WiMFlow: a distributed, self-adaptive Architecture for Flow Monitoring in Wireless Mesh Networks

Cristian Popi, Olivier Festor

WiMFlow: a distributed, self-adaptive Architecture for Flow Monitoring 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 present WiMFlow, a dynamic and self-organized flow monitoring framework in Wireless Mesh Networks. The protocol allows for an autonomic organization of the probes, with the goal of monitoring all the flows in the backbone of the mesh network accurately and robustly, while minimizing the overhead introduced by the monitoring architecture. A new mechanism that adapts the control messages emission interval to changes in the topology is introduced to keep the cost of the monitoring overlay low. The proposed mechanism is described and the performance of the monitoring framework is evaluated by simulation and experiments on a small scale wireless mesh network testbed.**

I. Introduction

Easy deployment and the absence of any need of centralized coordination make Wireless Mesh Networks a promising wireless technology [5]. The centerpiece of such a network is the wireless distribution system, its backbone. Such a backbone comprises a mesh of fixed nodes which play the role of access points, routers (when they relay packets for other routers), or both. The nodes in the backbone communicate in a multi-hop fashion over the wireless links established between them and based on technologies like WiFi or WiMax. The mesh structure provides robustness and availability: if one or multiple nodes fail, the packets will be rerouted on different paths by the "alive" nodes.

While these networks provide flexibility, their heterogeneous nature (especially in the case of community networks) comes with a range of challenges and difficulties; performance, interoperability and security being some of them. Mesh elements may operate under severe constraints such as reduced bandwidth, significant signal quality fluctuation due to environment conditions: obstacles, interferences, or hidden hosts. When designing a monitoring infrastructure for a wireless mesh network, one should therefore provide for situations like broken links, out of range nodes (weakening of the radio signal), or failed nodes. Monitoring such a network requires an architecture that is capable to react to these dynamic conditions in due time.

The frequency of updates of information needed by the monitoring system, versus the volume of transmitted control data is the trade-off the monitoring system has to cope with. To add to that, often, the monitored data uses the same network paths as normal traffic to reach the manager. Therefore one has to minimize the impact of monitoring on the network, while providing a system that adapts itself to unstable network conditions and yields accurate results.

In a previous paper [9] we proposed a flow monitoring framework for Wireless Mesh Networks that tries to answer all the challenges stated above. We presented an algorithmic and protocol mechanism that allows for an autonomic organization of the probes plane, that scales well and is robust to node and link failure. The emission intervals of the control messages were static configurable parameters; they do not follow the dynamics of the network topology changes, either causing the monitoring system to report flow count results that are not exact (when set to higher values), or generating too much control overhead (when set to lower values).

In this paper we focus on adapting the emission intervals of control messages to network topology changes. We set the following objectives: to find a mechanism for node self-organization such that all flows are monitored only once in their passage through the mesh backbone; to be resilient to topology changes generated by loss of links or nodes; to adapt control message exchanges to topology modifications, with the goal of keeping the management overhead low. As shown later in the paper, the proposed mechanism improves both the flow monitoring accuracy and the overhead of the monitoring service.

The rest of the paper is structured as follows: section II introduces related work. Section III presents the architecture of WiMFlow and describes the protocol for node self-organization. A mechanism for dynamic computation of control message intervals is shown in section IV and a prototype implementation of the flow monitoring system in section V. Section VI provides the evaluation of the proposed monitoring system, and the conclusions end the paper in section VII.

II. Related Work

In [11] the problem of sampling packets in a cost-effective manner is proposed, by solving the two conflicting optimization objectives: maximization of the fraction of IP flows sampled vs. minimization of the total monitoring cost. The solutions of the posed problems determine the minimal number of monitors and their optimal locations under different sets of constraints. In [6] Chaudet et al. also found that in terms of cost (of deployment and running of probes), it may be worthwhile to monitor only a part of the traffic (ie. 95%).

This reduces the number of probes needed to almost half in the examples presented in their simulations. These last two approaches are theoretical and they require a priori knowledge about the topology of the network, and the traffic flowing through the network. We aim to build a system that automatically and dynamically organizes the monitoring probes plane to capture at a minimum cost a high percentage of the traffic passing the network without knowing the flow graph in advance.

Huang et al. propose a monitoring architecture for IP flows in a wireless mesh network in [7]. MeshFlow records are being created on every mesh router in the path of a packet. By aggregation of these records a complete transportation path of packets can be deduced. Our proposal minimizes the export overhead by activating one probe only on the path of a flow to monitor it.

In [10], Gonzalez et al. build a network monitoring scheme, A-GAP, with accuracy objectives. To reduce the overhead of monitored information between the monitoring nodes and the management station the authors introduce a filter scheme by which a monitoring node does not send updates for small variations of its state or partial aggregate state computed on it. The filter is dynamically computed on each node based on a discrete-time Markov chains stochastic model. The results lead to the conclusion that accepting small errors in monitoring accuracy, overhead reduction is gained.

