A Scheme for Dynamic Monitoring and Logging of Topology Information in Wireless Mesh Networks

Cristian Popi, Olivier Festor

A Scheme for Dynamic Monitoring and Logging of Topology Information in Wireless Mesh Networks

Cristian Popi, Olivier Festor
MADYNES - INRIA Nancy Grand Est - Research Center
615, rue du jardin botanique
54602 Villers-les-Nancy, France
{popicris|festor}@loria.fr

***Abstract*—We propose a distributed monitoring scheme for an ad-hoc/MESH network topology, and design a Distributed Hash Table based Peer-to-Peer system to log the topology information. A manager can rebuild a model of the network topology at any time in the past of its history by querying any node in the network. Merging and splitting of the monitoring overlays over time are supported by our protocol.**

I. Introduction

Monitoring the topology of a Wireless Mesh Network [1] over time is useful for robust network operation and helpful to network designers. Because, in a wireless environment, users manifest a good degree of mobility, keeping a time sensitive topology helps understand their mobility while connected. Fault management also benefits from the topology history, since detection of faults in the system is most often based on analyzing past events [11]. In multi-radio WMNs, keeping a historical topology, can lead to strategies for improving QoS (ie. throughput) by creating algorithms that assign channels to radios on a node [9]. It is therefore important to keep a history of the network topology in such a dynamic and unreliable environment as that of an ad-hoc network, in our case the "client area" of a hybrid WMN (fig. 1).

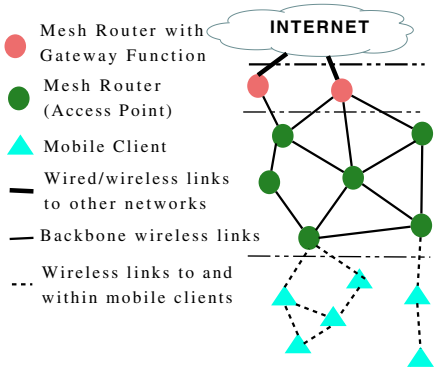

Fig. 1: Hybrid Wireless Mesh Network – abstract layered view

We propose a distributed topology monitoring framework to retain the topology information gathered over a predefined time period in the ad-hoc part of the network. A manager can query any node in the network for the neighborhood data relevant to a chosen number of time intervals to build up snapshots of the physical topology. There are however certain challenges that have to be surpassed, due to the nature of hybrid wireless mesh networks: failing/fluctuating links, or highly dynamic mobile users, resulting in frequent topology changes, and thus in an increase of topological information and signalling overhead. In view of these challenges, the requirements our solution must satisfy are: 1) the organization of the management plane must take into account the dynamic topology of the monitored system; solutions need to be provided for the case of network partitioning or merging; 2) the solution has to be robust to temporary link failure; for a certain time of disconnection, the monitored information should be retained in the disconnected component; 3) the management solution should not introduce overhead to user traffic in the network.

We use a structured peer-to-peer approach, by storing the monitored topology information in a Distributed Hash Table (DHT), to efficiently accommodate a large number of mobile nodes. Upgrades to Chord [13] and extended DHT storing support are needed to satisfy the requirements above: replication for the case of node failures, support for storing and lookup of time sensitive records in the DHT, and merging and decomposition of DHTs for underlying topology joining and splitting.

The paper is structured as follows: section II introduces related work. Section III presents the architecture of the distributed monitoring framework. The representation and collection of the topological data are shown in section IV. Section V describes the protocol between the cooperating nodes in the system to build and maintain the gathered topology information. An implementation of the solution is depicted in section VI, and the conclusions end the paper in section VII.

II. Related Work

DAMON [12] proposes a pervasive distributed monitoring architecture for ad-hoc networks. Agents running on active network nodes collect network information and send it to sinks in the network for storage. Because of reduced mobility, access points in a Wireless Mesh Network are perfect choices for monitoring the mobile client nodes, but this approach introduces single points of failure. We provide a fully distributed approach not relying on a single point of failure.

