

Automatic Topology Generation for a Class of Wireless Networks

Raghuraman Rangarajan and Sridhar Iyer
KReSIT, IIT Bombay, India
email: {raghu,sri}@it.iitb.ac.in

ABSTRACT

In this paper we present the Wireless Infrastructure Deployment (*WIND*) tool to automatically generate topologies for wireless networks. The heuristic tool considers the nodes to be deployed, their characteristics and abstract deployment scenarios as input and generates topologies satisfying node QoS requirements. *WIND* does this by recursively constructing complex network elements out of simpler elements while satisfying the constraints.

I. INTRODUCTION

Wireless technologies allow rapid deployment of networks by alleviating the difficulty and expense of deploying guided media between communicating nodes. They also allow rapid reconfiguration of networks with growth.

Currently, planning for wireless networks is done either in an *ad hoc* manner or through complex site survey with emphasis on RF propagation studies.

For example, in an 802.11 network the APs are distributed regularly across the site, i.e. the density of APs is uniform or they are distributed based on RF propagation studies. The fixed clients are also uniform in their distribution, while mobile clients are located in a random manner or are densely cramped.

While such an approach is valid for single-hop networks, providing one-hop backbone connectivity for all nodes and site surveys of large deployment areas restrict the scalability of such network planning techniques ([1]).

At the other end of the spectrum, *MANET* topologies are constructed *after* the network is deployed and is more of a *best-effort* attempt. Once the nodes are deployed as such, research has mainly concentrated on resource allocation, routing for QoS requests or topology control.

The successful deployment of 802.11 *WLANs* has demonstrated that internet/intranet connectivity remains the primary application driver. With the increase in capability of the network elements, there are many network applications that require QoS provisions, such as multimedia, collaborative work and distributed applications. The high bandwidth demand of such applications makes infrastructure support a necessity.

The multi-hop mesh connectivity of hybrid networks ([1]) presents a compromise between both these approaches and is suitable for scaling of networks while keeping the application scenario requirements (i.e. appropriate backbone connectivity) in mind. While hybrid networks result in larger wireless coverage and in some cases improved throughput, the pitfall of this approach is the increased channel contention due to the wireless links ([3]). Hence, network planning of such networks has to take the above factors into consideration.

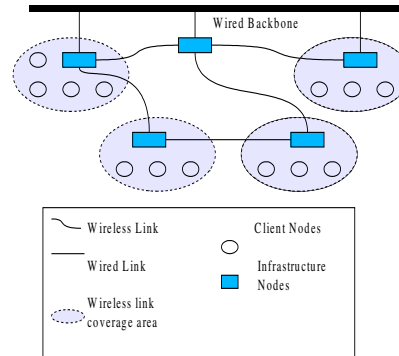


Fig. 1. An example mesh network.

In this paper we present a framework for building hybrid networks. The Wireless Infrastructure Deployment tool (*WIND*) generates appropriate topologies for hybrid networks under given constraints.

It considers the number of nodes deployed and their associated characteristics (number of links, application traffic etc.), an abstract graph of the deployment area and QoS constraints to generate appropriate logical topologies.

The crucial idea in our approach to this problem is the recursive construction of the topology. *WIND* starts with the set of network elements to be deployed. At each stage of the recursion, *WIND* successively generates a sub-graph of the topology by deriving it out of the sub-graphs generated at the previous stage and hence finally resulting in the network topology. This is done while satisfying the constraints.

II. RELATED WORK

Research work has mainly concentrated on topology control for adhoc wireless networks. The studies have concentrated on adjusting the transmission range (i.e. power control) for constructing a topology satisfying a given throughput ([4], [5]).

Recently, Jia et al presented a work [6], which attempts to create a QoS aware topology using power control. Even though they deal with meeting the overall QoS requirements, they do this by adjusting the transmission range of the nodes.

Marsan et al presented a technique in [7] to optimise the topology of Bluetooth, which aims at minimizing the maximum traffic load of nodes (and thus minimizing the power consumption).

The construction of a network topology from *ground-up* that can meet the QoS requirements of all its network elements has not yet been studied in detail.

TABLE I
INFORMATION BASE.

NU Type Information						
	Out Traffic	In Traffic	Src Addr	Destn Addr	Link Type	Mobility
NU_{PDA}	100	10000	$\langle N_{PDA} \rangle$	$\langle N_S \rangle$	1	Yes
NU_{WS}	1000	100000	$\langle N_{WS} \rangle$	$\langle N_S \rangle$	1	No
NU_S	1000000	10000	$\langle N_S \rangle$	<i>Undefined</i>	2	No
NU_{Relay}	500000	500000	<i>Undefined</i>	<i>Undefined</i>	1,2	No
...

