Bluenet II – A Detailed Realization of the Algorithm and Performance Analysis

Zhifang Wang, Zygmunt J. Haas, Robert J. Thomas
ECE Cornell University, Ithaca, NY 14850
{zfwang, haas}@ece.cornell.edu, rjt1@cornell.edu

Abstract

The recent interest in ad hoc networks, in general, and in the Bluetooth technology, in particular, has stimulated much research in algorithms for topology control of such networks. In particular, the issue of scatternet formation has been addressed by a number of papers in the technical literature. In [1], we have proposed the scheme of one such algorithm, the Bluenet algorithm, which allows efficient formation of scatternets.

In this paper, we further investigate the realization and properties of the Bluenets algorithm. The performance indices of resulting scatternets, such as, piconet density, usage of potential links, deviation of node degrees, average shortest path length, and maximum traffic flows are also studied. From the analysis, it is showed that the choice of p_0, the initial probability for each node to enter page state, is very important. Since each performance index only reflects one side of the scatternet performance, we need to make trade offs when selecting p_0 to build Bluenets.

I. Introduction

Our previous paper [1] proposed the Bluenet scheme based on the following rules:

Rule-1. Avoid forming further piconets inside a piconet;

Rule-2. For a bridge node, avoid setting up more than one connections to the same piconet;

Rule-3. Inside a piconet, the master tries to acquire some specific number of slaves; i.e. not too many and not too few while maintaining a connection only to active slave nodes if possible.

In the real world it is possibly very difficult to maintain rule-3. Therefore we only set the limitation about the largest number of slave nodes in a piconet. However in some extreme conditions, even this limitation has to be sacrificed in order to keep the connectivity of resulting scatternets.

According to our previous suggestion, the Bluenet scheme starts from a visibility graph. That is, through the "Inquiry" process of Bluetooth [2][3], each node has gotten to know the existence of its neighbors, including their Bluetooth addresses and clocks. At this time, all the Bluetooth nodes have no master or slave roles. Therefore we call them as void nodes, or phase-0 nodes. The scheme can be roughly organized into three phases:

- *Phase1*: Original piconets formed, and possibly some separate Bluetooth nodes may be left.
- *Phase2*: Separate Bluetooth nodes get connected to initial piconets.
- *Phase3*: Original piconets get connected through inter-piconet links to form a scatternet.

Original piconets are those piconets all of whose member nodes, i.e. master node as well as slave nodes were void nodes just before the time of joining the piconet. Namely, the original piconets are the fist bunch of piconets formed in the system. On the other side, the piconets formed later through phase-2 or phase-3 are called cross- piconets since all the member nodes, or all the slave nodes, in the piconet already belonged to some other piconet(s) before joining this one. For a slave node, though it may be a slave to multiple master nodes, its original master node is its first master. For a master in an original piconet, its original master is itself.

Once a node joins some original piconet, it is restricted from joining other piconets until its original master node instructs it to do so. After phase-1, the whole Bluetooth system is covered by original piconets. Possibly some nodes are left isolated because all of their neighbors are already associated with some original piconets. In this case, phase-2 is necessary. Otherwise, phase-2 is over passed.

It can be seen that [1] only provides a very rough scheme to build scatternets, with no details about the realization of the algorithm. For example, at the very beginning, how do the void nodes determine to enter page or scan state? How does a Bluetooth know when to switch its phases from phase-1 to phase-3? How does an isolated void node become aware of its situation? And

how does an original piconet set up inter-piconet links to get connected with its neighboring original piconets?

In the following parts of the paper, we will answer all of these protocol questions (in Section II) and discuss the performance indices (in Section III). Finally, conclusion is presented in Section IV.

II. Realization of Bluenet Algorithm

Just like mentioned in the introduction part, a Bluenet Scatternet could be formed through three phases. In our detailed realization of the algorithm, it can be illustrated as in Fig.1. Condition B represents "Is that possible to form an original piconet?" If yes, B=1, otherwise, B=0.