OLSR [4] can inherently provide the topology of the network, as partial link state is flooded by participating nodes. The problem of storing that information "intelligently" occurs however, as the same relatively important amount of obtained topology information is distributed on each node uselessly. Using a DHT approach to uniformly distribute the collected data with a minimum necessary replication degree is better suited for network functionality. Besides, link state routing protocols are not the de facto routing protocols for wireless multi-hop networks (see AODV [10], or HWMP [2]). The fact that there are multiple routing protocols for WMNs calls for a solution that is on top of the routing layer, and supports any routing protocols for ad-hoc networks.

We propose an overlay that leverages on Chord [13]. Some works [8] try to adapt the usage of DHTs for Mobile Ad-hoc Networks (MANETs), in order to reduce the overhead due to routing at the overlay layer. Dabek *et al.* in [5] and [6] present a layered application based on Chord to provide distributed file storage. The middle layer, DHash, performs block management, by assigning identifiers to blocks and storing them in the proper location.

In section I the possibility of network partitioning and merging has been taken into consideration, due to the dynamic character of the client nodes. Two models for DHT composition: the absorption and the gateway models have been proposed by Cheng *et al.* in [3]. A similar approach, but more suitable to merging the tables of two partitions is presented in [7]. It also offers a solution for fluctuating nodes and unstable links, by introducing recovery mechanism delays and marking unstable routes as invalid for DHT messages.

III. System architecture

Our monitoring solution is a layered one (fig. 2), with the *management application* using the services offered by the underlying DHT for storing/looking up the topology information. Each managed node is given the responsibility to collect neighborhood information (as presented in section IV).

The storage system is represented in figure 2 by the *Chord* and the *DHTopo* layers. *Chord* maps identifiers to nodes, builds and maintains the overlay routing tables, and looks up the nodes which are responsible for any given identifier. *DHTopo* uses the lookup service offered by Chord to find the node on which a topological record is (to be) stored. The role of DHTopo is to correctly store topological records pertaining to the same time slot for different nodes. A *merge* operation is defined to store records from multiple nodes under the same Chord key, the current time slot, t_i. The storage system provides the following services to the management application: topology records storage under the same key (for the same time slots), and topological records retrieval in one operation for one time slot.

The *management application* layer manages the time slots, organizes the neighborhood information into topological records for each time slot and adds the resulted records into the DHT using the storing service offered by the DHTopo layer. On the other hand it offers a **query service** to managers

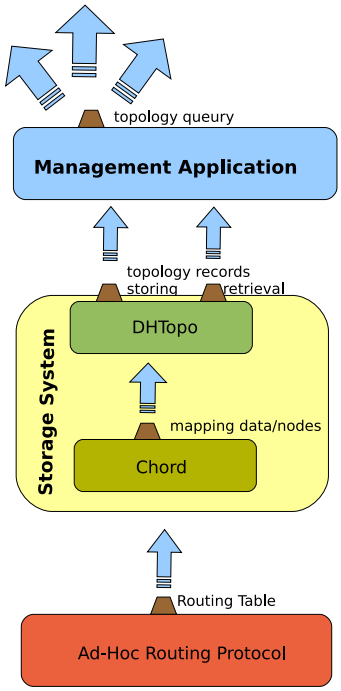

Fig. 2: Architecture of the management system

interested to rebuild a model of the network topology at any time in its history. The retrieval service of the DHTopo layer is used for getting a snapshot of the topology at any one slot with a single lookup operation in the Chord overlay. A query operation can be performed on any of the managed nodes.

IV. Data representation and collection

For node identification, we decided to choose the MAC address of the wireless interfaces present on each node, as unique identifiers. Each node runs an agent responsible for "listening" for beacons coming from neighboring nodes. The agents capture and interpret any beacon that reaches them, looking for the MAC address of the node that sent the beacon. We organize the neighborhood of any node n_i (the nodes n_i can overhear) as a list of MAC addresses: $n_i \rightarrow \{n_{i1}, n_{i2}, ..., n_{il_i}\}, \forall n_i \in \mathcal{N}$, where l_i defines the number of neighbors of node n_i.