III. PROBLEM STATEMENT

A wireless network (figure 1) typically consists of two distinct categories of network elements. One category of network elements are the client nodes, which represent the application users and providers of the network. The other category consists of the infrastructure nodes (i.e. Access Points, routers etc.) which are typically put in place with the primary purpose of aggregating and transporting the traffic for the clients.

We formalise our problem of *network planning* similar to the heuristic defined for cellular networks in [8]: *Given* the client nodes to be deployed, their characteristics and the deployment layout *construct*, the network topology *subject* to QoS constraints and *minimization* of network infrastructure nodes.

The node characteristics (model) are defined in an information base. The number of nodes to be deployed, the deployment layout and the constraints applied on the topology construction algorithm are defined as input parameters.

IV. NETWORK MODEL

A. Definitions

Some terms used in our network model are :

- *Node Unit (NU)*: Network elements; i.e. all client and infrastructure nodes.
- *C*: Capacity of a link in bytes/sec.
- *addr*: Address of a network element.
- *load*: Average offered load in bytes/sec.

B. Information Base, Input Parameters and Constraints

WIND makes use of an *information base* containing node and link type information about the network under study. The *information base* stores the following information :

- *Type* of a NU deployed in the network. The parameters that define the type are :
 - *Application scenarios (AS)*: Represents the source/sink traffic due to each application traffic type. Application traffic of a NU is represented by a tuple consisting of the average incoming and outgoing loads, the source and the destination addresses.

$$traffic = \langle load_{out}, load_{in}, addr_{src}, addr_{destn} \rangle$$

where,

$load_{out}, load_{in}$ = outgoing and incoming load respectively.

$addr_{src}, addr_{destn}$ = source and destination address respectively.

- *Total Load*: Summation of all AS traffic loads.
- *Links*: The type and the number of links present in a NU.

– *AS-Link mapping*: This mapping determines the link on which a particular AS traffic is carried.

– *Mobility*: Represents whether a node is mobile or not.

The *information base* also defines the NU types of the relay nodes deployable in the network. All infrastructure nodes (i.e. APs, routers, switches etc.) are considered as relay nodes. While constructing the topology, *WIND* chooses an appropriate relay node based on its link properties and the QoS criteria.

• *Link types*: Every link defined in a NU type has the following properties defined :

- Link name.
- *Capacity* of the link, *C*.
- *Link specification*: A MAC abstraction function is defined to capture the characteristics of the particular MAC scheme used. This is used to abstract the underlying MAC scheme. The function $maxNodes(C,l)$ computes the maximum number of network elements that can access the link were each element has a load *l*.

Table I shows a part of an information base for a network deploying PDAs (NU_{PDA}) and workstations (NU_{WS}). Other NUs representing servers and relay nodes are also shown.

Input parameters :

- The number of NUs of each type to be deployed.
- *Deployment layout*: The deployment layout is represented as a graph, corresponding to the deployment scenario. The graph is modeled as $G = (V, E)$, where *V* is the set of nodes representing areas where network elements are deployed and *E* is the set of edges between nodes indicating the physical connection between areas. An example layout for a simple office plan is shown in figure 2.
- *Affinity factor (af)*: Affinity factor is the probability with which a NU type is attracted to a node in the deployment layout, i.e. present in one of the *nodes* in the layout. While for non-mobile NUs, *af* represents the areas where such nodes are deployed, for mobile NUs *af* captures the probability of a mobile NU visiting that particular node. We adapt this definition of affinity factor from the attraction points characteristic defined in [2]. Nodes in the deployment graph are typed with the *af* of each NU type. Table II shows an example listing of affinity factors for NUs (PDAs and workstations) for the deployment layout in figure 2.

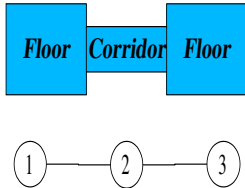


Fig. 2. An office floor plan and its corresponding deployment layout.

TABLE II
AFFINITY FACTORS.

Node	1	2	3
NU_{PDA}	0.6	0.2	0.2
NU_{WS}	0.6	0	0.4
NU_S	1	0	0
...