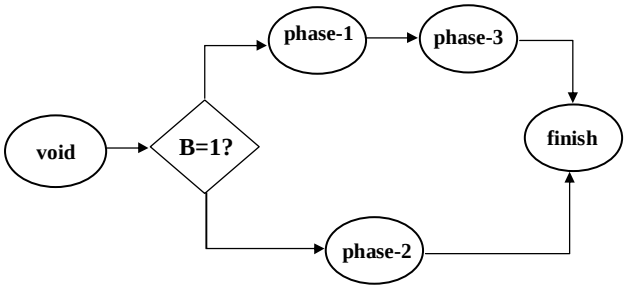

Fig .1. Phase transition for Bluetooth nodes in the Bluenet Algorithm

The realization of the algorithm requires each node to keep some local data records. Without loss of generality, we take *node-k* for example.

1. *ngbr_stat* – record the status for neighbors. *ngbr_stat(k,:,1)* contains the node ID's of all of *node-k*'s neighbors *ngbr_stat(k,:,2)* contains the original master ID's for each neighbor; and *ngbr_stat(k,:,3)* records how many times that *node-k* has paged the corresponding neighbors.
2. *nd_stat* – keep record of the node's own status. *nd_stat(k,1)* contains *node-k*'s original master ID, *nd_stat(k,2)* contains *node-k*'s phase number.
3. *msters* – if *node-k* is a master, *msters(k,:)* contains the node ID's of its slaves, otherwise, it is empty.
4. *slaves* – if *node-k* is a slave to some master node(s), *slaves(k,:)* contains the node ID('s) of all its master nodes. Otherwise it is empty.
5. *msters_p2* – only for phase-2 nodes. *msters_p2(k,:)* contains all the page-able neighbors of *node-k*.
6. *mstesr_p3* – only for phase-3 master nodes. *msters_p3(k,:)* contains all the slaves in *node_k's* piconet that are eligible to perform page action, i.e., whose *ngbrs_p3* still contains page-able neighbors.
7. *ngbrs_p3* – only for phase-3 slave nodes. *ngbrs_p3(k,:)* contains all the page-able neighbors for *node-k*.
8. *cross_scatters* – Only for original master nodes. Keep record of inter-piconet connections. *cross_scatters(k,:)* contains the node ID's of the original master nodes whose piconet is connected with *node-k*.

During the page/scan and match processes, participating Bluetooth nodes make decisions, select paged nodes (only for page nodes), exchange and update information based on their local records. That is, *node-k* only uses the parts associated with *k* in all above data structures. A page node selects its paged nodes only from a list of page-able nodes instead of from all its neighbors. That is because, as time goes on, some of its neighbors become unnecessary to be paged.

Since our algorithm starts from the completion of "Inquiry" state, the scatternet is formed through Page or Page Scan process of each node. We denote "page scan" as "scan" for simplicity in the later parts of this paper.

II.1 Page/Scan Logic

At first all phase-0 nodes, according to phase-0 page/scan logic in Table.1, determine whether to enter page or scan state. We assume that they do this randomly by a pre-assigned probability of p_0. The selection of p_0, helps to limit the number of piconets in the resulting scatternet. Without otherwise claim, all the following processes take *node-k* for example.

Table.1 Page/Scan Logic for phase-0 nodes

```
1   if (||A|| >0)
2      % page/scan by prob-p0,
3      x=rand( );
4      if(x<p0)
5           node-k enter page;
6           generate page-list
7            from A;
8      else
9           node-k  enter  scan;
10
11     end
12  else % ||A|| ==0
13     % enter phase-2,
14     init_p2(.)
15  end
```

A – the list of unknown neighbors,
||A|| – the number of unknown neighbors

For a phase-0 node, (and phase-1 nodes, too), *node-k*, its page-able neighbors vector *A* is defined as all of its neighbors whose original mater ID *node-k* doesn't know yet, i.e.

$$A = \{ngbr\_stat(k, j,1) : ngbr\_stat(k, j,2) = 0\}$$

Later if *node-k* decides to enter page state, it selects a random list of nodes only from A to page. We define *A* like that because if *ngbr-j* already joined some other piconet, it will not be restricted by its master from joining other piconet. Therefore we remove them in advance to save the unnecessary page time.

It is worth to note here that the following two statements are not equivalent:

(a) *ngbr-j*'s original master ID is unknown to *node-k*;

(b) *ngbr-j* has not got any original master ID.

