D3.3: Final platform design and set of dissemination strategies

<table>
<thead>
<tr>
<th>Work package</th>
<th>WP 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Tasks 3.1, 3.2, 3.3, 3.4, 3.5, 3.6</td>
</tr>
<tr>
<td>Due date</td>
<td>31/07/2017</td>
</tr>
<tr>
<td>Submission date</td>
<td>27/07/2017</td>
</tr>
<tr>
<td>Deliverable lead</td>
<td>AALTO/TUM</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Authors</td>
<td>T. Kärkkäinen (TUM), R. Baig (Guifi.Net), A. Sathiaseelan (UCAM), A. Ali (UCAM), M. Isah (UCAM), F. Arnal (TAS), R. Sallantin (TAS), D. Trossen (IDC), C. Theodorou (AVANTI), L. Pajević (TUM), J. Benseny (Aalto)</td>
</tr>
<tr>
<td>Reviewers</td>
<td>Martin Potts (Martel)</td>
</tr>
</tbody>
</table>

Abstract
This document describes the final platform design and set of dissemination strategies.

Keywords
Dissemination Strategies, Fronthaul, Backhaul, Delay Tolerant Networking, Satellite, OpenSAND, Gateway, edge caching
Disclaimer
The information, documentation and figures available in this deliverable, is written by the RIFE (Architecture for an internet for everybody) – project consortium under EC grant agreement 644663 and does not necessarily reflect the views of the European Commission. The European Commission is not liable for any use that may be made of the information contained herein.

Copyright notice
© 2015 - 2018 RIFE Consortium

Acknowledgment
This report is partially funded under the EC H2020 project RIFE, grant agreement 644663.

<table>
<thead>
<tr>
<th>Project co-funded by the European Commission in the H2020 Programme</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Nature of the deliverable</strong>*:</td>
</tr>
<tr>
<td>Dissemination Level</td>
</tr>
<tr>
<td>PU</td>
</tr>
<tr>
<td>PP</td>
</tr>
<tr>
<td>RE</td>
</tr>
<tr>
<td>CO</td>
</tr>
</tbody>
</table>

*R: report, P: prototype, D: demonstrator, O: other
EXECUTIVE SUMMARY

This document describes the final design and the set of dissemination strategies for the RIFE platform. First, we describe the physical network infrastructure targeted by the project. This includes a satellite based backhaul network connection to the Internet, and a community network based on Wi-Fi point-to-point links as a fronthaul network. This is the infrastructure that will also be used for the evaluation and validation of the final system.

We then briefly recap the RIFE architecture before moving onto the details of the ICN dissemination strategies employed by the platform. In particular, we describe the use of network coding in the backhaul and the use of Bloom filters in the fronthaul. Further, we describe a DTN-based fronthaul dissemination strategy that can be used in the absence of fixed network infrastructure.

Finally, we provide details of the design and implementation work done on the system. This includes an integrated NAP that combines the well-connected ICN dissemination strategy and the disconnected DTN dissemination strategy. We further describe a pure DTN NAP variant that can be used in scenario where no fixed network infrastructure exists. We also describe the RIFE gateway design, including the edge caching.
# TABLE OF CONTENTS

**EXECUTIVE SUMMARY** ........................................................................................................................................................................... 3

**TABLE OF CONTENTS** ............................................................................................................................................................................ 4

**LIST OF FIGURES** .................................................................................................................................................................................. 6

**ABBREVIATIONS** .................................................................................................................................................................................. 9

1 **INTRODUCTION** .................................................................................................................................................................................. 10

1.1 Scope of this Document ............................................................................................................................................................................. 10

1.2 Reading this document ........................................................................................................................................................................... 13

2 **NETWORK INFRASTRUCTURE** ......................................................................................................................................................... 14

2.1 Backhaul ............................................................................................................................................................................................... 14

2.1.1 Satellite Backhaul ............................................................................................................................................................................. 14

2.1.2 Satellite Backhaul Network Subsystem interfaces ...................................................................................................................... 15

2.2 Satellite communication emulation platform ........................................................................................................................................ 16

2.2.1 Overview .......................................................................................................................................................................................... 16

2.2.2 Architecture ...................................................................................................................................................................................... 17

2.2.3 Core functionalities ......................................................................................................................................................................... 17

2.2.4 Interfaces ......................................................................................................................................................................................... 18

2.2.5 Performance .................................................................................................................................................................................... 19

2.2.6 Management .................................................................................................................................................................................. 20

2.3 Guifi.net Line-of-Sight ....................................................................................................................................................................... 21

2.4 Fronthaul WLAN Testbed ................................................................................................................................................................. 23

2.5 Implications for the RIFE systems and its dissemination strategies ........................................................................................................ 25

3 **FIELD DEPLOYMENT OF THE RIFE PLATFORM** .......................................................................................................................... 27

3.1 Hosting Community Selection .............................................................................................................................................................. 27

3.2 Field Deployment ................................................................................................................................................................................ 29

3.3 Final Architecture of the Field Deployment ..................................................................................................................................... 29

4 **ARCHITECTURAL COMPONENTS** .................................................................................................................................................. 31

4.1 Network Attachment Point ................................................................................................................................................................. 31

4.2 Border Gateway ................................................................................................................................................................................... 31

4.3 User Equipment ................................................................................................................................................................................... 31

5 **DISSEMINATION STRATEGIES** ..................................................................................................................................................... 33

5.1 Backhaul Network Coding ............................................................................................................................................................... 33

5.1.1 Fountain codes .................................................................................................................................................................................. 33

5.1.2 Raptor codes ................................................................................................................................................................................... 34
5.1.3 Possible test scenario for Network Coding over ICN ............................................................... 34
5.2 Fronthaul Bloom filter .............................................................................................................. 35
  5.2.1 Design tenets ...................................................................................................................... 35
  5.2.2 Challenges of Bloom-filtering forwarding ........................................................................ 36
  5.2.3 On the zone building ......................................................................................................... 37
  5.2.4 Operations on Zones ......................................................................................................... 38
  5.2.5 Possible implementations ................................................................................................. 40
  5.2.6 Discussion ......................................................................................................................... 43
  5.2.7 Test Case: Leveraging False Positives of Bloom Filter-Based Forwarding ..................... 44
5.3 Fronthaul DTN ......................................................................................................................... 53
  5.3.1 Dense DTN Fronthaul Dissemination ............................................................................... 53
  5.3.2 Exploiting Resource Diversity for Dense DTN Front Haul Dissemination ..................... 60
  5.3.3 Modelling DTN Fronthaul Dissemination ....................................................................... 64
6 SYSTEM COMPONENT DESIGN .................................................................................................. 68
  6.1 Integrated Network Attachment Point ................................................................................... 68
    6.1.1 Detailed Description of the Functional Blocks ............................................................... 69
    6.1.2 Interfaces ....................................................................................................................... 70
    6.1.3 Sample implementation ................................................................................................. 70
  6.2 DTN Network Attachment Point ........................................................................................... 71
    6.2.1 Native DTN Network Attachment Point ........................................................................ 71
    6.2.2 DTN Web Network Attachment Point .......................................................................... 73
  6.3 Border Gateway ..................................................................................................................... 77
    6.3.1 Elements ....................................................................................................................... 78
    6.3.2 Interfaces ....................................................................................................................... 79
    6.3.3 Sample Implementation ................................................................................................. 79
  6.4 User Equipment ...................................................................................................................... 81
    6.4.1 Native DTN Host ......................................................................................................... 81
7 CONCLUSION ............................................................................................................................ 83
REFERENCES ................................................................................................................................. 84
LIST OF FIGURES

Figure 1: RIFE network topology overview with a single fronthaul as autonomous system .................................. 10
Figure 2: RIFE network topology overview with several jointly managed fronthaul networks .................................. 11
Figure 3: Satellite backhaul ........................................................................................................................................... 15
Figure 4: Satcom system emulation ............................................................................................................................. 16
Figure 5: OpenSAND architecture ............................................................................................................................... 17
Figure 6: OpenSAND core functionalities .................................................................................................................... 18
Figure 7: Testbed topology ............................................................................................................................................ 18
Figure 8: Residential scenario ...................................................................................................................................... 19
Figure 9: Collective access scenario ............................................................................................................................ 19
Figure 10: Corporate scenario ....................................................................................................................................... 19
Figure 11: OpenSAND management architecture ...................................................................................................... 20
Figure 12: Main interface ............................................................................................................................................... 21
Figure 13: Probe interface ............................................................................................................................................. 21
Figure 14: Guifi.net fronthaul architecture .................................................................................................................. 22
Figure 15: Well-connected and disconnected fronthaul networks ............................................................................. 23
Figure 16: Testbed system model for RIFE DTN fronthaul experimentation .................................................................. 24
Figure 17: Testbed hardware setup in a radio frequency shielded room ..................................................................... 24
Figure 18: Geographical distribution of the received proposals for hosting the field deployment. ... 28
Figure 19: Field deployed supernodes, links and connections to other networks as of June 2017. .... 29
Figure 20: Field deployment configuration ................................................................................................................ 30
Figure 21: Fountain coding information items ............................................................................................................. 33
Figure 22: Backhaul network coding use case scenario ............................................................................................... 34
Figure 23: Inter-zone forwarding ................................................................................................................................ 36
Figure 24: Dynamics of the zone definition ................................................................................................................ 37
Figure 25: Zone gaining ................................................................................................................................................. 39
Figure 26: Zone splitting ................................................................................................................................................ 39
Figure 27: Joining a zone ............................................................................................................................................... 40
Figure 28: Zone draining ................................................................................................................................................. 40
Figure 29: ICN message header structure for SDN-like approach .................................................................................. 41
Figure 30: Joining of two zones in an SDN network .................................................................................................... 41
Figure 31: ICN-packet header structure for popper compliant routing ........................................................................... 42
Figure 32: Splitting of a zone with poppers .................................................................................................................. 43
Figure 33: Scenario A: Improving Network Robustness by changing false positive rate (FPR) ................................. 46
Figure 34: Scenario B: Improve user mobility by changing false positive rate ........................................ 46
Figure 35: Total nodes = 119 .................................................................................................................. 48
Figure 36: Total nodes = 112 .................................................................................................................. 48
Figure 37: Total nodes = 52 .................................................................................................................... 48
Figure 38: Total nodes = 119 .................................................................................................................. 49
Figure 39: Total nodes = 112 .................................................................................................................. 49
Figure 40: Total nodes = 52 .................................................................................................................... 49
Figure 41: True positive redundancy (total nodes = 119) ....................................................................... 50
Figure 42: True positive redundancy (total nodes = 112) ....................................................................... 50
Figure 43: True positive redundancy (total nodes = 52) ....................................................................... 50
Figure 44: Transmission accuracy (total nodes = 119) ....................................................................... 50
Figure 45: Transmission accuracy (total nodes = 112) ....................................................................... 50
Figure 46: Transmission accuracy (total nodes = 52) ....................................................................... 50
Figure 47: Node infection; total nodes = 119 ......................................................................................... 51
Figure 48: Node infection; total nodes = 112 ......................................................................................... 51
Figure 49: Node infection; total nodes = 52 ......................................................................................... 51
Figure 50: Link infection; total nodes in topology = 119 ................................................................. 51
Figure 51: Link infection; total nodes in topology = 112 ................................................................. 51
Figure 52: Link infection; total nodes in topology = 52 ................................................................. 51
Figure 53: Multicast group size = 29; BF length = 32 bits ............................................................... 52
Figure 54: Multicast group size = 59; BF length = 64 bits ............................................................... 52
Figure 55: Multicast group size = 89; BF length = 128 bits ............................................................... 52
Figure 56: DTN fronthaul ...................................................................................................................... 53
Figure 57: Synchronisation time in a static dissemination scenario. Includes measurements both in an office environment and inside a shielded room ..................................................... 55
Figure 58: Transmission counts in the static dissemination scenario .................................................... 55
Figure 59: Request stacking behaviour for the first three complete transmissions ................................ 56
Figure 60: Multi-message synchronisation time .................................................................................. 57
Figure 61: Multi-message transmission count ...................................................................................... 57
Figure 62: Spreading speed of a single message ................................................................................. 58
Figure 63: Spreading speed of 20 messages ......................................................................................... 58
Figure 64: Recursively splitting the content dissemination to multiple Wi-Fi channels ...................... 61
Figure 65: Diagram of SIR compartments with transition rates .......................................................... 66
Figure 66: Functional components of an integrated Network Attachment Point (NAP) design ........... 68
Figure 67: Native DTN NAP system model ......................................................................................... 72
Figure 68: Scampi opportunistic networking platform ................................................................. 72
Figure 69: Traditional system design for web applications on the Internet ................................. 74
Figure 70: Adapting classical Internet based web applications into the DTN based RIFE dissemination strategy ................................................................................................................ 75
Figure 71: Two implementations for the Web support in the Scampi DTN NAP ......................... 76
Figure 72: Hardware configuration for the initial DTN NAP implementation ............................. 76
Figure 73: Two examples of native Scampi ICN applications that are served by the RIFE DTN NAP.... 77
Figure 74: RIFE border gateway .................................................................................................. 77
Figure 75: Initial component and interface design of edge cache .............................................. 78
Figure 76: Flow chart of the solution realisation ........................................................................ 81
Figure 77: Android native DTN host components ...................................................................... 81
ABBREVIATIONS

AS  Autonomous System
BF  Bloom Filter
BGP Border Protocol
BGW Border Gateway
CPE Customer Premises Equipment
CoAP Constrained Applications Protocol
DHCP Dynamic Host Configuration Protocol
DNS Domain Name Service
DTN Delay-tolerant networking
ECS Edge Cache Servers
FId Forwarding Identifier
FN Forwarding Node
GES Gateway Earth Station
HTTP Hypertext Transfer Protocol
ICN Information-Centric Networking
IP Internet Protocol
LId Link Identifier
NAP Network Attachment Point
NAT Network Address Translation
RV Rendezvous
SBA Scalable Bit Array
SDN Software-defined Networking
TCP Transmission Control Protocol
TM Topology Manager
UE User Equipment
INTRODUCTION

The RIFE project aims at providing sustainable Internet access to everyone, considering both the technical and the economic aspects of affordable Internet access. The RIFE team addresses these challenges in iterative steps: We started out from the comprehensive architectural framework (based upon the Technical Annex) for which we described various components at a conceptual level in deliverable D2.2. Using this architecture (a refined version of which is documented in deliverable D2.3), we developed an initial design of the RIFE platform and its dissemination strategies. This design formed the basis of our demonstrator implemented for the period 1 review after 18 months. Additionally, the deliverable D3.1 described the initial design of the technical realisation. The present deliverable – D3.3 – documents the augmented and revised final design, based on our implementation, integration and testing experience gained since the deliverable D3.1. We note that, while our technical design is influenced by economic considerations and may be refined based upon further insights gained, those are not part of the technical design and will be addressed separately.

1.1 Scope of this Document

The RIFE architecture foresees a networking topology in which one or more fronthaul networks are connected to the core Internet via a – usually somewhat constrained – backhaul network. The architecture supports different ways of – technically and organisationally – attaching the fronthaul (stub) network to the Internet.

1. The first option is depicted in Figure 1. The backhaul network is enclosed by an IP Border Gateway towards the Internet core and a RIFE Border Gateway towards the RIFE-enabled local network, also referred to as the fronthaul network. The RIFE fronthaul network terminates in RIFE-enabled Network Attachment Points (NAPs) that support access by the user equipment (UE). Both the RIFE Border Gateway and the NAPs may offer application-level edge caching functionality as indicated exemplary by the RIFE edge cache node function attached to the RIFE Border Gateway. In this scenario, the autonomous RIFE system is logically confined to the fronthaul network, interconnected via the backhaul link to a peering provider. The fronthaul operates independently of other fronthaul networks and thus provides an effective way for localised management.

![Figure 1: RIFE network topology overview with a single fronthaul as autonomous system](image-url)
2. The second alternative is depicted in Figure 2: In this case, the RIFE autonomous system spans multiple fronthaul networks that are related (e.g., operated by the same service provider) and thus the RIFE Border Gateway moves to a common point (e.g., an IXP) on the other side of the backhaul network close to the Internet. The backhaul links are then enclosed by the RIFE Border Gateway on the Internet side and RIFE Backhaul-to-Fronthaul (B2F) routers interfacing to the fronthaul networks. Having pairs of RIFE components on either side of the backhaul allows RIFE-specific optimisations for utilising the backhaul network, e.g., similar to (but not limited to) functions that would be found for performance enhancements across wide area networks. Those functions may be implemented by means of dedicated backhaul dissemination strategies, but they may also be realised at the application layer independent of or, in addition to, those network layer mechanisms. For example, edge caching and cache push functionality may be provided also on the Internet side of the RIFE-enabled backhaul as indicated by an extra RIFE Edge Cache collocated with the RIFE Border GW.

Note that the application layer functions (such as the RIFE Edge Cache) and the network layer components such as the RIFE Border Gateway are orthogonal. That is, those components may be combined, collocated, or placed independently of one another. This deliverable will focus on the network layer functionality, but we will consider especially the RIFE Edge Caching functionality as well, especially as this offers one potential integration point for different (fronthaul) dissemination strategies.

Figure 2: RIFE network topology overview with several jointly managed fronthaul networks

RIFE has chosen a concrete networking scenario that combines a satellite link as the expensive and likely constrained backhaul, with Ethernet as well as WLAN (the latter in a mesh, ad-hoc, or opportunistic network setup) – as the fronthaul. (Mobile) end user devices connect to the NAPs via
WLAN. We present the details of the network architecture and review its implications in Section 2. This deliverable describes the involved components and the required network protocols in detail.

As shown in the figure above, the following key components make up the RIFE architecture, completed by the (existing) user equipment. These components constitute the RIFE platform.

1. **IP Border Gateway**: An IP gateway connects the backhaul network to the core Internet. It provides network layer peering to the RIFE Border Gateway, irrespective of where the RIFE Border Gateway is located.

2. **RIFE Border Gateway**: This gateway is the administrative and technical boundary of the RIFE network to the Internet. It will communicate via IP towards the Internet and via ICN towards the RIFE network.