Optimization constraints : The tool also takes as input the constraints applied on the topology being constructed. The constraints can be the QoS requirements demanded by the applications running in the network. Other constraints such as a bound on the number of infrastructure NUs deployed by the system (i.e. infrastructure cost) can also be enforced.

V. TOPOLOGY CONSTRUCTION

A. Preliminaries

The basic building block in the topology construction algorithm is that of a *Composite Unit (CU)*. A CU is recursively defined as a virtual network element constructed out of one or more CUs or NUs. The properties of a CU are derived out of the properties of its child CU/NUs.

A composite unit is defined as:

$$CU = (CU' | NU)^+$$

where properties of CU are :

- $links = unreserved.links((CU' | NU)^k)$, where k is the number of units forming the CU.

The function returns the unused links of underlying network elements. Once the bandwidth on a link is allocated, a link is considered to be used. Hence the function calculates the links of child CU/NUs on which bandwidth is still available.

- $traffic = outgoing.traffic((CU' | NU)^k)$, where k is the number of units forming the CU.

Outgoing traffic is the traffic which is not destined for any descendant of this CU and hence is considered as unfulfilled traffic. Traffic of underlying CUs which have found a path (i.e. both source and destination CUs are descendants of the same CU), are considered to be satisfied. Therefore, the function calculates traffic which is bound for some NU not a descendant of this CU.

- $totalload = \sum_{\forall cu \in child(CU)} unusedload(cu)$.

The unused load of child CUs is assigned to their parent as its load.

These functions are discussed in detail while presenting the algorithm.

B. WIND overview

The tool (figure 3) consists of three modules.

1. *Preprocessor*: This module configures the inputs for use by the topology generator. It takes as input the information base (NU and link models) and the input parameters (deployment layout, affinity factors of NUs and the number of elements of each NU to be deployed).

It constructs the following for use by the topology generator :

- *NU list*: Using the NU models, the deployment layout graph and the affinity factors associated with each NU, the list of NUs to be deployed in each area in the deployment layout graph is calculated.

- *CU rules*: The topology generator also requires a set of rules while generating the topology. The rules define the criteria for CU formation and relay node selection.

2. *Compute CU*: This is the topology generator module which uses the CU rules and NU list to construct the topology. The module takes in one more input, the optimization criteria, which defines the QoS constraints on the construction process. At each stage in the construction process, the module considers the CU rules and makes sure that the criteria are kept satisfied. The process works as follows :

- (a) Given a list of nodes, the function calculates the nodes to be used for construction of the CU using the CU rules.

- (b) The properties of the constructed CU and the nodes are set using the criteria defined for optimization.

- (c) The algorithm continues until all nodes in the list are exhausted. The generated list of CUs is then recursively fed to the module again, until the root CU of the topology is generated.

3. *Output topology*: The constructed topology is then processed for input to a simulator for verification.

C. Implementation and validation by simulation

We have currently implemented a version of the tool for single-hop wireless networks, providing guarantees on bandwidth. The tool takes input parameters and information base as input files. The description language for the input file is based on the XML description formats defined for simulation in Opnet.

The algorithm 1 works as follows :

1. We calculate the number of NUs deployable in each node in the deployment layout graph (Step 4) and pass it to the $computeCU()$ function (Step 5). $computeCU()$ returns a CU for each node which is collected in $cuList$.

2. Finally we compute the root CU of the network using the above list of CUs.

3. We print the topology by performing a DFS on the root CU.

Given a list of NU/CUs, $computeCU()$ is defined as follows :

1. We first find the common link types among all NU/CU in $cuList$. This results in an unordered list of NU/CUs for all link types (Step 16).

