University of Trento
Department of Information and Computer Science

Master Thesis

Computing centrality on real hardware: challenges and results

Author:
Quynh P.X. Nguyen

Supervisor:
Professor: Renato Lo Cigno
Co-supervisor:
Leonardo Maccari, PhD.

April 6, 2016
Academic Year 2015-2016

Acknowledgements

This work is done under the supervision of Prof. Renato Lo Cigno Antonio and co-supervior Leonardo Maccari. I would like to express my sincere gratitude for their throughout guidance. They have been immensely patient with my progress, given me a little push every now and then to keep me on track.

I would like to thank Opera Universita, University of Trento and Erasmus+ program for funding my Master study, giving me a chance to study in a world-class university, and also in University of Edinburgh in an exchange program. Thank you for giving me education.

Thank you friends, for your supports and encouragement.

And finally, I want to dedicate these last lines to my family in Vietnam.

[Written in Vietnamese] Cam on Ba Me da tin tuong con, khong tao ap luc len chuyen hoc hanh cua con, cho con tu do lam moi thu con thich. Va quan trong la luon dung o dang sau de giup do con.

Contents

Chapter 1

Introduction

People have a strong need for acquiring information, and they want faster, more reliable, more secured, and more affordable mean to communicate. During the 90s we have seen the boom of the Internet, at that time most devices was connected to the Internet through a cable with an ISP standing in between to provide services. In the late 2000s, connecting to the Internet without plugging anything in became a norm. Most of the wireless networks that we are familiar with now, such as 3G, LTE, 4G, Wi-Fi are centralized networks, which means those networks rely on base stations to broadcast signal, or access points in a managed wireless network to give the service. Wireless Mesh Networks (WMNs) in recent years are emerging as a key technology for next-generation wireless networking. They do not have any centralized control to route and forward packets, but rather all nodes cooperate together to relay each others' packets. The efficient protocol is important than ever in WMNs since it would affect the performance largely. And several papers [1], [2] have proposed the use of centrality to enhance routing in WMNs.

Centrality[1] is a measurement to quantify how influential a node is, given its position in the network. We concerns with centrality measurements since in WMNs, the ability to find central node can serve many great purposes such as adding more routers around those most central one to balance the load, and to lessen the impact when one of the most central routers fail. Or information about the central node can help with routing, to allow information to be spreaded faster in the network.

In this thesis, the main focus is on one specific centrality measurement called *betweenness centrality* $c_B(v)$. Intuitively, a node with high betweenness centrality has a large control over the traffic flowing within the network, given that the information is transfered using the shotest paths. And betweenness centrality defined as the sum of the fraction of total number of shortest paths that pass through that node. The best known algorithm [3] to calculate $c_B(v)$ runs in quadratic time, making the calculation for a large graph impractical. Another algorithm in focus is heuristic-based [4] which demonstrated its effectiveness in several input graphs of choice. The main work of thesis is implementing those forementioned algorithms, and carrying out the performance evaluation for both of them on a computer, and on a low-cost wireless router. We want to know whether using the heuristic-based algorithm to calculate the betweenness centrality for every nodes in topology of WCNs would scale better

[1] it is also referred to by other terms: centrality indices or centrality measures

than using the original algorithm.

We begin with a short overview of WMNs in Section 1.1. Then we present one application of WMNs in reality – the Wireless Community Networks (WCNs). We cover the definition plus some challenging problems of WCNs in Section 1.2. After having all the background knowledge for WMNs and WCNs, the contribution of this thesis for WCNs is laid out in Section 1.3. The outline for the subsequent chapters can be found in Section 1.4.

1.1 Wireless Mesh Networks

A WMN is any wireless network having a mesh topology[2]. In full mesh topology, every pair of nodes is directly connected with each other, while in partial mesh topology, not all nodes are directly interconnected with each other. Even though the definition of WMN mentions the full mesh topology, in practice, WMN is usually built over a partial mesh topology. This is due to the limited coverage of signal from wireless devices making it impossible for each node to directly connect with all other nodes.

For classification, WMNs belong to a class of multi-hop wireless network, where information traverses from a source to a destination using two or more wireless hops.

The main advantages of a WMN is its ability of fault intolerance. Since WMN is a decentralized wireless network, when several nodes fail, the network can find a new path way to link the network together. The term *autonomic system* is often used to describe properties of WMN.

WMNs find interesting applications in many areas [5] such as: broadband home networking, enterprise networking, community networking, city-wide wireless coverage, spontaneous mesh network, transportation systems. All the different areas of application can be classified into two categories. First, the WMN acts as a backbone network to extend the Internet connectivity for nodes that are not in the coverage of Internet access. This might due to the fact that installing wired infrastructure is too expensive for rural areas, or areas with challenging terrains. Another scenario is in a disaster-stroke area where the current communication facilities (cables, base stations) were destroyed, then WMNs can provide temporary solution for communication. There are many examples for this usage in reality. It has been used in Dharamsala [6] to provide internet access for refugee monks living in the mountainous area in Dharamsala. [7] introduced *SKYMESH* to provide Internet access in a disaster area. It use helium-filled balloons carrying WiFi transceivers, and together they form the mesh network.

The second application of WMNs focus on the communication within the network. For example, in transportation management, [8] proposed a prototype using wireless mesh networks to help with monitoring traffic. The car has sensors to collect data, and the real-time data is gathered using the WMN. Wireless Community Networks are another applications making use of mesh topology. The WCN contains a group of local users setting up and managing a network for themselves. Actually, WCNs can have a gateway to the Internet, and it can act as a backbone to deliver Internet connectivity. However, in reality, most WCNs

[2]Mesh topology is used to denote any network that every nodes in it participating in the distribution of data

were set up to promote an alternative communication channel that offers higher degree of privacy and neutrality [9].WCNs is discussed more in Section 1.2.

1.2 Wireless Community Networks

Application: Wireless Community Network (WCN) [10] is one application of Wireless Mesh Network. WCN is created by a group of local users to have another channel of communication that is self-managed and offer higher degree of privacy. WCNs have their presents on all continents, but most of them are in Europe and North America. The largest WCN all over the world now is Guifi[3] in Spain with over 30000 nodes in 2015.

For WCNs, new router can join the existing WCNs independently by following the technical information listed on the Web. Every nodes in WCNs must have a way to know the existence of the new router, or the unavailability of some other router when it leaves the network. There are several popular routing protocols used in WCNs, taking into account the ever-changing topology of WCNs to deal with that problem. They are Optimized Link State Routing (OLSR), Babel, and B.A.T.M.A.N. But probably OLSR is the most widespread routing protocol in used. More thorough discussion for the mentioned protocols can be found in [11].

In this thesis, we focus on three WCNs: Funk Feuer Wien (FFW)[4] and Funk Feuer Graz (FFG)[5] in Austria and Ninux (NNX)[6] in Italy. And they all use Optimized Link State Routing Protocol (OLSR), where every nodes in the WCNs have the topology information, and compute the next hop destinations for all nodes in the network using shortest hop forwarding paths.

1.3 Contributions of this thesis

In graph theory, centrality metrics (which is explained in depth in Chapter 2) is used to determine how influential a node is in the network. The authors in [12] presented two types of centrality metrics that he considered them to have significant applications in analysing wireless networks. They are degree based centrality, and shortest-path based centrality.

The authors in [9] proposed to use *closeness group centrality* $C_C(\gamma)$ – one type of degree based centrality – to identify the central group in the WCNs. γ is a group of nodes. Knowing the central group, the group with the lowest $C_C(\gamma)$ can be useful in, such as, determining where to place the Internet gateway. Suppose that this WCNs is designed to provide Internet access for its client, then the most central group will give the best average Internet connectivity to the nodes in the network.

The type of centrality metrics to choose depends on the purpose of analysis. [2] also mentioned shortest-path based centrality $C_{sp}(k)$ should be employed when we want choose a set of nodes that have control a fraction η of the entire traffic of the network. There

[3] https://guifi.net
[4] https://www.funkfeuer.at/
[5] graz.funkfeuer.at/en/index.html
[6] wiki.ninux.org/

are 3 scenarios in which betweenness centrality would be employed. Given that WCNs is not a static network, nodes can join and leave the network as they wish. Moreover, there is no single machine managing the whole network, but rather every router has the whole information of the topology and cooperates together to keep the information flowing freely in the network. In sum, we know that betweenness centrality is a useful measure for WCN, and that computing betweenness centrality must not be too demanding for the routers since routers do not have enough computational power to execute as quick as the normal computer. Therefore, to make it feasible for WCNs to detect the set of most influential nodes, the algorithm must run fast enough on the computationally limited physical device.

In this thesis, I test the performance of shortest-path based betweenness centrality (from now on it is called *betweenness centrality* algorithm on a real router typically used in WCNs. The state-of-the-art for calculating betweenness centrality is $\mathcal{O}(mn + n^2 \log n)$ [3]. Even though it is a huge improvement comparing to $\mathcal{O}(n^3)$, it is still not feasible to compute centrality in large graphs. The second algorithm is the heuristic-based presented in [4]. This algorithm follow a divide-and-conquer design paradigm, the network is divided into bi-connected components. Betweenness centrality is calculated for each bi-connected components with a bit of modification, and then they are summed up to achieve the same result as the latest algorithm. The presentation of both algorithms can be found in Chapter 3.