III. Overview of WiMFlow

All routers are possible probes. To take decisions on which one monitors a flow, probes are required to hold a global vision of the routing entries of all the nodes in the backbone. This allows a probe that sees a flow passing through its interfaces to trace the flow's path. A prerequisite for monitoring a flow is that a probe P that sees a flow F on one of its interfaces has to know the flow's entry and exit points in the backbone, as well as the next hop towards the exit point for each node on the path of the flow. The functional architecture of the monitoring system is presented in figure 1. The routing plane builds up the routing table of the mesh nodes (pro-active routing protocol needed). It then provides the routing table entries of all nodes to the monitoring overlay, which uses this information to organize the nodes into monitoring or non-monitoring probes. Two components come into the decision making process when organizing the nodes for monitoring: the routing information received from the routing plane to locally build the path of a flow on a node, and the metrics that allow to differentiate between nodes located on the path of the flow. These metrics are distance (in number of hops) of the node from the collector (to which the node is configured to send flow records), connectivity degree and up link quality. Nodes with better distance, higher connectivity or up link quality are the ones elected to monitor the flow.

In order to reduce the number of control messages we use the concept of Multi Point Relay (MPR) employed by the OLSR routing protocol to flood topology control messages. The MPR Set selection scheme is that of OLSR ($Hello$ messages are used). For flooding the network with routing entries, routing control message (RC) are sent by every node and broadcast via the MPRs, containing the entire routing table of the sender.

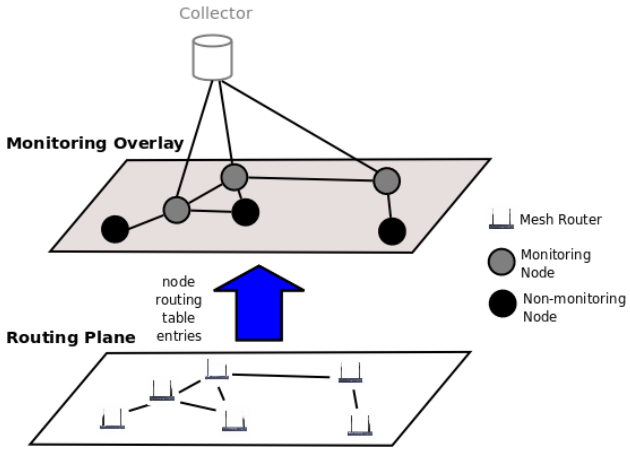

Fig. 1: Wireless Mesh Network flow monitoring architecture

With a global routing table available on each node, the full path of a flow, from entry to exit intro/from the mesh backbone can be computed. Once the complete path of a flow is obtained, the node compares its metrics to those of the nodes on the paths. The distance from the collector is the number of hops a message sent from the node to the collector has to travel, and it is computed by periodically using traceroute to the collector. The connectivity degree is the number of one-hop neighbors of a node. This can be easily deduced from the size of its one-hop neighbor list. In case both collector distance and connectivity degree are identical for two nodes, an up link quality metric, which estimates the quality of the first hop link towards the collector from the node is used to choose between the nodes. As soon as a node finds that there is another node on the flow's path with better metrics, it gives up comparing, and does not monitor the flow.

IV. Dynamic Computation of Control Message Intervals

Message Types and Waiting Intervals

Our system uses two types of messages to convey information between the nodes in the monitoring overlay:

- Hello messages: used for link sensing, they help build up the one and two-hop neighbourhood as well as the MPR set of a node; a node reports these neighbor sets via the Hello messages it periodically broadcasts
- Routing Control (RC) messages: used for local routing table entries dissemination in the overlay network. They contain $\{Next\_hop_i : Destination\_Node\_ID\_List\}$ pairs from the local routing table of the node that sends the RC messages.

In our protocol, we use two type of soft state timers: for message generation and for state maintenance. The message generation timers are the Hello and the Route-control (RC)

message periods. The state-maintenance timers are the holding timers for the neighbor set (one hop, two-hop and MPR) and for the routing table entries received from a node via an RC message. These state-maintenance timers are used by a node to remove any obsolete information after time-out, if it has not been renewed by a new Hello or RC message.

The $Hello\_Period$ is the duration of time between two successive *Hello* messages broadcast by a node. The $Hello\_Store\_Period$ is the time duration neighbourhood and MPR information are stored on a node receiving the *Hello* message. The RC_Period is the duration of time between two successive *RC* messages sent by a node. After a node A receives an *RC* message from a node B, it updates the routing table record information for B , and keeps this information for a RC_Store_Period time duration. If it doesn't receive a *RC* message from B during this time it will consider B is lost and will discard all routing information from it. This does not affect the accuracy of the monitoring system, since if B is lost, or there is no path from B to A, there cannot exist a flow that passes both A and B.

In [9] we inferred, through simulation, that the optimal sending intervals for the Hello and RC messages were 5 and respectively 10 seconds. An increase of these intervals yielded a weaker monitoring accuracy, while a decrease yielded a higher management overhead. We now propose a mechanism that automatically and dynamically modifies the soft state timers in the system, to adapt to changes in topology or local routing table (due to decisions in the underlying routing protocol). We first look at the changes in the underlying network that can influence the monitoring overlay, and how these influence the performance of the monitoring system as a whole.

