Internet of Things: Least Interference Beaconing Algorithms

Emmanuel Tuyishimire (tuyishimire@aims.ac.za)

Thesis presented for the degree of Master of Science

In the Department of Computer Science

Supervised by:
Prof. Bigomokero Antoine Bagula and
Prof. J. W. Sanders

November 21, 2014

Abstract

The emerging sensor networking applications are predicting the deployment of sensor devices in thousands of computing elements into multi-technology and multi-protocol platforms. Access to information will be available not only anytime and anywhere, but also using anything in a first-mile of the Internet referred to as the internet-of-things (IoT).

The management of such a large-scale and heterogeneous network, would benefit from some of the traditional IP-based network management techniques such as load and energy balancing, which can be re-factored to achieve efficient routing of sensor network traffic.

Research has shown that minimizing the path interference on nodes was necessary to improve traffic engineering in connection oriented networks. The same principle has been applied in past research in the context of the IoT to reveal that the least interference beaconing protocol (LIBP); a protocol derived from the least interference beaconing algorithm (LIBA) outperforms the Collection Tree Protocol (CTP) and Tiny OS Beaconing (ToB) protocol, in terms of energy efficiency and lifetime of the sensor network. However for the purpose of efficiency and accuracy, it is relevant, useful and critical to revisit or re-examine the LIBA algorithm in terms of correctness and investigate potential avenues for improvement.

The main contributions of this research work are threefold. Firstly, we build upon formal methods to verify the correctness of the main principles underlying the LIBA, in terms of energy efficiency and interference minimization. The interference is here defined at each node by the number of routing paths carrying the sensor readings from the motes to the sink of the network that traverse the node. Our findings reveal the limitations in LIBA. Secondly, building upon these limitations, we propose two improvements to the algorithm: an algorithm called LIBA+ that improves the algorithm performance by keeping track of the energy usage of the sensor nodes, and a multi-sink version of the algorithm called LIBAMN that extends the algorithm to account for multiple sinks or gateways. These enhancements present preventive mechanisms to include in IoT platforms in order to improve traffic engineering, the security of network protocols and network stability. Lastly, we present analytical results, which reveal that the LIBA algorithm can be improved by more than 84% in terms of energy balancing. These results reveal that formal methods remain essential in the evaluation and performance improvement of wireless sensor network algorithms and protocols.

Plagiarism declaration

I know the meaning of Plagiarism and declare that all of the work in the document, save for that which is properly acknowledged, is my own.

SIGNATURE: [illegible]

DATE: November 21, 2014

Contents

List of Figures

List of Tables

1. Introduction

1.1 Motivation

Integration of Radio-Frequency Identification (RFID) and sensor technologies are emerging as an important component of first mile connectivity of Internet called the Internet of Things (IoT) that enables information to be accessed not only at any time and anywhere but also using anyone to access anything.

A typical IoT deployment scenario consists of a proactive monitoring system. here a network of RFID tags is attached to objects, and a set of readers integrated into sensor motes. These are used as an ubiquitous sensor network (USN) [2] collecting information on identification and environmental parameters of these objects, and transmitting to a gateway where the information is processed and different services derived from this information are delivered to users. This may be applied in many fields including healthcare, environment monitoring and protection, smart cities, public safety and precision agriculture.

Emergence of the IoT has raised a need for new communication protocols. It has also necessitated a redesign of some of the traditional protocols in order to achieve efficient routing of information over islands of interconnected lightweight networks. The aim of which is to support human-to-human, machine-to-machine, and machine-to-human communications. Such protocols require a very high level of reliability and trust as failure to achieve their assigned task may lead to risks that may result in human loss. This could be seen when a train derails because the IoT system has failed to deliver the right signal to the right node at the right time during machine-to-machine interaction or when fire-fighters are misled in a rescue operation during machine-to-human interaction as a result of a faulty IoT visualization algorithm directing them towards an empty room while people who are to be rescued are in another room.

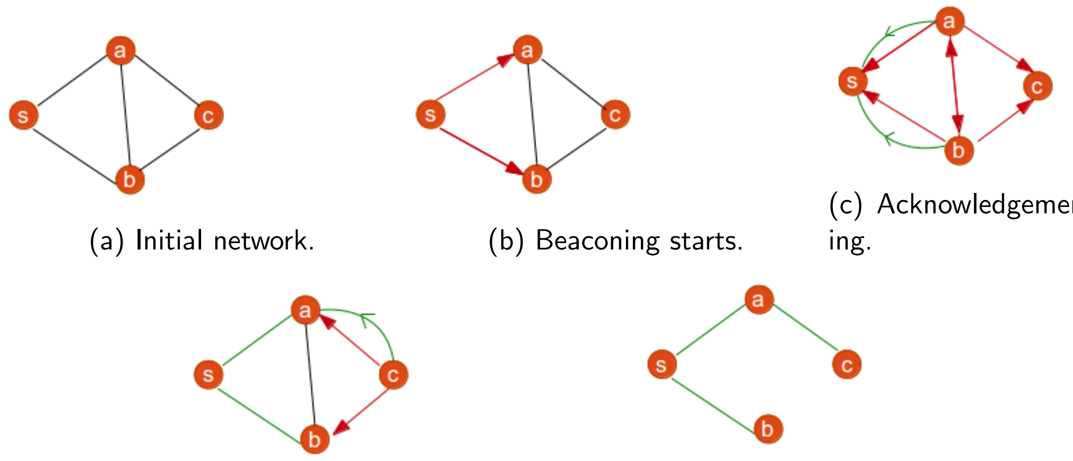
Reliability and trust enhancements may be obtained through protocol verification by using formal methods such as process algebra [15] or Z-notation [38, 14, 36, 1] to prove the correctness of some of the fundamental properties of protocols and/or their underlying algorithms. Formal method can also be used to discover hidden properties or errors that have never been unearthed. A body of research work on formal verification of network protocols has up until now been more focused on security issues. The correctness of network algorithms and the choice of an appropriate formalism for different types of protocols are two issues which need to be investigated more thoroughly by the research community.

1.2 Previous works

The Least Interference Beaconing Algorithm (LIBA) is a recently proposed routing algorithm for improving the resource sharing in a network (it is detailed in [3, 7]).

The LIBA was proposed in the context of the Internet-of-Things (IoT) to minimise the path interference on nodes which the expectation of achieving energy savings through load balancing. It exploit the interference measure which is considered as the network epidemic. Further more, the algorithm assumes a single-sink network and could be extended to support multi-sink networks. The LIBA protocol is called LIBP (Least Interference Beaconing Protocol) and consists of building a routing tree from an initial graph of the network as illustrated by Figure 1.1.

1.2.1 Basic protocol. We refer to the protocol as stated in [7] and describe the LIBA using Figure 1.1.



(a) Initial network.

(b) Beaconing starts.

(c) Acknowledgement and relaying.

(d) Acknowledgement and relaying.

(e) Resulting tree.

Figure 1.1: Simple routing.

Consider the initial network in Figure 1.1a, where node s is the sink. As shown in Figure 1.1b, the sink starts by broadcasting a beaconing message (a beacon) in the network.

Figure 1.1c shows that each node receiving the beacon, relays it in the network as well as acknowledging the chosen parent in the route from the node to the sink. Note that all neighbours of the sink have a unique parent choice which is the sink. Figure 1.1d shows that the processes of relaying and acknowledging continue to the remaining nodes and no node acknowledging or relaying a message more than once.

Notice that each parent receives one acknowledgement from each of its children if selected by the child as the minimally weighted parent. This is why the number of received acknowledgement messages can be refereed to as the weight of a node (The weight equals zero if and only if the node did not receive any acknowledgement message).

In this section we discuss the previous works on LIBA-related routing protocols, and present network epidemic models and routing in multi-sink networks, which can be used for the stability, performance and improvement of LIBA algorithms.

1.2.2 Routing protocols. The Collection Tree Protocol (CTP) described in [17] is a routing protocol which consists of sending periodic routing messages in a network to find paths from each node of the network to a node which initiated the routing. [19] shows that the main CTP focusses are reliability, robustness, efficiency and hardware independence. Simulation shows that this is achieved by using a collection tree and adaptive beaconing features described in [17]. However, the algorithm is challenged by the link dynamics and route inconsistencies such as loop creation.

Tiny OS Beaconing (TOB) is described in [23] as a protocol with simplified data structure. It is a simple node structure. This is because in TOB each node keeps in its routing table information of only a parent which it will use as the next neighbour for the traffic routed from the node to the base station. This makes the used routing tables simpler than in CTP where nodes keep information about a whole path to the sink. However, TOB raises issues including lack of resilience to node failures and also the tree-like many-to-one dissemination model can cause uneven power consumption across the network, and the potential of a big sub-tree being removed from the network in case of the failure of a single node.

As presented in in [3, 7], $LIBP$ is a recently proposed protocol and $LIBA$ its underlying algorithm. while the CTP and TOB algorithms may lead to uneven power consumption, the $LIBA$ has been proposed as a routing algorithm that uses simplicity to enable scalability of Ubiquitous Sensor Networks (USN) It also uses a beaconing process that supports load balancing to improve energy efficiency. Note here that as summarised in Table 1.1, the algorithm and protocols reviewed in this section are for static networks with a single sink.

Table 1.1 summarises the key comparison features between LIBA/LIBP and the other routing protocols.

Algorithm /Protocol	Assumed network	Main characteristics	Method	Strength	weakness
CTP [17]	Mesh network with a single sink	Reliability, robustness, efficiency, hardware independence	Adaptive beaconing and Data path validation	High reliability	Link dynamics, route inconsistency, large routing tables and messages
TOB [23]	Mesh network with a single sink	Small size messages and routing tables	keep track of only parents and children	Quick communication	Uneven power consumption and lack of resilience
LIBA/LIBP [7, 3]	Mesh network with a single sink	Periodic beaconing and nodes weights redistribution	Beaconing and acknowledgement messages	Quick communication, resource sharing, simple routing table and messages, load balancing	Unverified and improvable algorithm

Table 1.1: Protocols comparison.

1.2.3 Network epidemic models. Epidemic models started with the aim of describing the interaction between susceptible, infected and recovered (or removed) people when a disease enters a given population. Such models were known as SIR models and were pioneered from 1927, [28]. From this time, the initial SIR-model has been continuously adopted to solve various problems, including network problems.

In many cases [37, 31, 26], nodes of a network are grouped in disjoint compartments and the resulting model is known as the epidemiological compartment model. Analysis of the models is usefully done by using the basic reproduction number R_0 or its approximate [22, 11]. Further details on the basic reproduction number are discussed in Section 2.2.1.

It is assumed that removing a node x can not cause removal of some other nodes connected to x. On the other hand however, in various networks removal (death) of a node predicts or may imply disconnection of a network which causes death of other nodes. We have not yet seen a model which takes into account the fact that the removed node could cause other nodes (including non infected ones) to leave the network. Table 1.2 compares different approaches of epidemic modelling.

Model type	Assumed population	Domain	Role	Strength	Weakness
Susceptible-Infected-Removed/Recovered(SIR) [28]	People	Mathematical Biology	Model of reference	Initial model	Represents a very ideal system and recovered and Removed states have the same behaviour
Susceptible-Infected-Recovered-Susceptible ($SIRS$)[37]	Social Network nodes	Communication	reference	Recent general compartmental model and stochastic model	No death impact is considered and uncontrolled transitions
Epidemic routing models [42]	Connected computers	Routing	Modelling the routing issues	Support mobile network	No death impact is considered and nodes are assumed to be the same
Electronic-Susceptible-Infected-Recovered-Susceptible ($E - SIRS$) [31]	Connected computers	Network security	Model for Worms and virus	Compartmental model	No death impact is considered
Susceptible-Infected-RemovedSIR [26]	People	Mathematical Biology	Analysis of susceptibility per group of people	Compartmental model	No death impact is considered sensors behave likely on the attacks
Susceptible-Infected-Recovered-Maintenance ($SIR - M$) [39]	Connected sensors	Epidemiology	SIR with maintenance analysis	Network flexibility analysis	No death impact is considered
Basic reproduction number(R_0) [22]	N/A	Modelling and analysis	Stability analysis methods	Clarifies the stability conditions	N/A
Next generation matrix (K) [11]	N/A	Modelling and analysis	Calculation of R_0	Precise way to calculate R_0	N/A

Table 1.2: Models comparison.

1.2.4 Routing in Multi-sink networks . One of the main properties of good routing algorithms is that the energy resources of the nodes in a network are efficiently managed. In addition, large scale networks, especially sensor networks, need to be deployed with many sinks to increase energy efficiency by reducing energy dissipation.

The multiple-sink-networks design explained in [33] would help to manage the network. The support of the mobility of sensor nodes, [18] proposed a dynamic approach to extend the life time of sensor nodes in terms of energy. The Dijkstras algorithm is used to compute the shortest path in a network weighted using the sum of eight metric of any two connected nodes.

However, there is a possibility that a node would unnecessarily be part of many paths which could cause inefficient routing. This has been addressed in [32] by minimising the popularity of nodes. On the other hand, the $LIBA$ for one-sink networks has been implemented in [7, 3] where the multi-sink networks case is one of the steps forward. Other multisink routing protocols are proposed in [35, 12, 16] where focus was put on QoS parameters and mobility and do not necessarily ensure the efficient minimization of interference in a network. In all these approaches, numerical results are used to measure their efficiencies. This may cause unpredicted inefficiency and hence their formal verification remains essential.

1.3 Contribution

Research has shown that minimizing the path interference on nodes was necessary to improve traffic engineering in connection oriented networks [5, 6, 4]. The same principle has been applied in past research in the context of the IoT to reveal that the least interference beaconing protocol (LIBP) [3, 7]; a protocol derived from the least interference beaconing algorithm (LIBA), outperforms the Collection Tree Protocol (CTP) and Tiny OS Beaconing (ToB) protocol in terms of energy efficiency and lifetime of the sensor network. However for the purpose of efficiency and accuracy, it is relevant, useful and critical to revisit or re-examine the LIBA algorithm in terms of correctness and investigate potential avenues for improvement.

The main contribution of this thesis is threefold.

1. **Correctness:** In this work we use formal method to formalise the description of a recently proposed routing algorithm: the Least Interference Beaconing Algorithm (LIBA). Properties including its correctness are formally verified.

2. **LIBA enhancement:** Formally specified and verified LIBA enhancement algorithm is provided and it is called LIBA$^+$. Furthermore, The existing LIBA version assumes a network with a single sink. In adopting the LIBA$^+$ formalisation, we propose the Least Interference Beaconing Algorithm for Multi-sink Networks LIBAMN. The algorithm is presented formally and its properties such as correctness are proved.

3. **Stability:** We provide and analyse a new epidemic model for a network using LIBA$^+$ and this enables us to study network stability. The model supports the fact that the death of nodes may causes the death of others. To achieve this, nodes have been grouped in groups we call Interference sets. This set the precedent to define a new mathematical structure of nodes in a network, where the work-related theorems have been proved.

1.4 Thesis organisation

After introducing the work in this chapter (1), we explain useful concepts for this work in Chapter 2. Mathematically, we formalise and verify the Least Interference Beaconing Algorithm in its current version Chapter 3. The algorithm weakness is explained in Chapter 4 and a newly improved algorithm is provided and its correctness is proved. An analytical study of interference transmission in a network using the improved algorithm is done in Chapter 5. In Chapter 6 we further extend the algorithm to correctly support multi-sink networks. The general conclusion and future work in Chapter 8 are drawn from the results of this work. Extended results are presented in appendix (See Chapter 9)

1.4.1 Chapters dependency. We use Figure 1.2 (the network) to show the dependency of chapters in this work where for instance Chapter 2 is referred to as just Chap 2.

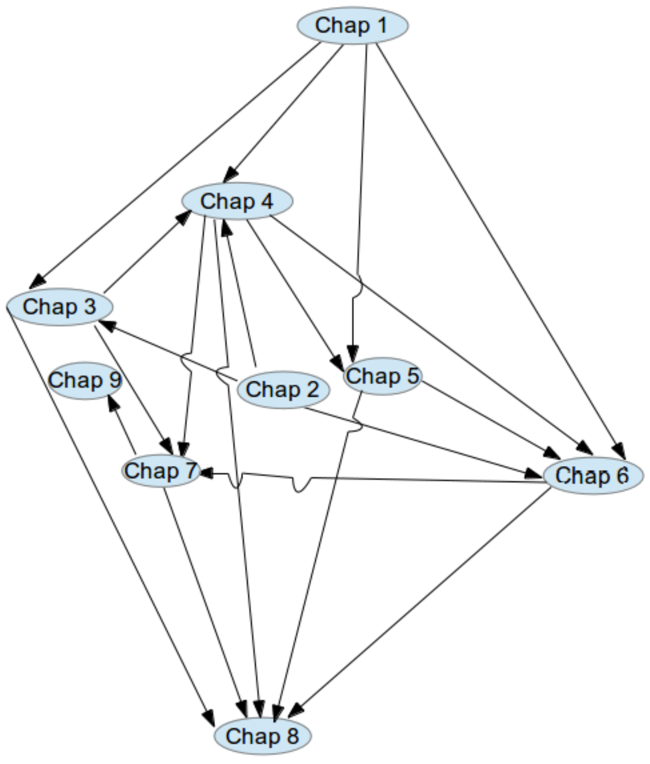


Figure 1.2: Chapters dependency.

As depicted by Figure 1.2, the arrows pointing to a node n come from the prerequisite chapters to understand the one shown by the node n. A tail of each arrow can be refereed to as a pre-chapter. For instance the pre-chapters of Chapter 5 are Chapters 1 and 2. Notice that there is no chapter explicitly using conclusion and future work (8) and appendix (9) as pre-chapters.

2. Background

In this chapter, we discuss the concepts useful for this work. We include definitions, notations and methods which are needed for our studies.

2.1 Set theory: definitions and notations

For this work, we refer to [40] to define useful set theory terms.

2.1.1 Set. A set is a collection of different objects, no matter what order in which they are collected. the object o in set A (expressed by $o \in A$ or $o : A$ for Z notation as described in Section 2.4) is called the **element** of S. A set with no element is called the **empty set** and it is denoted by $\varnothing$. A set with only one element is called the **singleton** while a set consisting of two elements is called the **pair**.

2.1.2 Subset. The set S is a subset of set A (this is denoted by $S \subset A$ or $A \supset S$) if all elements of S are also elements of A. The set of all subsets of a set A is denoted by $\mathcal{P}(A)$ or $\mathbb{P}A$ and it is called the **power set** of A.

2.1.3 Equal and different sets. Two sets A and B are said to be **equal** if they contain each other and otherwise they are **different**. That is

$$A = B \Leftrightarrow A \subset B \wedge B \subset A.$$

2.1.4 Proper subset. It is a non empty subset of a set say S which is different from S.

2.1.5 Maximal subset. It is a proper subset of a set say S which is not included in any other proper subsets of S. This means that if the set A is a maximal subset of S then for any subset B of S we have

$$A \subset B \Rightarrow A = B.$$

2.1.6 Set Calculus.

- **Set cardinality $\#$:** It is the number of elements in a set. Given the set A, its cardinality is denoted by $\#A$. For instance $\#\{1, 2, a, b, c\} = 5$.
- **Union $\cup$:** The union of two sets A and B is a set denoted by $A \cup B$, of the elements belonging to A or B. That is $A \cup B = \{x \mid x \in A \vee x \in B\}$.
- **Intersection $\cap$:** The intersection of two sets A and B is a set denoted by $A \cap B$ containing the elements belonging to both A and B. That is $A \cap B = \{x \mid x \in A \wedge x \in B\}$.
- **Difference $\setminus$:** The difference of two sets A and B is a set denoted by $A \setminus B$ containing the elements belonging to A but not in B. That is $A \setminus B = \{x \mid x \in A \wedge x \notin B\}$.

2.1.7 Set partition. A partition P of a set A is a set of non empty mutually disjoint subsets of A such that [1] the union of all elements in P is equal to the set A. That is

$$P : \mathbb{P}A \wedge \forall x, y : P \bullet x \cap y = \varnothing \wedge \bigcup_{z:P} z = A.$$

The sets in P are called the **blocks**, **parts** or **cells** of the partition and in this work are refereed to as **compartments**.

[1] In this document, the term "such that" is denoted by the symbol "•" or "—"

2.2 Epidemiology

The term "epidemiology" has been used first to describe the study of epidemics in 1802 by the Spanish physician Villalba. Epidemiology consists of description and causation of diseases in general, and many non-disease health-related conditions such as high blood pressure and obesity. Therefore, Epidemiology is based upon how the patterns of a disease causes changes in function of everyone.

One of the main studies in Epidemiology is stability of a system at its Disease Free Equilibrium (Equilibrium point with no infection) denoted by DFE. Many approaches have been used. However, as mentioned in [22], the most appreciated is the use of a number called **basic reproduction number** which is denoted by R_0.

2.2.1 Basic reproduction number R_0 . Generally speaking, R_0 is "*the expected number of secondary individuals produced by an individual in its lifetime*" [22]. In Epidemiology, it is the expected number of infected individuals by a single infected one when interacting with susceptible people.

- If $R_0 < 1$, then the expected number of new infections is less than one. Hence there is no new infection and the system is unstable at the corresponding DFE.
- if $R_0 > 1$, then new infections are expected to occur and the system is stable at corresponding DFE.

Many approaches to compute the number R_0 are discussed and compared in [22], where the most appreciated is the **next generation method**.

2.2.2 Next generation method. This is the method which is widely used to compute the basic reproduction number (discussed in Section 2.2.1). It consists of computing the next generation matrix denoted by K and the basic reproduction number is its trace ($R_0 = Trace(K)$), as proposed in [10].

Construction of the Next Generation Matrix K: As described by [11], the next generation matrix K is defined as the matrix which relates the number of newly infected individuals in the different categories in consecutive generation.

Given a system of Difference or Ordinary Differential Equations ($ODEs$) describing the change with time for a population, the matrix K is determined by a subsystem describing the production of new infections and the change in infection among individuals, which is called **infected subsystem**.

Using the Jacobian at DFE, an infected subsystem is linearised and the linearised subsystem is the starting point of the computation of the matrix K.

In fact, as done in [37], the infection subsystem can be described as the sum of two other subsystems namely $\mathcal{F}$ and $\mathcal{V}$, where $\mathcal{F}$ consists of **transmission** part, describing the production of new infection in the system, and $\mathcal{V}$ consists of **transition** part, describing the change in states.

Note that the linearisation of both $\mathcal{F}$ and $\mathcal{V}$ requires two Jacobian matrices F and Vrespectively, whose sum is the Jacobian of the infection subsystem. The next generation matrix is obtained by taking the product of the matrix F and the inverse of matrix V. That is

$$K = FV^{-1}$$

Various examples of finding the next generation matrix can be found in [11].

2.3 Graph theory

2.3.1 Network. We assume the concepts from Graph Theory; see [9] and [27]. Any routed, connected and undirected graph G is called **a network**. The edges in a network are called **links** and vertices are called **nodes**.

A network G_1 is a **sub-network** of G if and only if G has all links and nodes of G_1; in this case we say that G_1 is an **induced sub-network** of G if it is obtained by removing some nodes and all their corresponding edges from the parent network. A network that associates a real number with every edge or every node in the graph is called **edge-weighted network** or **node-weighted network** respectively.

We represent a network as $G(L, N, W, s)$ where, G is an identity of a network, L is a set of its links, N is a set of its nodes, W is a set of all the node weights and $s : N$ is its sink.

2.3.2 Hops. Given a network $G(L, N, W, s)$ as described in 2.3.1, the number of links between in the path connecting two nodes is called the hops. The minimum number of hops between two nodes n_1 and n_2 is denoted by $d(n_1, n_2)$ and in this work, it is refereed to as the distance between n_1 and n_2.

Note that,

$d(n_1, n_2) \geq 0$

$d(n_1, n_2) = 0$ if and only if $n_1 = n_2$

$d(n_1, n_2) = d(n_2, n_1)$ (symmetry)

$d(n_1, n_3) \leq d(n_1, n_2) + d(n_2, n_3)$ (triangle inequality).

So the set N of nodes of a network is a **metric space** as it is defined in [29].

For simplicity we denote the distance between node n and a sink of a network by $d(n)$.

2.4 Introduction to Z notation

The Z notation [38] was developed, largely at Oxford in the early to mid $1980's$ to clarify the description of complex discrete transition systems (the analogue case is dealt with satisfactorily by differential equations). It views a transitions system, like $LIBA$, as a state-based system with operations on it and this provides techniques for the structured description of state and for the structured description of the operations on state (with input and output). Here we give a summary of the notation, referring to [38] for details.

The state S of a discrete system consists of several typed observables, vector $v : V$, subject to some invariants P (where P is a predicate with free variables v). This is expressed by schema:

```
┌ S ──────────────────────
│ v : V
├──────
│ P
└─────────────────────────
```

For example a system $Even$ whose state consists of just an even integer n is written

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Even}} \\ \hline
n : \mathbb{Z} \\ \hline
\exists\, m : \mathbb{Z} \bullet n = 2m \\ \hline
\end{array}$$

In writing P, conjuncts are written on separate lines, with $\wedge$ omitted.

Schemas may be nested by using a schema as a type V. An observable $v : V$ regarded as input is written $v? : V$ and one regarded as output is written as $v! : V$ (notation inherited from process algebra). State-before is written s and state-after written s' for each observable s.

Each system operation Op takes a state $s : S$ before and an input $\mathit{in}? : \mathit{In}$ and returns a state $s' : S$ after and an output $\mathit{out}! : \mathit{Out}$, subject to the invariant property $Q(s, s', \mathit{in}?, \mathit{out}!)$:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Op}} \\ \hline
s, s' : S \\
\mathit{in}? : \mathit{In} \\
\mathit{out}! : \mathit{Out} \\ \hline
Q \\ \hline
\end{array}$$

For convenience ΔS is defined to be

$$\begin{array}{|l}
\multicolumn{1}{l}{\Delta S} \\ \hline
S \\
S' \\ \hline
\end{array}$$

Note that inclusion of ΔS brings into scope the observables s and s' of type S, for use in predicate Q. Both s and s' are automatically constrained by the predicate P in S, which does not need to be repeated. In Z, a variable not explicitly constrained is assumed to take an arbitrary value (of its type). We therefore find that it is convenient to use the variation, due to object Z [36], $\Delta(vs)$ where state variables not in the vector vs of variables remain unchanged. Thus with the type Even for a pair of distinct even numbers,

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{Evens}} \\ \hline
a, b : \mathit{Even} \\ \hline
a \neq b \\ \hline
\end{array}$$

we have

$$\begin{array}{|l}
\multicolumn{1}{l}{\Delta Evens} \\ \hline
Evens \\
\\
Evens' \\ \hline
\end{array}
\quad = \quad
\begin{array}{|l}
\multicolumn{1}{l}{\Delta Even} \\ \hline
a, a', b, b' : Even \\ \hline
a \neq b \\
a' \neq b' \\ \hline
\end{array}$$

whilst

$$\begin{array}{|l}
\multicolumn{1}{l}{Evens(a)} \\ \hline
\Delta Even \\ \hline
b = b' \\ \hline
\end{array}$$

Projection notation is inherited from that for records in programming. If $x : Even$ then $x.a$ and $x.b$ denote the two components of x. Note that no ordering implied between the variables a and b of $Evens$. For example an operation which inputs an even integer, adds it to the state and outputs the result, is written

$$\begin{array}{|l}
\multicolumn{1}{l}{Eop} \\ \hline
\Delta Even \\
in?, out! : \mathbb{Z} \\ \hline
in? : Even \\
n' = out! = n + in? \\ \hline
\end{array}$$

State is initialised by further constrained state to satisfy the initialization property; for example.

$$\begin{array}{|l}
\multicolumn{1}{l}{Init} \\ \hline
Even \\ \hline
n = 0 \\ \hline
\end{array}$$

Finally all ingredients are combined to produce a class consisting of state, initial state and operations. For example functions examples appear in the remainder of the report.

Eve

Even

$n : \mathbb{Z}$

$\exists\, m : \mathbb{Z} \bullet n = 2m$

Init

Even

$n = 0$

Eop

$\Delta \mathit{Even}$

$\mathit{in}?, \mathit{out}! : \mathbb{Z}$

$\mathit{in}? : \mathit{Even}$

$n' = \mathit{out}! = n + \mathit{in}?$

Figure 2.1: The class *Eve*.

2.5 Computation tree logic (CTL)

CTL is a temporal logic, with connectives that enable us to describe time-varying state without mentioning time explicitly. It models time as a tree of executions, extending infinitely into the future. Each execution consists of a sequence of states, here called a path. Since in most cases the future is not fully determined, several paths are considered, any one of which might be realised and hence a tree arises.

As it is detailed in [25], CTL uses quantifiers A and E, and the temporal operators G, F, X, and U. The (state) formula Aφ is satisfied in a state if all paths starting in that state satisfy φ, while Eφ is satisfied if some path satisfies φ. Note here that an atomic formula do need to be prefixed by the quantifier.

The (path) formulas Gφ, Fφ and X φ mean that φ holds globally in all states, in some state, and in the next state of a path, respectively. The formula φUψ means that eventually ψ will hold and, until a state occurs along the path that satisfies ψ, property φ has to holds.

Logical connectives such as conjunction ($\wedge$), disjunction ($\vee$), implication ($\Rightarrow$) and negation ($\neg$) are used to connect formulae. Here are two (linear) examples:

1. Eventually the variable x becomes 3 and stays 3 is written $E(A(x = 3))$.
2. Infinitely often x takes the value 3 is written $A(E(x = 3))$.

Note here that $A(E(x = 3)) \Rightarrow E(A(x = 3))$.

2.6 Routing

Routing is a process of selecting a path in a network along which to send a message. When a network is static, a transmitting node must first find a path for each message it wishes to send. Many routing protocols have been implemented to handle this issue [30, 41]. For the case of our work, we define the key words of routing as follows.

2.6.1 Sequence number. This is the measure of the freshness of information. The bigger the sequence number the fresher the corresponding information.