It is possible that *ngbr_j* already joined some original piconet while *node-k* doesn't know that and still thinks that *ngbr_j* is page-able.

The function *rand(.)* chose x from a uniform distribution on the interval $(0,\ 1.0)$. Therefore *node-k* will enter page state with probability of p_0 and enter scan state with probability of $(1-p_0)$. And the scan nodes keep listening and wait for some other nodes to page it.

Obviously after several rounds of page and scan, some page nodes successful invite some scan nodes to become its slaves so that an original piconet results in. Once a page node becomes a master to a slave node, both of them enter phase-1 and keep that way until the master itself instructs to change. The phase-1 master node then takes over the control to determine the action for all the members in its piconet according to phase-1 page/scan logic in Table.2.

Table.2 Page/Scan logic for phase-1 piconets:

(a) for master nodes – Page or scan according to the following logic:

```
1  if (||A||>0)
2      if(ns<Nmax & ||up|| >0)
3          keep paging, chose paged nodes
4          from its unpaged neighbors,"up";
5      else %i.e. ns==Nmax or ||up|| ==0
6          alternate between page/scan
7          by prob-0.5,choose paged nodes
8          from A;
9      end
10 else
11     node-k and all its slaves enter
12     phase-3;
13     init_p3(.);
14 end
```

ns – the total number of slaves in its piconet,
||*A*|| – the number of unknown neighbors,
||*up*|| – the number of unpaged nodes among its unknown neighbors.

(b) for slave nodes –

```
keep scanning only for information exchange,
until otherwise instructed by its master.
```

When the phase-1 master node gets to know all about its neighbors' states, i.e., $A = \phi$, all the member nodes in its piconet enter phase-3. The master starts to instruct all of its slaves to get ready for setting up inter-piconet links. Function on line-13 *init_p3(.)* is to initialize *msters_p3*, containing all the slaves, for the master node and to initialize *ngbrs_p3*, containing all of its neighbors, for each slave.

It is possible that some nodes will be left isolated finally because all of its neighbors already joined some piconet and refused to accept its page invitations. These nodes would enter phase-2 state and try to get connected with its neighbors that belong to different original piconets and then go to finished state, according to below phase-2 page/scan logic. Obviously if no phase-2 nodes left at all, this part will be skipped automatically.

Table.3 Page/Scan logic for phase-2 nodes

```
1  if (ns_p2>0)
2          keep paging its page-able neighbors;
3          //if ns<Nmax
4          //      the page is for set up
5          //      master-slave link
6          // if ns==Nmax
7          //      the page is only for
8          //      information exchage.
9  else
10     enter "finish" state
11 end
```

mster_p2 – the neighbors that belongs to a different piconet than those node-k has connected with.
ns_p2 – the total number of available neighbors in msters_p2

The *init_p2(.)* function is to initialize the local record msters_p2, for the new phase-2 node, say, *node-k*. Initially, *msters_p2* contains all of *node-k*'s neighbors. Whenever a new slave node joins *node-k*'s phase-2 piconet, *node-k* will update its msters_p2, through function *updat_p2(.)*, by removing those neighbors, which belong to the same piconet as the new slave node. When there is no page-able neighbor left in msters_p2, the phase-2 node enters "finish" state.

After initializations, the Bluetooth nodes in a phase-3 piconet begin their page/scan process according to the following phase-3 page/scan logic in Table.4.

Whenever the phase-3 slave, *node-k*, from original piconet of *master-m*, gets response from a new node, say *node-j*, from a new original piconet whose belonging original piconet, say, *piconet-n*, has not been connected with *piconet-m*, *node-k* will update its *ngbrs_p3(k,:)*,

through *updat_p3(.)*, by removing some of its neighbors that belong to the new original *piconet-m*. If there is no page-able neighbors for *node-k*, i.e., *ngbrs_p3(k,:)* is empty, then remove *node-k* from its master's *msters_p3(m,:)*, i.e., it is no longer eligible to page for phase-3. When there is no eligible slaves left in the mster_p3(m,:), the whole piconet enter "finish" state. The same update will be done for *node-j* and *piconet-n*, too.

