A Distributed Network Monitoring Framework for Wireless Networks

Roberto Riggio, Matteo Gerola, Daniele Miorandi, Andrea Zanardi
CREATE-NET
Via Alla Cascata 56/D, 38123, Povo, Trento, Italy
name.surname@create-net.org

François Jan
France Telecom, R&D Division
22300, Lannion, France
francois2.jan@orange-ftgroup.com

***Abstract*—Multi-hop wireless networks are emerging as a viable alternative for building access networks in areas where conventional solutions (cellular, fiber) are neither feasible nor attractive from an economical standpoint. The management of multi–hop wireless networks represents an overly complex task because of the joint effect of the time-varying nature of the radio channel, the mobility of users and the customary presence of adaptive, self-configuration features. Various solutions are currently being researched, whereby network management functionalities are performed autonomously at the network nodes themselves. Such approaches require a monitoring framework able to bring network–level information to the relevant decision points in an effective and robust manner. In this paper, we present a distributed network monitoring framework, specifically developed for wireless multi–hop networks. The system architecture and the protocols are presented together with results obtained using a prototypical implementation over a real–world testbed. Experimental results show that the framework generates a limited amount of traffic, and that the system is able to consistently recover from node failures.**

***Index Terms*—wireless networks, network management, network monitoring, mesh architecture**

I. Introduction

Multi-hop wireless networks, also referred to as Wireless Mesh Networks (WMNs) [1] represent an alternative to traditional (cellular-style) architectures for building wireless access networks over medium-sized areas. WMNs customarily rely on the use of wireless technologies operating in unlicensed bands, such as those specified within the IEEE 802.11 family of standards. This, coupled with the use of off-the-shelf hardware for building dedicated devices and with the self-configuration features typical of IEEE 802.11.x operations, enables the construction of networks presenting appealing features in terms of limited initial capital expenditures (due to low cost of devices and no need for complex network planning) and potentially low operational expenditures (due to the self-configuration capabilities of the devices). At the same time, the monitoring and management of such networks represents a complex task. The distributed nature of WMNs, channel fluctuations and the presence of self-configuration features make it difficult for an operator to apply conventional network management tools and techniques.

In the last few years, a tendency emerged to distribute network management functionalities within the network itself [2], [3], [4], [5], [6]. While such a trend spanned both wired and wireless domains, it results particularly appealing for multi-hop wireless networks, as it matches well the inherently distributed nature of such systems. The effective development of distributed network management solutions require, as common building block, a scalable and efficient infrastructure for gathering network status information and conveying it, properly aggregated whenever needed, to the relevant decision points, where network management tasks are performed.

Conventional network monitoring architectures and solutions, mostly based on SNMP [7], [8], are not suitable for such a purpose. First, they customarily rely on a centralized architecture[1], whereby the information collected is delivered to one single decision point, where management decisions are implemented. Second, they lack flexibility in the assignment of the operations to be performed at each node. As in wireless networks topology may change over time (due to, e.g., channel fluctuations, possible node failure, addition of new nodes etc.) it is necessary to envision systems where the location of the aggregation, analysis and decision points may change at run-time without disrupting system's operations. Third, redundancy in the storage of monitoring data should be envisioned, in order to ensure accessibility of information and robustness of the overall system in spite of possible node/link failures.

In this paper we present *OBELIX*, a distributed network monitoring framework specifically tailored to multi–hop wireless networks, such as IEEE 802.11–based Wireless Mesh Networks. The major contributions of this work are:

- We introduce a distributed architecture for monitoring wireless multi–hop networks, able to support run-time changes in the operations performed by different nodes;
- We define a set of algorithms and corresponding protocols, able to ensure robustness of the monitoring infrastructure with respect to nodes failure and to changes in the network topology.
- We provide a prototypical implementation of the proposed architecture and protocols and validate it over a real-world small-scale testbed, demonstrating the viability of the approach and evaluating its performance in terms of traffic overhead and resilience.

The remainder of this paper is structured as follows. Sec. II provides an overview of related work and of the state-of-the-art in the field. Sec. III describes a number of requirements and presents how they were turned into a set of design principles. Sec. IV describes in details the OBELIX system architecture, components and related protocols. Implementation details and numerical results, obtained from a small-scale testbed implementation, are

[1]Albeit in v2 of the standard an interface was specified for enabling communication among agents, SNMP is inherently centralized in nature.

reported in Sec. V. Sec. VI concludes the paper discussing a number of open issues and future extensions.