Time slots

We divide the time into discrete time intervals, which we call *time slots*. For one time slot, which we choose to be of a reasonable length (ie. 5 seconds) to capture enough neighborhood information, and also allow a manager to observe the dynamics of the network, we infer the neighborhood of one node, and create a *topological record*, by associating the neighborhood information with the time slot number. The obtained topological records of every node, and for each time slot are stored into the DHT spanning all of the managed nodes. For the selected time interval length, we also had in mind the beacon period which is typically 100ms, therefore allowing a fairly high number of beacons to be collected

during a time slot. For all time slots t_i, the actions the agent on any node n_k, with $k = \overline{1, card(\mathcal{N})}$, takes are as follows. At the beginning of the time slot, it initializes its neighbor list; the first element that is present in this list for every time slot is the MAC address of the node itself. For the entire duration of the time slot, the agent listens for beacons, parses them, and, if the number of beacons received from one node is above a system defined threshold, it adds the MAC address of the node sending the beacons in the neighbor list; if the address is already in the list, it neglects any further beacon coming from that address for this time slot. At the end of the time slot, each node creates the topological record, by timestamping the obtained neighbor list with the current time slot, and adds it to the DHT after waiting a small random amount of time (to avoid overloading the node responsible for holding the topological records for this time slot).

Each node, at the end of a time slot, calls the primitive **put**(*timestamp*, *list*) (API offered by DHTopo). Since we are choosing time slots as keys because of their uniqueness, neighborhood data collected from different nodes for a certain time slot t_i, as described above, map to the same identifier, and are thus stored on the same node. DHTopo takes care that data coming from different nodes for the same time slot, is not overwritten (as would happen with basic Chord), but merged in a list

$$\{t_i, \{n_1, n_{11}, n_{12}, ..., n_{1l_1}\}, \{n_2, n_{21}, n_{22}, ..., n_{2l_2}\}, ...\} \quad (1)$$

and stored on the node whose identifier is the least greater than the key obtained from hashing t_i, and on a number of r successors from the node's successor list of size q, where $r \leq q$, as shown in figure 3.

We define a maximum number of time slots, T_N that can be retained by the system. We manage the T_N time slots in a First In First Out (FIFO) manner, with the oldest stored data the first to be replaced by freshly collected neighborhood data. For an ordered list of time slots t_0, t_1,..., t_{T_N}, the agent running on a managed node, a, will do the following:

- if the current time slot t_k is less than t_{T_N}, then push the tuple $\{t_k$, {a, neighborhood(a)}$\}$ in the DHT and retain a $latest\_slot$ equal to t_k; this will allow ordered replaying of the topological data (ie. $(latest\_slot, t_{T_N})$ followed by $[t_0, latest\_slot]$);
- if the current time slot t_k equals t_{T_N}, then let current slot be t_0 and push $\{t_0$, {a, neighborhood(a)}$\}$ in the DHT; to avoid adding fresh information to the same list containing old information (for the same time slot), at the beginning of t_{i-1}, every node checks to see if it is responsible for the topological record associated with the time slot t_i; if it is, then it voids the list from equation 1 of its content.

V. Initialization and maintenance of managed group

Since a Wireless Mesh Network router, R, is usually fixed in place, the formation of the DHT starts with the router (access point). It is the router's responsability to set up the DHT. As soon as this is done, R sends out periodically (every 1 second) a broadcast message with TTL=1, advertising the existence of a management organization. A TTL of 1 is sufficient to have all one-hop neighboring nodes participate in the DHT, since after joining, a node recursively behaves as R, thus extending the coverage of the DHT incrementally. The following is what happens when a random node not part of the DHT receives the advertisement from R. An advertisement broadcast message with TTL=1 is sent out by each node in the managed group. A node outside the managed group that eventually intercepts the advertisement accepts to join and sends an accept message to show that it is willing to participate in the DHT. The inviting node then initiates the joining procedure within the DHT: it hashes the IP of the invited node, looks up a successor for the new node and then sends back to the invited node both the successor in the Chord ring, and the tuple ($current\_slot$, $slot\_time\_remaining$). Finally the node joining the DHT computes the Round Trip Time (RTT) to the inviting node (ie. by doing a ping) and estimates the $actual\_slot\_time\_remaining = slot\_time\_remaining - RTT/2$; it now knows what the current time slot in the system is and it is also synchronized with the other nodes when changing it.