2.6.2 Routing table. It is a table storing information about a path to a destination.

2.6.3 Ways of communication. We define the following ways in which nodes communicate:

1. **Uni-cast:** A message is sent from a sender to one receiver.
2. **Group cast:** A piece of information is sent from one or more nodes to a predefined group of nodes in the network.
3. **Multicast:** A message is sent from one or more nodes to a set of other nodes in the network defined just for the message.
4. **Broadcast:** A message is sent from one sender to all its neighbours.
5. **Cast:** A messages is sent by any of the above ways.

Note that in this work, all messages are cast by either broadcasting or uni-casting.

3. Least Interference Beaconing Algorithm (LIBA)

Section 1.2.2 revealed LIBA [7] to be an algorithm which might be preferable in terms of energy efficiency. But currently, it has no mathematical formalisation and so there is no proof of its correctness. Its use requires trust, and hence proof of its correctness remains essential.

In this chapter we formalise the problem and explain LIBA in its current state. Furthermore, we define the structure of observables which helps us to formalise LIBA mathematically, using the Z notation as described in Section 2.4. We compare our problem with other routing problems and this enables us to study correctness of one run of the algorithm.

3.1 Routing problem

We describe a routing problem which consists of finding the shortest path from each node of a network to a specific node so as to minimise interference of paths at each node in the network. The routing is performed periodically to ensure resource sharing and hence efficient traffic engineering in IoT settings.

We assume an m-to-1 routing process where sensor readings are collected by each of m-to-1 nodes of a sensor network and routed to a unique sink connecting a gateway to the sensor network.

We define **path interference** of a node as the number of its children in the routing tree rooted at the sink of the sensor network. The path interference is also referred to as the weight of the nodes which, in case there is no established routing, is assumed to be equal to zero for all nodes.

It is assumed that the routing process is mapped into a spanning tree construction where a breadth-first algorithm is used to build routing tables enabling sensor nodes to identify the next hop to the gateway as parents in the spanning tree rooted at the sink of the sensor network.

Assume a network $G(L, N, W, s)$ as described in Section 2.3.1.

The problem consists of finding a network $G'(L, N, W', s)$, whose nodes and link addresses are the same as in G and W' is the set of node weights such that, if i is a node in the network G of weight w_i, it computes the subset of its neighbours $nbr(i)$ so as its weight $w'_i \in W'$ is the solution of the following optimization problem:

$$min \quad w'_i = w_i + \#\{j : nbr(i) \mid d(j) > d(i)\} \tag{3.1.1}$$

where, $d(j)$ and $d(i)$ are distances as explained in Section 2.3.2 (see a last sentence of the section).

Note that the network G' can be used to further compute network G'' to solve the above problem, and generally, the network $G^{(n)}$ can inductively be used to compute $G^{(n+1)}$. Furthermore, since each node i computes its children $nbr(i)$, a corresponding routing tree can be extracted.

3.2 Existing solution

The current approach for the solution of the problem stated in Section 3.1, is the distributed algorithm called Least Interference Beaconing Algorithm (LIBA). Simulations in [7] suggest that the algorithm is preferable compared with TOB [23] and CTP [19]. The basic protocol of LIBA has been introduced in Subsection 1.2.1.

3.3 Data structure

To identify the possible observables for the algorithms, we refer to the structure of a network, the way messages are sent in the network and the way nodes keep information while the algorithm is processing. We mostly define their structure using Z schemas which are translated and explained.

3.3.1 Node address. The set of all node identifiers (i.e. addresses) is denoted by IP. We extend the set IP to

$$IP_{-} = IP \cup \{-\}$$

where, the symbol "$-$" stands for a non-existing or undefined element. Giving a value a non-existing element helps us to model a routing table of each node in the network as shown in the next subsection (3.3.2).

3.3.2 Routing table R. This is a table of a node storing information about the network. A routing table of a node contains information about the path to the sink including the node parent pip, its current set of children cip and its sequence number sn which is chosen to be a positive integer showing whether the routing table is updated or not.

Note that the condition $pip : IP_{-}$ expresses the fact that a parent of a node does not necessarily exist. Some nodes do not have parents. This scenario will be clarified in Subsection 3.3.4.

$$\begin{array}{|l} R \\ \hline pip : IP_{-} \\ cip : \mathbb{P} IP \\ sn : \mathbb{N} \\ \hline pip \notin cip \\ \hline \end{array}$$

A parent has an identifier different from that of each of its children ($pip \notin cip$).

Updating routing tables:

There are three ways to update a routing table:

1. **Updating the children lists:** A list of children is updated by making it empty or adding a child to the existing list of children.

2. **Updating the sequence number:** The sequence number of a routing table is changed to a new one by replacing it with a new one.

3. **Changing a parent chp:** This is done by first removing the existing element by the operation $remp$ and then adding a new one by the function adp. That is

$$chp(rt, nip) = adp(remp(rt), nip)$$

where the two operations are described:

$$\begin{array}{|l}
\multicolumn{1}{l}{\textit{remp}[R]} \\
rt?, r! : R \\
\hline
r!.pip = - \\
rt?.cip = r!.cip \\
rt?.sn = r!.sn \\
\hline
\end{array}
\qquad
\begin{array}{|l}
\multicolumn{1}{l}{\textit{adp}[R, IP]} \\
rt?, r! : R \\
n? : IP \\
\hline
rt?.pip = - \\
r!.pip = n? \\
rt?.cip = r!.cip \\
rt?.sn = r!.sn \\
\hline
\end{array}$$

The schema $remp$ shows that the operation $remp$ takes the routing table $rt?$ and returns a new routing table $r!$ obtained by making the parent in $rt?$ a non-existent element but all other entries of $rt?$ are the same as those of $r!$.

The schema adp shows that the operation adp takes a routing table with no parent in it and then adds a parent $n?$ to replace $-$, where all other entries are kept unchanged.

3.3.3 Node N**.** This is described by its routing table rt, its address ip and its weight wip. The weight of a node wip is a natural number which represents the latest number of children a node has.

$$\begin{array}{|l}
\multicolumn{1}{l}{N[IP, R]} \\
rt : R \\
ip : IP \\
wip : \mathbb{N} \\
\hline
ip \notin \{rt.pip\} \cup rt.cip \\
\hline
\end{array}$$

A node's address is always different from that of its parent or its children $(ip \notin \{rt.pip\} \cup rt.cip)$.

3.3.4 Network NT**.** This is determined by its nodes, links and sink. Note here that the weight distribution in the network can be obtained from the nodes distribution since the descriptors of a node include its weight. A network is described using the following schema:

$$\begin{array}{|l}
\multicolumn{1}{l}{NT[N]} \\
nodes : \mathbb{P}N \\
links : N \leftrightarrow N \\
nbr : N \rightarrow \mathbb{P}N \\
s : N \\
\hline
links \in nodes \leftrightarrow nodes \\
ran(link^{*}) = nodes \\
links^{\sim} = links \\
id_{nodes} \cap linkd = \varnothing \\
nbr(n) = links(\!|\ n\ |\!) \\
s \in nodes \\
s.pip = - \\
d(s) = 0 \\
\forall\, n_1, n_2 : nodes \bullet n_1.ip \neq n_2.ip \\
\hline
\end{array}$$

As shown in the schema NT, a network of type NT is defined as a connected and bi-directed graph with no self link. The neighbourhood nbr of a node n consists of nodes connected to n. The sink s is one of the nodes in the network at distance equal to zero. Finally, any two nodes of the network have different identifiers.

3.3.5 Beacons BC**.** This is a message broadcast periodically from a sink, and relayed in a network to enable all nodes of the network to select their (best) parents. A beacon contains the address of its sender sip, the sequence number of the beacon bsn (a natural number which identifies a beacon), the hop count hop which helps to determine the nodes ready to be parents (nodes at distance minimal to the sink) and the weight sw of the sender. The set of all beacons is denoted by BC. The schema describing a beacon is as follows:

$$\begin{array}{|l}
\hline
BC[IP] \\
sip : IP \\
bsn, hop, sw : \mathbb{N} \\
\hline
\end{array}$$

3.3.6 Acknowledgement message AC**.** It is a message sent by a node to only its newly selected parent to confirm the selection. It contains its sender sip and the destination dip of the message. The set of all acknowledgement messages is denoted by AC. The following is a schema for an acknowledgement message.

$$\begin{array}{|l}
\hline
AC[IP] \\
sip, dip : IP \\
\hline
sip \neq dip \\
\hline
\end{array}$$

The sender of the acknowledgement message is different from the destination of the message $(sip \neq dip)$.

3.3.7 Message MSG**.** In all $LIBA$ processes, a message is either a beacon or an acknowledgement. We describe the set of all messages MSG as the following schema.

$$\begin{array}{|l}
\hline
MSG \\
bc : \mathbb{P}BC \\
ac : \mathbb{P}AC \\
\hline
\#ac \leq \#bc \\
\hline
\end{array}$$

Since a beacon is broadcast and an acknowledgement message is unicast, the number of received beacons is always at least the number of received acknowledgements $(\#ac \leq \#bc)$. This condition is also an agreement with the protocol requirement that upon reception of beacons from different potential parents, a node selects only one parent by sending an acknowledgement message to that parent.

3.4 Beaconing

In this work, the word beaconing refers to LIBA routing mechanisms. The beaconing process starts from the sink and goes to all nodes in the same network as the sink. We describe the beaconing process using three Z-classes namely, **Stater**, **Transmitter** and **LIBA**. In this section we study the classes and finally we provide an example showing how the classes cooperate for routing.

3.4.1 Class Starter. This consists of a sink node as its object and the operation $Initiate$ as its method. The sink $sink$ of the network is considered to be constant and it is the same as that in the network nt on which the operation $Initiate$ is used.

The operation consists of initiating the beaconing message $bc!$, and the routing table rt of the sink. In an initiated beaconing message, the sender $bc!.sip$ is the sink's address $sink.ip$, the number of hops

$bc!.hop$ is set to zero and the beacon's weight $bc!.sw$ is the weight $sink.wip$ available in the routing table of the sink.

The set $sink.rt.cip$ of children in the sink's routing table is emptied and the sequence number in the routing table is incremented by one.

The initiated beacon has the same sequence number as in the routing table $sink.rt.sn'$.

$$
\begin{array}{|l}
\textit{Starter} \\
\hline
\begin{array}{|l}
nt : NT \\
sink : N \\
\hline
sink = nt.s
\end{array} \\
\\
\begin{array}{|l}
\textit{Initiate} \\
\hline
\Delta(sink.rt.cip, sink.rt.sn) \\
bc! : BC \\
\hline
bc!.sip = sink.ip \\
bc!.hop = 0 \\
bc!.sw = sink.wip \\
sink.rt.cip' = \varnothing \\
sink.rt.sn' = sink.rt.sn + 1 = bc!.bsn \\
\hline
\end{array} \\
\hline
\end{array}
$$

Figure 3.1: The class *Starter*.

3.4.2 Class Transmitter. This consists of one object *Node* and two operations (methods), namely *bcHandle* and *ReceiveAck* as presented in the following schema:

Transmitter

Node

$nt : NT$

$n : N$

$n \in nt.nodes$

bcHandle

$\Delta(n.rt.cip, n.rt.pip, n.rt.sn, n.wip)$

$bc? : \mathbb{P}BC$

$bc! : BC$

$ac! : AC$

$\forall\, bc_1, bc_2 : bc? \bullet bc_1.bsn = bc_2.bsn > n.rt.sn$

$(bc?sip, n.ip) \in nt.liks$

$$\begin{aligned}\exists\, x : bc? \bullet (&x.sw = \sqcap\{bc.wip \mid bc \in bc?\}\\ &\wedge ac!.dip = x.sip\\ &\wedge n.rt.pip' = chp(n.rt, x.sip)\\ &\wedge bc!.hop = x.hop + 1\\ &\wedge n.rt.sn' = x.sn)\end{aligned}$$

$ac!.sip = bc!.sip = n.ip$

$bc!.sw = n.wip$

$n.rt.cip' = \varnothing$

$n.wip' = 0$

ReceiveAck

$\Delta(n.wip, n.rt.cip)$

$ac? : AC$

$n.ip = ac?.dip$

$n.wip' = n.wip + 1$

$n.rt.cip' = n.rt.cip \cup \{ac?.sip\}$

$(ac?.sip, ac?.dip) \in nt.liks$

Figure 3.2: The class *Transmiter*.

Node: It is the node n of type N in the network nt of type NT. Each node in nt can have the value n.

bcHandle: It is an operation which takes a set of beaconing messages $bc?$, updates its weight $n.wip$ and routing table rt of the receiver of the beacon $bc?$ and output the new transformed beacon $bc!$ and acknowledgement message $ac!$ as well as updating the routing table of the receiver.

> Handling the set of beacons requires them to have the same sequence number which is greater than the one of the receiver.The node n handles them if it is connected to their senders. This guarantees the fact that the handled beacons are only the new ones.
>
> The node n chooses the beacon x of the smallest weight and uses it to update its routing table and to form the acknowledgement message $ac!$ and the new beacon $bc!$.
>
> A destination in acknowledgement message $ac!$ is a sender of the beacon x and hence the parent $n.rt.pip$ of node n. The number of hops in the beacon $bc!$ is obtained by incrementing by one that of the beacon x whereas the sequence number $n.rt.sn$ of the node becomes a copy of the beacon x.
>
> A sender of the beacon $bc!$ and the acknowledgement message $ac!$ is the node n and the weight in the beacon $bc!$ is the same as the weight $n.wip$ of the node n.
>
> The routing table of the node n also changes in a way that the set of children $n.cip$ becomes empty and the weight of the sender is changed to zero.

ReceiveAck: It is an operation which takes the acknowledgement message $ac?$ and changes the state of its receiver n, provided that the receiver's address $n.ip$ is the same as the destination address $ac?.dip$ specified in $ac!$.

The weight $n.wip$ of n is incremented by one and the sender of $ac?$ becomes one of the children of n. Note here that all this happens if the sender and the destination of the acknowledgement message are connected i.e $(ac?.sip, ac?.dip) \in nt.liks$.

3.4.3 Class *LIBA***.** It is a class which consists of processors namely *start* and *transmitters* as its objects. The starter *start* is the sink in the network and the transmitters are those nodes in the network which handle the beaconing messages or acknowledgement messages.

$LIBA$

$processors$

$start : Starter$

$transmitters : \mathbb{P}\, Transmitter$

$Generate \mathrel{\widehat=} start.Initiate \gg [p : Starter.sink.nbr] \bullet p.bcHandle$

$Trans \mathrel{\widehat=} [\gg p : transmitters] \bullet p.bcHandle [] p.ReceiveAck$

Figure 3.3: Beaconing class.

The operation *Generate* consists of initiating the beacon which is followed by handling the initiated message by the neighbours of the sink. The operation *Trans* consists of sequentially handling beacons or acknowledgement messages by all concerned nodes in the network.

3.5 Example

We illustrate the formalism in Section 3.4 by example. We start by explaining the notation and colours and then we represent the LIBA idea using the interpreted graphs.

3.5.1 Notation. In this example, the contents of the schemas are written in tuples for simplicity.

A beacon is represented by $bc(sip, bsn, hop, sw)$ which expresses the beacon bc from the sender sip of sequence number bsn where the sender is at the hop^{th} hop and its weight is sw.

An acknowledgement message is expressed as $ac(sip, dip)$ where sip and dip are the sender and destination of the acknowledgement message respectively.

A routing table looks like (pip, cip, sn) where the node having such routing table has the parent pip, the set of children cip and the last handled beaconing message had the sequence number sn.

3.5.2 Colouring logic. In this example, we clarify the effects of messages by showing their consequences in the same colours as their causes (messages). After considering an initial network as shown in Figure 3.4a, LIBA processes are shown by the following colours:

- **Red**: The red arrows show the traces of the beaconing messages. The changes in routing tables caused by the beacon are also shown in Red.
- **Green:** The green bent arrows show the traces of acknowledgement messages, where the green undirected links show the established routes, and the changes in routing tables or weights caused by the corresponding messages are also shown in Green.

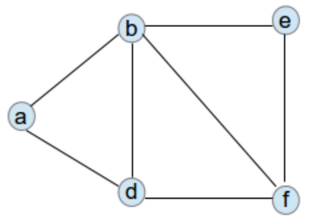
We now present two runs of the algorithm, to show all details.

3.5.3 Steps of the first run of LIBA. Suppose we want to find a path from each node to the sink a in the network shown by Figure 3.4a, where all nodes are assumed initially to have zero weight.

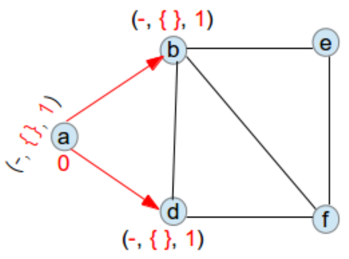
As shown by Figure 3.4b, the sink a starts by initiating its routing table and weight as well as broadcasting the beacon $bc(a, 1, 0, 0)$. Its neighbours b and d receive it and all update their routing tables to (-,$\{\}$, 1). Note here that whenever a node sends a beacon, its weight becomes zero. This is why the weight of a remained zero. Since each node updates only the beacon from its selected parent (sender whose weight is minimal) and relays it in the network, the beacon from node b is $bc(b, 1, 1, 0)$, for example.

We know that a routing table of a node is updated only by a beacon from its selected parent. This is why nodes b and d have the same routing tables $r(a, \{\}, 1)$ because they have selected the same parent a. Each node which receives a new beacon sends back an acknowledgement message (green bent arrows) to a parent it has selected, and relays the beacon from the parent.

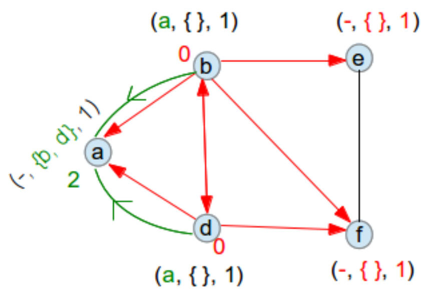
Figure 3.4c shows the nodes relaying the beacons and acknowledging their selected parents. If a node receives an acknowledgement, it increments its weight by one. This is why the weight of node a has been 2: It received two acknowledgement messages. Note that node a has rejected the beacons from all its neighbours since they were not new. That is, the sequence number of the incoming beacon was not greater than that of the node n. This happens to each node receiving an old beacon.



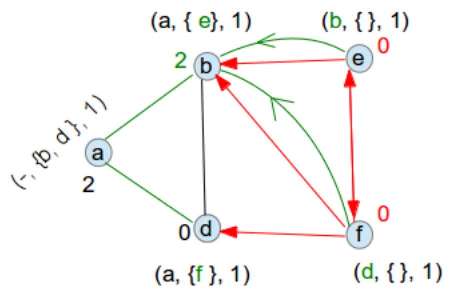
(a) Initial network.



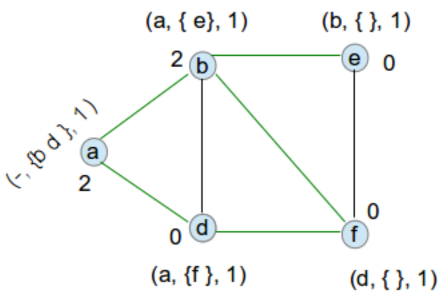
(b) Beaconing starts.



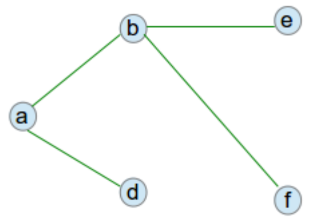
(c) Acknowledgement and relaying.



(d) Acknowledgement and relaying.



(e) Resulting network.



(f) Resulting tree.

Figure 3.4: First iteration of $LIBA$.

Figure 3.4d shows a continuation of parent selection as in the process described in Figure 3.4c, and figure 3.4e clarifies that the nodes which did not receive any acknowledgement have weight equal to zero.

Since each node knows its parent or all its children, the corresponding tree can be extracted as shown in Figure 3.4f by using the trace of the acknowledgement messages.

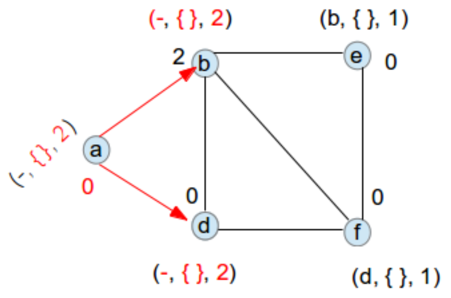
3.5.4 Steps of the second run of LIBA on the same network. Now suppose we start with a weighted network with routing tables and the node weights as shown in Figure 3.4e. The sink a starts the process by initialising its routing table, weight and a new beacon to broadcast.

As shown by Figure 3.5a, the initiated routing table of the sink is $r(-, \{\}, 2)$ and consists of no parent, empty list of children and a new sequence number 2 which is the same as the sequence number of the new beacon originating from the sink a. The new sequence number is obtained here by incrementing the previous one by one (that is 2=1+1). Thus the beacon from a is $bc(a, 2, 0, 2)$. Note that the sequence number 2 is obtained by incrementing the previous one by one, and the weight 2 is a copy of weight of the sender of the beacon prior being changed to zero.

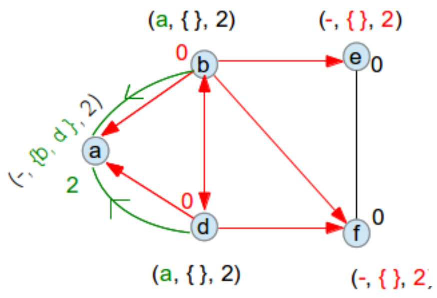
A routing table is updated by a newly incoming beacon, by emptying the existing list of children and by replacing the existing parents by a non existing element, and by making its sequence number a copy of that of the incoming beacon (see 3.5a). This is why for instance the routing table of node b, before acknowledging the beacon from its parent a, is $(-, \{\}, 2)$.

Figure 3.5b shows an action of relaying and acknowledging the messages, and the weight of the sink becomes 2 because it has received 2 new acknowledgement messages. Furthermore a destination of the message becomes a parent of its sender. This is why after acknowledging, the routing tables of nodes b and d included a in their routing table showing that a is their parent, and on the other hand the node a updates its list of children by including the nodes b and d.

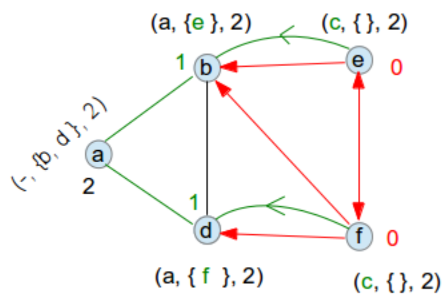
Figure 3.5c shows that node f makes a better choice and chooses node b to be its parent since it has the minimal weight (in comparison with node d which was also ready to be a parent).



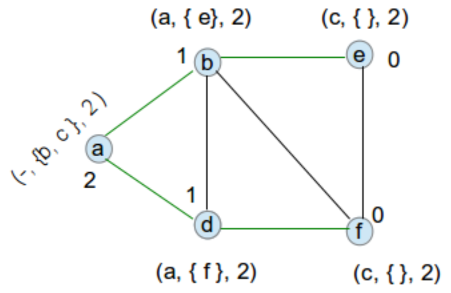
(a) Beaconing starts.



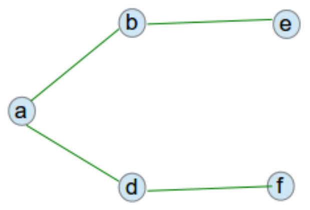
(b) Acknowledgement and relaying.



(c) Acknowledgement and relaying.



(d) Resulting network.



(e) Resulting tree.

Figure 3.5: Second iteration of $LIBA$.

Figure 3.5d shows that the nodes keep their weight to zero if they do not receive any acknowledgement message. Based on the new parent selection done so far and the new weight distribution, the related tree is shown by Figure 3.5e.

3.6 Abstractions and assumptions

- We abstract all optional processes such as repairing, mobility and issues related to multi-sinks networks.
- We assume that all nodes are working well and we do not model any probabilistic issue.
- We assume that no node is better than the other.
- We assume that the weight of a node is calculated after the reception of acknowledgement messages from all its children.
- We assume that a sequence number can increase up to infinity.

3.7 Related problems and correctness of LIBA

Routing problems consist of finding essentially the route satisfying some conditions/metrics such as minimum cost(see [13, 21, 20]), minimum distance (see [20]), etc., in an assumed graph. In general case, the word **weight** which can represent any condition/metric, is used.

The following are some problems and their solutions (algorithms) related to the routing problem explained in Section 3.1.

3.7.1 Minimum spanning tree problem. Given the connected, undirected and weighted graph G, the minimum spanning tree problem consists of finding the minimum spanning tree of G whose total[1] weight is less or equal to the weight of any other spanning tree of G. This problem is solved by the algorithms of Kruskal and Prim, see [34].

3.7.2 Single-source shortest path problem. Given a graph $G = (V, E)$, the aim is to find a shortest path from a given source vertex $s \in V$ to each vertex in V . The algorithm for the single-source shortest path problem can solve many other problems, including single-destination shortest-paths problem, single-pair shortest-path problem, all-pairs shortest-path problems as discussed on page 644 in [34].

This problem is solved by the following algorithms:

- **Breadth-first algorithm:** It is preferred when the weight of each edge in E is one unit. The time complexity is expressed as $\mathcal{O}(\#L + \#V)$.
- **Dijkstra's algorithm:** It is preferred for the case in which all edge weights are non-negative. This algorithm runs in $\mathcal{O}((\#V)^2)$ time.
- **Belman-Ford algorithm:** It exploits the breadth-first idea in general case, where edge weights may be positive or negative, See [34].The time complexity is expressed as $\mathcal{O}(\#L \times \#V)$.

All the above algorithms are not periodic (they terminate) and thus can only be compared with a single run of LIBA. We refer to the single-source shortest-path problem ($SSSP$) [34] and its solution to prove correctness of of one run of $LIBA$. $SSSP$ is solved by Bellman-Fold Algorithm (BFA). This algorithm uses the breadth first idea to choose the next links for adding to existing paths rooted at the sink (operation called **relaxation**). However, edges-weighted graph is considered and the weights of all edges remain unchanged. In contrast, $LIBA$ uses the weights of nodes and the weights are modified after each run. It is good to take advantage of the fact that the nodes choices in one run of $LIBA$ can be reduced to link choices done in BFA, in polynomial time. Note that correctness of the Bellman-Fold algorithm is proved on page 653 in [34].

We first show that all assumptions made in in $LIBA$ are also assumed in the Bellman-Fold theorem.

Consider a graph G as described by the schema NT in Section 3.3.4. The following are properties of G after any iteration of $LIBA$ on G:

3.7.3 Lemma. All nodes in G are reachable. .

Proof

The schema NT describes the graph G as a connected graph. On the other hand the condition $(bc?sip, n.ip) \in nt.liks$ in at schema $Transmitters$ together with the operation

$$Trans \mathrel{\widehat{=}} [\gg p : transmitters] \bullet p.bcHandle[]p.ReceiveAck$$

defined in schema $LIBA$ shows that each neighbour of the sender of beacon, handles or has handled a beacon by the operation $bcHandle$. This shows that each node in the network G is reachable.

[1]The total weight of a tree is equal to the sum or all the weights involved in finding (computing) the tree.

3.7.4 Lemma. No negative weights cycles are reachable from the source.

This comes from the fact that the weight of each edge is a non-negative valued function.

Note here that each edge in the minimum interference spanning tree [2] joins a parent (receiver of an acknowledgement message) and its child (sender of an acknowledgement message).

3.7.5 Theorem. (Correctness of one run of $LIBA$) *At termination of one iteration of $LIBA$ on graph $G : NT$ (as described by the schema NT), the path from each node to the source is shortest in number of hops and consists of nodes of minimum interference (weight).*

Proof:

The condition $\forall\, bc_1, bc_2 : bc? \bullet bc_1.bsn = bc_2.bsn \neq n.rt.sn$ in Figure 3.2, guarantees that a each node handles a beaconing message only once. Hence $LIBA$ terminates.

Furthermore, for the network G, define the links weights function as follows:

$$w : G.links \to \mathbb{N}$$

$$(p, c) \mapsto \begin{cases} p.wip & \text{if } d(p) \leq d(c) \\ min\{p.wip, c.wip\} & \text{if } d(p) = d(c) \end{cases}$$

Figure 3.6 shows how the nodes weights are transformed into links weights, after (or before) each run of LIBA. The two top graphs in Figure 3.6 show how initial nodes weights can be translated in initial links weights and the bottom ones show how any nodes-weights-distribution caused by LIBA can be translated into links-weights-distribution. From there the graphs show how relaxation done in Bellman-Fold algorithm is identified with parent choice in LIBA (see green arrows).

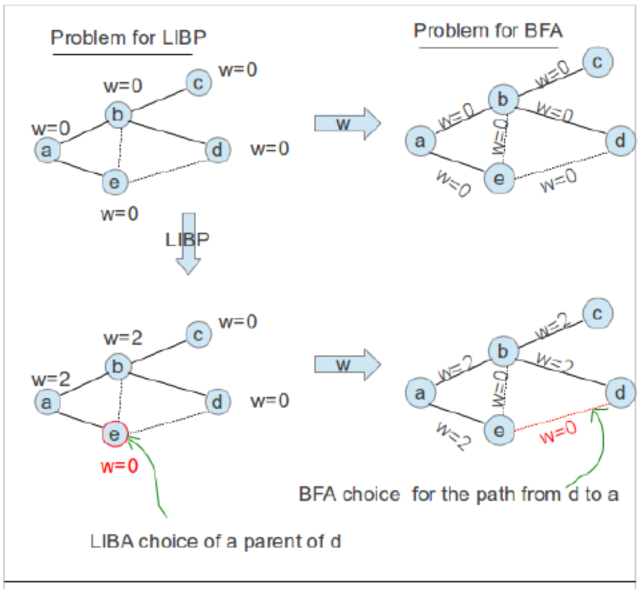


Figure 3.6: LIBA versus BFA.

The node a is considered to be the sink. Selecting an edge $e = (e, d)$ of weight w_e to join the shortest path to the source in Bellman-Fold algorithm, is equivalent to selecting a parent e of node d whose weight $w = w_e$ in $LIBA$ (this follows directly the definition of the weight function w).