3. **RIFE B2F Gateway**: This optional component is present when the RIFE Border Gateway is located on the Internet side of the backhaul link and if a distinct backhaul dissemination strategy is in use (distinct from the one for the fronthaul). In this case, it translates between the ICN-based backhaul and fronthaul dissemination strategies.

4. **Network Attachment Point**: This component terminates the fronthaul dissemination strategies and offer plain IP-based access to the Internet for end user equipment. As part of the latter, it may offer application layer functionality to perform this translation and to offer improved user experience.

5. **User Equipment**: The end user devices are supposed to remain unchanged. Beyond this, RIFE may offer selected application support for certain mobile device platforms in future revisions.

On the networking side, interconnecting the above components, two classes of protocols are covered referred as dissemination strategies:

- **Backhaul dissemination strategy**: we define an ICN-based dissemination strategies tailored to specifically support efficient communication across low key backhaul links. This strategy combines (1) efficient interactive communication to effectively deal with satellite-incurred latency and (2) possibly time-shifted point-to-multipoint content dissemination to exploit the broadcast characteristics of satellite networks. The backhaul dissemination strategy is optional because the backhaul link may just use vanilla IP in certain deployments.

- **Fronthaul dissemination strategy**: we define two complementary fronthaul dissemination strategies to address the diverse expected networking environments. The ICN fronthaul dissemination strategy is designed for well-connected fronthaul networks whereas the DTN fronthaul dissemination strategy supports disconnected and opportunistic fronthaul networks.

RIFE aims at keeping user equipment “as it is” thus offering Internet access based upon plain IP connectivity from the NAP towards the (mobile) user devices. However, should users decide to upgrade their devices to become RIFE-enabled, the fronthaul network would support ICN- or DTN-based access to Internet content, so user devices could directly tap into the fronthaul dissemination strategy. These dissemination strategies are described in chapter four, including their integration and mapping.

To offer IP-based application access to legacy IP devices under different backhaul and fronthaul networking constraints, RIFE foresees the deployment of application enhancements throughout the system. Such application layer functions may be found on one or more of the RIFE key components and they may operate in concert to offer the best possible user experience. The bulk of our system design work has focused on the most prominent Internet application: the world-wide web. Further applications, such as IoT, to be added into the system will be considered during the remaining part of
the project. We note that whenever end-to-end connectivity exists, RIFE supports all IP-based applications simply due to the very nature of its network architecture and protocol design. However, the IP-based applications will not be able to benefit from any of the RIFE-enabled optimisations. Furthermore, the initial version of the RIFE system won’t have application-specific security enhancements. Again, due to the very nature of the RIFE system design, it will not prevent today’s established end-to-end security functions from operating properly and reliably; but the functions will not be optimised for the somewhat constrained environment in which RIFE operates. While we touch on some of these application functions in this deliverable, their specifications will be provided in deliverable D3.4.

Chapter five of this deliverable describes the detailed design of the individual components and the logical functions they offer to implement and interwork fronthaul and backhaul dissemination strategies, support optimised content delivery for the web, and interface to plain IP user devices.

1.2 Reading this document

As described above, the deliverable D3.1 provided the initial platform design and set of dissemination strategies. This deliverable is an extended version of D3.1 which includes the work done towards the final system design. We have opted to include the content already presented in D3.1 in order to produce a fully self-contained deliverable, that does not have an external dependency to the initial deliverable.

The majority of our new work since D3.1 has gone into the dissemination strategies, which are the key component of the RIFE system design since the RIFE system architecture has remained stable. The sections of this deliverable that contain major new work since the previous deliverable are:

- 2.4: DTN fronthaul testbed.
- 3: RIFE Platform Field Deployment.
- 5.2: Fronthaul Bloomfilter
- 5.3: Fronthaul DTN
2 NETWORK INFRASTRUCTURE

This chapter addresses the basic networking infrastructure of RIFE, into which we will integrate our components and for which we will design dissemination strategies. While RIFE itself offers a generic architecture that is by no means limited to a specific network environment or topology, nor to certain link layers, we use this concrete case to demonstrate how our overall system design gets applied to a specific target setting. The target environment comprises both satellite links for the backhaul offered by a commercial service provider and a mixed wired/WLAN-based community network as a fronthaul. Moreover, we consider WLAN subnetworks that may not be permanently connected. This yields a comprehensive coverage of network characteristics in terms of capacity limits, round-trip time, error rate, and disconnectedness that subsumes most environments a RIFE system would face in the future in the wild.

Section 2.1.1 describes the architecture of the satellite backhaul whereas Section 2.2 presents the fronthaul setup. Finally, Section 2.4 briefly captures the implications for the dissemination strategies designed for these networks.

2.1 Backhaul

2.1.1 Satellite Backhaul

In the RIFE project, the satellite backhaul is defined as the RIFE network infrastructure element that acts as the medium over which multiple backhaul dissemination strategies can be deployed. The use of existing satellite networking technology is preferred because it is cost effective and efficient for the delivery of data and content in locations where terrestrial telecommunication networks are unavailable or unreliable.

In addition, satellite IP networks offer application protocol transparency and efficient multicasting capabilities, enabling the implementation of different backhaul dissemination strategies depending on the use cases and deployment scenarios. Therefore it is evident that all applications and services that are currently offered over typical IPv4/IPv6 connections can be backhauled through the satellite link seamlessly.

The satellite backhaul interfaces with the RIFE enabled network at the IP interface of the RIFE Border Gateway. The bi-directional backhaul satellite link provides access to content, data resources and information services available via the internet. The link is controlled by the satellite network hub infrastructure that implements various satellite networking technologies to ensure reliability, efficient bandwidth use and that TCP performance is kept within accepted industry standards. All communications between the RIFE IP border gateway and the traditional IP networks is implemented in Layer 3 utilising the IPv4/IPv6 internetworking protocol.

The satellite backhaul can be deployed in a point to point or in a point to multipoint configuration allowing flexible RIFE deployments and distributed backhaul network architectures to ensure optimal use of available network resources regardless of geographical constraints and terrestrial network infrastructure availability. This open architecture and flexibility allows certain novel features like time shifting and secure content placement techniques to be used to achieve unified mechanisms for efficient internet access and local content redistribution in challenging environments with constraints like unavailability of reliable and affordable internet services, thus addressing the major RIFE objectives.

The architecture and positioning of the satellite backhaul in the RIFE architecture baseline is displayed in Figure 3 below.
2.1.2 Satellite Backhaul Network Subsystem interfaces

Within the “satellite backhaul” network there are certain interfaces responsible for the communication between the various components in the IP network.

The main functionality of the backhaul network is to provide IP transit connection via terrestrial networks terminating to the satellite provider’s WAN and therefore to the Internet.

Below there is a detailed description of the interfaces within a satellite operator’s backhaul Network (see Figure 3):

A. The interface between the RIFE BGW (of the ICN enabled network) and the VSAT Terminal

This interface stands between the ICN enabled core network and the satellite backhaul network. It is the IP trunk of all the data and content traffic and is the central point where the IP protocol is converted into an ICN protocol inside the RIFE BGW component which is part of core network subsystem. The RIFE BGW and VSAT router components are interfaced using an IPv4 core protocol of standards-based internetworking method to connect to the internet through an Ethernet cable (cat5) using high speed network interface Ethernet cards of up to 1 Gigabit/s installed on both components.

B. The interface between the VSAT Terminal and VSAT Network Hub (Located inside the GES)

This is the air interface between the earth station and remote VSAT terminal. It is a bidirectional interface that carries all the RF modulated user traffic to and from the satellite spacecraft that is to be transmitted or received by the gateway earth station (GES) antenna (the interface is extended to the replicated earth station using a fibre optic line).

In addition to the above, this interface defines the forward and return path using the air interface as physical medium to transmit the users’ traffic between the satellite and the GES and vice versa.

Finally it supports Layer 3 TCP/IP networks and Layer 2 over Satellite (L2oS) networks that transport Ethernet frames over the satellite link. It uses the DVB-S2 standard for the forward channel path implementing ACM (Adaptive Coding and Modulation) technique to increase the performance during signal fading conditions, and TDMA for the return path.
C. The interface between the VSAT Network Hub and the WAN (Internet)

This interface is defined as the gateway between the backhaul network and the Internet. It is the IP trunk of all the traffic transmitted between the VSAT Network Hub (located inside the GES) and the WAN or the Internet. It also allows the communication using the IP network service with different components in the earth station like the VSAT Network hub where it provides functions such as routing, traffic shaping and network security to the backhaul subsystem with the satellite operator’s WAN and therefore the Internet. STM-1 over IP is used as a cost-effective solution to migrate to the IP packet networking, and optical fibres used as an interface for the communication between the two subsystems. The data and content transmission of the IP technologies are evaluated in terms of bandwidth utilisation, efficiency, latency, energy efficiency and many other parameters necessary for sustaining the standards of quality transmission.

2.2 Satellite communication emulation platform

The RIFE project has at its disposal an emulation platform for the two satellite scenarios (backhaul and fronthaul). The following sections describe the emulator platform, the different interfaces that can be used (IP, Ethernet and control), and the performance that can be reached for each scenario.

This platform does not preclude real experiments, but it is an essential tool in designing and validating the first version of the RIFE platform.

2.2.1 Overview

OpenSAND is a computing platform that provides an easy and flexible way to emulate satellite communication systems. The objectives of the OpenSAND Satellite network emulation testbed are:

- Provide a research and engineering tool able to validate access and network innovative functionalities
- Provide measurements points and analysis tools for performance evaluation
- Ensure interconnection with real terrestrial networks and applications for demonstration purposes

Figure 4: Satcom system emulation
2.2.2 Architecture

The OpenSAND architecture is described in the figure below:

![OpenSAND Architecture Diagram](image)

Figure 5: OpenSAND architecture

The three blue areas represent the different components of OpenSAND software:

- the Satellite Terminal (ST),
- the Satellite Emulator (SE),
- the Gateway (GW).

2.2.3 Core functionalities

OpenSAND currently emulates the below listed functions as described in Figure 6:

- Network
- QoS
- Encapsulation
- Header compression
- Radio resource control
- Physical layer
- Satellite
In addition, the latest version includes support for multi-spot and multi gateway, as well as SCPC access. The DVB-RCS2 emulation is currently on-going.

2.2.4 Interfaces

As noted in the previous section, OpenSAND enables the interfacing with any devices in IPv4 and IPv6. The network topology of OpenSAND can be configured on every testbed component (SE, GW and ST) either by the user or by the daemon. A simple topology is presented in Figure 7.

In addition, OpenSAND also support Ethernet 802.1q (VLAN) and 802.1ad (Q in Q). Different scenarios...
can be setup as shown in Figure 8 to Figure 10 below:

![Figure 8: Residential scenario](image1)

![Figure 9: Collective access scenario](image2)

![Figure 10: Corporate scenario](image3)

The remaining interface provided by OpenSAND is dedicated to QoS and resource management, allowing the configuration of SLAs of the different terminals in a dynamic way, and the ability to get information on the queues load. This interface is instantiated at both ST and GW sides.

### 2.2.5 Performance

We use the following reference hardware for performance evaluation:

- Rack DELL PowerEdge R310 Intel Xeon X3430 4 core 2.40GHz for the OpenSAND components (SAT/GW/ST)
- Switch Netgear Smart Switch ProSafe 24

For communication based on DAMA allocation, OpenSAND is able to support up to 65 Mbps
symmetrical bidirectional UDP flows. For SCPC mode, 140 Mbps can be achieved for symmetrical bidirectional communications, and up to 350 Mbps on the forward link and 100 Mbps on the return link for asymmetrical communications.

2.2.6 Management

OpenSAND provide a full set of tools to operate the overall testbed using a dedicated GUI. Operations include:

- Testbed deployment
- Configuration of the emulation scenario
- Start and stop of the platform
- Probes and log collection

The overall management architecture is described in Figure 11. In addition, Figure 12 and Figure 13 respectively show the interface to control the platform and to display the different probes provided by the testbed.

![Diagram](image)
2.3 Guifi.net Line-of-Sight

The main fronthaul network will be a community network as managed by Guifi.net. This network combines fixed lines and wireless links between nodes to make up a mesh network.
The fronthaul deployment, shown in Figure 14, is depicted in accordance with the standard guifi.net scheme, the "infrastructure mode". This mode has two types of nodes: the "core nodes" (or "supernodes"), and the "leaf nodes" (or "end-user nodes", or simply "nodes"). The supernodes are connected to each other through Point-to-Point links. The nodes are attached to the supernodes through master-client links. Supernodes must run the appropriate routing protocol (BGP or OSPF). The leaf nodes do not run any routing protocol.

The supernodes are implemented by adding a computing device meant to run the ICN functionalities in addition to the current standard guifi.net supernodes implementation – "hybrid" implementation. In this implementation, all wireless devices are bridged through Ethernet cables to a core indoor router, which does all the routing. Wireless devices are either single box devices with the antennas and electronic components integrated or, at most, two pieces, the antenna and the electronics. End user nodes are implemented using any low cost outdoor CPE (Customer-Premises Equipment).

While the above community networks represent well-connected environments, many of the remote regions – or individual nodes within – may not be well or permanently connected, e.g., due to power outages. This difference is depicted in Figure 15: The left hand side shows a well-connected front end in which the different NAPs are permanently linked to each other and to the RIFE Border Gateway. The right hand side, in contrast, features a number of individual nodes and temporarily linked NAPs (indicated via the dashed lines).
The disconnected NAPs may connect to each other and/or to the Border Gateway via predictable or unpredictable links. Predictable links would come up regularly, e.g., every three hours for a known duration, which could be orchestrated across nodes to save energy and link capacity. Unpredictable links would come up and go down without upfront notice, both in terms of which nodes they interconnect, when and for how long (and what their expected capacity could be). This might be the case for scavenging scenarios in which permanent links exist but the presence or absence of other customer traffic would dictate if, when, and for how long a link would be usable.

Links in disconnected networks may come in two flavours: instant links in space, or space-time, paths. The aforementioned links offer instant communication between the neighbouring nodes in the network, but they would not be up permanently. This is a direct physical link using, e.g., a wire or fibre, a directed optical wireless link, or a radio channel. All those constitute instant links in space.

But links may also be space-time paths in a delay-tolerant network (DTN), in which there is no direct and instant interaction via a single physical medium between the neighbouring nodes. Instead, communication may happen indirectly via other nodes that would receive, store, carry, possibly relay, and finally forward content from one node to another. Those kinds of delay-tolerant or opportunistic networks may also be predictable or not: in the former case, schedule transportation means (e.g., buses, trains, etc.) might be used for relaying data from one node to others; in the latter case, people or non-scheduled transportation means might serve as message carriers. Carriers would be dedicated embedded devices (e.g., built into vehicles, balloons, or drones) or could be smart mobile devices carried by humans. Hop-by-hop communication occurs using Bluetooth or WLAN or other types of short-range radio.

In all of the above cases, the net effect is that communication may have to be delay tolerant because the next hop for an incoming piece of data may not be available immediately. This requires storing the data until – direct or indirect – forwarding becomes possible (again). This in turn presents a challenge for the established Internet protocols that would usually operate end-to-end, so that application-specific support is necessary to support disconnected environments.

### 2.4 Fronthaul WLAN Testbed

To optimise the performance of the RIFE DTN dissemination strategy in realistic scenarios, we built an experimental testbed. The goal for our experimental setup is to replicate a real world dense wireless
network segment scenario – which is a likely deployment case for RIFE – as closely as possible, while maintaining the control and repeatability necessary for producing statistically significant results. To this end, we built a testbed of 50 embedded Linux devices connected to a single Wi-Fi access point, along with a test suite software for automatically executing the experiments and collecting the results. This creates a realistic environment that includes all the layers from the physical radio communication to the application layer, running on standard off-the-shelf hardware and software.

The results shown later in Section 5.3.1 are obtained from the experiments on this testbed.

Our system model – shown in Figure 16 – replicates a single access point wireless local area network. The main components are the wireless access point, the local service node, the client grid, and the experiment controller. The dense wireless segment is created by connecting the client devices in the grid to the access point. The local services needed to support the segment, e.g., DHCP and NTP servers, are run on the local service node connected to the access point. The experiment controller runs the experiments by, e.g., flashing the client devices with the correct operating system images, running the test cases, and collecting the results. Due to the limitation of our chosen client hardware, the experiment control signalling is done over the same wireless network as the experiment itself. This has the potential to interfere with the measurements, so our test suite is specifically designed to generate minimal control traffic while the experiment is in progress.

The physical configuration of the hardware is shown in Figure 17. The white circular device in the middle of the figure is the D-Link DWL-6600AP chosen as the access point. We evaluated a number of access point devices including Cisco WAP610N and Zyxel NWA1121-NI, and found them to be incapable of supporting 50 clients. We further found that the choice of the access point has a major impact on the overall system performance.

The testbed is designed to be physically small, so that it can be easily moved to different locations. This
is important because the wireless spectrum used – and thus external interference – can vary greatly depending on the location. We exploit the mobility of our testbed to provide measurements from both a normal office environment, with multiple sources of interference not under our control, and a radio frequency shielded room where the only source of radiation is our testbed. This allows us to study the impact of interference in the content dissemination process.

For the client hardware, glowing green in the figure, we selected embedded Linux devices. In reality the clients are likely to be smartphones, but building a testbed with them presents multiple challenges: 1) Most are closed platforms, which makes it hard to control the software to the degree necessary; 2) they include much unneeded functionality, such as the screen and cellular connectivity; 3) they are physically larger and not designed to be easily mounted in a fixed setup; and 4) the cost is 2-10 times higher than embedded devices.

The specific platform we selected is the Intel Edison system-on-chip with Intel's mini breakout board, which is similar in hardware performance to typical smartphones. It is a dual core 500 MHz x86 Intel Atom based system with 1 GB of RAM and 4 GB of on-board flash memory, and an integrated 802.11n Wi-Fi chipset operating in the 2.4 GHz and 5 GHz bands. The devices are powered by two power supply units via the breakout board’s 12V input pins with ferrite beads added to filter out high frequency noise.

