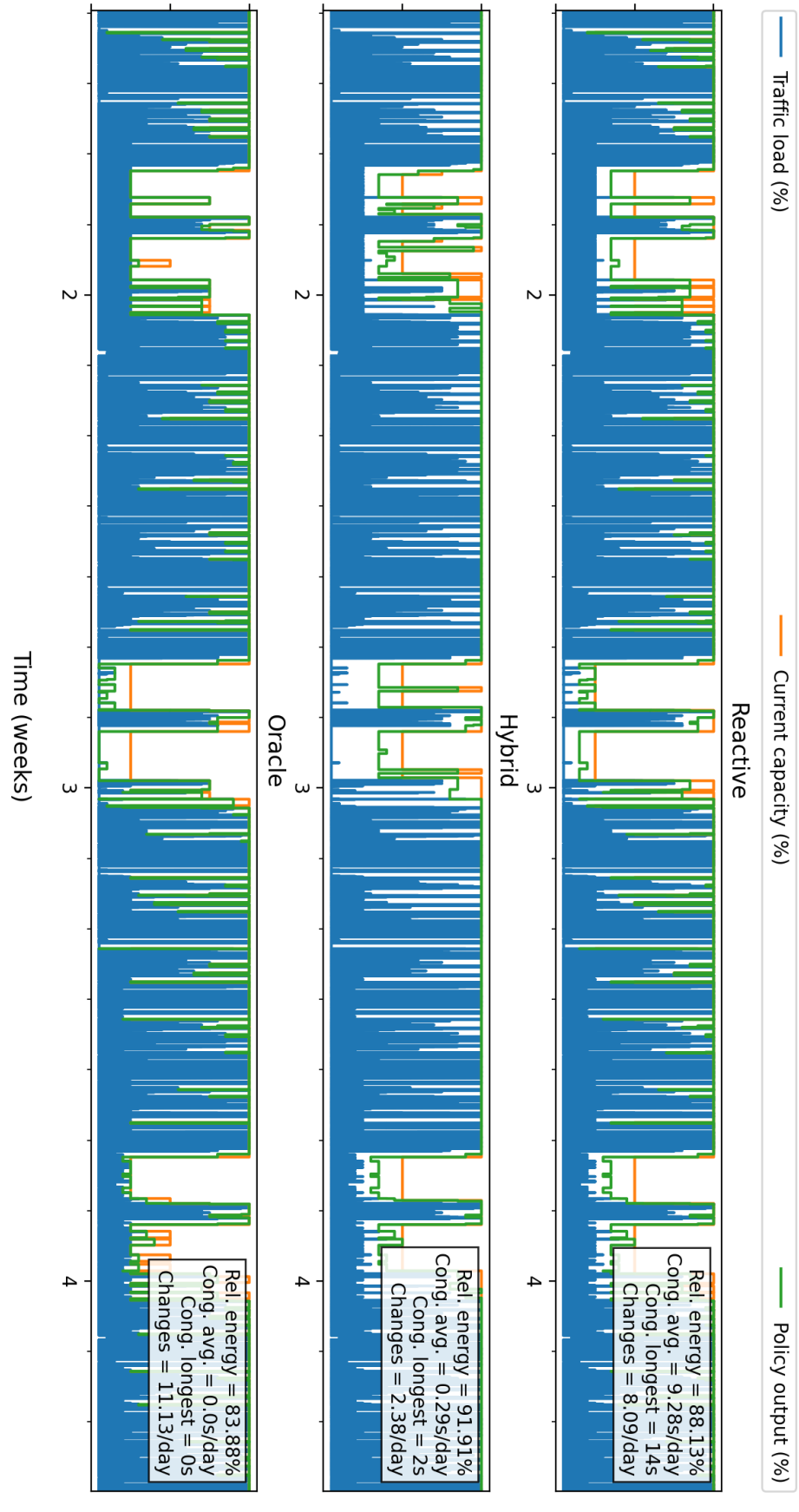


**Figure B.3:** Simulation results for interval C

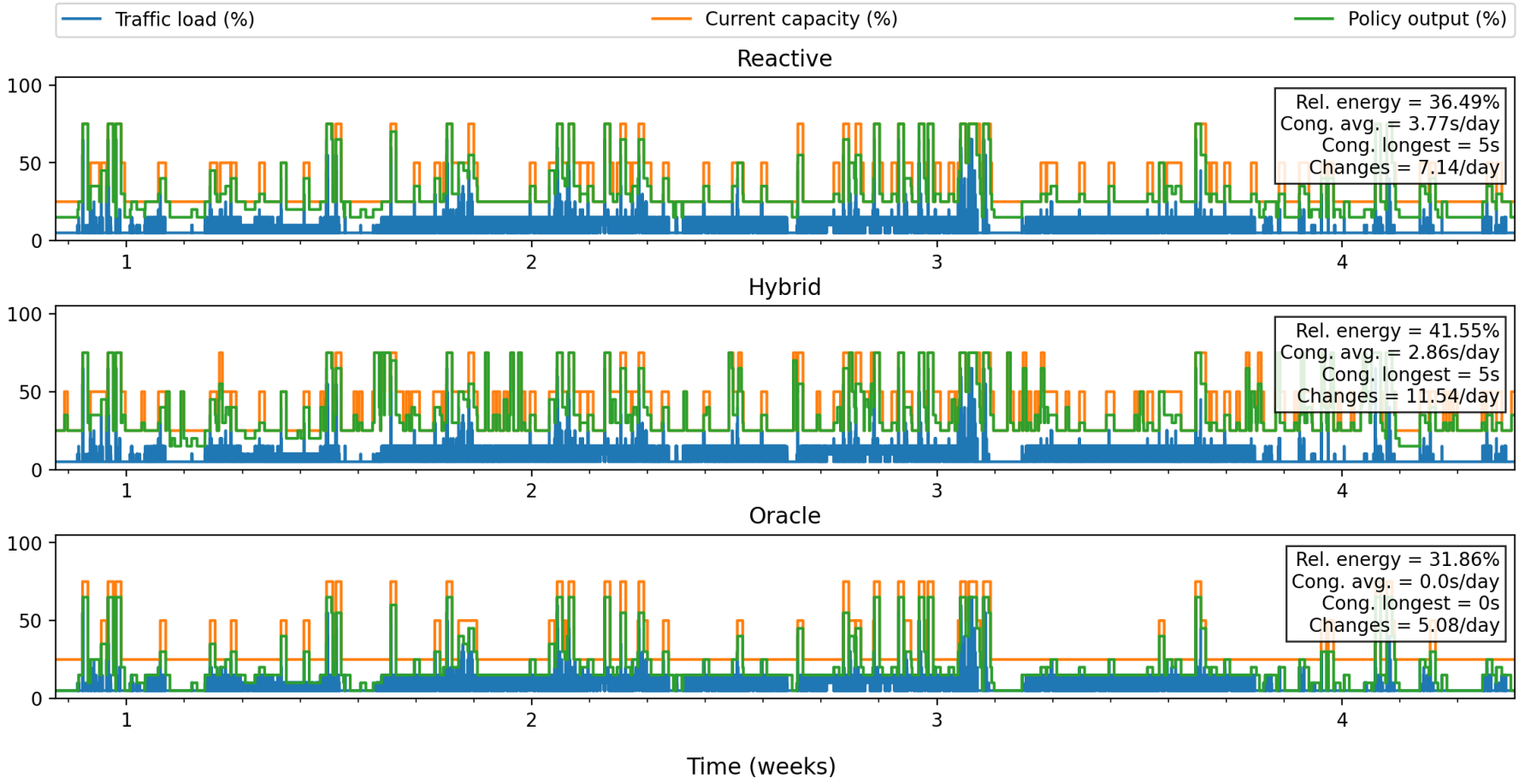


**Figure B.4:** Simulation results for interval D

# Appendix C

## Additional policy experiment results

This appendix contains complementary statistics to those shown in Section 6.4, where the actual radio link configuration was disregarded and all links were simulated as if they all consisted of either two or four bonded carriers.

| | Reactive | Hybrid | Oracle |
|---|---|---|---|
| Relative energy usage | 53.35 % | 54.08 % | 52.37 % |
| Avg. changes per day | 0.62 | 0.76 | 0.43 |
| Avg. congestion per day | 0.26 s | 0.12 s | 0 s |
| Longest congestion interval | 23 s | 23 s | 0 s |

**Table C.1:** Aggregate statistics of simulations when using different policies with the historical utilization data, considering all radio links to consist of **two** carriers

| | Reactive | Hybrid | Oracle |
|---|---|---|---|
| Relative energy usage | 33.09 % | 34.39 % | 30.20 % |
| Avg. changes per day | 2.18 | 2.58 | 1.75 |
| Avg. congestion per day | 0.91 s | 0.28 s | 0 s |
| Longest congestion interval | 23 s | 23 s | 0 s |

**Table C.2:** Aggregate statistics of simulations when using different policies with the historical utilization data, considering all radio links to consist of **four** carriers