[2] The tree node-weighted-minimum spanning tree obtained after the run of $LIBA$.

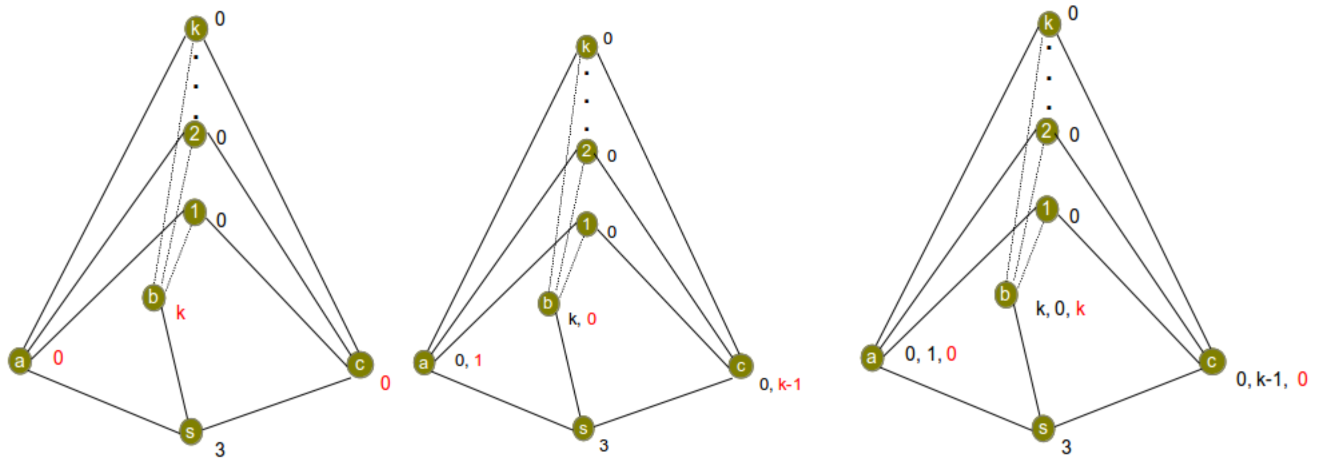
Figure 3.6 is a picture showing that the choice of a parent p of weight $p.w$ by a node c is equivalent to choosing a link (p, c) whose weight is $w(p, c)$. Since Bellman-Fold algorithm returns the shortest path, $LIBA$ does so too. According to the definition of w, it is clear that the shortest path consists of nodes with the least interference (weight). Since Bellman-Ford algorithm is correct [34], one run of $LIBA$ is correct.

4. Improved $LIBA$: LIBA$^+$

As shown in [7], LIBA was designed to improve energy efficiency in ubiquitous sensor networks. The main feature of LIBA to achieve efficiency is load balancing. [1] Simulation in [7] has revealed that the algorithm satisfies that property. However, no proof has been given. In this chapter, we show that the load balancing mechanisms in LIBA lead to weight cycling issues which are handled by proposing a new algorithm, LIBA$^+$. Furthermore, the improved algorithm (LIBA$^+$) is proven to be correct.

4.1 Weight cycling

In this section we show that LIBA does not efficiently satisfy load balancing and hence is incorrect. Consider the situation shown in Figure 4.1.



(a) Initial network. (b) Beaconing starts. (c) Acknowledgement and relaying.

Figure 4.1: Weight cycling.

We consider a family of three-dimensional graphs presented in Figure 4.1a, where the set $P = \{1, 2, ..., k\}$ consists of nodes which are all connected to the nodes a, b and c. We choose the non-zero natural number k to be big for clarifying how effective the problem is, and we assume that the node s is the sink of the network. Finally we assume that all nodes have initial weight zero (this can be generalized to the case where all nodes have equal weights).

Figure 4.1a shows a case where the sink sends a beacon and, as a result, all nodes in the set P choose the node b and this makes zero the weights of the nodes a and c.

In the next LIBA run, the nodes in P have to choose a or c because they have lower weights (they all have weight zero). Figure 4.1b shows that only one node from the set P has chosen node a, and all others have chosen node c. At this stage node b has not been chosen and as a result it has the weight zero.

The problem appears in third run of LIBA (Figure 4.1c): As node b has weight less than that of a and c, all nodes in P have to choose node b and its weight becomes k again, whereas nodes a and c update

[1] A routing algorithm satisfies the load-balancing property if and only if each established route consists of nodes which were previously less used, to ensure resource sharing.

their weights to zero. Notice that the weight redistribution in the third run of LIBA (4.1c) is the same as that in the first run (see Figure 4.1a).

Since node a, b and c forward the beacons to the same nodes (in $P \cup \{s\}$), the loads to nodes in P could tend to be balanced if the nodes a, b and c are chosen by around $k/3$ nodes which is the average number of choosers (nodes in P).

On the other hand, since the nodes redistribution in the second run (Figure 4.1b) is transformed in that of the first run (see Figure 4.1c) and vice versa, we consider the case this succession of weight redistribution is uniform (equivalent to the cycle (1 2)).

In this case the weight succession of a, b and c, for many runs is as follows:

- **node a:** 0, 1, 0, 1, 0, 1, ... (the weight of a is either 0 or 1)
- **node b:** k, 0, k, 0, k, 0, ... (the weight of b is either k or 0)
- **node c:** 0, k-1, 0, k-1, 0, k-1, ... (the weight of c is either o or k-1).

Note that all terms in the above three sequences are far from the average weight which is $k/3$ (k is considered to be a big natural number). As a result, there is no run at which the load will be balanced differently.

4.2 Prioritisation: Load balancing handling

We still consider Figure 4.1. To ensure efficient load balancing, node a would be prioritised because it has a lower weight (comparing with b and c), whereas node b would not be less preferred because it has a greater weight in comparison with other potential parents (a and c). However, since the current algorithm does not prioritise the node a in any way, it is possible to get a case where load balancing fails, as shown in the previous section.

The problem discussed in Section 4.1 comes from the fact that the weight computation does not depend on all previously computed weights. We want to define a function which takes all previously calculated weights and returns the weight which prioritises a node which has supported the least number of children so far. The weight will represent in this case the history of the usage of a node and hence its energy consumption.

Consider a node to have support w_k children at the k^{th} run of LIBA. We suggest the following prioritisation function :

$$h : \mathbb{N}^k \to \mathbb{N}$$
$$(w_1, w_2, ..., w_k) \mapsto \sum_{i=1}^{k} w_i$$

To prioritise a less busy node, we choose to define the function $h(w1, w2, ...)$ as its current weight.

Considering Figure 4.1, we see that at the third run the node a would have lower total weight (which is 1) and hence will be the one to be chosen and hence this clearly improves the resource sharing which was the main aim of LIBA.

This enables us to improve the class $Transmitter^{+}$ as follows:

$$
\begin{array}{l}
\textbf{Transmitter}^{+} \\
\hline
\mathit{Transmitter.Node} \\
\\
\quad \mathit{bcHandle}^{+} \\
\quad \hline
\quad \Delta(n.rt.cip, n.rt.pip, n.rt.sn) \\
\quad bc? : \mathbb{P} BC \\
\quad bc! : BC \\
\quad ac! : AC \\
\quad \hline
\quad \forall\, bc_1, bc_2 : bc? \bullet bc_1.bsn = bc_2.bsn \neq n.rt.sn \\
\quad \exists\, x : bc? \bullet x.sw = \sqcap\{bc.wip \mid bc \in bc?\} \\
\qquad\qquad\qquad \wedge ac!.dip = x.sip \\
\qquad\qquad\qquad \wedge n.rt.pip' = pch(x, x.sip) \\
\qquad\qquad\qquad \wedge bc!.hop = x.hop + 1 \\
\qquad\qquad\qquad \wedge n.rt.sn' = x.sn \\
\quad ac!.sip = bc!.sip = n.ip \\
\quad bc!.sw = n.wip \\
\quad n.rt.cip' = \varnothing \\
\quad (bc?sip, n.ip) \in nt.liks \\
\quad \hline
\\
\quad \mathit{ReceiveAck}^{+} \\
\quad \hline
\quad \Delta(n.wip, n.rt.cip) \\
\quad ac? : AC \\
\quad \hline
\quad n.ip = ac?.dip \\
\quad n.wip' = n.wip + 1 \\
\quad n.rt.cip' = n.rt.cip \cup \{ac?.sip\} \\
\quad (ac?.sip, ac?.dip) \in nt.liks \\
\quad \hline
\\
\hline
\end{array}
$$

Figure 4.2: The class *Transmitter*$^+$.

4.2.1 Class *Transmitter*$^+$**.** This consists of one object: the same node used in the class *Transmitter*, and two operations (methods) namely *bcHandle*$^+$ and *ReceiveAck*$^+$ as presented in the following schema:

Operation *bcHandle*$^+$**:** This operation takes a set of beaconing messages $bc?$, makes changes of the routing table rt of the receiver of the beacon $bc?$ and outputs the new transformed beacon $bc!$ and acknowledgement message $ac!$ as well as updating the routing table of the receiver.

To handle the set of beacons requires them to have the same sequence number, and the receiver n handles them if it has a different sequence number. This guarantees the fact that the handled beacons are new.

The node n chooses a beacon x of smallest weight and uses it to update its routing table and to form the acknowledgement message $ac!$ and the new beacon $bc!$.

A destination in acknowledgement message $ac!$ is a sender of the beacon x and hence the parent $n.rt.pip$ of node n. The number of hops in the beacon $bc!$ is obtained by incrementing by one that of the beacon x whereas the sequence number $n.rt.sn$ of the node becomes a copy of that of the beacon x.

A sender of the beacon $bc!$ and the acknowledgement message $ac!$ is the node n and the weight in the beacon $bc!$ is the same as the weight $n.wip$ of node n.

The routing table of node n also changes in a way that the set of children $n.cip$ becomes empty.

Note that the above changes apply to the node n if the sender of the beacon is connected to it.

Operation $ReceiveAck^{+}$**:** is an operation which takes the acknowledgement message $ac?$ and change the state of its receiver n, provide that the receiver's address $n.ip$ is the same as the destination address $ac?.dip$ specified in $ac!$.

The weight $n.wip$ of n is incremented by one and the sender of $ac?$ becomes one of the children of n. Note here that all this happens if the sender and the destination of the acknowledgement message are connected i.e. $(ac?.sip, ac?.dip) \in nt.liks$.

4.2.2 Class $LIBA^{+}$**.** This class consists of processors namely $start$ and $transmitters$ as its objects. The starter $start$ is the sink in the network and the transmitters are those nodes in the network which handle the beaconing messages or acknowledgement messages.

$LIBA^{+}$

$processors$

$start : Starter$

$transmitters : \mathbb{P}\,Transmitter^{+}$

$Generate \mathrel{\widehat=} start.Initiate \gg [p : Starter.sink.nbr] \bullet p.bcHandle^{+}$

$Trans \mathrel{\widehat=} [\gg p : transmitters] \bullet p.bcHandle^{+}[]p.ReceiveAck^{+}$

Figure 4.3: Beaconing class for $LIBA^{+}$.

The operation $Generate$ consists of initiating the beacon which is followed by handling the initiated message by the neighbours of the sink. The operation $Trans$ consists of sequentially handling beacons or acknowledgement messages by all concerned nodes in the network.

4.3 Example

We illustrate the formalism in Section 4.2 by an example. We start by explaining the used notations and colours and after we represent the LIBA^{+} idea using the interpreted graphs.

4.3.1 Notation. In this example, the contents of the schemas are written in tuples for simplicity.

A beacon is represented by $bc(sip, bsn, hop, sw)$ which expresses the beacon bc from the sender sip of sequence number bsn where the sender is at the hop^{th} hop and its weight is sw.

An acknowledgement message is expressed as $ac(sip, dip)$ where sip and dip are the sender and destination of the acknowledgement message respectively.

A routing table looks like (pip, cip, sn) where the node having such routing table has the parent pip, the set of children cip and the last handled beaconing message had the sequence number sn.

4.3.2 Colouring logic. In this example we clarify the effect of messages by showing the consequences in the same colours as their causes (messages). After considering an initial network as shown in Figure 4.4a, LIBA$^+$ processes are shown by the following colours:

- **Red**: The red arrows show the traces of beaconing messages. The changes in routing tables caused by the beacon, are also shown in red.
- **Green:** The green bent arrows show the traces of acknowledgement messages, where the green undirected links show the established routes, and the changes in routing tables or weights caused by the corresponding messages are also shown in Green.

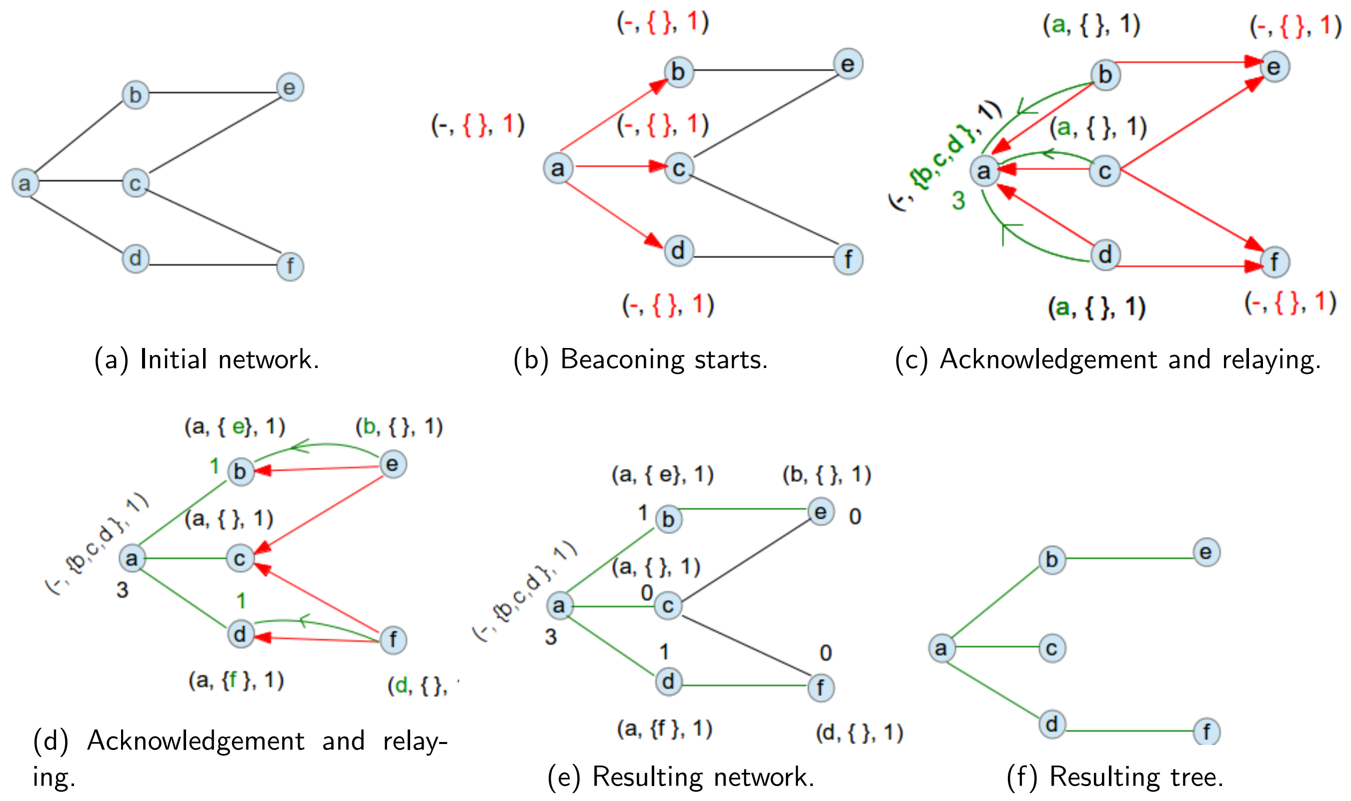
We now present two runs of the algorithm, to show all details.

4.3.3 Steps of a first run of LIBA$^+$. Suppose we want to find a path from each node to the sink a in the network shown by Figure 4.4a, where all nodes are assumed to have zero weight.

As shown by Figure 4.4b, the sink a starts by initiating its routing table and broadcasts the beacon $bc(a, 1, 0, 0)$. Its neighbours b, c and d receive it and all update their routing tables so as to become (–,{}, 1). Since each node updates only the beacon from its selected parent (sender whose weight is minimum) and relays it in the network, the beacon from node b is $bc(b, 1, 1, 0)$, as an example.

We know that a routing table of a node is updated only by a beacon from its selected parent. This is why nodes b, c, d have the same routing tables $r(a, \{\}, 1)$: They have selected the same parent a. Each node receives a new beacon sends back an acknowledgement message (green bent arrows) to the selected parent, and relays the beacon from its selected parent.

Figure 4.4c shows the nodes relaying the beacons and acknowledging their selected parents. If a node receives an acknowledgement, it increments its weight by one. This is why the weight of node a has been 3 since it received three acknowledgements messages. Note that the node a has rejected the beacons from all its neighbours since they were not new, that is, the sequence number was the same as in its routing table.



(a) Initial network. (b) Beaconing starts. (c) Acknowledgement and relaying.

(d) Acknowledgement and relaying. (e) Resulting network. (f) Resulting tree.

Figure 4.4: First iteration of $LIBA^{+}$.

Figure 4.4d shows a continuation of the process in Figure 4.4c and Figure 4.4e shows that the nodes which did not receive any acknowledgement message keep their previous weight (in this case their weights remain zero).

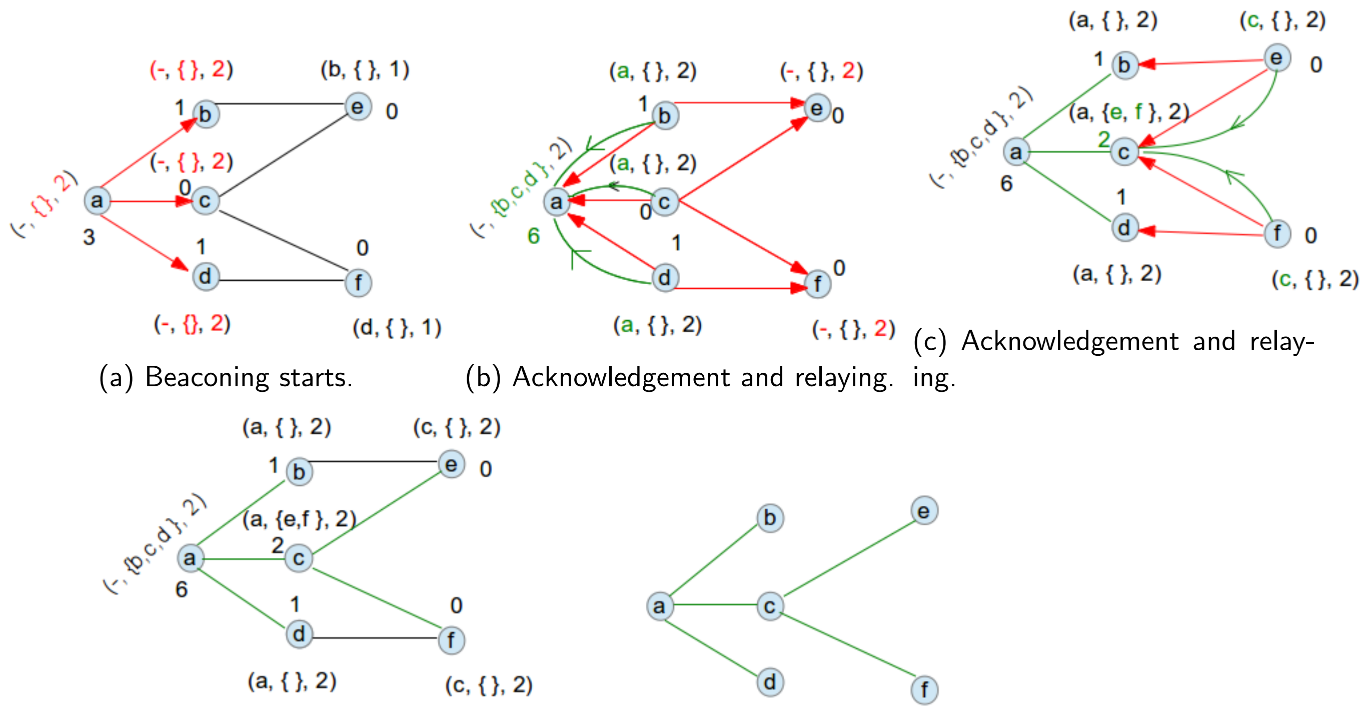
Since each node knows its parent or all its children, the corresponding tree can be extracted, as shown by Figure 4.4f by using the trace of the acknowledgement messages.

4.3.4 Steps of a second run of LIBA^{+} on the same network. Now suppose we start with a weighted network with routing tables and the node weights as in Figure 4.4e. The sink a starts the process by initialising its routing table and the new beacon.

As shown by Figure 4.5a, the initiated routing table of the sink is $r(-, \{\}, 2)$ and consists of no parent, empty list of children and a new sequence number 2 which is the same as the sequence number of the new beacon originating from the sink a. Thus the beacon from a is $bc(a, 2, 0, 3)$. Note that the number 2 is obtained by incrementing the previous sequence number by one.

The routing tables are updated by the beacon, by empting the existing list of children and by replacing the existing parents by a non-existing element, and its sequence number becomes a copy of that of the incoming beacon (see 4.5a). This is why, for instance, the routing table of node b, after acknowledging the beacon from its parent a, is $(-, \{\}, 2)$.

Figure 4.5b shows an action of relaying and acknowledging the messages, and the weight of the sink becomes 6 because it has received three new acknowledgement messages. Figure 4.5c shows that nodes e and f make a better choice and choose the node c to be their parent since it has the minimum weight (in comparison with nodes b or d which were also ready to be the parent).



(a) Beaconing starts. (b) Acknowledgement and relaying. (c) Acknowledgement and relaying.

(d) Resulting network. (e) Resulting tree.

Figure 4.5: Second iteration of $LIBA^{+}$.

Figure 4.5d shows that the nodes keep their weight if they do not receive any acknowledgement message. Based on the new parent selection done so far and the new weight distribution, the resulting tree is shown in Figure 4.5e.

4.4 Verification

We use Computation Tree Logic (CTL) as described in Section 2.5 to express properties and hence verify them.

4.4.1 Theorem. Parent choice

Suppose a beacon is broadcast k rounds by the sink of a network of type NT (see Section 3.3). At each round, each node different from the sink keeps on choosing its parent to be its neighbour, satisfying the following properties:

- ▶ *Being closest node to the sink, i.e. a node at fewest hops from the sink.*
- ▶ *Having the minimum weight which means a node of minimal interference.*

To express this in CTL syntax, we first define the following predicates:

- $beacon(s, k)$: holds whenever the sink s of the network of type NT(see Section 3.3.4) initiates and broadcasts a beacon of type $Starter.Initiate.bc!$ (see Figure 3.1), for the k^{th} round. That is

$$beacon(s, k) \Leftrightarrow \exists\, bc : Starter.Initiate.bc! \bullet bc.sip = s \wedge bc.bsn = k.$$

- $parent(p, n, k)$: holds whenever node n makes a choice of parent p (its neighbour of minimum weight and nearest to the sink of the network) for its k^{th} round. Note that the number of rounds equals the sequence number of a newest beacon. Hence,

$$parent(p, n, k) \Leftrightarrow \exists\, ac : Transmitter^{+}.bchandle.ac! \bullet \exists\, bc : Transmitter^{+}.bchandle.bc! \bullet (ac.sip, ac.dip) = (p, n) \wedge bc.bsn = k.$$

The expression of the parent choice is then as follows:

$$\forall\, k : \mathbb{N} \bullet beacon(s, k) \;\Rightarrow (AG(n \neq s \;\Rightarrow\; parent(p, n, k))).$$

Proof

Assuming that the predicate $beacon(s, k)$ holds, we proceed by induction, to show that the CTL formula $AG(n \neq s \;\Rightarrow\; parent(p, n, k))$ holds too.
Consider the predicate $bchandle(n, 1)$ which holds if the node n handles the new incoming beacons by the operation $Transmitter^{+}.bchandle$, for the first round. From Figure 4.3 (see operation $LIBA.Generate$ and $LIBA.Trans$), it follows that,

$$\forall\, x : nt.nodes \bullet bchandle(x, 1).$$

On the other hand, $bchandle(x, 1) \Rightarrow (x \neq s \;\Rightarrow\; parent(x, 1))$, as shown by Figure 4.3 and 4.2 (see operation $Bchandle$).
Hence,

$$beacon(s, 1) \;\Rightarrow AG(n \neq s \;\Rightarrow\; parent(n, 1)).$$

Assume that $\forall\, r < k \bullet beacon(s, r) \;\Rightarrow AG(n \neq s \;\Rightarrow\; parent(n, r))$.
We verify that $beacon(s, r + 1) \;\Rightarrow AG(n \neq s \;\Rightarrow\; parent(p, n, r + 1))$.
At each node, the r^{th} computed weights can be used to compute the new ones as shown by $Transmitters.ReceiveAck$ in Figure 4.2.
In all r beaconing rounds, no operation changes the structure of the network, this is why
$beacon(s, r + 1) \Rightarrow \forall\, x : nt.nodes \bullet bchandle(x, r + 1)$.
On the other hand, the sink is the only node which does not satisfy the condition $\forall\, bc_1, bc_2 \;:\; bc? \;\bullet\; bc_1.bsn \;=\; bc_2.bsn \;\neq\; n.rt.sn$ in Figure 4.2, operation $Bchandle$, which is why it is the only node in the network which does not choose a parent.
Hence,

$$AG(n \neq s \;\Rightarrow\; parent(p, n, r + 1)).$$

Thus,

$$\forall\, k : \mathbb{N} \bullet AG(n \neq s \;\Rightarrow\; parent(p, n, k)).$$

In Section 3.7.5 we proved correctness of **one** run of the algorithm. In the following theorem, we prove that in all runs, the improved LIBA which is LIBA^{+} is correct that is $LIBA^{+}$ establishes the shortest path from each node to the sink consisting of nodes of minimum weight.

4.4.2 Theorem. *Consider the network $nt : NT$ whose sink is the node s. After each run of $LIBA^+$ on nt, the path from each node of the network to s is shortest and consists of the nodes of minimum interference (weight).*

By $path$, we refer to a sequence of adjacent nodes and their chosen parents where each node is the parent of the previous one.

To express this theorem in CTL, we first define the following predicates.

- $shortest(path(n, s))$: The path $path(n, s)$ from node n to the sink s consists of a minimum number of nodes.
- $cheapest(path(n, s))$: The path $path(n, s)$ consists of a minimum total weight (sum of its node weights).

The correctness theorem is then expressed as the following CTL expression:

$$AG(shortest(path(n, s)) \wedge cheapest(path(n, s))).$$

Proof

Suppose $\exists\, m : \mathbb{N}$ such that at the m^{th} run, the chosen path to the sink is not shortest or cheapest.
Then there exists a node in that path which has a parent which is not cheapest of closest to the sink. This contradicts theorem 4.4.1.

5. Interference transmission in a network using LIBA$^+$

$LIBA^+$ consists of periodically sending the messages in a network for nodes to choose a better parent. Each chosen parent increments its weight which is the measure of its interference, and hence a new message influences a change in interference in the network. In this chapter we consider interference as a disease and its measure indicates the stage of contamination of the network. In Section 5.1, we study the weight redistribution when a network is using the algorithm LIBA$^+$. The nodes which can transfer interference to each other are grouped in sets defined and studied in Section 5.2, and the interference-related states of nodes are classified in Section 5.3. A proposed model is presented in Section 5.4, with assumptions stated in Section 5.5. The model is analysed in Section 5.6 and numerical results are interpreted in Section 5.7.

5.1 Interference redistribution when LIBA$^+$ is being used by a network

In this section, we study interference redistribution when LIBA$^+$ is newly used on a graph. PYTHON is used to perform 21 redistributions on the same network and results are shown in hash tables whose keys and values are the nodes' addresses and their corresponding weights respectively. We are interested in interference redistribution in a safe network and then in an attacked one. Note that a network is attacked if one of its node sends the messages having wrong information: wrong interference measure for interference case.

5.1.1 Example. Consider the graph shown in Figure 5.1.

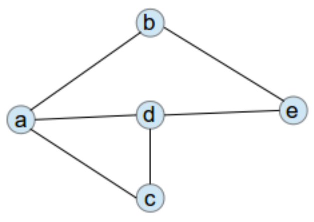
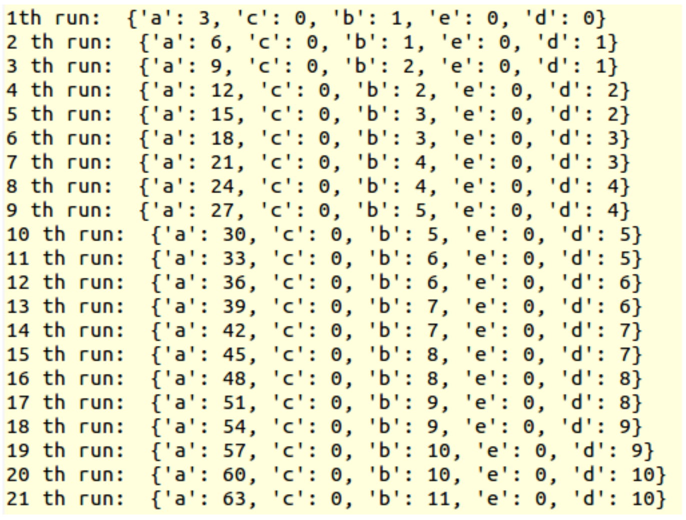


Figure 5.1: Initial network.

We assume that all nodes have initial weight zero. Consider 21 beacons successively sent from the sink a and we study the weight redistribution. We consider a safe network in one case and attacked ones in the other.