Since the algorithm of [4] is heuristic-based, there is no guarantee on the running time. I carry out an extensive experiments to compare the two algorithms. The experiment is run on synthetic graphs: Erdos, Power Law, Community Network[7]. And it is also run on three real wireless community networks: Funk Feuer Wien (FFW)[8] and Funk Feuer Graz (FFG)[9] in Austria, and Ninux[10] in Italy. And the performance is measured in the virtualized environment with QEMU[11], and on the real router. More details can be found in Section 4.4.

This is the first time that both algorithms for calculating betweenness centrality has been tested on the real device. They both have the same polynomial time-bound, but we show later that heuristic modification can greatly improve the running time for the Community Network graphs.

1.4 Outline of the thesis

The outline of this paper is as follows. The first two chapters cover all the necessary background information. Section Chapter 2 examines the concept of centrality. Since there are numerous measures for centrality, only those centrality measures that are most relevant is presented. And betweenness centrality is going to be discussed in deep since it is the measure that we we choose to evaluate the performance for this thesis. Section Chapter 3 presents two algorithms to calculate the exact betweenness. One is the state-of-the-art algorithm, with the mathematically proven upper bound at $\mathcal{O}(mn + n^2 \log n)$. Another algorithm is

[7] This is not the real wireless community networks, but the generation algorithm was designed such that the topology is similar to the topology found in WCNs

[8] https://www.funkfeuer.at/

[9] graz.funkfeuer.at/en/index.html

[10] wiki.ninux.org/

[11] http://wiki.qemu.org

based on heuristic for speeding up the betweenness centrality computation. And since there is no upper bound for the heuristic algorithm, we want to compare both algorithms to see the performance of both.

The next two chapters deals with the experimental to evaluate performance of these two forementioned algorithms. Chapter 4 gives detail on the experiment setup. This includes the description of the synthetic graphs, and real WCNs topology that we use in the experiment. Chapter 5 summarises the results of the experiment and gives the justification for the results.

Chapter 2

Centrality

Centrality[1] is one the fundamental and useful measure in social network analysis to identify the influential vertices or edges in the network. However, how the vertices (or edges) are considered to be important depend on different centrality measures. In this thesis, we are concerned mainly with betweenness centrality, which belongs to the shortest-path based centrality according to the classification by [13].

Since [14] first described the concept of centrality, many centrality indices were defined and used for different purposes. Since there is no single centrality indices that are supreme over all others, we should consider which centrality indices to use in certain application.

In Section 2.1 we begin with the definition of centrality. Then in Section 2.2 we go deeper on the family of shortest-path based centrality, starting from the stress centrality, moving on to the betweenness centrality, and finally the relative betweenness centrality.

For those who wants to have a broader knowledge on the centrality, then [13] classified centrality for vertices and edges into nine families, and summarized them in great detail.

In different paper, different type of notations are used for the same type of concepts. Hence, to keep this report coherent, I use the same of notation for the same concept, and this might different from the notation used in the original papers.

2.1 Definition of centrality

Centrality index can be roughly defined as a function assigning score representing the importance for nodes or edges in the network. When changing the way of assigning the score for each vertex, we arrive at different centrality indices. However, we cannot have arbitrary function assigning random values for nodes in graph, and call those values the centrality.

Brandes and Erlebach [13] pointed out that even though there is no strict definition for centrality, we can still find the common ground shared by different centrality indices. The underlying concept is that centrality indices must depend only on the structure of the network. To put it simply, centrality indices a way of assigning a score for each vertex (or edge) in the network, based completely on structure of the graph. Formally, he coined the term *Structural Index*, and stated that a centrality index c is required to be a structural index.

[1] it is also referred to by other terms: centrality indices or centrality measures

Definition 2.1.1. Structural Index Let $G = (V, E)$ be a weighted, directed or undirected multigraph and let X represent the set of vertices or edges of G, respectively. A real-valued function s is called a structural index if and only if the following condition is satisfied: $\forall x \in X : G \simeq H \Rightarrow s_G(x) = s_H(\phi(x))$, where $s_G(x)$ denotes the value of $s(x)$ in G

Before beginning with the main section describing betweenness centrality, a simple centrality measure called *degree centrality* is going to be presented, so that reader can have an easy-to-understand example on what is centrality. The score of degree centrality for each node is equal to the number of connection that node has. See Figure 2.1. Then later, we move on to more complex centrality measurements based on shortest-paths.

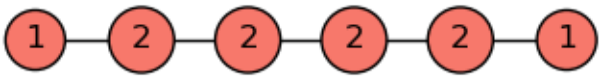


Figure 2.1: Degree centrality score for each node

2.2 Shortest-path based centrality

This section presents two types of shortest-path based centrality: *stress centrality*, and *betweenness centrality*. Basically those two centrality indices are based on the set of shortest paths between a pair of vertices or edges. Section 2.2 and Section 2.2 presents those two measures in deep.

In communication, information does not only flow in the shortest path, where the shortest path might represent the path where the total traversing time is shortest. However, in a lot of application, shortest path are chosen to forward information through the network. For example, OLSR routing protocol provides shortest path routes for all nodes in the network. And OLSR routing protocol is used widely in Wireless Community Networks – the network that we want to apply centrality to improve their existing routing protocol.

Stress Centrality

The concept was presented by [15], the idea is the know quantify how much information is flowing through a vertex in a network. With that definition, if a vertex v lies on more shortest paths, then more information is passed through v and it must handle a higher workload, or "stress". Formally, it is defined in Equation (2.1). It is the summation over all possible source s, and all possible target t that are different than v.

$$c_S(v) = \sum_{s \neq v} \sum_{t \neq v} \sigma_{st}(v) \tag{2.1}$$

where $\sigma_{st}(v)$ denotes the number of shortest paths containing v. To simplify the notation for Equation (2.1), we might write it as follow:

$$c_S(v) = \sum_{s \neq t \neq v} \sigma_{st}(v) \tag{2.2}$$

However for the clarity, I will stick with the notation in Equation (2.1).

Note that the formula for stress centrality $c_S(v)$ does not include the shortest paths that start or end in v. And the stress centrality is calculated for all shortest paths between any pair of vertices.

Betweenness Centrality

The concept of *betweenness centrality* was introduced independently by Freeman [16] and Anthonisse [17]. It can be viewed as a relative version of stress centrality. Here we first define the betweenness centrality, then continue with the motivation for betweenness centrality and its application.

We define $\delta_{st}(v)$ - the *pair-dependency*[2] of a pair $s, t \in V$ on an intermediary $v \in V$.

$$\delta_{st}(v) = \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{2.3}$$

where $\sigma_{st}(v))$ is the number of shortest paths from s to t that are passing through v. And σ_{st} is the number of shortest paths from s to t. $\delta_{st}(v)$ represents the probability that vertex v falls randomly on any of the shortest paths connecting s and t. Assume that the communication in the network follows the shortest path, then $\delta_{st}(v)$ can be interpreted as the probability that vertex v can involve, intercept, enhance or inhibit the communication between s and t.

The betweenness centrality of a single vertex $c_B(v)$ is defined as:

$$c_B(v) = \sum_{s \neq v} \sum_{t \neq v} \delta_{st}(v) = \sum_{s \neq v} \sum_{t \neq v} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{2.4}$$

From the original formula for betweenness centrality in Equation (2.4), several variants for betweenness centrality was introduced in [19], such as edge betweenness centrality, group betweenness centrality, etc. The one we are mostly interested in is the betweenness centrality with endpoints vertices included. It means that the $c_S(v)$ take into account even the shortest paths starting or ending with v, and it doesn't require the source s or the target vertex t to be different from v. Equation (2.5) shows the formula for betweenness centrality with source and target included.

$$c_B(v) = \sum_{s \in V} \sum_{t \in V} \delta_{st}(v) = \sum_{s \in V} \sum_{t \in V} \frac{\sigma_{st}(v)}{\sigma_{st}} \tag{2.5}$$

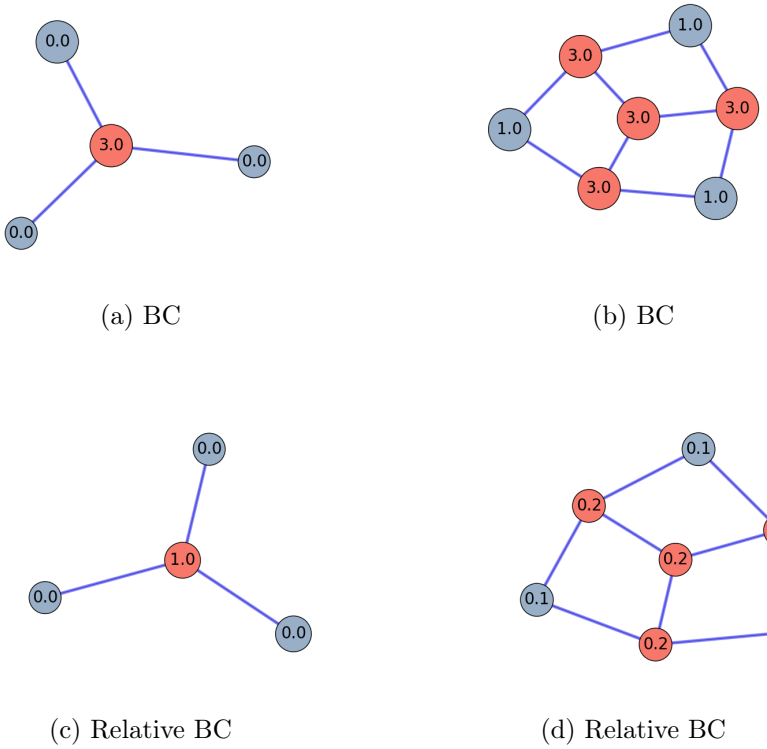
Relative Betweenness Centrality

After introducing betweenness centrality, Freeman presented another measurement [16], called *relative betweenness centrality* under the argument that betweenness centrality cannot

[2] In [18], the term pair-dependency is equivalent with the term *dependency* used in [3]. To keep consistency, in this thesis, the definition of pair-dependency and dependency follow the [3]

be used to compare the influential of vertices from network of different size (e.g. different number of nodes).