Scenarios

In order to see the behavior of the system in the case of topology disturbance or routing table modifications we present 4 basic scenarios:

1) **Leaf node joins**. One Hello message has to be sent by all the nodes in the two-hop neighbourhood of the new node (in figure 2, denoted by G) to allow the new MPR nodes to be selected. The RC messages can then be properly flooded into the network via the MPR nodes. For a flow F having the path A-B-D-F-G, there is no path (and complete metrics) information available on any node, before all of the nodes on the path of the flow have had the chance to publish their local routing tables via RC messages. Therefore, in this scenario, both Hello and RC messages need to be dispatched promptly by the nodes, so that the concerned nodes (A, B, D, F and G) be aware of the path of the flow and of the metrics of the other nodes on the path. Flows that do not start/end at node G can be monitored without RC updates being sent after G joins.
2) **Leaf node leaves**. No fresh information that helps choose the monitoring node needs to be distributed in the network, because all nodes already have a complete global routing table image and know the metrics associated to all the other nodes. Eventually, the entries relative to the leaving node will time-out.
3) **Non-leaf node joins**. Hello messages need to be broadcast in the neighbourhood, such that the MPR set be recomputed around the recently joined node. For the flow F having the path A-B-D-F-G', there is no path (and complete metrics) information available on any node, before all the nodes on the path of the flow have had the chance to publish their local routing tables via RC messages. Since G' is not a leaf node, routes for flows that do not start/end at G' may also be modified (due to underlying routing protocol decisions). The accuracy of the system may be affected. Again, RC messages need to be dispatched promptly by the nodes.
4) **Non-leaf node leaves**. Flows starting/ending at G' no longer exist, but flows that were routed through G', before it left, are rerouted through other nodes (ie. B-D-G'-E becomes B-D-F-E, or B-C-E). Nodes on the new path of the flow may not have the complete path information. RC messages need to be dispatched to update routing table information available on each node. Hello messages also need to be sent to recompute the MPR nodes.
5) **Routing table change**. Link quality metrics may change over time, and the underlying routing protocol may choose a new path for a flow inside the backbone of the mesh network. The global routing table seen by the nodes in the overlay, may not be consistent with the changes, and flow monitoring decisions will not be able to be made, due to lack of complete flow path information. Again, RC messages need to be dispatched promptly by the nodes. Without any topology changes, Hello messages do not need to be sent immediately.

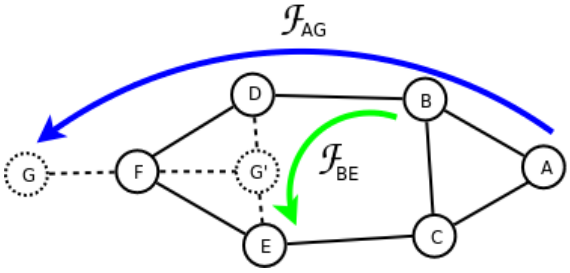

Fig. 2: Topology change scenarios - leaf node, G, joins/leaves; non-leaf node, G, joins/leaves

From the above scenarios, we can conclude the following. For stable topology and routing tables in the backbone the refresh rate of the Hello and RC messages can be maintained at a level that assures node aliveness. Whenever a change in the topology or in the routing table of the nodes is detected, the rate of the Hello and RC messages should be augmented to build a consistent global routing table view at the overlay level. We next develop a model for the Hello and RC intervals

that is in accord with the above conclusion.

A. Model for the Hello and RC Periods

Since variations for $Hello\_Period$ and RC_Period are triggered by two different events - link level modifications and local routing table changes, respectively (although the former can be the cause of the latter) -, we treat the cases differently, and develop individual models for each of them.

Hello_Period

We set $Hello\_Period$ to vary in between a minimum (m_H) and a maximum (M_H) configurable values. The mechanism that adapts the $Hello\_Period$ at node, n, is the following:

1) if n receives a Hello message from a node it doesn't have in its one-hop neighbor list, it sets $Hello\_Period$ to equal m_H (the case of a new node joining);
2) if n receives a Hello message from a node which contains information that is different from what n holds in its internal data sets (ie. symmetric and asymmetric neighbors, or MPR nodes), it sets $Hello\_Period$ to m_H;
3) if n does not receive a Hello message from a neighbor it holds in its internal data sets within the expiration time for that particular neighbor, all the data coming from or related to that neighbor is dropped, and $Hello\_Period$ is set to m_H;
4) if n receives Hello messages from its neighbors before the data holding time expires, and there is no difference to the data in its internal data sets, then $Hello\_Period$ grows exponentially towards M_H by $F_H(t) = 2^{\frac{t}{6}} + m_H - 1$, with t being the time elapsed since the $Hello\_Period$ was dropped to m_H. The $Hello\_Period$ then remains constant, at M_H until one of the first 3 conditions are met.