For the set of experiments documented in this deliverable, the client devices run a custom operating system image based on the Intel Yocto distribution (ww05-15, Linux 3.10.7). The image is stripped down from various components not needed for our tests, and with Oracle Java 8 runtime and Scampi router added – otherwise the image is close to standard.

To run the experiments, we built extensible test suite software in Python, which is run in the experiment controller. The test suite first sets up the initial state in the clients, including clearing the previous test state, synchronising the clocks and copying over the correct configuration files. Then the test case is invoked, which implements the test specific logic. Finally the results are gathered from the clients, and the previous steps are repeated until the required number of iterations have been run.

2.5 Implications for the RIFE systems and its dissemination strategies

In the above backhaul and fronthaul networks, we face major challenges that need to be addressed properly by the RIFE system and its dissemination strategies:

- The backhaul satellite network is expensive and may exhibit severe capacity limitations, especially when this is shared with primary users who have priority. A backhaul dissemination strategy should take such limitations into account to: a) be efficient in resource utilisation and thus avoid repetition of content; b) be flexible in demand allocation and support time-shifted operation to circumvent immediate resource constraints.

- The satellite channel inherently faces high latencies that need to be taken into account by a dissemination strategy to operate properly. Moreover, receivers may face varying reception quality in terms of bit error rate and thus resulting packet losses.

- The fronthaul network may comprise nodes of different degrees of connectivity. Thus, we need fronthaul dissemination strategies that are able to cope with both well-connected and disconnected nodes as well as with nodes that may oscillate between those states.

One important observation is that the need for delay tolerance – ranging from satellite latency in the order of 500ms, to disconnection periods of hours or even days – are an essential ingredient of a ubiquitous Internet access solution. This notion also appears when it comes to offering time-shifted content transmission and efficient and cost-effective satellite-based distribution.
It is evident that such a wide range of system characteristics cannot be addressed at the network layer alone. Rather, a comprehensive system architecture is needed that embeds application knowledge at the appropriate places and supports active application interactions. This is supported by our overall system design: its individual components, the diversity of the complementary dissemination strategies, and the system integration presented in Sections 3-5.
FIELD DEPLOYMENT OF THE RIFE PLATFORM

3.1 Hosting Community Selection

The selection of the location of the RIFE field deployment was conducted through an open call within the guifi.net community. The information of the call was centralised in a blog post in the guifi.net website. The call was made on May 25 2016 and was opened until July 3 2016. The announcement was posted in the two main guifi.net mailing lists, guifi-coord and guifi-users and was presented in the yearly gathering event of guifi.net in June 2016.

During the open call period we received several questions from guifi.net participants. In order to ensure equality of opportunities, all the questions were answered publicly through the mailing lists and by updating the blog post if needed.

The call included a preliminary submission deadline (June 23 2016) aimed at those who wanted feedback on their proposals. The objective of the pre-submission mechanism was to give an equal opportunity among community members (some of whom have remarkable technology skills but are not used to write proposals), and to get an idea of the number and quality of proposals we could expect. We received 5 submissions within the pre-submission period and 7 for the final call.

Due to the number and the good quality of the proposals, the announcement of the selected proposal was delayed until July 7 2016 (initially scheduled for July 3 2016).

In order to ease the selection process by the consortium, but also because the proposals were in Catalan or Spanish, guifi.net summarised them in a report. The final selection was made after a discussion in which guifi.net answered the questions asked by other partners. The selection was made through a vote excluding guifi.net.

Table 1 summarises the proposals received; the selected one is highlighted (Tarragona). Figure 18 shows their geographical distribution over the North-East Iberian Peninsula. Table 2 is an excerpt of the summary report from the selected community.

Table 1: Summary of the received proposals for hosting the field deployment

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Environment</th>
<th>Scope</th>
<th>Population</th>
<th>Guifi connectivity</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Benasque</td>
<td>High mountain</td>
<td>6 villages</td>
<td>Few 100s</td>
<td>Isolated</td>
</tr>
<tr>
<td>2</td>
<td>Fortià</td>
<td>Plain – sea level</td>
<td>1 village</td>
<td>Few 100s</td>
<td>Connected</td>
</tr>
<tr>
<td>3</td>
<td>Goierri</td>
<td>Mountain</td>
<td>4 villages/towns</td>
<td>Some 1.000s</td>
<td>Connected</td>
</tr>
<tr>
<td>4</td>
<td>Vallecas</td>
<td>Plain – plateau</td>
<td>2 city districts</td>
<td>&gt;10.000</td>
<td>Connected</td>
</tr>
<tr>
<td>5</td>
<td>Mataró</td>
<td>Plain – sea level</td>
<td>1 city, 1 town</td>
<td>&gt;10.000</td>
<td>Connected</td>
</tr>
<tr>
<td>6</td>
<td>Tagamanent</td>
<td>Mountain – valley</td>
<td>4 towns</td>
<td>Some 1.000s</td>
<td>Connected</td>
</tr>
<tr>
<td>7</td>
<td>Tarragona</td>
<td>Plain – sea level</td>
<td>1 city, 3 towns</td>
<td>&gt;10.000</td>
<td>Connected</td>
</tr>
</tbody>
</table>
Table 2: Summary of the selected community to host the field deployment.

<table>
<thead>
<tr>
<th><strong>7 Tarragona</strong></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Orography, population, and connectivity</strong></td>
<td>Plain area next to the Mediterranean Sea. The proposal includes 1 city (the second biggest of Catalonia), 2 towns and a 1 village with more than 300,000 inhabitants in total. The telecommunications offer in the centre city and the towns is according to the standards, but there are many underserved districts and small settlements.</td>
</tr>
<tr>
<td><strong>Submitter and guifi.net presence</strong></td>
<td>Submitted by a hacklab with long experience in guifi.net and with strong social local links. The proposal includes the participation of several local organisations. The hacklab has already deployed several supernodes (~10) in the area.</td>
</tr>
<tr>
<td><strong>Commitment</strong></td>
<td>Supernodes: in existing positions and positions of the partners. End-user nodes: the proposal includes a table with the guaranteed number (17 in total) and the potential (100 in total) per supernode.</td>
</tr>
<tr>
<td><strong>Quality of the document</strong></td>
<td>High. Recommendations included. Includes links feasibility study.</td>
</tr>
<tr>
<td><strong>Local impact</strong></td>
<td>Good opportunity to boost guifi in an area with long tradition but limited coverage.</td>
</tr>
<tr>
<td><strong>RIFE suitability</strong></td>
<td>Average</td>
</tr>
<tr>
<td><strong>Distance from Barcelona</strong></td>
<td>80 km; 1 hour by train</td>
</tr>
</tbody>
</table>
3.2 Field Deployment

Figure 19 shows the field deployment status. It depicts the geoposition of the supernodes, the point-to-point links among them, and the links to the Internet (FTTH), to the satellite (VSAT), and to the rest of the guifi.net community network (guifi). It is worth to mention that two supernodes must be reallocated in the future.

There are 22 working end-user nodes and at least 13 more are expected to be installed by the end of August 2017.

![Figure 19: Field deployed supernodes, links and connections to other networks as of June 2017.](image)

3.3 Final Architecture of the Field Deployment

Figure 20 shows the generic supernode architecture jointly agreed by all interested consortium partners. The description of most significant parameters follows:

- **PtP** Point-to-Point WiFi link connecting the supernode to another supernode
- **AP** Point-to-Multipoint WiFi link connecting the supernode and end-user nodes (Access Point)
- **DHCP mgnt** Management interface (with DHCP enabled)
- **SW** Unmanaged switch
- **APUx** ICN devices, one per ICN functionality
- **PC** General purpose computing device
- **FTTH** Internet or VSAT or guifi.net link
- **10, 21, 112, etc. VLAN tags**
Figure 20: Field deployment configuration.

The implementation is done through a Router Board 30111 as core router, and PC Engines APU2C4\(^2\) as APUs.

---

1 http://www.liberouter.mobi
2 https://pcengines.ch/apu2c4.htm
4 ARCHITECTURAL COMPONENTS

The following architectural components relate to those identified in section 1.1.

4.1 Network Attachment Point

The Network Attachment Point (NAP) plays an important role in RIFE, as it enables standard IP endpoints to be capable of utilising the future Internet architecture adopted from the POINT project [POI21, POI22]. The NAP essentially attaches an IP endpoint to the RIFE network providing standard IP GW functionality such as IP assignment, Network Address Translation (NAT) and firewalling. For example, functions to assign IP addresses, remap one IP address space into another while they are in transit across the network, and monitors and controls the incoming and outgoing network traffic based on predetermined security rules. Furthermore, the NAP allows an IP endpoint to translate its IP communication into any dissemination strategy chosen to be the most suitable one based upon NAP internal decision logic, which is completely hidden from the IP endpoints. A more concrete list of envisaged dissemination strategies in RIFE can be found in Section 5.

4.2 Border Gateway

The key network component to make the RIFE proposition attainable is the design of the border gateway. In order to preserve the IP interfaces towards the User Equipment (UE) and the applications, RIFE uses a gateway approach, where the bridging between IP and ICN is performed in the NAPs, i.e., the access gateways from customers to the network, and the ICN border gateway (ICN BGW), i.e., the access from and to peering networks or operator server resources. On the other hand, the RIFE border gateway is expected to perform content caching functions to place popular contents closer to the end users and offer those contents from edge caches to users when they are next called, therefore users near edge servers can download content via a much shorter path and enjoy better quality of experience.

Therefore, the RIFE border gateway is responsible for the following functions:

- Provide interface between IP and ICN via NAP, handles all the offered abstractions supported by the IP interface (e.g., IP, HTTP, COAP), and provides standard gateway functions such as NAT, firewalling and dynamic IP address assignment wherever required, as described in Section 4.1.

- Provide edge caching function via edge cache server (ECS) to improve the system performance. The edge cache shall be capable of computing the most cost-efficient methodology to place the content over distributed edge server nodes, allocating policy-based cache resources and queuing methods to store contents (whole page or fragments) and thereafter allowing restricted access to defined user group with given security constraints.

The component and interface design of the NAP and the ECS can be found in deliverable D2.2.

4.3 User Equipment

The hosts in the RIFE network infrastructure represent the end-point devices, e.g. server PC or client devices. We expect minimum modification to standard IP-based hosts, where the user can access applications and services transparently from the RIFE network, and standard existing servers in the IP network can be used with minimum configurations and serve as RIFE network component, e.g. caching server.

The communication interfaces to and from all hosts are considered as standard IP, where all hosts including server and client devices will be operating seamlessly with the RIFE ICN core via NAP.
deployed at the edge of the ICN network.
5 DISSEMINATION STRATEGIES

In this section we describe the backhaul and front-haul dissemination strategies to be used in RIFE. Each part of the network realises a different use-case in terms of requirements. Firstly, to provide increased resilience, we propose the use of Network Coding in the backhaul so that we will not only mitigate the intermittent network access but also profit from multicasting capabilities of Satellite Networks. Secondly, to increase the scalability of RIFE, we propose a new dissemination strategy, namely Scalable Bit Array (SBA) to be used in the fronthaul. SBA will allow a convenient naming system for an unlimited number of links, while preserving the efficiency of the multicasting (i.e., in terms of group formation and forwarding operation).

5.1 Backhaul Network Coding

5.1.1 Fountain codes

Fountain codes are record-breaking sparse-graph codes for channels with erasures, such as the Internet, where files are transmitted in multiple small packets, each of which is either received without error or not received. Standard file transfer protocols simply chop up a file into K packet-sized pieces, then repeatedly transmit each packet until it is successfully received. A back-channel is required for the transmitter to find out which packets need retransmitting. In contrast, fountain codes make packets that are random functions of the whole file. The transmitter sprays packets at the receiver without any knowledge of which packets are received. Once the receiver has received any N packets, where N is just slightly greater than the original file size K, the whole file can be recovered [MacKay 2005]. By optimally setting the parameters, the overhead can be as low as 5% [Cataldi et al. 2006].

As depicted in Figure 21, a sender stores the content to be encoded and for each encoded symbol calculates how many fragments (of constant and known size) of the content will be encoded in the symbol; this is called the degree of the encoded symbol. For example, the degree of encoded symbol 2 is 3. Selecting the degree is the most crucial step in the process since it affects the efficiency of the information delivery in terms of the number of required symbols to decode the content and the decoding complexity.

The sender uniformly selects degree number of fragments and encodes them into the final symbol by XORing them; this set is called the symbol’s neighbours’ set. For instance, the neighbour set of symbol 2 includes fragments A1, A2 and A4. A receiver utilises symbols with degree 1 to partially or fully decode other symbols by XORing them with the decoded symbol. In Figure 21, the receiver utilises the encoded symbol 1, to decode fragment A1. It then XORs it with symbol 2. A2 is decoded using symbol 3, which is also XORed with symbol 2. Symbol 2 now only contains A4, which is decoded. Lost packets do not affect the decoding process as long as encoded symbols continue to be received. No feedback channel is required. Note that a sender can produce a very large number of encoded symbols and continue sending them as long as interested receivers exist. The only information that must be known by the receiver is the neighbours’ set of each symbol. As shown in the Figure 21, encoded symbols may...
follow different network paths and arrive to receivers via multiple network interfaces. In fact receivers do not know nor need to know from which path or publisher each symbol is sent.

### 5.1.2 Raptor codes

Proprietary Raptor code implementations report a network overhead of 1% [Qualcomm]. In order to make Fountain codes work in practice, one needs to ensure that they possess a fast encoder and decoder, and that the decoder is capable of recovering the original symbols from any set of output symbols whose size is close to optimal with high probability. We call such Fountain codes universal. The first class of such universal Fountain codes was invented by Luby [Luby 2001][LT-codes]. The codes in this class are called LT-codes. For many applications it is important to construct universal Fountain codes for which the average weight of an output symbol is a constant and which have fast decoding algorithms. A class of Fountain codes is called Raptor codes. The basic idea behind Raptor codes is a precoding of the input symbols prior to the application of an appropriate LT-code.

### 5.1.3 Possible test scenario for Network Coding over ICN

Figure 22 depicts a use case scenario. We intend to establish a pub/sub architecture between Edge Caches. Cache-P in this figure acts as a publisher in the first stage of the scenario, while Cache E1, Cache E2 and Cache E3 are acting as subscribers. The subscribers receive partial chunks of content via limited/disrupted/unreliable connectivity via satellite links from Cache-P because the satellite terminals SAT1, SAT2 & SAT3 receive data from the satellite in distributed as well as overlapping timeslots, and then the final subscriber S1, residing in Community 1 retrieves the network coded content by decoding these chunks of distributed data in an attempt to retrieve full content. Note that in this scenario, Community1 doesn’t have an edge cache which will make the retrieval for S1 more costly from the Cache-E1, Cache-E2 and Cache-E3.

![Figure 22: Backhaul network coding use case scenario](image-url)
5.2 Fronthaul Bloom filter

Conventional IP routing has been suffering from state explosion at the network core. It is well known that the backbone routers have to maintain routing tables with tens of thousands of entries. This not only introduces a significant amount of management overhead, but also hinders the scalability of IP-routing and IP-multicast in particular. The issue is exacerbated by the emergence of the Internet-of-Things (IoT) paradigm, where an even larger number of devices are expected to be connected over IP-enabled networks, creating even more state in the network. Furthermore, the routing operations in IP networks require advanced memory and packet switching technology in order to implement the longest-prefix matching necessary to forward packets from the sender to the receiver.

One approach to address these pinch points of IP routing is that of Bloom-filter-based source-routing\(^3\). In these schemes, the path from sender to receiver is computed by a dedicated path computation element, while each link along the path is identified with a domain-unique link identifier [Jokela et al., 2009] [Tapolcai et al., 2012]. Bloom filters (BFs) are used to encode all link identifiers along the path from the sender to the receiver(s) within a single bitfield [Jokela et al., 2009]. With that, multicast is naturally supported while the forwarding operation becomes that of a membership test in a bitfield (the Bloom filter). In other words, the forwarding decision at each switch can be realised as a simple bitwise AND/CMP operation, compared to longest prefix matching in IP routers.

However, BF forwarding comes with a significant scalability drawback. The size of the bitfield that is used for forwarding (and which holds the path information) determines the maximum number of links that can be stored on it. Thus, the size of the bitfield defines the maximum size of the network topology and the maximum size of the multicast group. Extending the bitfield size is surely possible but only within the limit of defining a larger bitfield, requiring the change of all protocol headers, which is hardly a suitable approach for flexible deployment across a range of topologies.

5.2.1 Design tenets

In order to better frame the particular design choices we make in our solution to this scalability problem, we formulate the following main goals for a suitable BF forwarding solution:

1. Any solution must support any topology and multicast group size: in contrast to existing solutions, we are seeking a solution that does not limit the topology sizes supported.

2. Any solution should allow for instantaneous multicast group formation: one aspect of BF forwarding is that multicast trees can easily be formed by uni-cast path information by simply performing bitwise OR operation on the unicast BF information into a single (now multicast) BF identifier. Any solution should preserve this capability.

3. The basic forwarding operation should be kept as simple as possible to the original AND/CMP based forwarding, i.e., an extensible BF scheme should not increase individual forwarding costs.

4. Any solution should be easily realisable over an SDN-based infrastructure in order to ensure easy deployment with such emerging network environments. Solutions that have outlined how to implement basic BF forwarding over SDN equipment. Any solution should preserve such capability as much as possible.

---

\(^3\) While work on BF forwarding does not prevent its application for inter-domain routing, most solutions have been applied to intra-domain routing only. We shall therefore limit our presentation to the intra-domain problem, too.
We therefore have developed the Scalable Bit Array (SBA) solution to address the requirements outlined above. For this, we take a partitioning-based approach in which we partition a network into several zones (i.e., sub-networks) so that each part contains a certain number of links that follow a specific criteria (Figure 23). The forwarding inside every zone happens with a standard fixed-length Bloom-filter. When a packet traverses from one zone to another, the corresponding zone-specific Bloom-filter must be swapped into the portion of the header that is used for BF matching; this swapping operation causes some processing overhead. In order to reduce this processing overhead, SBA can be accompanied with an intelligent partition algorithm which is able to exploit both network topological properties and traffic patterns, so that inter-partition traffic volume is minimised. Moreover, the resulting traffic-aware partitioning can also improve header overhead by effectively reducing the number of Bloom-filters carried in a packet. For this, in [Antikannen et al], authors formulate the SBA design problem into a balanced edge partition problem, and present *Jigsaw*, an intelligent partition algorithm that adopts the traffic-aware weighting scheme in order to minimise both header and processing overhead.