He argued that for vertices v, u to have the same betweenness centrality only mean that they have the same potential for control in absolute terms. That means they can facilitate or inhibit the same number of communications. Note, we implicitly assume that all communications are conducted along shortest paths. However, the c_B does not show the relative potential for control within the network. Figure 2.2 illustrate that even though $c_B(v) = c_B(u_i) = 3, i = 1, 2, 3, 4$, the potential for control of vertex v is much larger than vertex u_i. For example, removing vertex v and the network would be disconnected and no communication can happen between vertices. Therefore, v have a total control of the network. Meanwhile, removing any u_i does not have that same disastrous effect since each u_i only control part of the communications between pair of vertices.



(a) BC (b) BC

(c) Relative BC (d) Relative BC

Figure 2.2: Comparision for Betweenness Centrality and Relative Betweenness Centrality: We see that even though the red nodes have the same betweenness centrality score, it is clearly that the red node of the left graph has more control over the traffic flow within the network. For example, when the red node of the left graph is destroyed, the graph is disconnected, and no information can flow between blue nodes. On the other hands, if any red node of the right graph is cut out from the graph, the graph is still connected. The relative betweenness centrality can reflect the important of the red node for both graphs better, by setting it to 1 for the left graph, and 0.2 for the right graph.

When a vertex of interest v is the center node of the star graph, just as Figure 2.2 (a), its

betweenness centrality score is the biggest score that a graph with n nodes can have. And the maximum betweenness centrality score that a vertex can achieve is shown be to:

$$\max c_B(v) = \frac{n^2 - 3n + 2}{2} \tag{2.6}$$

The so-called *relative betweenness centrality* measure $c'_B(v)$ is the fraction between thet betweenness centrality $c_B(v)$ in Equation (2.4) divided by its possible maximum score. See Equation (2.7) for the formula of relative betweenness centrality $c'_B(v)$:

$$c'_B(v) = \frac{2c_B(v)}{n^2 - 3n + 2} \tag{2.7}$$

where $c_B(v)$ is the betweenness centrality defined in Equation (2.4), and n is the number of vertices in the network.

Chapter 3

Computation of Betweenness Centrality

Recall the definition of betweenness centrality:

$$c_B(v) = \sum_{s \neq t \neq v \in V} \delta_{st}(v) = \sum_{s \neq t \neq v \in V} \frac{\sigma_{st}(v)}{\sigma_{st}}$$

Traditionally, just by looking at the equation, we normally divide the task of computing betweenness centrality into 2 steps:

- First we need to compute tables with length and number of shortest paths between all pairs of vertices in the network;
- Second for each vertex v, we need to consider all the pair s, t, we use the tables calculated above to determine the fraction of shortest paths between s, t that are passing through v. Summing all the fraction for every possible s, t and we arrive at the $c_B(v)$. This way of summing is referred to by the term *naive summation of pair-dependencies*.

The chapter starts with presenting some important concept in Section 3.1. Then Section 3.2 gives an overview on classic ways to find the shortest paths. And Section 3.3 introduces 2 ways to sum all the pair-dependencies: (i) the naive way, (ii) the faster approach to accumulate pair-dependencies gradually by [3]. Up until now, the approach in [3] is the state-of-the-art in calculating the exact betweenness centrality, and it is implemented in many standard graph libraries such as *networkx*[1], *Boost Graph Library*[2]

The Section 3.4 intends to summarize the result of two previous section by recapping the complexity of some algorithms used in determining betweenness centrality.

After having a throughout background, Section 3.5 presents the heuristic way to calculate also the exact betweenness centrality by [4]. However, the complexity of the Heuristic Betweenness Centrality is not guaranteed to perform better than the algorithm of [3], its performance depends on the topology.

[1] https://networkx.github.io/

[2] www.boost.org/libs/graph/doc/

3.1 Important concepts

Define the set of *predecessors* of a vertex v on the shortest path from s as:

$$P_s(v) = \{u \in V : u, v \in E, d(s,v) = d(s,u) + \omega(u,v)\} \tag{3.1}$$

where $d(s,u)$ is the minimum length of any path connecting s and u. And $\omega(u,v)$ is the weight of the edge u, v.

Given $P_s(v)$, then we have the following relation:

$$\sigma_{st} = \sum_{u \in P_s(v)} \sigma_{su} \tag{3.2}$$

3.2 Counting the number of shortest paths

Before we begin, there are 2 types of shortest paths problem. The first one is called Single Source Shortest Paths (SSSP), where we compute the shortest paths between one given source vertex s to all other vertices. The second one is All-Pairs Shortest Paths (APSP) where we compute the shortest paths between all vertex pairs.

The SSSP is a classical problem in Graph Theory, and it is usually solved by Breadth-First Search (BFS) for unweighted graphs in $\mathcal{O}(m+n)$, and Dijkstra's algorithm for the weighted graphs in $\mathcal{O}(m + n \log n)$.

The APSP can be computed straightforwardly by applying algorithm for SSSP n times, for all vertex $s \in V$. The resulting time complexity thus will be $\mathcal{O}(mn + n^2)$, and $\mathcal{O}(mn + n^2 \log n)$ respectively. In the sparse graph, it is a good practice to solving the SSSP multiple times. However, in non-sparse graph, where the number of edges is close to the maximal number of edges, i.e. $m \simeq n(n-1) \simeq n^2$, then it is better to use Floyd/Warshall algorithm with the complexity of $\mathcal{O}(n^3)$.

3.3 Summing all the pair-dependencies

Naive ways

In order to calculate the betweenness centrality for vertex v, we have to sum all the pair dependencies $\delta_{st}(v)$ for all possible source s and target t that are different from v. The first part of the algorithm calculates all the distance d between all pair of vertices. Vertex v lies on the shortest path between a pair (s,t) if $d(s,t) = d(s,v) + d(v,t)$. And the number of shortest path from s to t passing through v is calculated as $\sigma_{st}(v) = \sigma_{sv} \cdot \sigma_{vt}$.

Therefore, summing of all possible $\delta_{st}(v)$ yields $\mathcal{O}(n^2)$, and we have the betweenness centrality per vertex v: $c_B(v)$. Doing for all other vertices in the network, and it yields $\mathcal{O}(n^3)$.

Gradually accumulation of pair-dependencies

Brandes noticed that there is a better way to sum all the pair dependencies [3]. The basic way is to finish completely the first part of finding shortest paths and distance between all

pair of vertices, and then start with the summation of pair dependencies. The better way is achieved by exploiting the recursive relation of the *dependency* $\delta_{s\bullet}(v)$ of a vertex $s \in V$ on a single vertex $v \in V$, defined as:

$$\delta_{s\bullet}(v) = \sum_{t \in V} \delta_{st}(v) \tag{3.3}$$

[3] proved that the *dependency* of $s \in V$ on any $v \in V$ obeys:

$$\delta_{s\bullet}(v) = \sum_{w: v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\bullet}(w)) \tag{3.4}$$

The set of predecessor of vertex w on the shortest path from s: $P_s(w)$ can be obtained from the first part of the algorithm. And the dependencies of s on all vertices can be computed in $\mathcal{O}(m)$. Therefore, instead of solving the APSP algorithm from start to end, we solve the SSSP for each source vertex $s \in V$. After completing the SSSP algorithm, we calculate the dependencies of s as laid out in Equation (3.4).

Next, we look at the relation between the betweenness centrality for $v \in V$ $c_B(v)$ and dependency $\delta_{s\bullet}(v)$. Summing all the dependencies of s inV on a single vertex v, and we arrive at the betweenness centrality $c_B(v)$ as in the following equation:

$$c_B(v) = \sum_{s \neq v} \delta_{s\bullet}(v) \tag{3.5}$$

Therefore, we can then gradually accummulate the betweenness centrality score $c_B(v)$ by summing up the dependency $\delta_{s\bullet}(v)$ to $c_B(v)$.

Next section is intended to provide the high-level on the algorithm to calculate betweenness centrality

3.4 Complexity of determining betweenness centrality as a whole

This section summarizes the complexity for naive way, and for Brandes' algorithm to calculate betweenness centrality. The Algorithm 1 run in $\mathcal{O}(n^3)$ for both unweighted and weighted graphs. And the Brandes's algorithm in Algorithm 2 can run in $\mathcal{O}(mn)$ for unweighted graph, and $\mathcal{O}(mn+n^2 \log n)$. The breaking down of the computational time of those two algorithms are given below.