Figure 3 shows the variation in time of the $Hello\_Period$ for a topology change, that was observed by node n, and which triggered the drop to m_H of the $Hello\_Period$ value. We have chosen the increase of the $Hello\_Period$ to be exponential (with a slow increase around m_H) precisely to be sensitive to the topological instability which caused the $Hello\_Period$ drop mechanism to be activated in the node. With the chosen exponent coefficient, a = 1/6, the time the $Hello\_Period$ needs to increase from m_H to M_H is given by $t_{exp} = 6log_2(M_H - m_H + 1)$. For m_H=2 and M_H=6, t_{exp} = 14 seconds.

RC_Period

Like for the $Hello\_Period$ case, the RC_Period varies between a minimum, m_{RC}, and a maximum, M_{RC}. Whenever the node detects a change in its routing table, the RC_Period is set to m_{RC}. If there is no change in time in the internal routing table, the RC_Period increases exponentially towards M_{RC} by $F_{RC}(t) = 2^{\frac{t}{8}} + m_{RC} - 1$ and stays constant at M_{RC} until the first modification of an entry in the routing table. The exponent coefficient, a = 1/8, yields a RC_Period increase from m_{RC} to M_{RC} in $t_{exp} = 8log_2(M_{RC} - m_{RC} + 1)$. For m_{RC}=5 and M_H=10, t_{exp} = 21 seconds.

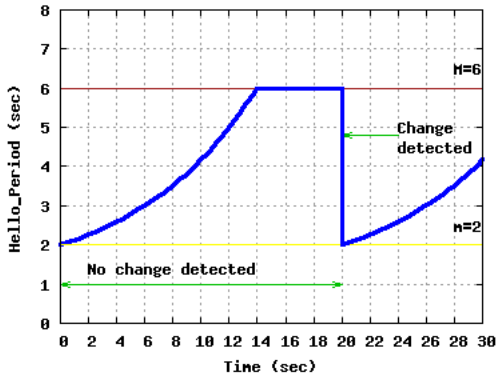

Fig. 3: Evolution of the Hello_Period over detection of changes in either the topology or the routing table

Since the nodes adapt individually to topology or routing table modifications, the mechanism that changes the $Hello\_Period$ and RC_Period for a node will not necessarily change these values the same way on another node. Therefore, nodes receiving topology information, via Hello messages, or routing information, via RC messages, have to know what the emission rate of the sender is for Hello, respectively RC messages. The expiry time (or the holding time) for the received information is calculated by multiplying the $Hello\_Period$ and RC_Period respectively, by a preconfigured parameter (ie. we take it to be the same as the OLSR default, 3). The simplest way to let receiving nodes know what is the emission period for the Hello and RC messages, is to explicitly send it via these messages. The holding period is computed by the receiving node for each of the received messages, and then associated to the topology/routing table information table entries for the sending node.

V. Implementation

We implemented a prototype of WiMFlow, which runs on every router in the mesh backbone as a management process. In figure 4 we present the design of WiMFlow on a mesh router. The Routing Protocol Service offers the routing table entries present on the node to the Global Routing Table Manager. The Global Routing Table Manger, on request from the Packet Filter, assembles the path of a flow throughout the mesh backbone, compares the metrics, and provides the Filter with an answer to whether it should monitor the flow. The Packet Filter, on packet arrival, checks in the flow cache to see if there is already an active flow entry for this packet. If there isn't any, it goes through the verification process with the Global Routing Table Manager, to check whether the node is supposed to monitor the flow. If the answer is positive, it sends the packet to the Flow Cache Manager, which creates a new entry for the flow in the cache. The Flow Cache Manager is also responsible for exporting the flows to the collector.

WiMFlow is entirely written in C. The flow cache management is based on nProbe ([1]). Flow export is in Netflow v5 and v9 format.

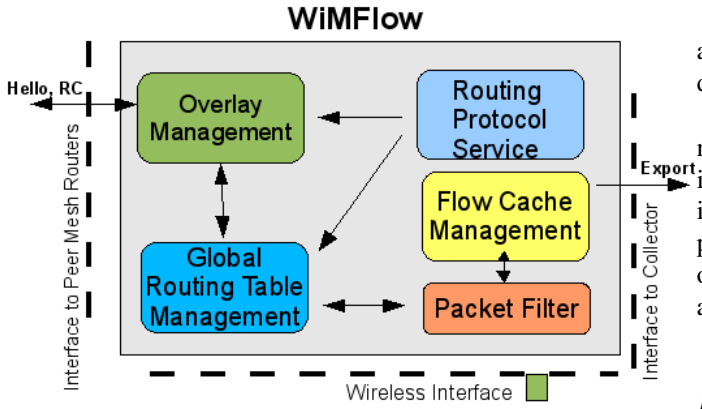

Fig. 4: Design of WiMFlow on a mesh router

VI. Evaluation

We evaluate the proposed flow monitoring system, with the embedded dynamic adaptation mechanism for the control messages intervals. We compare it against a probabilistic flow monitoring model, and against MeshFlow [7], a flow monitoring architecture, wherein nodes monitor all the traffic flows that pass through. The results are obtained through simulation with the NS-2 network simulator. We then present the results of our prototype implementation on a small scale wireless mesh testbed.