II. Related Work

In [9] the authors aim at optimizing a distributed polling system by sharing the load among a certain number of monitoring agents, whose location is chosen in such a way to ensure bounded link utilization in an IP network. The resulting hierarchical system exploits an intermediate aggregation layer between the monitoring agents and a centralized manager, which acts also as (single) information repository. The work presented in [10], similarly, proposes a set of techniques and algorithms to measure links' delay and to identify faulty links in IP networks. In order to achieve such a goal, the authors aim at selecting the optimal location of monitoring agents, able to cover all the links in the networks in the presence of a bounded number of link failures. The approach presented in [11] is slightly different, in that it addresses the concurrent passive monitoring of traffic flows at multiple locations in an IP network. The objective in such a context is to sample the packets belonging to the same IP flow by distributing several monitors within the network and by controlling their sampling rate. A trade–off exists between monitoring cost (numbers of monitors and their sampling rate) and coverage. The authors propose an heuristic for choosing the location of monitoring nodes and their sampling rate. In [12] heuristics are used in order to determine the optimal number and the placement of instrumentation in order to obtain distance maps of the Internet assuming an unknown yet static network topology. In all the above mentioned works, the location of active network monitoring elements is *statically* selected at deployment time, using a variety of proposed algorithms and heuristics. In wireless multi–hop networks, on the other hand, the topology can change over time because of a number of phenomena, including, e.g., node/link failures, channel fluctuations, addition of new nodes etc. In all such cases, a monitoring framework should be able to adapt its configuration (including location of aggregation/analysis nodes) to match the features of the current topology in a transparent way. Further, the above mentioned works do not address issues related to the communication and transfer of information among monitoring agents. In this work we aimed at filling such a gap, by proposing a scalable group communication system capable of (i) dynamically adapt at run-time the operations performed by single nodes in the monitoring framework (ii) marshaling the communication among a group of monitoring agents, thereby effectively realizing a distributed cache of the global network state that can exploited in order to implement distributed/autonomic/in–network management solutions.

A large set of protocols exists to support network and network devices management. Common solutions include SNMP [8], ICMP [13], netconf [14], and capwap [15]. Most of these protocols have been designed around a centralized architecture. Accordingly, all network nodes run an information gathering process which responds to queries coming from a centralized management entities. When a problem is recognized, the running process may send alerts to some (predefined at deployment time) management entities. Upon receiving these alerts, the management entities are programmed to react by taking appropriate actions (e.g., operator notification, event logging, system reboot/shutdown, etc.). A Distributed Architecture for Monitoring Mobile Networks (DAMON) is introduced in [16]. DAMON relies on agents within the network to actively monitor network behaviour and to send this information to data repositories. DAMON's generic architecture supports the monitoring of any protocol, device, or network parameter. In [17], the authors propose a novel monitoring framework capable of dynamically tuning the granularity of the data collection procedures according to some observed events (e.g. threshold crossing). Albeit showing adaptive characteristics from a data gathering perspective, the proposed systems still relies on centralized storage and processing of information. In contrast to the aforementioned works, our approach relies on a fully distributed repository to maintain the global network state and to make it available to all the nodes running network management tasks.

An approach similar in spirit to ours is the one presented in [18]. In this paper, the authors present a decentralized autonomic network management solution where management decisions are taken by each node autonomously, based on its local view of the network and on the knowledge obtained from its one–hop neighbours. A central authority is used to enforce global policies and constraints. OBELIX improves the network monitoring part of this work by delivering a platform where each node's participation in the monitoring efforts (i.e. its role) can be dynamically changed at run–time to adapt to changing conditions (e.g., addition of new nodes in the network, nodes becoming unreachable due to faulty links etc.).