For Algorithm 1, the overall computation is dominated by line 2 to sum all the pair dependencies.

Algorithm 1 Basic way to calculate betweenness centrality

```
1: procedure BASIC BETWEENNESS CENTRALITY
2:     Solving APSP problem
3:     Summing all pair-dependencies
```

For Algorithm 2, within the `for` loop, for unweighted graph, each execution from line 2 - 3 executes in $\mathcal{O}(m)$. For weighted graph, solving SSSP problem dominates the computation with $O(m + n \log n)$ for unweighted graph. The `for` loop is executed n times. Thus, we have the running time stated in the beginning of the section.

Algorithm 2 Brandes betweenness centrality

```
1: procedure BRANDES BETWEENNESS CENTRALITY
2:     for s ∈ V do do
3:         Solving SSSP problem for source s (with BFS or Dijkstra's algorithm)
4:         Finding all the dependencies of s for all other vertices v Equation (3.4)
5:         Accumulate the betweenness centrality for vertices v Equation (3.5)
```

3.5 Heuristic algorithm for the exact betweenness centrality

We start the section with the high-level description of steps in the heuristic algorithm. We also provides definition for some term that deem to be important. For the details and all the proofs, reader can always refer to [4]

Puzis et al. uses the insight that computing betweenness centrality on tree can be done in linear time in [4]. From now on this algorithm is denoted as HBC. He defined a heuristic algorithm to speed up the betweenness centrality calculation. The original graph is partitioned into one or many bi-connected components (BCCs). Then the bi-connected components tree is formed, and several analysis is performed on the BCC tree. The summary of HBC's steps are as follow:

1. Divide the graphs into bi-connected components and form BCC tree;
2. Compute the *Component Tree Weight*. This step involves the knowledge of the whole graph;
3. Form the matrix with the *communication intensity* value for each pair of vertices in a BCC;
4. Calculate the betweenness centrality for each vertices in each BCC using Equation (3.6);
5. Finalize the betweenness centrality for the original graph.

Partitioning a graph into bi-connected components

Given a graph $G = (V, E)$. If the removal of any $v \in V$ disconnects G, then v is said to be a *cut-point* (or *articulation point*). A graph is said to be *bi-connected* if at least 2 vertices must be removed to disconnect the graph. Figure 3.1 from [4] gives an example of partitioning the graph into BCCs.

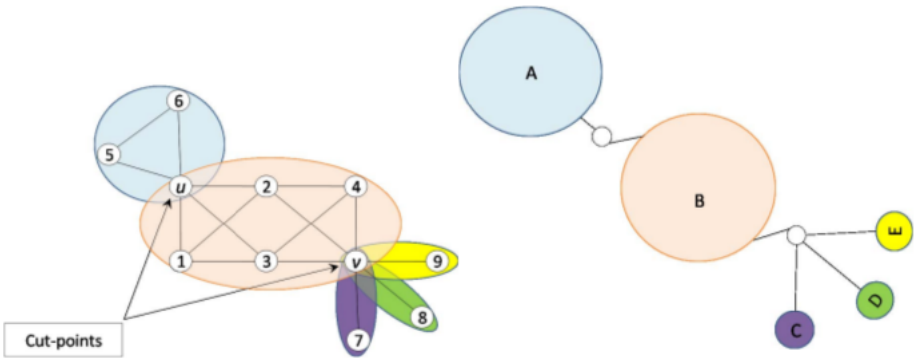


Figure 3.1: BCC partition and cut-points. Reproduced from [4].

Component Tree Weight

The Component Tree Weight is intended to keep track of how many number of nodes can be reach via a cut-point of a specific BCC when that BCC is separated from the original graph.Let B be a bi-connected component and $v \in B$ be a cut-point. Let $B^+(v)$ be the *connected component* in G which includes B after the removal of v. In another term, $B^+(v)$ include all the vertices in G that are connected to v via B. An example is for a bi-connected component BCC_B in Figure 3.1. After removing vertex v, the connected component $BCC_B{}^+(v) = \{5, 6, u, 1, 2, 3, 4\}$. The *cut with* value is defined to be the size of $D_B^{v+} = |B^+(v)|$.

Define $B^-(v)$ as a set of vertices that can only be reach through v from B. Therefore, when v is cut out from the graph, those vertices in $B^+(v)$ (a connected component including B without the vertex v) cannot reach $B^-(v)$. The "cut without" value is defined to be $D_B^{v-} = |B^-(v)|$. For example, when removing vertex v from BCC_B, $BCC_B{}^-(v) = \{7, 8, 9\}$. Those vertices cannot be reach from $B^+(v)$ anymore.

The relation between "cut with" and "cut without" is as follow: $|B^+(v)|+1+|B^+(v)| = |V|$. It can be thought as the number of vertices.

Table 3.1 shows the value assigned to Component Tree Weight after the Step 2.

Table 3.1: Component Tree Weight calculation for Figure 3.1

Component	Cut-point	"Cut with" value	"Cut without" value
BCC_A	u	2	8
BCC_B	u	8	2
BCC_C	v	7	3
BCC_D	v	1	9

Communication intensity

For each bi-connected component B_i, we calculate the traffic matrix h_{st}^i representing the *communication intensity* between a pair of vertex s, t. The traffic matrix is symmetric. There are 3 cases:

- If x and y are not cut-points, then $h_{xy} = 0$ for $x = y$ and $h_{xy} = 1$ for $x \neq y$
- If x is not a cut-point, and y is a cut-point, then $h_{xy} = D_B^{y-} + 1$
- If both x and y are cut-points, then $h_{xy} = (D_B^{x-} + 1) \cdot (D_B^{y-} + 1)$

Betweenness computation for bi-connected component

The betweenness centrality calculated

$$c_B(v)_{v \in B_i} = \sum_{s,t \in B_i} \delta_{st}(v) \dot{h}_{st}^i \tag{3.6}$$

And the heuristic algorithm also use the faster Brandes's algorithm to calculate betweenness centrality. Recall the equation for summing the dependency of $s \in V$ on any $v \in V$

$$\delta_{s\bullet}(v) = \sum_{w: v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_{s\bullet}(w))$$

For any bi-connected component B_i, the dependency of $s \in B_i$ on any $v \in B_i$ can be written as follow:

$$\delta_{s\bullet}(v) = h_{sv}^i + \sum_{w: v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \delta_{s\bullet}(w) \tag{3.7}$$

Betweenness computation for original graph G

It was proven in the paper that for non-cut-point, the betweenness value calculated locally in each B_i is equal to the betweenness value calculated on the whole graph.

$$c_B(v) = c_B(v)_{v \in B} \tag{3.8}$$

For the cut-point betweenness value, we first have to figure out the inter-component communication, denoted as $BC^{inter}(v)$:

$$BC^{inter}(v) = \sum_{B \ni v} D_B^{v-} \cdot D_B^{v+} \tag{3.9}$$

For cut-point v, the betweenness value on the whole graph is defined as:

$$c_B(v) = \sum_{B \ni v} c_B(v)_{v \in B} - BC^{inter}(v) \tag{3.10}$$

Chapter 4

Experiment

The previous chapter presents two algorithms to calculate Betweenness Centrality (BC). One algorithm was found out by Brandes in [3]. For the ease of notation, from now on it is called *BBC*. The other heuristic version to calculate betweenness centrality was devised by Puzis et al. [4], and its short-form name from this point is *HBC*.

This chapter presents all the setup information to compare the running time of two algorithms. In order to perform the analysis, there are few things needed to address first. To analyze the running time of the algorithm, we need to know about all the factors that can influence the running time. Those factors are the implementation, the type and size of graph, the system that runs the algorithms. The following section addresses those factors in depth. Section 4.1 presents the implementation of BBC and HBC were run on the physical router and server. Section 4.2 provides the hardware specifications for the system we use. Section 4.3 concerns with the technical side of porting the program to embedded device, in our case is the Ubiquiti router. This section contains a lot of technical note, which might be served as a guideline on how to make a custom package for embedded system. Section 4.4 presents about two types of input that we use in the experiment: the randomly generated graphs, and the real topologies of the WCNs. Lastly, there are fluctuation when measuring the running time of the algorithms. Section 4.5 presents the method to get the average running time, how the experiment was carried out, e.g. how many graphs was generated, how many times the algorithm was run for one specific input.

4.1 Implementation

The BBC is implemented in the Boost Graph Library[1] (BGL). That is the generic library with implementation for many classical graph algorithms.

From Section 3.5, the implementation for HBC contains four main steps. The first step involves dividing the main graph into some bi-connected components B_i. And then for each articulation points in all B_i, the link weight is computed. To calculate the link weight, we need to use the information from the original graph. Second, for each B_i, create the traffic matrix h^i to represent the *communication intensity* between all pair of vertices in each B_i.

