MeshMan: A Management Framework for Wireless Mesh Networks

Vivek Aseeja and Rong Zheng
Department of Computer Science, University of Houston
Email: {*vivekian,rzheng*}*@cs.uh.edu*

***Abstract*— As wireless mesh networks become more popular, there exists a need to provide centralized management solutions, which facilitate network administrators to control, troubleshoot and collect statistics from their networks. Managing wireless mesh networks poses unique challenges due to limited bandwidth resources and dynamic channel quality. A robust management solution should function despite network layer failure. In this paper, we propose *MeshMan*, a network layer agnostic, low overhead solution to network management to cope with unreliable wireless channels, link and network level dynamics in wireless mesh networks. It combines the concepts of source routing with hierarchical addressing, and provides a native efficient query interface. A prototype of MeshMan has been implemented as a user space daemon on Linux and evaluated using a 12-node wireless mesh network testbed. Experimental studies demonstrate that MeshMan has comparable or better performance than the Simple Network Management Protocol (SNMP) in management overhead and response times when the network is stable while having much better performance in presence network dynamics.**

I. Introduction

Wireless Mesh Networks are multihop static wireless networks where each node can connect to every other node via multiple hops. They have become popular due to their low cost and ability to provide last mile connectivity. They hold great potentials to provide a communication infrastructure in areas where it is not financially viable to layout a wired infrastructure such as developing countries or rural communities [1]. Wireless mesh networks are also used to share broadband Internet access and provide low cost Internet and local access to wireless users in housing communities. As wireless mesh networks become more ubiquitous, it is imperative to provide a management solution, which can be utilized by network administrators to easily operate and trouble-shoot these networks. Moreover, the research community can benefit from such a solution that would allow them to remotely collect data and statistics from experimental mesh testbeds.

A straightforward solution to managing wireless mesh networks would be to extend the Simple Network Management Protocol (SNMP) [4]. SNMP is a widely used *application layer* protocol, which is designed to work over *heterogeneous* link layer technologies where network bandwidth and storage space are relatively abundant. For instance, the size of poll messages for one metric to a single interface using SNMP GET request can be up to 1 KB. A wireless mesh network on the other hand is typically operated by a single service provider and uses a common link layer protocol (e.g., IEEE 802.11). Mesh routers are generally embedded devices with limited storage capacities. Running SNMP in mesh networks can be both heavyweight in bandwidth consumption [4], [5] and ineffectual if the network layer has not been configured properly as SNMP relies on a functioning network layer to reach nodes that are more than one hop away [6]. It has been reported in [20] that routing flaps and instability exist in mesh network deployments. We have also observed temporary unavailability of nodes due to misconfiguration and/or slow convergence of routing table in our own mesh testbed. In the case of network-level disconnection of a physically connected network, node state information such as routing table entries becomes unavailable when it is most needed (for trouble shooting purposes). Therefore, the following design constraints necessitate an alternative solution:

- **Robustness to network layer failure:** The solution needs to be independent of network layer functionalities for the purpose of diagnosing and correcting network layer faults.
- **Self-reconfigurability**: Failure of nodes and addition of new nodes should be handled gracefully.

Motivated by the afore-mentioned design constraints, we propose *MeshMan*, a network layer agnostic, light-weight solution to network management to cope with unreliable wireless channels and network dynamics in wireless mesh networks. MeshMan does not rely on IP or a functioning network layer. In MeshMan, each mesh router is identified by a *mesh ID*, which is automatically configured in a hierarchical fashion. Such mesh IDs are stored at a central location, the *mesh manager*, and serve as routing directives for queries generated from the mesh manager to mesh routers. Unicast queries are routed using source routes encoded in the mesh ID and delivered over link layer, which eliminates the dependency on mesh routing protocols in use. Broadcast queries are flooded in the network for maximal resilience to packet losses. In events of missing or mismatched information at the mesh manager, MeshMan provides ARP-style resolution over multiple hops for mappings between MAC addresses and mesh IDs as fail-safe. A lightweight query-response mechanism has been designed in place of SNMP with simplified encoding rules. In MeshMan, addressing and routing are decoupled from the query-response mechanism. Therefore, one can easily extend and integrate it with conventional network management solutions such as SNMP across heterogeneous networks. Alternatively, mesh

978-1-4244-3487-9/09/$25.00

manager can serve as a gateway to translate SNMP queries to native queries.

A prototype of MeshMan has been developed as a collection of user space daemons. Evaluations are carried out on a 12-node indoor wireless mesh testbed. We ported our implementation to the OpenWRT Linux distribution [12] on the embedded Wireless Router Application Platform (WRAP) board. The optimal link state routing protocol (OLSR) [8] is used as the default mesh network routing protocol for baseline comparison. We compare the query response time, query loss ratio and recovery time upon node join and leave across various schemes. Our experimental studies demonstrate that MeshMan has comparable or better performance in both management overhead, response times than SNMP; and is shown to be resilient to network dynamics.

The rest of the paper is organized as follows. In Section II, an overview of existing network management systems is presented. We highlight the key design considerations and features that distinguish MeshMan from related work. The design of MeshMan is described in Section 3 with the implementation details in Section 4. We present the testbed setup and experiment results in Section 5. Finally, we conclude the paper in Section 6.