### 5.2.2 Challenges of Bloom-filtering forwarding

The scalability issues caused by the fixed Bloom filter length are well known and have been identified already in early proposals such as Free-Riding Multicast [Ratnasamy] and LIPSIN [Jokela]. In general, there are three approaches that can be found from the literature regarding increasing the scalability of Bloom filter source routing. The first, and simplest, way to achieve scalability is to disallow too large multicast trees and instead use several trees when serving content to a very large audience [Rizvi, Nikolaevskiy]. While the solution is very simple and effective, it naturally increases the amount of traffic in the network and is suitable only in fairly small networks.

The second way to improve the scalability of Bloom filter-based forwarding is to use some method to decrease the False Positive Rate (FPR). The idea is that when the FPR gets smaller, a fixed size Bloom filter can store larger multicast trees. One commonly used way to reduce FPR is to use several coexisting link identifiers for every link: when the multicast trees are constructed, the link identifiers are chosen so that the number of false positives is minimised [Jokela, Hao]. The drawback of this approach is that it increases the amount of state stored in the switches. Also, these approaches only slightly decrease the FPR and do not provide scalability for arbitrarily large networks. Furthermore, because the FPR still remains positive, these forwarding schemes must employ extra countermeasures against forwarding loops caused by false positives thus further complicating the router design [Sarella, Lee and Nakao].
The third way to solve the scalability problem is to encode the multicast trees into variable sized Bloom filters [Sarella, Tapolcai]. This basically means that, when a multicast tree grows too large (and thus causes too many false positives) the in-packet Bloom filter is dynamically extended to keep the FPR acceptable. This requires that there is an additional length field in the packet header which tells the length of the packet’s Bloom filter. These variable-length Bloom filter solutions would appear acceptable if they scale to indefinitely large networks and multicast groups.

Making the Bloom-filters variable length has, however, two major drawbacks. First, it makes the multicast group management more difficult. This is because variable length Bloom-filters cannot be combined together simply by performing bitwise OR operation on them (they are of different lengths, after all). Because of this, the topology manager becomes a bottleneck as it must be consulted whenever the multicast tree changes (i.e., a node joins or leaves the multicast group). The second problem is that variable-length Bloom-filters vastly increase the complexity of the forwarding. When the Bloom-filter lengths vary from packet to packet, the link identifiers have to be calculated based on the length of the Bloom filter. Thus, the switches have to calculate link-identifiers at line-speed. Alternatively, the switches have to store pre-calculated link-identifiers for every possible Bloom-filter length. This, on the other hand, would significantly increase the amount of state stored in the network. There are also proposals that try to combine the aforementioned approaches. For example, Multi-stage Bloom-filter (MSBF) tries to reduce the false positives by splitting the multicast tree into several variable-length stages which use different Bloom-filters [Tapolcai, Yang, Tapolcai]. The problem with this kind of solutions is, again, they tend to make group management very difficult and increase the complexity of the forwarding nodes.

Antikainen et al provide further discussion on this topic [Antikainen et al.].

### 5.2.3 On the zone building

In an ICN network, a zone is defined as a group of nodes that share a unique address space defined by any node-identification strategy (e.g., Bloom filters). As shown in Figure 23, there are two different zones joined by a special element called popper which corresponds to a forwarder node with special capabilities for routing ICN messages between two different zones (specified by different piggybacked bloom filters). In the specific example of the figure, when going from zone 1 to zone 2, the popper is
able to bring to the front of the message a different bloom filter (i.e., SBA2), from the back of the id-list.

In order to build network zones, we describe several processes that model the dynamics of the zone existence as presented in Figure 24. Originally a zone must be bootstrapped through a specification language, designating the role of every member of the network. Not only there must be a description of every forwarder node (and poppers), but there should also be one for the specification of the Topology Management and Rendezvous services. As the scale increases it might be necessary to specify more than one Topology Management and Rendezvous services.

In Figure 24, after Zone i has grown enough, or even has been congested enough, it could be split into two different zones: i+1 and i+2. Eventually zones i+1 and i+2 can be joined on the basis of opposite arguments, e.g., both zones have shrunk enough or there is less congestion in the network, thus forming a new zone i+3.

5.2.4 Operations on Zones

Bootstrapping zones. Zones should be defined, and along with them, the common message interchange nodes, also known as poppers. These nodes will act for interfacing different portions of the network. At bootstrapping, the roles of certain nodes have to be defined at the Topology Manager (TM) and at the node level. The TM has to be able to compute the FIDs and LIDs and assign them to every node, according to the defined zones.

An example of a specification of a zone definition at bootstrap time is as follows:

```
network= {
    nodes = {
        { // define node one
        },
        { // define node two
            role = [“TM”, “PP”]
        }
    }
}
```

Gaining a zone. The multi-zoning approach has to provide a simple way to add-up new zones whenever the network has to be extended. The TM must know the new nodes in order to properly incorporate the new portion of the network into the existent topology.
As shown in Figure 25, initially a zone can be gained through the registration process of a single node delegated as popper. In Figure 25.A, the TM is aware of all the nodes within the network, at some point, as shown in Figure 25.B., a delegated node within the new zone would try to join the network. The TM will, in turn, assign a popper from the existent topology in order to interface the new zone as in Figure 25.C.

**Splitting a zone.** For scalability purposes, a zone can be split into two parts. Splitting a zone can be specified both from the TM and within the forwarder nodes themselves. The TM can split a network by designating new nodes to be into different parts of the network as the traffic engineering suggests.

As shown in Figure 26, a zone may also be split after a zone overflowing. As a zone grows, the available identifiers will run out depending on the identification technique (e.g., bloom-filters). These IDs are easily wearable, e.g., using bit arrays, 1-bit per element in a network will allow less number of IDs to be available than, using \( k \) random bits defined by \( H(k) \) random functions as a regular bloom filter.

**Joining zones.** Whenever required by the Topology Manager or by a rough consensus of active nodes of joining zones, two different zones can be joined. As shown in Figure 26, the network with nodes in yellow want to join an existing zone. Special care should be taken for the reconciliation of the LIDs and FIDs since this operation may be resource consuming, both in calculating new IDs and distributing them to be consistent in the newly formed network.
**Draining zones.** As congestion in a zone may become a problem, draining points may be defined as the network grows (see Figure 28), and as an alternative solution for avoiding splitting. A zone is drained by a collaborative approach between two or more zones. There must be a designated forwarder (popup, or exchange points) within two (or more) zones that allow the traffic to diverge and therefore to have a more balanced use of the resources. The process is shown in Figure 28: First a node within zone 1 reports to the TM its ability to become a draining node (i.e., a new popup). The TM then finds the node’s counterpart in the zone 2 and finally creates a new traffic path between the two nodes.

**5.2.5 Possible implementations**

In this section we present and discuss the possibilities for implementing a high-scale zoning solution for identifying networks, and sub parts of a network. It is well known that bloom filters have a natural restriction of scalability due to the occurrence of false positives. Then in order to keep the false positives bounded to a reasonable limit, there is a need to restrict a Bloom filter’s size. This limit is imposed either by the implementation [Riihijärvi et al.] or by the technology being used, e.g., an SDN/openflow implementation.

**About the identification techniques.** Before proceeding to describe the implementation approaches to implement zoning in ICN networks, we emphasize on different techniques to identify links and nodes. Many different approaches can be considered in order to identify nodes within an ICN network. The LIPSIN approach uses classical Bloom filters to identify every direction within the network. For an m-bit long name, having k bits set to 1, with m >> k, there are m!/(m-k)! different IDs. A bit-array approach can also be implemented at the expense of having m different IDs for an m-bit long name,
i.e., much fewer number of IDs than LIPSIN. However, a bit-array technique promises to be much faster than the bloom-filter based approach.

Aggregated throughput in multicast transmissions is a fundamental metric for selecting among the different techniques. This metric can be improved by properly selecting the size of a zone, along with a different encoding technique for node identification. Previous studies have revealed that there is no single answer for encoding multiple paths in the same filter.

**About the stratification within zones.** An identifier summarises the route that an ICN message will take within a zone. Then, a single zone may be represented as:

- A bit array, with every bit identifying a different node
- A single stage bloom filter
- A LIPSIN filter, for which a threshold on false-positives should be set for zonification purposes.
- A multistage bloom-filter considering every zone as a single stage.
- A multistage bloom-filter considering n-stages per hop (with a small n, regularly 1) within every single zone.

### 5.2.5.1 Protocol-agnostic offset forwarding solution

This solution is characterised by having a contiguous view of the network at the TM. The TM keeps the complete view of the network and controls (e.g., update) the status and the identification of all nodes. Therefore, the TM should be able to keep the state of all nodes and its respective bloom-filter offset.

![ICN message header structure for SDN-like approach](image)

Figure 29: ICN message header structure for SDN-like approach

In Figure 29, we observe an ICN message structure for the SDN-like approach, it contains a series of concatenated identifiers (e.g., bloom filters) set within a predefined position (off-set) of the ICN-message header. For every operation within the network under the control of a single TM, all nodes must have information about which ID is going to be inspected within every forwarder node.

![Joining of two zones in an SDN network](image)

Figure 30: Joining of two zones in an SDN network

This solution could be well implemented in (newer) SDN switches that allow a simple specification of an (implicit) offset for every single zone. However, at the time of the writing this document, the only way of implementing this solution is via simulation or emulation.

Within every forwarder there has to be a message indicating the off-set from which the ID is going to be parsed. As the order of the algorithm is O(N) (with N the total number of forwarders, thus it
corresponds to the order of messages) the bootstrapping or updating of the network can take some time (one improvement is that the processing can be parallelised). During the bootstrapping time, one could expect a consistent state of the network, once we have enough confirmations from the participant forwarders (e.g., making a fully connected network with respect to the identification strategy). However, it may not be the case for future changes on the network. For zone joining or splitting, nodes not acknowledging the reception of new IDs can lose packets until the new ID is properly set. The process is as follows:

1. Partition the whole network into convenient zones
2. Create a zone table (T) identifying every FW node
3. For every FW registered in T do
   a. send a message with its correspondent off-set
4. On the reception of an acknowledgement from a forwarder
   a. Change the status of the node for future route calculation

By updating every node on the network, we equally distribute the load of properly calculating the ID to be inspected in every forwarder. Note also that an alternative to “3.a” is to form a minimal spanning tree to send updates to all forwarders.

5.2.5.2 Popper Solution (selected nodes involved)

This solution relies on special nodes namely poppers that allow the transfer of information between two different zones.

![Image](image.png)

Figure 31: ICN-packet header structure for popper compliant routing.

In Figure 31 we observe an ICN message header in which there is a first field carrying the usable BF for the current zone (within Zone 1 or Zone 2 as depicted in Figure 23). There are two specific events in which this field must be updated. First, when the initial IDs header is generated from the TM calculating the initial delivery route. Second, as it enters into the popper looking for the next zone. Notice that all the BFs included in the path from the publisher to the subscriber, must be in the header. Then, whenever the zone number comes into consideration at the popper, the corresponding bloom filter will be pushed to the front of the message at the usable ID field.

Figure 32 exemplifies a zone splitting when using poppers. Every designated popper is updated with new messages on the setup of the new zone. Later on, while routing regular traffic, every popper just pops into the front of the message a piggybacked ID.
5.2.5.3 Techniques for identifying nodes

The identification of the nodes within an ICN network depends on the application and the scalability of the identifiers. While using an ICN network for video multicasting demands high speed operations in forwarder nodes (i.e., for joining different multicast trees), massive updates for operating systems is a more elastic use case.

5.2.6 Discussion

Dependance on the bloom filtering technique: There are solutions such as multi-stage bloom filter or extensible bit-arrays that may well suit the structure proposed in Section 2. This means that the labelling of the nodes and links is independent of the operations described for the dynamics of the zones.

Implementation choices for Popper Node: There are two implementation choices for assigning the popper’s role to a forwarding node. Firstly, a forwarder node can hold a special code that implements the popper functionality. Secondly, a forwarding node can be uniquely identified with one LID. There is a trade-off for choosing one of these options, i.e., the complexity of implementing a different forwarding function versus using one LID as an alternative recourse for assigning the forwarding function. Given that preliminary experimentation suggests that zones are composed by a low number of LIDs, there is no noticeable impact on using one LID for identifying the popper.

1. Explicit LID: use ‘internal’ LID that points to the popper function. This internal LID will need to be included in the zone BF for every popper at the right edge of the zone. If the internal forwarding function applies the BF test, it will test positively for the internal LID and the packet is sent to the popper function where the zone BF swap can happen.
   a. Pro: no change to forwarding function in the ICN platform - almost interpret the popper function as an internal or application function.
   b. Con: needs one LID per popper in each zone - question is how many poppers one expects per zone and whether it matters given that SBA tackles scalability anyway?

2. Change BF test: a popper to a different zone is by definition an element that has no outgoing port to any other forwarder in the originating zone. Hence, a BF test for its outgoing ports would fail for the originating zone BF. However, in the default forwarding function, such failed BF test would lead to discarding the packet (since there’s no apparent receiver). This test would need to change in that the packet is only discarded if the forwarder is not a popper - otherwise, the packet is sent to the popper function where the zone BF is swapped.
   a. Pro: no explicit LID is being wasted on the popper element...
b. **Con**: change needed in forwarding element, which might be more complex. Moreover it requires an extra test on every node to differentiate poppers from non-poppers. This modification could be expected to have \( O(1) \) complexity on the code.

*Increasing the number of Topology Managers.* As the network increases in scale as well as the load, new topology managers may arise. Then there will be a need for an inter-TM protocol dealing with the different interchange points for regular routing, and managing of traffic among multiple zones. Another reason to grow multiple TM corresponds to the deployment of different portions of the network by different operators.

*Scale of the number of messages.* Based on the order of the Algorithm for joining or splitting a network, for solution 1, the total number of messages grows proportionally to the total number of nodes on the network, thus it has order of \( O(N) \). For solution 2, there will be a fixed number corresponding to the designated poppers, thus \( O(M) \) with \( M \) being the total number of popper nodes.

### 5.2.7 Test Case: Leveraging False Positives of Bloom Filter-Based Forwarding

So far, we have seen that Bloom-filter-based routing is important as it inherently supports multicasting with minimum state maintenance. False positives have been considered as one disadvantage of Bloom-filter-based routing schemes since they introduce redundant traffic that could lead to congestion in networks. In this section, however, we investigate how the redundancy introduced by false positives can be leveraged in order to make a network more robust and resilient against failures. We try to lay the ground work for a technique that could utilise redundant traffic generated by the false positives of Bloom filter-based routing to make the network more robust. We conduct extensive simulations and characterise various statistical properties of false positives and then give detailed insights on results that can be considered in order to develop an efficient mechanism of robustness based on redundancy introduced by false positives.

#### 5.2.7.1 Motivation

As we are aware that Bloom filter (BF) is a probabilistic data structure, there is a chance of *false positives* (FPs) i.e., a random element can be wrongly detected as member of a certain set. As we have seen so far that FPs are usually taken in a negative context that cause redundant traffic in the network. We, however, are of the view that this redundant traffic can be used to our benefit to make a network more robust against failures since, according to system science’s perspective, introducing redundancy into a system is one of the most effective solutions to address scalability, availability and robustness. An FP can essentially lead to an alternate path of traffic flow and if this path is constructed further and utilised in some intelligent way then this multiplicity can play an important role in making a communication reliable. The multiplicity in computer networks is a well-studied subject and it has been shown to play a positive role to improve the network reliability along with other metrics such as throughput, latency, jitter etc. Similarly, if there are some ways to leverage the redundant traffic caused by the FPs then a lossy network (or a lossy partition of a network—in our case we can think in the perspective of a ‘zone’ as described in one of the earlier sections) can be made more resilient against failures. It is also important to consider potential challenges that could arise while designing a mechanism to leverage FPs for network robustness. One of the prominent challenges is the fact that BF are based on uniform hash functions that give rise to uniform FPs. Since we can only tell probabilistically about an FP event, it is difficult to detect and then leverage (e.g., somehow re-route a redundant packet towards one of the destination nodes) an FP transmission. This probabilistic nature of BF and leveraging the associated FPs is hence a non-trivial problem which we plan to systematically define and tackle in what follows.
The rate of FPs occurring in a particular network partition (or a zone) can be well controlled. As we saw the scalability of BF is discussed by using an SBA. SBA is a variable length bit vector whose length is varied based on the size of a zone. The length of SBA is adjusted with respect to the total number of links in a network partition so that BF rate is zero in this zone. We also study the effect of change of FP rate in different network topologies for varying multicast group sizes (or subscribers), however, we encourage FPs to occur (instead of completely diminishing them and suggest controlling their rate), so that a desirable amount of redundancy is introduced in a network zone depending on its lossiness. We aim to characterise various aspects of FPs to ease the design of a scheme that can leverage FPs to introduce redundancy in a networking environment. We believe that, to the best of our knowledge, the Bloom filter’s FPs have been considered in a positive role for network robustness for the first time here. In what follows we try to study and characterise different statistical properties of FPs that may be used to design an efficient routing mechanism for network robustness.

5.2.7.2 Usage Scenarios

In this section we present two motivational scenarios wherein the false positives can be exploited for improved network robustness.