[1] http://www.boost.org/doc/libs/1_60_0/libs/graph/doc/index.html

Table 4.1: Specification of a computer

CPUs:	2 × Intel(R) Xeon(R) CPU E5-2630 v3 2.40GHz
Memory:	4 × 16GB 1866 MHz

The third, and the final part is the procedure to calculate betweenness for each B_i. Since the original graph is divided into some bi-connected components B_i, performing the original BBC for each B_i will not generate the correct result. Actually, the accumulation step, as shown in Equation (3.6) involves multiplying the pair-dependency $\delta_{st}(v)$ with its corresponding communication intensity value h^i_{st}. For an ease of comparison, the pseudo code of the accumulation step for BBC and HBC are presented in Algorithm 3 and Algorithm 4, respectively.

Algorithm 3 Accumulation procedure for BBC

1: **procedure** ACCUMULATION
2: $c_B[s] \leftarrow c_B[s] + (|S| - 1)$
3: **for** $v \in V$ **do** $\delta[v] \leftarrow 0$
4: **while** S not empty **do**
5: pop $w \leftarrow S$
6: **for** $v \in Pred[w]$ **do** $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]}(1 + \delta[w])$
7: **if** $w \neq s$ **then** $c_B[w] \leftarrow c_B[w] + \delta[w] + 1$ ▷ w is target of s once

Algorithm 4 Accumulation procedure for HBC

1: **procedure** ACCUMULATION
2: **for** $v \in V$ **do** $\delta[v] \leftarrow 0$
3: **while** S not empty **do**
4: pop $w \leftarrow S$
5: $h \leftarrow$ get_the_communication_intensity_between (w, s)
6: $c_B[s] += h$
7: **for** $v \in Pred[w]$ **do** $\delta[v] \leftarrow \delta[v] + \frac{\sigma[v]}{\sigma[w]}(h + \delta[w])$
8: **if** $w \neq s$ **then** $c_B[w] \leftarrow c_B[w] + \delta[w] + h$

4.2 Hardware specifications

Server

The experiment is run on the machine provided by Advanced Networking Systems (ANS) lab[2], with the hardware components as in Table 4.1

The machine was running GNU/Linux with kernel 3.16.

[2]https://ans.disi.unitn.it/

Table 4.2: Specification of Ubiquiti NanoStation M2 router

Platform:	Atheros AR7240
CPU MHz:	390
Flash MB:	8
RAM MB:	32

WiFi Router

We use Ubiquiti NanoStation M2 router to test the performance. The specification is in Table 4.2. Ubiquiti router is flashed with a firmware obtained from OpenWrt as described in the next section.

4.3 Porting a program to OpenWrt

OpenWrt[3] "is described as a Linux distribution for embedded devices.". It allows people to build custom firmware to run on many supported devices. And it aids with cross compiling the package which can be installed on those devices with custom firmware.

Even though we can build the firmware image with OpenWrt toolchain. However, there is the firmware image available on the Download page

ootnotehttps://wiki.openwrt.org/toh/hwdata/ubiquiti/ubiquiti_nanostationm2 for Ubiquiti NanaStation M2 router, so we use it and flash the device with that firmware.

The source code of the program running on the machine with GNU/Linux kernel 3.16, and the program running on Ubiquiti router are the same. To make it possible to run a program in the Ubiquiti router, OpenWrt helps with compiling the source code into a package that can be installed on Ubiquiti router.

4.4 Input data

The experiment involves testing with 2 main sources of input: the artificially generated graphs, and the real topologies from the Wireless Community Networks.

4.4.1 Synthetic graphs

For randomly generated graphs, we are interested in three types: Erdos, Power Law, and Community Graph. All the graphs were generated following the open-source code made available by Baldesi and Maccari [20] in a framework called *NePA TesT*. NePA TesT provides a way to generate synthetic graph for different graph type such as Erdos, Power Law, and Community Network.

The reason why we wanted to test on synthetic graphs is because we want to know how BBC and HBC would perform with different graph structures, and graph sizes. And as will

[3]https://openwrt.org/ – the most widely used open-souce operating system for wireless routers.

Table 4.3: All synthetic graphs generated for the experiment

Graph type	Number of nodes	Edges / Nodes Ratio
CN	from [100, 1000] with the step 100	1.2
ER	from [100, 1000] with the step 100	3
PL	from [100, 1000] with the step 100	1.2

see later in Chapter 5, performance of BBC and HBC depends on the structure and property of each graph type.

For every line in Table 4.3, 30 different graphs were generated per size. So for CN graphs, we have 10 different sizes multiplies by 30 graphs, and we have 300 CN graphs in total. For all three graph types, we have 900 graphs. We use the notation *CN100*, *PL100*, *ER100* to denotes the Community Network, Power Law and Erdos graphs with 100 nodes, respectively. The same format of notation is used for graphs with larger number of nodes.

4.4.2 Real topologies of WCNs

In conjunction with the synthetic networks, we also want to evaluate BBC and HBC on the real networks. Since in the end, what matters is which algorithm would work better in reality. In this thesis, we focus on three WCNs: Funk Feuer Wien (FFW)[4] and Funk Feuer Graz (FFG)[5] in Austria and Ninux (NNX)[6] in Italy.

The WCNs are ever evolving with some current node leaving the network, while some new nodes might join it. [9] provides a depth and throughout analysis on these networks for a week [9]. He found out that the fluctuation in the number of nodes and the number of links are small. According to him, those small changes are from a small subset of leaf nodes leaving and entering network sporadically. Since that fluctuations happen at peripheral nodes, the shortest paths for pair of nodes did not vary much with other time-periods. The reason is that those peripheral leaf nodes does not lies in the shortest paths when routing for other nodes. Hence the performance of shortest-path based betweenness measurement varies slightly. There are other measurements taken as well, but the conclusion from the paper is that for a week, the network features of these real WCNs are stable. With the above analysis, we limit the performance testing of BBC and HBC on one snapshot for each WCNs in consideration.

4.5 Experiment Design

Those previous subsections present all the component of the experiment. We have the input data composing of synthetic and real graphs. We have 2 algorithms needed to evaluate: BBC and HBC. We specified the target system to evaluate the running time of the 2 mentioned algorithms. We start the section with definition of running time. The execution step of the

[4]https://www.funkfeuer.at/

[5]graz.funkfeuer.at/en/index.html

[6]wiki.ninux.org/

```
#include<time.h>

clock_t begin = clock()

// Run BBC algorithm / HBC algorithm for one graph

clock_t end = clock()

double running_time = double (end - begin) / CLOCKS_PER_SECOND

```

Figure 4.1: Code to calculate the CPU time

Graph type	On the computer	On Ubiquiti router
CN with nodes in the range [100, 1000]	Yes	Yes
ER with nodes in the range [100, 1000]	Yes	No
PL with nodes in the range [100, 1000]	Yes	No
Ninux	Yes	Yes
FFG	Yes	Yes
FFW	Yes	Yes

Table 4.4: Summary of the number of runs (repetitions) for different input graph

experiment is presented after, aka how many times one single algorithm (BBC or HBC) is evaluated for each input graph.

Every time we mention about the running time, or execution time of BBC or HBC algorithm, we are talking about the time that a processor spent on calculating result for each algorithm. In C++, there is the function call in standard library to help measure the number of CPU clocks consumped by the program.

The experiment consists of running each algorithm 10 times for each input graph. In this experiment, the measurement we want to get is the running time for each algorithm. And we want that measure to be as close to the actual value as possible. When running for one time, there are various things which might affect the result. And we need to repeat the experiment multiple times to make sure the measurement we get is not a coincidence.

Note: Though we run the program on the machine with CPU containing 8 cores and supporting hyper threading, the program does not support parallelism. One program only run on one single thread from the beginning to the end.

For example, for the Ninux graph, we will run BBC 10 times with it, and then we will run HBC 10 times. The running time were all noted down for the analysis.

Chapter 5

Results

The chapter begins with the result from the synthetic graphs in Section 5.1. Since HBC only improve performance for one type of synthetic graphs – the Community Network graphs, then in Section 5.2 we provides commentary on the reason HBC works for CN, but not the other topology types ER and PL. Then we moves on to the next section presenting about the three real topologies of WCNs – Ninux, FFG, FFW in Section 5.3. The chapter ends with the running time analysis for all those WCNs in both the Ubiquity router and a computer.

5.1 Running time for synthetic graphs

As mentioned in Table 4.4, only the Community Network synthetic graphs are run on both the computer and router. While the remaining two synthetic graphs – Erdos and Power Law – are only tested for performance with the computer.

The value on the y-axis for the running time of HBC and BBC for those figures 5.1, 5.2, 5.6 is the average of average.

Let $r_{ij}^{(100)}, i \in [1, 10]$ (because we repeat the same measurement 10 times) denote the running time of BBC algorithm for a CN graph with 100 nodes $g_j^{(100)}, j \in [1, 30]$ since for each type of graphs, we generate 30 graphs in total. Then the value on the y-axis corresponding to graph with 100 nodes is the average running time of all the repetitions for all CN graphs with 100 nodes. Formally, check out Equation (5.1)

$$v^{(100)} = \frac{1}{300} \sum_{j=1}^{30} \sum_{i=1}^{10} r_{ij}^{(100)} \tag{5.1}$$

For the CN graphs, Figure 5.1 summarizes the running time observed for graphs with different node types. We see a huge improvement of time of HBC over BBC when performing the calculation on both the computer and the router. The difference between two graphs is the scale of the y-axis. The program executes much faster in the computer than in the router, but this is simply due to the higher processing power of the computer. What is interesting from Figure 5.1 is how fast HBC complete the calculation comparing to BBC. We see that for CN graphs with 1000 nodes, HBC finishes the computation 100 faster than BBC with the computer, and 60 times fater with the router. Also for CN graphs with 1000 nodes, with

the router, BBC takes more than 6 minutes to compute betweenness centrality, while HBC can finish in less than 7 seconds. This is the stark constrast on the performance. Given the router has small processing power, and it has other tasks to do such as forwarding packets, faster execution time can result in better performance overall. From looking at a graph, we can see that HBC running time increase linearly, while BBC running time quickly soars.