Algorithm 1 Pseudo-Algorithm for topology generation.

```

1: procedure WIND( $ib, ip$ )  $\triangleright ib$ : Info base,  $ip$ : Input
   parameters
2:  $cuList \leftarrow NULL$ 
3: for all  $v \in V(ip.G_{DL})$  do  $\triangleright G_{DL}$ : Deployment layout
4:  $deployedList \leftarrow \bigcup_{v_i} (v.af_i \times ip.num_{NU_i}).NU_i$   $\triangleright af$ :
   affinity factor
5:  $cuList \leftarrow cuList + computeCU(deployedList, ib)$ 
6: end for
7:  $cuList \leftarrow computeCU(cuList, ib)$ 
8:  $printTopology(cuList)$ 
9: end procedure

10: procedure COMPUTECU( $cuList, ib$ )
11: if  $sizeOf(cuList) = 1$  then
12:  $return cuList$ 
13: end if
14:  $newCUList \leftarrow NULL$ 
15:  $L \leftarrow linktypes\_present(cuList)$ 
16: for all  $lt \in L$  do
17:  $cuList_{lt} \leftarrow cuList_{lt} + \{cuList[i], cuList[i].linktype = lt\}$ 
18:  $t \leftarrow \frac{\sum_{j} cuList[j].total\_load}{sizeOf(cuList)}$   $\triangleright t$ : average load
19:  $n = lt.maxNodes(lt.C, t)$ 
20: while  $cuList_{lt}$  NOTEMPTY do
21:  $new\ cu'$ 
22:  $childCUList \leftarrow NULL$ 
23:  $m \leftarrow 0$ 
24: while  $m < n$  do
25:  $cu \leftarrow cuList_{lt}.getnext()$ 
26:  $cu'.child(cu)$ 
27:  $childCUList.add(cu)$ 
28:  $cuList_{lt}.remove(cu)$   $\triangleright cu$  also removed from  $cuList$ 
29:  $m \leftarrow m + 1$ 
30: end while
31:  $cu_{relay} = findRelayNode(lt, t)$ 
32:  $cu'.child(cu_{relay})$ 
33: for all  $cu \in childCUList$  do  $cu.resetProperty()$ 
34: end for
35:  $cu_{relay}.resetProperty()$ 
36:  $cu'.setProperty()$ 
37:  $newCUList.add(cu')$ 
38: end while
39: end for
40:  $return computeCU(newCUList, ib)$ 
41: end procedure

```

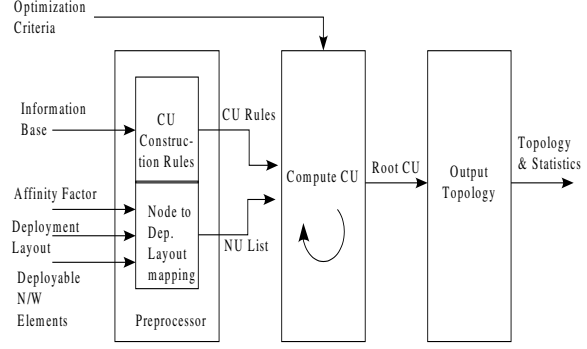


Fig. 3. WIND: Tool Overview.

2. We now calculate the maximum number of nodes n that can be merged together considering the average load (Step 17). We do this by using the MAC abstraction function ($maxNodes(C, t,)$) defined for each link type (Step 18).

3. We form now a CU and add a maximum of n children to it (Step 19-32). We then set the properties of the child CUs and the newly formed CU .

Once n NU/CUs are added as children to the newly created CU , we select an appropriate NU_{Relay} from the information base and add it to CU (Step 27). A child CU having a suitable relay link can also be selected as a relay node (for example, nodes having multiple links can act as relays).

Before defining the properties of the CU , we set the properties of the child CUs . The $resetProperty()$ (Step 29, 31) function calculates :

- Used links: For all NU/CUs merged together to form the CU , we set the link on the basis of which they were merged as a *used* link.

- Fulfilled traffic: Traffic from one child CU which is destined for another child CU is considered to be fulfilled traffic.

Now the properties of the CU ($setProperty()$) are defined as (Step 32) :

- links = Unused links of NUs/CUs + Outgoing link of Relay Node

- load = $\sum_{i=1}^k traffic$

- traffic = Unfulfilled traffic of child NUs/CUs .

4. The computed CU is added to a $cuList$ and the function recursively calls itself until the root CU is computed.

D. An illustrative example

We explain the workings of the algorithm with an example. The example builds a network topology for seven PDAs and workstations deployed in an office environment. The office floor plan and its corresponding deployment layout are shown in figure 2. The affinity factors for the nodes are defined in table II.

Both type of nodes run an application accessing some service from server NU_S . The corresponding information base (table I) represents the node information.