II. Background and Related Work

Networks are formed using disparate hardware and software components that interact together to provide end-to-end data transport. As networks grow in size and complexity, the probability of failure in network components increase. As a result, there is a great need for network management solutions, which are a combination of various tools, protocols, and techniques to help a network administrator to manage various devices [7]. A network management system generally consists of several components:

- **Managing entity**: an application which is running in a centralized network management station for controlling the collection, processing, analysis and presentation of the network management information which has been collected [7].
- **Managed device and agent**: This refers to the network component which is being managed by the managing entity. The agent which is generally a daemon runs in the background collecting statistics and is responsible for transferring information back to the central management entity.
- **Network management protocol**: This protocol is the communication medium between the managing entity and the managed entity. It defines the set of rules which both follow to communicate with each other.

The simple network management protocol (SNMP) is an application layer protocol that facilitates the exchange of management information between network devices. It defines both a protocol for communication of queries and responses and also an agent for sending responses within a network [7]. SNMP operates on top of the UDP or TCP transport protocol and relies on a functioning IP layer to route information. In the Internet, the Internet Control Message Protocol (ICMP) provides a mechanism to send error messages when services on routers and hosts are not available.

Chen *et. al* [9] proposed the Ad-hoc Network Management Protocol (ANMP), a protocol for managing mobile wireless ad hoc networks. The design objective of ANMP is to handle mobility and power constraints for mobile nodes. The protocol uses hierarchical clustering of nodes to reduce the number of messages exchanged between the manager and the agents (mobiles). Clustering also enables the network to keep track of mobiles as they roam. ANMP is compatible with SNMP and can be integrated with existing network management solutions. Although ANMP is similar to our work since both operate on multihop wireless networks, our work focuses on providing a robust and lightweight management solution to static multi hop wireless networks, where power management and mobility are non-issues. Another work in this area is Nucleus [18], which is a network management system developed by Tolle *et. al* for wireless sensor networks. Nucleus can retrieve the values of user-defined attributes and RAM variables from nodes. In Nucleus, dissemination of query message is carried out using unrestricted flooding. The flooding messages generate a single sink tree for collection of management state. Such a sink tree is regenerated next time a query message is flooded and thus is resilient to network dynamics to certain extent. MeshMan differs from Nucleus in two key aspects. First, in addition to broadcast queries, it also supports unicast queries. This necessitates an addressing and unicast routing scheme. Broadcast queries incur much network traffic and should be avoided if possible. Second, Nucleus reconstructs the sink tree each time a query is initiated. MeshMan utilizes local update based on hop distance to maintain routes to the mesh manager. It is expected that though link quality may vary, physical connectivity is likely to be more stable in wireless mesh networks.

Two seemingly conflicting considerations motivate our design of MeshMan in wireless mesh networks. On one hand, wireless mesh networks exhibit intricacies such as volatility of wireless channels, dynamic routes and limited bandwidth, which are not present in wired networks. A management solution should be *customized* to handle these complexities. On the other hand, there is a need for wireless mesh networks to inter-operate with wired networks to provide access to wide-area network connectivity. It is thus desirable to have a uniform management interface for an ISP to operate both its wireless and wired infrastructure. To accommodate both requirements, in our design of MeshMan, the query interfaces are decoupled from the mechanisms used to route management information, and thus allow interoperability with other management solutions. For routing of management information in mesh networks, we design a solution that is tailored to handle the intricacies.

III. Design of MeshMan

Similar to other network management systems, MeshMan consists of a managing entity, called *mesh manager* running

on a central server, managed devices and agent running on each mesh node, and a set of network management protocols. The management protocols in MeshMan not only defines the set of rules that the mesh manager and managed devices follow to communicate with each other, it also specifies an addressing scheme for the auto-configuration of mesh IDs, the associated adoption protocol and a source routing mechanism. As stated earlier, this is one key difference between MeshMan and existing management solutions.

A. Hierarchical addressing

To query management states on individual mesh nodes, traditional ad hoc routing protocols require maintenance of routing information proactively or reactively on all nodes. The MAC and IP address spaces of mesh nodes are flat and cannot be aggregated easily. This leads to significant message and space complexity to determine and store the routing information at the intermediaries, and does not scale well [19]. We observe that for management purpose, it is sufficient for the mesh manager to know how it can reach each mesh node and vice versa. Intermediate mesh nodes do not need to keep routing information to one another. Therefore, we choose to use source routing for unicast queries and maintain states at the mesh manager only. This also simplifies the implementation of managed agents on mesh nodes.

In MeshMan, each node is assigned a mesh ID of variable length in the dot decimal representation. Starting from the mesh manager with mesh ID 1, the number of fields of the mesh ID reflects the distance to the mesh manager in a tree rooted at it. The mesh ID of a child is obtained by appending an additional field to the mesh ID of the parent node. Therefore, all children with a common parent share the same prefix. As an example, the children of node 1.2 are assigned mesh ID of 1.2.1, 1.2.2 etc. Allocation of the mesh IDs is done using the adoption protocol presented in Section III-B. The mesh manager maintains the (*mesh ID*, *MAC address*) mapping for all active nodes in the network.