Table.4 Page/scan logic for phase-3 nodes

```
(a) master nodes – keep scanning but only for
    exchange information;
(b) slave nodes – enter page/scan according to
    the logic below;
```

```
 1 if (ns_p3>0)
 2     if(ns_p3>1)
 3         randomly select one available slave
 4         to page, while the others scan
 5     else // (i.e. ns_p3==1)
 6         all slaves randomly enter page or
 7         scan by prob-0.5
 8     end
 9 else
10     the whole piconet enters "finish" state
11 end
```

msters_p3- contains the slaves in the original piconet;
ngbrs_p3 – the list of available neighbors for each slave ;

II.2. Match process

During the page/scan process, if *node-k* pages *nod-j* while node-j is in scan state and hears the page, the following actions taken is called match process. First the matching two nodes would set up a temporary master-slave link, and then exchange the node-state information. Finally they will determine whether to keep this link or not. There are only four cases in which a master-slave link will be kept finally, i.e.

a. A void node pages another void node;
b. A phase-1 master with less than $N_{\max}$ slaves pages a void node;
c. A phase-2 node with less than $N_{\max}$ slaves pages another node, which belongs to a different piconet other than those already connected with the phase-2 master;
d. A phase-3 slave node pages another phase-3 slave node. Only if the former has less than $N_{\max}$ slaves in its piconet and the two original piconets, to which the two slaves belong, are not connected yet.

For different cases, the information exchange and update are different, too. Now we will explain them in more details. We denote the page node, its original master, its phase number, and the number of slaves in the page node's piconet as `page_nd`, `pg_m`, `pg_p`, and `pg_ns` respectively. And the corresponding terms for the scan nodes are denoted as `scan_nd`, `sc_m`, `sc_p` and `sc_m`..

If scan_nd and page_nd both are void nodes, the match process is like below:

Table.5 Match process for two void nodes

```
1   if(pg_p+sc_p==0)% both are void nodes
2       appdslave (page_nd,scan_nd);
3       nd_stat(page_nd,:)=[page_nd 1];
4       nd_stat(scan_nd,:)=[page_nd 1];
5       ngbr_stat=exchg_p0(page_nd,…
6                          scan_nd);
```

Line-2 is to finally set up the master-slave link page_nd→scan_nd. Line-2 and 3 is to update the node states. Line-5 is to exchange information for updating each other's ngbr_stat.

If page_nd is a phase-1 master and scan_nd is a void node, the match process is as below:

Table.6 Match process for phase-1 masters

```
1  elseif(pg_p==1 & sc_p==0)
2      if(pg_nsl>0 & pg_nsl<Nmax )
3          appdslave(page_nd,scan_nd);
4          nd_stat(scan_nd,:)=[page_nd 1];
5          exchg_p1(page_nd,scan_nd );
6      else %(pg_nsl==Nmax)
7          exchg_p1full(page_nd,scan_nd);
8      end
```

Line-2-3 shows that if page_nd has less than $N_{\max}$ slaves, the final master-slave link is kept. The function *exchg_p1(.)* on line-5 is the exchange of information for phase-1 piconet when a new slave joins. The master will provide the new slave information about all the other members in the piconet and broadcast the new slave's information in the piconet so that the other member nodes can update their *ngbr_stat*, too. Otherwise if the page_nd's piconet already has $N_{\max}$ slaves, the match pair exchanges the information by the function *exchg_p1full(.)* on line-7, i.e., only the scan_nd update its *ngbr_stat* by collecting node state from page_nd's piconet. Since the scan_nd doesn't join page_nd's piconet, its status remains unchanged; there is no need for the page_nd to broadcast scan_nd's status in its own piconet.

If page_nd is a phase-2 node, the match process between the page/scan pair is depicted as in Table.7. The function *check_cross()* in Line-2 is to check if page_nd has connected with the piconet which scan_nd belongs to. If yes, the function returns 0, means the following action is unnecessary. Otherwise, the following action is

carried on. If page_nd's piconet is not full yet, the master-slave link will be kept and node state be changed and the phase-2 node also expend its local record cross_scatters(k,:) to append the new piconet it has already connected. Otherwise, only update msters_p2 for the phase-2 node and exchange neighbor status information.

