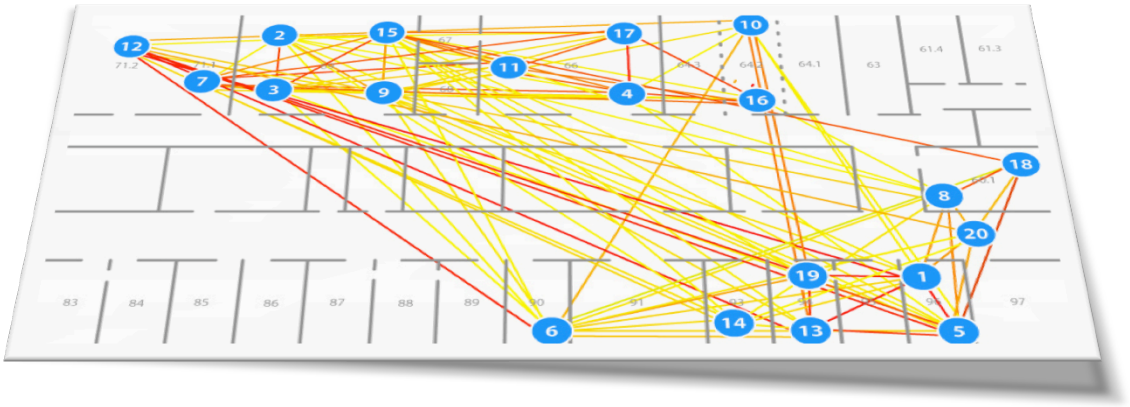
Broadcast Synchronization in

Wireless Ad Hoc Podcasting



Semester Project,

September 2008 – November 2008

Student: Mihai Calin

ETH – ITET Master Studies

Advisor: Dr. Franck Legendre

Supervisor: Prof. Dr. Bernhard Plattner

Task Formulation

Opportunistic content dissemination targets the broadcasting of content to a group of users using wireless ad hoc communications (e.g., PodNet). Previous works mainly focused on how the content dissemination process is affected by node collaboration. The performances [3,10] are significantly improved when nodes collaborate (i.e., store-and-forward).

Assuming nodes collaborate, the goal of this thesis is to (i) get insight on the mobility dynamics to better understand how content spreads and (ii) study the benefit of node gatherings on content dissemination. Since we target the broadcast dissemination of content, it is straightforward to use broadcast at the link layer instead of pair-wise connections as it has been done until now.

The work is split in the following tasks:

T1. Review work on opportunistic content dissemination (i.e., Podnet literature)

T2. Characterize mobility using real-world traces (i.e., CSG traces)

a- Statistical and structural properties (static topology)

b- Dynamic properties (clique evolution)

T3. Design dissemination strategies using previous findings

T4. Evaluate and compare the proposed strategys

Abstract

Opportunistic content dissemination targets the broadcasting of content to a group of users (e.g., PodNet). Previous works mainly focused on how the content dissemination process is affected by node collaboration. The performances are significantly improved when nodes collaborate (i.e., store-and-forward). Assuming nodes collaborate, our primary interest is the benefit of node gatherings on content dissemination. Since we target the broadcast dissemination of content, it is straightforward to use broadcast at the link layer instead of pair-wise connections as it has been done until now. Using real-world traces collected in a research office environment, we first characterize mobility to get more insights on how the dissemination process can be enhanced. This is done with a static and dynamic characterization of the topology. We study how we can take profit of gatherings (or cliques) to perform link layer broadcasts. From our findings, we design new broadcasting content dissemination strategies. We evaluate our solutions by replaying traces and comparing them to the original pair-wise (link layer unicast) approach. Results show that our approach doubles the overall capacity of the network when multiple contents are being spread concurrently. We then study different approaches for optimizing the broadcast content dissemination in terms of overall propagation delay and energy expenditure. Delaying dissemination until two nodes can profit from the broadcast lowers the total number of transmissions from 16.5 to 8.55, while keeping a good overall propagation delay.

Table of Contents

1. Introduction

1.1. Podcasting

A podcast refers to a new episode of data available for download over the internet by users that have subscribed for it [11]. The data is usually in the form of audio or video and can represent news bulletins, episodes of a TV-series, a personal blog entry etc. Although the data can be available for direct downloading, the podcast distinguishes from other means of transport by the fact that the user subscribes for a podcast and new episodes are downloaded automatically for him when they are available. The downloading is done by a program that is usually referred to as Podcatcher and that runs on the subscribers device; this program requests an RSS from a well-known address in the Internet. The obtained RSS document will point to, if existing, new episodes of the podcast.

RSS is a family of formats that contain a summarized text and metadata like authorship of a frequently changing work (e.g. blog entries, news etc) [12].

Podcasting was developed mainly for users of mobile devices that have no continuous connection to the Internet. The users would subscribe to a podcast and when they connect their mobile device to the Internet, this automatically checks and downloads new episodes of the podcast, if any available. There is still no mechanism for the podcast provider to push data toward the subscriber, only the latter can check for and request new episodes.

The main advantage of podcasting represents its asynchronous nature; compared to streaming, the content provider and the subscriber don't have to be online at the same time. The provider can upload a new episode to a web server and the user can download it at a later time. Moreover, he user can save the episode for watching at a later time than when it was downloaded.

1.2. PodNet

PodNet suggests a new way of content distribution to subscribers of podcasts. Compared to the traditional way in which each subscriber has to be connected to the Internet in order to check and download podcasts, podnet wirelessly implements an ad hoc podcasting between mobile devices that decouples the sharing from Internet based platforms. At gatherings of people, like in the bus on the way to work or at a late party, the devices can connect in an ad hoc manner and share stored information. If a subscriber didn't get the chance to connect to the Internet to download the latest podcasts, it can do so by querying mobile devices in the vicinity for the subscribed podcasts.

2. Problem Statement

2.1. Motivation

The current implementation of PodNet is disseminating content on a peer-to-peer base. This means two devices first establish a communication link to each other and then send the content over this link. Doing this in a wireless scenario raises more problems than in a wired one, because of the shared communication medium. The setup of a wireless link for the communication is a big time consumer and this represents a problem in the case of high mobility networks, where connections have to be switched quickly.

Being in a wireless environment could be also used in our favor. Because each transmission is actually a broadcast, we could use that in order to send the information not only to one partner, but to every station that is in the wireless range. Using this approach we could disseminate content into a network much faster, especially if the station population is very dense. Not only could the overall propagation delay fall but the energy used to propagate the data might be lower on the whole network and also the network capacity could increase.

Understanding the advantages and disadvantages of both dissemination strategies will help us find the right tradeoff for the communication parameters to make the system more robust and also to increase performance in terms of energy expenditure, overall propagation delay and network capacity.

The goal of this thesis is to analyze and quantify the benefits obtained from implementing broadcast data dissemination instead of the actual unicast (peer-to-peer) dissemination. The properties of networks will be analyzed and how they influence the benefits generated by broadcast dissemination. Also we plan to improve the standard broadcast dissemination strategy in order to obtain a good tradeoff between the energy expenditure and overall propagation delay.

2.2. Approach

For analyzing the network properties and study their influence on the broadcast content dissemination scenario, we will use data from real-world mobility traces that were collected at ETH Zurich as part of the Semester Project of Jörg Wagner [5]. Nineteen people were asked to wear mobile devices the whole day for a week. Each device would then record the presence of other devices in its wireless range; the connection logs of the devices are the starting point of the analysis since they allow us to read which stations, where and for how long were connected. This traces are replayed over and over again with different parameters giving us a better insight in data propagation in a real environment. For more details, refer to 4.1.1.The Trace Data.

The analysis sets off by looking at the nodes in the network and their properties. The average node degree will give a good insight about the size of groups that usually form in our network. We are interested in gatherings of nodes, since here broadcast is expected to be more effective than unicast. More about this in chapter 4.1.First Approach – Node Degree. Other questions we are interested in are: how many times we can talk about gathering of nodes and how long do they last on average? Are the groups of nodes mainly stable or do the members change continuously? How often do

nodes travel from one gathering to the other and thereby propagate the information to other groups of nodes? To find answers to these questions, we need to compute the connections between the nodes not only as node degrees but also as edges in a graph. This allows us to dynamically analyze the mobility of the nodes in the graph. Also when talking about connections between two nodes, the duration of a connection and its quality are essential. Threshold values for the quality will be analyzed and applied to filter the edges of the graph. More about this in chapter 4.2.Second Approach – Graphs.

To better understand the properties of the network formed by the traces, we will compute the cliques present in the network in every second. The identification of and grouping by cliques seems to simplify the broadcast approach since all nodes that belong to a clique can use broadcast to reliably transfer information to the clique members. Also characteristic problems of wireless communications are diminished through this approach, like the hidden node problem. As information can be propagated to all nodes of a clique in one hop, cliques can be regarded as entities (or as single nodes) and the problem of content dissemination reduces to propagating the information to the clique itself instead of each node.

We will analyze the cliques from both, a static and a dynamic point of view. From the static point of view, we will see what groups of nodes are mainly connected and can be seen as unities. From the dynamic point of view, we can extract information about the PDF of the cliques and use this information to propose improving dissemination strategies. Also the consistency of cliques in time will be analyzed showing, mainly giving an insight in the mobility character of the analyzed trace and how this influences the obtained simulation results. These results are presented in 5.Cluster-based Trace Analysis.

When the parameters of the network are known, direct simulations by replaying the traces for different dissemination strategies will give us results that we can numerically interpret to compare the unicast to the broadcast approach. In chapter 6.1.Propagation Time and chapter 6.2.Network Capacity, we will be able to analyze the propagation time and the network capacity of both unicast and broadcast dissemination strategies simulated on the same traces and hence to directly compare the two approaches.

After analyzing the benefits and faults of unicast and broadcast, dissemination strategies have to be developed that take advantage of the findings and show how to approach the node gatherings in order to have a positive effect on the propagation parameters (overall delay, energy expenditure). This is done in chapter 6.3.Strategies for Broadcast Content Dissemination. By limiting the maximum number of transmissions or delaying transmissions until some requirements are fulfilled, we will try to find the right balance between overall propagation delay and energy expenditure.

3. Related Work

The topic covered in this paper is building upon results already presented in other papers in the field of wireless ad hoc podcasting, while enhancing these results through a new approach – that of broadcast communication between mobile devices.

A starting point of the wireless ad hoc podcasting is [2] and presents protocols for substituting the client-server paradigm with a peer-to-peer paradigm where mobile nodes provide each other with contents. Also possibilities of caching and distribution strategies among the nodes participating in the communication are presented and analyzed.

The paper [1] is a degree project that analyzes the principles of podcasting and creates a substituting system based on peer-to-peer communication, rather than on the client-server architecture, present in traditional podcasting scenarios. A device-independent C++ software carries out the peer-to-peer synchronization and message exchange between two neighboring devices. Also a series of tests were run providing future researches with data gathered in a real life research-office environment. The data is also used to analyze and simulate the broadcast data communication in this current paper.

The problem of broadcast in a network can be addressed in terms of cliques in a graph. Finding maximum cliques in a graph can be a time consuming task, because there is no better solution than backtracking. Paper [4] presents an interesting approach of a backtracking algorithm with an early cut of useless branches for computing maximum cliques in a graph.

Based on real-world mobility traces, [3] analyzes different dissemination strategies comparing cooperative versus non-cooperative behavior of nodes for storing and/or forwarding content. In particular, it evaluates how the performance is impacted by mobility. The main outcome is that node collaboration drastically increases the performances of content dissemination while the per-device overhead (or load) is very low and remains on average evenly distributed. [10] models the different dissemination strategies of [3] using analytical stochastic models (Markov chains) that agree with the empirical results obtained by replaying the traces.

4. Terminology

Some terms in the paper tend to be ambiguous; a short explanation of the most used concepts seems in place.

Definition 1 – Station: refers to a mobile phone, ipod, palm or other device with wireless networking capabilities, able to transmit, receive, store and play any information that is transmitted through podcasting. The term "node" is also used as representing a mobile device, but always in connection with a graph.

Definition 2 – Beacon: is the small data package used to create the real life traces. It is emitted by each mobile device twice per second and all neighboring devices that are in the wireless range receive it.

Definition 3 – Connection: is a wireless connection between two mobile devices, in which each device can "hear" the other one. When talking about graphs, it will be referred to as edge.

Definition 4 – Node: is physically the same as a station; is used in connection to graphs in order to keep the graphs terminology consistent.

Definition 5 – Clique: is a maximal complete sub graph (MCS) of a graph, meaning nodes that form a fully connected graph within another graph and has a maximal size.

Definition 6 – Edge: is also referred to as connection in the early stages of the analysis; represents a possible two-way communication between two nodes of a graph.

Definition 7 - Not infected: nodes that didn't yet receive the information that is currently broadcasted through the network

Definitioin 8 – Infected: nodes that have received the information that is currently broadcasted through the network

5. General Trace Analysis

5.1. First Approach – Node Degree

In the sequel of this work, our analysis relies on mobility traces that were captured by Jörg Wagner during his SA [5]. We deal in this chapter with the raw data of the beacons received from other stations. We cannot yet talk about edges and graphs because we don't have enough information about the interaction between the stations. So for now we only have stations with connections between them.

5.1.1. The Trace Data

In the Related Work chapter we have talked about live traces that are to be analyzed to provide information about broadcasting benefits. In order to understand how to decode this data, let's see how it came to it in the first place.

We use real-world mobility traces that we collected at ETH Zurich. These traces are then replayed in a simulator developed by Jörg Wagner. The setup used to collect these traces consists of 20 HP iPAQs probing their neighborhood every half a second using their integrated IEEE 802.11b chipset operated in ad hoc mode. Nineteen test users were asked to to carry the devices during five consecutive working days (Monday-Friday from 11am to 5pm). The test users were researchers, staff members, and students of a networking research lab, all working on the same floor having a size of 100 meters x 30 meters. The map in Figure 5.1 – Devices Location shows the users' desks. The test users were instructed to carry the iPAQs with them throughout the day. A majority of the test users were researchers and spent most of the time at their desks. The users became mobile mainly due to lunch and coffee breaks, for going to the rest room, picking up printouts in the hallway, or meeting each other for discussions. Only few test users occasionally left the building or the campus for a limited period.

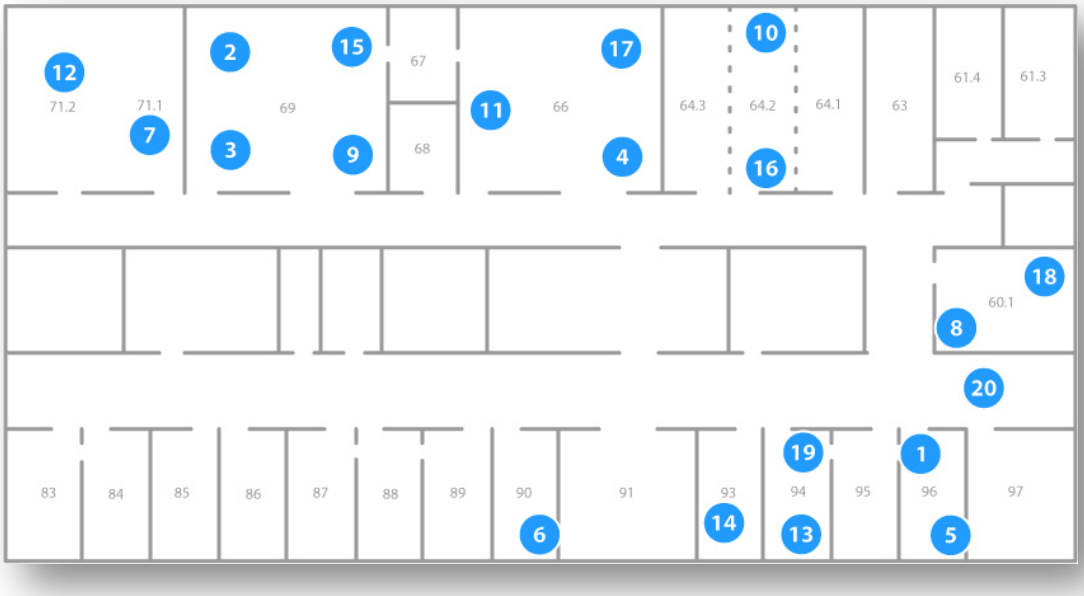


Figure 5.1 – Devices Location

In order to have reliable data, we first strip the communication done data until 10 AM and after 17 PM.