5.1.2 Safe network. Considering the case where the network is not attacked, we get the interference redistribution in Figure 5.2.



```
1th run:  {'a': 3, 'c': 0, 'b': 1, 'e': 0, 'd': 0}
2 th run:  {'a': 6, 'c': 0, 'b': 1, 'e': 0, 'd': 1}
3 th run:  {'a': 9, 'c': 0, 'b': 2, 'e': 0, 'd': 1}
4 th run:  {'a': 12, 'c': 0, 'b': 2, 'e': 0, 'd': 2}
5 th run:  {'a': 15, 'c': 0, 'b': 3, 'e': 0, 'd': 2}
6 th run:  {'a': 18, 'c': 0, 'b': 3, 'e': 0, 'd': 3}
7 th run:  {'a': 21, 'c': 0, 'b': 4, 'e': 0, 'd': 3}
8 th run:  {'a': 24, 'c': 0, 'b': 4, 'e': 0, 'd': 4}
9 th run:  {'a': 27, 'c': 0, 'b': 5, 'e': 0, 'd': 4}
10 th run:  {'a': 30, 'c': 0, 'b': 5, 'e': 0, 'd': 5}
11 th run:  {'a': 33, 'c': 0, 'b': 6, 'e': 0, 'd': 5}
12 th run:  {'a': 36, 'c': 0, 'b': 6, 'e': 0, 'd': 6}
13 th run:  {'a': 39, 'c': 0, 'b': 7, 'e': 0, 'd': 6}
14 th run:  {'a': 42, 'c': 0, 'b': 7, 'e': 0, 'd': 7}
15 th run:  {'a': 45, 'c': 0, 'b': 8, 'e': 0, 'd': 7}
16 th run:  {'a': 48, 'c': 0, 'b': 8, 'e': 0, 'd': 8}
17 th run:  {'a': 51, 'c': 0, 'b': 9, 'e': 0, 'd': 8}
18 th run:  {'a': 54, 'c': 0, 'b': 9, 'e': 0, 'd': 9}
19 th run:  {'a': 57, 'c': 0, 'b': 10, 'e': 0, 'd': 9}
20 th run:  {'a': 60, 'c': 0, 'b': 10, 'e': 0, 'd': 10}
21 th run:  {'a': 63, 'c': 0, 'b': 11, 'e': 0, 'd': 10}
```

Figure 5.2: Non-attacked network.

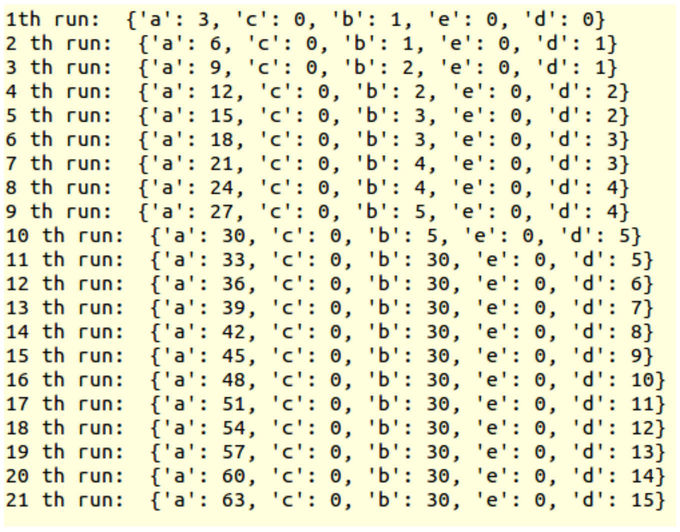
Figure 5.2 shows weight redistribution in 21 runs (rounds) of $LIBA^+$. For instance, line one represent the weight distribution in the first run where node a has the weight 3, c has 0, b has 1 where d and e have 0.

The figure shows that the nodes c and e have always weights equal to zero. This is due to the fact that each of them could not be parent of any node in the network, in all 21 runs.

The sink a is always chosen by the nodes b, c and d. This is why its weight is always incremented by three since the three nodes have no other parent choice.

An important property of $LIBA^+$ is shown by the way the weights of nodes b and d are computed. If a weight of one of the two nodes is incremented, the weight of the other node remains constant. This is because node e has to choose one of them depending on a low weight of any of them. Thus, the two nodes subsequently succeed to be parents of node e and hence to increment their weights.

5.1.3 Externally attacked network. We consider a case where at the eleventh run the network is attacked. The network is attacked by making the node b sending the beacons with wrong weight (weight=30).



```
1th run:  {'a': 3, 'c': 0, 'b': 1, 'e': 0, 'd': 0}
2 th run:  {'a': 6, 'c': 0, 'b': 1, 'e': 0, 'd': 1}
3 th run:  {'a': 9, 'c': 0, 'b': 2, 'e': 0, 'd': 1}
4 th run:  {'a': 12, 'c': 0, 'b': 2, 'e': 0, 'd': 2}
5 th run:  {'a': 15, 'c': 0, 'b': 3, 'e': 0, 'd': 2}
6 th run:  {'a': 18, 'c': 0, 'b': 3, 'e': 0, 'd': 3}
7 th run:  {'a': 21, 'c': 0, 'b': 4, 'e': 0, 'd': 3}
8 th run:  {'a': 24, 'c': 0, 'b': 4, 'e': 0, 'd': 4}
9 th run:  {'a': 27, 'c': 0, 'b': 5, 'e': 0, 'd': 4}
10 th run:  {'a': 30, 'c': 0, 'b': 5, 'e': 0, 'd': 5}
11 th run:  {'a': 33, 'c': 0, 'b': 30, 'e': 0, 'd': 5}
12 th run:  {'a': 36, 'c': 0, 'b': 30, 'e': 0, 'd': 6}
13 th run:  {'a': 39, 'c': 0, 'b': 30, 'e': 0, 'd': 7}
14 th run:  {'a': 42, 'c': 0, 'b': 30, 'e': 0, 'd': 8}
15 th run:  {'a': 45, 'c': 0, 'b': 30, 'e': 0, 'd': 9}
16 th run:  {'a': 48, 'c': 0, 'b': 30, 'e': 0, 'd': 10}
17 th run:  {'a': 51, 'c': 0, 'b': 30, 'e': 0, 'd': 11}
18 th run:  {'a': 54, 'c': 0, 'b': 30, 'e': 0, 'd': 12}
19 th run:  {'a': 57, 'c': 0, 'b': 30, 'e': 0, 'd': 13}
20 th run:  {'a': 60, 'c': 0, 'b': 30, 'e': 0, 'd': 14}
21 th run:  {'a': 63, 'c': 0, 'b': 30, 'e': 0, 'd': 15}
```

Figure 5.3: Attacked network.

Figure 5.3 shows that only node d is affected: it increments its weights continuously and hence the last weight of the node becomes bigger than that in Figure 5.2. So infected nodes pass on infection.

5.2 Interference set I

In this section we define a new structure of nodes in a network. We refer to the fact that an increase of interference level (weight) of a node may cause other nodes to be preferred and in this case we say that interference is transferred from one node to another.

Note that increase of interference level of a node does not necessarily affect all nodes in the network. It rather affects a selected set of nodes which we call an **interference set**.

5.2.1 Definition. Consider $G(L, N, W, s)$ to be a network as described in Section 2.3.1. An interference set I is a non-empty subset of N satisfying the following properties:

P_1**:** All nodes in set I are at the same distance from the sink. That is,

$$\forall\, x, y : I \bullet \; d(x) = d(y) \text{ i.e. } d \restriction_I \text{ is constant } (d \restriction_I \text{ is the restriction of the function } d \text{ on } I.)$$

P_2**:** I is a singleton, or for each node x in I there is another node y in I such that x and y share the next neighbour. [1] Mathematically,

$$\#I = 1 \;\vee\; (\forall\, x : I \bullet \exists\, y : I \setminus \{x\} \bullet \exists\, c : N \;\bullet d(c) = d(x) + 1 \wedge c : nbr(x) \cap nbr(y)).$$

P_3**:** For each node x in N, if x shares a next neighbour (which is at $d(x) + 1$) with some node in I, then $x : I$. That is,

$$\forall\, x : N \bullet \exists\, y : I \bullet \exists\, c : N \;\bullet d(x) = d(y) = d(c) - 1 \wedge c : nbr(x) \cap nbr(y) \Rightarrow x : I.$$

The predicates (properties) P_1, P_2, P_3 in Definition 5.2.1 are used to define an interference set:

```
┌ 𝓘 ────────────────────────────
│ nt : NT
│ I : ℙN
├──────────
│ I ⊂ nt.nodes
│ P₁
│ P₂
│ P₃
└───────────────────────────────
```

As an example, Figure 5.4 shows the graph whose nodes are grouped in Interference sets. Note that an interference set is a state property which is determined by the network and not the dynamics on the algorithm.

[1] The next node of the node n refers to a node connected to n which is at $(d(n) + 1)^{th}$ hop.

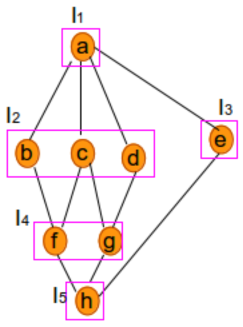


Figure 5.4: Interference sets.

From Figure 5.4, nodes are grouped in interference sets as follows:

$I_1 = \{a\}$, $I_2 = \{b, c, d\}$, $I_3 = \{e\}$, $I_4 = \{f, g\}$ and $I_5 = \{h\}$.

It is clear that the set of all nodes N of the presented graph is the union of all interference sets of the graph. That is $N = \cup_{i=1}^{5} I_i$. On the other hand all interference sets of the graph are pairwise disjoint. That is $I_i \cap I_j \neq \varnothing \Leftrightarrow i = j$. Thus the set $\{I_1, I_2, I_3, I_4, I_5\}$ of all interference sets forms **a partition** of the set N.

The figure also shows that any two nodes in the same interference set do not need to have the same next neighbour. For instance, the nodes b and d are in the same interference set I_2 but do not have the same next neighbour: The next neighbour of b is f whereas the next neighbour of d is g.

Furthermore the nodes e and g (or f) share the same next neighbour h, but they are not in the same interference set because they are not at the same distance from the sink a. In fact $d(f) = d(g) = 2$ but $d(e) = 1$ and clearly $d(f) = d(g) \neq d(e)$.

5.2.2 Lemma. Consider the set $\mathcal{I}$ of all interference sets of a graph $G(L, N, W, s)$ (see Section 2.3.1). Each interference set I of G is maximal in $\mathcal{I}$. That is,

$$\forall\, I_1, I_2 : \mathcal{I}, I_1 \subset I_2 \Rightarrow I_1 = I_2$$

Proof

Let I_1 and I_2 be two interference sets such that $I_1 \subset I_2$.
We want to show that $I_1 = I_2$. Since $I_1 \subset I_2$, it is sufficient to show that $I_2 \setminus I_1 = \varnothing$ because $I_2 = I_1 \cup (I_2 \setminus I_1)$.
We proceed by contradiction.
Let $x : I_2 \setminus I_1$. This means that $x : I_2$ and $x \notin I_1$. Since $x : N$, property P_3 in Definition 5.2.1 is valid. That is,

$$\exists\, y : I_1 \bullet \exists\, c : N \;\bullet\; d(x) = d(y) = d(c) - 1 \wedge c : nbr(x) \cap nbr(y) \Rightarrow x : I_1$$

Taking the contrapositive and using the fact that $x \notin I_1$,

$$\forall\, y : I_1 \bullet \forall\, c : N \bullet d(x) \neq d(c) - 1 \vee d(y) \neq d(c) - 1 \vee c \notin nbr(x) \cap nbr(y)$$

So no node $y : I_1$ shares the next hope with the node x. It follows that no node in I_1 can be in the same interference set as x. This contradicts the fact that nodes in I_1 and x are in the same interference set I_2.

In two steps (Theorems 5.2.3 and 5.2.4), we now show that the interference sets partition the set of all nodes of a the network G(L,N,W,s).

5.2.3 Theorem. *Given the graph G(L,N,W,s) as described in Section 2.3.1, the set $\mathcal{I}$ of all interference sets of G are pairwise disjoint. That is,*

$$\forall\, I_1, I_2 : \mathcal{I} \bullet I_1 \cap I_2 \neq \varnothing \Rightarrow I_1 = I_2.$$

Proof

Let I_1 and I_2 be any two interference sets. We want to show that

$$I_1 \cap I_2 \neq \varnothing \Rightarrow I_1 = I_2.$$

Let $x : I_1 \cap I_2$.
If both I_1 and I_2 are singletons, then we are done.
If I_1 is singleton and I_2 is not, $I_1 \cap I_2 \neq \varnothing$ implies that $I_1 \subset I_2$ and hence $I_1 = I_2$ because I_1 and I_2 are maximal in $\mathcal{I}$ (Lemma 5.2.2).
Consider I_1 and I_2 to be two Interference sets of size greater than 1, and let us proceed by contradiction.
Let $I_1 \neq I_2$ and suppose $y : I_1 \setminus I_2$ ($I_1 \setminus I_2 \neq \varnothing$ and if not $I_1 \subset I_2$ which this contradicts Lemma 5.2.2). It follows that there is no node belonging to I_2, sharing its next neighbour with the node y.
Since $I_1 \cap I_2 \subset I_2$, there is no node in $I_1 \cap I_2$ sharing the next hop with the node y.
Hence, by property P_2 in Definition 5.2.1 (definition of interference set), there is no node in $I_1 \cap I_2$ belonging in the same interference set as the node y.
This implies that the nodes x and y are not in the same interference set.
On the other hand $x : I_1 \cap I_2$ implies that $x : I_1$ and $y : I_1 \setminus I_2$ implies that $y : I_1$, and this contradicts the fact that the nodes x and y do not belong in the same interference set. Hence $I_1 = I_2$.

5.2.4 Theorem. *Let $\mathcal{I}$ be the set of all interference of the graph $G(L, N, W, s)$ (see 2.3.1). For each node x in N there exist an interference set I in $\mathcal{I}$ containing x. That is*

$$\forall\, x : N \bullet \exists\, I : \mathcal{I} \bullet x : I.$$

Proof

For $x : N$ we want to find an interference set which contains x. Define a subset M of N satisfying the following characteristics:

C_1: All nodes in M are at the distance $d(x)$. That is,

$$\forall\, y : M \bullet d(y) = d(x)$$

C_2**:** $M = \{x\}$, or for each node n in M there is another node y in M such that n and y share a next neighbour. That is,

$$M = \{x\} \vee (\forall\, n : M \bullet \exists\, y : M \backslash \{n\} \bullet \exists\, c : N \ \bullet d(c) = d(n)+1 \wedge c : nbr(n) \cap nbr(y)).$$

C_3**:** For each node n in N, if n shares a next neighbour with node y in M, then n is a member of M. That is,

$$\forall\, n : N \bullet \exists\, y : M \bullet \exists\, c : N \ \bullet d(n) = d(y) = d(c)-1 \wedge c : nbr(n) \cap nbr(y) \Rightarrow n : M.$$

C_4**:** x shares a next neighbour with a node y in M. That is,

$$\exists\, y : M \bullet \exists\, c : N \ \bullet d(x) = d(y) = d(c) - 1 \wedge c : nbr(n) \cap nbr(y)$$

Claim: $x : M : \mathcal{I}$.

Since $x : N$ and C_4 is valid, C_3 shows that $x : M$ (by setting $n = x$).
On the other hand $C_1 \Rightarrow P_1$, $C_2 \Rightarrow P_2$ and $C_3 \Rightarrow P_3$ (see 5.2.1), and hence $M : \mathcal{I}$. Thus

$$\forall\, x : N \bullet \exists\, M : \mathcal{I} \bullet x : M.$$

5.2.5 Corollary. The set $\mathcal{I}$ of all interference sets of the network $G(L, N, W, s)$ (see 2.3.1) partitions N.

Proof

By definition in 5.2.1, $\mathcal{I}$ consists of non-empty elements. In addition Theorem 5.2.3 shows that $\mathcal{I}$ consists of disjoints elements. It is then sufficient to show that the union of the sets in $\mathcal{I}$ is equal to N. That is

$$\bigcup_{I:\mathcal{I}} I = N$$

By Definition 5.2.1, $\forall I : \mathcal{I} \bullet I \subset N$. Hence

$$\bigcup_{I:\mathcal{I}} I \subset N \tag{5.2.1}$$

On the other hand, from Theorem 5.2.4 follows,

$$N \subset \bigcup_{I:\mathcal{I}} I \tag{5.2.2}$$

Hence from Equations 5.2.1 and 5.2.2 the result follows.

Note that since $\mathcal{I}$ is a partition of the set N of all nodes of a network , we can say that

$$\sum_{I:\mathcal{I}} \#I = \#N.$$

This helps us make a quantitative study involving interference sets in Section 5.3.

5.2.6 Proposition. Given the network G(L,N,W,s) as described in Section 2.3.1, if the sink s belongs to the interference set I_s, then I_s is a singleton. In other words, s occupies its own interference set.

Proof

Let s_1 be another node in I_s. We want to show that $s_1 = s$. Since $d(s) = 0$, it follows that $d(s_1) = 0$ due to the fact that s and s_1 are in the same interference set. On the other hand $d(s_1)$ is the distance between s and s_1. Hence $s = s_1$ because d is a metric (see Section 2.3.2). So I_s is a singleton.

Since the structure of interference sets is known, they can be computed using the following operation:

$$\begin{array}{|l}
\multicolumn{1}{l}{\mathit{IAlg}} \\
\hline
\mathit{net?} : \mathit{NT} \\
\mathit{Isets!} : \mathbb{P}\mathcal{I} \\
\hline
[\bigcup I : \mathit{Isets!}] = \mathit{net?.nodes} \\
\hline
\end{array}$$

The operation IAlg takes the network $\mathit{net?}$ as input and output a set of interference sets (of type $\mathcal{I}$) with a condition that the union of all formed interference sets is equal to the set of all nodes of the network.

5.3 Interference diffusion

Based on epidemic disease transmission, we propose a model for Interference diffusion in a network when LIBA$^+$ is being used. In this work, the nodes are considered to be distributed in three interference-related states and we need two thresholds T_1 and T_2 to specify susceptible, attacked or removed nodes, as shown by Figure 5.5.

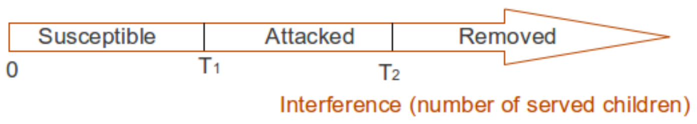


Figure 5.5: Thresholds for interference states subdivision.

Considering the network $nt : NT$ Figure 5.5 enables us to define the considered states as follows:

1. **Susceptible Nodes:** are the nodes less or not interfering in a network, and their total number is denoted by S. Each susceptible node is assumed to have weight less than the threshold T_1. That is
$$S = \#\{n : nt.nodes \mid n.wip < T_1\}.$$
2. **Attacked nodes:** are the highly interfering nodes, but still being able to operate. The total number of attacked nodes in a network is denoted by A. An attacked node is assumed to have weight less than the threshold T_2 but at least to the threshold T_1. That is
$$A = \#\{n : nt.nodes \mid T_1 \leq n.wip < T_2\}.$$
3. **Infection of nodes:** are the nodes which are no longer working because of the high interference of themselves or their neighbours. The nodes are refereed to as dead nodes and their total number is denoted by R. A node is considered to be removed if its interference is at least the threshold T_2. That is
$$R = \#\{n : nt.nodes \mid T_2 \leq n.wip\}$$

The beaconing processes [3] show that each infected node causes infection or death to a particular group of nodes in the network. This is why the proposed model will be the **compartmental model** (model involving disjoints groups) where nodes in the same group (interference set) have the same behaviours on infection or death.

In this case a node is said to be attacked if its interference is high, and its considered to be removed if its interference if very high.

We grouped the network nodes in groups called interference sets as described in Section 5.2.

We use Figure 5.7 to show a picture of the cases considered and this will enable us to clarify the assumptions made.

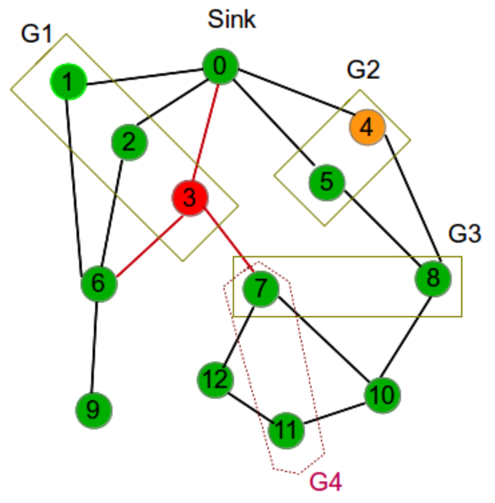


Figure 5.6: Interference transmission in the interference sets of a network.

Figure 5.7 shows a network partitioned in interference sets $G1$, $G2$, $G3$ and singletons. Node 4 is infected and node 3 has died. Note that death may be caused by infection or any other reason. The node 5 which is in the same interference set as node 4 is susceptible to get the infection caused by its infected interference set mate (node 4).

Since node 3 is not working (dead), all adjacent links can not work. So the link $(3, 7)$ is not working and hence it is considered to be removed. This causes node 7 to change its group membership: an interference set consisting of the singleton 11 becomes $G4$ which consists of nodes 11 and 7. Note here that node 7 could be susceptible or infected.

5.4 The proposed diffusion model

Nodes are distributed in interference sets, where nodes in the same interference set are assumed to be infectiously similar to each other and those in different interference sets behave differently. Each interference set i contains a set of nodes which are normally working and not yet affected whose size is denoted by S_i and this set is referred to as a set of **susceptible nodes**. The interference set i may also contain the set of size A_i which represents the set of **attacked nodes** in i and finally the set i includes the set of size R_i containing the **removed nodes** from the set i. Interference set i in which $A_i \neq 0$ or $R_i \neq 0$ is said to be the **attacked set**. The following figure shows the possible migration rates of nodes in a network partitioned into m interference sets.

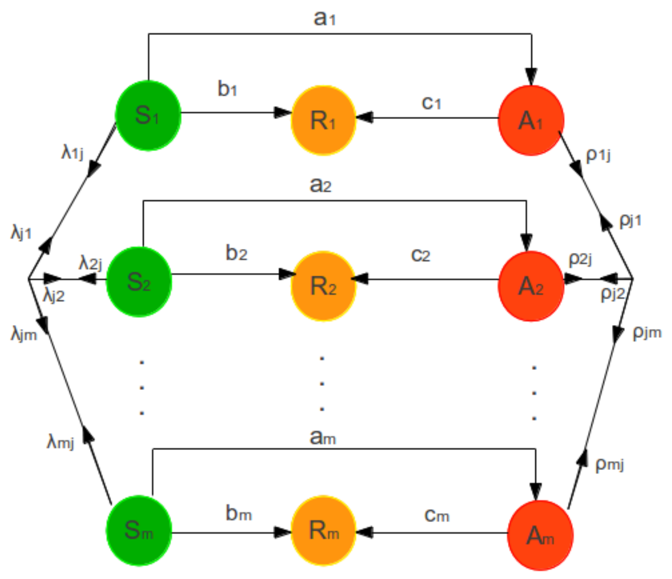


Figure 5.7: Representation of interference transmission.

Susceptible nodes in the interference set i may be attacked (infected) at the rate a_i, whereas attacked nodes from i get removed at the rate c_i.

Susceptible nodes in the interference set i may highly increase their interference levels so as to be directly removed without being considered as attacked. On the other hand, removed nodes may cause some of the susceptible or attacked nodes to leave the network because of the destruction of links. We consider bi to be the rate with which susceptible nodes in i are removed.

Removed nodes cause migration of nodes from one interference set to an other. Susceptible nodes from interference set i migrate to interference set j with the rate λ_{ij}, and attacked nodes in i migrate to j with the rate ρ_{ij}

5.4.1 Analytical description. We consider the fact that the changes in S_i, A_i and R_i are due to positive rates (rates of increase) and negative rates (rates of decrease). Taking account of all the cases discussed in Section 5.4, we add all changes in each quantity to get the following difference equation.

$$\begin{cases} S_i' = -a_i S_i + \sum\limits_{j\neq i} \lambda_{ji}\, S_j - \sum\limits_{j\neq i} \lambda_{ij}\, S_i - b_i S_i \\ A_i' = a_i S_i + \sum\limits_{j\neq i} \rho_{ji} A_j - \sum\limits_{j\neq i} \rho_{ij} A_i - c_i A_i \\ R_i' = b_i S_i + c_i A_i \end{cases} \tag{5.4.1}$$

Note that S_i, A_i and R_i are functions of time t for each interference set i.

The parameter a_i stands for the transmission rate between susceptible and attacked nodes. This parameter depends directly on the number of susceptible nodes S_i and the attacked ones A_i. This is why it makes sense to say that a_i relates to two other measures:

1. The susceptibility rate of each node in the interference group i which is denoted by β_i.
2. Infectiousness rate of nodes in the attacked interference set i denoted by γ_i.

On the other hand the structure of an interference set clearly influences the attack ability, since different interference sets have not the same infection effects. We use the parameter η_i for the measure of the structure impact when a node gets infected.

So to compute a_i, we use the formula

$$a_i = \beta_i \gamma_i \eta_i \frac{A_i}{N} \tag{5.4.2}$$

where $\frac{A_i}{N}$ denotes the fraction of attacked nodes in interference set i .

Using the Equations 5.4.2 and in 5.4.1, we get the following equations.

$$\begin{cases} S_i' = -\beta_i \gamma_i \eta_i \frac{A_i}{N} S_i + \sum\limits_{j \neq i} \lambda_{ji}\, S_j - \sum\limits_{j \neq i} \lambda_{ij}\, S_i - b_i S_i \\ A_i' = \beta_i \gamma_i \eta_i \frac{A_i}{N} S_i + \sum\limits_{j \neq i} \rho_{ji} A_j - \sum\limits_{j \neq i} \rho_{ij} A_i - c_i A_i \\ R_i' = b_i S_i + c_i A_i \end{cases} \tag{5.4.3}$$

5.5 Model assumption

1. We consider all assumptions in Section 3.6.
2. We assume that there is no node newly joining the network. That is, at any time t, if N is the number of nodes at time $t = 0$, then $N = \sum\limits_i S_i(t) \; + \; \sum\limits_i A_i(t) \; + \; \sum\limits_i R_i(t)$.
3. The death (not caused by interference) and birth rate are assumed to be zero.
4. We assume that the rates in Equation 5.4.3 are constant.
5. The removal of a node could cause new interference sets formation. We abstract this and consider a case were interference sets are only changed either by removing or adding another node (that is the number of interference sets is constant).

5.6 Stability analysis

In this section, we study the stability of the system at the disease-free equilibrium points. We first compute the disease-free equilibrium of the system which will be used to compute the basic reproduction number R_0 which is the number used for studying the stability.

5.6.1 Disease-free equilibrium (DFE). Consider equation 5.4.3 and assume the sets of the network interference sets in $\mathcal{I}$ whose size is m. Since the system is not affected by the number of removed nodes R, the equation of R is omitted. We present DFE as $E = (e_1, e_2, \cdots, e_{2m}) = (S_i, A_i = 0)\;, i = 1, 2, ..., m$, which verifies the equations

$$\forall\, i : \mathcal{I} \bullet \beta_i \gamma_i \eta_i \frac{A_i}{N} S_i + \sum_{j \neq i} \lambda_{ji}\, S_j - \sum_{j \neq i} \lambda_{ij}\, S_i - b_i S_i = 0. \tag{5.6.1}$$

Since $A_i = 0$, Equation 5.6.1 is reduced to:

$$\forall\, i : \mathcal{I} \bullet \sum_{j \neq i} \lambda_{ji}\, S_j - \sum_{j \neq i} \lambda_{ij}\, S_i - b_i S_i = 0. \tag{5.6.2}$$

Since the system of equations (5.6.2) is linear, it can be written in matrix form

$$SA = 0 \tag{5.6.3}$$

where, $S = (S_1 S_2 \cdots S_m)$ and $A = \begin{pmatrix} -\sum\limits_{j\neq 1} \lambda_{1j} - b_1 & \lambda_{12} & \cdots & \lambda_{1m} \\ \lambda_{21} & -\sum\limits_{j\neq 2} \lambda_{2j} - b_2 & \cdots & \lambda_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ \lambda_{m1} & \lambda_{m2} & \cdots & -\sum\limits_{j\neq m} \lambda_{mj} - b_m \end{pmatrix}$

Case1: If $detA \neq 0$ then $S_i = 0,\ i = 1, 2, \cdots m$ is the unique solution of Equation (5.6.3). In this case the DFE is $E_0 = (S_i^* = 0, A_i^* = 0)$.

Case2: If $detA = 0$ then the system of equations (5.6.3) has infinitely many solutions, and thus the system will have infinite number of DFE whose form is $E = (S_i^*, A_i^* = 0)$ where S_i^* may not all be zero.

Note that according to Linear Algebra, $det(A) = 0$ if and only if the rows or columns of A are linearly **dependent**. This can help us to study the dependency of interference sets in terms of interference transmission.

5.6.2 Stability of a network at DFE. We study stability using the basic reproduction number R_0. We calculate R_0 using the next-generation matrix approach as described in Section 2.2.2.

After removing the equations for R_i let us decompose the remaining systems of equation (5.6.2) into two subsystems as follows.

$$\mathcal{F}_i(S_i, A_i) = \begin{cases} 0 \\ \beta_i \gamma_i \eta_i \frac{A_i}{N} S_i \end{cases} \tag{5.6.4}$$

$$\mathcal{V}_i(S_i, A_i) = \begin{cases} \beta_i \gamma_i \eta_i \frac{A_i}{N} S_i - \sum\limits_{j\neq i} \lambda_{ji} S_j + \sum\limits_{j\neq i} \lambda_{ij} S_i + b_i S_i \\ -\sum\limits_{j\neq i} \rho_{ji} A_j + \sum\limits_{j\neq i} \rho_{ij} A_i + c_i A_i \end{cases} \tag{5.6.5}$$

The next-generation matrix is $K = FV^{-1}$ where F and V are the Jacobian matrices of $\mathcal{F}$ and $\mathcal{V}$ respectively, evaluated at the DFE.