III. Requirements and Design Principles

A. Monitoring Framework Requirements

The OBELIX framework has been designed around the following set of requirements:

- *Flexibility and adaptivity in the allocation of monitoring tasks.* Different network scenarios and topologies may require different setups of the monitoring infrastructure. Nodes may just provide information on locally measured variables or, instead, aggregate information gathered from nearby nodes. This can depend on either the amount of available resources (storage, computing) or on the node's location in the network topology. As in the wireless domain network conditions and topology may change over time (due to, e.g., node/links faults, etc.), the monitoring framework should be able to automatically adapt at run–time the allocation of tasks to devices.
- *Efficient spatial reuse.* The monitoring framework should make an effective use of the available network resources. In particular, it should avoid overloading certain portion of the networks and should evenly distribute the signalling traffic.
- *Robustness and resilience.* The monitoring framework shall be robust with respect to node or link failures. In particular, the availability of information with global scope should be preserved in spite of failures of the nodes holding it. No single point of failure should be present in the system. Further, the system should possess self-healing properties, being able to automatically recover from the failure of nodes. This means that the monitoring system should be able to reconfigure itself automatically in order to restore correct operations once failures have been detected.

- *Ubiquitous network management.* Network administrators should be able to both monitor and manage the network from anywhere using technologies such as an SNMP–compliant network management software or through a Web interface. This implies the ability to retrieve from any point in the network information on the whole network status.

The design principles derived from the above requirements are presented in the following subsections.

B. Monitoring Roles

OBELIX supports different levels of participation in the monitoring efforts by the nodes in the network. Two main tasks are envisaged:

- *Information gathering.* Monitoring agents (referred to also as *Taps* in the remainder of the paper) running within a node gather the local network state either by sniffing the traffic flow in their neighbourhood (passive information gathering) as well as by performing on-demand/periodic measurements (active information gathering).
- *Information analysis.* The local network state information gathered by the Taps is periodically sent to a set of management entities (referred to also as *Sinks* in the remainder of the paper), where it is stored to maintain a global view of the network and analysed in order to assess the arising of possible risk situations.

Information gathering and information analysis represent separate yet not mutually exclusive functionalities. Any node in the network can support a single functionality, both, or none at all. In the latter case, information about the state of a network device can only by inferred through passive sniffing from neighbouring Taps. It is worth noticing that, for the sake of brevity, this last network configuration has not been analyzed in this work.

C. Node-Level Autonomy

In OBELIX nodes possess autonomy at two levels. First, nodes running information gathering tasks periodically send the detected changes in the monitored parameters to the relevant Sink. This 'push-based' approach contrasts with the polling-based mechanisms used in the vast majority of network monitoring solutions (where active delivery of messages by Taps is considered only for warning/alarm messages).

Second, nodes may autonomously decide which role they should play in the monitoring framework, i.e., whether they should act as Sink or Tap. In particular, Taps can "upgrade" themselves to the Sink role whenever there is no Sink available within a given distance. Such a mechanism is used to both obtain efficient spatial reuse as well as to ensure that the overall monitoring framework configuration matches well changes in the network topology.

D. Support of Different Monitoring Configurations

OBELIX supports the following configurations:

- *Centralized.* Collection of the network state is done by a single network element. It implements both Information Gathering and Information Analysis functionalities; the other nodes in the network implement only Information Gathering functionalities. Such an approach is conceptually equivalent to an SNMP-based network management architecture, where a single network element periodically polls network devices and revives alarms and events.
- *Fully distributed.* Collection and analysis of the network state is done by every node in the network. Such an approach effectively creates a distributed repository holding the global network state information, where every node has a global knowledge of the network. While delivering the highest level of resiliency to network failures, this setup is characterized by high signaling overhead, as the information collected must be circulated among all nodes.
- *Hybrid.* Only a subset of network nodes implement the information analysis functionality, i.e. the Sinks. Nodes implementing Information Gathering functionalities, i.e. the Taps, are organised into clusters. In this configuration, Each cluster is composed by a variable number of Taps and one Sink, which acts as cluster head. Sinks have a complete knowledge about the state of the nodes in their cluster, while information about other clusters is limited to Global Managed Objects.

E. Management Dashboard

A management dashboard shall allow network administrators to both monitor and manage the network from anywhere using just a Web browser. Such Dashboard provides the network administrator with a synthetic representation of the network status in order to enable quick and efficient troubleshooting of critical situations. Retrieval of information encompass appropriate look-up and routing functionalities, in order to ensure that, through the dashboard, information on any node can be accessed (and not only that directly available on a Sink). In other words, request for information coming from the dashboard should be resolved and routed to the Sink where such information is present. In order to provide backward compatibility with existing network management systems, an SNMP interface is also supported.