5.1.2. Node Degree analyzes

5.1.2.1. ***Instantaneous Node Degree***

To evaluate the stationary regime (if any) of the traces, the minimum/maximum/average node degree is plotted in Figure 5.2.

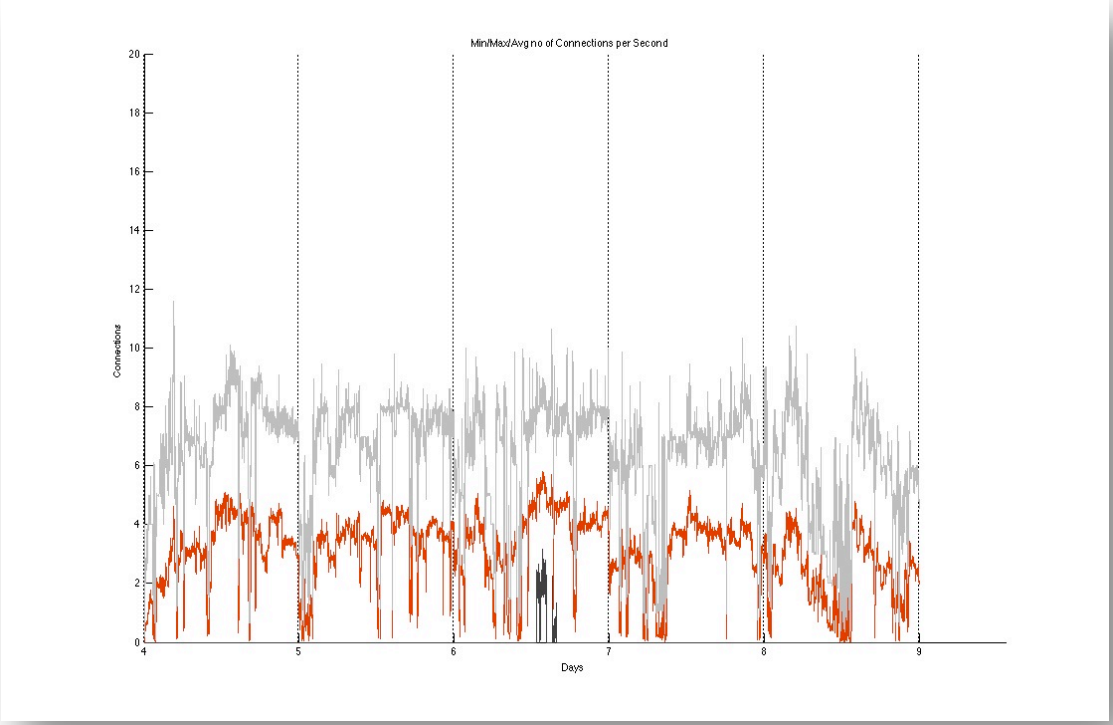


Figure 5.2 – Instantaneous Node Degree

It represents the number of beacons each station receives in a certain second during the five days in which the activity was tracked. Red is the average over all stations, light gray the maximum over all stations and dark gray the minimum over all stations.

The two following events are notable: on Friday, we can see that the maximum node degree drops heavily. This shows that no device “hears” beacons from more than two different stations.

On Wednesday noon, we can see that the minimum node degree rises above zero, meaning that all devices receive a minimum of three beacons from different stations. This part of the trace can be interpreted as a meeting of the people carrying the devices.

5.1.2.2. ***Averaged Node Degree***

Based on the instantaneous node degree presented in 4.1.2.1, the averaged node degree represents the average node degree of each second in a day. Hence Figure 5.3 shows us the same data plotted in Figure 5.2, but averaged/maximized also over the five days.

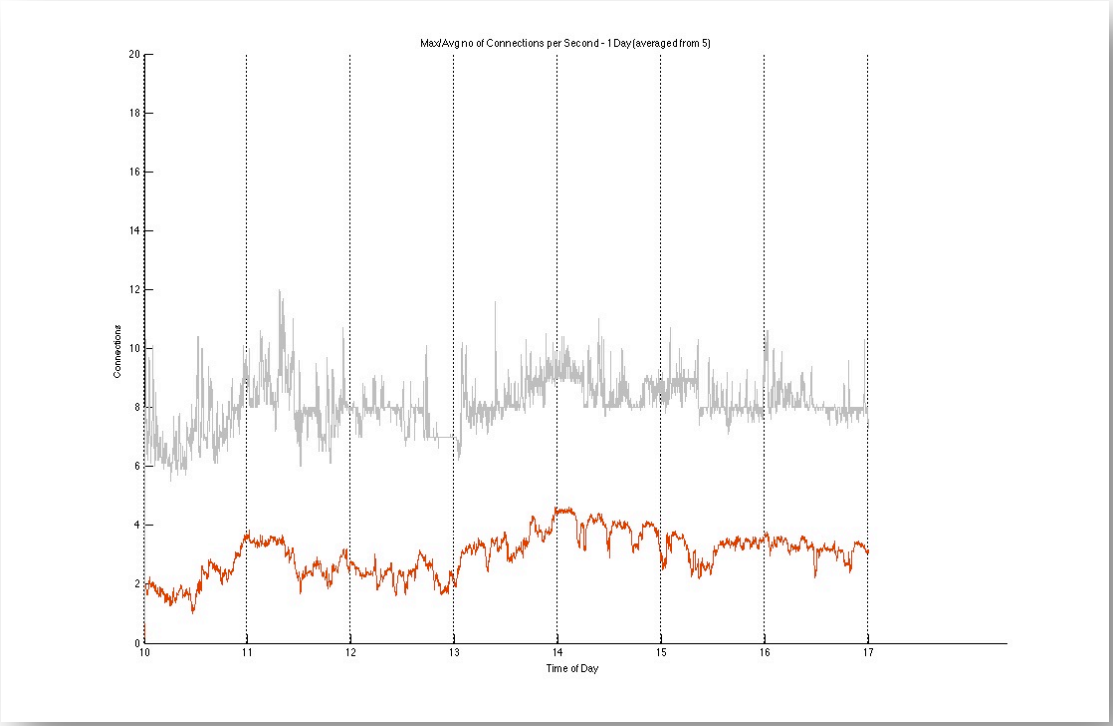


Figure 5.3 – Averaged Node Degree

The plot shows that the node degree is mainly stationary during the day.

For the maximum values (light gray) we can notice the formation of lines at the values of 7, 8, 9 etc connections. These appear due to the way this plot was obtained: it's the maximum value of a certain second, over all nodes and over all five days. The question would then be why we also have non-integer values? It's because the plots were softened with a window of 10 seconds to avoid scattering.

From the plot we can approximate the average node degree to about 3.5. We will see that this result will be somehow consistent across all measurements of the paper.

5.1.2.3. ***Node Degree PDF***

The PDF of the Node Degree is plotted in Figure 5.4. It shows the PDF in red, averaged over all devices, and also the max/min of the PDFs of all nodes.

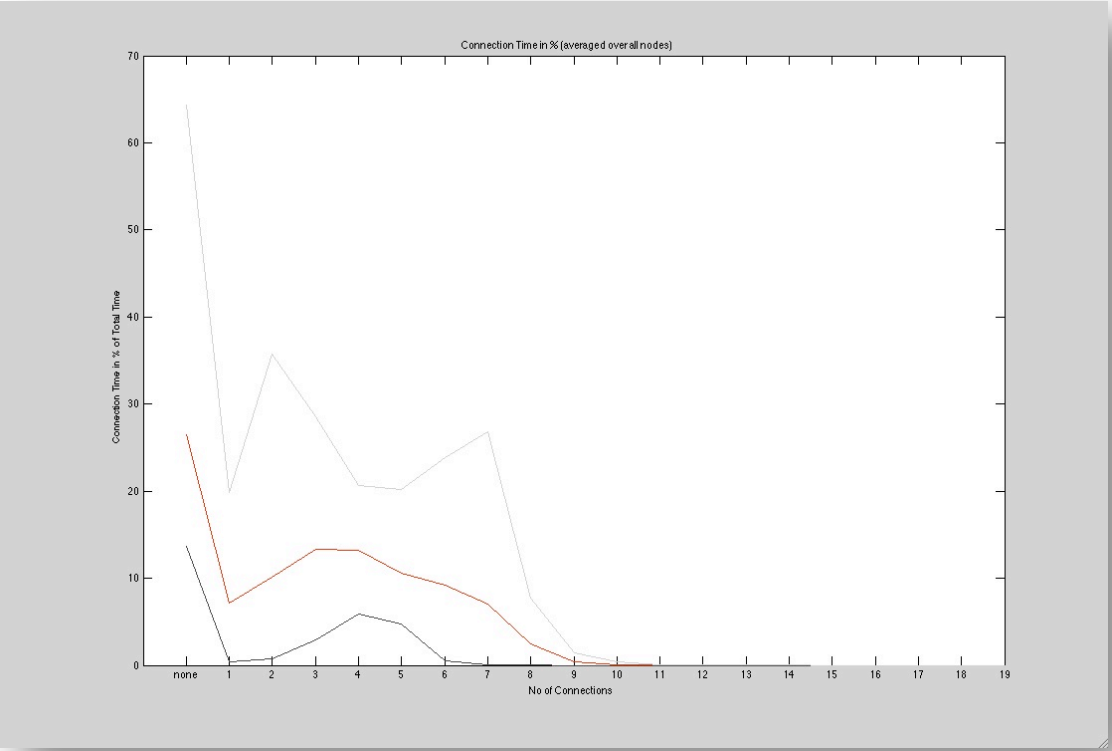


Figure 5.4 – Node Degree PDF

Regardless of the zero values, the PDF reaches a maximum value for node degrees of 3 and 4, proving our assumption from the previous plot.

Also important to note is that the minimum PDF over all devices for node degrees of 4 and 5 is no smaller than approx. 5 meaning that each node has a node degree of 4 or 5 at least 5% of the time.

An explanation for this behavior could be found if we refer to figure 1.1. We an see that because of the arrangement in the offices, usually groups of 4 to 5 stations are near to each other, having the possibility to "hear" each other. Having this explanation, we could also try to interpret it as the nodes not having much mobility. We will get back to the mobility later on.

5.1.3. Summary

The first section, entitled "First Approach", mainly deals with the edge degree of the Graph. It is a first approach, because in the next section we will start over and look at the connections between our nodes as edges of a graph and not as undirected connections between pair of nodes. We will continue to use the graphs approach in further analyzes because of the numerous algorithms already developed for analyzing graphs. Also Clusters will be introduced in a future section, that are based on graphs.

The results of the first approach will serve as comparison for the results we will obtain in the next sections. By plotting the instantaneous node degrees for both the whole week and also averaged over one day we have shown that the node degree of the graph is mainly stationary.

The most important finding of this section is the average node degree of 3.5. The bottleneck discussed in subsection 4.1.2.3 also shows that each node has a node degree of 4 or 5 at least 5% of the time. This result was interpreted as relying on the arrangement of the stations in offices, usually in groups of 4 or 5.

In the next section we will approach the analysis of the network from graph perspective. The connections between the nodes will become undirected edges and parameters for edge computation will be analyzed and optimized.

5.2. Second Approach – Graphs

5.2.1. Introduction

The first approach served to get a quick overview of the traces and the data we are dealing with. We have seen the node degree is 3.5 on average, which leads to the assumption that we can compute a connected graph, also with node degree of 3.5.

In the first approach we only analyzed the node degree by the number of received beacons, but never traced the beacons back to see if it's an undirected communication. We also missed a metric for the connection quality such that we can filter faulty connections through a threshold. All in all we can say that the values are rather statistical.

In the second approach, we plan to look at the stations as nodes of a connected graph and to compute the edges based on an undirected communication between the nodes. Hence we will check in the log of two stations to see if both received the beacon send by the other and if yes, we will assume the stations are connected. Also a quality second-based metric will be applied to the edges (explained next). Having the quality metric, we can easily apply a sliding quality window and a threshold for weak edges.

5.2.2. Sliding Quality Window and Quality Threshold

5.2.2.1. Link Quality Metric

The quality metric we want to apply is based on the number of beacons that are sent and received in a second. Remember that each node transmits two beacons per second to all nodes in its wireless range. So if we have an edge between two nodes, we calculate the quality of this edge with the following formula:

$$Link\_Quality = \frac{received\_beacons}{4 * t}$$

Basically, if in one second each of the two nodes we're looking at receives two beacons from the other node, the quality of the connection is 1.

We have no guarantee that this quality metric is accurate. One station receiving a number of beacons sent by another stations doesn't mean sending more data over the link will also work. But because our traces only report the number of sent and received beacons, we can only rely on this metric and hope it is different from the reality by a factor.

Based on this quality metric, we also implement a sliding quality window that averages the quality of an edge over several seconds. The ideal length of the window will be discussed next.

5.2.2.2. *Sliding Quality window*

The sliding quality window computes the quality of the connection in one second as an average of the qualities of several seconds. This way, short or mainly faulty connections are suppressed and only strong, long lasting connections are promoted.

If in an ideal environment, we want to consider connections between people walking by each other, we could make the sliding window 6 seconds and the threshold of 0.5. This means that the quality of an edge in a second is averaged from the quality of the same edge in 6 surrounding seconds and we only keep edges having a quality higher than 0.5.

5.2.2.3. *Impact of Quality Window Length*

Figure 5.5 shows the impact of different sliding quality windows – for 3, 5 and 7 seconds. It plots the number of edges currently in the graph for each second in the five days of measurement.