One key advantage of the proposed hierarchical addressing scheme is that it naturally facilitates address aggregation. A mesh ID encodes the source route to the corresponding node, and serves as routing directive. For instance, to query a node with mesh ID 1.2.1, the route taken is via node 1, node 1.2. By limiting the number of child nodes a parent node can have, we can bound the length of each address field (e.g., 1 byte in our implementation). In comparison with a naïve source routing scheme that concatenates 48-bit MAC addresses of nodes along a path, mesh ID representation is more compact and incurs less overhead. Multiple mesh managers can be supported if the address space is partitioned such that a separate address is used for different mesh managers. For ease of presentation, we assume in the rest of the paper, there is only one mesh manager.

B. Adoption protocol

In this section, we describe the adoption protocol to allocate mesh IDs to nodes in a fully distributed manner.

1) The core algorithm: The adoption process starts from the mesh manager and propagates toward the peripheries of the network. A node is both a "client" and a "server" in that it can assign addresses to its child nodes once it is allocated an ID by its parent.

A node periodically broadcasts a Discovery frame announcing its path cost to the mesh manager. A node chooses the node among its neighbors that has the least cost path to the mesh manager and sends back a mesh ID Request to the potential parent. The parent responds with a mesh ID Offer to the child and the adoption process is complete. If a node has a parent, and receives an offer from another neighbor in its broadcast domain, it checks if the offer contains a less costly path to the mesh manager. If so, the node abandons its previous parent and chooses the new parent. Link metrics such as Expected Transmission Count (ETX) [16] and Weighted Cumulative ETT (WCETT) [17], can be used to choose routes based on the quality and the bandwidth of the link. They were proved to be effective as compared to hop counts for choosing higher throughput routes. However, routing flapping and instability may occur if the threshold values are not tuned properly as reported in [20]. In MeshMan, management traffic is loss resilient and light in bandwidth consumption. We choose to use hop counts instead as the link metric for route selection so as to maintain stability and minimize variability in routes. This also helps in reducing the traffic incurred to update the (*mesh ID*, *MAC address*) mapping at the mesh manager which occurs when a new route is added or deleted. Measurements from a wireless mesh testbed also confirm our hypothesis as detailed in Section V-B.1.

2) Handling packet losses: Since the protocol runs over error-prone wireless channels, it is essential to account for the loss of any frame involved in the adoption process. One possible solution is to use soft state and have a time out mechanism for each Request or Offer sent. Expiration of the timer triggers retransmissions of the corresponding frames. This is the mechanism adopted in DHCP [10]. However, it adds additional complexity to maintain soft states at mesh nodes. The design of our adoption protocol is inherently robust to withstand frame losses due to the periodicity of Discovery frames. Consider for instance the loss of Request Frames. In this case, the child node does not receive an Offer frame, which would have been sent otherwise. The node sends a Request frame again the next time a Discovery frame is received from the potential parent. In the case that the Offer frame is lost, the parent has added the node as its child, but the child is not aware of it. The child node will send a Request frame again to the parent upon reception of the next Discovery frame. The parent simply retrieves the existing mesh ID and offers it to the child.

3) Managing node failure: Since Discovery frames are transmitted periodically, they also act as Keep Alive messages. Upon reception of a Discovery frame a node updates the timestamp. Node failure or disconnection is detected using the timeouts. If the timestamp associated with the current parent becomes stale, a node can request a new mesh ID from a

different neighbor. Similarly, a parent removes a child and informs the mesh manager if the child is not active for a prolonged period of time.

C. Routing of management traffic

There are two types of queries originating from the mesh manager, namely, broadcast queries and unicast queries. Broadcast queries are delivered to all mesh nodes using flooding. Techniques such as random initial back-off can be adopted to suppress redundant broadcast transmissions. Unicast queries originate from the mesh manager and are destined to a specific MAC address. The mesh manager first looks up the mesh ID and then routes the query hop-by-hop using the source route information included in the corresponding mesh ID. In the unicast queries, the MAC address information is included as the destination address.

Query responses are delivered to the mesh manager hop-by-hop. A node sends the response to its parent, which forwards the message onwards. We do not enforce reliability in the delivery of query responses since queries can be retransmitted if no response has been received upon timeouts at the mesh manager. This requires additional bookkeeping at the mesh manager but simplifies the implementation of mesh nodes.

1) Failure handling: Due to the lack of end-to-end reliability, update messages that carry the (*mesh ID, MAC address*) mapping can be lost. Several failure scenarios may arise. First, the mesh manager does not have the record of a particular MAC address. This may occur when the update message was lost when the node first joined the network. Second, a mesh manager may have incorrect mapping. For example, when a node's parent fails, it requests a mesh ID from a different node and updates the mesh manager its latest binding. If the message is lost, the mesh manager still has the old binding and will wrongly route the message to a non-existing node. Lastly, a mesh manager may have out-dated information about a failed node.

When the mesh manager has incorrect information about a failed node or a node whose mesh ID changes, query messages with the old mesh ID would eventually arrive at a node where the source route and/or the MAC address in the destination field is invalid. This node is responsible to generate an error message including the header portion of the query message informing the mesh manager. Upon reception of the error message, the mesh manager deletes the corresponding entry.