IV. OBELIX System Architecture and Protocols

A. System Model and Architecture

The general model of a wireless mesh network is illustrated in Fig. 1. We can identify the following logical roles:

- *Routers.* They build and maintain the multi-hop wireless backhaul by establishing wireless links between nodes.
- *Gateways.* They interface the WMN with another network, typically the Internet.
- *Access points.* They provide wireless connectivity to the end-users' clients.
- *Clients.* They are used by the end-users to gain access to the Internet.

OBELIX builds a monitoring overlay in the network, whereby physical network devices are associated to a role that defines the monitoring functionalities they perform. The resulting system architecture is reported in Fig. 2. We define two logical roles played by nodes in the monitoring framework: Taps and Sinks.[2] As described in the previous section, Taps implement information

[2]With a slight abuse of terminology, we define by Tap (respectively: Sink) both the software agent performing the information gathering (respectively: information analysis) functionalities as well as the node running the corresponding software agent.

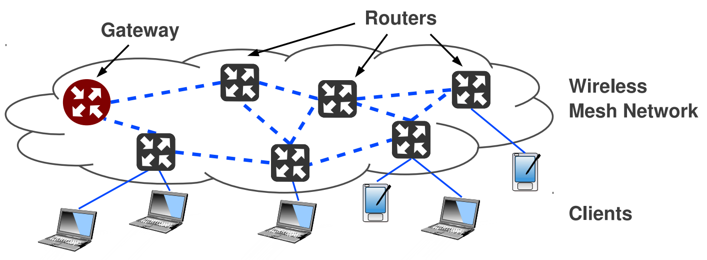


Fig. 1: A wireless mesh network: physical topology and logical roles of devices (clients, routers, gateways).

gathering functionalities, while Sinks implement information analysis functionalities. At bootstrap, Taps selects one Master Sink for their normal operations and a number of Slave Sinks to be used if the Master Sink fails. During its normal operations each Tap is always associated to at least one Sink.

The network-level parameters and quantities monitored by a Tap are termed managed objects. Taps periodically send information about their managed objects (e.g., routing tables) to the Sink node(s) they are associated to. Such information is structured as {`key, value`} pairs, where:

- the `key` uniquely identifies the managed object;
- the `value` holds the actual value of the managed object.

Managed objects are defined as global or local by means of a configuration file. The latter is used to identify pieces of information that have limited scope, e.g. the number of bytes transmitted over an interface. The former identifies pieces of information that have a network-wide scope, such as, e.g., the link quality metrics, the characteristics of the radio interfaces available, the hostname and the geographical position of the network device (whenever available).

In order to ensure robustness and availability of information in spite of possible nodes' failures, two mechanisms are encompassed. First, Taps can be associated to more than one Sink allowing monitoring information to be stored at multiple locations. Among the Sinks a node is associated to, one is denoted as *Master Sink*, while the other ones are termed *Slave Sinks*. Second, information on global managed objects is replicated on all Sinks. The *Master Sink* is responsible for handling the replication of the global information coming from the Taps associated to it.

Sinks periodically broadcast a beacon to advertise their presence. This beacon is flooded across the network by Tap nodes allowing all nodes to have an updated knowledge on the Sink nodes available, and on their distance from them (in terms of number of hops or some other link quality metric supported by the underlying routing protocol).

In line with the design principles outlined above, decisions on which role shall be played is chosen at run–time by each node. By default, nodes play the Tap role only. If no Sink is available within a given distance, the node may decide to "upgrade" itself to the Sink role. No explicit downgrade of nodes from Sink to Tap is foreseen. The details of the association process will be described in the following subsections.

B. *OBELIX Software Architecture*

The building blocks of the OBELIX framework and their relationships are sketched in Fig. 3.

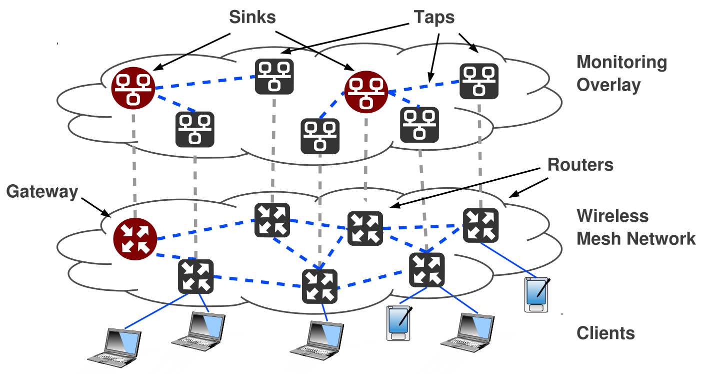


Fig. 2: The OBELIX system architecture: roles played in the monitoring overlay (Taps, Sinks) and their mapping to the physical network topology.