A. Comparison to other approaches

The probabilistic model against which we compare our system is described next.

Probabilistic model

Flows are counted by the probes independently. This means that a flow F, passing through a node N_i, will be counted with probability P_i (or discarded with probability 1-P_i), independently of the decisions taken by the other nodes. In the following we define the notations we use further on.

- F_i - a traffic flow passing through the network
- N_i - the number of routers in the network traversed by F_i
- P - the probability that a node monitors a flow that traverses it
- S_{F_i} - the sample space resulted from the sequence of Bernoulli trials that model the succession of counting "decisions" of the nodes traversed by a flow F_i; an element in the sample space is represented as a succession of N_i 1's or 0's, where "1" means that the flow was monitored on a node n_j, with $1 \leq j \leq N_i$, and "0" means that it was discarded;

The monitoring decisions are modeled by the same parameterizable probability, P, on all nodes n_j, $1 \leq j \leq N_i$. We have:

$$P_j = P, forall 1 \leq j \leq N_i \quad (1)$$

Next we see what is the distribution of the probabilities that a flow is counted a k number of times, for a varying P, and different path length sizes.

For this, we define the random variable, C_{F_i}, to be the number of times the flow F_i is counted, with the domain in S_{F_i} and the image in $\mathcal{N}$. Then the probability that F_i is monitored on k nodes out of the N_i through which it passes, is the binomial density of the random variable, C_{F_i}, or the probability of k successes in N_i independent trials of an experiment that has probability P of success on each trial:

$$P(C_{F_i} = k) = p_{C_{F_i}}(k) = \binom{N_i}{k} P^k (1-P)^{N_i - i}, 0 \leq k \leq N_i \quad (2)$$

In order to optimize for P with various flow path lengths, we need to have a general knowledge about the flow path length distribution across the network. Let D be the size of the backbone (number of infrastructure elements). If, in the equation 2, the flow path length, N_i follows a normal distribution law with the mean, $\mu = D/2$ and the standard deviation, $\sigma = D/4$, then the probability that a flow F is monitored k times while passing the network is given by:

$$P_{C_F}(k) = \sum_{n=1}^{D} P(N_i = n) P(C_{F_i} = k | N_i = n) \quad (3)$$

$$= \sum_{n=1}^{D} [\int_{n-0.5}^{n+0.5} \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2} dx \binom{n}{k} P^k (1-P)^{n-k}] \quad (4)$$

$$= \sum_{n=1}^{D} [\int_{n-0.5}^{n+0.5} \frac{4}{D\sqrt{2\pi}} e^{-\frac{8}{D^2}(x-\frac{D}{2})^2} dx \binom{n}{k} P^k (1-P)^{n-k}] \quad (5)$$

$$= \frac{4}{D\sqrt{2\pi}} \sum_{n=1}^{D} [\int_{n-0.5}^{n+0.5} e^{-\frac{8}{D^2}(x-\frac{D}{2})^2} dx \binom{n}{k} P^k (1-P)^{n-k}] \quad (6)$$

Figure 5 illustrates the probability that a flow F (with any path length) is counted at least once, once, twice and three times as it passes the backbone of the mesh network, plotted against the parameterizable probability, P. The network size, D, is 20.

We want to have a good flow monitoring accuracy, while at the same time reduce the number of probes that monitor the same flow. If we take P to be 0.2, then F is counted exactly once with probability 23%, almost the same as the probability it is counted twice, 22%, or the probability it is counted three times, 16%. The accuracy of the system (given by the probability that a flow is counted at least once) is 85% for P=0.2.

In simulating the probabilistic model we choose P = 0.2. Note that MeshFlow is a particular case of the probabilistic model, where P=1 on each node.

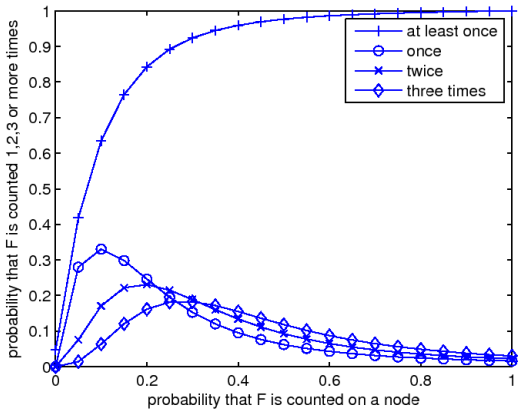

Fig. 5: Probability that a flow is monitored once, twice, three times and at least once for a network size D=20

Simulation setup and scenario

We simulated a mesh network using the NS-2 simulator [3]. For the MAC layer we used the 802.11b with RTS/CTS (Request to Send/Clear to Send). The RTS/CTS mechanism is enabled to effectively realize multi-hopping in the backbone. For the routing protocol, we chose OLSR, and use the UM-OLSR patch [4] for ns-2. We conducted our experiments on a variety of network topologies with sizes between 20, 50, and 100 nodes. Nodes in all topologies are fixed, to simulate the static routers in the wireless mesh network backbone.