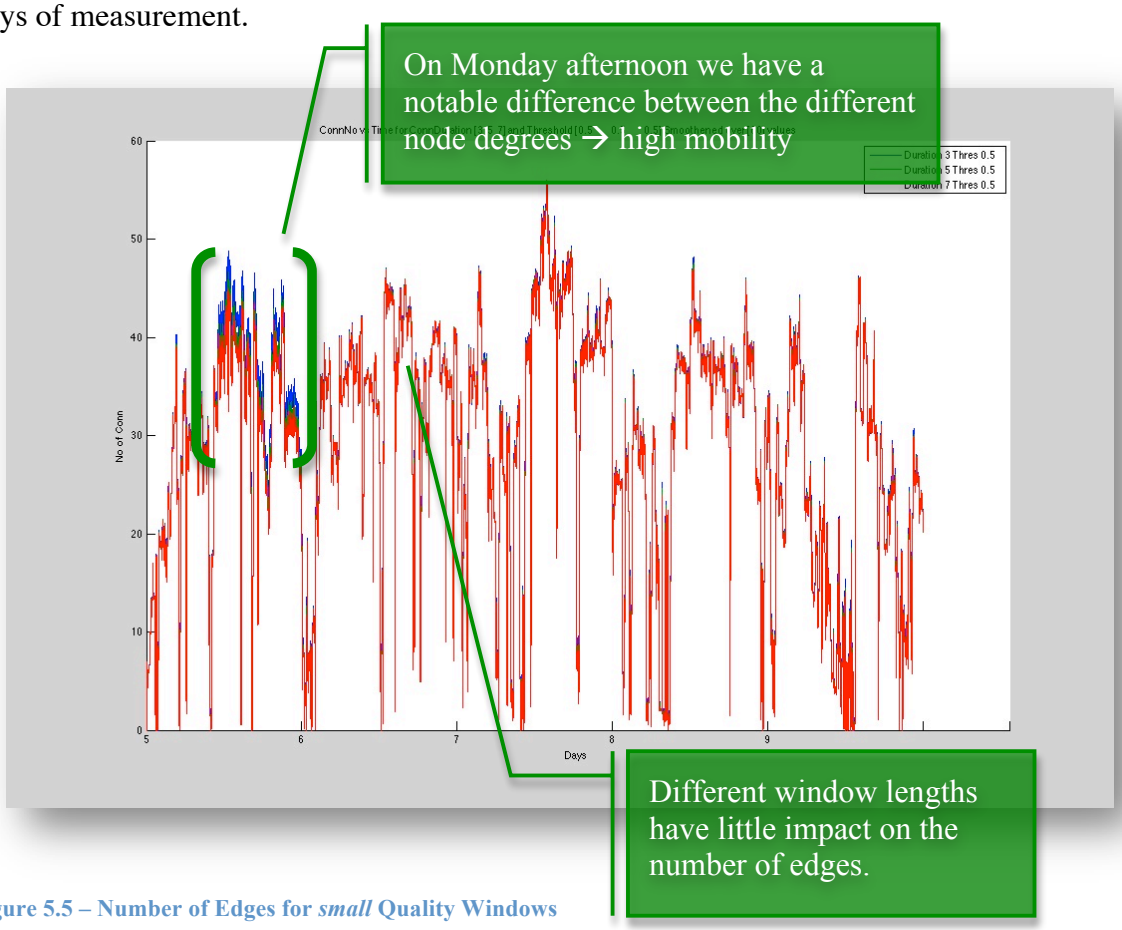


Figure 5.5 – Number of Edges for *small* Quality Windows

We can see that the impact of the three window durations is rather low; only on Monday afternoon we see a notable difference between durations 3 and 7 of approx. 5 edges.

The outcome of this test can be interpreted as following: our graph has not a very high mobility, hence windows of 3 and 7 seconds don't have a big impact on the number of edges. On Monday afternoon on the other hand, we can talk about a higher mobility, where the difference between durations of 3 and 7 is about 10% of the edges.

Running the simulation with higher values for the duration confirms that the graph has certain mobility, seen for values of eg. 25 seconds. Figure 5.6 shows the impact of

sliding window durations 4, 11, 18 and 25 as the number of edges in each second of Monday and Tuesday, the first two days of the experiment.

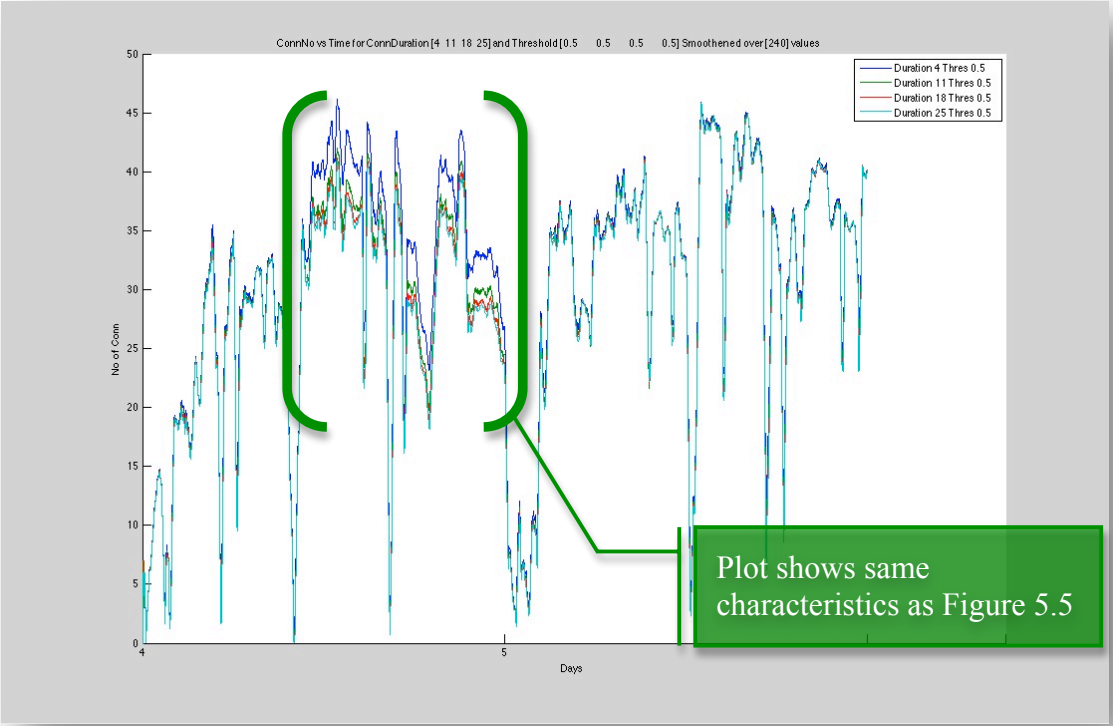


Figure 5.6 – Number of Edges for *large* Quality Windows

On Monday afternoon we again see the big edge difference among the values of the sliding window, whilst in the rest of the period we see a small difference. The sliding window length of 25 making a difference to the one of 4 proves that our graph is sensitive to different sliding window sizes, but the mobility is so low that connections last usually over 24 seconds.

The special case of Monday afternoon shows us that there is a logarithmic difference between the edges for values of 4, 11, 18 and 25. Hence, Monday represents a high mobility state of our setup.

Based on the obtained plots, choosing a sliding window size of 5 seconds seems appropriate for both the low and high mobility characteristics of our graph. Furthermore, Monday has both low and high mobility character, so it will be used in our further analyses rather than the data over the whole week.

5.2.2.4. Impact of Quality Threshold

The quality threshold represents a threshold for the averaged quality values in each second. Figure 5.7 shows the total number of edges in each second on Monday for different threshold values (0.5, 0.6, 0.7, 0.8, 0.9).

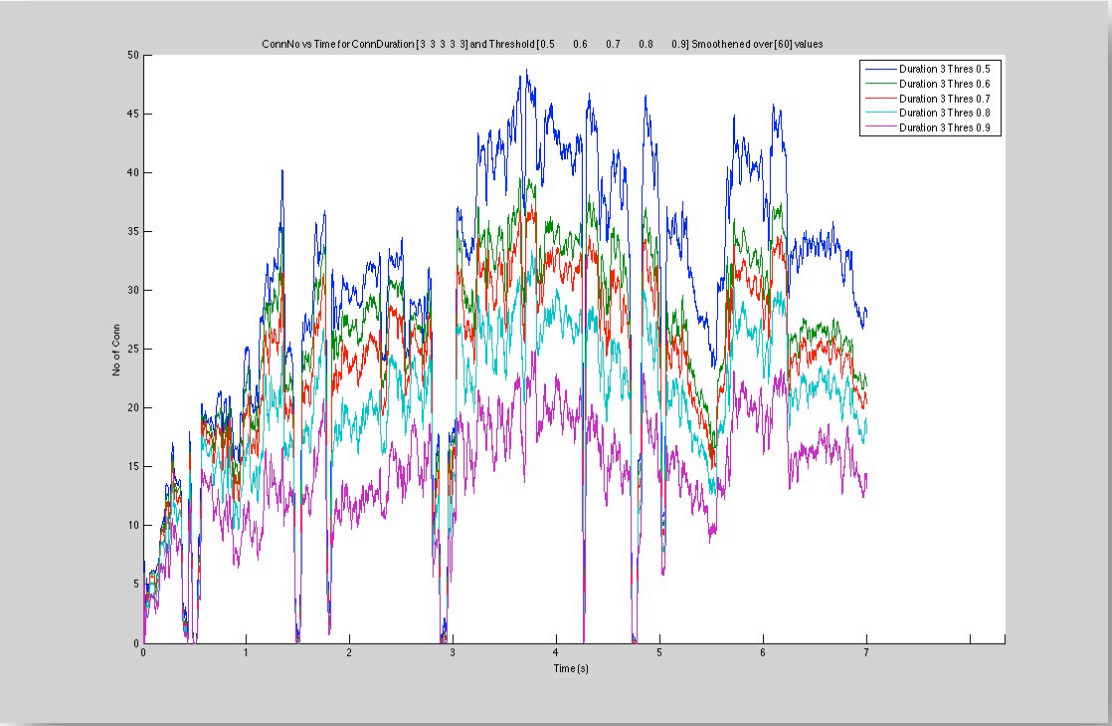


Figure 5.7 – Number of Edges for different Quality Thresholds

Obviously, the differences are more pronounced in the case of high mobility (Monday afternoon) than in the case of low mobility (Monday morning). But a general tendency is for the plots to center around the value of 0.67, from which the spread is logarithmically in both ways.

Based on this observation, we choose an ideal quality threshold of 0.67 for our next computations; the value is not high allowing a lot of edges to appear, but also over the average of 0.5.

5.2.2.5. ***Edge Quality***

Some interesting results can be obtained by plotting the connection between two specified nodes. The blue curves are the actual connection qualities between the two nodes, whilst the green curves are smoothed values such that a tendency can be seen.

After recalling the geographic position of the nodes (Figure 5.1) we can see that Figure 5.8 Edge between 11 and 17shows nodes (11 and 17) that were together in the same office. On Tuesday these nodes were not connected at all, while on the other days we have casual connections.

Figure 5.9 - Edge between 11 and 9 shows the contact between two nodes belonging to different offices, but separated by only two walls and a short distance.

Figure 5.10 - Edge between 11 and 19 shows the contact between two nodes very far apart in our layout. The very few connections are distributed over the 5 days and are mainly very short. Only on Wednesday we can talk about a connection long enough to have a data transfer under real conditions (green line also rises above 0).

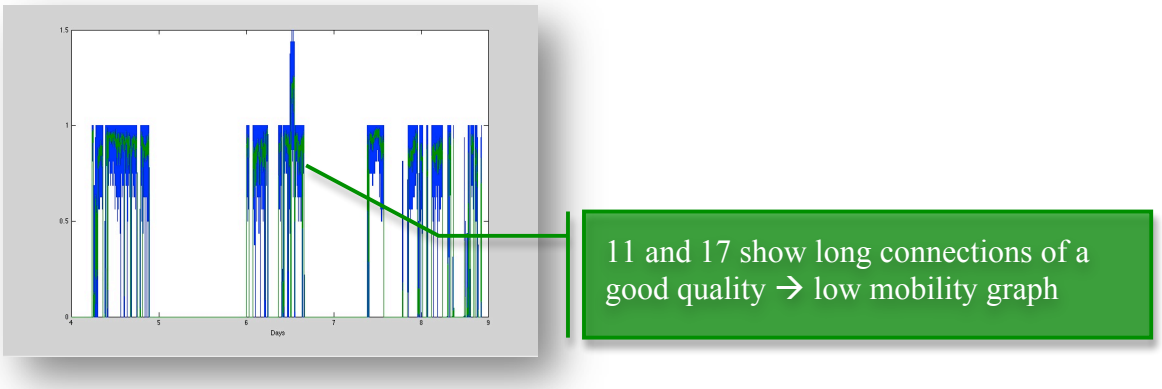


Figure 5.8 Edge between 11 and 17

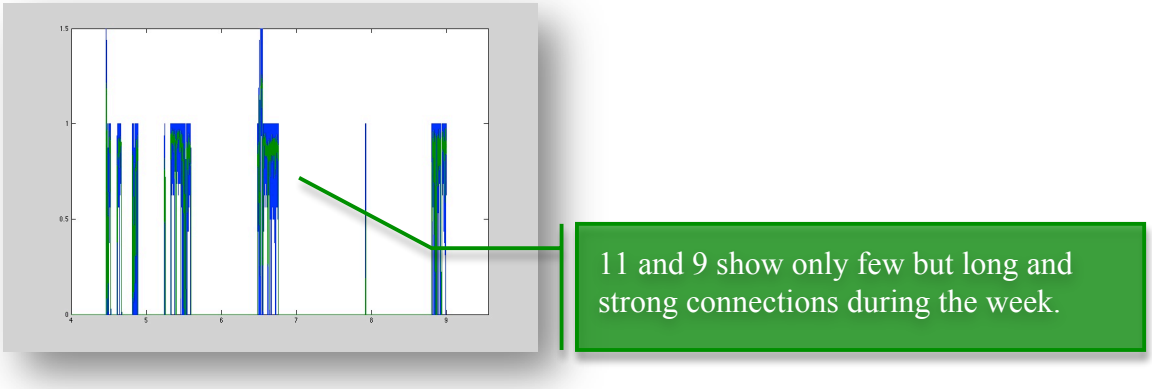


Figure 5.9 - Edge between 11 and 9

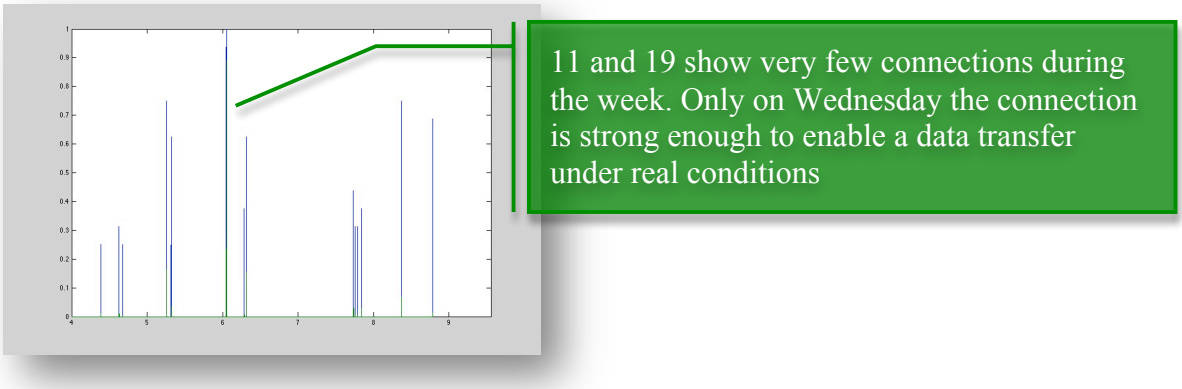


Figure 5.10 - Edge between 11 and 19

5.2.3. Summary

In this section we discussed the interpretation of the network as a graph. The connections between the nodes were computed as undirected edges of a graph. Also a quality metric was introduced for each edge of the graph such that faulty edges can be dropped from the analysis by applying a threshold.

The use of sliding windows helps eliminating very short connections and also helps identify the quality of alternating connections. A good choice for the sliding window

length was proved to be 5 seconds. Also connection quality threshold was considered when computing the edges and experiments show that a threshold of 0.67 is a good choice for accepting or dropping a possible edge.

The setup of this section also demonstrates that our traces have a rather low mobility, excepting the Monday afternoon when it rises.

In the next section we will use the obtained results to compute the static and dynamic edges of the graph. By static, a stationary regime is referred where all edges are taken and examined, regardless of time. In the dynamic approach, the edges of each second will be analyzed.