The Tap is a software process running in each managed device. A Tap has knowledge of the local (node-specific) parameters to be monitored. Domain-specific information (e.g. routing tables, link status, etc.) is collected by specialized plugins, termed *Backends*, and presented to Tap using a protocol agnostic representation. In Fig. 3, two possible routing protocols (OLSR and WING) are illustrated. The Tap process periodically sends the information collected to the Sink(s) it is associated to. The Tap also includes means to run measurement campaigns, in the form of components (in the figure, `ping` and `iperf` plugins are represented), that can be triggered by means of control messages coming from Sinks for actively monitoring the network status.

The Sink is a software process running in a subset of the nodes composing the network. As depicted in Fig. 3, Sinks gather information from the Taps associated to them, store and analyse it. Information coming from the Taps and marked as having global scope is replicated across all Sinks in order to offer robustness against node's failures. By means of this replication process, Sinks effectively build a distributed base of monitoring information.

The following types of messages are used:

- `SINK_HELLO`: messages broadcasted by Sink nodes to signal their presence. `SINK_HELLO` messages are flooded across the network by Tap nodes. This message also contains the number of Tap currently associated to the Sink broadcasting the message.
- `TAP_UPDATE`: messages sent by a Tap to the Sink(s) it is associated to, containing information on the monitored object (in the form of changes with respect to the previously communicated values).
- `SINK_UPDATE`: messages exchanged among Sinks containing updated values of the global managed objects.
- `SINK_ASSOCIATE`: association request sent by a Tap to a Sink node.
- `SINK_ASSOCIATE_ACK`: positive reply sent by a Sink in response to an association request.
- `SINK_ASSOCIATE_NACK`: negative reply sent by a Sink in response to an association request.

Every sink has an HTTP interface through which a web-based management dashboard can access network information. This

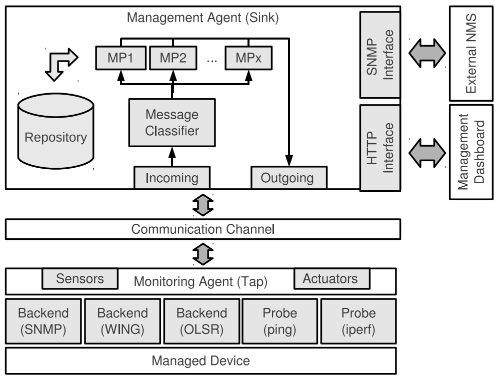


Fig. 3: System Architecture: Taps, Sinks and their interactions.

feature relies on the ability to retrieve, from any Sink in the network, information on *any* device in the system. This implies, in turn, the ability to look-up information, referring to a given Tap and having local scope, by redirecting the query (coming from the dashboard and injected at a given Sink) to the Sink managing that given Tap. An SNMP interface is provided in order to enable interoperability with standard network management systems.

C. Algorithms and Protocols

In this subsection we provide a detailed description of the procedures and algorithms implemented by the OBELIX monitoring framework. They can be conceptually grouped into three main areas, namely *autonomic role selection*, *data dissemination*, and *data look-up*.

1) Autonomic Role Selection: This set of mechanisms covers the bootstrapping phase of a node, including procedures for sink discovery, association and upgrade.

As outlined in the previous subsection, nodes in the network autonomously choose their role in the monitoring task. The goal is to build a monitoring overlay where Sinks are uniformly distributed across the network and where any Tap is not too far from its Master Sink (i.e., no more than a given number of hops). If more than one Sink satisfies this criteria, the Tap chooses the Sink with the lowest number of Taps associated to it (improves load balancing). We denote by D_{TH} the maximum distance from a Sink.

Two operating modes for sink discovery are supported:

- *Proactive.* Sink discovery is based on a multi–hop beaconing infrastructure. Sink nodes broadcast their presence via `SINK_HELLO` messages; Tap nodes receiving such messages forward them, so that newly arrived nodes can gather information on the current availability of sinks to associate to. This approach is robust and works well in volatile environments.
- *Reactive.* Sink discovery is initiated by the newly arrived node. This approach is faster to select the Master Sink, but could lead to instability in rapidly changing networks.