The traffic pattern is modeled such that 80% of the traffic is directed to or coming from the gateway nodes ([8] states that practically all traffic is to/from a gateway). TCP flows are used to emulate user traffic, with two-way TCP agents randomly disposed over the backbone nodes, and application-level data sources (traffic generator) attached to the agents. We use two-way TCP because it implements SYN/FIN connection establishment/teardown.

For the experiments that follow we use the following parameters:

- for OLSR, the $TC\_Interval$ is set to 5 seconds, the $Hello\_Interval$ to 2 seconds and the holding intervals, three times the $Hello\_Interval$ and $TC\_Interval$;
- for WiMFlow, without the control message interval adaptation, the $Hello\_Interval$ is set to 5 and the $TC\_Interval$ to 10 seconds; the holding intervals are three times as big;
- for WiMFlow, with the control message interval adaptation, we set m_H to 2, M_H to 6, m_{RC} to 5 and M_{RC} to 10 seconds.
- for the probabilistic model, we use $P = 0.2$;

Accuracy

The accuracy measures how close to the proposed goal the monitoring system is, in terms of the number of monitored flows, and the number of monitoring probes per flow.

We first measured the accuracy for a stable mesh network (the topology did not change during one experiment and the routing tables on all of the nodes were stable). We conducted series of 10 experiments for network sizes of between 20 and 50 nodes with random topologies. As expected, our flow monitoring system (with and without dynamic control messages intervals) counted all flows exactly once. For MeshFlow we obtained a 100% number of flows counted, and an average number of counts per flow of 9.544. For the probabilistic model there was an average flow count of 71%.

Next, we compute the accuracy for an unstable mesh network topology. We simulated this by adding 5 nodes, one node every 5 minutes to the low-density network (size=20). The total number of TCP flows sent in this experiment was 3000. The experiment lasted 30 minutes. We repeated it 5 times for different topologies (same network size). The first node was added after 5 minutes. Due to the chosen policy: not to monitor flows, when the full flow path can not be reconstructed, the average monitored flow percentage in our monitoring system is 94.7% without and 97% with the control message interval adaptation mechanism. In MeshFlow, the accuracy stays at 100%, at the expense of monitoring a flow on average 8.2 times. With the probabilistic model, the percent of monitored flows is 69.3%.

MeshFlow has the better accuracy, but that at the expense of counting flows more times. We will see what that means to the export overhead next. The dynamic adaptation system for the control message intervals improves slightly the accuracy of flow monitoring. This is due to the fact that the lower limits of the $Hello$ and RC periods in the adaptive system are lower than the default Hello_Period and RC_Period values in the system without the adaptive mechanism. We see how much we gain in management overhead further on.

Accuracy vs Management Overhead

The management overhead measures the number of coordination packets sent by the probes. The packets sent by the monitoring overlay are $Hello$ and RC messages.

We consider 20 and 50 node mesh networks. We try different values for the $Hello$ and RC message periods. In all experiments, randomly, nodes fail for a duration of maximum 1 minute (only one node at a time) and then come back alive. This way we simulate more realistically periodic link failure or quality variations. The experiments are all 5 minutes long. The figure 6 lists the average values of the accuracy and number of control messages over a 5 minute period for network sizes of between 20 and 50 nodes for the 4 monitoring approaches.

The adaptive control messages intervals mechanism seems to be efficient: it increases the accuracy of the monitoring system to 95.2%; this is higher then the accuracy of the system without the adaptive mechanism at (Hello_Period, RC_Period) = (2,7) seconds, but is lower on overhead by 6 messages/node/sec.

MeshFlow and the probabilistic approach don't introduce any management overhead.

Export Overhead

We also consider the export overhead, which we quantify in terms of the number of flows a monitoring node counts per the

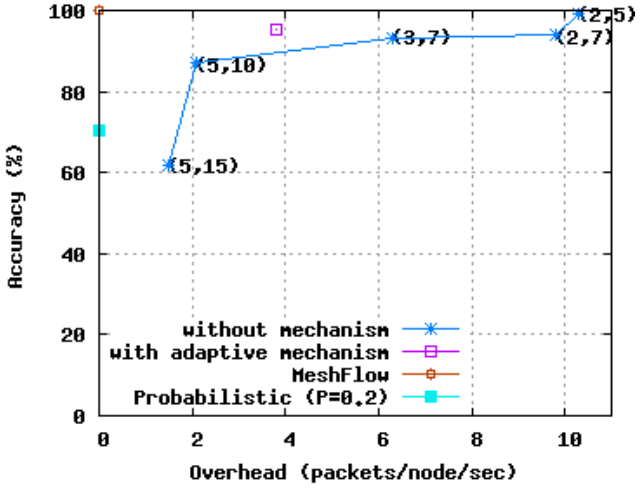

Fig. 6: Protocol accuracy vs overhead for different pairs of ($Hello\_Period$, RC_Period) vs embedded adaptation mechanism, MeshFlow and probabilistic model