When the mesh manager cannot find a mesh ID entry for a MAC address, we utilize the broadcast query to resolve the MAC address by asking "who has MAC address xx?" This is similar to the Address Resolution Protocol (ARP) with the difference that the query is propagated via multiple hops. Frequency of broadcast queries is rate limited to avoid excessive traffic in the network.

IV. Implementation

MeshMan is implemented as user space daemons in Linux. This allows fast prototyping and portability to other POSIX-based operating systems.

A. System architecture

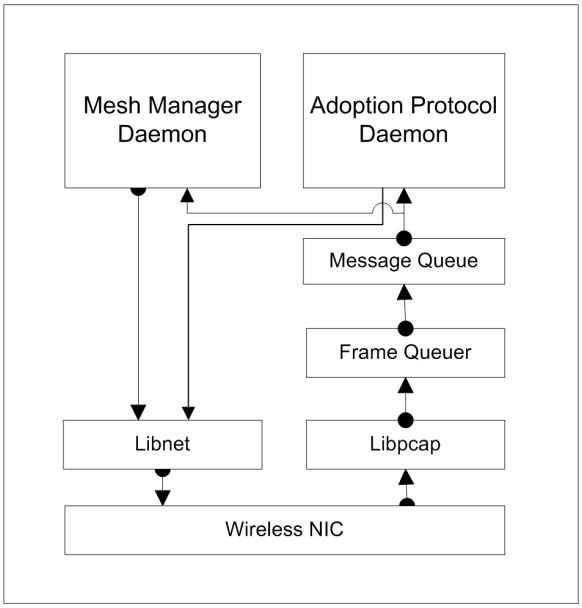

Fig. 1. MeshMan system architecture (colored blocks are modules implemented in MeshMan)

Fig. 1 shows how the different components of the system fit together. The framework consists of the following modules:

- *Adoption Protocol Daemon (APD)*: This process runs on mesh node. There are two sub-components in this module, i.e., the Adoption Layer implementation and the routing module, which is responsible for routing queries and responses.
- *Mesh Manage*r Daemon *(MMD)*: This daemon acts as the managing entity that maintains MAC address to mesh ID mapping. Update messages from mesh nodes are delivered to this daemon.
- *Frame Queuer Daemon*: This process is responsible for demultiplexing frames that are destined for MMD or APD from the kernel space and placing them in a message queue in user space based on the message type.
- *Queryd*: Queryd is an administrative tool that takes user input and serializes queries to the specified MAC addresses or broadcast addresses and process responses.

The Libnet (Libpcap) [11] open source library is used in our implementation to inject (capture) packets to (from) the network interface card bypassing the TCP/IP network stack.

B. MeshMan message format

The structure of MeshMan message format is similar to ICMP message. It consists of a common header and a message body specific to each type of messages. The common header (IV-C) contains 4 fields, i.e., *version*, *type*, *code* and *length*. The first field version indicates the version of the protocol. The byte long *type* field broadly defines the frame type for different modules: i) *AP* for Adoption Protocol, ii) *Route* for the Routing component and iii) *MM* for MMD. The two byte

code field further identifies the "subtype" of message within each MeshMan message Type value.

C. Adoption Protocol Daemon

Implementation of the APD is summarized in Fig. 2. Messages arriving at a node are placed in the message queue by Frame Queuer. Frames destined for the Adoption Protocol are picked up by the *FrameDemultiplexer* module. The *FrameDemultiplexer* module passes the frame to the *FrameParser*, which decodes the frame and classifies it as per the *type* and *code* field in the common header. The frames are then demultiplexed to the Adoption Protocol or the Routing submodule. It is then handled by the *FrameProcessor, which* makes the algorithmic decision of what needs to be done upon reception of the frames. For instance, an Offer needs to be sent in response to a *Request*; or a query needs to be forwarded if it is not the final destination.

For messages leaving the node, the *FrameConstructor* module is responsible for constructing responses, notifications, offers and discovery messages, and injecting them into the network stack using the Libnet library [11]. Since mesh IDs are of variable length, they are encoded using a definite-form scheme where the length of the mesh ID is placed before the mesh ID.

The *ParentManager* maintains the parent of a node. The *ChildManager* keeps track of the adopted children. It has three tasks, i.e., to grant relative mesh IDs to new children, to clean up dead children and to translate relative mesh IDs (i.e., excluding the prefix of the parent) to MAC addresses. To accomplish this, the *ChildManager* maintains a list of the children MAC addresses that are associated with their relative mesh IDs and the timestamp of the last update. In addition, it also maintains a "dead children" list, which are found during clean up. Before a new child is added, it is checked if the MAC address already exists in the list. In the case that it exists, the timestamp is updated and the relative mesh ID is returned. In the case where the MAC address is new, the dead children list is first checked for an available mesh ID. Otherwise, a new mesh ID is generated by incrementing the mesh ID counter and allocated to the child.

The *RouteManager* routes queries to the destination nodes and responses back to the mesh manager. Upon reception of a query, the destination MAC is compared with the node's MAC address. When a match is found, the query is processed and the response is returned. To forward queries, *ChildManager* is called to resolve the relative mesh ID to the next-hop MAC address and the frame is forwarded. In a similar manner, responses are forwarded to the mesh manager using the parent's MAC address provided by *ParentManager*.