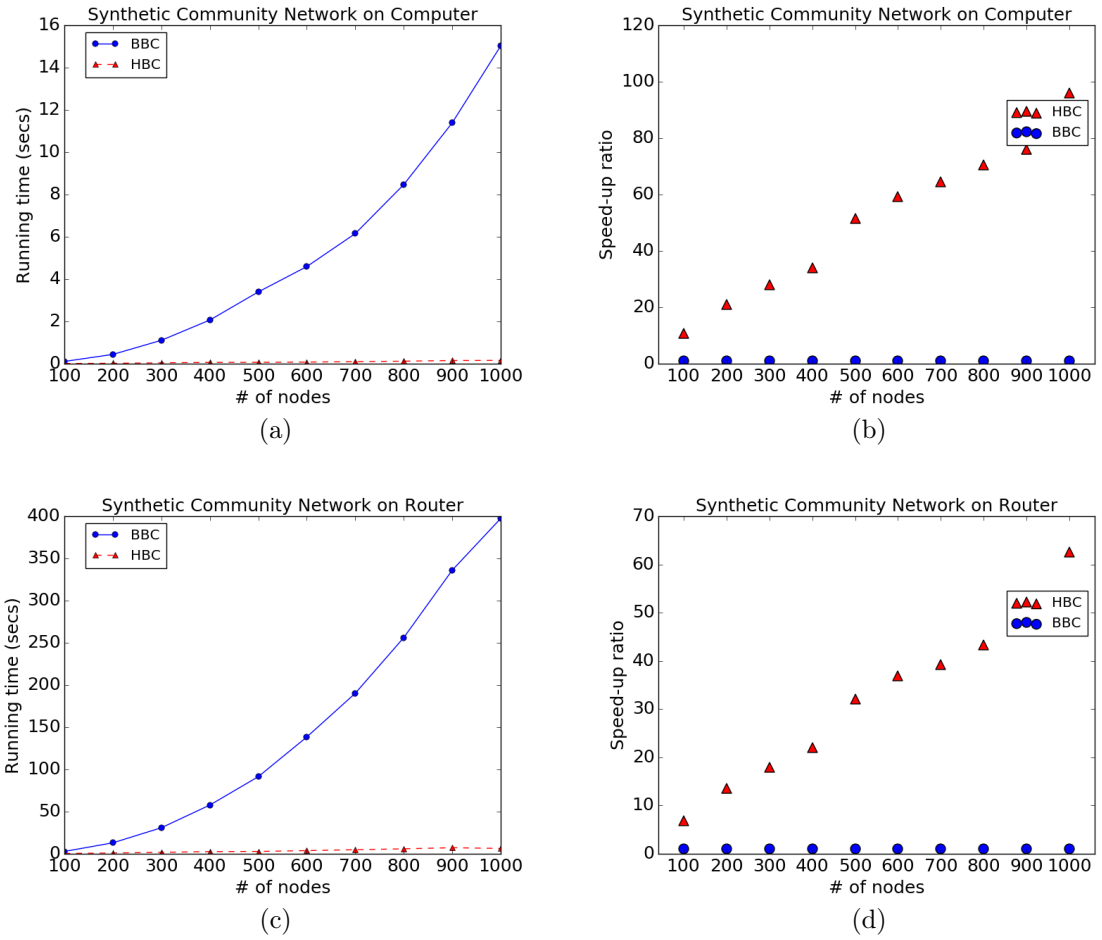
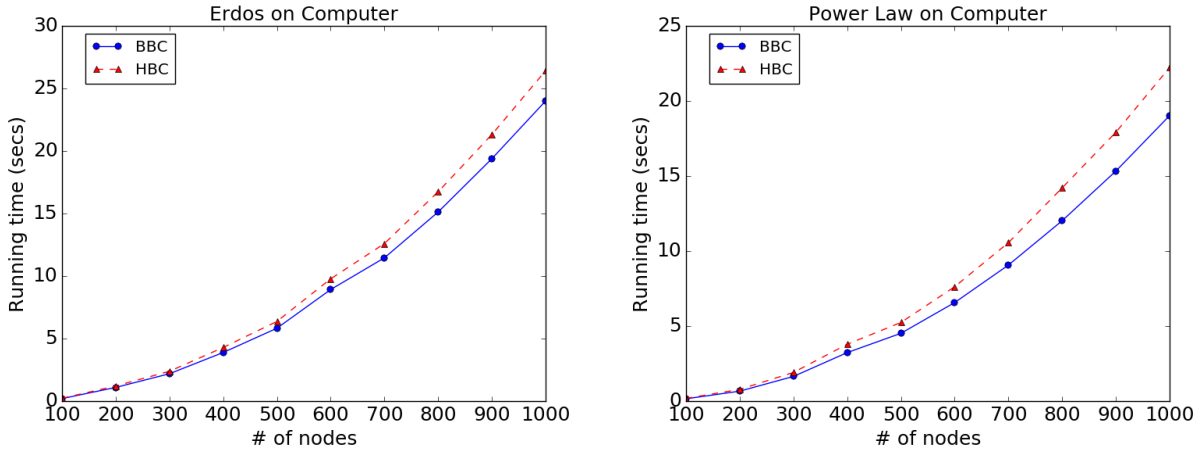


Figure 5.1: Performance for CN graphs on the computer and the Ubiquiti router. Left column shows the running time for both algorithms. Right column shows the speed-up ratio between HBC and BBC.

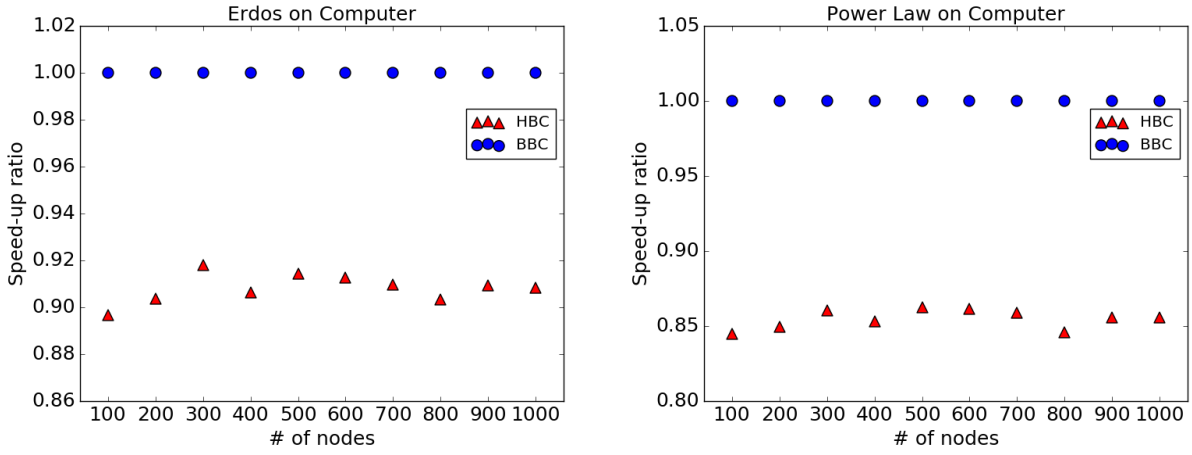
We have just looked at one type of graph where HBC outperforms BBC - the CN graphs. Next we are going to look at two other types of graph where using HBC worsens the overall performance. Figure 5.2 shows the running time on server for Power Law and Erdos graphs. Both lines follow the same pattern: the running time of BBC is a bit smaller than HBC's running time.

The question then arises is when HBC can offer improvement over BBC, what kind of topologies should we use HBC to calculate betweenness centrality.



(a) (b)

Figure 5.2: Performance for Erdos and Power Law graphs on the computer.



(a) (b)

Figure 5.3: Speed up ratio of HBC over BBC

5.2 Synthetic graphs analysis

In the previous section, we noticed that HBC helps improving the performance of Community Network graphs. While for Erdos and Power Lab graphs, HBC takes longer time to run. This section analyzes all the 900 synthetic graphs to give an explanation on why HBC improves performance for some graphs, but not for the others.

The section has two parts, the Section 5.2 seeks answer for whether the number of bi-connected components (BCCs) found in a single graph affect the performance. Section 5.2 justifies that the number of vertices (or nodes) in any single bi-connected components is the main factor affecting the performance.

Notations: For each graph type [CN, PL, ER] and for each graph size [100, 200, ..., 1000], we generated 30 graphs in total. We call those graphs $G^{(i)}, i \in [1, 30]$. For each $G^{(i)}$, suppose that we can find $n^{(i)}$ bi-connected components, $B_j^{(i)}, j \in [1, n^{(i)}]$.

The number of bi-connected components

The average number of BCCs for each type of graphs is summarized in Table A.1, and visualized in Figure 5.4.