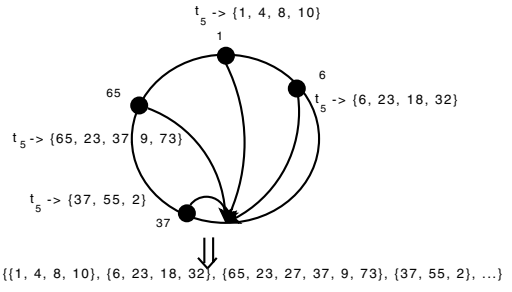

Fig. 3: Merging of collected neighborhood data into a single data list containing elements of the type $\{n_k, n_{k1}, n_{k2}, ..., n_{kl_k}\}$, where l_k represents the number of neighbors for node n_k; because the time slot, t_5 maps to the same node (node with id=37), the neighborhood data of nodes 1, 6, 37, 65,... will be stored on node 37 in a list that is formed by merging the individual neighbor lists

The solution is resilient to sudden node failures, due to the data replication mechanism present in the DHTopo layer: a $(t_i, neighborhood\_information)$ pair is stored on r successors of the node responsible for the key t_i; when the node responsible for t_i fails, the data will be available at its successor as it takes charge of the keys of its predecessor. The case of a group of nodes leaving (ie. network partitioning) is a special case of the individual node failure: the nodes in each of the two split DHTs retain their keys, and are assigned all keys that formerly belonged to the nodes in the other DHT.

The joining of two separate DHTs with the same function (of storing topological information) will not affect the topological records in the two merging groups, for the same time slot, because the *merge* operation defined in the DHTopo layer will merge the two lists of $node-neighbors$ pairs. This

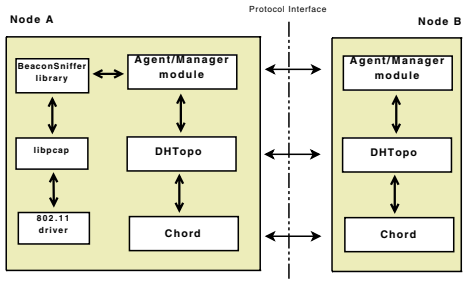

Fig. 4: Architecture of the management system; the interfaces exposed by each of the involved modules are shown

mechanism allows a manager to make a single query in one of the managed nodes after the merging of the two partitions, to get a single view of the topology of the network for a time slot corresponding to a moment in time when the network was partitioned.

VI. System Implementation

The monitoring system is made up of the following modules as shown in figure 4:

- the **Agent/Manager module** is responsible for initiating/joining a managed group (a DHT). It builds the list of neighbors with the neighbors' MACs received from the **BeaconSniffer** library, stamps it with the current time slot, and then calls *put(*$time\_stamp$*, data)* to add it to the DHT. UDP messages are used for time slot synchronization and for DHT advertisement and joining/merging.
- **DHTopo** is a middleware on top of Chord, that offers topology information management. It exposes a simple API to applications on top of it. *put(*key*, data)* takes key, hashes it into an m-bit Chord identifier, looks up the node responsible for this identifier and hands the data to that node for storing via an RPC call. *lookup(*key*)* hashes key and looks up the node which is responsible for the m-bit Chord identifier resulted; this is used to support nodes joining the DHT, when the node advertising the DHT and in communication with the guest node searches the successor of the new node. *get(*key*)* hashes key and returns the data stored on the node responsible for key.
- **Chord** offers a lookup mechanism, by mapping keys to nodes;
- the **BeaconSniffer** library filters the raw 802.11 MAC frames provided by libpcap to interpret the beacon frames. For every beacon it receives it sends the MAC address of the node broadcasting the beacon to the **agent/manager module**;
- **libpcap** is a packet sniffer library
- the **driver** of the wireless interface: support for running in monitoring mode, and multiple virtual interfaces is needed.