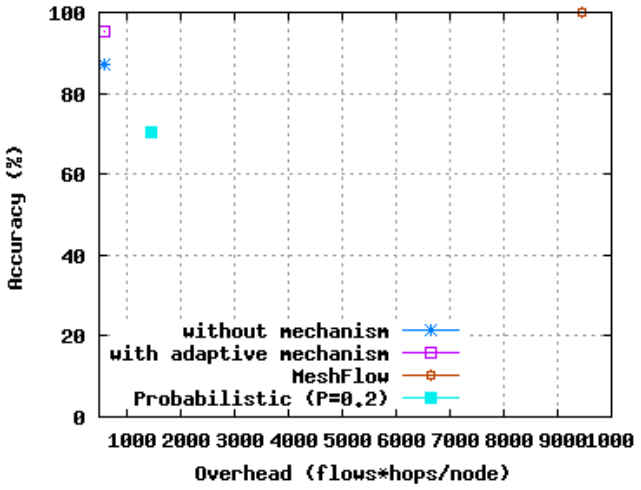

Fig. 7: Protocol accuracy vs export overhead counted as the number of flows monitored by a node, over the distance in hops to the collector

distance in number of hops from the monitoring node to the collector. This is computed over all probes and a comparison is presented against the overhead generated by exporting flow information by using the probabilistic model and MeshFlow (computed the same way). We considered the same setup as above, and sent 3000 TCP flows (traffic generators placed randomly over between 20 and 50 nodes).

We multiplied the number of flows counted by a node and the number of hops from the node to the collector and averaged the results up for all the monitoring nodes. The obtained results are shown in figure 7.

Our monitoring system generates under half the export overhead generated by the probabilistic model, and as much as 15 times less then MeshFlow.

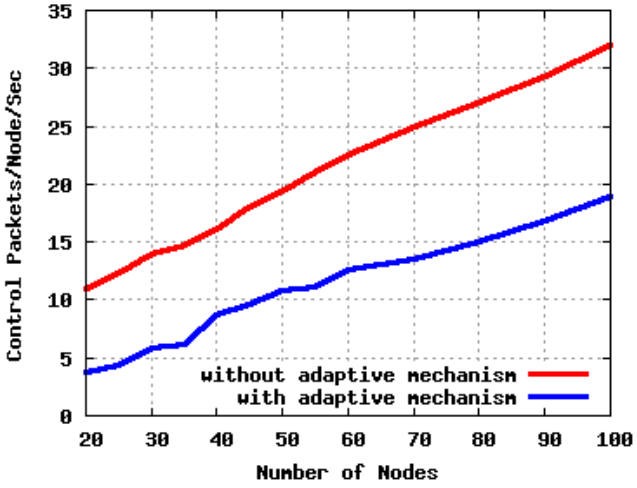

Fig. 8: The average number of control packets per node per second as network size increases

We next see how increasing the number of nodes in the backbone affects the control plane.

B. Scalability

In these experiments we compared the new flow monitoring system, with the embedded adaptation mechanism, against the monitoring system with the Hello_Period and RC_Period set to (2,5).

The amount of control traffic and message size were measured as the network size increased. We performed a set of experiments, each one 5 minutes long, where we varied the number of nodes from 20 to 50 (in increments of 5) and then 100. Figure 8 shows the effect of increasing the number of nodes in the topology to the number of control packets conveyed in the network. The number of packets vary linearly with the number of nodes. The number of control messages can be reduced provided the accuracy of monitoring is lowered, or by using the adaptive control message interval mechanism while keeping the accuracy high.

The adaptive mechanism doesn't bring any improvement to message size, which increases linearly with the number of nodes. The RC message size depends on the routing table size of the nodes; with a pro-active routing protocol, each node holds entries for all nodes. Therefore, RC messages, even though optimized to include a node only once still holds at least all backbone nodes' identities.

C. Testbed Experiments

We also ran experiments with an implementation of WiMFlow to test how the monitoring system behaves in a real environment. We next present the testbed on which we ran WiMFlow and the results obtained.

Testbed Setup

The testbed comprises 4 routers: one Linksys WRT54GL, with a 200 MHz processor and 16 MB of RAM, running

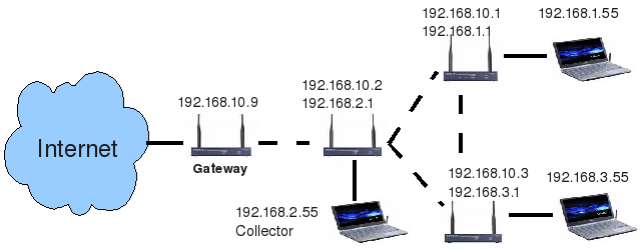

Fig. 9: Testbed layout

OpenWRT, and three Alix based boards from PC Engines with 500 MHz processor and 256 MB of RAM, running Voyage Linux. All 4 devices have one wireless card running in the 2.4GHz bandwidth and supporting rates up to 54Mbps, by which they form the backbone of the mesh network. All the wireless interfaces were set to run on the same channel. The Linksys is also acting as a gateway to the Internet. We attached wired clients to the three non-gateway routers. The routing protocol inside the backbone over the wireless interfaces (192.168.10.0/24) is OLSR.