Case1: If $detA \neq 0$ the DFE is $E_0 = (S_i^* = 0, A_i^* = 0)$. and the Jacobian of $\mathcal{F}$ evaluated at E_0 is $F_{ij}(E_0) = \frac{\partial \mathcal{F}_i}{\partial e_j}(E_0)$ is the zero matrix.

Consequently, the matrix $K = FV^{-1}$ is the zero matrix. The eigenvalues of the matrix K are all zero and hence the basic reproductive number is $R_0 = 0$. Since $R_0 < 1$, the DFE E_0 is globally stable. This is explained by the fact that at E_0 the network is empty and will remain empty because no new nodes join it.

Case2: If $detA = 0$ then the system of equations (5.6.3) has more than one solution, and thus the system will have more than one DFE point whose form is $E = (S_i^*, A_i^* = 0)$ where S_i^* may not all be zero.

$$F = \begin{pmatrix} \beta_1\gamma_1\eta_1\frac{S_1^*}{N} & 0 & \cdots & 0 \\ 0 & \beta_2\gamma_2\eta_2\frac{S_2^*}{N} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \beta_m\gamma_m\eta_m\frac{S_m^*}{N} \end{pmatrix} = [\delta_{ij}(\beta_i\gamma_i\eta_i\frac{S_j^*}{N})]_{ij}$$

$$V = \begin{pmatrix} \sum\limits_{j\neq 1}\rho_{1j} + c_1 & 0 & \cdots & 0 \\ 0 & \sum\limits_{j\neq 2}\rho_{2j} + c_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sum\limits_{j\neq m}\rho_{mj} + c_m \end{pmatrix} = [\delta_{ij}(\sum_{j\neq 1}\rho_{1j} + c_i)]_{ij}$$

where δ_{ij} is the Kronecker delta. That is $\delta_{ij} = (i = j)$.

$$K = FV^{-1} = [(\frac{\beta_i\gamma_i\eta_i\frac{S_j^*}{N}}{\sum\limits_{j\neq i}\rho_{ij} + c_i})\delta_{ij}]_{ij}$$

Since K is a diagonal matrix, the basic reproduction number is

$$R_0 = Trace(K) = \sum_{i=1}^{m}\frac{\beta_i\gamma_i\eta_i\frac{S_i^*}{N}}{\sum\limits_{j\neq i}\rho_{ij} + c_i}.$$

5.7 Numerical results

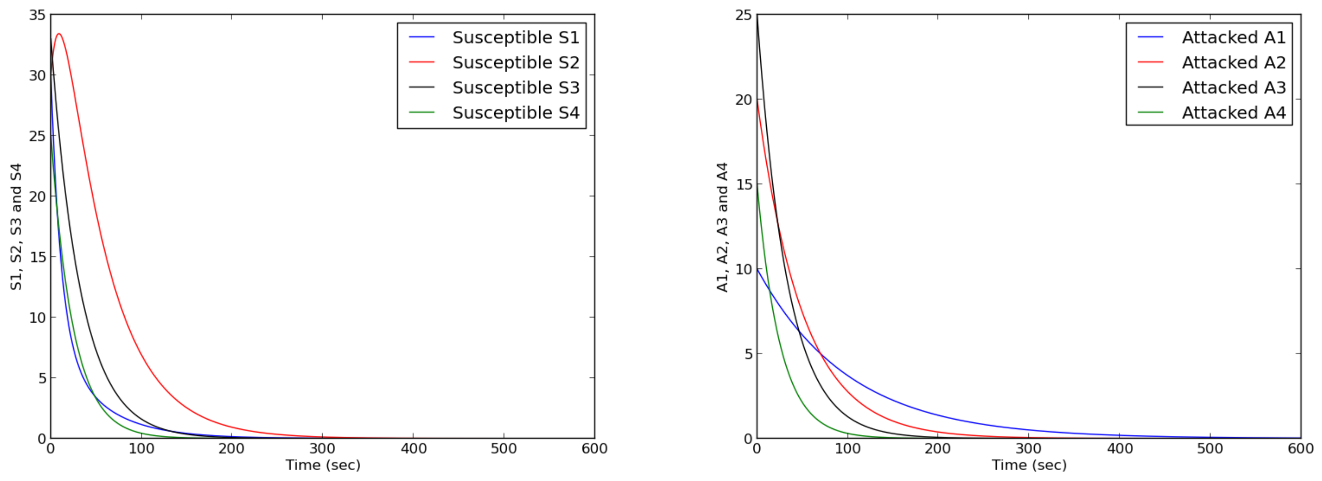
Table 5.1 shows the initial conditions of the system and also the values of considered parameters. It enabled us to use the Euler method (see [24]) to solve numerically Equation (5.4.1). The PYTHON package called PYLAB has been used in simulation (numerical computation and plotting of related graph) where the solution has been computed with 5999 iterations.

5.7.1 Considered network. As shown by Table 5.1, we considered a network whose nodes are mainly elements of four interference sets. We assume that interference transmission can mostly be observed in the four chosen interference sets.

The considered network has 200 nodes, 193 of which are elements of four considered interference sets. Assumed parameters and initial values are shown in Table 5.1.

Parameter	Description	Value
N	Total number of nodes of a network	200
m	Number of chosen interference sets	4
S_i^0	Initial number of susceptible nodes in the set i	$S_1^0 = 30,\ S_2^0 = 30\ ,\ S_3^0 = 33,\ S_4^0 = 35$
A_i^0	Initial number of attacked nodes in the set i	$A_1^0 = 10,\ A_2^0 = 20\ ,\ A_3^0 = 25,\ A_4^0 = 25$
R_i^0	Initial number of removed nodes in the set i	$R_1^0 = 0,\ R_2^0 = 0\ ,\ R_3^0 = 5,\ R_4^0 = 0$
λ_{ij}	Transmission rate from interference set i to j	$\lambda_{12} = 0.04,\ \lambda_{21} = 0.03,\ \lambda_{ij} = 0,$ with $i > 2$ or $j > 2$
ρ_{ij}	Transmission rate from interference set i to j	$\rho_{12} = 0.01 = \rho_{21},\ \rho_{ij} = 0,$ with $i > 2$ or $j > 2$
b_i	Migration rate from susceptible nodes in interference set i to attacked nodes in i	$b_1 = 0.01,\ b_2 = 0.02,\ b_3 = 0.03,\ b_4 = 0.04$
c_i	Migration rate from attacked nodes in interference set i to removed nodes in i	$c_1 = 0.02,\ c_2 = 0.02,\ c_3 = 0.03,\ c_4 = 0.04$
β_i	Susceptibility of a node in interference set i	$\beta_1 = 0.11,\ \beta_2 = 0.1,\ \beta_3 = 0.2,\ \beta_4 = 0.3$
γ_i	Infectiousness of a node in interference set i	$\gamma_1 = 0.4,\ \gamma_2 = 0.4,\ \gamma_3 = 0.5,\ \gamma_4 = 0.6$
η_i	Network impact if a susceptible node in interference set i becomes infected	$\eta_1 = 0.4,\ \eta_2 = 0.4,\ \eta_3 = 0.5,\ \eta_4 = 0.6$

Table 5.1: Numerical values.



(a) Susceptible nodes in Sets $1\ ,2\ ,3$ and 4.

(b) Attacked nodes in Sets $1\ ,2\ ,3$ and 4.

Figure 5.8: States comparison.

According to Figure 5.8a, the number of susceptible nodes in interference set 2 first increases and after few seconds, it decreases towards zero. In all other interference sets the number of susceptible nodes decreases immediately towards zero. The number of susceptible nodes in interference set 3 is decreasing faster than in all other interference sets.

Figure 5.8b shows that in both interference sets the number of attacked nodes is a decreasing function towards zero. The number of attacked nodes in interference set 3 decreases faster whereas that in interference set 1 decreases slower than other interference sets.

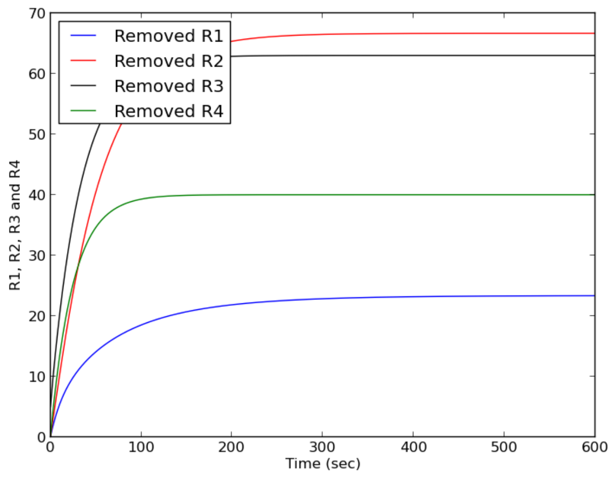


Figure 5.9: Removed nodes in Sets 1 , 2 , 3 and 4.

Figure 5.9 shows that the number of removed nodes in the four interference sets increases until it tends to a non-zero value. The figure shows that the number of removed nodes tends. Note that the removed nodes in the interference set 1 does not tend to the total number of nodes of in the set because some of them migrate to interference set 2 (see Table 5.1).

5.8 Conclusion

In this chapter, interference set is a new structure of nodes in a network and the understanding of the structure enabled us to study the diffusion model. We have presented an SIR model describing the diffusion of interference when $LIBA$ is being used by the network.

If the interference sets behave differently (linearly independent), all nodes will tend to be used until they become all old. On the other hand, if the behaviours of interference sets are related (linearly dependent), then the network tends to be destroyed where some nodes will leave the network not because they became old, but because other nodes leave the network.

Numerical results show that the number of susceptible and attacked nodes tends to zero and the corresponding cost have the same trend whereas the expected cost of removed nodes tends to a non-zero value. This shows that the use of LIBA needs a certain control to replace or repair old nodes.

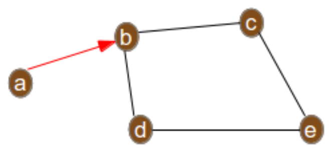
6. Least Interference Beaconing Algorithm for Multi-sink Networks: LIBAMN

As described in Chapter 4, LIBA$^+$ assumes a network having unique sink. However, more practically, a network might have several sinks. In this chapter we provide a new version of LIBA$^+$ called LIBAMN (Least Interference Beaconing Algorithms for multi-sink Networks) which supports a network with more than one sink. The sinks are refereed to as originators of beacons. Data structures discussed in Section 3.3 are modified by considering the originator addresses and their routing impacts. Each node of the network may be the sink, and this enables us to say that LIBAMN can be used for a network with k sinks ($k : \mathbb{N}$).

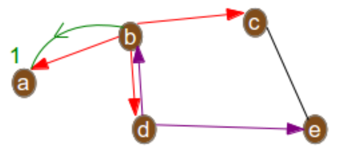
6.1 Routing mechanism

In this section we show how in a multi-sink network nodes interact to find routes to each sink. We extend beaconing messages to include their originators and this enables nodes to know beacons to reject or to consider.

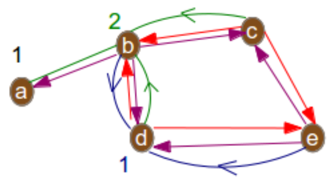
We use Figure 6.1 to introduce the algorithm. In the figure, we consider a network has two sinks and node weights are all initially zero.



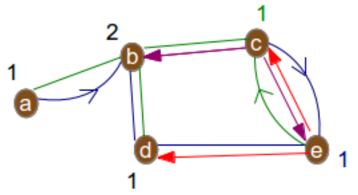
(a) Beaconing initiation at a.



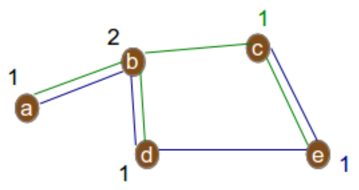
(b) New beaconing initiation at d.



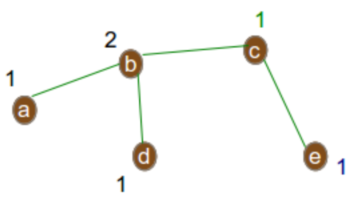
(c) Acknowledgement and relaying.



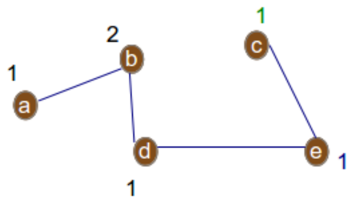
(d) Acknowledgement and relaying.



(e) Resulting trees.



(f) Resulting tree rooted at a.



(g) Resulting tree rooted at d.

Figure 6.1: Routing with LIBAMN.

Figure 6.1a shows sink a initiating beaconing by sending a beacon to its neighbour.

In Figure 6.1b the node b receives the beacon from a and relays it in the network as well as acknowledging the sink a as its parent. In addition, the figure shows that a new sink d starts a new beaconing process by also flooding a beacon in the network. Since node a has received one acknowledgement message, it has to increment by one its weight to be 1 (computed as $0 + 1$).

Figure 6.1c shows the nodes b, d and e, acknowledging their parents and relaying beacons where acknowledged nodes compute their weights. In fact the node b relays the beacon from d because it is new (it is coming from a different sink) and comes from the chosen parent. The figure shows that nodes b and d acknowledge each other because each one of them has sent a new beacon to other. Since the network supports two different beacons, all nodes in the network handle two beacons (from different sinks) and acknowledge the chosen parent. Note that a parent is chosen because it has a least weight (interference).

Figure 6.1d shows nodes ending the parent choice process. We emphasise that nodes make their weights global so that those weights can be updated by any acknowledgement message or even can be used to update beacons. This is why the node c could not chose node b to be its parent: the weight of b is 2 and bigger than that of e which is still zero.

Figure 6.1e shows the established routes and to separate them, Figures 6.1f and 6.1g show the resulting trees routed at a and d respectively.

6.2 Abstractions and assumptions

- We abstract all optional processes such us repairing and mobility.
- We assume that all nodes are well working and we do not model any probabilistic issue.
- We assume that no node is better than any other.
- We assume that each node receives messages from all its parents with minimal number of hops to the sink s.

6.3 Data structures

Each beacon now contains the address of its originator (sink). Hence routing tables and other structures change. In this section we use Z notation calculus as discussed in [8] to make changes to the different structures discussed in 3.3. The types of observables useful during execution of the protocol are described and the operation used are identified. Definitions are grouped according to the various aspects of LIBAMN and the host network.

We refer to Section 3.3 and the new aspects of LIBAMN to identify the following data types:

▶ As described in Section 3.3, the set of node identifiers is IP whereas the set of sequence numbers is SQN.

▶ **Beacons** BC'**:**

A beacon is a message broadcast periodically from a sink, and relayed in a network enabling all nodes of the network to select their (good) parents. A beacon contains the address of its originator oip, its sender sip, the sequence number of the originator osn (a natural number which identifies an originator of a beacon and the beacon itself), the hop count hop which helps to determine which nodes are candidate to be parent (nodes at distance minimal to the sink) and the weight sw of the sender. The type of all beacons is denoted by BC'. The schema describing a beacon of type BC' adopts the schema BC in Section 3.3 as follows:

$$\begin{array}{|l}
\multicolumn{1}{l}{BC'[IP]} \\
\hline
BC[osn/bsn] \\
oip : IP \\
\hline
oip = sip \Leftrightarrow hop = 0 \\
\hline
\end{array}$$

The type BC was modified by renaming the sequence number bsn by osn and including the address of the originator oip. Note that the origination of a beacon is the same as its sender if and only if the number of hops is equal to zero.

- **Acknowledgement messages AC':**

  An acknowledgement message is a message sent by a node to its newly selected parent to confirm the selection. It is considered as a response to some beacons it has received (beacons from the same originator with the same sequence number and sent by nodes nearest to the sink). It contains the originator oip of the beacons to which it is responding, its sender sip and the destination dip. The type of all acknowledgement messages is denoted by AC'. The following is the schema for an acknowledgement message.

$$\begin{array}{|l}
\multicolumn{1}{l}{AC'[IP]} \\
\hline
AC \\
oip : IP \\
\hline
sip \neq oip \\
\hline
\end{array}$$

  As shown, the schema AC (3.3) was used to describe the acknowledgement message for LIBAMN. The message AC' is formed by adding the observable oip to schema BC with the condition that the originator oip of the beacon last handled by the sender sip is the same as the sender sip if and only if the the number of hops is zero.

- **LIBAMN messages:**

  In LIBP any message is either a beacon or an acknowledgement. We describe the set of all messages MSG by the following schema.

$$\begin{array}{|l}
\multicolumn{1}{l}{MSG} \\
\hline
bc : \mathbb{P} BC' \\
ac : \mathbb{P} AC' \\
\hline
\#ac \leq \#bc \\
\hline
\end{array}$$

  Since a beacon is broadcast and an acknowledgement message is unicast, the number of received beacons is always at least the number of received acknowledgements.

Referring to the entries of the routing table, further types are described as follows :

- The number of hops $hops$ is the natural number which is equal to the distance of a node from a sink. A node has to increment by one the number $hops$ before relaying the received beacon in a network. Note that the number of hops of each node depends on a particular sink.

- The type of routing table entries is a table storing information about a route to the sink. It contains the address oip of the originator (initiator of a beacon which made the entry to be created), the parent pip and children cip of the host node and finally the sequence number osn of the originator which is a natural number measuring freshness of information from the originator oip. The type of routing table entries is denoted by R' and it is built by modifying the schema R (explained in 3.3) as follows:

$$\begin{array}{|l}
\multicolumn{1}{l}{R'[IP]} \\
\hline
R \\
oip : IP \\
osn : \mathbb{N} \\
\hline
oip \neq cip \\
\hline
\end{array}$$

  R' is considered as the combination of the schema R and the details of the current originator which are its address oip and sequence number osn. In each routing table or type R' entry, the originator oip is different from each node in the set or children cip.

- The type of routing tables is described as the type of all routing table entries possessed by a node. The following is its representative schema:

$$\begin{array}{|l}
\multicolumn{1}{l}{RT[R]} \\
\hline
rt : \mathbb{P} R \\
\hline
\forall r_1, r_2 \in rt \bullet r_1 \neq r_2 \Leftrightarrow r_1.oip \neq r_2.oip \\
\hline
\end{array}$$

  As represented in the schema RT, each two different routing table entries have different originators.

- **Accessing the routing table contents:**

  The contents of touting tables are accessed using the following partial functions.

  We start by defining the partial function $entry$ to select an entry in a routing table rt corresponding to a given originator oip.

  $entry : RT \times IP \nrightarrow R$

  $$entry(rt, oip) := r \text{ if and only if } r \in rt \wedge r.oip = oip.$$

  This permits the selection of any content of a routing table entry as follows:

  (a) The **parent** pip of a node having a given routing table in a path towards the originator oip.
  $fpip : RT \times IP \nrightarrow IP$
  $$fpip(rt, oip) := entry(rt, oip).pip$$

  (b) The **children** cip of a node having the routing table rt in a path towards the originator oip.
  $fcip : RT \times IP \nrightarrow \mathbb{P} IP$
  $$fcip(rt, oip) := entry(rt, oip).cip$$

  (c) The **sequence number** of the lastly handled beacon from the originator oip.
  $fosn : RT \times IP \nrightarrow \mathbb{P} IP$
  $$fosn(rt, oip) := entry(rt, oip).osn$$

Updating routing tables:

There are two main ways to update a routing table which are:

- Updating an existing entry.
- Inserting new information in a routing table.

Updating an existing entry:

A routing table entry is updated in one of the following three ways.

1. **Updating the parent:** is done when a new beacon is handled by a node.
2. **Updating the children set:** It is done by empting the set of children by the incoming beacon and add new children in the list when a new acknowledgement messages is received.
3. **Changing the sequence number:** done by replacing the existing sequence number by one present in a newly received beacon.

Inserting new information in routing tables:

This is done when a beacon from an originator is newly received. The inserted entry, denoted by nr has the following representative schema.

$$\begin{array}{|l}
\hline \textit{nr} \\
R' \\
\hline
cip = \varnothing \\
pip = - \\
\hline
\end{array}$$

As shown by the schema nr, the inserted entry in has no parent and the list of its children is empty.

▶ **Node structure:**

Each node in the network is determined by its address ip, its sequence number sn, its weight wip and its set of routing tables rt. The node structure is represented by the following schema.

$$\begin{array}{|l}
\hline N'[IP, RT] \\
ip : IP \\
wip, sn : \mathbb{N} \\
rt : RT \\
\hline
\forall\, r \in rt \bullet ip \notin \{r.pip\} \cup r.cip \\
\hline
\end{array}$$

The schema N' shows that the address of a node is always different from its parent and from each of its children.

▶ **Considered network:**

It is determined by its links and nodes with their neighbourhood . As in Section 3.3.4, the weight distribution in the network can be obtained from the nodes distribution since the descriptors of a node includes its weight. A network is described using the following schema:

$$\begin{array}{|l}
\multicolumn{1}{l}{NT'[N']} \\
\hline
nodes : \mathbb{P} N' \\
links : N' \leftrightarrow N' \\
nbr : N' \rightarrow \mathbb{P} N' \\
\hline
links \in nodes \leftrightarrow nodes \\
links^{\sim} = links \\
ran(link^{*}) = nodes \\
id_{nodes} \cap linkd = \varnothing \\
nbr(n) = links(\!|\ n\ |\!) \\
\forall\, n_1, n_2 : nodes \bullet n_1.ip \neq n_2.ip \\
\hline
\end{array}$$

As shown in the schema NT', a network of type NT' is defined as a connected and bi-directed graph with no self link. The neighbourhood nbr of a node n consists of nodes connected to n. As shown by the schema, the sink is not taken care because each node is assumed to be able to behave like the sink of the network.

6.4 Beaconing process

The beaconing process starts from a sink and goes to all nodes in the same network as a sink. In contrast with beaconing in Section 3.4, the sink is not fixed. It may rather be any node in the network which is interested in starting the routing. The following are then the classes involved:

6.4.1 Class Starter. This consists of a sink node and the operation $Initiate$. The sink $sink$ of the network may be any node of the network. The operation $Initiate$ consists of initiating the beaconing message $bc!$, and the routing table rt of the sink. In an initiated beaconing message, the sender $bc!.sip$ is the sink's address $sink.ip$, the number of hops $bc!.hop$ is set to zero, and the beacon's weight $bc!.sw$ is set to the weight $sink.wip$.

The set of children $fcip(sink.rt, sink.ip)$ of the sink's routing table entry for itself is emptied and the sequence number of the sink's routing table entry for itself is incremented and is considered to be the sequence number of the initiated beacon $bc!$. Finally, the parent of the sink whose originator is the same as the sink is set to nothing (still denoted by $-$).

Starter'

snode

$nt : NT'$
$sink : N'$

$sink \in nt.nodes$

Initiate

$\Delta(fosn(sink.rt, sink.ip), fcip(sink.rt, sink.ip), fpip(sink.rt, sink.ip))$
$bc! : BC'$

$bc!.sip = sink.ip$
$bc!.hop = 0$
$bc!.sw = sink.wip$
$fcip(sink.rt, sink.ip)' = \varnothing$
$fosn(sink.rt, sink.ip)' = fosn(sink.rt, sink.ip) + 1 = bc!.osn$
$fpip(sink.rt, sink.ip)' = -$

Figure 6.2: The class $starter'$.

6.4.2 Class Transmitter'. This consists of one object $Node$ and two operations namely $bcHandle$ and $ReceiveAck$ as presented in the following schema:

$Transmitter'$

$Node$

$$
\begin{array}{l}
nt : NT' \\
n : N' \\
\hline
n \in nt.nodes
\end{array}
$$

$bcHandle$

$$
\begin{array}{l}
\Delta(n.rt) \\
bc? : \mathbb{P} BC' \\
bc! : BC' \\
ac! : AC' \\
\hline
\forall\, bc_1, bc_2 : bc? \bullet bc_1.osn = bc_2.osn \neq n.rt.sn \land bc_1.oip = bc_2.oip \\
\exists\, x : bc? \bullet x.sw = \sqcap\{bc.wip \mid bc \in bc?\} \\
\qquad\qquad \land ac!.dip = x.sip \\
\qquad\qquad \land fpip(n.rt', bc?.oip) = x.sip \\
\qquad\qquad \land bc!.hop = x.hop + 1 \\
\qquad\qquad \land fosn(n.rt', bc?.oip) = x.sn \\
ac!.sip = bc!.sip = n.ip \\
bc!.sw = n.wip \\
fcip(n.rt', bc?.oip) = \varnothing \\
fpip(n.rt', bc?.oip) = ac!.dip \\
(bc?sip, n.ip) \in nt.liks
\end{array}
$$

$ReceiveAck$

$$
\begin{array}{l}
\Delta(n.wip, n.rt) \\
ac? : AC' \\
\hline
n.ip = ac?.dip \\
n.wip' = n.wip + 1 \\
fcip(n.rt', ac?.oip) = fcip(n.rt, ac?.oip) \cup \{ac?.sip\} \\
(ac?.sip, ac?.dip) \in nt.liks
\end{array}
$$

Figure 6.3: The class $Transmiter'$.

Node: It is a node in a network of type NT'.

bcHandle: This is an operation takes a set of beaconing messages $bc?$, makes changes of the routing table rt of the beacon receiver and outputs the new transformed beacon $bc!$ and acknowledgement message $ac!$.

> To handle the set of beacons requires them to have the same sequence number and originator, and the receiver n handles them if it has a sequence number different from that of the incoming beacon. This is to guarantee the fact that the handled beacons are the new ones.
>
> The node n chooses the beacon x of the minimal weight and uses it to updates its routing table as well as the acknowledgement message $ac!$, the new beacon $bc!$ to send in in the network.
>
> The destination dip of the acknowledgement message $ac!$ becomes the sender of the beacon x and hence the parent $fpip(n.rt, bc?.oip)$ of the node n. The number of hops in the beacon $bc!$ is obtained by incrementing by one the number of hops of the beacon x whereas the sequence number $fsn(n.rt, bc!.oip)$ is a copy of that of the beacon x.
>
> The sender of the beacon $bc!$ and the acknowledgement message $ac!$ is the node n and the weight in the beacon $bc!$ is the same as the weight $n.wip$ of the node n.
>
> The routing table of the node n for the entry of the originator changes in a way that the set of children $fcip(n.rt, bc!.oip)$ becomes empty, the parent $fpip(n.rt, bc!.oip)$ becomes the destination of the acknowledgement from n.
>
> Note that all above changes apply to the node n if the sender of the beacon is directly connected to it. That is $(ac?.sip, ac?.dip) \in nt.liks$ (last predicate in Figure 6.3).

ReceiveAck: It is an operation which takes the acknowledgement message $ac?$ and changes the the state of its receiver n, provided that the receiver's address $n.ip$ is the same as the destination address $ac?.dip$ specified in $ac!$.

The weight $n.wip$ of n is incremented by one and the sender of $ac?$ becomes one of the children of n. All this happens if the sender and the destination of the acknowledgement message are connected i.e $(ac?.sip, ac?.dip) \in nt.liks$.

6.4.3 Class $LIBAMN$**.** This class consists of processors namely $starts$ and *transmitters* as its objects. The starter $stat$ is the sink in the network and the transmitters are those nodes in the network which handle the beaconing messages or acknowledgement messages.

$LIBAMN$

$processors$

$starts : \mathbb{P} Starter' \setminus \varnothing$

$transmitters : \mathbb{P} Transmitter'$

$Generate \mathrel{\widehat{=}} [[]s : starts]s.Initiate \gg [p : Starter'.snode.sink.nbr] \bullet p.bcHandle$

$Trans \mathrel{\widehat{=}} [\gg p : transmitters] \bullet p.bcHandle[]p.ReceiveAck$

Figure 6.4: Beaconing class.

$LIBAMN$ consists of processors namely $starts$ and *transmitters*. The set $stats$ is consists of all sinks in the network and the transmitters are nodes in the network which handle the beaconing or acknowledgement messages.

The operation $Generate$ consists of initiating the beacon and this is followed by handling the initiated message by the neighbours of the sink which initiated beaconing. The operation $Trans$ consists of sequentially handling beacons or acknowledgement messages by all concerned nodes in the network.

6.5 Example

6.5.1 Notations. As in Section 4.3.1, messages are expressed in tuples. Instead of presenting the whole routing table, we show only its involved entries in tuples. We explain the used tuples as follows:

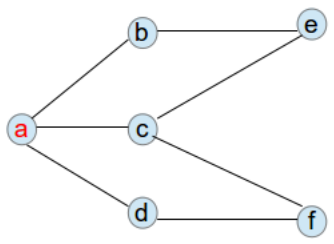
A beacon is represented by $bc(oip, sip, osn, hop, sw)$ which expresses the beacon bc from the originator oip, sent by the sender sip of sequence number osn where the sender is at the hop^{th} hop and its weight is sw.

An acknowledgement message ac is expressed as $ac(oip, sip, dip)$ where oip is the originator of the acknowledged beacon, and the nodes sip and dip are the sender and destination of the acknowledgement message respectively.

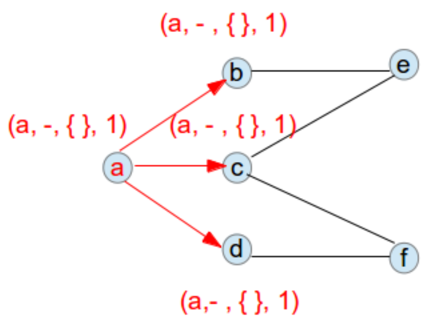
A routing table is considered to be the set of entries corresponding to each originator. It looks like (oip, pip, cip, osn), where the node having such a routing table has received a beacon originated from oip and confirmed that the node pip is its parent. The set cip contains all its children of the node having the entry in its rooting table and finally the last handled beaconing message had the sequence number osn.

6.5.2 Colouring logic. What we change from Section 4.3.2 is to give each sink its colour. We follow the same idea of colouring as in Section 4.3.2 where the change of sink (to be the node e) is associated with the change in all colours. Red is changed to **violet** and Green is changed to **blue** .