Table.7 Match process for phase-2 nodes

```
elseif(pg_p==2)
    if(check_cross(page_nd,scan_nd,nd_stat))
       if( pg_nsl<Nmax)
            appdslave(page_nd,scan_nd);
            nd_stat(page_nd,1)=page_nd;
            pg_m=page_nd;
            expd_cross(pg_m,sc_m);
       end
       update_p2(pg_m,sc_m,nd_stat);
       exchg_p2(page_nd,scan_nd);
```

If the match process is going on between two phase-3 slaves, their match process can be showed as:

Table.8 Match process for phase-3 nodes

```
elseif(pg_p==3 & sc_p==3 & sc_m~=scan_nd)
  if(check_cross(page_nd,scan_nd,nd_stat))
      tmp=[page_nd scan_nd];
       ind=find([pg_nsl sc_nsl]<Nmax);
       if(~isempty(ind))
            mst_nd=tmp(ind(1));
            slv_nd=tmp(find(tmp~=mst_nd));
            appdslave(mst_nd,slv_nd);
            expd_cross(pg_m,sc_m);
            update_p3(pg_m,sc_m);
            update_p3(sc_m,pg_m);
       else
            update_p3full(page_nd,scan_nd);
       end
      ngbr_stat=exchg_info(page_nd,…
                           scan_nd,);
  end
```

Line-3 and 4 show that if page_nd's piconet is not full, page_nd will take scan_nd as its new slave. Otherwise, if scan_nd's piconet is not full, the opposite master-slave link would be set up finally. Otherwise, if both piconets are already full, the match pair only to update each other's ngbrs_p3, by removing the other node from its local record of ngbr_p3, meaning that the other node is no need to page later.

Table.9 Match process for other cases

```
elseif(pg_m~=sc_m)
   % only if the match nodes comes from
   % different orig-piconet
    ngbr_stat=exchg_info(page_nd,…
           scan_nd,ngbr_stat,nd_stat);
   %exchage ngbr info
end
```

For the other cases except all above, if belonging to different piconets, they will only exchange the node information about all the member nodes in each other's piconet, as in Table.9.

II.3. Back-off Algorithm to guarantee connectivity

The only chances in the algorithm to cause isolation for the scatternets are:

a. A master node can't have more than $N_{\max}$ slaves in its piconet
b. Phase-3 masters don't set up inter-piconet master-slave links with phase-3 slave or masters.

In the realization of Bluenet Algorithm, we try to decrease the probability of isolation as small as possible. After the finished state, all the Bluetooth nodes need to examine their neighbor state records, to check if any neighbor, from a different original piconet than those directly connects with its own, is left unconnected. If yes, the node, say *node-k*, will inform its original master, *mster-m*, and ask its master node to make sure the questioned neighbor node, say, *node-j*, can be finally connected. First the master node, *master-m*, contacts with all of its neighboring original piconets from its *cross_scatters(m,:)*, to see *node-j* has potential to be really unconnected. If node-j falls in any piconet that is directly connected with master-m's neighboring piconets, it is fine. Otherwise, *node-k* has to set up a master-slave link with *node-j*. They do it by entering page or scan prob-0.5 independently each other and finally can match each other. In fact, above cases occurs very rarely. Mostly the process can be just omitted.

III. Performance Analysis

It is necessary to distinguish between efficient scatternets and inefficient ones but it is uneasy to find ways to make the evaluation. Based on recent literatures [4][5], we decide to adopt following performance indices to evaluate the quality of resulting scatternets.

a. Piconet Density – $\xi = n_{mst}/b_n$,

which is defined as the ratio between the number of piconets over the number of Bluetooth nodes.

In some extent this index reflects the interferences level among the piconets in a resulting scatternet. Because all piconets share the common 79 Bluetooth channels, the more piconets exist in the same neighborhood, the heavier interference among them. Therefore, a too high piconet density should be avoided.

b. Link coverage – $link\_p = N_L/(b_n - 1)$,
Which is defined as the ratio between the number of links N_L in a scatternet and the smallest number of links that is needed to form a connected network $(b_n - 1)$.