6. Cluster-based Trace Analysis

6.1. Static View

First we are going to analyze the graph from a static point of view, followed by the dynamic analysis. The static point of view neglects the time evolution of the graph, watching and analyzing all edges in the same time. It's useful for PDF and frequent cliques computation in the graph.

6.1.1. Edge Quality Graph

Having computed the edges of the graph we can now easily represent the edge quality (Figure 6.1).

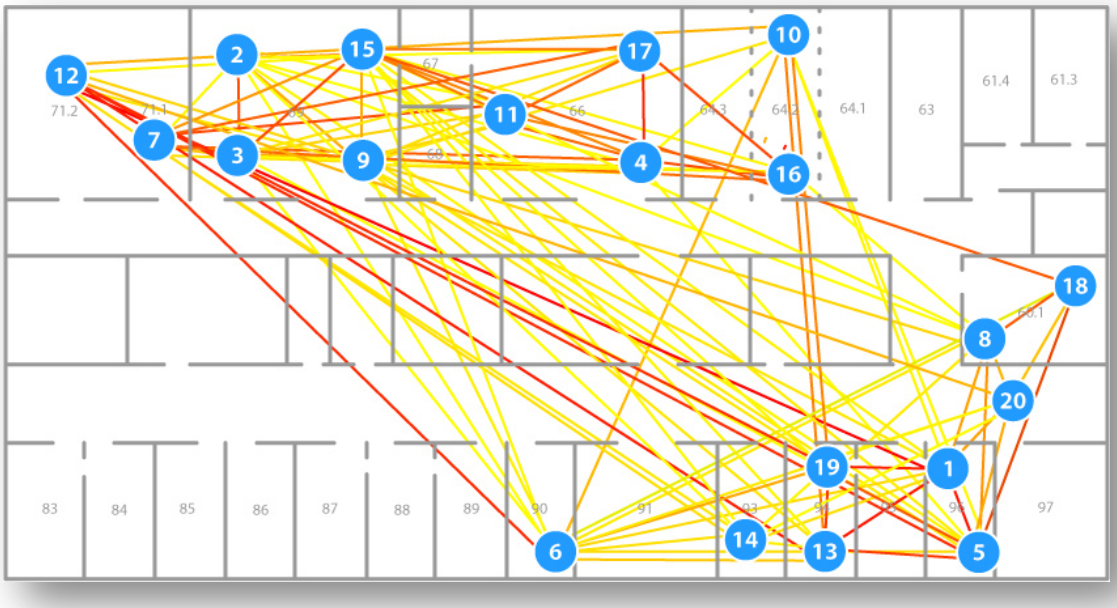


Figure 6.1 – PDF of all edges

Weak edges are marked with yellow and strong ones with red. Weak and strong only refers to how often this edge is present in our graph (meaning the number of seconds this edge is present to the total number of analyzed seconds) analyzed on the data from Monday. The graph can also be referred to as the edge PDF, because each edge is colorized by the probability that the specific edge is present at any chosen time instant.

Node 12 has an awkward behavior: it has very strong connections to the nodes in the lower right part of the graph, but not to its neighbors. Because this is the case for all of the edges starting in 12, we believe that node 12's position is plotted wrong and it was rather in the same office with 19 or 14.

Node 6 has no strong connection to any other node (but for 12), but has a very high node degree. It is connected to most nodes of our graph.

We can see that we have week edges between the lower right and the upper left of our graph – the main edges between these two clusters would be 10-19, 16-13 and 18-5, 18-15.

6.1.2. Small World Example

Having the PDF of all edges in our graph, we can go further and cut down the edges based on a strength threshold. A small-world network is a type of mathematical graph in which most nodes are not neighbors of one another, but most nodes can be reached from every other by a small number of hops or steps [6]. In Figure 6.2 we see the cliques computed with only those edges that are present in the graph more than half of the time.

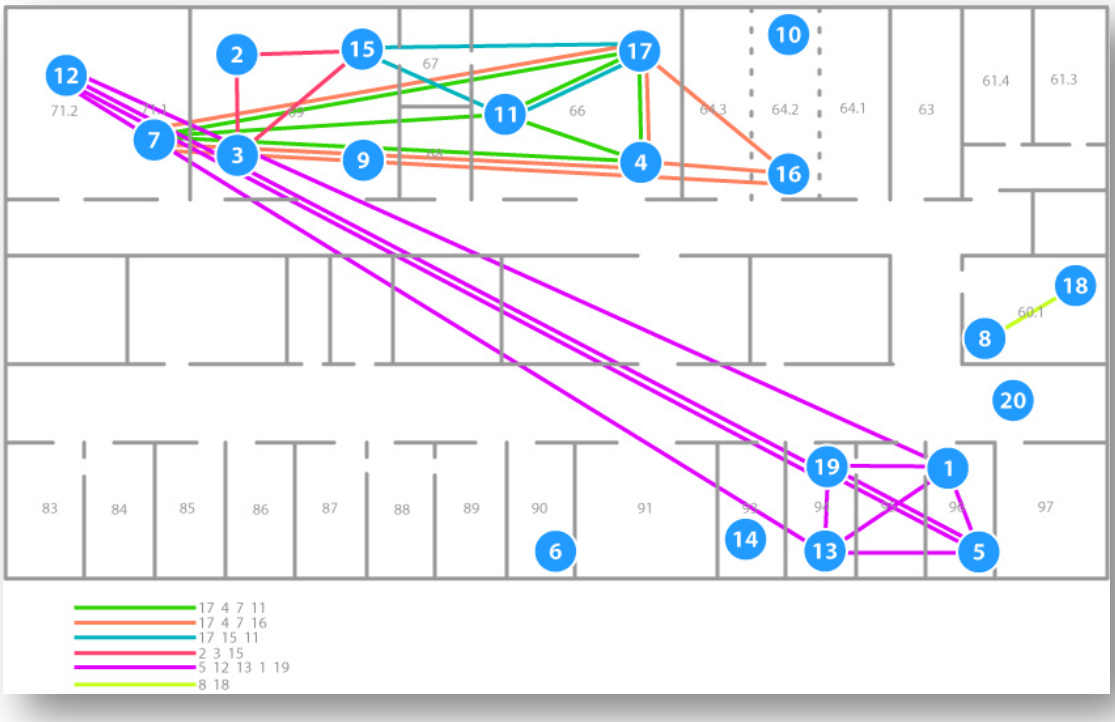


Figure 6.2 – Cliques Present more than half of the Time

Although the number of computed cliques is 6, we can see that the upper cliques (green, blue, orange and red) mainly have the same nodes as members; they differ mostly by one node. This is why we could say we have a single clique involving all nodes, but not fully connected (missing one or two edges).

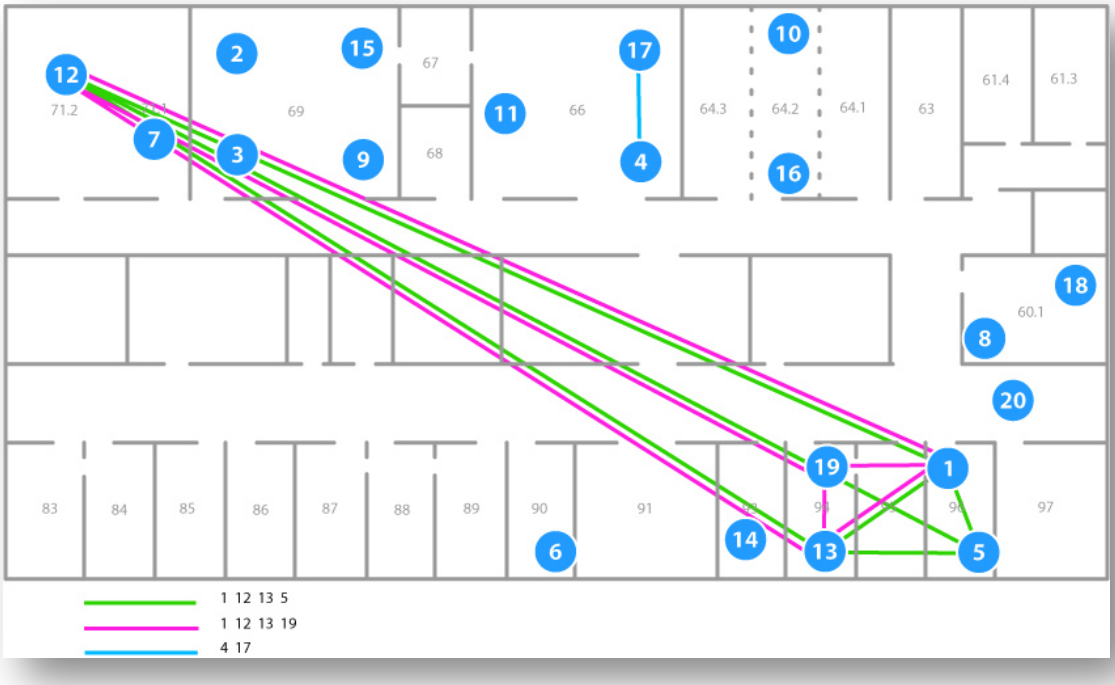


Figure 6.3 – Cliques present more than 0.75 of the Time

In this case, only the lower clique is present, but in the same situation as the upper clique in Figure 6.2 – meaning we again have overlapping cliques and we could talk about only one clique, not fully connected (missing one edge).

It's clear now that the nodes 12 actual position is rather near to nodes 19, 1 etc and not in the same office with 7, because it was connected to nodes 19, 1, 5, 13 more than 75% of the time.

6.1.3. Clique Size Distribution

We continued the analysis by plotting the clique size distribution – Figure 6.4.

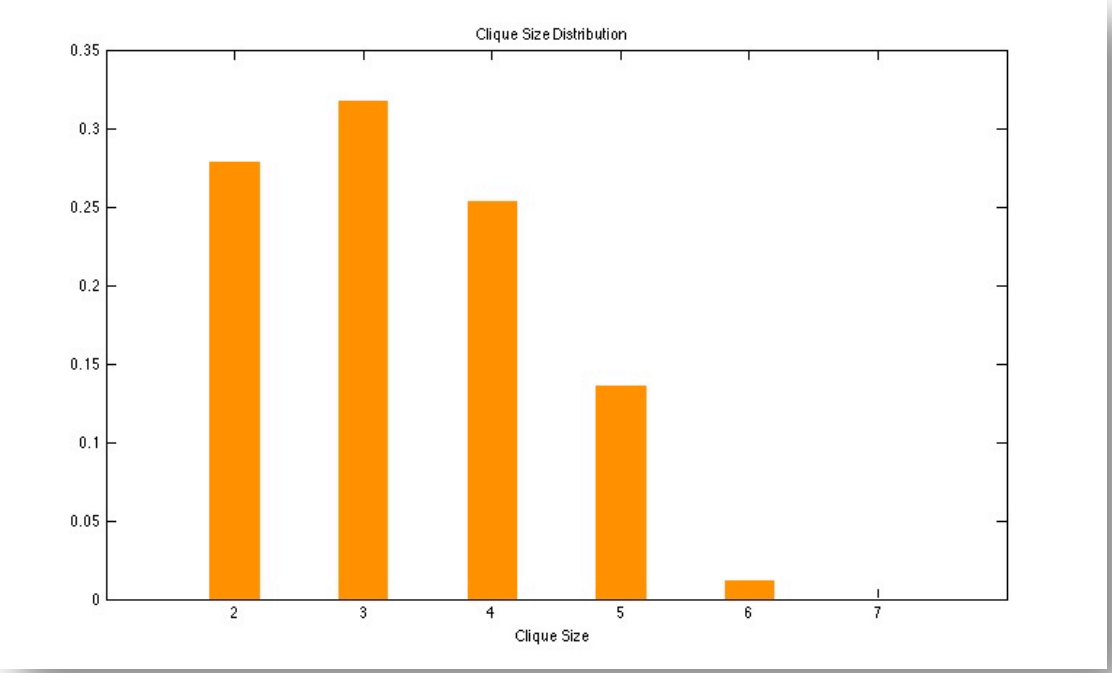


Figure 6.4 – Clique Size Distribution

Although the title suggests it is the PDF of the clique size, it's actually the PDF of the size of the clique that holds a node.

This example is going to illustrate it better: if we take a random node, with a probability of 0.325 it will be a member of a clique of size three. With probability of 0.28 a member of a clique of size two and so on.

If the real clique size distribution were to be represented, the number of cliques of size two would be greater than that of cliques of size three and so on. The proof can be expressed as an example: let's consider the case in which we have 12 nodes with equal probability of being in a 2-node or a 6-node clique. That means that half of the nodes will be in a 6-node clique, and the other half in two-node cliques. That gives us one 6-nodes clique and three 2-node cliques. So the probability of having 2-node cliques is three times greater than the one of having 6-node cliques.

Cliques of size one were not considered in this graph because of their trivial state – they have only one member so they don't have all the properties of a clique.

We can see here the concordance to our first results: we were talking about an average node degree of 3.5. Starting from that data, that was computing directed edges, we filtered data by computing the undirected edges. We filtered some more by applying the quality sliding window and the threshold value of 0.67 and we ended up with nodes being in cliques with two other nodes (on average). The obtained value of 3 nodes in a clique is a decent value considering the filtering done.

At this point we can already say that broadcast is going to help the communication in our graph, let's just evaluate to what extend. From here we will go further and analyze the graph from the dynamic point of view, starting with the clique stability.

6.2. Dynamic View

6.2.1. Clique Stability

Clique stability is basically the way a clique evolves in time. Figure 6.5 shows the stability of the clique formed by nodes 2, 3 and 15.

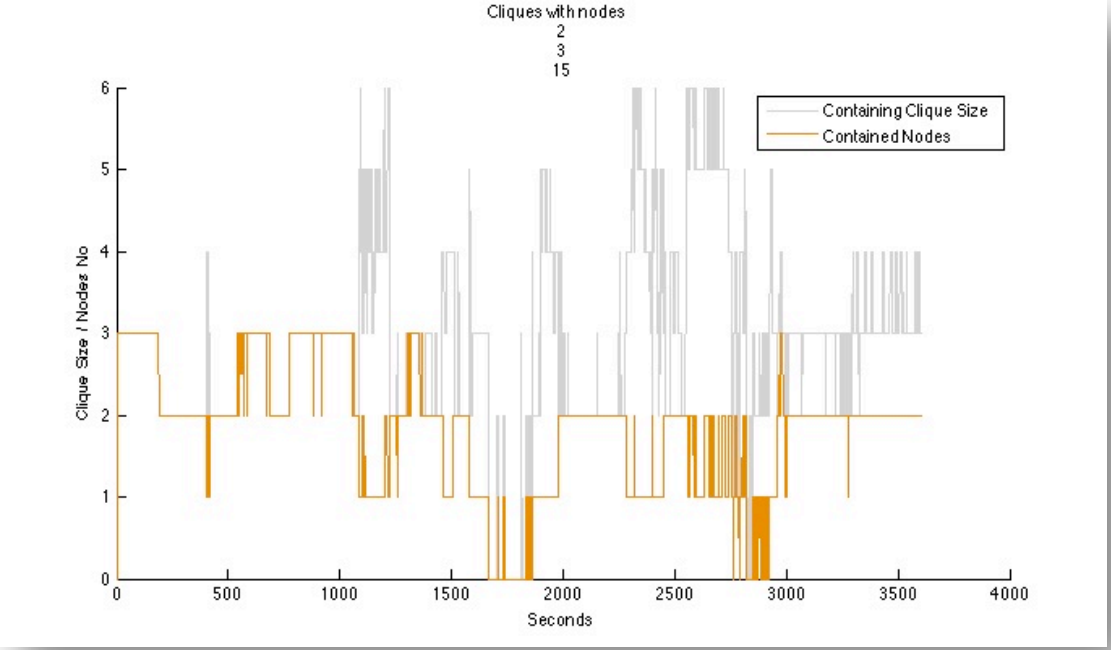


Figure 6.5 – Evolution of the Clique [2 3 15] over one Hour

The plot represents the evolution of this data for one hour from 11AM to 12PM on Monday. The analyzed nodes (in this case 2, 3, 15) can themselves form a clique of three nodes or can be contained inside a bigger clique. The ContainedNodes (orange line) shows the number of analyzed nodes that are currently in a clique. Sometimes all three are in a clique, but other times only two of them are in the same clique. In the latter case, we only look at the two that are in the same clique and state the ContainingClique Size. This ContainingClique Size (gray line) shows the size of the clique currently holding the analyzed nodes. This can be greater than three if the analyzed nodes together with other nodes form a clique.