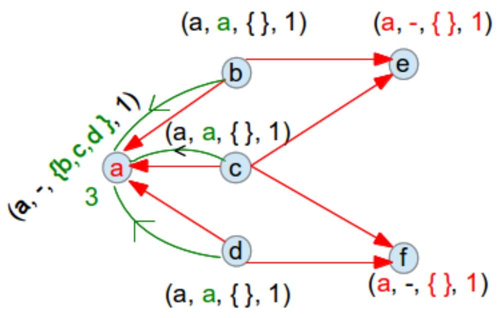
Consider the network as shown in Figure 6.5a where a is a sink. All nodes are assumed to have the wait equal to zero. We are interested on finding the routing tree using LIBAMN.



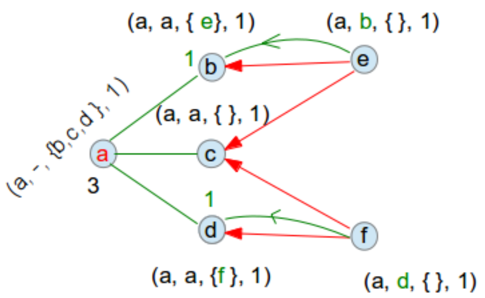
(a) Beaconing starts.



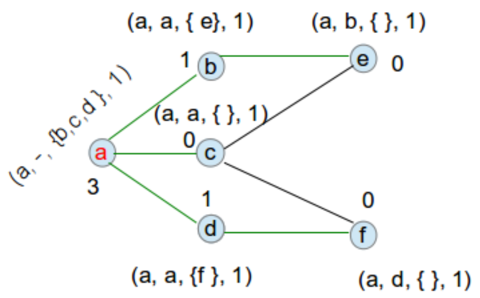
(b) Acknowledgement and relaying.



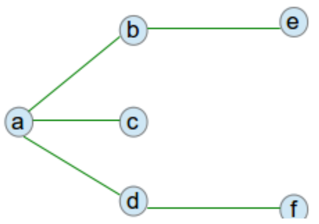
(c) Acknowledgement and relaying.



(d) Acknowledgement and relaying.



(e) Resulting network.



(f) Resulting tree.

Figure 6.5: Routing with the sink a.

As shown by Figure 6.5b, the sink a starts by initiating and broadcasting the beaconing messages in the network. While initiating, the routing table entry of the sink for itself is $(a, -, \{\}, 1)$, which means that the node a originated a beacon of sequence number 1, where the parent and children of a are not yet known. Note here that the parent of the sink remains unknown. The initiated beacon is $(a, a, 1, 0, 0)$, meaning a beacon of sequence number 1, originated from a which is sent from a where its weight and hop count from the sink equal zero.

Once the neighbours of a sink receive the beacon, they update or form the routing table entries for the originator of the beacon. This is why nodes b, c and d form the entry $(a, -, \{\}, 1)$ in their routing tables.

Figure 6.5c shows the process of relaying the beacon and acknowledging the chosen parent.

The relayed beacon is the update of one from a chosen parent. The update is done by incrementing by one the hop count hop, changing the sender and the weight to the chooser and its weight respectively. This is why the beacons from node b, c, and d are $(a, b, 1, 1, 0)$, $(a, c, 1, 1, 0)$ and $(a, d, 1, 1, 0)$ respectively.

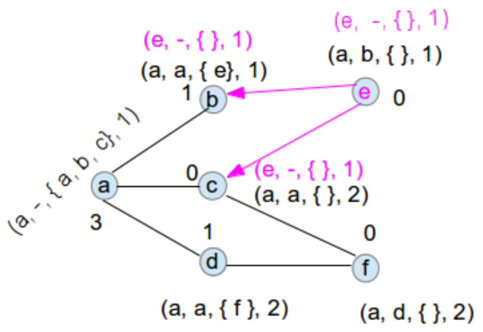
On the other hand, the the nodes a,b and c acknowledge their chosen parent (a) by sending the acknowledgement messages (a, b, a), (a, c, a) and (a, d, a) respectively. The nodes include the selected parent a in their routing tables entries, and send the acknowledgement messages to the parent.

The node a receives the acknowledgement messages and update the list of its children which becomes the set of the senders of all acknowledgement messages it has received (i.e. $\{b, c, d\}$), and its new weight is 3 which is the sum of the previous weight (which was zero) and the number of new incoming acknowledgement messages.

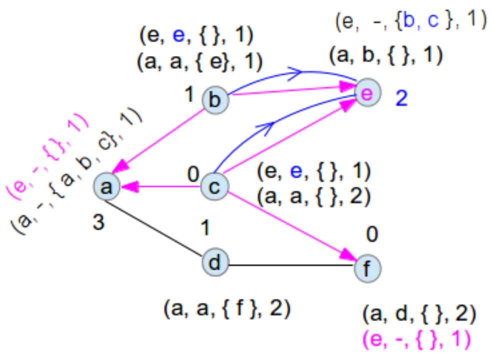
Figure 6.5d shows the nodes e and f acknowledging and sending beacons. However the beacons from the nodes do not update any routing table in the network because all nodes have already received the same beacon (the beacon from a of sequence number 1).

The route consists of the trace of the acknowledgement messages in the network as shown in Figure 6.5e. The routing tree is extracted as in Figure 6.5f.

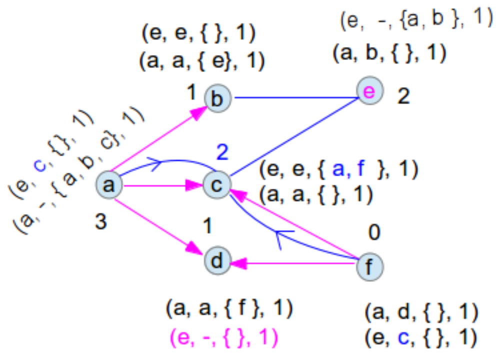
We now consider the second run of LIBAMN on the same network. This is why the considered network is the one in Figure 6.5e but we now consider the node e to be the sink.



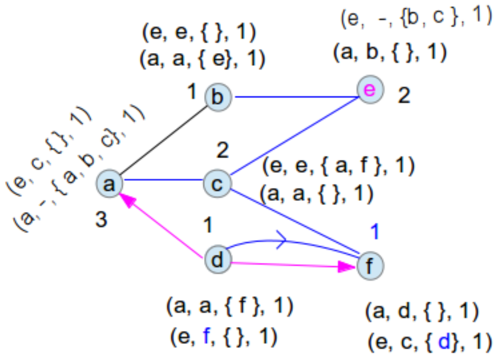
(a) Beaconing starts.



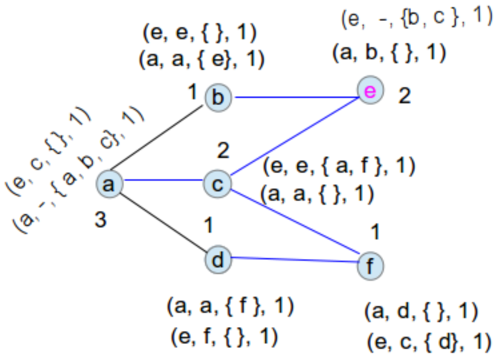
(b) Acknowledgement and relaying.



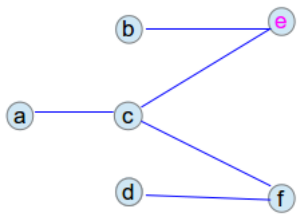
(c) Acknowledgement and relaying.



(d) Acknowledgement and relaying.



(e) Resulting network.



(f) Resulting tree.

Figure 6.6: Routing with the sink e.

Figure 6.6a shows that the sink e starts by forming the routing entry for itself and broadcasting the initialised beacon $(e, e, 1, 0, 0)$. Since the neighbours b and c did not yet receive any beacon originated from the node e with sequence number equal to 1, they form the routing table entries for e as in the previous example.

As in Figure 6.5, Figures 6.6b, 6.6c and 6.6d show how nodes update their routing tables while receiving or sending the beacons and acknowledgements.

Note that the weight of a node is incremented every time the node chooses the parent, and the originator does not matter. The computed weight is used to make future choices and this why in Figure 6.6c the node a chooses the node c instead of b since c had lower weight as shown by Figure 6.6a.

After completing the parent acknowledgement of nodes in Figure 6.6d, the established route and node weights are presented in Figure 6.6d and the extracted route is shown in Figure 6.6b.

6.6 Properties and verification

In this section, we specify LIBAMN by stating and verifying its properties. The algorithm is specified by the following four properties stated as theorems. We adopt the verified properties of LIBA$^+$ (see Section 4.4) to state and verify the properties for LIBAMN. We use CTL as described in Section 2.5 to express properties and hence verify them. We start by identifying the ways of beaconing and we compare them in the last property.

6.6.1 Ways of beaconing. In a multi-sink network, beaconing is done in the following two ways.

- **Parallel Beaconing** a node starts routing while another node is still routing. In this case, involved sinks are called **parallel sinks**. See Figure 6.1 for example.
- **Alternating Beaconing:** a node initiates beaconing only when no other node (sink) is routing. In this case, involved sinks are called **alternating sinks**. As an example, see Figures 6.5 and 6.6.

6.6.2 Parent choice and established route. We aim to verify that the chosen parent is the chooser neighbour having the following properties:

CP_1**:** Being the closest node to the sink, i.e. node at the fewest hops from the sink.

CP_2**:** Having the minimal weight which means a node of minimal interference.

On the other hand we target to verify that

PP_1**:** An established route (by LIBAMN) from each node to a given sink is **shortest**.

PP_1**:** An established route (by LIBAMN) from each node to a given sink consists of nodes of **least** interference.

As shown in Section 6.4, LIBA$^+$ has been changed by considering more details of the originator (address and sequence number). The parent choice and the weight are the same in both LIBA$^+$ and LIBAMN with alternating beaconing. This is why Theorems 6.6.4 and 6.6.5 hold.

6.6.3 properties of alternating LIBAMN..

6.6.4 Theorem. *Suppose a beacon is initiated and broadcast k rounds by a single node acting as a single sink of a network of type NT' (see Section 6.3). At each round, each node different from the sink in the same network keeps on choosing its parent to be its neighbour satisfying the properties CP_1 and CP_2, and the path from each node to the initiator satisfies the property $PP_1 \wedge PP_2$.*

6.6.5 Theorem. *Suppose a beacon is initiated and broadcast k rounds by two or more nodes acting as* **alternating** *sinks of a network of type NT' (see Section 6.3). At each round, each chooser (node acting differently from the a originator of the received beacon) keeps on choosing its parent to be its neighbour satisfying the properties CP_1 and CP_2 and in addition, the path from each node to the sink satisfies the properties PP_1 and PP_2.*

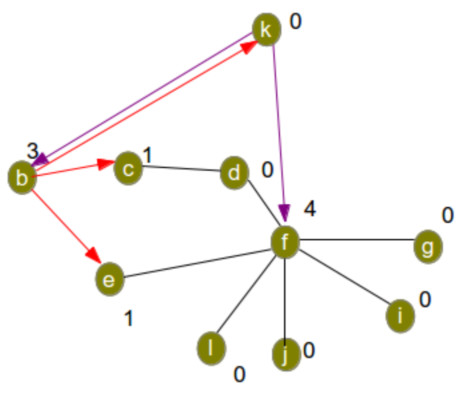
Note:

As shown in Section 6.4, LIBA$^+$ has been changed by considering more details of the originator (address and sequence number). The parent choice and the weight are the same in both LIBA$^+$ and LIBAMN with alternating beaconing. This is why Theorems 6.6.4 and 6.6.5 hold.

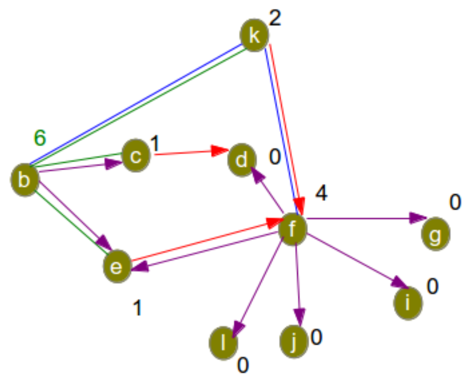
6.6.6 properties of parallel LIBAMN .

6.6.7 Proposition. Suppose a beacon is initiated and broadcast k rounds by two or more nodes acting as the **parallel** sinks of a network of type NT' (see Section 6.3). At each round, in some cases, each chooser **does not need** to keep on choosing its parent to be its neighbour satisfying the properties CP_1 and CP_2, and hence the path from each node to the initiator does not need to satisfy the properties PP_1 and PP_2.

To show this we use the following case:



(a) Beaconing starts.



(b) Acknowledgement and relaying.

Figure 6.7: Efficiency of $LIBAMN$.

Figure 6.7a shows a network of type NT' whose two sinks b and k are initiating beaconing. We assume that initially the node b had sent a beacon which produced the shown weight at each node.

Figure 6.7b shows the node relaying messages as well as confirming routes. Beacons to be dropped are ignored here. The node b updates its weight before the node f and both relay the beacon from node k to node e (see Violet arrows towards the node e). So the node f advertised the non updated weight whereas the node b advertised the updated and hence bigger, one.

This shows that the node e prefers to choose the node f since it has less weight (according to the beacons received). However, Figure 6.7a shows that node f had weight greater than that of b.

Hence a node to be chosen does not mean that it satisfies property PC_2.

6.6.8 Theorem. *Consider an arbitrary node n in a network of type NT' described in section 6.3, an increase of interference at n after parallel beaconing is at most that after alternating beaconing.*

Proof

We know that for a node n to be chosen as a parent of another node, it has first to send a beacon to the chooser containing its weight to be compared with other nodes weights which have also sent the beacons to the chooser. See $\exists\, x : bc? \bullet x.sw = \sqcap\{bc.wip \mid bc \in bc?\}$ in Figure 6.3.
Consider two nodes n and m transferring a beacon to another node say x. We assume that the beacon has been initiated after another one whose originator (sink) is different, and the weight of n is less than that of m i.e.

$$n.wip < m.wip.$$

Define the weights w_a and w_p to be the new weights of the node n in alternating and parallel beaconing respectively. We want to show $w_p \leq w_a$. For the case of alternating beaconing, the condition $\exists\, x : bc? \bullet x.sw = \sqcap\{bc.wip \mid bc \in bc?\}$ in Figure 6.3 shows that n is chosen to be the parent of x (it has minimal weight) and hence its weight becomes

$$w_a = n.wip + 1 \tag{6.6.1}$$

as shown by the predicate $n.wip' = n.wip + 1$ in Figure 6.3.
For the case of parallel beaconing, there might be a beacon which caused node n to increase its weight but not yet the node m. See Figure 6.7 as an example. We have three cases:

1. The updated weight of n is greater than the weight $m.wip$. The node will not be chosen and as a result its weight does not change.

$$w_p = n.wip \tag{6.6.2}$$

2. The updated weight of n is equal to $m.wip$. So

$$w_p = n.wip \tag{6.6.3}$$

   or

$$w_p = n.wip + 1 \tag{6.6.4}$$

   because the parent choice is random.

3. The updated weight of n is still less than $m.wip$. It follows that

$$w_p = n.wip + 1. \tag{6.6.5}$$

Equations 6.6.2, 6.6.3, 6.6.4 and 6.6.5 show that

$$w_p \leq n.wip + 1 \tag{6.6.6}$$

Using Equation 6.6.1 in 6.6.6 follows the result.

7. Experimental comparison of LIBA, LIBA$^+$ and LIBAMN

In this chapter, we compare the algorithms using four test networks on three different performance parameters:

1. **The highest accumulated interference:** It allows us to determine what nodes are highly used and how much they are being used [1] by each of the three algorithms.
2. **The interference standard deviation:** It is used to identify which algorithm is better in terms of load balancing in a network. The smaller the standard deviation, the more balanced interference in the network.
3. **Computational time:** We use this parameter to evaluate how quick are the algorithms to find one route (the first route).

7.1 The test networks

We considered four test networks grouped into two categories as follows:

- **Two randomly generated networks:** a 13-nodes (see Figure 7.1) and a 51-nodes (see Figure 7.3).
- **Two regular networks:** a 16-nodes and 25-nodes. See Figure 7.2 and 7.4 respectively.

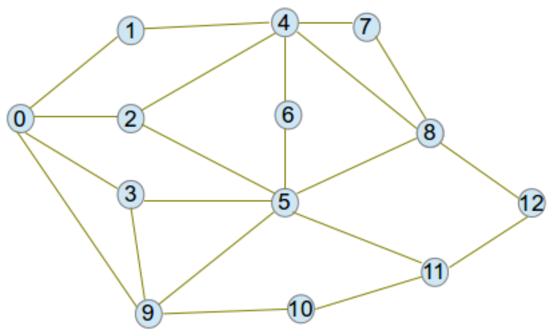


Figure 7.1: Network 1.

For a network shown by Figure 7.1, the considered sinks were nodes 0 and 12 for the LIBAMN case and for the LIBA and LIBA$^+$ cases, the considered sink was node 0.

[1]A node is used by any of the three algorithms if the algorithms updates the node's interference at least once in its process.

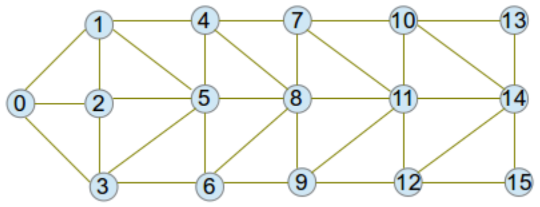


Figure 7.2: Network 2.

Figure 7.2 represents a network whose sink was node 0 for all the three algorithms and for LIBAMN the only other sink was node 15.

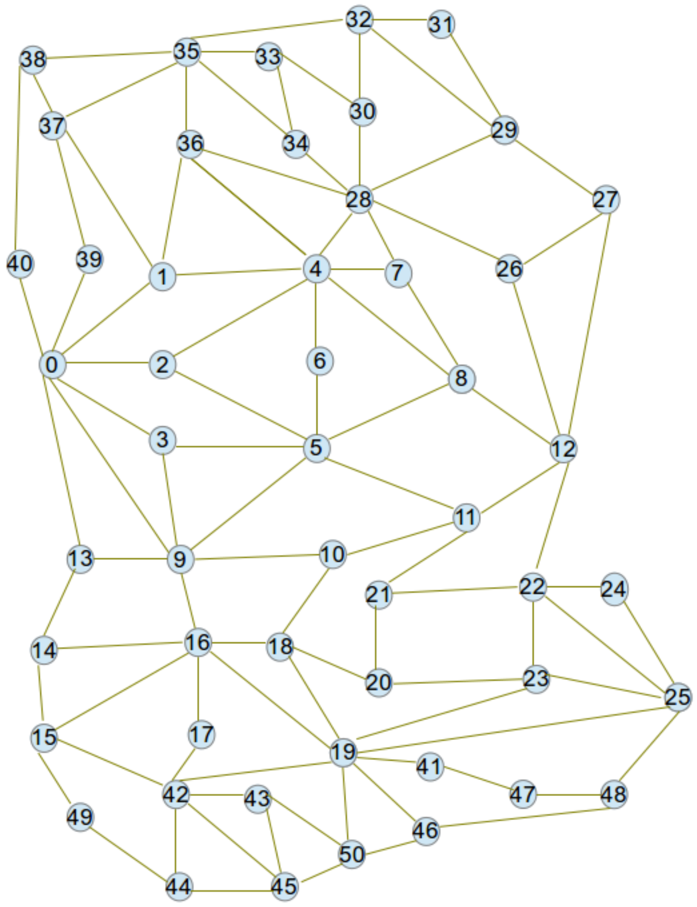


Figure 7.3: Network 3.

Figure 7.3 shows a big network of 50 nodes, where node 0 was still a sink used by all the three algorithms and node 50 was the only other sink used for the case of LIBAMN.

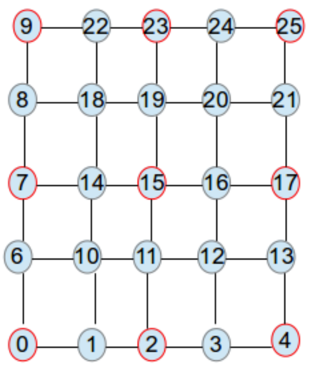


Figure 7.4: Network 4

For Figure 7.4, sinks were chosen in various ways:

– In Section 7.2, node 0 was chosen to be a sink for all the algorithms and node 25 was a second and last chosen sink for the LIBAMN case.

– In Section 7.3, sinks were chosen in nine ways summarised in Table 7.1

Case	LIBA	LIBA$^+$	LIBAMN
1	0	0	0 and 15
2	2	2	2 and 17
3	4	4	4 and 15
4	7	7	7 and 23
5	9	9	9 and 15
6	23	23	23 and 17
7	25	25	25 and 15
8	17	17	17 and 2
9	4	9	0 and 25

Table 7.1: Considered cases.

7.2 The highest accumulated interference

In this section, we used Python to perform 56 runs (iterations) of the three algorithms on all the networks described in Section 7.1. We plotted cumulated interference against nodes' names to compare nodes usability in each network by evaluating the highest cumulated interference. Since at each node we observed a local highest interference, we referred to highest interference by the global one and the local ones were referred to by their related indices (eg: second highest interference) corresponding to their decreasing order.

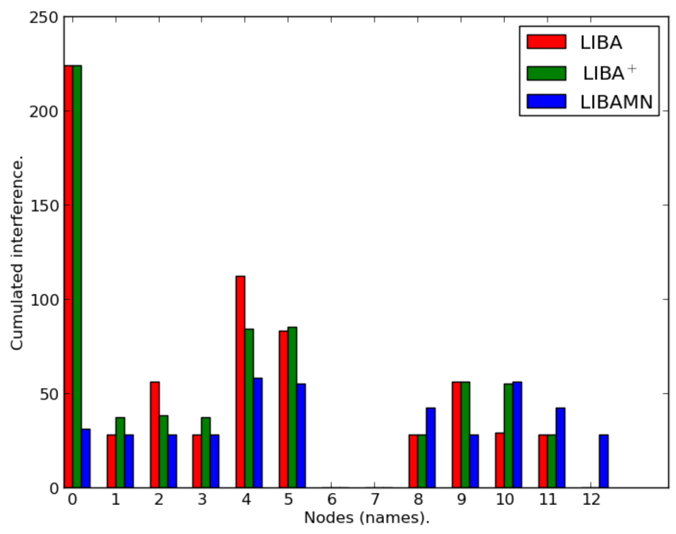


Figure 7.5: Cumulated interference for the Network 1.

As shown by Figure 7.5, for the network in Figure 7.1, LIBA and LIBA$^+$ have the same highest interference which is at node 0, the sink for both algorithms. The second highest interference is held by node 4 and corresponds to the $LIBA$. In this case it is clear that node 12 is only used by LIBAMN while no algorithm uses nodes 6 and 7 (this is indicated by the fact that interference at the nodes is equal to zero).

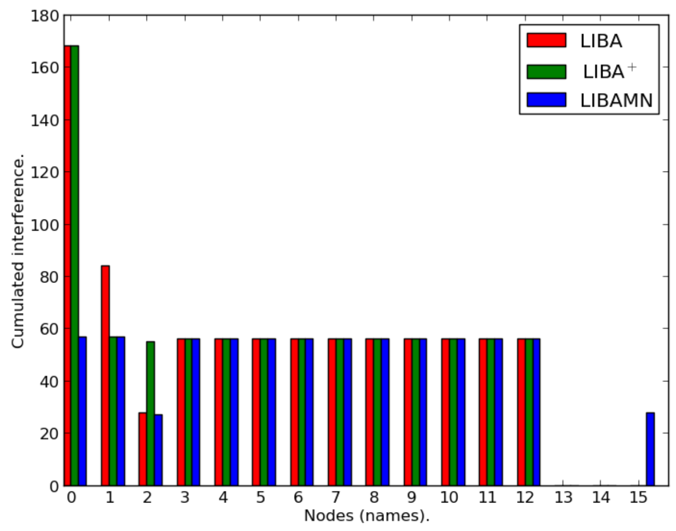


Figure 7.6: Cumulated interference for the Network 2.

The highest interference corresponds to LIBA and LIBA$^+$ and it is held by the sink for both algorithms (node 0). The second highest interference is observable at node 1 and corresponds to LIBA. Figure 7.6 also shows that the third highest interference is at many nodes of the network in Figure 7.2. Note that the nodes 13 and 14 are used in neither of the three algorithms and node 15 is used in only LIBAMN.

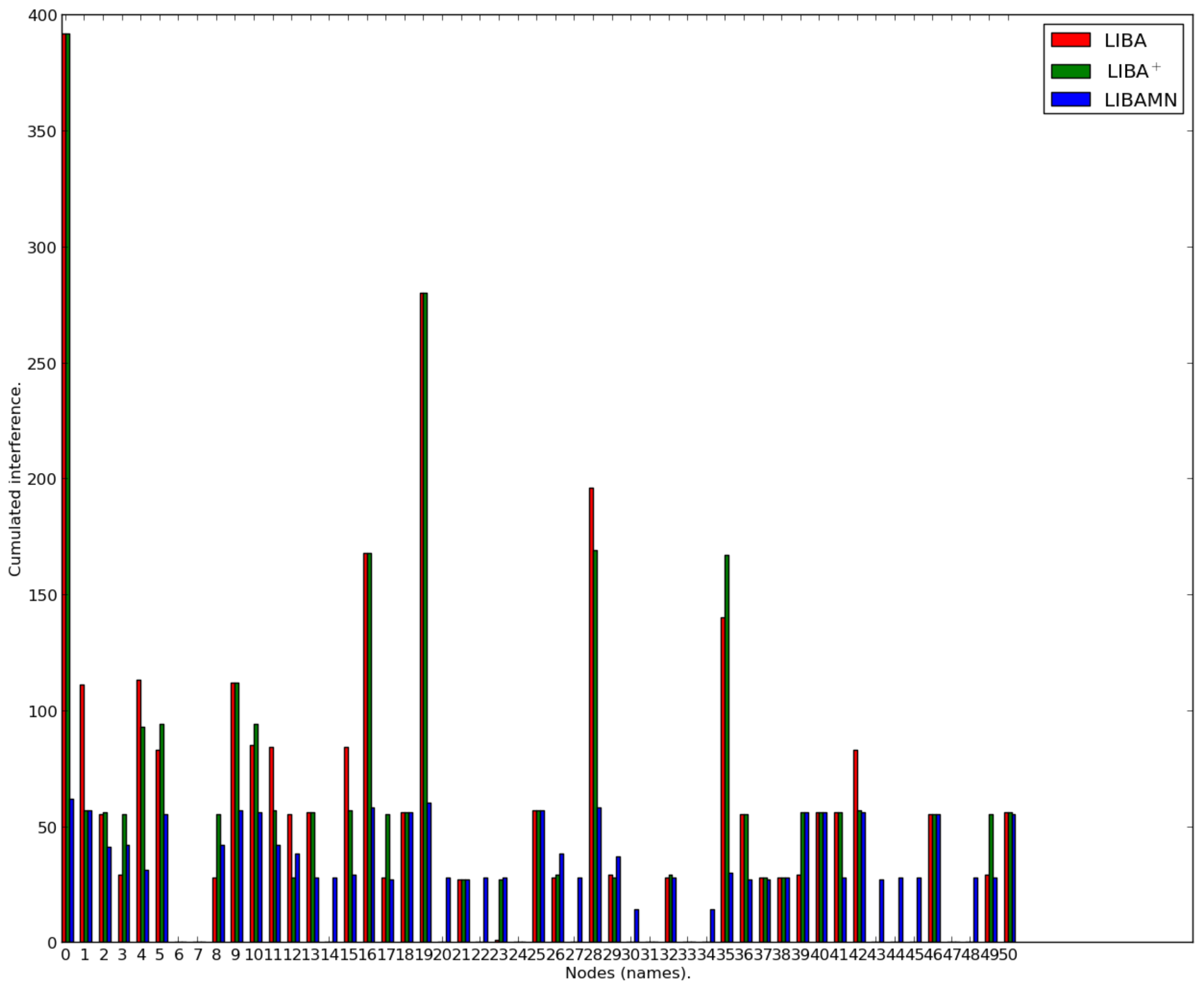


Figure 7.7: Cumulated interference for the Network 3.

Considering a bigger network shown by Figure 7.3, we see that the highest interference is also at the considered sink for all the three algorithms which is node 0. The first and the second highest interference corresponds to LIBA and LIBA$^+$ and the third one corresponds to LIBA. Figure 7.7 shows that 6 nodes are not used by any algorithm and 9 nodes are only used by LIBAMN.

We finally consider a grid-like network as show by Figure 7.4.

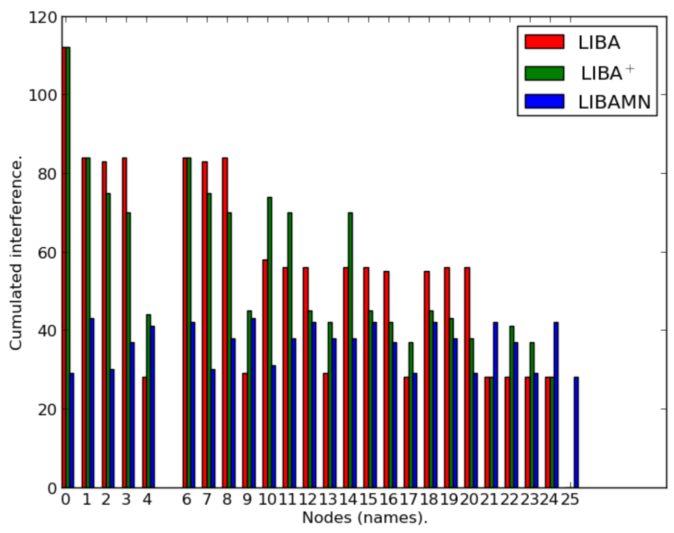


Figure 7.8: Cumulated interference for the Network 4.

Figure 7.8 reveals that the highest interference is still at the sink (node 0). The second highest interference is observable at more than one nodes and at many of them LIBA dominates LIBA$^+$ in terms of interference. No unused node (node 5 is not on the considered network) and only node 50 is used in one algorithm which is LIBAMN.