5.2.7.2.1 Scenario A: Improved Network Robustness

Consider a scenario in which Alice wants to deliver an important message to Bob. The message has to traverse a network which can be further divided into several zones (e.g., A, B, and C as shown in the Figure 33). It is known that the devices in zone C suffer from high failure rate, therefore the messages passing by are subject to high loss rate accordingly. However, both Alice and Bob expect reliable communication between them despite certain failing devices. The reliability of the message delivery can be improved by utilising multiple paths between the sender (or publisher) and the receiver (or subscriber). One extreme case will be a network-wide flooding which maximises the probability of delivery but introduces significant transmission overhead. Unicast, on the other hand, only selects one path (e.g., the shortest one) between sender and the receiver. Although the transmission overhead can be reduced, conventional unicast works poorly in this case since a successful delivery may require several attempts which inevitably increases the delivery latency. In contrast, with the false positives in BF the reliability and overhead can be well balanced. For the reliable parts of the network (i.e., zones A and B), we decrease the FP rate so that the routing behaviour is close to unicast when the message traverses through it. For the unreliable part (i.e., zone C), we deliberately increase the FP rate based on the level of reliability we aim to achieve, so that the messages can be forwarded on multiple links similar to multicast and flooding. Here it is important to mention that flooding should be well controlled and contained within the network zone C. The amount of flooding can be controlled by adjusting the FP rate (e.g., by altering the BF length) or it can be contained by an approach similar to the one taken while designing SBA in which a device (like a popper switch) is present at the junction of two network zones that can act as a valve to contain the excess traffic from flowing from one network zone to another. Such a device can also be used to adjust the FP rate when a packet traverses from one network partition to another. Eventually, the network can be made more resilient to node and link failures by exploiting the FPs to provide path redundancy in unreliable parts of the network.
5.2.7.2.2 Scenario B: Improved User Mobility

Since networks were originally designed for point-to-point communication and users were assumed to remain stationary most of the time, user mobility remains as an everlasting research topic in data communication and networking. However, nowadays large-scale content distribution and widely used mobile devices invalidate such assumptions. Even though there are novel proposals such as Information-Centric Networking (ICN) to shift the conventional communication paradigm from point-to-point to content-centric, mobility (especially sender mobility) still remains one of the largest challenges confronting every architectural proposal.

As ICN and similar architectures already solved receiver mobility, we focus on sender mobility in the following. In this scenario, Alice is the content provider and is moving around within zone A as the Figure 34 shows. Bob is the content subscriber who (periodically) sends requests to Alice to retrieve specific content of his interest. The challenge is that since Alice is moving around, Bob is not certain about where to send the request. Although there are multiple solutions to solve this issue (such as through caching in ICN architectures, each has its own pros and cons regarding handover delay, communication latency, scalability etc. In our current scenario, the network zone where there is a sender would have greater redundancy to increase the probability of request messages being received by the sender. The introduced redundancy will be measured by the number of FPs that further depends on the length of a BF.
In summary, the two motivational scenarios described above have one thing in common i.e., both would benefit from the introduction of controlled redundancy in zones of a network. In what follows, we will first describe our simulation setup and then we present the results related to different characteristics of FPs that can help characterise the FP events given a specific BF length and multicast group size. These metrics can be used to devise a mechanism where the redundancy introduced by FPs can be controlled according to the network requirements (e.g., loss rate in a network zone etc). As a result, the gain in robustness of a network can then be defined as the increase in the delivery probability of a transmission (in the face of loss or sender mobility) by adjusting the redundancy (i.e., FP rate) in a network.

5.2.7.3 Simulation Setup

We adopt a graph theoretic approach to conduct our experiments. For each experiment, we use a realistic topology in graph modelling language (GML) format. We use networkx library\(^4\) in Python to perform different operations on the topology graphs. We have written a BF-based forwarding function in Python that takes a topology, a source node (or a publisher), a set of destination nodes (or subscribers) and BF length as an input. First, the input topology is converted into a directed graph and then each link in the graph is assigned a random ID called link identifier (LID). These LIDs are bit vectors of a specified length. These bit vectors are computed using multiple hash functions (or using the same hash function multiple times). In our experiments, we use MurmurHash3 (mmh3) hash function multiple times. Let us assume a bit vector of length \(m\) bits all initialised to zeros. A hash function is used \(k\) times with random seed values to set the \(k\) bits of the \(m\)-bit long bit vector. This process is used to encode membership information of an element that belongs to a set which contains a total of \(n\) elements. After that BF for a forwarding path is computed by performing OR operation among the LIDs of links that constitute a forwarding distribution tree (path) from a publisher (or a caching node) to the subscriber(s).

The computed BF is then used to check the membership of a random LID. First, AND operation is performed between any given LID and the BF. Then, this is followed by a compare (CMP) function between the result and the given LID and if, as a result, we get in return the same LID then the set membership test for this LID is passed. There is also a chance of FP event during this process. It means that few LIDs that were not originally part of a set (or the distribution tree) can pass the membership test for this set and result in a faulty forwarding decision causing redundant traffic in the network. The probability of FPs in a BF of length \(m\) bits in which \(n\) elements of a set are encoded and \(k\) number of hash functions are used is given as:

\[
p = \left(1 - e^{-\frac{k(n+0.5)}{(m-1)}}\right)^k
\]

In our simulations, we specify a random source node (a publisher) and a multicast group (subscribers’ set) whose sizes we vary as 25%, 50% and 75% of the total nodes in a topology. We collect all the LIDs of the shortest paths (that we predetermine using networkx) among the source and all the destination nodes in a multicast group to computer a BF. We then perform BF-based forwarding from the source to the destination nodes of a multicast group while keeping the track of FP events along the way. The approach we adopt for this BF-based forwarding is the same as taken by the LIPSIN forwarding. The results of our simulations that we discuss in detail in the next sections are average trends of five simulation runs per experiment. It can be noted in the upcoming results that only the average

\(^4\) https://networkx.github.io/
behaviour is shown due to the small variance observed in the experimental results.

5.2.7.4 Characterising the False Positive Events

In this section we will try to extensively analyse and characterise the FP events over a diverse set of topologies while tweaking the parameters of multicast group size and BF length. We analyse how much redundancy is produced in a topology due to FPs that can potentially be used to make a network more resilient and robust against losses and failures. We will also analyse what percentage of an active multicast tree takes part in FP-based forwarding and how many nodes/links get infected due to these FP-based transmissions. These insights can further pave the path to develop an efficient mechanism of leveraging FP events for network robustness.

5.2.7.4.1 Distinct False Positive Events

First we analyse how many distinct FP events occur in a topology while changing the parameters of multicast group size and BF length. By distinct we mean the number of unique nodes that receive a transmission due to a FP-based forwarding. We term such a transmission as unintended transmission.

We perform each experiment on a topology with three different multicast group sizes. The three group sizes are set by taking 25%, 50%, and 75% of total nodes in a topology. The nodes in a topology are given numeric IDs starting from 1. So if a topology has 119 nodes then the node IDs will be 1, 2, 3,...,119. The nodes that are part of each multicast group size are selected sequentially based on node IDs. As an example for a topology with 119 nodes the multicast group of size 25% of total nodes will contain 29 nodes with IDs 1, 2, 3,...,29 and so on for other multicast group sizes. For each given multicast group size, we vary the length of BF shown in bits along the x-axis in the graphs.

It can be observed from the Figure 35 to Figure 37 that as the total number of nodes in a topology decreases so does the total number of distinct FP events. For each topology, on the other hand, we observe that the FP events are highest for the smallest multicast group size and decrease with increasing multicast group size. This observation seems counter intuitive at first. Since one expects a smaller number of FP events when a BF contains less elements (i.e., the smallest of all the multicast group sizes considered). However, since we are assuming distinct FP events, it is more likely that the number of unique nodes in a network topology that receives an unintended transmission is maximum when a relatively smaller number of nodes comprise a multicast tree. When our multicast group grows then it is less likely that a node is outside our legitimate multicast tree and hence we observe a decrease in FP events. The case when our multicast group is larger, it is more likely that a packet forwarded due to a FP is received by a node that is already in our multicast delivery tree and hence such a node will not signal a FP event. However, the effect of packing more elements in a BF and expecting more FPs can be observed in the gradual slow decrease in FP events for higher multicast group sizes. In smaller multicast groups the BF size quickly dominates and the FP events diminish while these persist longer for bigger multicast groups. It can be seen that the larger a topology is the more FP events are going to occur. Hence it can be concluded from the analysis of these figures that there is an opportunity in the face of redundant transmissions in the form of FP events that can potentially be
leveraged to introduce redundancy and robustness in a network, however, care must be taken in choosing right parameters (multicast group size and BF length) so that a network is not overly congested by the redundant transmissions.

5.2.7.4.2 Redundant False Positive Events

In the last subsection we analysed distinct nodes (neglecting their frequency) that receive an unintended transmission due to FPs. Now we analyse the total amount of redundancy introduced in a network due to FP transmissions. We will now also take into account a node receiving multiple FP transmissions. Next graphs are an analysis on the redundancy introduced in the topologies by FPs. On the x-axis the length of BF is varied. More precisely, against each BF length three different multicast group sizes are considered, and against each multicast group size two values are compared.

The blue (or bottom) bars represent the distinct FP events (not taking their frequency into consideration). The orange (or top) bars represent the total number of FP events, i.e., a node might have received multiple unintended transmissions. The analysis of orange bars (i.e., FP redundancy) shows that for each given length of BF the FPs peak when we consider multicast group size as half of the total number of nodes. In most cases the group size smaller than or greater than 50% of the total nodes shows relatively smaller number of FP events. The reason of this phenomenon is due to the fact that for smaller group size we have small number of LIDs to be stored in a BF—hence less number of FPs—while when we have the largest group size (i.e., 75% of the total nodes) then most of the FP events are not taken into account as multiple packets are forwarded to the legitimate nodes that comprise a forwarding tree (we will analyse this type of redundancy in the upcoming subsections).

It can be observed from the Figure 38 to Figure 40 that the total redundancy introduced by FPs is much larger than the values observed in the last subsection. This can be utilised in our benefit if we come up with a mechanism where the overall network is not overwhelmed by these redundant transmissions that can be decided based on the current level of congestion in a network. In conclusion, the results of this and the last subsection can be used to devise a heuristic, by analysing the traffic traces in a network combined with level of congestion and given multicast group size, that can potentially compute a BF length for the required (or optimal) redundancy given the desired gain in network robustness.

5.2.7.4.3 True positive redundancy

The total true positive (TP) redundancy is analysed in this section. TP redundancy implies that the legitimate transmissions (including self-loops) are considered with three different multicast group sizes and varying BF length. It is important to note that in our case the TP redundancy is actually caused by FP events. The FP transmissions that end up on nodes that are part of an active multicast group actually give to rise to TP redundancy.

It can be observed from the Figure 41 to Figure 43 that there is a significant TP redundancy that decreases with the decrease in the total number of nodes in a topology and for smaller multicast group sizes. The decrease in this redundancy also happens with increasing BF length up to a certain point after which this redundancy completely vanishes. Hence, it can be concluded that if we are to leverage
the TP redundancy then a BF length within a specific window should be set. The size of this window will be determined by considering the level of redundancy we want to introduce in a network given the existing state of congestion in this network.

5.2.7.4.4 Transmission Accuracy

We define the transmission accuracy as follows:

\[
\text{Accuracy} = \frac{TPs}{TPs + FPs} \times 100
\]

Where TPs represent all the true positive events occurred (as explained in the previous subsection) and FPs represents all the false positive transmissions. Figure 44 to Figure 46 show the transmission accuracy for three different topologies.

The figures show that the transmission accuracy remains high (greater than 70%) for the three topologies and for all multicast group sizes. This is an encouraging insight. This shows that with all the redundancy introduced due to FPs in a network, the major portion constitute the redundant transmissions that are actually forwarded to the nodes that comprise a multicast tree. For each topology the accuracy is maximum when we consider the largest multicast group size. This is because more nodes, out of the whole network, are part of an active multicast tree. It can also be observed that the smaller the total number of nodes in a topology the sooner BF length dominates and makes the transmission accuracy 100%. As an example, in Figure 44 (total nodes = 119) and Figure 45 (total nodes = 112) the accuracy becomes 100% when BF length reaches 512 bits. Whereas in Figure 46 the 100% accuracy is achieved earlier at BF length of 256 bits. It can be concluded that while designing a mechanism based on BF length variation we can use a smaller BF length while traversing from a network partition with lesser nodes for the same transmission accuracy.

5.2.7.4.5 Percentage of Infected Nodes and Links

In this section we analyse the nodes and the associated links that take part in forwarding a packet due to a FP. These are the legitimate nodes i.e., they are part of an active multicast tree. We call these
nodes as *infected nodes* and the links associated with these nodes over which the FP transmission is forwarded as *infected links*. Figure 47 to Figure 49 show the percentage of nodes that are infected (i.e., they are responsible for a FP forwarding). It can be seen from these figures that these nodes comprise only a small percentage of the overall nodes in an active multicast tree. This is encouraging since it implies that lesser management overhead would be required to devise a mechanism where these nodes can be leveraged efficiently.

Similarly, Figure 50 to Figure 52 show the percentage of infected links out of total links of the infected nodes. These figures show that less dense a network is the more infected it gets from FPs. Whereas for each network the most infected links are observed when we have the smallest multicast group size. The reason for this is same as discussed in earlier subsections which is that there are lesser nodes (and associated links) that are part of an active multicast tree and hence it is more likely that a link over which a FP is forwarded is infected (i.e., not part of an active multicast tree).

**5.2.7.4.6 Degree Distribution of Infected Nodes**

In this subsection, we analyse the statistical properties of the infected links (out of the total links) of the infected nodes. We analyse what is the actual degree distribution of the infected nodes and how the degree distribution changes when we only consider those links of such nodes that are infected (i.e., a FP transmission passes through them).
We take a topology with 119 nodes as an example and consider three different cases, corresponding to three different multicast group sizes. First, we consider the smallest multicast group size (i.e., containing 25% of the total nodes) for a BF length of 32 bits as shown in Figure 53. It can be observed that nodes with degrees more than 5 are responsible for the FP forwarding. However, the infected degree of these nodes (i.e., the links associated with these nodes over which the FP transmission is actually forwarding) is quite small. This also paints an encouraging picture where we worry less about the FP infection spreading to a wider region of the network. The Log Normal distribution best fits (according to mean square error). This distribution is as follows:

\[
F(x|\mu, \sigma) = \frac{1}{\sigma \sqrt{2\pi}} \int_{0}^{x} \frac{e^{\left(-\frac{(\ln(t)-\mu)^2}{2\sigma^2}\right)}}{t} dt
\]

The same observations are true for the higher multicast group sizes. Again, we observe that nodes with degrees of around 5 and more are responsible for the FP forwarding while a smaller number of links are actually responsible to spread the FP infection. We fit Generalized Pareto (again according to mean square error) distribution which is given as follows:

\[
F_{(k, \theta, \sigma)} \left( \frac{x-\theta}{\sigma} \right) = \begin{cases} 
1 - \left( 1 + \frac{k(x-\theta)}{\sigma} \right)^{(-1/k)} & \text{for } k \neq 0 \\
1 - e^{-\left( \frac{x-\theta}{\sigma} \right)} & \text{for } k = 0 
\end{cases}
\]
5.3 Fronthaul DTN

The DTN fronthaul is based on a “neighbourhood networking” concept. We use a rather loose definition of neighbourhood, mainly to contrast our work from obtaining global connectivity: for the purpose of this work, a neighbourhood is a group of at least two users with some common interests. We expect those users to be sufficiently close to each other and/or sufficiently densely distributed that they are often in range of WLAN routers that are either directly connected or can be indirectly connected through message ferrying, e.g., because users with mobile devices move frequently back and forth between the different locations. Intuitively, this would yield distances of 100m to maybe a few km between users. Examples for such neighbourhoods include school campuses, parks, groups of houses in a street or around a lake, entire villages, and downtown areas, but also camp sites in the middle of nowhere.

![Diagram of DTN fronthaul](image)

Neighbourhood networking may take a number of different shapes as shown in Figure 56, all of which we seek to accommodate. At the very core is using 802.11 WLAN to interconnect different nodes, for which our RIFE DTN NAPs serve as the infrastructure, i.e., (groups of) wireless access points: (1) Devices may be connected to an individual NAP as shown at the bottom left, but (2) multiple DTN NAPs may also be interconnected using a wired or wireless link, as shown on the right of the figure. To extend communication beyond the reach of an individual or connected groups of DTN NAPs, (3) we utilise delay-tolerant networking concepts [Fall2003] and allow mobile nodes to carry messages between the DTN NAPs. Finally, (4) we leverage mobile opportunistic networking ideas for information exchange between mobile nodes when they encounter each other (top left of the figure) [Trifunovic2011], possibly assisted by WLAN hotspots [Kärkkäinen2012].

The key constituents are, on the one hand, (stationary) DTN NAP boxes providing the minimal connectivity infrastructure and offering persistent storage, and, on the other hand, mobile devices (e.g., feature phones, smart phones, tablets, laptops) serving both as endpoints and as message carriers. Finally, stationary devices connected to DTN NAPs serve as user equipment or servers.

5.3.1 Dense DTN Fronthaul Dissemination

Trace analysis has shown that human mobility is characterised by large periods of time spent in dense clusters and infrequent flights between the clusters [Gonzalez2008, Rhee2011, Lee2009]. This is intuitive since humans tend to aggregate and spend their time in places such as cafeterias, shopping malls, office buildings, and class rooms. These aggregation points are often covered with Wi-Fi...
networks since ubiquitous Internet access is expected by users of smartphones and laptops. The resulting wireless local area networks can have hundreds of connected mobile devices – we call these dense network segments.

As mentioned in the previous subsection, Wi-Fi networks can be used to create direct contacts between mobile devices to compose an opportunistic network [Kärkkäinen2012], which can serve as a RIFE DTN fronthaul. Much of the existing opportunistic networking research and most system implementations [Doering2008, Helgason2010, Kärkkäinen2012] assume that, at any given time, each node only has a small number of neighbours to which it can connect. This assumption does not hold in dense network segments with hundreds of potential communication partners.

In the DTN fronthaul dissemination work done in the RIFE project and documented here, we show that the basic opportunistic networking mechanisms do not scale to dense segments and suggest mechanisms for improving the resulting performance problems. The results are based on a 50 node testbed from embedded Linux devices, connected to the same Wi-Fi access point, running the Scampi middleware, which we used to run experiments to evaluate message dissemination performance. Our focus is on the content level performance (i.e., application layer) and we do not directly study lower layer, or short timescale mechanisms, or phenomena such as MAC algorithms.