We can see that the average stability time for the nodes in this clique is approx. 100 seconds. This means that once each 100 seconds one of the nodes leaves or rejoins the original clique. Assuming our time to transmit the data to be 1 second, 100 seconds characterizes the graph as having a low mobility.

6.3. Summary

In this section we have analyzed the static and the dynamic view of the graph defined by the edges between the nodes. In the static view, the edge PDF was elaborated giving us an insight in the communication of the graph. We have seen that the most communication happens between nodes in the same offices and interoffice communication is rare. Also we saw that there are usually two groups of nodes that generate fully or almost fully connected subgraphs (cliques): these are the upper ones and lower ones, as presented in Figure 6.1 – PDF of all edges.

The computed clique size distribution shows a theoretically very important measurement. It shows that with the highest probability, a node will appear in a clique of 3 nodes. This result is in concordance with the result presented in 4.1.2.3, where the average node degree was proved to be 3.5. After removing directed connections (one way communication channels), sporadic edges (by applying the sliding window) and low quality ones (by applying the threshold filter) and ending up having an probabilistic clique size of 3 is a positive feedback.

The evolution of a clique has also been discussed in this section and the results were presented in Figure 6.5 – Evolution of the Clique [2 3 15] over one Hour. This states that a clique is mostly stable for an average of 100 seconds. If we look at a maximal data transmission time of 1 second, we can again characterize the system as having a low mobility.

The clique size distribution and the evolution of a clique are important parameters in determining the theoretical improvements of broadcast content dissemination. The next section proves in a theoretical approach why we shouldn't expect a lower propagation delay of broadcast dissemination vs. unicast dissemination but rather an increase of the network capacity – with a factor depending on the average clique size (in our case the factor is two).

The next section will present the experimentally obtained propagation time and network capacity, which will match the theoretical ones.

7. Content Dissemination Strategies

7.1. Propagation Time

7.1.1. Insight on Dissemination Process

If we analyze the problem theoretically, a graph having a high or low mobility would profit differently from the broadcast data transmission. Let's see why.

If we consider a **high mobility** graph, we can think that in the first second a node sends data to all other nodes that can hear it. Because the mobility is high, in the next seconds, all nodes that received the data in the first second are sending it to other nodes they came across in this second, and these are ideally totally different nodes than the neighbors of the first second. In contrast to this broadcasting approach, consider the peer-to-peer connection. In the first second, the first node transmits data to only another node. In the next second two nodes transmit data to other two nodes. The overall propagation delay is much higher than in the case of broadcasting, especially if the average clique size is big.

Let's consider a **low mobility** graph. In the first second a node sends data through broadcast to all nodes that can hear him. In the second second all nodes have the information, but are still there and not transmitting it to anyone else. Only after a while the nodes that have the information come across other nodes and transfer the information to those. The propagation time is slower than in the previous example, and taken to the extreme case it's as fast as the case without broadcasting. In the peer-to-peer case, a clique stays connected very long, so there's enough time to transmit the data to all nodes in a peer-to-peer manner.

From these theoretical considerations we can draw the conclusion that broadcasting lowers the overall propagation delay only if we are in a high mobility graph. Otherwise there is not much of a difference to the peer-to-peer scenario. Since we characterized our graph as having a low mobility, we will see if the theoretical results mirror in the simulation results.

7.1.2. Unicast vs. Broadcast Dissemination

In the case of peer-to-peer (unicast) communication, one node is assumed to have the information that will be spread throughout the network. The information quantity is such that it is fully transmittable in a frame of one second. In the peer-to-peer strategy each node tries to transmit the data to a neighboring node that doesn't already have it. This generates a somehow cascade flow of the data in the network.

In the case of broadcast communication, again a node is assumed to have the information at start. In each second, each node that has the information picks a random clique he's a member of and that contain nodes that didn't yet receive the data. It then broadcasts the data to all the nodes belonging to this clique. A node can be a member of several cliques (e.g. Node 19 in Figure 6.3 – Cliques present more than 0.75 of the Time – he is both a member of the green and purple cliques) so one clique is randomly chosen.

7.1.3. Simulation Scenario

In this setup we consider two dissemination strategies: peer-to-peer (unicast) and broadcast. These strategies are simulated over the traces described in 4.1 for the time window Monday.

If we run the propagation time as a comparison between the broadcast and the peer-to-peer strategies, we obtain the results in Figure 7.1 and Figure 7.2. The plots represent the propagation time averaged over 20 runs with a different initial content provider in each run and only for the traces of Monday.

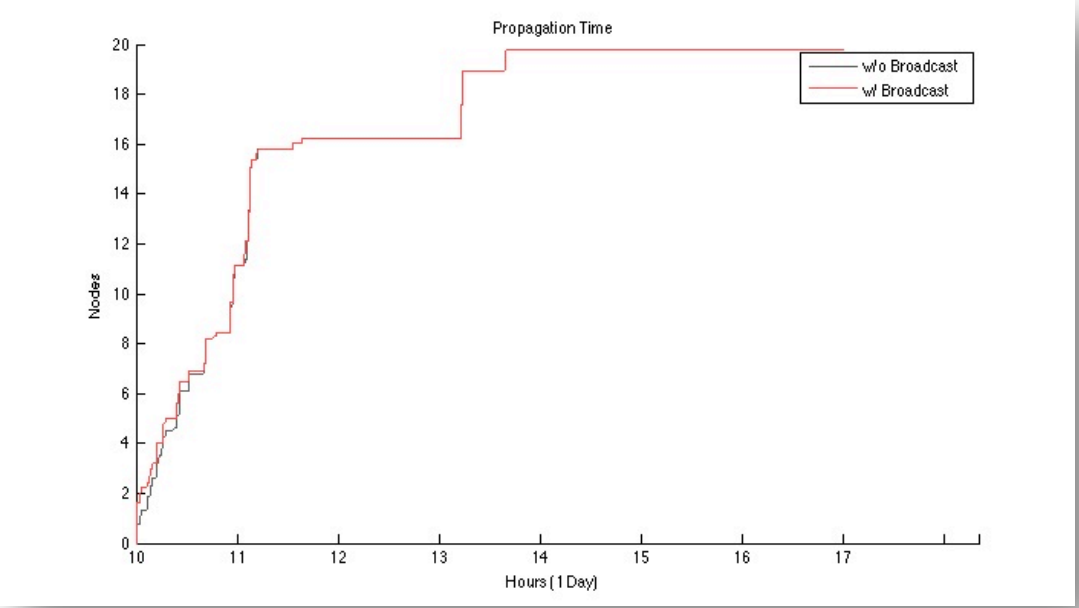


Figure 7.1 – Propagation Time for Broadcast and Peer-to-Peer Strategies

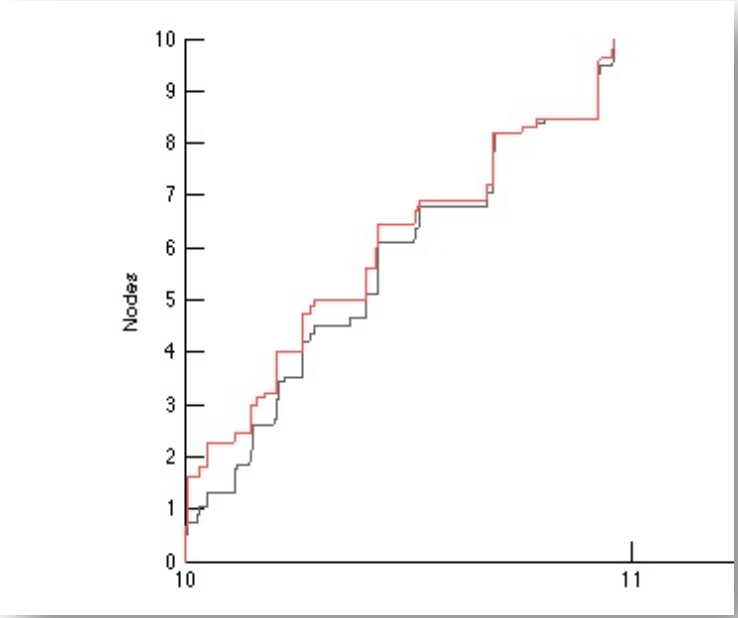


Figure 7.2 – Propagation Time for Broadcast and Peer-to-Peer Strategies – Detail

As we can see in Figure 7.2, that is basically just a zoomed version of Figure 7.1, the effect of broadcasting is rather modest. There is an increase in propagation time, but not to the extent that would justify the implementation.

We can see a relatively high propagation at start until approx 16 nodes are infected in the graph. Only later between 13.00 and 14.00 the propagation continues and all nodes get infected. An explanation for this behavior could be the fact that the people wearing the devices meet at lunch break and so help propagating the content to nodes barely reachable before.

At this point it has to be said that from an intuitive point of view this rather modest progress has an explanation in the mobility of the graph. In the case of high mobility, we could imagine to have much better results with broadcast rather than with peer-to-peer strategies.

7.2. Network Capacity

7.2.1. Theoretical Considerations

We have already seen that the average clique size is 3 nodes. Let's briefly think of the case in which data travels through the graph. In the case of cliques of size 3, one source node always transmits data to two other nodes in the same time instance, in the case of broadcasting, whilst in the case of peer-to-peer, a node transmits only to another node at a time hence half of the dissemination rate. These being the theoretical results, we can see that Figure 7.3 confirms the results through the simulation over our graph.

7.2.2. Simulation Scenario

Again we compare the results of a peer-to-peer and a broadcast dissemination strategies, this time from the propagation capacity point of view. For both strategies, a random node in the graph is considered to have enough buffered information, such

that it doesn't run out of content that waits to be sent out. Hence, we consider that there is always content to be disseminated.

In the case of peer-to-peer communication, each node transmits data to the neighboring nodes that don't already have the information. When all neighbors received the information, a node proceeds to sending the next chunk. Inside a clique the data transmission is done cascade like in sequential seconds.

In the case of broadcast communication, a node searches for the biggest clique he's a member of and that contain nodes that didn't jet receive the information. It sends the information to all nodes of the clique, even to those already having it, through broadcasting. When all cliques a node is member of received the information, the node proceeds to sending the next chunk of data.

Assuming n nodes in a clique and a channel capacity C, we can say the amount of data that is transferred is C in the unicast case and (n-1)C in the broadcast case. We can also represent these considerations by comparing the unicast case with a perfect time division multiplexing scheme such that the capacity is shared between the n-modulo(n,2) nodes with a possible modulo(n,2) idle node. For the broadcast case, the full capacity is used and all nodes receive C such that the total amount of data transferred is (n-1). Hence, the capacity ratio of broadcast vs. unicast is [(n-1)C]/C = (n-1). If we apply the numerical evaluation to our case, then we have to look at the average clique size, which is around three (Figure 6.4 – Clique Size Distribution), which gives us a doubling of the dissemination capacity. In terms of energy expenditure, these formulas also apply i.e., we have (n-1) transmissions with the broadcast case vs n transmissions in the unicast case. Given that the unicast case requires N-1 transmissions in total to spread the content to the all network, we could expect less required transmissions in the broadcast case. Further analysis of this issue is out of the scope of this thesis and is discussed in the Discussion Section.

In both cases, we measure the quantity of information that is transmitted in each second through the network

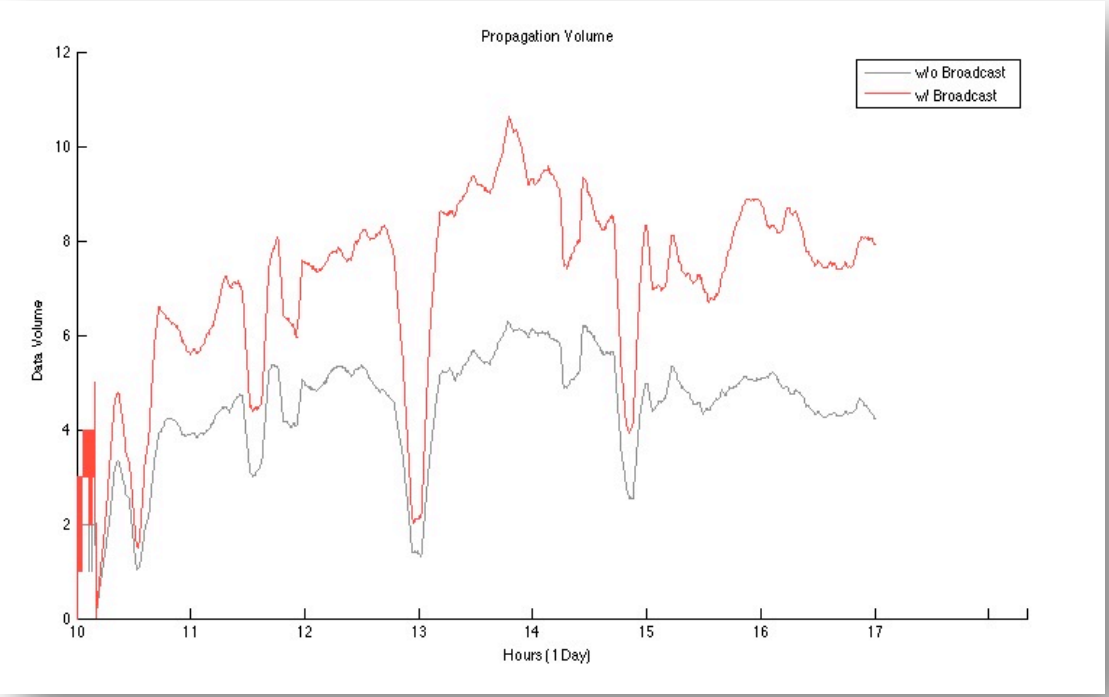


Figure 7.3 – Propagation Capacity for Broadcast and Peer-to-Peer Strategies

7.3. Strategies for Broadcast Content Dissemination

7.3.1. Introduction

Chapter 7 has shown that broadcast is a better choice than peer-to-peer communication especially from the network capacity (and energy expediture) point of view. Also, although it has not been pointed out, the nature of the wireless broadcasting of transmitting information to all communication partners using the same amount of energy encourages us to look closer at this possibility and to analyze further advantages that can be obtained through different dissemination strategies.

First we will show the simple way of broadcasting information in the network, based on random transmissions to random cliques. Next we are going to compute the best solution for propagating the content in our network; it's an unrealistic approach since global knowledge is needed to search for the best way to transmit information in each second, but it will give us a good upper bound for comparing the other strategies. Next we will use the idea of limiting the number of transmissions of each node. This should help us lower the number of average transmissions by, hopefully, keeping the overall propagation delay constant. The last approach, which will also yield the most interesting results, will be to postpone the transmission of data until a certain number of nodes are in the vicinity that would profit from the information.