VII. Conclusion and Perspectives

A topology (self-)monitoring framework for Hybrid Wireless Mesh Networks based on Distributed Hash Tables was proposed. The system stores topological information gathered by the mobile clients roaming around mesh routers, in discrete intervals of time, which we called time slots. The snapshots of the topology taken every time slot are stored in a distributed fashion on every node in the system. The system can then be queried at any moment for the topology in any of the time slots in the history by one or multiple managers from any of the managed nodes.

For this we have chosen to base our monitoring and logging system on a DHT peer-to-peer system as it offers two important advantages to keeping historical topological records: the resiliency to node failures and, secondly, the scalability offered by Chord is an attractive incentive for an ad-hoc/hybrid mesh environment with an important number of nodes involved. We have proposed a solution to temporary partitioning of the network, that allows a topology build-up of two separate components, after their merging.

Acknowledgement

This paper was partially supported by the French Ministry of Research project, AIRNET, under contract ANR-05-RNRT-012-01.

References

[1] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. Wireless Mesh Networks: a survey. *Computer Networks*, 47(4):445–487, March 2005.

[2] Michael Bahr. Proposed routing for IEEE 802.11s WLAN mesh networks. In *WICON '06*, page 5, New York, NY, USA, 2006. ACM Press.

[3] Lawrence Cheng, Roel Ocampo, Kerry Jean, Alex Galis, Casba Simon, Rbert Szab, Peter Kersch, and Raffaele Giaffreda. Towards Distributed Hash Tables (De)Composition in Ambient Networks. In *DSOM*, volume 4269, pages 258–268. Springer, 2006.

[4] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC Experimental 3626, Internet Engineering Task Force, Oct 2003.

[5] Frank Dabek, Emma Brunskill, M. Frans Kaashoek, David Karger, Robert Morris, Ion Stoica, and Hari Balakrishnan. Building Peer-to-Peer Systems With Chord, a Distributed Lookup Service. In *HotOS-VIII*, Schloss Elmau, Germany, 2001. IEEE Computer Society.

[6] Frank Dabek, M. Frans Kaashoek, David Karger, Robert Morris, and Ion Stoica. Wide-area cooperative storage with CFS. In *SOSP '01*, pages 202–215. ACM Press, 2001.

[7] Tobias Heer, Stefan Gotz, Simon Rieche, and Klaus Wehrle. Adapting Distributed Hash Tables for Mobile Ad Hoc Networks. In *PERCOMW '06*, page 173. IEEE Computer Society, 2006.

[8] Olaf Landsiedel, Stefan Götz, and Klaus Wehrle. A churn and mobility resistant approach for DHTs. In *MobiShare '06*, pages 42–47, New York, NY, USA, 2006. ACM Press.

[9] M.K. Marina and S.R. Das. A topology control approach for utilizing multiple channels in multi-radio wireless mesh networks. In *2nd International Conference on Broadband Networks*, volume 1, pages 381–390, Oct. 2005.

[10] Charles E. Perkins, Elizabeth M. Belding-Royer, and Samir R. Das. Ad hoc on-demand distance vector (AODV) routing. RFC Experimental 3561, Internet Engineering Task Force, July 2003.

[11] Lili Qiu, Paramvir Bahl, Ananth Rao, and Lidong Zhou. Troubleshooting wireless mesh networks. *SIGCOMM Comput. Commun. Rev.*, 36(5):17–28, 2006.

[12] K.N. Ramachandran, E.M. Belding-Royer, and K.C. AImeroth. Damon: a distributed architecture for monitoring multi-hop mobile networks. In *IEEE SECON 2004*, pages 601–609.

[13] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM '01*, New York, NY, USA, 2001. ACM Press.