We installed and configured WiMFlow on each router to monitor the wireless interface which is used for communication in the mesh backbone. Netflow v9 flow information was exported by each device running WiMFlow to the collector. We used ntop [2], running on a Linux PC attached to one of the routers (see figure 9), as a flow collector.

Results

We measured the accuracy and the overhead of WiMFLow. We used Iperf to generate both UDP and TCP traffic between 192.168.1.55 and 192.168.3.55 in one experiment, and 192.168.1.55 and the "Internet" (an Iperf server was set up on a machine with a routable IP address) in a second experiment. In the first experiment ntop reported around 87% of the number of flows that were sent by 192.168.1.55 and monitored by its attachment router. The flows were correctly monitored by 192.168.10.1, instead of 192.168.10.3 due to the better link quality towards the attachment router of the collector. The 13% non-monitored flows are due to interference most probably caused by access points in the lab. For the second experiment, 84% of the number of flows were monitored by 192.168.10.2, which is the better placed router on the path of flows from 192.168.1.55 to the Internet.

We then computed the overhead of both the routing protocol and the WiMFlow, by capturing all the packets going out the ports defined for these services and averaging their sizes and number over 30 minutes. The results show OLSR generates an average overhead per node of 21.7 kbps with the default settings (Hello_Interval at 2 seconds, and TC_Interval at 5 seconds). WiMFlow generates on average 2.3 KBps per node. The CPU usage in any of the routers running WiMFlow was low (more than 90% idle).

VII. Conclusions and Future Work

In this article we have proposed WiMFlow, a flow monitoring architecture for Wireless Mesh Networks that is both resilient and adaptive to environment interferences and topology changes, with the goal of counting all of the flows passing through the backbone on only one mesh node. We build a monitoring overlay, in which a probe is able to decide if it monitors a flow that passes through its interfaces. By sensing changes in the topology or local routing table, a node can automatically change the intervals of the control messages to allow the probe overlay to adapt quickly in maintaining the global routing table image.

We have evaluated the monitoring system to see its accuracy and the impact of the introduced management overhead. We have found that the proposed models for the adaptive mechanism regarding the $Hello$ and RC intervals are improving the original flow monitoring system. Overall, WiMFlow outperforms both a simple probabilistic model that was proposed by us, and MeshFlow, a flow monitoring approach developed for Wireless Mesh Networks.

We have implemented the monitoring system, and experimented with a real testbed set up in the premises of our lab. The results of the tests seem conclusive for a small scale testbed.

In the future, since the monitoring system is based on the metrics advertised by the participating nodes, we plan to investigate the situation where one or more nodes maliciously send faulty metrics, to either avoid monitoring (to save processing and communication bandwidth) or to monitor all traffic flows with the intention of eavesdropping, for instance.

References

[1] nProbe - http://www.ntop.org/nProbe.html - accessed on september 2009.
[2] ntop - http://www.ntop.org/overview.html - accessed on september 2009.
[3] The Network Simulator NS-2 - http://www.isi.edu/nsnam/ns/.
[4] UM-OLSR patch for ns-2 - http://masimum.dif.um.es/?Software:UM-OLSR.
[5] Ian F. Akyildiz, Xudong Wang, and Weilin Wang. Wireless mesh networks: a survey. *Computer Networks*, 47(4):445–487, March 2005.
[6] C. Chaudet, E. Fleury, I. Guérin-Lassous, H. Rivano, and M-E. Voge. Optimal positioning of active and passive monitoring devices. In *CoNEXT*, Toulouse, France, October 2005.
[7] Feiyi Huang, Yang Yang, and Liwen He. A flow-based network monitoring framework for wireless mesh networks. *Wireless Communications, IEEE*, 14(5):48–55, October 2007.
[8] Jun, Jangeun and Sichitiu, M. L. . The nominal capacity of wireless mesh networks. *Wireless Communications, IEEE*, 10(5):8–14, 2003.
[9] Cristian Popi and Olivier Festor. Flow monitoring in wireless mesh networks. In *AIMS '09: Proceedings of the 3rd International Conference on Autonomous Infrastructure, Management and Security*, pages 134–146, Berlin, Heidelberg, 2009. Springer-Verlag.
[10] Alberto Gonzalez Prieto and Rolf Stadler. A-GAP: An adaptive protocol for continuous network monitoring with accuracy objectives. *IEEE Transactions on Network and Service Management (IEEE TNSM)*, 4(5), june 2007.
[11] K. Suh, Y. Guo, J. Kurose, and D. Towsley. Locating network monitors: Complexity, heuristics and coverage. In *Proceedings of IEEE Infocom*, March 2005.