To the concepts used in this chapter, some notes have to be made. The terms of infected and not infected will appear more often, because of their simple way of explaining the concept we are dealing with. If we see the information as a disease that travels from a node to another one, at any moment of time we can break the nodes into two groups: infected and not infected, simply meaning nodes that have already

received the information (infected) and others that didn't (not infected). As soon as a not infected node comes in the range of an infected one (and the latter transmits it's information to the first one) we can say that the first node becomes infected, meaning it now also has the information and can potentially transmit it to other not infected nodes.

7.3.2. Dissemination Strategies

7.3.2.1. ***Simple Dissimination***

This approach is the most easily to realize in terms of implementation at node-level. Each infected node transmits the information to one of the cliques he's a member of and that contain uninfected nodes. The choosing of the clique is done randomly but if an infected node is a member of at least one clique that contains uninfected nodes, it will get the chance to transmit the information. Recall that a node can be a member of more than one clique (see Figure 6.3 – Cliques present more than 0.75 of the Time: node 19 is both a member of the purple and the green cliques).

While broadcasting in a real network, infected stations concurrently transmit information to the uninfected nodes in their vicinity. In order to avoid data collisions, the infected nodes first synchronize to the nodes that they will infect, setting the communication parameters. On our traces we cannot simulate the concurrency because of the software limitations, so a sequential algorithm is used instead. In the case of the Simple Dissemination, the order in which the nodes transmit is chosen randomly while in the case of the Best Search Dissemination the order is computed through a greedy algorithm to maximize the infected nodes and minimize the number of transmissions.

For more information, see the Appendix.

7.3.2.2. ***Search Best Dissemination***

In the Search Best approach broadcasting the data in the network relies on global knowledge and can therefore be considered unrealistic for propagation scenarios. In each second, all nodes are analyzed and the best transmission order is chosen to assure a maximum spread of the information per number of broadcast transmissions.

In more clear terms, in each second the node is searched that can infect most other nodes with a broadcast transmission. These nodes are infected and the next best node is chosen to transmit. This goes on until no node can infect any other nodes. Also we assume that one node can not only infect nodes of one clique with one broadcast, but can infect all nodes of the cliques he's a member of with only one broadcast. This is a realistic approach as long as we consider longer synchronization times between the nodes of the different cliques (longer synchronization times = avoiding the hidden terminal problem).

For more information, see the Appendix.

7.3.2.3. ***Limited Transmissions Dissemination***

We have seen that in the case of peer-to-peer communication, nodes improve the tradeoff between overall propagation delay and number of transmissions by limiting the maximal number of transmissions per node. It is interesting to see if we have the same case in the broadcast scenario. This idea also forces an even distribution of the number of transmissions per node (global fairness).

The main goal is to lower the number of overall transmissions since more nodes are forced to transmit information and also these new nodes might have more optimal conditions for broadcasting. On the other side, a node could have the opportunity for an optimal broadcast but has already reached the maximum number of transmissions. In this case, the optimal state would be wasted.

For more information, see the Appendix.

7.3.2.4. ***Delayed Transmission Dissemination***

The Delayed Transmission Dissemination relies on the simple fact that broadcasting to a low number of nodes wastes energy. If we delay one nodes transmission until that node has at least a certain number of uninfected neighbors, we can propagate the information with a lower number of transmissions. On the downside, waiting for several nodes to join until a transmission is done, rises the overall propagation delay.

The tradeoff between overall propagation delay and number of transmissions is what we are interested in. Also this strategy could be used only at the beginning, when the most resources are used for suboptimal information dissemination. After a while, the network could switch to using another dissemination strategy, probably more suitable for the last phase of the propagation, such that an overall good overall propagation delay is reached.

7.3.3. Analysis

7.3.3.1. ***Simple vs Search Best Disseminations***

Figure 7.4 - Simple vs Best Search Overall propagation delay shows the overall propagation delay for the simple dissemination strategy compared to the overall propagation delay of the best search approach.

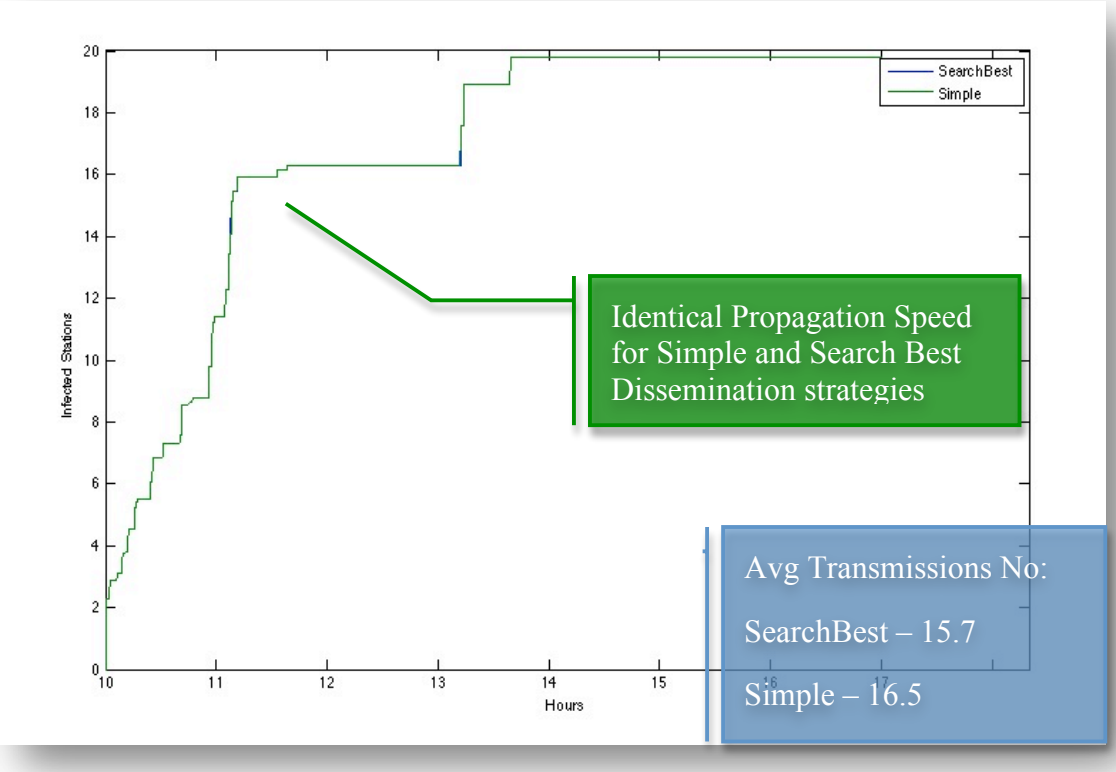


Figure 7.4 - Simple vs Best Search Overall propagation delay

Both strategies have an identical overall propagation delay. Although the Search Best strategy searches for the best propagation in each second, compared to the Simple strategy that just randomly chooses the propagation in each second, the speeds are identical. This is because of the low mobility of the graph, which allows for complete propagating each new setup of the nodes.

The average number of transmissions however yields a difference. While the Simple strategy achieves the propagation with an average number of transmissions of 16.5, the Search Best strategy does it with 15.7 Transmissions on average. This result shows that the Simple strategy works almost as good as the Search Best strategy (recall the latter is not possible in a realistic scenario because of the generally unavailable global information).

Compared to the peer-to-peer communication this would mean we have a big decrease in transmissions number, from 19 (recall that the network has n=20 nodes and N-1 unicast communications are required to disseminate content in the network given one node as a source) transmissions necessary in the case of peer-to-peer communication to infect all nodes, to only 16.5 in the case of the simplest broadcast strategy.

*7.3.3.2. **Limited Transmissions vs Search Best Disseminations***

Figure 7.5 - Overall propagation delay for Limited Transmissions (1,2,3) vs Search Best shows the overall propagation delay for the cases of limited transmissions. Gray plots the case of the Search Best strategy as a comparison.

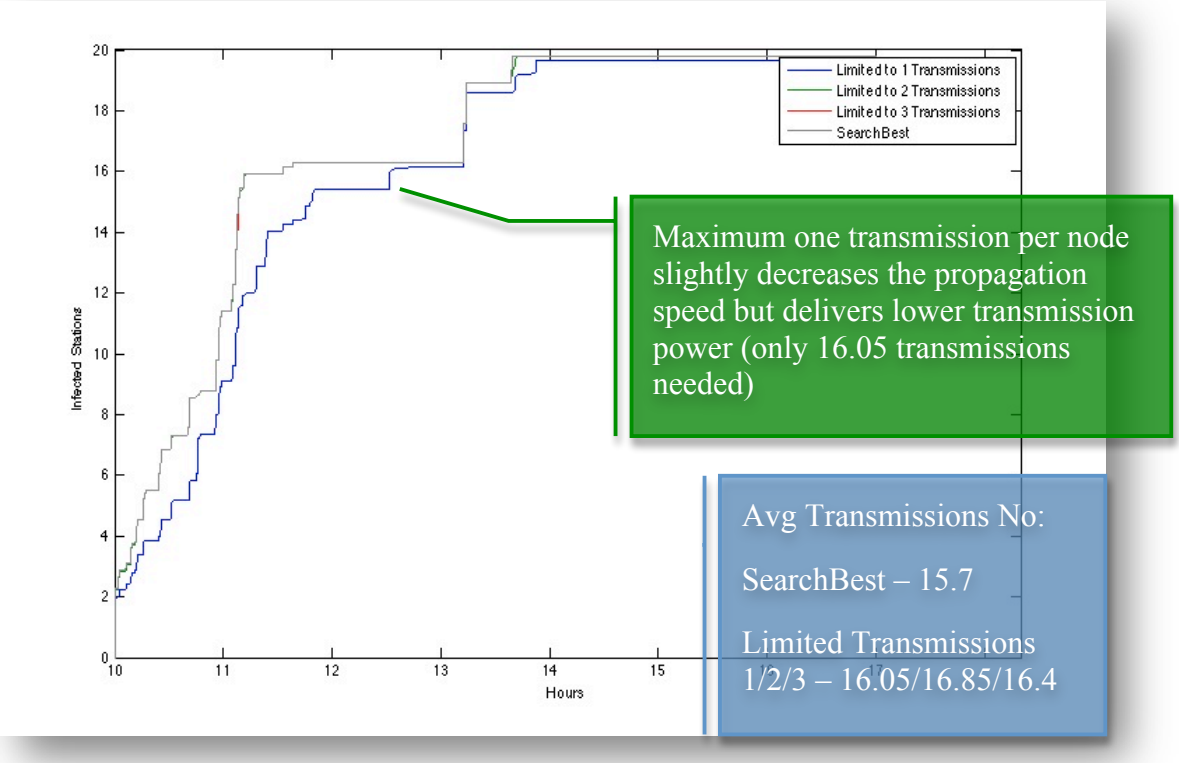


Figure 7.5 - Overall propagation delay for Limited Transmissions (1,2,3) vs Search Best

We can see that in the case of transmissions limited to 1, the overall propagation delay slightly drops under the optimum (search best) while for the cases with 2 and 3 transmissions per node, there is no difference between the overall propagation delays and the optimal overall propagation delay.

From the average number of transmissions point of view, we can see that the case of limitation to 1 transmission yields the best result out of the three. 16.05 transmissions on average to propagate the information are also better than in the Simple strategy.

The rise of the average number of transmissions to 16.85 and the fall again to 16.4 in the case of limitations to 2 and 3 transmissions shows it is suboptimal to increase the maximum allowed number of transmissions, but a high number can take advantage of some optimal broadcasting situations rising up.

In [10] we had a clear decrease of the performances with k=1. In this case the decrease seems lower and a possible explanation could be that in a clique it is not important which node sends infects all other nodes of the clique, as long as one does it.

Similar results have been shown for the unicast case both empirically (using the same traces) and analytically (using a CTMC) in [10].

*7.3.3.3. **Delayed Transmissions vs Search Best Disseminations***

Figure 7.6 - Delayed Transmissions vs Search Best Overall propagation delays shows the overall propagation delay in case of a Delayed strategy. This delays the broadcast of information until the number of uninfected neighbors that would profit from it reaches a certain number.

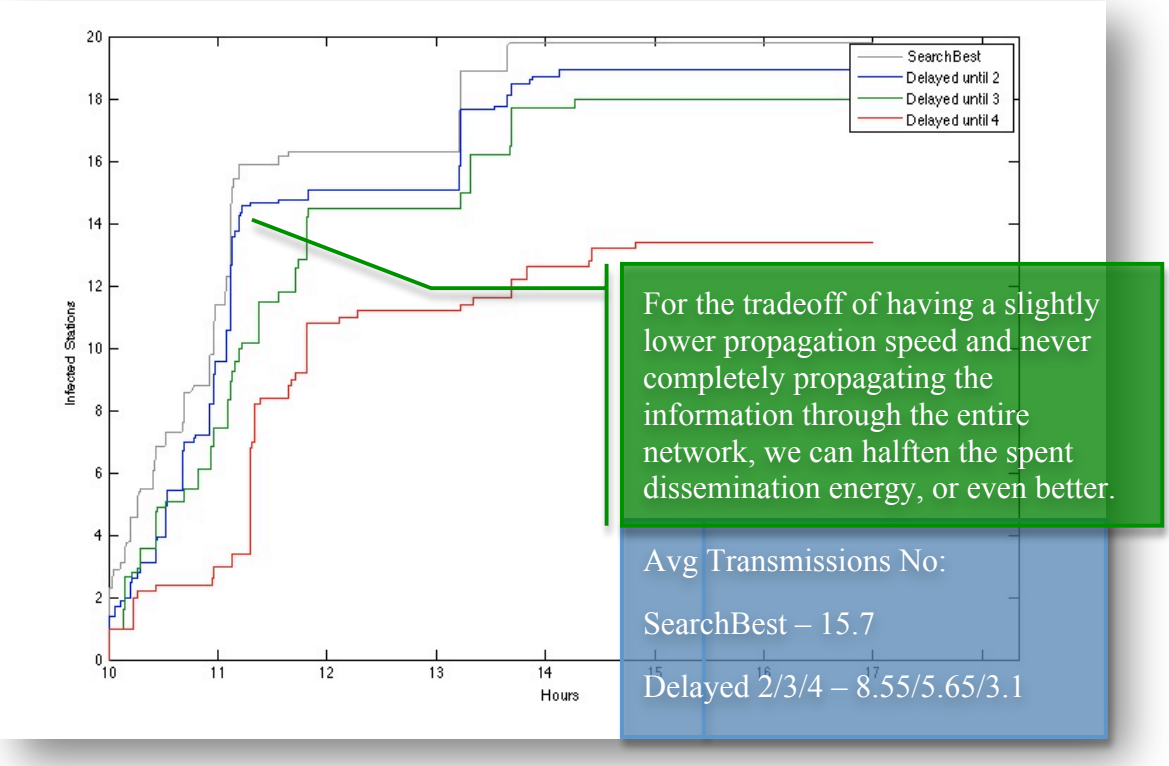


Figure 7.6 - Delayed Transmissions vs Search Best Overall propagation delays

As we can see from the graph, the overall propagation delay is highly influenced by the number of awaited uninfected neighbors. If this number is two and we only broadcast if we have at least two not infected neighbors, the overall propagation delay is only slightly higher than the one of the Search Best strategy. If the awaited uninfected nodes number rises to 4, the differences rise exponentially.