In this deliverable, we document three aspects of RIFE DTN fronthaul optimisation for dense environments: 1) We analyse content distribution performance in a dense network segment using Scampi. 2) We propose and evaluate topology control and request randomisation mechanisms for improving the performance of a single dense network dissemination. 3) We design a practical divide and conquer algorithm for breaking a single dense network segment into multiple smaller, less dense segments by exploiting resource diversity. These advances enable the RIFE architecture to provide good performance in dense DTN fronthauls.

5.3.1.1 Communication Model

We assume a simple model for the communicating nodes, which matches the basic behaviour of various opportunistic communication platforms. We use the Scampi [Karkkainen2012] opportunistic networking middleware – on which the RIFE DTN fronthaul design is based – which gives us control of the basic opportunistic mechanisms that we need. We focus on the relative scaling properties, not the absolute performance of Scampi.

The basic functions that each node implements are:

1) peer discovery,
2) link creation between peers,
3) control messaging, and
4) content exchange.

These are all done on the local network segment through the access point. As a first step after connecting to a network segment, each node first discovers directly reachable peers using, e.g., local broadcast messaging. Next, the node will choose some set of discovered peers to which it opens links, resulting in a logical network topology. We consider the full mesh topology where every node opens a link to every discovered node, and limited mesh where every node randomly opens n links – “n-link limited mesh”. Both sides of a new link will send their content vectors, containing the identifiers of the cached messages, to the other side. The nodes then request a list of all the messages that they wish to receive from the peer, which the peer will start transmitting in order, one at a time, over the link. The peers will continue to send content vectors whenever their cache contents change. In this analysis, we assume that nodes will request all messages from the peers that they do not already have, resulting in
epidemic spreading of the content. More intelligent routing and content dissemination strategies exist, but we focus on the baseline case against which other algorithms can be compared in further work.

5.3.1.2 Single Message Dissemination

To evaluate the basic message dissemination performance, we set up a test with \( N \) nodes, \( N-1 \) of which were connected to the access point with empty caches (5 ≤ \( N \) ≤ 25) and a full mesh of interconnecting links between them. We then connected a node with a 10 MB message in its buffer to the access point and measured the synchronisation time, i.e., the time taken until all \( N \) nodes have a copy of the message. Further, we recorded the number of Scampi message transmissions that took place during the synchronisation.

The results – Figure 57 and Figure 58 – show that both the synchronisation time and the number of message transmissions to reach the synchronised state increase as a function of \( N \) (circles in Figure 57 and Figure 58). We ran the experiments both in a normal office environment with multiple interfering Wi-Fi access points and stations, and in a radio frequency shielded room with only the testbed nodes transmitting. As expected, the shielded room performs better overall, with a significant margin in the
case of a full mesh topology, but the interference does not impact fundamental behaviour of the synchronisation process. We also confirmed that when the number of nodes is kept constant, the synchronisation time only depends on the message size and the total network capacity.

In a simple model we would expect that the node entering the segment will open links to all the other nodes and then send its message over the links in parallel. Although the transmissions over the links are logically parallel, the underlying shared wireless medium serialises and interleaves the packets, leading to a synchronised network roughly in time \( T = (N-1)S/C \), where \( S \) is the size of the message and \( C \) the total capacity of the wireless segment. However, in the results we observe the synchronisation time growing at a faster rate, implying that there is another phenomena hurting the performance. This can be clearly seen in the number of transmissions, which in a simple model should equal the number of nodes, but in our results grows at almost six times that rate – going from 10 to 15 nodes results in almost 30 more transmissions, when it should result in only 5 more according to the simple model.

![Diagram](image)

**Figure 59: Request stacking behaviour for the first three complete transmissions**

We have identified the cause of this growth to be what we call *request-stacking* behaviour -- illustrated in Figure 59. The main cause of this behaviour is that while the node entering the segment will start sending the message to all the peers simultaneously, due to packet drops and other lower level behaviour the transmissions will not finish at the same time. Instead, one of the transmissions will finish first, at which point the node will advertise the received message to all its peers. The peers that have not yet received the full message from the original source will then request a copy, triggering a number of duplicate transfer as shown at the left of the figure. These transfers then takes up capacity in the wireless segment and thus slows down the ongoing original message transfers. This process will continue as the second node finishes receiving the message, and so on, resulting in slower than expected synchronisation time and – given the lack of transfer cancellation – more transmissions.

The aggressive request mechanism that causes request-stacking is a valid approach in the face of unpredictable and highly dynamic contacts. In general, it makes sense to aggressively try every opportunity to get a copy of the message even if a transmission is already ongoing, since the new contact could have much higher capacity or the existing transmission might get cut before finishing. However, in dense, largely stable network segments this approach leads to severely degraded message dissemination performance.
5.3.1.3 Impact of Logical Topology

The request-stacking phenomena observed in the previous section gets worse the more direct neighbours each node has, with the worst case being the full mesh topology where every node is directly connected to all other nodes. We therefore continue by looking at the message dissemination performance when the logical topology is limited, i.e., where each node opens links only to a subset of the potential peers.

**Random topologies:** In the full mesh topology, after receiving a message, each node will start a transmission to every other node that has not yet received the message. This means that the worst-case number of transmissions will be

\[
C = \left( \begin{array}{c} N \\ 2 \end{array} \right) = \frac{1}{2}(N^2 - N),
\]

out of these the number of duplicates is

\[
C_e \equiv C - (N - 1) = \frac{1}{2}(N^2 - 3N + 2).
\]

Therefore, the number of duplicate transmissions is limited to \(N^2\), which makes the full mesh with the naive dissemination strategy, poorly scalable in dense segments.

The natural way to reduce duplicate transmissions is to reduce the number of direct neighbour links – TCP links to other nodes in the segment – that each node maintains. With fewer direct neighbours, each node will advertise its newly received messages to fewer other nodes, and thus fewer duplicate retransmissions will occur. We evaluate 1, 2 and 3 link limited mesh topologies (see Section 5.3.1.1), the results of which are given in Figure 60 and Figure 58 (triangles and crosses).

From the results, it is clear that the duplicate transmission count is roughly proportional to the number of links in the logical topology. And a smaller number of links in the logical topology in turn results in shorter overall synchronisation time, all the way down to 1-link limited mesh. The impact of the limited topology on performance is substantial already in relatively small segments of 10 nodes, and increases as the density increases; in the densest analysed segment the 1-link limited mesh synchronises 4.5 times faster than a full mesh.

We can derive an estimate for the transmission counts – which we have shown to be the major factor in the performance – in limited mesh topologies, shown as dashed lines in the Figure 58. First, since we assume that the topology limitation is done by limiting the number of peer links each node opens, a node will have on average \(2M\) open links. In a network with \(N\) nodes, the topology will approach full
mesh when \( M \geq \frac{(N - 1)}{2} \). Next, from the Figure 58 we can see that there is a linear relationship between the link count \( M \) and the number of nodes \( N \). From this, we can use the full mesh threshold as a base value to which a linear increment of \( M \) times the number of new nodes is added to form the basic equation for estimating limited mesh performance:

\[
C = \begin{cases} \\
\frac{1}{2}(N^2 - N), & 0 < N \leq 2M \\
(N - 1) \cdot M, & N > 2M \\
\end{cases}
\]

for all \( N > 0 \) and \( M > 0 \) describes the behaviour of an \( M \)-link limited mesh topology in a network with \( N \) nodes.

**Star topology:** Another approach to reach a more efficient logical topology is to build it explicitly.

Two obvious topologies that cannot have duplicate message transfers are a spanning tree and a star topology with one central “hub” to which the rest of the nodes connect. Out of these, the former would require running a distributed algorithm to create and maintain the spanning tree as nodes enter and exit the segment, while the latter can be organised simply by, e.g., having every node open a link to the node with the lowest address. The network operator could even provide a fixed node in the segment to act as the hub. Therefore the star topology is likely the more practical topology.

We evaluated the performance of the star topology in the same static scenario as the random topologies, and found that its performance was consistently between the performance of 1-link and 2-link limited mesh topologies. An example of this is given in Figure 60 for different amounts of data to be synchronised. All topologies show linear growth as a function of the amount of data to be transferred, with the star topology performing slightly worse than 1-link limited mesh. This performance difference occurs even though in both cases the number of transmissions is almost the same – with star topology having no duplicate transmissions. We attribute this to the different pattern of parallel transfers in the topologies causing different congestion characteristics on the lower layers. The star topology will start all the transfers in parallel after the hub has received the message, while the limited mesh will have fewer parallel transfers. More parallel transfers means more nodes competing for their share of the shared medium, leading to, for example, less efficient allocation of the resources by the CSMA-CA MAC mechanism.

### 5.3.1.4 Message Spreading Speed

![Spread time, 25 nodes, single 20 MB bundle](image)

![Spread time, 25 nodes, 20 bundles of 1 MB each](image)

Figure 62: Spreading speed of a single message.  
Figure 63: Spreading speed of 20 messages.

In the previous sections we focused on the time and transmissions taken to reach a synchronised state, but did not consider the details of how the state was reached. To fill this gap, we measured the fraction of nodes reached over time as the content was disseminating in the network. The results – depicted in
Figure 66 – show that the previously observed performance behaviour is consistent throughout the message distribution process. The full mesh topology performs worst; while limiting the logical topology improves performance all the way down to the case of 1-link limited mesh, which is the optimal topology. The star topology overall seems to fall between the 1-link and 2-link limited mesh topologies, but has a linear behaviour over time. This is as expected, since the limited mesh will have few concurrent transfers in the beginning that will finish quickly, while the star topology starts all the transfers concurrently. There is a discontinuity for the star topology at around 200 seconds due to a hardware effect. We also confirmed that the spread speed is linear w.r.t. the file size regardless of the topology. This supports the hypothesis that the limiting performance factor is the network node count and not the message size or count.

Importantly, the dissemination starts slower in random topologies with a higher number of links, as can be seen when comparing the spread speed between 1-link limited mesh and full mesh for the first data points (crosses and circles in Figure 62). This is significant in scenarios where a node visits a dense segment for a short period of time, for example when walking by a cafe and connecting briefly to its Wi-Fi access point. In such scenarios using a full mesh topology could mean that the node does not have time to fully transfer the message before exiting the segment. Therefore, new nodes entering a dense segment should have a small number of links to maximise the number of messages they can disseminate to the segment before exiting.

5.3.1.5 Multiple Message Dissemination

The analysis thus far has mainly considered a single message being carried into the network segment. To analyse the behaviour of disseminating multiple messages, which is the realistic case, we ran a set of experiments with 25 nodes and 5 to 25 messages being disseminated instead of a single message. We found that the number of transmissions grows linearly in every topology as a function of the number of messages. We found no other behaviour than the increased capacity taken to transfer the increased amount of data.

We did, however, find the same request-stacking behaviour described previously. Each node will request all $k$ messages the same way as it would for a single bundle, resulting in duplicate transmissions. The number of transmissions for $k$ messages, $N$ nodes and a full mesh topology is at worst

$$C = k + 2k + 3k + \cdots + (N - 1)k = \frac{k}{2}(N^2 - N),$$

which is exactly $k$ times the single bundle case. When looking at the spread speed – shown in Figure 63 – the previously described slow start phenomena is even more severe. This is because in the single message case the message will be requested concurrently by at worst N-1 nodes, after which the number of concurrent transfers will decrease, freeing up capacity. In the multiple message case, however, the number of concurrent transfers stays at N-1 until the first node in the network has received all the messages, only after which capacity is gradually freed. In the example shown in the figure, it takes over six minutes for the first node in the network to receive all the content when there are 20 MB of data to be disseminated.

Random Request Order (RRO): A simple strategy to alleviate both the request-stacking and slow start phenomena – particularly in the full mesh topology – is to randomise the order in which nodes transfer messages. When nodes are transferring different messages between different pairs, the number of duplicate transfers due to request-stacking should decrease, and thus the spread speed should increase.

To analyse the impact of RRO, we apply a simple Fisher-Yates shuffle to the content request lists exchanged by the nodes in the full mesh topology, where the impact is most visible. The result is a
significant decrease in the number of transmissions required to synchronise the network – as shown in Figure 61 – with the full mesh RRO approaching the limited mesh performance. We can further see from Figure 61 and Figure 62 (diamonds) that the spread speed behaviour of the full mesh is radically changed by the RRO. While the total time taken to synchronise the full mesh topology is almost the same in both cases, applying RRO spreads the set of messages in the network more evenly over time. This reduces the slow start phenomena; the first node to receive all the messages is almost the same as in a 2-link limited mesh.

While we have considered basic epidemic spreading of content, the results have implications for more advanced algorithms – for example utility or interest based dissemination. In particular, such algorithms should be wary of mechanisms that might lead to identical request rankings in a large number of co-located peers, and if necessary, add randomisation techniques to prevent it.

5.3.2 Exploiting Resource Diversity for Dense DTN Front Haul Dissemination

The previous sections analysed the scaling of content dissemination in a single network segment where all the transmissions share a limited communication resource. In practice this scenario often arises in Wi-Fi networks, e.g., a single access point with a large number of clients, where the limited communication resource is a Wi-Fi channel (i.e., 20 MHz or 40 MHz of the ISM band). This resource is shared not only between the stations of the access point, but between all the stations of all the access points using the same Wi-Fi channel. This means that only a small fraction of the potentially available 745 MHz ISM bandwidth (depending on the jurisdiction) is being used for the content dissemination. From this it follows that an obvious way to increase the scalability of the content dissemination is to take advantage of the Wi-Fi channel diversity by moving some transmissions to the other channels.

In this section we show that a practical divide and conquer algorithm can be used to break down the content dissemination problem into multiple smaller problems – identical to the one analysed in previous sections – each of which uses a separate communication resource without interfering with each other. Although our experiments focus on the use of Wi-Fi channel diversity, the algorithm can be applied to any heterogeneous set of communication resources that are available to the nodes. This has the potential to greatly increase the scalability by concurrently exploiting an order of magnitude more communication resources than the single channel approach.

We start by explaining the general idea behind the algorithm, then give the details of the algorithm, and finally we discuss various ways in which its behaviour can be specialised for different use cases. While we focus mainly on a single iteration of the dissemination process, we do briefly discuss the continuous operation where content is continuously generated and the algorithm is invoked repeatedly.

5.3.2.1 General Idea

The basic idea of our approach is to move some of the transmissions to use other resources (e.g., Wi-Fi channels) by instantiating a leader-follower group for each additional resource. To do this, we assume a star topology (see Section 5.3.1.3: Star topology), where the star node (“leader”) controls the communication resource (e.g., runs a software access point) and all the connected clients. The leader can then create leader-follower groups by transmitting the content to a new leader and assigning it a communication resource to use and a set of followers to serve. This is done as a separate step before the normal content dissemination takes place using the mechanisms explained in the previous section.
5.3.2 Algorithm Details

Details of the algorithm are given in Algorithm 1 below. For each level of recursion, the nodes to be served are first divided into local clients (served by the current leader) and follower to be served by other leaders (lines 8 and 9). The followers are further divided into sets according to the branching factor, and each follower set has an associated set of resources to be used for serving that group (lines 11..13). For each follower/resource set, one of the nodes is picked as the new leader and one of the resources as the resource to be used by that leader (lines 15 and 16). The algorithm is called recursively for the new leader (line 17) and the associated followers are instructed to connect to their new leader (lines 19..21). Finally the leader serves the local clients according to some dissemination strategy as explained in 5.3.1 (line 24).
While the main structure of the algorithm is given in the listing, there are a number of sub-procedures that can be defined in different ways depending on the scenario and desired behaviour of the process. These are:

- **SELECTFOLLOWERS()**: Selects the set of followers to be recursively passed onto other leaders. The remaining clients will be directly served by the current node.
- **BRANCHINGFACTOR()**: The number of new leaders to pick. The recursion ends when the branching factor is zero.
- **DIVIDERESOURCES()**: Divides the available resources among the new leaders.
- **DIVIDEFOLLOWERS()**: Divides the followers among the new leaders.
- **PICKLEADER()**: Picks a new leader from a set of nodes.
- **PICKRESOURCE()**: Picks a resource from a set of resources.
- **DISSEMINATE()**: Disseminates a set of content to a set of clients in the current network segment. This implements one of the strategies discussed in the previous sections.
The first important sub-procedure is the division of the nodes into locally served clients and followers for the recursively instantiated new leaders, done by the \texttt{SELECTFOLLOWERS()} procedure. To achieve optimum performance, the division should be done such that the time it takes to serve the local clients is equal to the total time it takes to recursively serve the remaining nodes – which leads to every leader finishing dissemination at the same time. Assuming every resource has the same capacity and the recursive leader activation time is negligible, the split can be calculated so that the number of locally served clients is
\[
|S'_c| = \frac{|S_c|}{|S_R| + 1},
\]
and the number of followers for the other leaders is
\[
|S_f| = \frac{|S_c| \times |S_R|}{|S_R| + 1}.
\]
We call this the \textit{naive} split. We have shown that if the activation of the communication resources by new leaders is not instantaneous, the naive split leads to significantly suboptimal performance, which can be alleviated by modifying the \texttt{SELECTFOLLOWERS()} procedure.

The second important issue is the division of the nodes and their resources into the recursive branches by the sub-procedures \texttt{DIVIDEFOLLOWERS()} and \texttt{DIVIDERESOURCES()}. This division can take into account the heterogeneous communication capabilities of the nodes – e.g., the support for the 5 GHz Wi-Fi band – and group the nodes accordingly. Further, the differing capacity of the communication resources can be taken into account. This can include differences due to the technologies (e.g., 20 MHz vs. 40 MHz Wi-Fi bandwidths, or Bluetooth vs. Wi-Fi bitrates) and due to the environment at runtime (e.g., the number of other Wi-Fi access points operating on the different channels). We have shown that imbalanced division of the load among the branches of the recursion leads to significantly suboptimal performance, and therefore care needs to be taken when realising the division procedures.