For each graph type, and for each graph size:

$$N_{BCC} = \frac{1}{30} \sum_{i=1}^{30} n^{(i)} \tag{5.2}$$

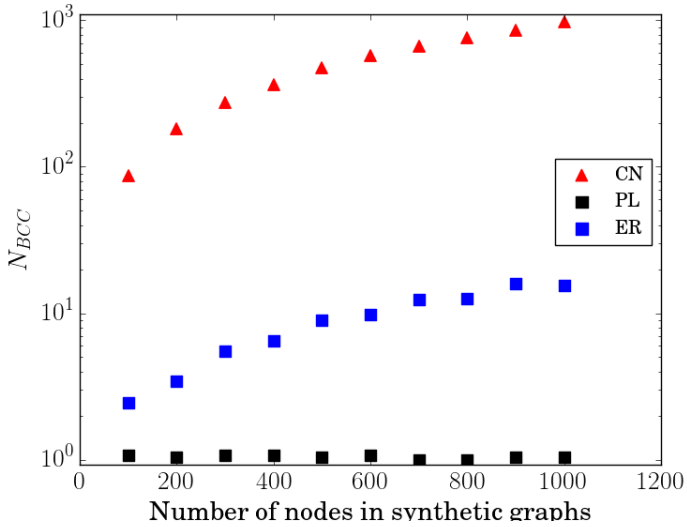


Figure 5.4: Average number of BCCs for synthetic graphs. Note that the y-axis is drawn in log-scale with base 2.

For CN graphs, the number of BCCs is almost equal to the number of nodes in the graph. While for PL graphs, in general we find only 1 BCC, and this BCC is the original PL graph. The case for ER graphs is a bit different, there are several BCCs found, and the number of

BCCs found increase with the number of nodes in ER graphs, e.g from 2.4 average number of BCCs for graph with 100 nodes, up to almost 16 BCCs for graph with 1000 nodes.

Our hypothesis is that if the original graph can be divided into many BCCs, then HBC will save the running time. However, the data shows us that even though ER graphs can be divided into many BCCs, HBC does not perform better for ER graph, see more in Figure 5.2. Hence, the number of BCCs can not be used as a prediction for whether or not HBC would decrease the calculating time for betweenness centrality.

The number of node in each bi-connected component

In the previous Section 5.2, we showed that the number of BCCs found in each graph cannot predict whether HBC would run faster comparing to BBC. This subsection takes a different approach, it analyzes the bi-connected components in term of the its size – the number of nodes in a bi-connected component. More specifically, we focus on the biggest BCC.

After the preprocessing step done in HBC algorithm, then it uses a modified version (see Algorithm 4 of the original Brandes Betweenness Centrality to accumulate the betweenness centrality score. Therefore, if the size of each BCC is as big as the original graphs, then HBC does not save time when calculating betweenness. Not to mention the fact that HBC adds overhead to the overall running time when it pre-processes the original graph.

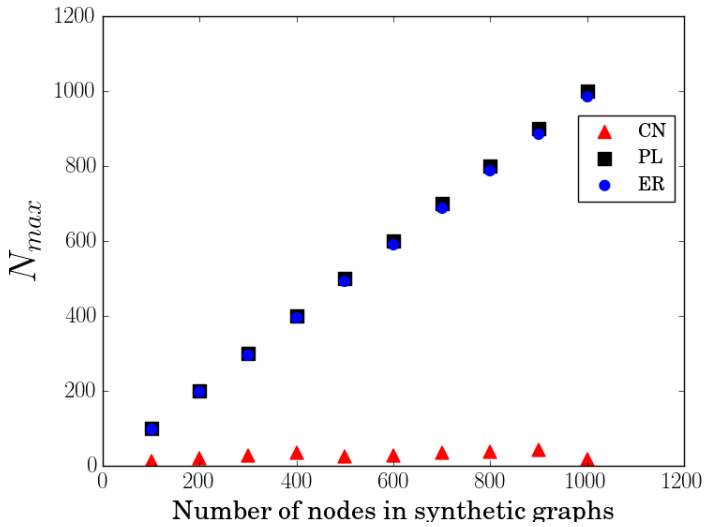


Figure 5.5: The biggest bi-connected components for each synthetic graph

Let $MB^{(i)}$ denotes the maximum number of nodes for a bi-connected component in a graph $G^{(i)}$:

$$MaxB^{(i)} = \max(B_j^{(i)}), j \in [1, n^{(i)}] \tag{5.3}$$

Let $|B_j^{(i)}|$ denotes the number of nodes within each bi-connected component $B_j^{(i)}$. Then the y-value N_{max} in the Figure 5.5 is defined as follow:

$$N_{max} = \frac{1}{30} \sum_{i=1}^{30} MaxB^{(i)} \tag{5.4}$$

For each graph type [CN, PL, ER] and for each graph size [100, 200, ..., 1000], we calculate the average size of the biggest bi-connected component for each graph following the Equation (5.4). And the result is plotted in Figure 5.5. The graph shows that for PL and ER, the largest BCC contains almost all the nodes in the original graph (see the scatter points for PL and ER seems to follow the straight line dividing the Q1 into two halves).

In the case of CN graphs, even for the graph size of 1000 nodes, the largest BCC on average only have 17 vertices. Therefore, computing the betweenness centrality for such the small BCC will improve the running time of HBC significantly.

For the raw data, see Table A.2

5.3 Three real WCN

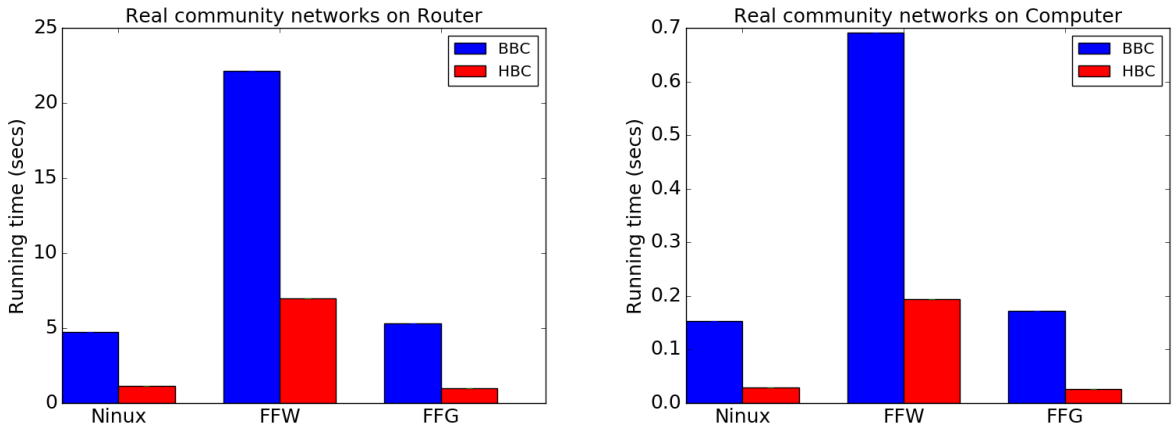
Table 5.1 shows the high-level features of the FFG, FFW, Ninux such as the number of vertices and nodes, and the analysis on the bi-connected components. It shows that the number of nodes in the largest bi-connected component accounted for 27%, 46%, 35% the total nodes in the FFG, FFW and Ninux network, respectively.

	FFG	FFW	Ninux
number of vertices	126	235	126
number of edges	181	370	143
edges/nodes ratio	1.44	1.57	1.13
density	0.022	0.013	0.018
number of bi-connected components	67	120	80
max number of nodes in BCC	35	107	44

Table 5.1: Features of the network under analysis

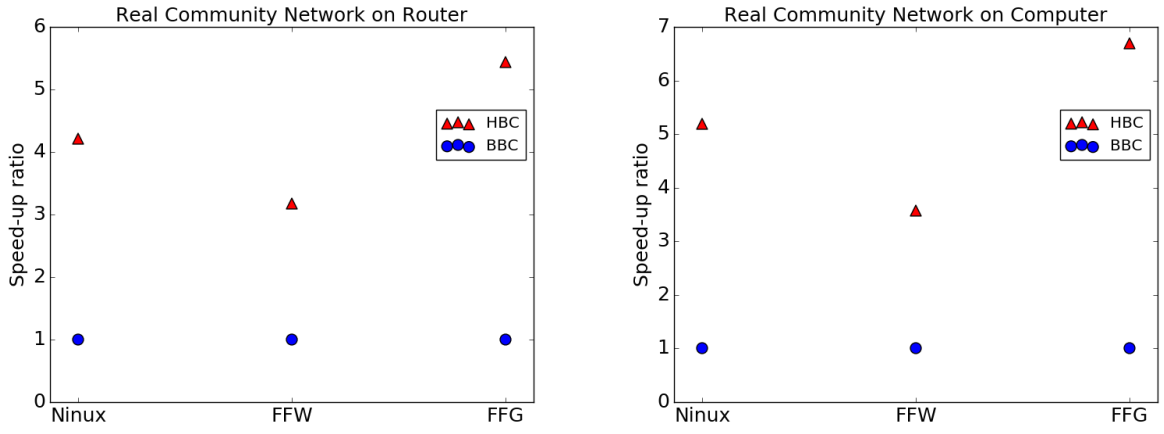
For the real wireless community networks, HBC consistently outperform BBC in the running time. In the case of FFG, where the largest bi-connected component includes 27% of total nodes, we also observe the largest speed-up ratio between HBC and BBC – more than 6 times when running on the router. On another hand, for FFW, where the biggest bi-connected component has 46% of total nodes, we only observe the speed-up ratio around 3. But in all cases, HBC reduces the time to calculate betweenness centrality for all real topologies. For the running time of HBC and BBC, see more in Figure 5.6; for the speed-up ratio, check out Figure 5.7. The raw data is provided in Table A.3 and Table A.4

The raw data is also provided in Table A.4 and Table A.3



(a) (b)

Figure 5.6: Running time of HBC and BBC for real WCNs. The X-axis denotes name of Wireless Community Networks.