Figure 4 shows the progression of the $computeCU()$ algorithm. It recursively then builds the topology by bringing together NU/CUs based on CU rules. The algorithm re-

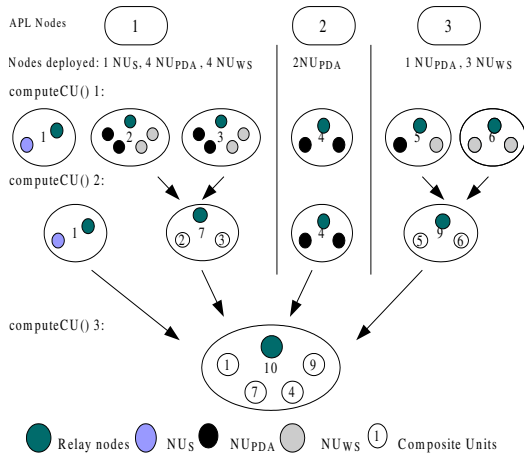


Fig. 4. computeCU() algorithm for given example.

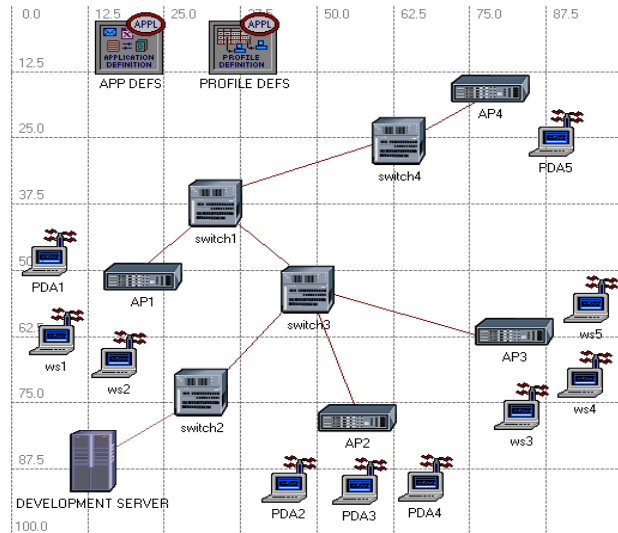


Fig. 5. Constructed topology.

turns finally the root CU of the network which represents the topology.

E. Validation of the topology

An example generated topology for the following parameters is shown in figure 5 :

- Input parameters: 5 PDA and 5 workstations to be deployed each running a ftp client with an average load of 10 KBPS and 100 KBPS respectively.
- Deployment layout is as shown in figure 2.

The output topology of the tool was fed as input to Opnet (along with suitable node and link models) to validate the generated topology. The average observed bandwidth was 9.877 KBPS and 99.838 KBPS for PDAs and workstations respectively, hence validating the generated topology.

VI. CONCLUSION AND FUTURE WORK

In this paper we have presented a tool (*WIND*) for planning hybrid networks. The tool generates automatically appropriate topologies based on heuristic inputs. The topology generated by *WIND* was fed to a simulator to validate it. Currently the tool has been implemented for 802.11 networks. In the next stage, we intend to expand the scope of the tool to take care of issues in multi-hop mesh networks. Also, we have only considered a static network for analysis. Mobile nodes will present a more dynamic environment with a difficult to predict traffic patterns. Another aim is to verify whether the topology generated is optimal, i.e. the cost of the infrastructure deployed is minimal for the constraints specified.

REFERENCES

- [1] Yang, Conner, Guo, Hazra, and Zhu, "Common Wireless Adhoc Network Usage Scenarios," *Internet draft*, 2003.
- [2] M. Feeley, N. Hutchinson, and S. Ray, "Realistic Mobility for Ad Hoc Network Simulation," *ADHOC-NOW*, LNCS 3158, pp. 324-329, 2004.
- [3] S. Lee, S. Banerjee, B. Bhattacharjee, "The Case for a Multi-hop Wireless Local Area Network," *IEEE Infocom*, 2004.
- [4] L. Hu, "Topology Control for Multihop Packet Radio Networks," *IEEE Trans. on Communications*, vol. 41, no. 10, pp. 1474-1481, 1993.
- [5] T. Hou and Victor O.K. Li, "Transmission Range Control in Multihop Packet Radio Networks," *IEEE Trans. on Communications*, vol. 34, no. 1, pp. 38-44, Jan 1986.
- [6] X. Jia, D. Li and D. Du, "Topology Control in Adhoc Wireless Networks," *IEEE Infocom*, 2004.
- [7] M. A. Marsan et al, "Optimizing the Topology of Bluetooth Wireless Personal Area Networks," *IEEE Infocom*, 2002.
- [8] B. Jabbari, G. Colombo, A. Nakajima, and J. Kulkarni, "Network Issues for Wireless Communications," *IEEE Commun. Mag.*, vol. 33, no. 1, pp. 88-98, January, 1995.