Obviously, a connected scatternet always has $link\_p \geq 1$. This index represents the usage of potential links in a scatternet. In order to form an efficient communication network, the resulting scatternet should bear a decent of connectivity. Either extremity is undesirable. Because too large a $link\_p$ means wastes of network resource since each active link costs some node bandwidth to maintain. If $link\_p$ is too small, it may cause bottlenecks for multi-pair communications.

c. Degree Deviation – $\sigma = \sqrt{\frac{1}{b_n - 1}\sum_{i=1}^{b_n}(\rho_i - \bar{\rho})}$,
Node degree ρ_i is defined as the number of piconets that the Bluetooth node-*i* joins in. σ and $\bar{\rho}$ are the standard deviation and the mean of all node degrees.

A "good" scatternet may spread its network resources as evenly as possible otherwise the bottlenecks in the system may bring down the whole network's performance. Therefore the index of node deviation should not become too large in the resulting scatternet.

d. Average Shortest Path Length – $\bar{d} = \frac{2}{b_n(b_n - 1)}\sum_{ij} d_{ij}$,
d_{ij} is the short path length (hop count) between *node-i* and *node-j* in the resulting scatternet.

This index shows the routing efficiency of the resulting scatternet. It provides us with an estimate of the average routing delay in the resulting scatternet.

e. Max Traffic Flow – MTF_m is defined as the average max traffic flow that can be carried by the resulting scatternet for all *m*-pairs of communication nodes. This index reflects the information carrying capacity for the resulting scatternets.

MATLAB simulations are carried on to analyze performance of resulting Bluenet scatternets from the realization process described in Section II. At first b_n Bluetooth nodes are uniformly distributed in a square area with node density 10 [$nodes/m^2$]. 100 sample scatternets (with different node distribution) are generated each parameter of p_0, the initial probability for nodes to enter page. The maximum number of slaves in a piconet $N_{\max}$ is set to 5.

Fig.2. shows that the piconet density increases when p_0 become larger. In order to limit the number of piconets in a scatternet, we should choose p_0 to be not too large. Fig.3 ~5 shows the trends of $link\_p$, node degree deviation, and average shortest path length when p_0 increases. Clearly, $link\_p$ and node degree deviation increase when p_0 goes up, while Average Shortest Path Length becomes smaller. This is easy to understand. Since when more links are used in the scatternet, there are more possible paths between any two pair of nodes, and the shortest path length between them can possibly be decreased. Fig.6. presents the MTF performance with different p_0 for 40-node Bluetooth system with a uniform node distribution. From Fig.6 we can see that when $p_0 = 0.2$, the resulting scatternets have best information carrying capacity. With the combination of all performance indices above, we found that $p_0 = 0.2$ is an appropriate choice.

IV. Conclusion

This paper presents a detailed realization for the Bluenet Algorithm first proposed in [1]. The realization shows that the algorithm is applied in a distributed way, i.e., each Bluetooth nodes carry on its page or scan process based on the local knowledge about the network

Performance analysis is also performed to show the effect of p_0 on the performance indices such as piconet density, link usage and node degree deviation, average shortest path length and maximum traffic flow. Since each index only reflects one side of the scatternet performance, some trade off has to be made when determine how to build a scatternet.

Reference

[1] Z. Wang, R. J. Thomas, and Z. Haas, "Bluenet -- a new scatternet formation scheme", in *Proceedings of the 35th Hawaii International Conference on System Science (HICSS-35)*, Big Island, Hawaii, January 7-10 2002

[2] Bluetooth Special Interest Group, " Specification of the Bluetooth System, version 1.0B, http://www.bluetooth.com/

[3] B. A. Miller, and C. Bisdikian, *Bluetooth Revealed*, Prentice Hall PTR, upper Saddle River, NJ 07458, 2001

[4] G. Miklos, A. R. Racz, Z. Turanyi, A. Valko, and P. Johansson, "Performance Aspects of Bluetooth Scatternet Formation", *MobiHoc 2000*, pp147-148, Boston, Aug 2000

[5] "A Bluetooth Scatternet Formation Algorithm", C. Law and K.Y. Siu, Proceedings of the IEEE Symposium on Ad Hoc Wireless Networks 2001, San Antonio, Texas, USA, November 2001