D. Mesh manager daemon

The MMD as the managing entity has the sole important task of maintaining source routes from itself to every node in the network. To accomplish this, MMD keeps a hash table which maps the MAC address of a node to the corresponding mesh ID. In addition, it stores pointers from parents to child

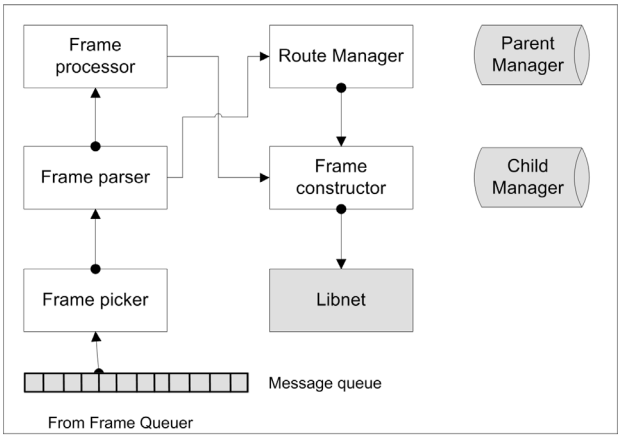

Fig. 2. Implementation of the adoption protocol daemon

nodes. This corresponds to a tree rooted at the mesh manager. When a node is allocated a mesh ID, its APD sends an AddRoute notification to the MMD. The MMD checks if the MAC Address exists. It updates the routes, adding a forward pointer from the parent to the new child. When a parent determines a child fails or moves away, its APD sends a DeleteRoute notification to MMD. The MMD deletes the child entry and its subsequent children in the tree using the Depth First Search algorithm.

If the MMD fails to find a mesh ID for a given MAC address for query messages, it sends out a broadcast query message requesting such information.

E. Queryd

Queryd is a simple tool for administrators to send out queries to an agent as well as return the responses in human readable forms. It uses the MAC address and the query as the input. The Linux /proc file system allows direct access and modification of kernel memory variables. We utilize the /proc file system as the management information base (MIB). Hence, the query corresponds to an in-memory value to be read from the remote system via the agent. For example, the following command fetches the routing table from a remote agent with the MAC address 00:FF:23:AB:11:22.

00:FF:23:AB:11:22 /proc/net/route

Serialization of query and response messages from/to users is handled by the MMD automatically.

V. Evaluation

We have carried out extensive evaluations of the proposed management framework on our wireless mesh network testbed. Large-scale simulations in Qualnet are omitted due to space limits.

A. Testbed setup

A 12-node wireless mesh testbed has been setup on the 2^{nd} and 3^{rd} floors of the Phillip G Hoffman building at the University of Houston to run wireless experiments. Fig. 3

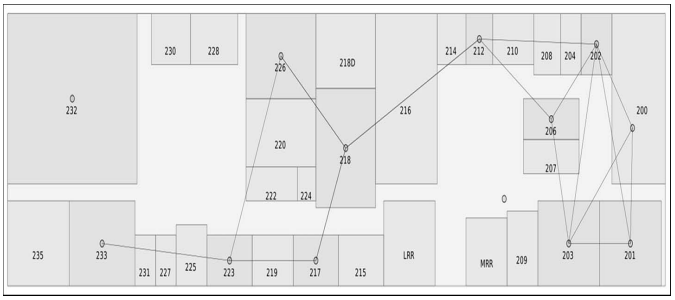

Fig. 3. The WiSeR Mesh Testbed at the University of Houston. Dots are mesh nodes. Blue lines indicate the existence of links among mesh nodes reported by OLSR

shows a snap shot of the real time connectivity map of the testbed. Each node is an embedded Wireless Router Application Platform (WRAP) board with 233 MHz AMD Geode SC1100 CPU, 64Mb DRAM, with dual Mini PCI Atheros 802.11a/b/g Wireless Cards and one Ethernet port.

We have ported the OpenWRT Linux distribution on the nodes. OpenWRT is a Linux distribution based on the buildroot system and designed to be deployed on wireless embedded devices [12]. Although each node has 2 wireless mini PCI cards, we used only one radio to run the experiment sets. The second radio is put to promiscuous mode to sniff ongoing transmissions for collection of traffic statistics. Each node in the test bed has an Ethernet connection acting as the control pipe to issue experiment commands from a central server and for data collection and acquisition.

We chose the Optimal Link State Routing (OLSR) protocol as the routing protocol for SNMP queries. The Net-SNMP [13] implementation of the Simple Network Management Protocol was selected as the management tool for baseline comparisons with MeshMan. To provide fair and accurate measurements, we used the RDTSC assembly instruction, which is a mnemonic to read the timestamp counter on Intel x86 processors [14]. The instruction returns a 64 bit value that represents the count of ticks from the processor reset. To gain a better understanding and dissect the time measurements, we instrumented our implementations and the Net-SNMP implementation with the RDTSC instruction at the appropriate code locations.

The parameters used in the experiments are summarized in Table I.