Another notable aspect is that while using this strategy, the information will not or only in special cases disseminate to the whole network. Hence, if this is a necessary condition for the dissemination, this strategy won't fulfill it. We can also consider changing the strategy if the number of infected nodes passes over a certain percentage of the total number. This however requires synchronization and extra communication between the nodes.

A very interesting aspect of this approach is the average number of transmissions achieved. For the case of delayed 2, we have an average number of transmissions of 8.5, which is almost half of the one in the Search Best, of course at comparable overall propagation delays. Not to forget this is a tradeoff for never having a complete dissemination in the network.

8. Conclusions

We set off analyzing real life data captured in a research-office environment during one week. Data was made off the beacons send by each device twice per second in a broadcasting manner and received by the ones that could hear it.

First analyses only looked at the node degree and the average node degree of 3.5 proved to be consistent along the further more complex simulations. The second approach introduced a measurement for the quality of each edge in the network, allowing us to drop faulty connections based on a threshold. Important insight information was provided through the static analyzes of the graph, where the most common cliques were identified and analyzed.

The last step was to make a direct comparison between the two implementations: with or without broadcasting.

The results have shown that for the traces we have analyzed, the broadcasting scenario didn't bring significant improvement to the overall propagation delay, but doubled the capacity of our network for concurrent spreading of content. Another main conclusion we arrived to, mainly sustained by the theoretical analyzes of the different propagation scenarios, is that the graphs mobility and its clique size distribution is mainly responsible for the benefits of broadcast compared to the normal peer-to-peer data propagation. In a graph with high mobility and high average clique size, the broadcast method is the desired one, while in a low-mobility, low average clique size graph the peer-to-peer can also be considered, based mainly on it's simplicity in terms of connection protocols.

In most of the scenarios however, broadcast communication is superior to the peer-to-peer data propagation, either in terms of propagation time or network capacity for concurrent information spreading, or both.

For further improving the broadcast scenario, different dissemination strategies were analyzed. For comparing them, we computed the best propagation time achievable on our traces together with the needed transmissions number in order to disseminate in the whole network.

By running simulations, we have proven a plain, random based, broadcasting similar in results as the computed best propagation. Also we have shown that applying the strategy of limiting the number of transmissions per node doesn't help much either he overall propagation delay or the number of transmissions.

The best results were obtained by a dissemination strategy that delays the broadcast until a certain number of the neighbors could profit from the broadcast. For the tradeoff of having a slightly higher overall propagation delay and never completely propagating the information through the entire network, we can lower the spent dissemination energy from 16.5 to 8.55 transmissions, or even lower if we consider longer overall propagation delays.

9. Discussion and Future Work

9.1. Clique Lifetime PDF

The clique lifetime PDF would be a very interesting insight in the characterization of the network. It could also be used to implement different dissemination strategies (as discussed in 8.1).

The cliques of each second were computed as part of this paper and the results show tables containing the cliques present in each second. To analyze the lifetime of each clique, one should take this clique and search for it in every second in order to count the appearances. To have the lifetime PDF of our graph, this has to be computed for all possible cliques. Having 20 nodes, combinable in number of 2, 3, 4… 20 nodes per clique, makes this task computationally difficult. Other approaches should be explored in order to compute the PDF easy enough to use it in ad-hoc networks.

9.1. Other Traces

Throughout the paper we have come to the conclusion that the used traces have a mainly stationary character. Some of the results (e.g. the small difference between the Best Search and Simple content dissemination strategies) were also traced back to this property of the network. Also our traces are limited to 20 nodes; sometimes this limit doesn't provide enough granularity to the results.

In order to prove some of the findings and possible come up with new ideas on how to improve dissemination strategies, other traces can also be used (eg. MIT Traces [9]) that use a larger number of nodes.

9.2. Other Dissemination Strategies

As we have seen, the dissemination strategy that delays the content dissemination until a certain number of neighbors can use the broadcasted information proved to be a very effective method of propagating the information because of it's overall propagation delay that was only slightly higher than that of the best case and because of it very low energy expenditure (half of the best case). The major downside of this approach is however that not all nodes in the network are guaranteed to receive the information. To avoid this, another strategy (possibly the Simple Dissemination Strategy) can be used after a certain threshold has been reached in time or in propagation percentage.

Also another approach would be to read a node's profile, that is to know it's own node distribution, and the delay that is applied to each node should depend on the node distribution of the respective node. There is no point for a node to wait until it has four nodes in his vicinity, if his PDF shows that he usually has only two.

A reliable broadcast scheme can use local recovery benefiting from other clique members to re-send unreceived packets because of wireless channel errors as done in [14]. Another solution is to use a network coding approach as explained in [13].

The time was not enough to fully address this approach but the first steps show promising results.

9.3. Network Capacity

In this paper we have talked about a capacity increase from the unicast to the broadcast case. In order to scientifically characterize the capacity of the network in both cases also theoretical limits have to be taken into account and compared ([7] and [8]).

9.4. Transmission of Bigger Files

In all simulations, only content that can be transmitted in one time unit has been analized. One can argue that this can be achieved by using UWB USB, yet using the current 802.11 technology, the transfer time might take longer. Hence a simulation that also considers longer packages has to be run and the results compared to the current results for all proposed dissemination strategies.

10.Bibliography

[1] Clemens Wacha – *Wireless Ad Hoc Podcasting with Handhelds* (Diplomarbeit)

[2] Vincent Lenders, Gunnar Karlsson, Martin May – *Wireless Ad Hoc Podcasting*

[3] Franck Legendre, Vincent Lenders, Martin May, Gunnar Karlsson – *Narrowcasting: An Empirical Performation Evaluation Study*

[4] Coen Bron, Joep Kerbosch – Algorithm 457 – *Finding all Cliques of an Undirected Graph (H)* – 1971 Technological University Eindhoven

[5] Jörg Wagner – *SA: Analysis of Dynamics in Mobile Ad-Hoc Networks* – September 2005

[6] Watts, Duncan J.; Strogatz, Steven H. (June 1998) - *Collective dynamics of 'small-world' networks* - Nature 393: 440–442. doi:10.1038/30918

[7] J. Li, C. Blake, D. S.J. De Couto, H.I. Lee, and R. Morris - *Capacity of ad hoc wireless networks* - Proceedings of ACM Mobicom 2001

[8] K. Jain, J. Padhye, V. Padmanabhan and Lili Qiu - *Impact of Interference on Multi-Hop Wireless Network Performance* - Proceedings of ACM Mobicom 2003

[9] MIT Reality Mining Traces, http://crawdad.cs.dartmouth.edu/

[10] Olafur Ragnar Helgason, Franck Legendre, Vincent Lenders, Martin May and Gunnar Karlsson - *Performance of Opportunistic Content Distribution* - Submitted to Infocom'09

[11] Wikipedia – *Podcasting* - http://en.wikipedia.org/wiki/Podcasting

[12] Wikipedia – *RSS* - http://en.wikipedia.org/wiki/Rss

[13] S. Deb, M. Effros, T. Ho, D. R. Karger, R. Koetter, D. S. Lun, M. Medard, and N. Ratnakar - *Network coding for wireless applications: A brief tutorial* - in Proc. International Workshop on Wireless Ad-hoc Networks, London, UK, May 2005.

[14] Majid Ghaderi, Don Towsley and Jim Kurose - *Reliability Gain of Network Coding in Lossy* - Wireless Networks, Infocom'08 mini-conference.

Appendix I

1. How to reobtain the Plots

1.1. Impact of Quality Window Length

Function "connStatusByDurationAndThreshold.m" plots the status of the connection for the different duration and threshold parameters.

Parameters:

Data – Has to be data in the form of a table having on the first column a uniqueSecond (the second in which the other columns are valid), on the second column a connID (the id of the connection between two stations in the second mentioned by the first column) and on the third column a beaconID (the id of the communication between the two nodes in this second). The last column is not needed for any further use. The second column contains the connectionID between the stations. From this, the stations have to be recoverable. Furthermore, a certain sort must be applied to the number of the stations in the connectionID, first stating the smaller station and then the greater one.(eg. Conn 5 – 11 = 0511; Conn 11-5 = 0511; Conn 7 – 6 = 0607; Conn 11 – 14 = 1114 etc). In each second, we have to have 4 connections between two stations in order to talk about a 100% connection. (eg. For stations 5 and 11, we need 2 connections from 5 to 11 and 2 connections from 11 to 5).

DURATION – Is a value or an array containing the duration of the sliding window that will be applied to the data. If it is an array, it has to have the same length as THRES

THRES – the threshold of the connections. All connections with a threshold smaller than this value will be filtered out.

Code:

```
%function returns no of connections in each second, respecting the duration
%and threshold.
%THRES has to be specified as a real value between [0,1];
%the first col in data (seconds) has to start with 0,1...

function countRow = connStatusByDurationAndThreshold(data,DURATION,THRES)
    %first create the quality col for the duration
    data=linksAndQuality(data,DURATION);
    %now filter by keeping everything which has a quality higher than the
    %threshold
    data=filterRows(data,4,'>',THRES);
    %now count the no of connections in each second
    countRow=zeros(1,length(data));
    for(f=1:length(data))
        countRow(1,(data(f,1)+1))=countRow(1,(data(f,1)+1))+1;
    end
    %crop the countRow vector to the no of seconds we have data for
    countRow=countRow(1:(max(data(:,1))+1));

end
```

1.2. Impact of Quality Threshold

Graph obtained through the same function as 1.1.

1.3. Edge Quality

Function "plotConnQuality.m" plots the quality of a connection between two nodes.

Parameters:

Data – refer to 1.1.

connID – the ID of the connection to be plotted. To find out more about the format of the ID, refer to 1.1.

connDuration – the duration of the sliding window. To find out more, refer to 1.1.

Code:

```
%function plots the connection quality in time of a connection between two
%certain devices (specified by the connID). Quality averaged over
%connDuration seconds

%the data argument is a table having the cols: second,conn,quality

function data = plotConnQuality(data,connID,connDuration)
    data=filterRows(data,2,'=',connID);
    data=linksAndQuality(data,connDuration);
    %create a seconds array representing all seconds of the transmitted
    %data
    seconds=zeros(length(data),1);
    for(f=1:length(data))
        seconds((data(f,1)+1),1)=data(f,4);
    end
    seconds = [seconds(:),smoothen(seconds(:),1,60)];
    plot(seconds);

end
```

1.4. Clique Size Distribution

Function "cliqueSizeDistribution.m" plots the probability that a random node appears in a clique of a certain size.

Parameters:

Cliques – a cell table having on the first column the second in which the cliques are present. On the next columns, if available each column contains an array with the elemens that form a clique. There are so many cells filled as many cliques there are in this second.

Nodes – number of nodes in the graph.

Code:

```
%input: cliques: a cell table having on the first column the second and on
%the next columns arrays representing a clique that is present in that
%second
%nodes: number of nodes in the graph

%output: an normalized array representing the pdf of the clique size
```

```
function result = cliqueSizeDistribution(cliques,nodes)
    result=zeros(1,nodes);
    for(f=1:length(cliques))
        %if this row has no cliques, skip it
        if(isempty(cliques{f,1}))
            continue;
        end
        g=2;
        while(~isempty(cliques{f,g}))
            result(length(cliques{f,g}))=result(length(cliques{f,g}))+1;
            g=g+1;
        end
    end
    %normalize
    resSum=sum(result);
    for(f=1:length(result))
        result(f)=result(f)/resSum;
    end
    result=result(2:7);
    plot(result);
end
```

1.5. Clique Stability

Function "plotCiqueEvolution.m" plots the evolution of a clique regarding the number of original elements that are part of the clique in time.

Parameters:

Data – same as "cliques" of 1.4

Clique – an array containing the elements of the clique that is to analyze.

Code:

```
% input:
% data: a table having: on the first col the current second. On the next
% cols the cliques present in that second. The table cells that contain no
% clique are empty
% clique: the clique to search for in each second

% output: function plots and returns two functions:
% ContainingCliqueSize: the size of the clique that contains most of the
% elements of the specified clique
% NodesContained: no of nodes from the provided clique that appear in the
% selected clique

function [ccs, nc] = plotCliqueEvolution(data,clique)
    ccs = zeros(1,length(data));
    nc = zeros(1,length(data));
    %the position in data of the maximal clique
    position = zeros(1,length(data));
    for(f=1:length(data))
        %if this row has no values, skip it
        if(isempty(data{f,1}))
            continue;
        end
        %in each clique of this second search for the nodes of the provided
        %clique and take the biggest that contains most of our nodes
        appearancesMax=0;
        lengthMax=0;
        posOfClique=0;
        g=2;
        while(~isempty(data{f,g}))
            appearances = 0;
            %for every node in the provided clique
```

```
            for(h=1:length(clique))
                %check if it appears in the selected (g) clique
                appearances = appearances + sum(data{f,g}==clique(h));
            end
            %now check if this is the clique that is mostly similar to the
            %provided one
            if(appearances>appearancesMax || ...
                    (appearances==appearancesMax &&
length(data{f,g})>lengthMax))
                appearancesMax=appearances;
                lengthMax=length(data{f,g});
                posOfClique=g;
            end
            g=g+1;
        end
        %save the results
        ccs(f)=lengthMax;
        nc(f)=appearancesMax;
        position(f)=posOfClique;
    end
    hold('off');
    hold('all');
    myPlot=plot(ccs);
    set(myPlot(1),'DisplayName','Containing Clique Size');
    myPlot=plot(nc);
    set(myPlot(1),'DisplayName','Contained Nodes');
    xlabel('Seconds');
    ylabel('Clique Size / Nodes No');
    legend('show');
    title({'Cliques with nodes ' clique});
end
```

1.6. Propagation Time

Function “propagationTime.m” and function “propagationTime2.m”, both plot the propagation time for the peer-to-peer strategy and the broadcast strategy, respectively.

Parameters:

Cliques – refer to 1.4 for details.