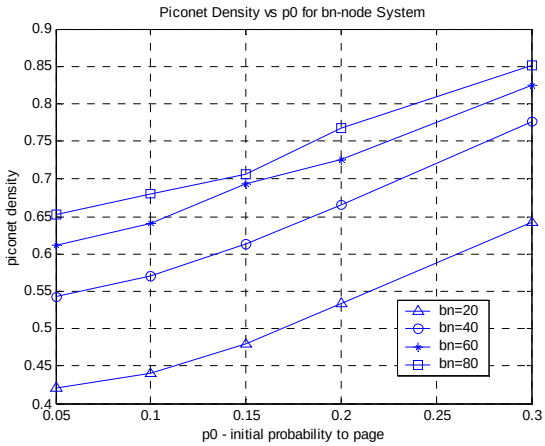

Fig2. Piconet Density vs. p_0

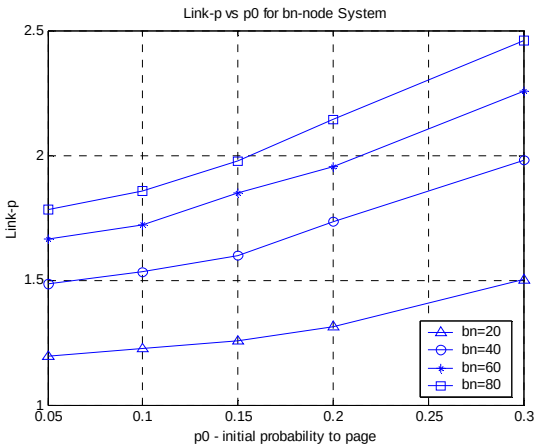

Fig 3. Link-p vs. p_0

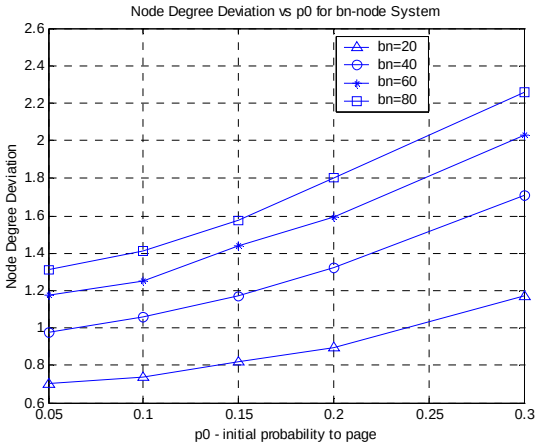

Fig 4. Node Degree Deviation vs. p_0

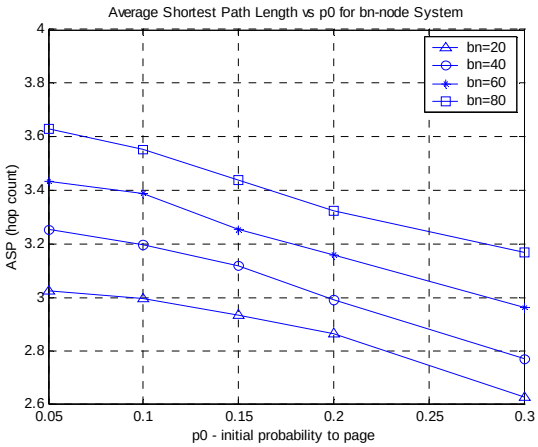

Fig.5. Average Shortest Path Length vs. p_0

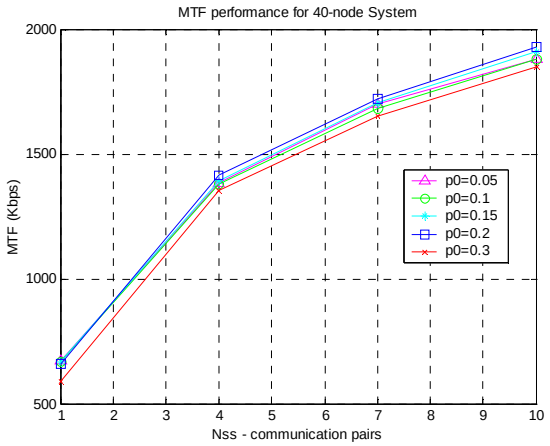

Fig.6. MTF performance for 40-node system