TABLE I
COMPONENTS OF THE ADOPTION PROTOCOL DAEMON

Parameters	Value
Frequency of Discovery Message	1 second
Parent Time Out	3 seconds
MMD Mapping Time out	5 unanswered queries

B. Experimental results

We measure the query-response time, traffic overhead, and reconfiguration time in presence of network dynamics under different schemes. For comparison, we have implemented/ported the following schemes:

- MeshMan routing and native query (MM-NQ)
- OLSR Routing and native query (OLSR-NQ)
- OLSR routing and SNMP query (OLSR-SNMP)

In the case of MeshMan, we experiment with both unicast queries and broadcast queries. Broadcast queries are not directly supported with OLSR on OpenWRT. It should be noted that the key advantage of MeshMan lies its robustness and flexibility, i.e., the ability in handling network level outage and node dynamics. Therefore, it is crucial to understand whether the robustness comes at the cost of degraded performance. All results presented are average of ten experimental runs.

1) Bootstrap time and convergence: In this set of experiments, we investigate the time it takes to reach a stable state when the network is first booted up. In particular, for MeshMan, during the bootstrap phase, mesh IDs are allocated starting from the mesh manager. Upon assignment of a mesh ID, the node will report the information to the mesh manager resulting in an update. In the case of OLSR, routing table entries are created and updated when link state information is collected or changed.

Fig. 4 shows the number of updates in every 10s period at the mesh manager and the gateway node. We see that after 10s, the number of updates with MeshMan quickly diminishes. The mesh IDs are stabilized. In comparison, the route entries keep changing during the first 100s at the gateway node in OLSR. In part, the fast convergence in MeshMan can be explained by the use of hop distance as link metrics. Even in presence of channel variation, hop distance remains constant most of the time.

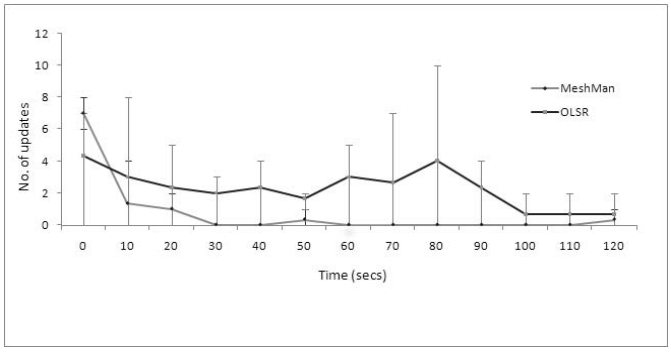

Fig. 4. Initial setup time (Number of updates in each 10s interval over time)

2) Query response time: In this set of experiments, we investigate the query response time under different schemes. With MeshMan, we evaluate the efficiency of both broadcast query and unicast query. In broadcast query, queries with broadcast mesh ID are flooded in the network. All nodes will respond to a broadcast query. SNMP does not support broadcast query directly. It is up to the network layer implementation for supports of IP broadcast/multicast, which is yet available in OLSR.

There are two types of queries, namely, single queries which request a single record of information; and bulk queries which request a collection of information. In SNMP, bulk queries

can be issued with multiple GET requests in serial or a single BULK GET. The later is more efficient.

Fig. 5 and Fig. 6 show the query response time for single queries for /proc/sys/kernel/hostname and bulk query for /proc/net/route. The total response time can be split in the following parts: i) *processing time* to generate a request at the managing entity, ii) *network delay* from the manager to an agent, iii) *processing time* at an agent to interpret message and generate response, iv) *network delay* from an agent to the manager, and v) *processing time* at manager to receive and interpret response.