(a) (b)

Figure 5.7: Speed up ratio of HBC over BBC for real WCNs. The X-axis is the name of Wireless Community Networks

Chapter 6

Conclusion

In this work, we presented the performance evaluation for two algorithms to calculate betweenness centrality on Ubiquiti router. One state of the art algorithm was found out by Brandes in [3] (called BBC). Another algorithm is the heuristic version aimed to improve the Brandes's algorithm (denoted HBC). On the router, we tested with three real topology of Wireless Community Networks (WCNs), and it was shown that those topology, HBC were able to calculate faster than BBC from 3 to 7 times in Figure 5.7.

We also evaluate the performance for synthetic Community Networks graphs as well. Those graphs were generated algorithmically using *NePa TesT* [20]. The result for the synthetic CN networks is promising. When running on routers for synthetic CN graphs with 1000 nodes, HBC achieves a speed up ratio of 60 times over BBC. On average, when running on the Ubiquiti router, a CN network with 1000 nodes take almost 7 minutes with BBC, and less than 7 seconds with HBC. This is the huge improvement given the limited capacity of the router. Testing with synthetic CN graphs, the result also showed that HBC improved the performance over BBC.

Besides CN graphs (both the real topology and the randomly generated ones), we also performed the analysis with synthetic Erdos and Power Law graph. HBC takes longer time than BBC to finish the calculation. For example, for Erdos graph with 1000 nodes, BBC takes 24 seconds, and HBC takes 26.4 seconds. HBC even though does not guarantee that the performance would be improved over the BCC, its complexity in worst case is also the same as BBC ($\mathcal{O}(mn)$ for an unweighted graph, and $\mathcal{O}(mn + n^2 \log n)$ for a weighted graph).

We also carried out an analysis for CN, Erdos and Power Law graphs to see the reason why HBC works for CN graphs, but not for the others. We concluded that the size of the bi-connected components (BCCs) is the main factor affecting the performance of HBC. For Funk Feuer Wien – a WCN in Vienna, the experiment shows that even though the biggest bi-connected components found contains almost half of nodes in the networks (46% of total nodes), HBC can still run 3 times faster than BBC.

So far, we have shown that HBC is an algorithm to choose when we need to calculate betweenness centrality of the Wireless Community Networks real topologies. There are ways that we think can improve the performance further, which may be implemented in the future. One property of WCN is that the topology evolves over time since any nodes can leave and join the network. However, in general, the new incoming or recently-left nodes are those in peripheral of the network [9]. We can thus keep track of the betweenness centrality in

each bi-connected component, and we would only calculate the betweenness centrality again when there are modification in that BCC.

For this thesis, both algorithms were run on the set of predefined graph. In the future, we can extend the functionality of the program, so that it can extract the real-time topology of the WCN of interest, and perform all the necessary calculation. Also for the router in the testing environment, its sole task was to run the experiment. The router was not busy with forwarding packets or finding routes for packets as it has to do in the reality. It will be interesting to see how the HBC work with the constantly evolving topology, and with the busy router.

Appendix A

Raw Data from the experiment

This chapter shows the raw analysis from the experiment.

Number of nodes in a graph	CN	PL	ER
100	87.5	1.1	2.4
200	181.9	1.0	3.4
300	273.0	1.1	5.5
400	366.1	1.1	6.5
500	477.0	1.0	8.9
600	572.3	1.1	9.8
700	666.8	1.0	12.4
800	763.7	1.0	12.6
900	859.2	1.0	15.9
1000	983.1	1.0	15.5

Table A.1: Average number of BCCs found for each category of synthetic graph

Number of nodes	CN	PL	ER
100	13.3	99.9	98.6
200	18.9	200.0	197.6
300	27.6	299.9	295.5
400	34.6	399.9	394.5
500	23.9	500.0	492.1
600	28.5	599.9	591.2
700	34.1	700.0	688.6
800	37.2	800.0	788.4
900	41.7	900.0	885.1
1000	17.7	1000.0	985.5

Table A.2: Average number of node in the largest found BCC for each category of synthetic graph

Wireless Community Network	BBC	HBC
Ninux	4.8	1.1
FFW	22.1	7
FFG	5.3	0.97

Table A.3: Running time of HBC and BBC on the router

Wireless Community Network	BBC	HBC
Ninux	0.15	0.03
FFW	0.7	0.19
FFG	0.17	0.03

Table A.4: Running time of HBC and BBC on the computer

List of Figures

List of Tables

Bibliography

[1] A. Vázquez-Rodas and L. J. de la Cruz Llopis, "Topology control for wireless mesh networks based on centrality metrics," in *Proceedings of the 10th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks*, ser. PE-WASUN '13. New York, NY, USA: ACM, 2013, pp. 25–32. [Online]. Available: http://doi.acm.org/10.1145/2507248.2507257

[2] L. Maccari and R. Lo Cigno, "Betweenness estimation in olsr-based multi-hop networks for distributed filtering," *Journal of Computer and System Sciences*, vol. 80, no. 3, pp. 670 – 685, 2014, special Issue on Wireless Network Intrusion.

[3] U. Brandes, "A faster algorithm for betweenness centrality," *Journal of Mathematical Sociology*, vol. 25, pp. 163–177, 2001.

[4] R. Puzis, P. Zilberman, Y. Elovici, S. Dolev, and U. Brandes, "Heuristics for speeding up betweenness centrality computation," in *2012 International Conference on Privacy, Security, Risk and Trust, PASSAT 2012*, 2012, pp. 302–311.

[5] I. F. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: a survey," *Computer Networks*, vol. 47, no. 4, pp. 445 – 487, 2005.

[6] X. Jardin. A wireless network for "little lhasa". [Online]. Available: http://www.npr.org/2006/08/10/5631353/a-wireless-network-for-little-lhasa

[7] H. Suzuki, Y. Kaneko, K. Mase, S. Yamazaki, and H. Makino, "An ad hoc network in the sky, skymesh, for large-scale disaster recovery," in *Vehicular Technology Conference, 2006. VTC-2006 Fall. 2006 IEEE 64th*, Sept 2006, pp. 1–5.

[8] H. Ahmed, M. El-Darieby, Y. Morgan, and B. Abdulhai, "A wireless mesh network-based platform for its," in *Vehicular Technology Conference, 2008. VTC Spring 2008. IEEE*, May 2008, pp. 3047–3051.

[9] L. Maccari and R. Lo Cigno, "A week in the life of three large wireless community networks," *Ad Hoc Networks*, vol. 24, Part B, pp. 175 – 190, 2015, modeling and Performance Evaluation of Wireless Ad-Hoc Networks.

[10] S. Jain and D. P. Agrawal, "Wireless community networks," *Computer*, vol. 36, no. 8, pp. 90–92, Aug 2003.

[11] C. Pisa, "Improving service support in wireless community networks," Ph.D. dissertation, Università degli studi di Roma "Tor Vergata", 2013. [Online]. Available: http://netgroup.uniroma2.it/wp-content/uploads/2010/01/tesi_Claudio_Pisa_submitted.pdf

[12] D. Katsaros, N. Dimokas, and L. Tassiulas, "Social network analysis concepts in the design of wireless ad hoc network protocols," *Network, IEEE*, vol. 24, no. 6, pp. 23–29, November 2010.

[13] U. Brandes and T. Erlebach, *Network Analysis: Methodological Foundations (Lecture Notes in Computer Science)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005.

[14] A. Bavelas, "A mathematical model of Group Structure," *Human Organizations*, vol. 7, pp. 16–30, 1948.

[15] A. Shimbel, "Structural parameters of communication networks," *The bulletin of mathematical biophysics*, vol. 15, no. 4, pp. 501–507, 1953.

[16] L. C. Freeman, "A Set of Measures of Centrality Based on Betweenness," *Sociometry*, vol. 40, no. 1, pp. 35–41, Mar. 1977.

[17] J. M. Anthonisse, "The rush in a directed graph," Stichting Mathematisch Centrum, Amsterdam, Tech. Rep., Oct. 1971.

[18] L. C. Freeman, "Centrality in social networks conceptual clarification," *Social Networks*, vol. 1, no. 3, pp. 215 – 239, 1978.

[19] U. Brandes, "On variants of shortest-path betweenness centrality and their generic computation," *Social Networks*, vol. 30, no. 2, pp. 136 – 145, 2008.

[20] L. Baldesi and L. Maccari, "Nepa test: Network protocol and application testing toolchain for community networks," *WONS*, 2016.