The final important issue defined by the sub-procedures is picking of the new leaders for the recursive branches, done by the \texttt{PICKLEADER()} procedure. This is important because the leaders control the communication resources and are responsible for disseminating the content to the clients and for running the dissemination algorithm. The choice of the leaders can be done based on their capabilities (e.g., the ability to run a software access point) and their resources (e.g., the battery level). Further, as far as possible, the leaders should be nodes that are likely to stay within the network for longer periods of time (e.g., not mobile nodes passing by), and could be (opportunistic) infrastructure components installed in the environment, such as the Liberouter devices.

\subsection*{Continuous Operation}

The description so far deals with a single iteration of a content dissemination process, while in a real system the process would run continuously. There are two main aspects to continuous operation beyond the algorithm described in the previous section: 1) the selection of the initial leader and the content to be disseminated, and 2) reforming back to the initial state once the recursion has completed. Varying degrees of complexity and intelligence are possible for both of these aspects. The first aspect is straightforward to solve by a distributed consensus algorithm such as Paxos [Lamport1998] or Raft [Ongaro2014]. First a set of potential leaders – nodes with content they wish to disseminate – announce their availability, after which a leader election process is run to pick the leader for the next dissemination round, or a sequence of leaders for multiple rounds.

For the second aspect, the simplest approach to reforming back to the initial state is for all the nodes
to connect back to the initial network segment once they have received the content. This approach is also practical since in real world use cases the initial network segment is likely to be a network that provides Internet access (e.g., a venue Wi-Fi network) – while the segments used for the recursive dissemination process do not – so it is desirable to move back to the initial network as soon as possible to minimise the disruption of Internet connectivity. However, an intelligent group forming process provides opportunities to optimise the dissemination, although detailed analysis of such mechanisms are outside the scope of this deliverable. For example, if the dissemination groups are formed such that the clients share common interests, or applications, it is likely that they can exchange more content than the initial set. This could be achieved by running a leader and content selection algorithm within the dissemination group – followed by another dissemination round – before reforming back up to the initial state. This process can also be done recursively, by each level of recursion returning to the group of the leader that initiated it originally. By doing more disseminations before forming back into the initial state, less load is also placed on the communication resource used for the initial leader selection.

5.3.3 Modelling DTN Fronthaul Dissemination

The feasibility and viability of the purely opportunistic variant of the RIFE fronthaul DTN solution depends on two fundamental questions: can the community of mobile users ensure the required application performance (e.g., connectivity or content spreading), and is the required contribution from the users low enough, so that they are willing to participate?

In this section we address a specific kind of DTN fronthaul networks where content is shared for location based services. The service is targeted to mobile users in urban areas, and can aim for spreading information, such as of local news, tourist information, transportation schedule or traffic alerts [Trifunovic2017], or for forming opportunistic social networks in transient communities [Kärkkäinen2016]. In these systems, the dynamically changing user community provides a virtual storage system, and the scheme is feasible if content can survive in the area for a long time.

We approach the question of feasibility utilising the framework of stochastic epidemic modelling [Tornatore2005]. For epidemic models, a main concern is to find conditions under which an infection introduced into the community will develop into a large outbreak, and if it does, to find the conditions under which it will become endemic — able to persist in the population: a direct analogy of the content survival.

We have defined a Markovian model of opportunistic sharing of localised content, and describe the content spreading with a system of stochastic differential equations (SDE), that can reflect deterministic changes in the population, as well as random disturbances. We have applied SDE stability theorems to capture the main operating regions of the opportunistic content sharing system. We have validated the model and evaluated the conditions of long-term content survival by comparing our analytic results to simulations utilising realistic mobility traces.

Our results show that the tractable epidemic model approximates well the main operating regions of the content sharing system, and can be used to tune the system parameters such that long-time content sharing becomes feasible. The results demonstrate the viability of opportunistic content sharing, showing that the contribution required from the individual mobile users is rather limited, but also prove the importance of adaptive schemes that can minimise the user contribution under changing population size.

In this deliverable, we present the basic model, while the details, analysis and the evaluation is
5.3.3.1 Content Sharing Scheme Operation

We consider an opportunistic service for sharing geographically localised contents where, in addition to obtaining the content, all users are willing to support further content spreading by contributing some amount of their resources for a limited time. The content is geographically tagged to the region of relevance, and it is considered irrelevant and will not be replicated outside of the boundaries of the locale.

The area of interest is characterised by frequent arrivals and departures of users equipped with mobile devices. Applications on user devices use publish/subscribe services [Helgason2010] to publish new contents, or to find peers within communication range and download or forward contents. That is, a single user publishes a content item initially, which is then shared relying exclusively on opportunistic content sharing mechanisms, without the support of an infrastructure. The opportunistic service is feasible if the content persists in the area for a very long time.

As the basic forwarding principle, opportunistic communication schemes usually assume some flavor of epidemic routing [Vahdat2000], where any node that has a content item forwards it to any encountered one that has not obtained the content yet. While this principle guarantees the largest spread of content in the network, it also imposes a high resource overhead. In this study, we consider a modified scheme with time-limited forwarding [Zhang2007], where each node starts a timer when it receives the content item and continues forwarding the content until the timer expires. While the mobility of the users, the resulting population size and contact pattern is determined by the environment, the forwarding time is a system parameter that can be tuned and optimised to ensure content survival with minimum user contribution.

5.3.3.2 The Epidemic Model of Localised Content Sharing

We aim at modelling the content sharing process as a stochastic susceptible-infected-recovered (SIR) epidemic model. In this context, nodes in the area of content sharing can be in one of three classes: susceptible (S) are the nodes that have not obtained the content item, infected (I) are the nodes currently holding and forwarding the content (we will also use the term content carriers) and recovered (R) are the nodes that have the content but already stopped forwarding.

This system model is commonly referred to as a compartmental model, since each node in the population belongs to one of the three classes – compartments; nodes in one compartment are indistinguishable from one another with respect to their infectious status and contact pattern.

In [Pajevic2015, Pajevic2016] we have validated that location based content sharing can be modelled by such a SIR model, by abstracting away the exact mobility pattern of the nodes. The results is a black-box system model where nodes are fed into the area of content sharing and are released after some time, characterised by the arrival process and by the sojourn time distribution. Inside the area, nodes interact according to the homogeneous-mixing model proposed in [Bensal2007], which implies that the probability of two nodes establishing contact is equal for any two nodes. Consequently, interactions can be characterised by the contact rate representing the number of peers each node meets in a unit time.

Epidemic processes that do not experience any noise can be modelled by deterministic SIR models. However, when modelling physical or biological systems, such as the considered content sharing

---

system, the mathematical models need to account for non-trivial random effects, and build on stochastic SDE models [Tornatore2005]. A key concept of the dynamic behaviour of the epidemic processes is the epidemic threshold [Anderson1992]. This threshold signifies a transition point between two cases: the first case when only a limited number of individuals are affected in the course of epidemic spreading and the epidemic eventually vanishes from the population, and the second case when the epidemic outbreak reaches a finite fraction of the population and stays around that level as the time progresses. One of the established results in theory of stochastic and deterministic models, is their significantly different behaviour with respect to the threshold [Pastor-Satorras2015]: while the deterministic models guarantee survival of the epidemic above the threshold, in stochastic models this is only a necessary condition: the epidemic is still prone to stochastic variations, which can lead to epidemic extinction.

5.3.3.3 System Model and Notation

Following [Pajevic2015, Pajevic2016] we build up a Markovian model of localised content spreading, accounting also for random disturbances of the system state. We consider a homogeneous system, because the changes of the node mobility patterns are typically much slower than speed of the content sharing.

Susceptible nodes arrive to the considered area according to a Poisson process with rate $\lambda$, and stay in the area for an exponentially distributed sojourn time with mean $TS = 1/\mu$. Once infected, nodes forward the content during an exponentially distributed forwarding time with mean value $TF = 1/\gamma$. Following a basic epidemic routing scheme, all nodes fully participate in the spreading process; thus each node forwards the content to any susceptible node it encounters during its infectious period. We assume that the contact time is long enough for forwarding the entire content. Following the homogeneous-mixing model, [Bensal2007], we assume that the inter-contact times that a node experiences are independent and exponentially distributed, and are characterised by the mean contact rate $c_N$ for an average population size $N$. The contact rate $c_N$ depends on the node mobility inside the area, and is one of the input parameters of the model.

![Diagram of SIR compartments with transition rates.](image)

$S(t)$, $I(t)$ and $R(t)$ denote the number of susceptible, infected and recovered nodes, respectively, at time $t$, and the total number of nodes is $N(t) = S(t) + I(t) + R(t)$. Nodes arrive to compartment $S$ with intensity $\lambda$, and may leave the system from any compartment with intensity $\mu S(t)$, $\mu I(t)$ and $\mu R(t)$. Nodes move from $S$ to $I$ as the infection spreads. We follow [Pajevic2016] to define the state dependent infection rate. The contact rate at time $t$ depends on the mean contact rate $c_N$ as well as on the instantaneous population size $N(t)$, and can be approximated by $c(t) = c_N N(t)/N$. Given that the infection to happen a susceptible node has to meet an infected one, the infection rate becomes $\beta(X(t),t) = S(t)I(t)c(t)$. Finally, nodes recover with a rate $\gamma I(t)$.

In addition to these deterministic state changes, the SIR model needs to account for random disturbances. The results of [Pajevic2015, Pajevic2016] show that all compartments experience randomness, with a magnitude that depends on the system state, and the epidemic process can be
modelled by system of stochastic differential equations over $X(t)=[S(t), I(t), R(t)] \in \mathbb{R}^3$ as

\[
\begin{align*}
\frac{dS(t)}{dt} &= [\lambda - \beta(X(t), t) - \mu S(t)]dt + g_1dW(t) \\
\frac{dI(t)}{dt} &= [\beta(X(t), t) - (\mu + \gamma)I(t)]dt + g_2dW(t) \\
\frac{dR(t)}{dt} &= [\gamma I(t) - \mu R(t)]dt + g_3dW(t) \\
X(0) &= [S(0), I(0), R(0)]^T,
\end{align*}
\]  

(1)

where $W(t)$ is a vector of six independent Wiener processes and $g_i, i = 1, 2, 3$ is the $i$-th row of the matrix

\[
G(X(t), t) = \begin{bmatrix}
\sqrt{\lambda} & -\sqrt{\beta(X(t), t)} & 0 \\
0 & \sqrt{\beta(X(t), t)} & -\sqrt{\gamma I(t)} \\
0 & 0 & \sqrt{\mu R(t)}
\end{bmatrix}.
\]  

(2)

Denoting by $F=F(X(t), t)$ the function governing the deterministic part of system (1)

\[
F(X(t), t) = \begin{bmatrix}
\lambda - \mu S(t) - \beta(X(t), t) \\
\beta(t, X(t)) - (\mu + \gamma)I(t) \\
\gamma I(t) - \mu R(t)
\end{bmatrix},
\]  

(3)

we get a compact form of our system model

\[
dX(t) = F(X(t), t) + G(X(t), t)dW(t).
\]  

(4)

At the time when the content is first published there is a single infected node in the area and $S(0)=S_0$ susceptible nodes, thus the initial system state is $X(0)=[S_0, 1, 0]$. The solution of (1) can be found in a form of probability distribution given by Fokker-Planck formula [Risken1984] in the case when such distribution exists. However, in this work we are seeking answer to the more fundamental question, that in turn would help the design of opportunistic content sharing systems: we would like to know, under what circumstances does the system ensure that the content survives in the area and can be shared for a long time.

5.3.3.4 Conclusion

Based on the previous model, we have studied under what conditions this content spreading scheme is feasible – that is, the combination of mobility parameters such as node arrival rate, sojourn time and contact rates, and the tunable system parameter, forwarding time. We modelled the spreading process with a stochastic epidemic model and employed Lyapunov stability theory for stochastic systems to establish conditions when the system reaches the state of persistence, in which the content is likely to survive for a very long time. The validity of the system model as well as the established analytical results are confirmed via simulations using realistic mobility traces.

Our results are valuable for the RIFE system design, as we provide theoretic tools to predict the success of the opportunistic content sharing, and to tune the system parameters to ensure that the content survives in the area for a long time. The numerical results considering realistic mobility scenarios show that the forwarding times required by the users are relatively short – ranging from less than a minute in dense environments to a couple of minutes in very sparse scenarios. This finding can help to give incentives for the users to participate and share contents since the requirement for user resources is obviously not that large. We also find that the population size and thus the contact rate has a significant impact on the forwarding time required for content survival. This motivates future research on the design of adaptive schemes, that can estimate the required forwarding time. These schemes could then ensure successful content sharing, while minimising the contribution and thus the energy consumption of the participating users.
6 SYSTEM COMPONENT DESIGN

This chapter describes the RIFE architecture system component design. We first describe the NAP design that integrates our two dissemination strategies (well-connected ICN and disconnected DTN) in Section 6.1. Since these two dissemination strategies differ greatly in their field of applicability, a typical scenario, especially for the DTN dissemination strategy, is to deploy it alone. Therefore we present a pure DTN-based NAP in Section 6.2, with two variations. The RIFE border gateway system design, including the edge cache design is given in Section 6.3. Although most RIFE deployments will not need modifications to the hosts, the native DTN dissemination strategy does require RIFE-specific components to run on the user equipment, which are described in Section 6.4.

6.1 Integrated Network Attachment Point

The design of an integrated NAP, as introduced in Sections 4.1 and 4.2, is provided in this section and focuses on the functional components identified in order to enable the architectural role of the NAP. Figure 66 illustrates those functional components which are all interconnected via arrows indicating the data flow.

![Functional components of an integrated Network Attachment Point (NAP) design](image)

All blocks coloured in black are functionalities which do not come with the NAP and are provided either by another software entity, e.g., Blackadder providing the ICN core node platform, or those blocks and standard functionalities realised by off-the-shelf solutions following existing and ratified standards. All blocks required to enable the translation from a communication attempt from an IP endpoint into the corresponding dissemination strategy and vice versa is grouped in Figure 66 into the square with the dotted line and is referred to as “integrated NAP”.

**IP endpoints** are devices which communicate with another IP endpoint via a standard IP stack in respect to the OSI model. All message exchange supporting the communication attempt with IP endpoints in Figure 66 is depicted with an orange arrow. IP endpoints located in less connected areas utilise DTN applications rather than standard IP-based software, e.g., web browsers.

The **IP Gateway (GW)** provides typical gateway functionality to IP endpoints, such as IP address assignment, NAT and firewalls, as outlined in Sections 4.1 and 4.2.

The **ICN Core Node Platform** is the interface towards the information-centric network. As the underlying ICN core node platform, the NAP will utilise the PURSUIT prototype Blackadder [TRO12].
6.1.1 Detailed Description of the Functional Blocks

The coloured functional blocks in Figure 66 depicting NAP internals will be described in this sub-section.

6.1.1.1 Packet De-multiplexer

The demux, short for packet de-multiplexer, allows the NAP to analyse the incoming IP packet and determines the correct handler which performs the ICN namespace abstraction according to the handler’s name. Furthermore, as ARP is an essential feature to enable IP communication between IP endpoints, a dedicated ARP listener is also available reacting to ARP requests if required (a more detailed description on the ARP listener is provided in Section 6.1.1.2). As this decision is based on the network protocol type and destination transport protocol port number, the demux switches based on the implicit knowledge of pre-defined packet header information.

6.1.1.2 ARP Listener

In order to allow the IP network protocol to enable the communication between two network elements, 802.3 is one of the most common data link protocols which allows to carry IP packets as their payload. As 802.3 requires the MAC addresses (EUI-48 format) of the two Network Interface Cards (NICs) which are supposed to exchange an IP packet, the Ethernet Address Resolution Protocol (ARP) is in place to allow a network element to obtain the MAC address of the destination NIC [RFC826]. The NAP must act as an ARP proxy if a default gateway has not been configured in the IP endpoint and for all destination IP addresses the IP endpoint attempts to reach. The NAP allows replies with its own MAC address so that the IP communication gets established. Furthermore, if the NAP receives a message over ICN and needs to place the packet towards the IP endpoint, an ARP request/response communication would be required in case the NAP does not know the MAC address of the destination IP endpoint.

6.1.1.3 DTN Handler

This handler basically represents the functional environment to enable applications specifically designed for DTNs. The functional concepts to enable a store-carry-forward instead of a directly connected end-to-end client server communication is described in detail in Section 5.3; in a nutshell the DTN handler combines a service discovery and transport layer to allow the exchange of data. Whenever the DTN handler attempts a communication with a remote NAP, the IP handler is used which translates the IP-session into ICN, as described in the next section.

The DTN handler also consists of a proxy element which allows the NAP to bundle websites such as fetching the sites elements, rewriting paths for local rendering and composing this into a message back to the DTN application. Furthermore, the DTN proxy acts as the TCP endpoint for IP endpoints and allows to perform caching functionalities.

6.1.1.4 IP Handler

For all IP traffic which cannot be handled by any other handler, the IP handler is used which allows the NAP to perform IP-over-ICN with another NAP using the IP namespace, as described in [POI21]. The IP handler performs any action required to publish the received IP packet to the ICN network including publication of scopes, advertising of available data and eventually publishing the data packet if a subscriber is available. In case there is no subscriber available at the time of attempted publication the IP handler buffers the packet for a predefined time which can be configured.
6.1.1.5 HTTP Handler

All IP traffic which is using TCP with the destination port 80 (HTTP) is forwarded to the HTTP handler which allows the NAP to perform HTTP-over-ICN following the HTTP namespace, as described in [POI31]. The HTTP handler basically analyses the HTTP message extracting the HTTP method, host and resource information in order to publish the packet under the correct HTTP scope path. As for the IP handler, the HTTP handler allows to buffer packets if a subscriber is not active at the time of advertising the availability of data.

As for the DTN handler TCP sessions towards IP endpoints are terminated at the NAP. Hence, a transparent HTTP proxy is required which handles the TCP session with endpoints connected to the NAP. This proxy acts as a TCP server of the IP endpoint when the endpoint issues an HTTP request and as a TCP client if the HTTP request has been received over ICN and needs to be placed towards a HTTP server.