We evaluated LIBAMN improvement in terms of interference gain by calculating the percentage of the difference of its related highest interference and that of LIBA or LIBA$^+$ since the two of them have the same highest interference which is at the considered sinks. Mathematically,

$$LIBAMN\text{improvement} = \frac{\text{Hight interference}(LIBA) - \text{Hight interference}(LIBAMN)}{\text{Hight interference}(LIBA)} \times 100\%.$$

Results for each network in Section 7.1 are presented in Table 7.2

Network	**LIBA**	**LIBA$^+$**	**LIBAMN**	**LIBAMN improvement (%)**
1 (Figure 7.1)	224	224	58	74.11
2 (Figure 7.2)	168	168	57	66.07
3 (Figure 7.3)	392	392	62	84.18
4 (Figure 7.4)	112	112	43	61.61

Table 7.2: LIBAMN improvement cases.

As shown by Table 7.2 improvement can be over 84% for some networks. The highest cumulated interference for LIBA and LIBA$^+$ is always the same and it is seen at the sink of the considered networks. This is justified by the fact that for both algorithms the sink is always chosen by all its neighbours.

In appendix (9), we showed in detail how weights are redistributed in all the 56 runs of the three algorithms for the network shown by Figure 7.1.

7.3 Standard deviation: load balance

We study the load balancing property for the three algorithms by evaluating the standard deviation of nodes' interference for all the networks described in Section 7.1. Since the three algorithms consist of periodically sending messages and those messages are the ones which stimulate increment of interference at each node, we have chosen to plot interference standard deviation against time.

We firstly considered a network shown by Figure 7.1 where we ran the three algorithms 120 times (using Python).

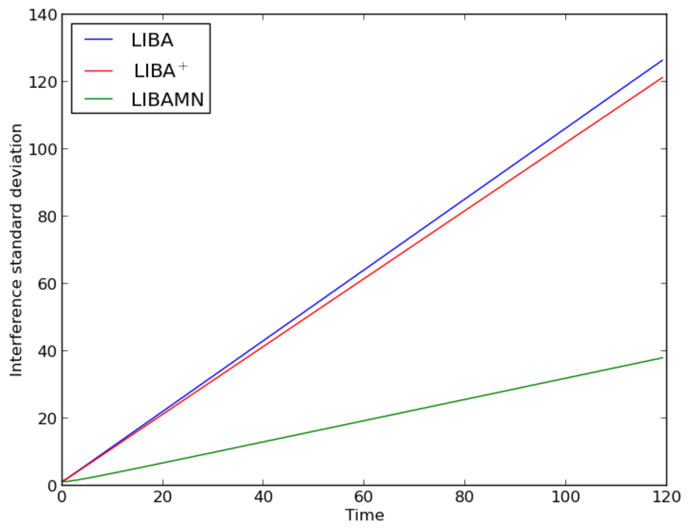


Figure 7.9: Standard deviation for Network 1.

Figure 7.9 reveals that as time increases, the difference of the three algorithms in terms of load balancing becomes wider. For LIBAMN, the standard deviation remains smallest, whereas the one for LIBA remains greatest.

We then considered the network shown in Figure 7.2.

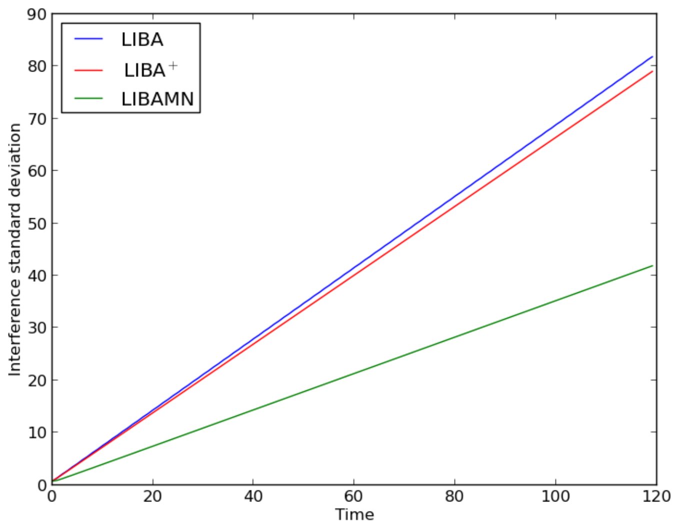


Figure 7.10: Standard deviation for Network 2.

Figure 7.10 reveals the same trends as Figure 7.9, even if the topologies and the number of nodes of the two networks were different.

Using the bigger network shown by Figure 7.3, we ran the three algorithms 1200 times on the same network.

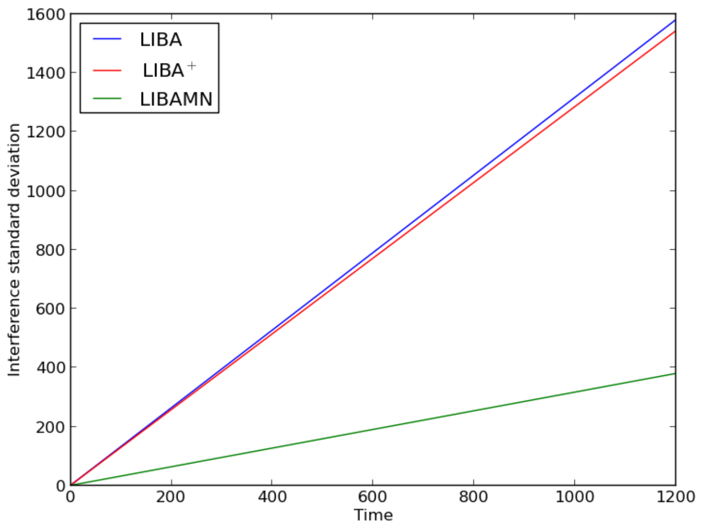


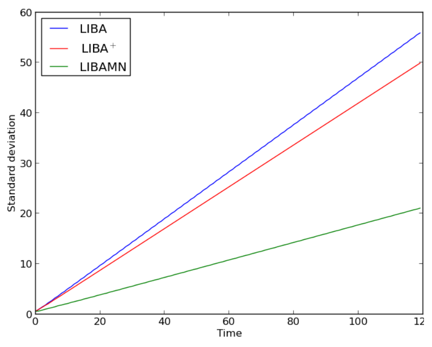
Figure 7.11: Standard deviation for Network 3 (7.3).

Figure 7.11 shows the same trend as Figure 7.2 and 7.1 even if the number of nodes, topologies and the number of runs were increased.

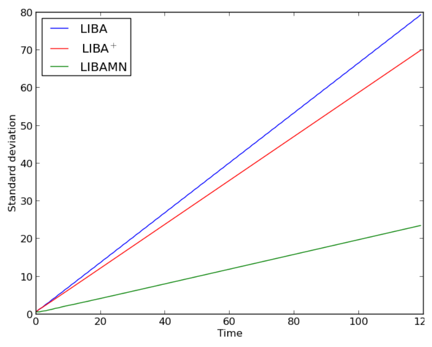
We further considered a regular graph (the grid) as shown by figure 7.4 and for sinks choice, we referred to Table 7.1.

As shown by Figure 7.12, All the nine cases show the same trends as Figures 7.9, 7.10 and 7.11. However, the graphs differ from each other by the difference in increase of interference standard deviation. For instance in Figure 7.12i, increase of interference standard deviation for LIBAMN is slowest while for

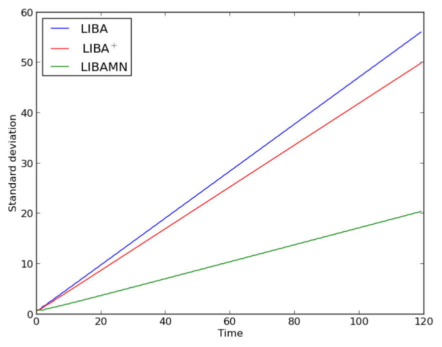
Figure 7.12d, it is fastest.



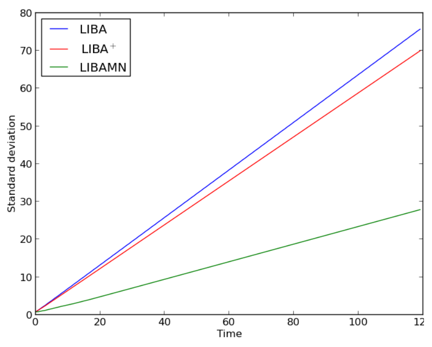
(a) Case 1.



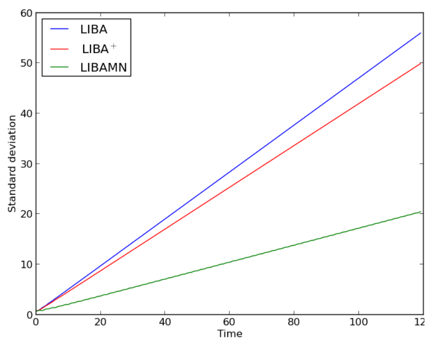
(b) Case 2.



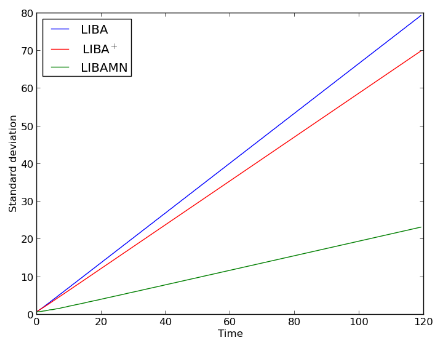
(c) Case 3.



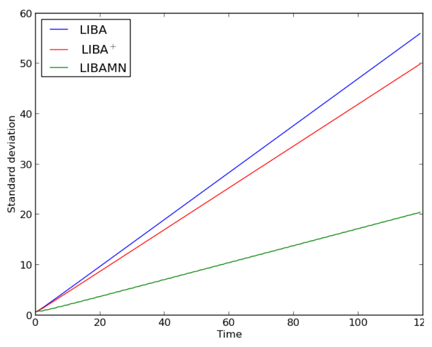
(d) Case 4.



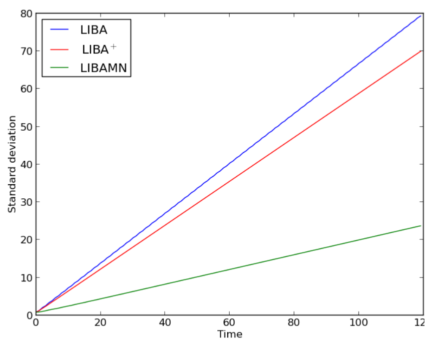
(e) Case 5.



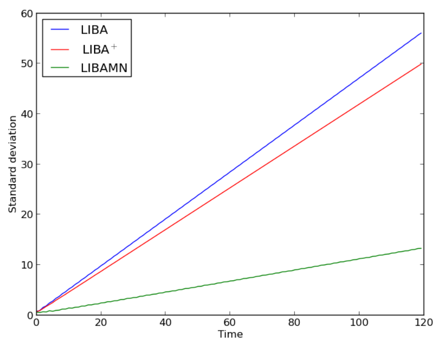
(f) Case 6.



(g) Case 7.



(h) Case 8.



(i) Case 9.

Figure 7.12: Interference standard deviation for Network 4 (7.4).

In all considered runs, all considered networks/cases show that LIBAMN has always the best standard deviation (the least) whereas LIBA$^+$ has the second best. This clarifies and emphasizes the fact that in terms of load balancing, LIBAMN remains the best whereas LIBA$^+$ remains better than LIBA.

It is important to note that the standard deviation levels of LIBA and LIBA$^+$ were closed because both algorithms increase the weight of the sink in the same way, and most of the cases the weight of the sink increases fastest. This is why the two algorithms are closed in terms of load balancing. However, LIBA$^+$ remains the best of the two.

7.4 Computational time

We compare the performance of LIBA, LIBA$^+$ and LIBAMN, in terms of computation time. We used a Python package called "time" to compute the time spent by each algorithm for every network described in Section 7.1. The time is computed in seconds and the time spent by every algorithm for every graph is in table 7.3.

Network	**LIBA**	**LIBA$^+$**	**LIBAMN**
1 (Figure 7.1)	0.177	0.073	0.173
2 (Figure 7.2)	0.090	0.068	0.120
3 (Figure 7.3)	2.224	0.66	2.090
4 (Figure 7.4)	3.394	0.260	3.40

Table 7.3: Computational time.

As shown by table 7.3, the quickest algorithm is LIBA$^+$, The LIBA is slower that LIBA$^+$ because after all nodes acknowledgement, the non-acknowledged nodes spent additional time to make zero their weights. On the other hand for LIBA$^+$ this does not happen: if a node is not chosen, its weight remains unchanged. LIBAMN is slower than LIBA$^+$ because each node has to evaluate and respond to more than one beaconing message (message from different sinks). For the network in Figure 7.2, LIBA is quicker that LIBAMN and slower in all other considered networks.

Table 7.3 also shows that the network shown in Figure 7.4 is the one in which all the three algorithms take longest. This is due to the fact that nodes of the network receive many messages and the choice is done based on a big list of potential parents.

8. Conclusion and future work

8.1 Conclusion

In this work, the Z notation has been used to formalise a description of the Least Interference Beaconing Algorithm. The formalisation of the algorithm has enabled its formal verification. A single run of the algorithm has been identified with the Bellman-Fold algorithm (see [34]) and this has enabled the prof of its correctness.

To ensure energy efficiency, we have adopted the Z schemas used in the LIBA case to design an improved algorithm (LIBA$^+$) and properties of a new designed version have been captured using Linear-time Temporal Logic. This has enabled the proving of the correctness of the algorithm.

Furthermore, using epidemiological approaches, interference transmission in a network using LIBA$^+$ has been modelled. In this case, the network stability analysis has been studied by evaluating the basic reproduction number R_0. This has been achieved by defining a new mathematical structure of nodes in a network named the interference set, and related theorems have been proved for this study.

The existing LIBA version assumes a network with a single sink. We finally adapted LIBA$^+$ to design a new version of the algorithm which supports a network with multiple sinks, LIBAMN. Its properties have been captured in the form of proven theorems. Note that the new version has been proven to be correct.

The three algorithms have been analytically compared and to clarify the results, we have made an experiment by using Python to run both of them on some networks. Considering the load balancing property, the results showed that the LIBA could be improved at more than 84% (see Table 7.2).

This work has mainly consisted of abstract reasoning; experiment has been used only to clarify the proven facts. It has been found that in situations like LIBA's, simulations or experiments are not feasible for more than a handful of nodes or some network topologies, and this shows that formal methods remain important. It is reasonable to say that the same approach would be important and applicable to a wide range of network protocols and hence to protocols in general.

8.2 Future work

LIBA is an algorithm developed to solve mainly the resource sharing problem. In this work it is improved to be correct and more efficient. The following are further works which might make LIBA more applicable and hence more preferable.

8.2.1 Route break issue. Due to environmental issues, a link or a node could suddenly leave a network or have a temporal break. This shows that repairing mechanisms would be of importance. This would be taken care of in future by making LIBA adaptive in such cases.

8.2.2 LIBA for Mobile network. Currently, devices need to communicate while moving. However the network assumed in this work is static. We would appreciate any work which will extend this work to provide a routing algorithm which uses the least interference idea, when the network is assumed to be mobile.

8.2.3 Mathematical analysis and approaches. Interference sets (see 5.2) are new mathematical structures in a network. Their further properties will be studied to enrich future interference related research.

8.2.4 LIBAMN and LIBA$^+$. The LIBAMN is considered as an improved version of LIBA$^+$. We would appreciate any work which will structure the specification and proof of LIBAMN in terms of several LIBA$^+$ in parallel.

8.2.5 Epidemiological investigations. Epidemiological approaches within the same formalism as the protocols will be done to study the different properties of other protocols.

8.2.6 Extensive simulations. We aim to extend simulations revealing more details about efficiency of the three protocols (LIBA, LIBA$^+$, LIBAMN).

9. Appendix

9.1 Comparison of the least interference beaconing algorithms

We perform 56 run on the network shown by Figure 9.1. We aim to compare how nodes interference(weights) are redistributed over the 56 runs of both LIBA, LIBA$^+$ and LIBAMN.

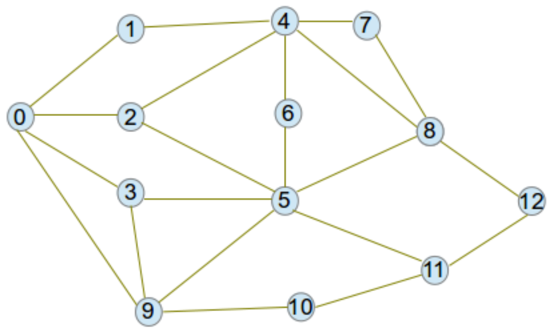


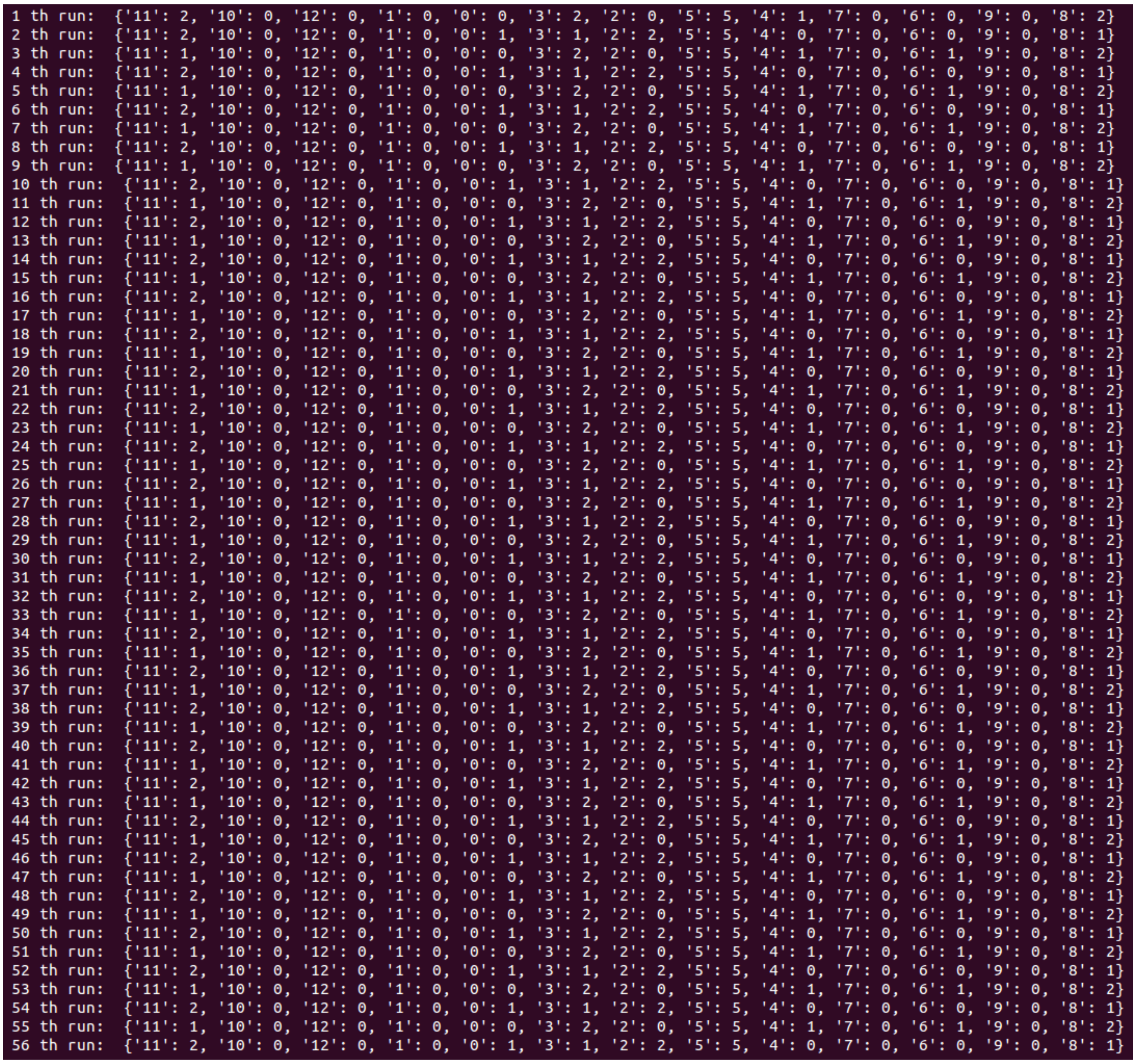
Figure 9.1: Network 1.

As shown by Figure 9.2, in the case of LIBA, all nodes which have not been chosen to be parents change their weights to zero.

Figure 9.3 shows a case of LIBA$^+$. Since the weight is updated by incrementing an existing weight, the total weights are shown by the last run, which is the same case as LIBAMN as shown by Figure 9.4.

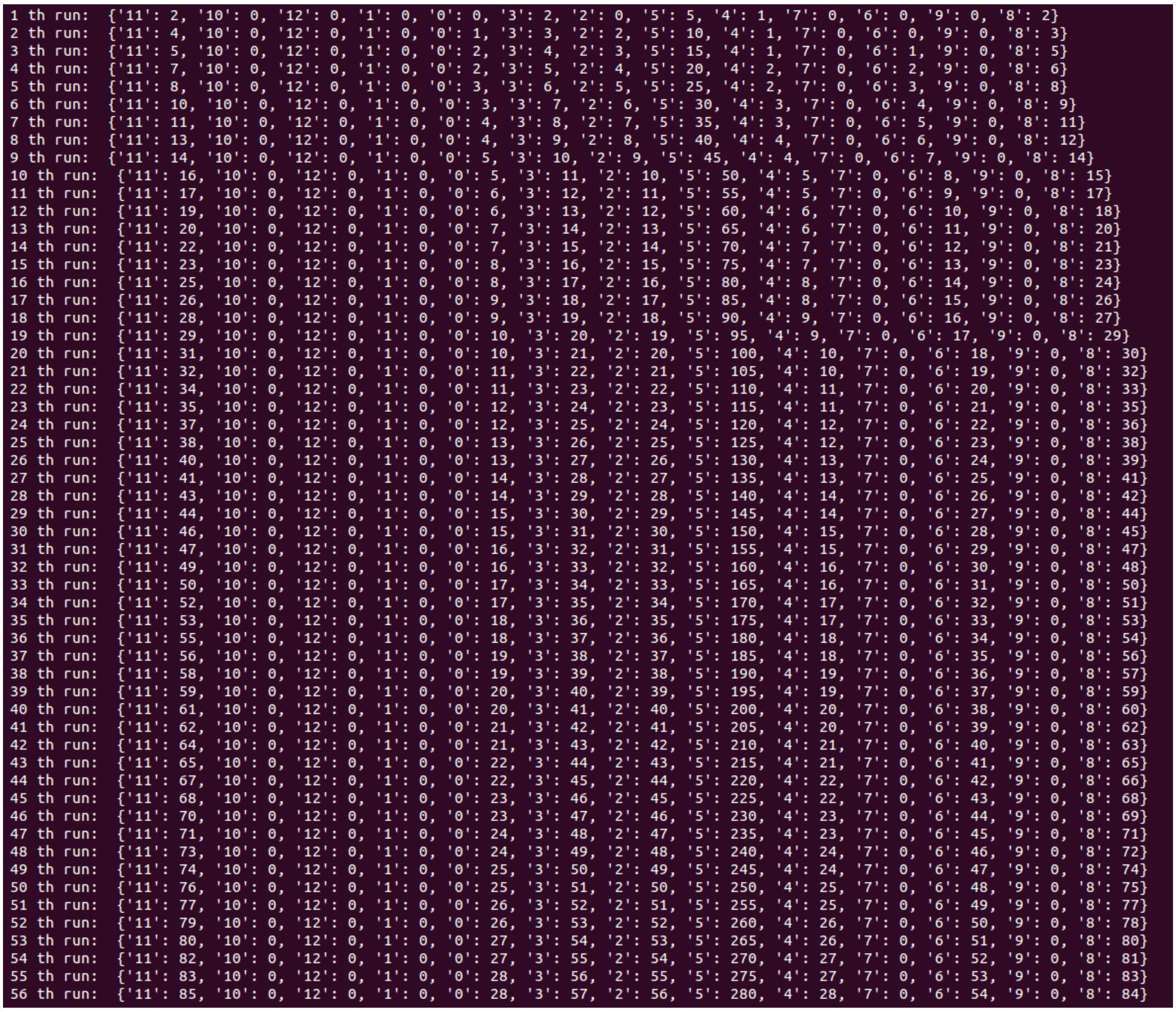
As shown by Figure 9.3, when LIBA$^+$ is processing, Improvement is done by reducing the difference between the nodes' levels of interference. Hence the standard deviation of weights in LIBA$^+$ case is clearly smaller than the one in LIBA case.

As shown by Figure 9.4, the interference level of nodes is more shared and from the last run, it is clear that the total weight (interference) of each node is more balanced. Note here that only two sinks namely, node 5 and 0, were considered.



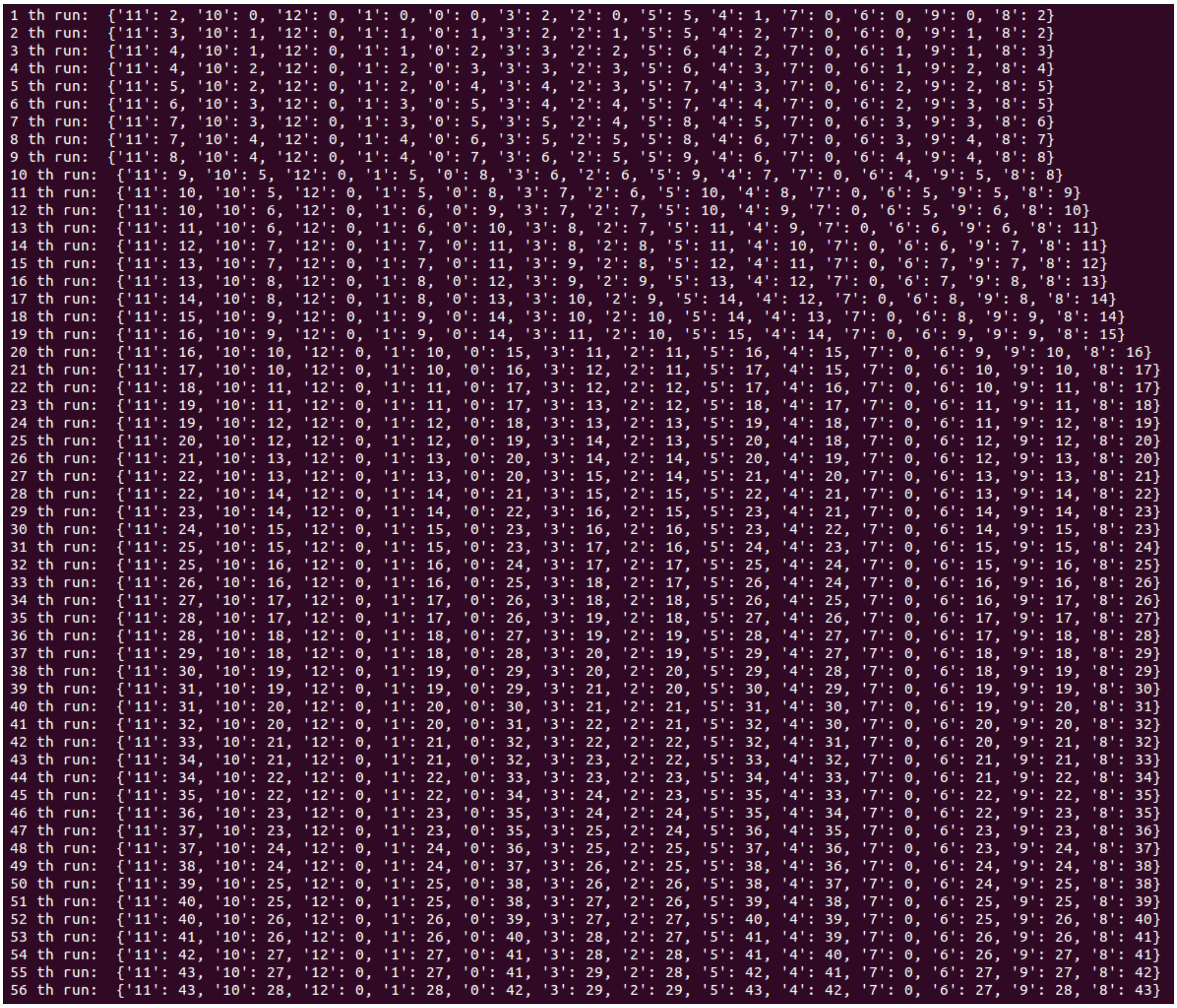
```
1 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 0, '9': 0, '8': 2}
2 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
3 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
4 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
5 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
6 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
7 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
8 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
9 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
10 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
11 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
12 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
13 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
14 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
15 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
16 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
17 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
18 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
19 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
20 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
21 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
22 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
23 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
24 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
25 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
26 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
27 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
28 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
29 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
30 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
31 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
32 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
33 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
34 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
35 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
36 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
37 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
38 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
39 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
40 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
41 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
42 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
43 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
44 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
45 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
46 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
47 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
48 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
49 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
50 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
51 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
52 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
53 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
54 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
55 th run:  {'11': 1, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 1, '9': 0, '8': 2}
56 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 1, '3': 1, '2': 2, '5': 5, '4': 0, '7': 0, '6': 0, '9': 0, '8': 1}
```