Code:

PropagationTime.m

```
function propagation = propagationTime(cliques)
%calculate the time the information arrives from one station to all
%others, in case of no broadcast, based on the cliques formed
propagation=zeros(20,length(cliques));
for(a=1:20)
    a
    infectedNode=a;
    nodes=zeros(2,20);
    nodes(1,:)=[1:20];
    nodes(2,infectedNode)=1;
    f=1;
    %while we still have data and not all nodes are infected
    while f<=length(cliques) && sum(nodes(2,:))<20
        %if it's an empty row, continue
        if(isempty(cliques{f,1}))
            f=f+1;
            continue;
        end
        %in each clique determine the infected and uninfected nodes. For
```

```
        %each infected one, infect an uninfected node. A node can only
        %infect one other node.
        %the array of nodes that already infected other nodes, marked with
        %1s
        jobDone=zeros(1,20);
        g=2;
        while(~isempty(cliques{f,g}))
            cliqueNodes=nodes(:,cliques{f,g});
            %if all nodes are already infected skip
            if(sum(cliqueNodes(2,:))==length(cliqueNodes(2,:)))
                g=g+1;
                continue;
            end
            %if no node is infected, skip
            if(sum(cliqueNodes(2,:))==0)
                g=g+1;
                continue;
            end
            %so we have infected and uninfected nodes in this clique,
            %infect as much as you can

            %remove nodes from cliqueNodes that already infected other
            %nodes
            h=1;
            while(h<length(cliqueNodes(1,:)))
                if(jobDone(cliqueNodes(1,h))==1)
                    cliqueNodes(:,h)=[];
                else
                    h=h+1;
                end
            end
            %how many nodes can we infect
            infect=min(sum(cliqueNodes(2,:)==1),sum(cliqueNodes(2,:)==0));
            infect2=infect;
            %infect them
            h=1;
            while h<length(cliqueNodes(1,:)) && infect2>0
                if(cliqueNodes(2,h)==0)
                    nodes(2,cliqueNodes(1,h))=1;
                    infect2=infect2-1;
                    %write it in the propagation
                    propagation(a,f)=sum(nodes(2,:));
                end
                h=h+1;
            end
            %save infectors not to infect other nodes
            h=1;
            while h<length(cliqueNodes(1,:)) && infect>0
                if(cliqueNodes(2,h)==1)
                    jobDone(cliqueNodes(1,h))=1;
                    infect=infect-1;
                end
                h=h+1;
            end

            g=g+1;
        end

        f=f+1;
    end

end

%arrange the data in propagation
for(f=1:20)
    value=propagation(f,1);
    for(g=1:length(propagation(f,:)))
        if(propagation(f,g)==0)
```

```
            propagation(f,g)=value;
        else
            value=propagation(f,g);
        end
    end
end
end
```

propagationTime2.m

```
function propagation = propagationTime2(cliques)
%calculate the time the information arrives from one station to all
%others, in case of broadcast, based on the cliques formed
propagation=zeros(20,length(cliques));
for(a=1:20)
    a
    infectedNode=a;
    nodes=zeros(2,20);
    nodes(1,:)=[1:20];
    nodes(2,infectedNode)=1;
    f=1;
    %while we still have data and not all nodes are infected
    while f<=length(cliques) && sum(nodes(2,:))<20
        %if it's an empty row, continue
        if(isempty(cliques{f,1}))
            f=f+1;
            continue;
        end
        %in each clique , if there's an infected node, instantaneously
        %infect all others
        nodesToBeInfected=[];
        g=2;
        while(~isempty(cliques{f,g}))
            cliqueNodes=nodes(:,cliques{f,g});
            %if there is an infected node, infect all others
            if(sum(cliqueNodes(2,:))>0)
                for(h=1:length(cliqueNodes(2,:)))
                    nodes(2,cliqueNodes(1,h))=1;
                    propagation(a,f)=sum(nodes(2,:));
                end
            end
            g=g+1;
        end


        f=f+1;
    end

end

%arrange the data in propagation
for(f=1:20)
    value=propagation(f,1);
    for(g=1:length(propagation(f,:)))
        if(propagation(f,g)==0)
            propagation(f,g)=value;
        else
            value=propagation(f,g);
        end
    end
end
end
```

1.7. Propagation Capacity

Function “propagationVolume.m” and function “propagationVolume2.m”, both plot the propagation capacity for the peer-to-peer strategy and the broadcast strategy, respectively.

Parameters:

Cliques – refer to 1.4 for details.

Code:

propagationVolume.m

```
%determines the no of seconds a station receives information from other
%stations in case of no broadcasting (1 to 1 communication).

%input: cliques: a cell having on the first column the seconds, and on the
%next succesive columns the cliques that are present in that second

%output: an array representing the no of seconds that that node received
%data

function volume = propagationVolume(cliques)
volume=zeros(1,length(cliques));

f=1;
while f<=length(cliques)
    %if it's an empty row, continue
    if(isempty(cliques{f,1}))
        f=f+1;
        continue;
    end
    %in each row, compute the number of communication seconds each node
    %has. This is the no of nodes in a clique/2. All nodes of a clique
    %that was analyzed are not analyzed in another clique of the same
    %second
    ignoredNodes=zeros(1,20);
    g=2;
    while(~isempty(cliques{f,g}))
        notIgnored=0;
        for(h=1:length(cliques{f,g}))
            if(ignoredNodes(cliques{f,g}(h))==0)
                notIgnored=notIgnored+1;
                ignoredNodes(cliques{f,g}(h))=1;
            end
        end
        volume(f)=volume(f)+floor(notIgnored/2);

        g=g+1;
    end

    f=f+1;
end

end
```

propagationVolume2.m

```
function volume = propagationVolume(cliques)
volume=zeros(1,length(cliques));
```

```
f=1;
while f<=length(cliques)
    %if it's an empty row, continue
    if(isempty(cliques{f,1}))
        f=f+1;
        continue;
    end
    %in each row, compute the number of communication seconds each node
    %has. This is the no of nodes in a clique - 1. All nodes of a clique
    %that was analyzed are not analyzed in another clique of the same
    %second
    ignoredNodes=zeros(1,20);
    g=2;
    while(~isempty(cliques{f,g}))
        notIgnored=0;
        for(h=1:length(cliques{f,g}))
            if(ignoredNodes(cliques{f,g}(h))==0)
                notIgnored=notIgnored+1;
                ignoredNodes(cliques{f,g}(h))=1;
            end
        end
        if(notIgnored>1)
            volume(f)=volume(f)+notIgnored-1;
        end

        g=g+1;
    end

    f=f+1;
end

end
```

1.8. Dissemination Strategies

Functions “broadcastSimple.m”, “broadcastSearchBest.m”, “broadcastDelayed.m”, and “broadcastLtdTx.m” provide in a similar way the results for the different dissemination strategies.

Parameters:

Cliques – same as in 1.4.

minUninfNodes – minimum number of infected nodes that have to be in the same clique to trigger an infecting broadcast

maxTx – maximum number of transmissions per node

Code:

broadcastSimple.m

```
function [infection,txNo] = broadcastSimple(cliques)
infection=zeros(20,length(cliques));
txNo=0;
for(f=1:20)
    txNoPerNode=zeros(1,20);
    f
    %array of infection status
    nodes=[1:20;zeros(1,20)];
    nodes(2,f)=1;
    %for each second in cliques
    g=1;
    while g <= length(cliques)
```

```
        %if there's no clique in this second, continue
        if(isempty(cliques{g,2}))
            g=g+1;
            continue;
        end
        %get the nodes that are infected
        infNodes=nodes;
        h=1;
        while h <= length(infNodes(1,:))
            if(infNodes(2,h)==0)
                infNodes(:,h)=[];
            else
                h=h+1;
            end
        end
        %randomize the order of the nodes
        infNodes=infNodes(:,randperm(length(infNodes(1,:))));
        %find out how many cliques we have in this second
        h=0;
        while(~isempty(cliques{g,2+h}))
            h=h+1;
        end
        noOfCliques=h;
        %for each randomly picked infected node, randomly find a clique
        %containing it and that also contains non-infected nodes.
        for h=1:length(infNodes(1,:))
            %create a vector to randomly access the cliques
            cliqueAccess=randperm(noOfCliques);
            for k=1:length(cliqueAccess)
                %is our selected node (infNodes(1,h)) part of this clique?
                %and
                %do we also have nodes that are not infected?
                if(sum(cliques{g,1+cliqueAccess(k)}==infNodes(1,h))>0 && ...
                        sum(nodes(2,cliques{g,1+cliqueAccess(k)})) < ...
                            length(cliques{g,1+cliqueAccess(k)}))
                        %infect the nodes in this clique
                        nodes(2,cliques{g,1+cliqueAccess(k)})=1;
                        txNo=txNo+1;

txNoPerNode(1,infNodes(1,h))=txNoPerNode(1,infNodes(1,h))+1;
                        break;
                end
            end
        end
        %all nodes possible to infect in this second were infected. Save
        %the new infected count
        infection(f,g)=sum(nodes(2,:));
        g=g+1;
    end
    txNoPerNode;
end

%arrange the data in infection
for(f=1:20)
    value=infection(f,1);
    for(g=1:length(infection(f,:)))
        if(infection(f,g)==0)
            infection(f,g)=value;
        else
            value=infection(f,g);
        end
    end
end

infection=sum(infection)./20;
txNo=txNo/20
```

broadcastSearchBest.m

```
function [infection,txNo] = broadcastSearchBest(cliques)
infection=zeros(20,length(cliques));
txNo=0;
for(f=1:20)
    f
    %array of infection status
    nodes=[1:20;zeros(1,20)];
    nodes(2,f)=1;
    %for each second in cliques
    g=1;
    while g <= length(cliques)
        %if there's no clique in this second, continue
        if(isempty(cliques{g,2}))
            g=g+1;
            continue;
        end
        %find out how many cliques we have in this second
        h=0;
        while(~isempty(cliques{g,2+h}))
            h=h+1;
        end
        noOfCliques=h;
        %get the nodes that are infected
        infNodes=nodes;
        h=1;
        while h <= length(infNodes(1,:))
            if(infNodes(2,h)==0)
                infNodes(:,h)=[];
            else
                h=h+1;
            end
        end
        %for every node in infNode, we infect its neighbours and remove it
        while(~isempty(infNodes))
            %make the 2nd row of infNodes 0, because it will stand for how
many
            %noes this node can infect
            infNodes(2,:)=0;
            %for each node calculate how many nodes it would infect by
            %broadcasting to all cliques he's a member of
            for h=1:length(infNodes(1,:))
                %take every clique of this second
                for k=2:noOfCliques+1
                    %if the current infected node is part of this clique
                    if(sum(cliques{g,k}==infNodes(1,h))>0)
                        %add to infNodes the number of uninfected nodes in
this
                        %clique
                        infNodes(2,h)=infNodes(2,h)+...
                            (length(cliques{g,k})-
sum(nodes(2,cliques{g,k})));
                    end
                end
            end
            %take the node that would infect most other nodes
            maxInfection=max(infNodes(2,:));
            %if the max is 0, it means that all nodes that are infected and
            %unanalyzed in this second cannot infect any other node. So we
            %break the while cycle.
            if(maxInfection==0)
                break;
            end
            %else find the node that infects most other nodes
            h=1;
            while(infNodes(2,h)~=maxInfection)
                h=h+1;
            end
            choosenNode=infNodes(1,h);
            %remove this node from the infNodes
            infNodes(:,h)=[];
```

```
            %infect all nodes that are neighbours with this node
            for(h=2:noOfCliques+1)
                %if the choosen node is a member of this clique
                if(sum(cliques{g,h}==choosenNode)>0)
                    %infect all nodes of this clique
                    nodes(2,cliques{g,h})=1;
                end
            end
            %increment the counters
            txNo=txNo+1;
        end


        %all nodes possible to infect in this second were infected. Save
        %the new infected count
        infection(f,g)=sum(nodes(2,:));
        g=g+1;
    end
end

%arrange the data in infection
for(f=1:20)
    value=infection(f,1);
    for(g=1:length(infection(f,:)))
        if(infection(f,g)==0)
            infection(f,g)=value;
        else
            value=infection(f,g);
        end
    end
end

infection=sum(infection)./20;
txNo=txNo/20
end
```

broadcastDelayed.m

```
function [infection,txNo] = broadcastDelayed(cliques,minUninfNodes)
infection=zeros(20,length(cliques));
txNo=0;
for(f=1:20)
    txNoPerNode=zeros(1,20);
    f
    %array of infection status
    nodes=[1:20;zeros(1,20)];
    nodes(2,f)=1;
    %for each second in cliques
    g=1;
    while g <= length(cliques)
        %if there's no clique in this second, continue
        if(isempty(cliques{g,2}))
            g=g+1;
            continue;
        end
        %get the nodes that are infected
        infNodes=nodes;
        h=1;
        while h <= length(infNodes(1,:))
            if(infNodes(2,h)==0)
                infNodes(:,h)=[];
            else
                h=h+1;
            end
        end
        %randomize the order of the nodes
        infNodes=infNodes(:,randperm(length(infNodes(1,:))));
```

```
        %find out how many cliques we have in this second
        h=0;
        while(~isempty(cliques{g,2+h}))
            h=h+1;
        end
        noOfCliques=h;
        %for each randomly picked infected node, randomly find a clique
        %containing it and that also contains non-infected nodes.
        for h=1:length(infNodes(1,:))
            %create a vector to randomly access the cliques
            cliqueAccess=randperm(noOfCliques);
            for k=1:length(cliqueAccess)
                %is our selected node (infNodes(1,h)) part of this clique?
                %and
                %do we also have more than minUninfNodes nodes that are not
infected?
                if(sum(cliques{g,1+cliqueAccess(k)}==infNodes(1,h))>0 && ...
                        length(cliques{g,1+cliqueAccess(k)})-...
                        sum(nodes(2,cliques{g,1+cliqueAccess(k)})) >=
minUninfNodes)
                        %infect the nodes in this clique
                        nodes(2,cliques{g,1+cliqueAccess(k)})=1;
                        txNo=txNo+1;

txNoPerNode(1,infNodes(1,h))=txNoPerNode(1,infNodes(1,h))+1;
                        break;
                end
            end
        end
        %all nodes possible to infect in this second were infected. Save
        %the new infected count
        infection(f,g)=sum(nodes(2,:));
        g=g+1;
    end
    txNoPerNode;
end

%arrange the data in infection
for(f=1:20)
    value=infection(f,1);
    for(g=1:length(infection(f,:)))
        if(infection(f,g)==0)
            infection(f,g)=value;
        else
            value=infection(f,g);
        end
    end
end

infection=sum(infection)./20;
txNo=txNo/20
```

broadcastLtdTx.m

```
function [infection,txNo] = broadcastLtdTx(cliques,maxTx)
infection=zeros(20,length(cliques));
txNo=0;
for(f=1:20)
    txNoPerNode=zeros(1,20);
    f
    %array of infection status
    nodes=[1:20;zeros(1,20)];
    nodes(2,f)=1;
    %for each second in cliques
    g=1;
    while g <= length(cliques)
        %if there's no clique in this second, continue
```

```
        if(isempty(cliques{g,2}))
            g=g+1;
            continue;
        end
        %get the nodes that are infected
        infNodes=nodes;
        h=1;
        while h <= length(infNodes(1,:))
            if(infNodes(2,h)==0)
                infNodes(:,h)=[];
            else
                h=h+1;
            end
        end
        %from these, remove the nodes that reached their maximum txs
        h=1;
        while h<=length(infNodes(1,:))
            if(txNoPerNode(infNodes(1,h))==maxTx)
                infNodes(:,h)=[];
            else
                h=h+1;
            end
        end
        %randomize the order of the nodes
        infNodes=infNodes(:,randperm(length(infNodes(1,:))));
        %find out how many cliques we have in this second
        h=0;
        while(~isempty(cliques{g,2+h}))
            h=h+1;
        end
        noOfCliques=h;
        %for each randomly picked infected node, randomly find a clique
        %containing it and that also contains non-infected nodes.
        for h=1:length(infNodes(1,:))
            %create a vector to randomly access the cliques
            cliqueAccess=randperm(noOfCliques);
            for k=1:length(cliqueAccess)
                %is our selected node (infNodes(1,h)) part of this clique?
                %and
                %do we also have nodes that are not infected?
                if(sum(cliques{g,1+cliqueAccess(k)}==infNodes(1,h))>0 && ...
                        sum(nodes(2,cliques{g,1+cliqueAccess(k)})) < ...
                            length(cliques{g,1+cliqueAccess(k)}))
                        %infect the nodes in this clique
                        nodes(2,cliques{g,1+cliqueAccess(k)})=1;
                        txNo=txNo+1;

txNoPerNode(1,infNodes(1,h))=txNoPerNode(1,infNodes(1,h))+1;
                        break;
                end
            end
        end
        %all nodes possible to infect in this second were infected. Save
        %the new infected count
        infection(f,g)=sum(nodes(2,:));
        g=g+1;

    end
    txNoPerNode;
end

%arrange the data in infection
for(f=1:20)
    value=infection(f,1);
    for(g=1:length(infection(f,:)))
        if(infection(f,g)==0)
            infection(f,g)=value;
        else
            value=infection(f,g);
```

```
        end
    end
end

infection=sum(infection)./20;
txNo=txNo/20
```