Furthermore, the HTTP handler also provides a MIME de-multiplexer which allows the NAP to further analyse the HTTP packet regarding the content type in order to enforce a more efficient handling of packets than a basic HTTP handler would do. The benefit of doing this is that features such as co-incidental multicast and less domain local RV involvement can be achieved, which significantly improves the overall performance of the network.

6.1.1.6 ICN Handler

The ICN handler acts as the counterpart to the IP and HTTP handlers and allows the NAP to handle incoming ICN packets forwarded by the local ICN platform and places them back into the HTTP handler where the proxy holds the active TCP session with the IP endpoint or it sends the packet straight to the IP endpoint. Not only do ICN packets arrive at this handler carrying data targeted at IP endpoints that are served by the NAP, scope related control messages also arrive at this handler notifying the NAP about new/leaving subscribers to a particular scope and new published/unpublished scopes under a father scope for which the NAP has subscribed.

6.1.2 Interfaces

6.1.2.1 NAP-IP interface

As depicted in Figure 66, the NAP communicates with IP endpoints using a standard IP stack with 802.3 as the data link protocol. This ensures that all use cases envisaged by RIFE can be supported given the penetration of devices and servers which communicate with IP over 802.3.

6.1.2.2 NAP-ICN interface

The interface towards the ICN core platform utilises the specification of Blackadder, developed in PURSUIT and currently further improved in POINT [Poi2015]. Currently, Blackadder v1.0.0, which is available at the Github repository under GNU general license v3, exports a pure publish/subscribe service model to applications. This service model allows for publishing and subscribing operations over an information model that is based on a directed acyclic graph, following the PURSUIT architecture [TRO12].

6.1.3 Sample implementation

Currently, POINT has released Blackadder 1.0.0 which provides the ICN core platform for well-connected front-hauls and a NAP implementing the IP-over-ICM dissemination strategy. Scampi – the DTN counter-part of the POINT NAP, has already reached a more sophisticated state considering the
larger amount of resources that went into it. Both solutions are going to be integrated at a platform level leveraging the IP interface Scampi provides which will directly feed into the POINT NAP.

As described in Section 2.1.2, disconnected ICN front-hauls require Scampi in order to allow a store-carry-forward communication. When integrating Scampi with the POINT NAP at the UE side of the front-haul, Scampi is going to be placed between the UE and the POINT NAP allowing both solutions to be working completely independent from each other. Each communication attempt from Scampi towards the RIFE BGW results in an IP-over-ICN communication which is entirely hidden from Scampi and its UEs. The RIFE BGW then translates the IP-over-ICN communication back to IP and feeds it to the edge cache, i.e., Scampi middleware.

6.2 DTN Network Attachment Point

The RIFE platform is capable of delivering communication services in environments without fixed front-haul infrastructure. This is done by applying Delay-Tolerant Networking (DTN) concepts of store-carry-forward communication (see D2.2 Section 3.4 for high level explanation). The realisation of this DTN-based dissemination strategy within the RIFE design is achieved through dedicated DTN NAPs and DTN components in the RIFE border gateway. The DTN platform used is the Scampi opportunistic networking middleware [Karkkainen2012], originally developed in the EU FP7 SCAMPI project and further developed by the Aalto University and Spacetime Networks Oy.

This section details the system component design for the Scampi based NAP for the DTN-based dissemination strategy. We present two configurations: 1) native Scampi NAP, and 2) Web NAP. The latter provides services to end users with standard web browsers, while the former requires Scampi support in the end user devices, but enables new applications and services to be built natively on the Scampi platform. Further, we present two different designs of the DTN Web NAP, which present a different platform towards the web application developers.

6.2.1 Native DTN Network Attachment Point

At system level, the native variant of the RIFE DTN NAP behaves like a message board from which passing nodes can read messages, and to which they can write their own messages. This conceptual model is shown in Figure 67. As shown at the top of the figure, the RIFE DTN NAP can be seen as a large message buffer, where each message is a semantically meaningful and self-contained application data unit (i.e., a full photo rather than an IP packet). Input and output ports allow nearby UEs to connect to the DTN NAP. Connected UEs can leave messages into the DTN NAP from their own message buffer, and pick up new messages from the DTN NAP’s message buffer.

6 http://www.ict-scampi.eu
The main component of the DTN NAP is the opportunistic networking middleware that implements the delay-tolerant store-carry-forward networking used by the fronthaul. While multiple different options for the middleware exist, the RIFE platform design is based on the Scampi middleware.

Figure 68 shows the high level design of the Scampi middleware. At its core, the Scampi middleware maintains filesystem based persistent storage for the messages, and creates transient communication links (“contacts”) with other nearby Scampi instances. The contacts can be created over any communication technology with Scampi drivers, but in the RIFE DTN NAP design only Wi-Fi is used. The main functionality of Scampi is to use the created contacts to spread the content from its storage around in the network, and to provide an API for applications to take advantage of the resulting communication network. Details of the Scampi architecture can be found in [SCAMPI-D1.3].
The native DTN NAP implements the following interfaces and protocols:

- **Link layer towards the UEs**: The DTN NAP creates an open 802.11 access point, and runs a DHCP server to provide IP addresses to the clients. From the UE’s perspective the NAP appears as a standard open Wi-Fi access point, but without Internet connectivity.

- **Service discovery**: The Scampi middleware implements an IP-multicast based discovery protocol for connected nodes to discover each other. The details of the discovery protocol are given in [SCAMPI-D1.3] Section 3.3.

- **Transport**: The Scampi middleware implements the TCP Convergence Layer (TCPCL) protocol for transport of messages between nodes. The details of the protocol are given in [RFC7242].

- **Messaging**: The Scampi middleware implements the Bundle Protocol for store-carry-forward messaging. The details of the protocol are given in [RFC5050].

- **Message framing**: The Scampi middleware defines a typed key-value map framing on top of the Bundle Protocol. The details of the framing are given in [SCAMPI-D1.3] Section 3.4.

- **Control protocol**: The Scampi middleware defines a control protocol over the TCPCL. The details of the control protocol are given in [SCAMPI-D1.3] Section 3.2.

- **API**: The Scampi middleware provides a native API to locally running applications. The API provides a publish/subscribe messaging model for applications. Details of the API protocol are given in [SCAMPI-D1.3] Section 3.14. In addition, the Scampi middleware provides and HTML5 based API, but it is not relevant for the RIFE NAP use.

### 6.2.2 DTN Web Network Attachment Point

The native RIFE DTN NAP presented in the previous section requires the UEs to be running the Scampi platform in order to access the services. While this enables novel applications to build directly on top of the DTN layer, and the presence of additional mobile Scampi nodes improves the fronthaul content dissemination performance, unmodified UEs must be supported for incremental deployability and device support reasons. The RIFE DTN Web NAP addresses this by building on the native DTN NAP with a framework for providing access to the services through a standard web browser in the UEs.

Supporting modern web applications in a DTN environment is challenging. The applications are architected around the assumption that a fast, low latency path exists between the web browser in the UE and the application developer’s servers. This leads to designs with centralised data stores and frequent interactions between the front-end in the browser and the back-end in the datacentre, as shown in Figure 69. Such designs are fundamentally incompatible with the DTN paradigm, where end-to-end low latency paths might never exist and the content is exchanged directly between nearby peers.
The starting point for our design is the conceptual decomposition of web applications into: 1) the *data*, and 2) the *logic* that operates on the data. In traditional web application architectures, the *data* is stored in centralised databases operated by the application developer (bottom right of Figure 69). As the first step in adapting these applications to work with a RIFE DTN fronthaul, the data storage model must be changed from a centralised into decentralised design. This is done by breaking the application content into self-contained, semantically meaningful units, which are then published as messages directly into the DTN fronthaul. For example, a post to a social media platform might be made up of a photo, text, and tags, all of which would be bundled together into a single DTN message. The caching nature of the DTN fronthaul network makes the message, and the post contained in it, directly available to nodes connected to it, without the need for a separate centralised data store.

In addition to the data, web applications are composed of the *logic* that operates on the data. In traditional web applications, the logic can be separated into back-end logic running on the servers (typically some form of Create-Read-Update-Delete logic to maintain the data), and front-end logic in the web browser that provides the interface for the user to interact with. In our DTN adaptation, we take the logic and bundle it together with the data into the same message, as shown in the middle of Figure 70. This combined data + logic message then contains everything needed by the RIFE DTN NAP to present the web application to a connected UE running a standard, unmodified web browser, as shown on the right of Figure 70.
Figure 70: Adapting classical Internet based web applications into the DTN based RIFE dissemination strategy

We are designing and prototyping two candidates for the web framework for RIFE, as shown in Figure 71. In both cases we add web server components to the RIFE DTN NAP, which allows connected UEs to access web applications. The core design in both approaches is the same, but they differ in how the application logic is handled.

In the first approach, on the left of Figure 71, the application logic is bundled with the data as scripts that implement a set of generic functions (summary, presentation, reply, new) that can be applied to the contained data item. These functions are then called by the web framework’s generic application model in order to create an HTML representation of the data to be sent to the UE’s web browser. The advantage of this approach is that the logic is distilled down to a small set of functions which do not add significant overhead to the messages. The disadvantage of the approach is that the web application developer has to develop these scripts separately. The technical details of the framework are given in [Nagy2015].

In the second approach, on the right of Figure 71, instead of using purpose built scripts executed by the web framework for the logic, the RIFE DTN Web NAP instead exposes a generic REST API and all the application logic is built similarly to the classical web application front-ends. The front-end is bundled with the data, and passed to the UE’s web browser by the NAP. The benefit of this approach is that the front-end logic is developed the same way as in the classic web applications, which enables most of the existing front-end code to be reused. The downside is that the full front-end logic is likely to be much larger than the small set of scripts required by the other approach, and thus adding more overhead. The technical details of the framework are given in [Arnold2016].
Figure 71: Two implementations for the Web support in the Scampi DTN NAP

The RIFE DTN Web NAP has the following interfaces and protocols:

- **RIFE DTN NAP**: The DTN Web NAP extends the native DTN NAP by adding the web framework component (built on the Scampi API). Thus all the interfaces of the native DTN NAP are included (see Section 6.2.1).

- **Script based web framework**: The framework exposes interfaces for application developers to create presentation scripts to be attached to content, and a normal web server interface towards the UEs. The details of the interfaces are given in [Nagy2015].

- **Full front-end based web framework**: The framework exposes interfaces for application developers to bundle full web front-ends (HTML, JS, CSS, etc.) with the content, a standard web server interface towards the UEs, and a REST API for the front-end. The details of the interfaces are given in [Arnold2016].

6.2.2.1 DTN NAP Sample Implementation

The native RIFE DTN NAP implementation is based on the Liberouter⁷ neighbourhood networking.

---

⁷ [http://www.liberouter.mobi](http://www.liberouter.mobi)
platform [Liberouter2014]. It is a combination of an embedded Linux device (e.g., a Raspberry Pi as shown in Figure 72) and the Scampi opportunistic middleware software. The device is configured to function as a normal Wi-Fi access point, but without Internet connectivity. A captive web portal on the device lets users download the Scampi platform and various native applications (shown in Figure 73) for Android devices, in order to access the services provided by the NAP.

![Figure 73: Two examples of native Scampi ICN applications that are served by the RIFE DTN NAP](image)

The implementations of the two different approaches to RIFE DTN Web NAPs are still ongoing. The initial implementation of the script based NAP implementation is described in [Nagy2015]. As described in the previous section, they both build on the native RIFE DTN NAP and therefore share the hardware and Scampi components, while adding implementations of the web framework into the device.

### 6.3 Border Gateway

As shown in Figure 74, the RIFE border gateway sits in between IP and ICN network and preserves all the functions as Blackadder and Scampi provided in ICN border gateway, where the community network and Delay Tolerant Network are attached to, while providing edge caching capability with Squid proxy via IP backhaul network. Further to the design of NAP as described in the previous section. This section will detail the system design of the edge cache.

![Figure 74: RIFE border gateway](image)
6.3.1 Elements

Figure 75 illustrates the minimum deployment of the proposed edge cache and shows the main components and interfaces.

The components are:

- **Policy manager** sits at the heart of the whole framework, it performs two key tasks as computational problems:
  
  a. **Utility estimation**: for each advertised content, computes and quantifies the relevance to respective edge caches, namely, utility index, based on the temporal and spatial information advertised by the publisher (e.g. popularity, location region, social relationship, publish period and network topology). This module is responsible for generating the following key metrics for facilitating the retrieval, store and dissemination of the content:
    
    i. When to place the content, e.g. 5pm-10pm every Tuesday.
    
    ii. Where to place the content, e.g. node x and y.
    
    iii. What is the retention policy, e.g. from today for 2 weeks.
    
    iv. A derived index format could be: dd:mm:yyyy-time-node-xxx

  b. Policy assignment: based on the utility index and QoS requirement, selects the most appropriate policy attached to the content.

- **Content placement**, based on the utility index, places the content to the most appropriate
node in the infrastructure.

- **Cache allocation** performs optimal allocation policy that is implemented via two stages of allocation: 1) in-node multi-cache allocation based on policy-based queuing/caching requirements; 2) inter-node cache allocation aimed at distributing the cache capacity across the network under storage budget constraints (optional, this can work with content placement where resource is limited in the network). This module is the manager of the physical cache storage and is responsible for determining:
  a. Which cache is to be used for caching the content, i.e. cache ID
  b. What type of cache shall be assigned based on the content characteristics, the content can utilise different types of storage spaces:
     a. In-memory cache: provides faster file retrieval, but the size of a cache is limited
     b. Disk cache: allows storing a larger amount of cacheable objects with the trade-off of a less effective response time

- **Access control** defines how the content can be accessed based on:
  b. Allowed user lists
c. Retention period
d. Allowed access time
e. Authentication methods (if applicable)

### 6.3.2 Interfaces

- The related interfaces are described as follows: PM-CP: provides communication between PM and CP where the decision made for which content is to be placed will be passed to CP for placement.
- PM-AC: provides communication between PM and AC where the allowed access time and user list will be passed to AC.
- AC-CA: provides communication between AC and CA where content will be cached only when the criteria of AC is met.
- PM-CA: provides communication between PM and CA where the decision made for how to allocate into cache will be passed to CA for allocation.
- CP-CS: provides communication between CP and CS to send information needed to store the content into physical storage.
- CA-CS: provides communication between CA and CS to send information needed to store the content into physical storage.

### 6.3.3 Sample Implementation

A sample realisation of our solution aims at implementing the followings:

- Interaction with the end user (some form of website dialog) to determine size (s) and retention time (t) for some web content, the latter defined through some algorithm as a set of URLs (u)
- Realise a policy agent that does the following:
Create a directory with quotas and add this directory to an exclusion list.

Temporarily configure Squid to use this directory only.

Issue HTTP requests to the set of URLs U (which Squid will now proxy upon return in the chosen directory).

Re-configure Squid to again use all directories, i.e., the previously configured ones as well as the newly created one.

- Have Squid modified so that the cache selection algorithm (e.g., RR) will NOT choose any directory from the exclusion list - Squid will, however, continue to use ALL directories for retrieving cached content (determined through the digest).

- A background daemon run that removes a directory from the exclusion list after its retention time \( t \) expires, therefore adding the directory to the normal pool of directories that Squid uses. Alternatively, the directory could be flushed or removed upon retention expiry.

The implementation involves the following main steps:

- Policymaker sends the request including URL sets, retention time, requestedSize, userList via policy submission GUI.

- The ECS will apply a revised policy configuration file upon receiving the policy request.

- The normal caches are all locked for storing.

- A dedicated policy cache is created with the size equals to the requested size.

- The requested URL is added to the allowed destination IP in the proxy setting, while all other IPs are temporarily blocked.

- The ECS will send the http request to fetch the content from network origin.

- Once the content fetching completes, the ECS will lock the policy cache until the policy start time.

- Allowed user list will be configured in the proxy settings for user access.

- When the requested retention time is over, the dedicated cached will be flushed and removed.

The implementation of the above procedures involves the interaction amongst different functional models between the client and server, Figure 76 shows the interaction of the related information modules.
6.4 User Equipment

Most dissemination strategies in the current platform design do not require RIFE-specific components in the UEs, and thus standard devices can be used. The only configuration that requires support in the hosts is the native DTN dissemination strategy, which is described below.

6.4.1 Native DTN Host

The system components required for the native DTN host support are shown in Figure 77. The main component is the Scampi opportunistic middleware instance that allows the device to take part in the DTN fronthaul network by connecting to other nearby devices and DTN NAPs. The middleware is shipped as a standalone Android application, which starts a permanently-running background process...
to run the middleware and provides a UI for controlling and monitoring it. The Scampi instance is identical to the one running in the native DTN nap, and described in Section 6.2.1.

The native DTN applications are developed as standalone Android applications that use the Scampi middleware services through the Scampi API.
7 CONCLUSION

This document provided the description of the final platform design, including components and dissemination strategies, and the related implementation work taken in the RIFE project. The starting point was the physical network infrastructure that the platform builds on, including a satellite backhaul and a community network as a fronthaul. Next, we provided a brief description of the RIFE architecture, before describing in detail the fronthaul and backhaul dissemination strategies. Finally, we provided the initial system component design for the RIFE platform.

This document relies on the work done in the deliverable D2.2, which describes the components and their interfaces in the architecture, as well as gives the ICN architectural background. This document supersedes the previous deliverable, D3.1, which described the initial design. The deliverable D3.4 will build on this deliverable to describe the final set of application functions for the RIFE system.
REFERENCES

doi: 10.1049/ip-com:20050237
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1561992&isnumber=33164

doi: 10.1109/IIH-MSP.2006.264994
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4041714&isnumber=4041646

[Qualcomm] Qualcomm website article “why raptor codes are better than reed-solomon codes for streaming applications” Mobile Computing and Communications Review, Volume 18, Number 1, January 2014


doi: 10.1109/TIT.2006.874390
URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1638543&isnumber=34354


[Tapolcai et al., 2015] Tapolcai, J., Biro, J., Babarci, P., Gulyas, A., Heszberger, Z., and Trossen, D.