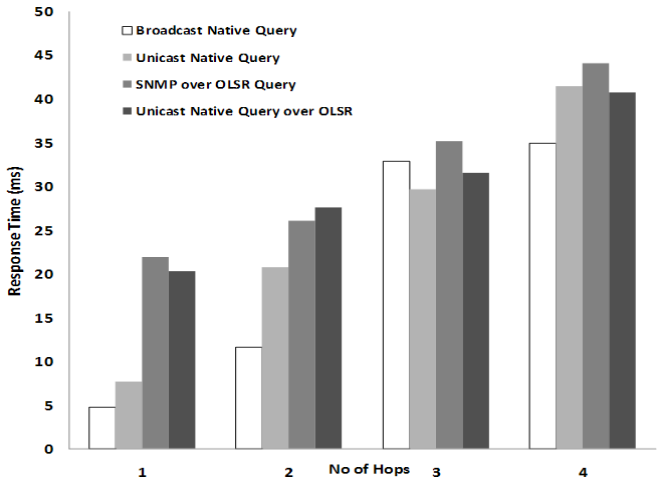

Fig. 5. Response time of single queries /proc/sys/kernel/hostname

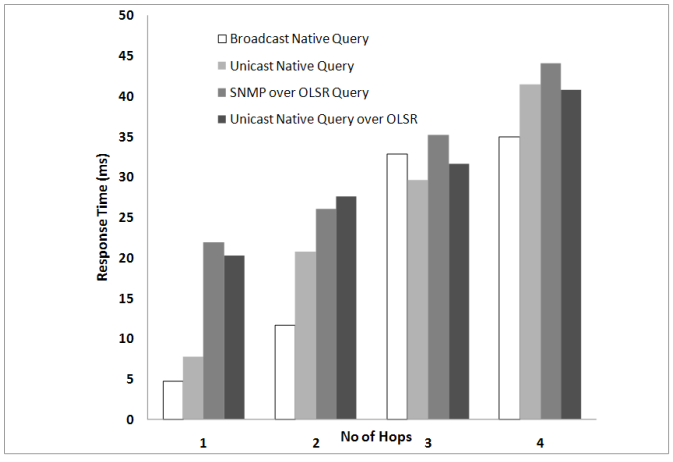

Fig. 6. Response time of bulk queries /proc/net/route

We see that the processing time at the managing entity and agents for single queries is quite significant as shown by the difference between OLSR-NQ and MM-NQ. In both cases, native queries are used. OLSR-NQ incurs slightly less response time compared to OLSR-SNMP as a result of more efficient encoding in the native query. Interestingly, compared to single queries, broadcast queries incur less response time even though the response traffic load is higher. This is because broadcast messages are not retransmitted in events of packet losses in the IEEE 802.11. Among all schemes, MM-NQ performs the best. Similar observation can be made for bulk queries (Fig. 6). However, the difference in response time is less pronounced. In this case, the total response time is dominated by network delay. The effect of processing time is not as significant. With broadcast queries, larger amount of traffic is generated resulting higher network delay.

3) Management overhead: To measure the network overhead we use tcpdump at the mesh manager to capture network activity, and the tcptrace tool [15] to estimate bandwidth consumption by the management applications.

Fig. 7 and Fig. 8 show the amount of management traffic in different schemes. For single queries (Figure 7), we see that SNMP incurs higher management overhead compared to other schemes. This is consistent with the response time measurements. Again, we observe that native queries are generally more efficient. For bulk queries (Fig. 8), the difference is less pronounced. As the query interval increases, the management overhead decreases roughly linearly.

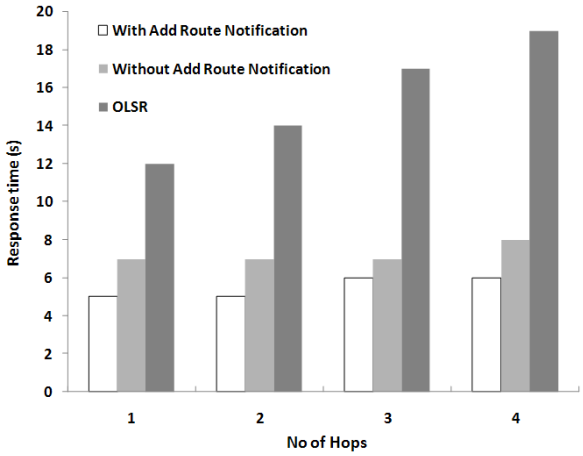

Fig. 7. Management traffic overhead vs unicast query interval

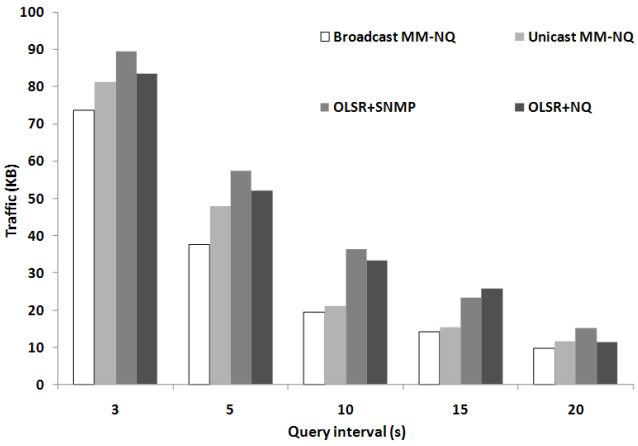

Fig. 8. Management traffic overhead vs broadcast query interval

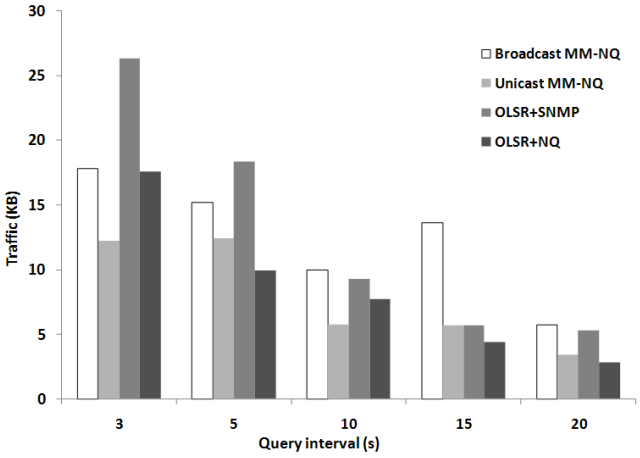

Fig. 9. Recovery time from node dynamics

4) Recovery from network dynamics: Finally, we evaluate how MeshMan reacts to network dynamics. In particular, three set of experiments are carried out:

- *With Add Route Notification*: The Mesh Manager keeps querying a particular node (node A) at an interval of 1s. We bring node A's parent down and wait for the node to be adopted by a sibling, which sends an *Add Route Notification* message to the Mesh Manager. We measure the time difference between the last successful query before nodes A's parent is down to the first successful query after that.
- *Without Add Route Notification*: In this case, the Add Route Notification from node A's new parent is turned off in order to simulate losses of *Add Route Notification*. In this case, when the Mesh Manager cannot reach a node, after five unanswered queries, it broadcasts a mesh ID mapping query to each node. If the end node has the requested MAC Address, it replies with its own mesh ID.
- *OLSR*: Here, we bring down a node and see how quickly OLSR adapts to such a scenario. We send SNMP queries to node A and bring down its parent (node B). Eventually as OLSR picks up an alternate path to node A, it starts responding to queries. We measure the time difference between two successful queries when node B is brought down. To have a fine-grained measurement, we changed the timeout value in SNMP to 1 second, which is comparable with the query interval in MeshMan.

We vary the hop distance from the node in question (i.e., node A). The results are shown in Fig. 9. We see that with *Add Route Notification*, the time to recover from a single node failure is the minimum. However, without Add Route Notification, broadcast query can still successfully determine the mesh ID and MAC address binding within a short period of time. OLSR, on the hand, takes longer to propagate the up-to-date routing information. This implies with MeshMan, we can potentially take a reactive approach to network dynamics, where the mesh manager queries (and caches) nodes' mesh IDs only when user query occurs.

VI. Conclusion

In this paper, we have designed, implemented and evaluated MeshMan, a management framework for wireless mesh networks. A management solution in wireless mesh networks needs to be designed to handle network dynamics, variation in link qualities, and node addition or failure with low overhead. MeshMan provides an efficient and robust way to retrieve information from mesh networks. Its most salient feature is the tolerance to network level failures. MeshMan has comparable or better performance compared to its counterpart - SNMP in terms of response time, and management overhead. We plan to extend it to provide configuration of network devices and port our implementation to kernel space on Linux platforms. Currently, MeshMan only supports a single logical mesh manager. Multiple mesh managers can be employed at the network edge to improve fault tolerance and scalability using anycast.

References

[1] J. Bicket, D. Aguayo, S. Biswas, and R. Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *Proceedings of ACM MOBICOM*, 2005.
[2] Fire Tide Networks. An Introduction To Wireless Mesh Networking, White Paper. *http://www.firetide.com*.
[3] Lili Qiu, Paramvir Bahl, Ananth Rao, Lidong Zhou. Troubleshooting wireless mesh networks. *Computer Communication Review* 36(5): 17-28, 2006.
[4] Julia Kantorovitch, Zach D Shelby, Tommi Saarinen , Petri Mähönen. Wireless SNMP, Technical Paper. University of Oulu, Finland, 2001
[5] R.Sprenkels, J.P.Martin-Flatin. Bulk Transfers of MIB Data. *The Simple Times*, 7(1):1-7, 1999
[6] S. Marti, T. Giuli, K. Lai, and M. Baker. Mitigating routing misbehavior in mobile ad hoc networks. In *Proceedings of ACM MOBICOM*, Boston, MA, Aug. 2000.
[7] Douglas Mauro, Kevin Schmidt. Essential SNMP (2nd Edition). O' Reilly Media, 2005
[8] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum and L. Viennot. Optimized Link State Routing Protocol for Ad Hoc Networks. In *Proceedings of the 5th IEEE Multi Topic Conference (INMIC 2001)*
[9] W. Chen, N. Jain and S. Singh. ANMP: Ad hoc network management protocol. *IEEE Journal on Selected Areas in Communications* 17(8) (August-1999) 1506–1531.
[10] Berry Kercheval. DHCP: a guide to dynamic TCP/IP network configuration. Prentice Hall PTR, c1999.
[11] Mike Schiffman. Building Open Source Network Security Tools: Components and Techniques. Wiley, 1st edition, 2002
[12] OpenWRT. *http://en.wikipedia.org/wiki/OpenWrt*
[13] Net-SNMP homepage. *http://net-snmp.sourceforge.net*
[14] RDTSC description. *http://en.wikipedia.org/RDTSC*
[15] Tcpdump home page *http://www.tcpdump.org*
[16] Douglas S. J. De Couto, Daniel Aguayo, John Bicket, Robert Morris. A High-Throughput Path Metric for Multi-Hop Wireless Routing. In *Proc. of ACM Mobicom* 2003.
[17] Richard Draves, Jitendra Padhye, Brian Zil. Routing in Multi-radio, Multi-hop Wireless Mesh Network. In *Proc. of ACM Mobicom* 2004
[18] Gilman Tolle, David Culler. Design of an Application-Cooperative Management System for Wireless Sensor Networks. In *Proc. of IEEE EWSN*, 2005. p 1 – 12
[19] Y.Ge, L. Lamont, L. Villaseno. Improving Scalability of Heterogeneous Wireless Networks with Hierarchical OLSR IN *Proc. of The OLSR Interop & Workshop*, San Diego, USA, August 2004
[20] Krishna Ramachandran, Irfan Sheriff, Elizabeth Belding, Kevin Almeroth, Routing Stability in Static Wireless Mesh Networks. *Passive and Active Measurement Conference*, Louvain-la-neuve, Belgium, April 2007.