Figure 9.2: LIBA results.



```
1 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 0, '9': 0, '8': 2}
2 th run:  {'11': 4, '10': 0, '12': 0, '1': 0, '0': 1, '3': 3, '2': 2, '5': 10, '4': 1, '7': 0, '6': 0, '9': 0, '8': 3}
3 th run:  {'11': 5, '10': 0, '12': 0, '1': 0, '0': 2, '3': 4, '2': 3, '5': 15, '4': 1, '7': 0, '6': 1, '9': 0, '8': 5}
4 th run:  {'11': 7, '10': 0, '12': 0, '1': 0, '0': 2, '3': 5, '2': 4, '5': 20, '4': 2, '7': 0, '6': 2, '9': 0, '8': 6}
5 th run:  {'11': 8, '10': 0, '12': 0, '1': 0, '0': 3, '3': 6, '2': 5, '5': 25, '4': 2, '7': 0, '6': 3, '9': 0, '8': 8}
6 th run:  {'11': 10, '10': 0, '12': 0, '1': 0, '0': 3, '3': 7, '2': 6, '5': 30, '4': 3, '7': 0, '6': 4, '9': 0, '8': 9}
7 th run:  {'11': 11, '10': 0, '12': 0, '1': 0, '0': 4, '3': 8, '2': 7, '5': 35, '4': 3, '7': 0, '6': 5, '9': 0, '8': 11}
8 th run:  {'11': 13, '10': 0, '12': 0, '1': 0, '0': 4, '3': 9, '2': 8, '5': 40, '4': 4, '7': 0, '6': 6, '9': 0, '8': 12}
9 th run:  {'11': 14, '10': 0, '12': 0, '1': 0, '0': 5, '3': 10, '2': 9, '5': 45, '4': 4, '7': 0, '6': 7, '9': 0, '8': 14}
10 th run:  {'11': 16, '10': 0, '12': 0, '1': 0, '0': 5, '3': 11, '2': 10, '5': 50, '4': 5, '7': 0, '6': 8, '9': 0, '8': 15}
11 th run:  {'11': 17, '10': 0, '12': 0, '1': 0, '0': 6, '3': 12, '2': 11, '5': 55, '4': 5, '7': 0, '6': 9, '9': 0, '8': 17}
12 th run:  {'11': 19, '10': 0, '12': 0, '1': 0, '0': 6, '3': 13, '2': 12, '5': 60, '4': 6, '7': 0, '6': 10, '9': 0, '8': 18}
13 th run:  {'11': 20, '10': 0, '12': 0, '1': 0, '0': 7, '3': 14, '2': 13, '5': 65, '4': 6, '7': 0, '6': 11, '9': 0, '8': 20}
14 th run:  {'11': 22, '10': 0, '12': 0, '1': 0, '0': 7, '3': 15, '2': 14, '5': 70, '4': 7, '7': 0, '6': 12, '9': 0, '8': 21}
15 th run:  {'11': 23, '10': 0, '12': 0, '1': 0, '0': 8, '3': 16, '2': 15, '5': 75, '4': 7, '7': 0, '6': 13, '9': 0, '8': 23}
16 th run:  {'11': 25, '10': 0, '12': 0, '1': 0, '0': 8, '3': 17, '2': 16, '5': 80, '4': 8, '7': 0, '6': 14, '9': 0, '8': 24}
17 th run:  {'11': 26, '10': 0, '12': 0, '1': 0, '0': 9, '3': 18, '2': 17, '5': 85, '4': 8, '7': 0, '6': 15, '9': 0, '8': 26}
18 th run:  {'11': 28, '10': 0, '12': 0, '1': 0, '0': 9, '3': 19, '2': 18, '5': 90, '4': 9, '7': 0, '6': 16, '9': 0, '8': 27}
19 th run:  {'11': 29, '10': 0, '12': 0, '1': 0, '0': 10, '3': 20, '2': 19, '5': 95, '4': 9, '7': 0, '6': 17, '9': 0, '8': 29}
20 th run:  {'11': 31, '10': 0, '12': 0, '1': 0, '0': 10, '3': 21, '2': 20, '5': 100, '4': 10, '7': 0, '6': 18, '9': 0, '8': 30}
21 th run:  {'11': 32, '10': 0, '12': 0, '1': 0, '0': 11, '3': 22, '2': 21, '5': 105, '4': 10, '7': 0, '6': 19, '9': 0, '8': 32}
22 th run:  {'11': 34, '10': 0, '12': 0, '1': 0, '0': 11, '3': 23, '2': 22, '5': 110, '4': 11, '7': 0, '6': 20, '9': 0, '8': 33}
23 th run:  {'11': 35, '10': 0, '12': 0, '1': 0, '0': 12, '3': 24, '2': 23, '5': 115, '4': 11, '7': 0, '6': 21, '9': 0, '8': 35}
24 th run:  {'11': 37, '10': 0, '12': 0, '1': 0, '0': 12, '3': 25, '2': 24, '5': 120, '4': 12, '7': 0, '6': 22, '9': 0, '8': 36}
25 th run:  {'11': 38, '10': 0, '12': 0, '1': 0, '0': 13, '3': 26, '2': 25, '5': 125, '4': 12, '7': 0, '6': 23, '9': 0, '8': 38}
26 th run:  {'11': 40, '10': 0, '12': 0, '1': 0, '0': 13, '3': 27, '2': 26, '5': 130, '4': 13, '7': 0, '6': 24, '9': 0, '8': 39}
27 th run:  {'11': 41, '10': 0, '12': 0, '1': 0, '0': 14, '3': 28, '2': 27, '5': 135, '4': 13, '7': 0, '6': 25, '9': 0, '8': 41}
28 th run:  {'11': 43, '10': 0, '12': 0, '1': 0, '0': 14, '3': 29, '2': 28, '5': 140, '4': 14, '7': 0, '6': 26, '9': 0, '8': 42}
29 th run:  {'11': 44, '10': 0, '12': 0, '1': 0, '0': 15, '3': 30, '2': 29, '5': 145, '4': 14, '7': 0, '6': 27, '9': 0, '8': 44}
30 th run:  {'11': 46, '10': 0, '12': 0, '1': 0, '0': 15, '3': 31, '2': 30, '5': 150, '4': 15, '7': 0, '6': 28, '9': 0, '8': 45}
31 th run:  {'11': 47, '10': 0, '12': 0, '1': 0, '0': 16, '3': 32, '2': 31, '5': 155, '4': 15, '7': 0, '6': 29, '9': 0, '8': 47}
32 th run:  {'11': 49, '10': 0, '12': 0, '1': 0, '0': 16, '3': 33, '2': 32, '5': 160, '4': 16, '7': 0, '6': 30, '9': 0, '8': 48}
33 th run:  {'11': 50, '10': 0, '12': 0, '1': 0, '0': 17, '3': 34, '2': 33, '5': 165, '4': 16, '7': 0, '6': 31, '9': 0, '8': 50}
34 th run:  {'11': 52, '10': 0, '12': 0, '1': 0, '0': 17, '3': 35, '2': 34, '5': 170, '4': 17, '7': 0, '6': 32, '9': 0, '8': 51}
35 th run:  {'11': 53, '10': 0, '12': 0, '1': 0, '0': 18, '3': 36, '2': 35, '5': 175, '4': 17, '7': 0, '6': 33, '9': 0, '8': 53}
36 th run:  {'11': 55, '10': 0, '12': 0, '1': 0, '0': 18, '3': 37, '2': 36, '5': 180, '4': 18, '7': 0, '6': 34, '9': 0, '8': 54}
37 th run:  {'11': 56, '10': 0, '12': 0, '1': 0, '0': 19, '3': 38, '2': 37, '5': 185, '4': 18, '7': 0, '6': 35, '9': 0, '8': 56}
38 th run:  {'11': 58, '10': 0, '12': 0, '1': 0, '0': 19, '3': 39, '2': 38, '5': 190, '4': 19, '7': 0, '6': 36, '9': 0, '8': 57}
39 th run:  {'11': 59, '10': 0, '12': 0, '1': 0, '0': 20, '3': 40, '2': 39, '5': 195, '4': 19, '7': 0, '6': 37, '9': 0, '8': 59}
40 th run:  {'11': 61, '10': 0, '12': 0, '1': 0, '0': 20, '3': 41, '2': 40, '5': 200, '4': 20, '7': 0, '6': 38, '9': 0, '8': 60}
41 th run:  {'11': 62, '10': 0, '12': 0, '1': 0, '0': 21, '3': 42, '2': 41, '5': 205, '4': 20, '7': 0, '6': 39, '9': 0, '8': 62}
42 th run:  {'11': 64, '10': 0, '12': 0, '1': 0, '0': 21, '3': 43, '2': 42, '5': 210, '4': 21, '7': 0, '6': 40, '9': 0, '8': 63}
43 th run:  {'11': 65, '10': 0, '12': 0, '1': 0, '0': 22, '3': 44, '2': 43, '5': 215, '4': 21, '7': 0, '6': 41, '9': 0, '8': 65}
44 th run:  {'11': 67, '10': 0, '12': 0, '1': 0, '0': 22, '3': 45, '2': 44, '5': 220, '4': 22, '7': 0, '6': 42, '9': 0, '8': 66}
45 th run:  {'11': 68, '10': 0, '12': 0, '1': 0, '0': 23, '3': 46, '2': 45, '5': 225, '4': 22, '7': 0, '6': 43, '9': 0, '8': 68}
46 th run:  {'11': 70, '10': 0, '12': 0, '1': 0, '0': 23, '3': 47, '2': 46, '5': 230, '4': 23, '7': 0, '6': 44, '9': 0, '8': 69}
47 th run:  {'11': 71, '10': 0, '12': 0, '1': 0, '0': 24, '3': 48, '2': 47, '5': 235, '4': 23, '7': 0, '6': 45, '9': 0, '8': 71}
48 th run:  {'11': 73, '10': 0, '12': 0, '1': 0, '0': 24, '3': 49, '2': 48, '5': 240, '4': 24, '7': 0, '6': 46, '9': 0, '8': 72}
49 th run:  {'11': 74, '10': 0, '12': 0, '1': 0, '0': 25, '3': 50, '2': 49, '5': 245, '4': 24, '7': 0, '6': 47, '9': 0, '8': 74}
50 th run:  {'11': 76, '10': 0, '12': 0, '1': 0, '0': 25, '3': 51, '2': 50, '5': 250, '4': 25, '7': 0, '6': 48, '9': 0, '8': 75}
51 th run:  {'11': 77, '10': 0, '12': 0, '1': 0, '0': 26, '3': 52, '2': 51, '5': 255, '4': 25, '7': 0, '6': 49, '9': 0, '8': 77}
52 th run:  {'11': 79, '10': 0, '12': 0, '1': 0, '0': 26, '3': 53, '2': 52, '5': 260, '4': 26, '7': 0, '6': 50, '9': 0, '8': 78}
53 th run:  {'11': 80, '10': 0, '12': 0, '1': 0, '0': 27, '3': 54, '2': 53, '5': 265, '4': 26, '7': 0, '6': 51, '9': 0, '8': 80}
54 th run:  {'11': 82, '10': 0, '12': 0, '1': 0, '0': 27, '3': 55, '2': 54, '5': 270, '4': 27, '7': 0, '6': 52, '9': 0, '8': 81}
55 th run:  {'11': 83, '10': 0, '12': 0, '1': 0, '0': 28, '3': 56, '2': 55, '5': 275, '4': 27, '7': 0, '6': 53, '9': 0, '8': 83}
56 th run:  {'11': 85, '10': 0, '12': 0, '1': 0, '0': 28, '3': 57, '2': 56, '5': 280, '4': 28, '7': 0, '6': 54, '9': 0, '8': 84}
```

Figure 9.3: LIBA$^+$ results.



```
1 th run:  {'11': 2, '10': 0, '12': 0, '1': 0, '0': 0, '3': 2, '2': 0, '5': 5, '4': 1, '7': 0, '6': 0, '9': 0, '8': 2}
2 th run:  {'11': 3, '10': 1, '12': 0, '1': 1, '0': 1, '3': 2, '2': 1, '5': 5, '4': 2, '7': 0, '6': 0, '9': 1, '8': 2}
3 th run:  {'11': 4, '10': 1, '12': 0, '1': 1, '0': 2, '3': 3, '2': 2, '5': 6, '4': 2, '7': 0, '6': 1, '9': 1, '8': 3}
4 th run:  {'11': 4, '10': 2, '12': 0, '1': 2, '0': 3, '3': 3, '2': 3, '5': 6, '4': 3, '7': 0, '6': 1, '9': 2, '8': 4}
5 th run:  {'11': 5, '10': 2, '12': 0, '1': 2, '0': 4, '3': 4, '2': 3, '5': 7, '4': 3, '7': 0, '6': 2, '9': 2, '8': 5}
6 th run:  {'11': 6, '10': 3, '12': 0, '1': 3, '0': 5, '3': 4, '2': 4, '5': 7, '4': 4, '7': 0, '6': 2, '9': 3, '8': 5}
7 th run:  {'11': 7, '10': 3, '12': 0, '1': 3, '0': 5, '3': 5, '2': 4, '5': 8, '4': 5, '7': 0, '6': 3, '9': 3, '8': 6}
8 th run:  {'11': 7, '10': 4, '12': 0, '1': 4, '0': 6, '3': 5, '2': 5, '5': 8, '4': 6, '7': 0, '6': 3, '9': 4, '8': 7}
9 th run:  {'11': 8, '10': 4, '12': 0, '1': 4, '0': 7, '3': 6, '2': 5, '5': 9, '4': 6, '7': 0, '6': 4, '9': 4, '8': 8}
10 th run:  {'11': 9, '10': 5, '12': 0, '1': 5, '0': 8, '3': 6, '2': 6, '5': 9, '4': 7, '7': 0, '6': 4, '9': 5, '8': 8}
11 th run:  {'11': 10, '10': 5, '12': 0, '1': 5, '0': 8, '3': 7, '2': 6, '5': 10, '4': 8, '7': 0, '6': 5, '9': 5, '8': 9}
12 th run:  {'11': 10, '10': 6, '12': 0, '1': 6, '0': 9, '3': 7, '2': 7, '5': 10, '4': 9, '7': 0, '6': 5, '9': 6, '8': 10}
13 th run:  {'11': 11, '10': 6, '12': 0, '1': 6, '0': 10, '3': 8, '2': 7, '5': 11, '4': 9, '7': 0, '6': 6, '9': 6, '8': 11}
14 th run:  {'11': 12, '10': 7, '12': 0, '1': 7, '0': 11, '3': 8, '2': 8, '5': 11, '4': 10, '7': 0, '6': 6, '9': 7, '8': 11}
15 th run:  {'11': 13, '10': 7, '12': 0, '1': 7, '0': 11, '3': 9, '2': 8, '5': 12, '4': 11, '7': 0, '6': 7, '9': 7, '8': 12}
16 th run:  {'11': 13, '10': 8, '12': 0, '1': 8, '0': 12, '3': 9, '2': 9, '5': 13, '4': 12, '7': 0, '6': 7, '9': 8, '8': 13}
17 th run:  {'11': 14, '10': 8, '12': 0, '1': 8, '0': 13, '3': 10, '2': 9, '5': 14, '4': 12, '7': 0, '6': 8, '9': 8, '8': 14}
18 th run:  {'11': 15, '10': 9, '12': 0, '1': 9, '0': 14, '3': 10, '2': 10, '5': 14, '4': 13, '7': 0, '6': 8, '9': 9, '8': 14}
19 th run:  {'11': 16, '10': 9, '12': 0, '1': 9, '0': 14, '3': 11, '2': 10, '5': 15, '4': 14, '7': 0, '6': 9, '9': 9, '8': 15}
20 th run:  {'11': 16, '10': 10, '12': 0, '1': 10, '0': 15, '3': 11, '2': 11, '5': 16, '4': 15, '7': 0, '6': 9, '9': 10, '8': 16}
21 th run:  {'11': 17, '10': 10, '12': 0, '1': 10, '0': 16, '3': 12, '2': 11, '5': 17, '4': 15, '7': 0, '6': 10, '9': 10, '8': 17}
22 th run:  {'11': 18, '10': 11, '12': 0, '1': 11, '0': 17, '3': 12, '2': 12, '5': 17, '4': 16, '7': 0, '6': 10, '9': 11, '8': 17}
23 th run:  {'11': 19, '10': 11, '12': 0, '1': 11, '0': 17, '3': 13, '2': 12, '5': 18, '4': 17, '7': 0, '6': 11, '9': 11, '8': 18}
24 th run:  {'11': 19, '10': 12, '12': 0, '1': 12, '0': 18, '3': 13, '2': 13, '5': 19, '4': 18, '7': 0, '6': 11, '9': 12, '8': 19}
25 th run:  {'11': 20, '10': 12, '12': 0, '1': 12, '0': 19, '3': 14, '2': 13, '5': 20, '4': 18, '7': 0, '6': 12, '9': 12, '8': 20}
26 th run:  {'11': 21, '10': 13, '12': 0, '1': 13, '0': 20, '3': 14, '2': 14, '5': 20, '4': 19, '7': 0, '6': 12, '9': 13, '8': 20}
27 th run:  {'11': 22, '10': 13, '12': 0, '1': 13, '0': 20, '3': 15, '2': 14, '5': 21, '4': 20, '7': 0, '6': 13, '9': 13, '8': 21}
28 th run:  {'11': 22, '10': 14, '12': 0, '1': 14, '0': 21, '3': 15, '2': 15, '5': 22, '4': 21, '7': 0, '6': 13, '9': 14, '8': 22}
29 th run:  {'11': 23, '10': 14, '12': 0, '1': 14, '0': 22, '3': 16, '2': 15, '5': 23, '4': 21, '7': 0, '6': 14, '9': 14, '8': 23}
30 th run:  {'11': 24, '10': 15, '12': 0, '1': 15, '0': 23, '3': 16, '2': 16, '5': 23, '4': 22, '7': 0, '6': 14, '9': 15, '8': 23}
31 th run:  {'11': 25, '10': 15, '12': 0, '1': 15, '0': 23, '3': 17, '2': 16, '5': 24, '4': 23, '7': 0, '6': 15, '9': 15, '8': 24}
32 th run:  {'11': 25, '10': 16, '12': 0, '1': 16, '0': 24, '3': 17, '2': 17, '5': 25, '4': 24, '7': 0, '6': 15, '9': 16, '8': 25}
33 th run:  {'11': 26, '10': 16, '12': 0, '1': 16, '0': 25, '3': 18, '2': 17, '5': 26, '4': 24, '7': 0, '6': 16, '9': 16, '8': 26}
34 th run:  {'11': 27, '10': 17, '12': 0, '1': 17, '0': 26, '3': 18, '2': 18, '5': 26, '4': 25, '7': 0, '6': 16, '9': 17, '8': 26}
35 th run:  {'11': 28, '10': 17, '12': 0, '1': 17, '0': 26, '3': 19, '2': 18, '5': 27, '4': 26, '7': 0, '6': 17, '9': 17, '8': 27}
36 th run:  {'11': 28, '10': 18, '12': 0, '1': 18, '0': 27, '3': 19, '2': 19, '5': 28, '4': 27, '7': 0, '6': 17, '9': 18, '8': 28}
37 th run:  {'11': 29, '10': 18, '12': 0, '1': 18, '0': 28, '3': 20, '2': 19, '5': 29, '4': 27, '7': 0, '6': 18, '9': 18, '8': 29}
38 th run:  {'11': 30, '10': 19, '12': 0, '1': 19, '0': 29, '3': 20, '2': 20, '5': 29, '4': 28, '7': 0, '6': 18, '9': 19, '8': 29}
39 th run:  {'11': 31, '10': 19, '12': 0, '1': 19, '0': 29, '3': 21, '2': 20, '5': 30, '4': 29, '7': 0, '6': 19, '9': 19, '8': 30}
40 th run:  {'11': 31, '10': 20, '12': 0, '1': 20, '0': 30, '3': 21, '2': 21, '5': 31, '4': 30, '7': 0, '6': 19, '9': 20, '8': 31}
41 th run:  {'11': 32, '10': 20, '12': 0, '1': 20, '0': 31, '3': 22, '2': 21, '5': 32, '4': 30, '7': 0, '6': 20, '9': 20, '8': 32}
42 th run:  {'11': 33, '10': 21, '12': 0, '1': 21, '0': 32, '3': 22, '2': 22, '5': 32, '4': 31, '7': 0, '6': 20, '9': 21, '8': 32}
43 th run:  {'11': 34, '10': 21, '12': 0, '1': 21, '0': 32, '3': 23, '2': 22, '5': 33, '4': 32, '7': 0, '6': 21, '9': 21, '8': 33}
44 th run:  {'11': 34, '10': 22, '12': 0, '1': 22, '0': 33, '3': 23, '2': 23, '5': 34, '4': 33, '7': 0, '6': 21, '9': 22, '8': 34}
45 th run:  {'11': 35, '10': 22, '12': 0, '1': 22, '0': 34, '3': 24, '2': 23, '5': 35, '4': 33, '7': 0, '6': 22, '9': 22, '8': 35}
46 th run:  {'11': 36, '10': 23, '12': 0, '1': 23, '0': 35, '3': 24, '2': 24, '5': 35, '4': 34, '7': 0, '6': 22, '9': 23, '8': 35}
47 th run:  {'11': 37, '10': 23, '12': 0, '1': 23, '0': 35, '3': 25, '2': 24, '5': 36, '4': 35, '7': 0, '6': 23, '9': 23, '8': 36}
48 th run:  {'11': 37, '10': 24, '12': 0, '1': 24, '0': 36, '3': 25, '2': 25, '5': 37, '4': 36, '7': 0, '6': 23, '9': 24, '8': 37}
49 th run:  {'11': 38, '10': 24, '12': 0, '1': 24, '0': 37, '3': 26, '2': 25, '5': 38, '4': 36, '7': 0, '6': 24, '9': 24, '8': 38}
50 th run:  {'11': 39, '10': 25, '12': 0, '1': 25, '0': 38, '3': 26, '2': 26, '5': 38, '4': 37, '7': 0, '6': 24, '9': 25, '8': 38}
51 th run:  {'11': 40, '10': 25, '12': 0, '1': 25, '0': 38, '3': 27, '2': 26, '5': 39, '4': 38, '7': 0, '6': 25, '9': 25, '8': 39}
52 th run:  {'11': 40, '10': 26, '12': 0, '1': 26, '0': 39, '3': 27, '2': 27, '5': 40, '4': 39, '7': 0, '6': 25, '9': 26, '8': 40}
53 th run:  {'11': 41, '10': 26, '12': 0, '1': 26, '0': 40, '3': 28, '2': 27, '5': 41, '4': 39, '7': 0, '6': 26, '9': 26, '8': 41}
54 th run:  {'11': 42, '10': 27, '12': 0, '1': 27, '0': 41, '3': 28, '2': 28, '5': 41, '4': 40, '7': 0, '6': 26, '9': 27, '8': 41}
55 th run:  {'11': 43, '10': 27, '12': 0, '1': 27, '0': 41, '3': 29, '2': 28, '5': 42, '4': 41, '7': 0, '6': 27, '9': 27, '8': 42}
56 th run:  {'11': 43, '10': 28, '12': 0, '1': 28, '0': 42, '3': 29, '2': 29, '5': 43, '4': 42, '7': 0, '6': 27, '9': 28, '8': 43}
```

Figure 9.4: LIBAMN results.

Acknowledgements

I am grateful to my supervisors Prof. Bigomokero Antoine Bagula and Prof. J. W. Sanders who guided me until the completion of this work. Thank you Prof J. W. Sanders for invaluable academic gift I got from your consistent training: Formal methods with Z notation.

I am thankful to UCT especially the department of Computer Science for its consistent assistance during my research life at the university. My invaluable gratitude goes to AIMS founder, director and the whole staff for the amazing courses, experience and various assistance which gave me enough back ground to work on this topic.

Thank you my mates especially Karisa Ingambe Dominique for your assistance.

References

[1] S. Ahmed, A. K. Ramani, and N. A. Zafar. Verifying route request procedure of aodv using graph theory and formal methods, 2011.

[2] A. Bagula, Zennaro, Marco, G. Inggs, S. Scott, and D. Gascon. Ubiquitous sensor networking for development (usn4d): An application to pollution monitoring. *Sensors*, 12(1):391–414, 2012.

[3] A. Bagula, D. Djenouri, and E. Karbab. Ubiquitous sensor network management: The least interference beaconing model. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*. IEEE PIMRC, 2013.

[4] A. B. Bagula. Hybrid traffic engineering: the least path interference algorithm. In *Proceedings of the 2004 annual research conference of the South African institute of computer scientists and information technologists on IT research in developing countries*, pages 89–96. South African Institute for Computer Scientists and Information Technologists, 2004.

[5] A. B. Bagula. Hybrid routing in next generation ip networks. *Computer Communications*, 29(7): 879–892, 2006.

[6] A. B. Bagula. On achieveing bandwidth-aware lsp/lambdasp multiplexing/separation in multi-layer networks. *Selected Areas in Communications, IEEE Journal on*, 25(5):987–1000, 2007.

[7] A. B. Bagula and E. Djenouri, Djameland Karbab. On the relevance of using interference and service differentiation routing in the internet-of-things. In *Internet of Things, Smart Spaces, and Next Generation Networking*, pages 25–35. Springer, 2013.

[8] S. Brien and A. Martin. A calculus for schemas in z. *J. Symbolic Computation*, 11:1–29, 1999.

[9] N. Deo. *Graph theory with applications to engineering and computer science*. PHI Learning Pvt. Ltd., 2004.

[10] O. Diekmann, J. Heesterbeek, and J. A. Metz. On the definition and the computation of the basic reproduction ratio r 0 in models for infectious diseases in heterogeneous populations. *Journal of mathematical biology*, 28(4):365–382, 1990.

[11] O. Diekmann, J. Heesterbeek, and M. Roberts. The construction of next-generation matrices for compartmental epidemic models. *Journal of The Royal Society Interface*, 7(47):873–885, 2010.

[12] D. Djenouri and I. Balasingham. Traffic-differentiation-based modular qos localized routing for wireless sensor networks. *Mobile Computing, IEEE Transactions on*, 10(6):797–809, 2011.

[13] M. Dorigo and L. M. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *Evolutionary Computation, IEEE Transactions on*, 1(1):53–66, 1997.

[14] R. Duke and G. Rose. *Formal object-oriented specification using Object-Z*. Macmillan, 2000.

[15] A. Fehnker, R. van Glabbeek, P. Höfner, A. McIver, M. Portmann, and W. Tan. A process algebra for wireless mesh networks used for modelling, verifying and analysing aodv. Technical report, Tech. Rep. 5513, NICTA, 2013.

[16] E. Felemban, C.-G. Lee, and E. Ekici. Mmspeed: multipath multi-speed protocol for qos guarantee of reliability and. timeliness in wireless sensor networks. *Mobile Computing, IEEE Transactions on*, 5(6):738–754, 2006.

[17] R. Fonseca, O. Gnawali, K. Jamieson, S. Kim, P. Levis, and A. Woo. The collection tree protocol (ctp). *TinyOS TEP*, 123:2, 2006.

[18] L. Friedmann and L. Boukhatem. Efficient multi-sink relocation in wireless sensor network. In *Networking and Services, 2007. ICNS. Third International Conference on*, pages 90–90. IEEE, 2007.

[19] O. Gnawali, R. Fonseca, K. Jamieson, D. Moss, and P. Levis. Collection tree protocol. In *Proceedings of the 7th ACM Conference on Embedded Networked Sensor Systems*, pages 1–14. ACM, 2009.

[20] R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[21] D. Granot and G. Huberman. Minimum cost spanning tree games. *Mathematical programming*, 21 (1):1–18, 1981.

[22] J. Heffernan, R. Smith, and L. Wahl. Perspectives on the basic reproductive ratio. *Journal of the Royal Society Interface*, 2(4):281–293, 2005.

[23] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *ACM SIGOPS operating systems review*, volume 34, pages 93–104. ACM, 2000.

[24] J.-W. Hu and H.-M. Tang. Numerical methods for differential equations. *City University, Hong Kong*, 2003.

[25] M. Huth and M. Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge University Press, 2004.

[26] J. M. Hyman and J. Li. Differential susceptibility epidemic models. *Journal of mathematical biology*, 50(6):626–644, 2005.

[27] M. Insight. Network tutorial, May 2013. URL http://mathinsight.org/thread/network_tutorial#introduction.

[28] W. Kermack and A. McKendrick. Contributions to the mathematical theory of epidemics—i. *Bulletin of Mathematical Biology*, 53(1):33–55, 1991.

[29] T. Körner. Metric and topological spaces. 2010.

[30] D. Luiz, A. Dasilva, T. Lin, T. Lin, D. Scott, and F. Midkiff. Mobile ad-hoc network routing protocols: Methodologies and applications. Technical report, ECE Department, Virginia Tech, 2004.

[31] B. K. Mishra and G. M. Ansari. Differential epidemic model of virus and worms in computer network. *IJ Network Security*, 14(3):149–155, 2012.

[32] L. Mottola and G. P. Picco. Muster: Adaptive energy-aware multisink routing in wireless sensor networks. *Mobile Computing, IEEE Transactions on*, 10(12):1694–1709, 2011.

[33] E. I. Oyman and C. Ersoy. Multiple sink network design problem in large scale wireless sensor networks. In *Communications, 2004 IEEE International Conference on*, volume 6, pages 3663–3667. IEEE, 2004.

[34] C. S. C. E. L. Ronald Rivest, Thomas H. Cormen. *Introduction to algorithms*, volume 3. MIT press, 2009.

[35] M. Senthilkumar and S. Somasundaram. Energy aware multiple constraints intelligent multipath qos routing protocol with dynamic mobility prediction for manet. In *Process Automation, Control and Computing (PACC), 2011 International Conference on*, pages 1–8. IEEE, 2011.

[36] G. Smith. *The Object-Z specification language*, volume 101. Citeseer, 2000.

[37] H. Sotoodeh, F. Safaei, A. Sanei, and E. Daei. A general stochastic information diffusion model in social networks based on epidemic diseases. *arXiv preprint arXiv:1309.7289*, 2013.

[38] J. M. Spivey. *The Z notation: a reference manual*. Prentice Hall International (UK) Ltd., 1992.

[39] S. Tang and B. L. Mark. Analysis of virus spread in wireless sensor networks: An epidemic model. In *Design of Reliable Communication Networks, 2009. DRCN 2009. 7th International Workshop on*, pages 86–91. IEEE, 2009.

[40] wikibooks. Set theory/sets, January 2014. URL http://en.wikibooks.org/wiki/Set_Theory/Sets.

[41] Wikipedia. List of ad hoc routing protocols, April 2013. URL http://en.wikipedia.org/wiki/List_of_ad_hoc_routing_protocols.

[42] X. Zhang, G. Neglia, J. Kurose, and D. Towsley. Performance modeling of epidemic routing. *Computer Networks*, 51(10):2867–2891, 2007.