The two operating modes are not mutually exclusive. In an OBELIX-managed network, all nodes must support the proactive operating mode. Upon this basic functionality, the reactive operating mode can be implemented in order to speed-up the bootstrap procedure.

Upon discovery of a suitable Sink node, a Tap sends a request for association; once this is acknowledged by the Sink, the Tap can start transmitting its network state updates. Each Tap also can also select one (or more) Slave Sink(s) in order to provide enhanced resiliency to nodes failures. In the latter case, network state updates are still sent by the Tap to its Master Sink which, in turn, forwards them to the Slave Sink(s).

When a Tap node dies, no re-configuration is required. When a Sink node dies, the associated Tap nodes, by noticing that the update messages sent have not been acknowledged, tries to associate to the closest available Sink. If none of them is available within D_{TH}, the node –after a random back-off introduced to avoid collisions– promotes itself to Sink.

The pseudo–code for the autonomic role selection procedures is reported in Alg. 1. For the sake of simplicity, the interactions with Slave Sink(s) are not reported.

Algorithm 1 Autonomic role selection procedure.

```
1: Node.startTap()
2: Node.broadcastSinkDiscovery()
3: DiscoveryTimeout.start()
4: while DiscoveryTimeout.notExpired() do
5:     if Node.receive(SinkHello) then
6:         KnownSinks.add(SinkHello)
7:     end if
8: end while
9: KnownSinks.sortByMetric()
10: for Sink in KnownSinks do
11:     if Sink.getMetricFrom(Node) $> D_{TH}$ then
12:         Node.sendSinkAssociate(Sink)
13:         SinkAssociateTimeout.start()
14:         while SinkAssociateTimeout.notExpired() do
15:             if Node.receive(SinkAssociateAck) then
16:                 Node.setMasterSink(Sink)
17:                 Break()
18:             end if
19:         end while
20:         if Node.notHasMasterSink() then
21:             Sink = Node.startSink()
22:             Node.setMasterSink(Sink)
23:         end if
24:     end if
25: end for
```

2) Data Dissemination and Replication: In order to improve both spatial reuse and the resilience of the monitored information, the global network view maintained by the Sinks shall be properly disseminated and replicated across the network. In particular:

1) Network state updates generated by a Tap are delivered to the Sink(s) it is associated to. As Taps tend to associate to close-by Sinks, efficient spatial reuse is achieved. The value of managed objects is checked periodically by the Tap. Upon detection of a change, an update is sent to the *Master Sink*.
2) Network state updates received by a Sink and carrying global managed objects are disseminated to the other Sinks in the network (improving information availability). This allows the network administrator to access global managed objects for any Tap in the network directly from the Sink it connects to. On the other hand, accessing local managed objects requires the query to be redirected to the Sink that is currently managing the information, as described below (“Data Look-up”).

3) Network state updates received by a Sink and carrying Local Managed Objects are replicated on *Slave Sinks* (improving resilience). The rationale behind this approach is that if a Sink goes off–line its cluster of Taps is likely to be re–distributed among the closest Sinks.

The pseudo–code for the data dissemination and replication procedure is reported in Alg. 2.

Algorithm 2 Data dissemination and replication procedure.

```
 1: if Node.isSink() then
 2:     Node.connectToAllKnownSinks()
 3: end if
    {Tap's control loop}
 4: while True do
 5:     Node.Wait(UpdatePeriod)
 6:     for Object in ManagedObject do
 7:         if Object.isChanged() then
 8:             Node.sendTapUpdate(Node.getMasterSink())
 9:         end if
10:     end for
11: end while
    {Sink's control loop}
12: while True do
13:     if Node.receive(TapUpdate) then
14:         Node.updateRepository(TapUpdate)
15:         if TapUpdate.isGlobal() then
16:             Node.sendUpdateToAllKnownSinks()
17:         end if
18:     else if Node.receive(SinkUpdate) then
19:         Node.updateRepository(TapUpdate)
20:     end if
21: end while
```

3) Data Look-up: When asked for the value of a particular managed object, a Sink shall either query the local repository or it shall identify the Sink currently managing that object and route the query accordingly. A managed object can be found in the local repository of a Sink only in two cases:

1) The managed object is provided by a local Tap, i.e. a Tap that is associated to the given Sink.
2) The managed object is provided by a remote Tap but is marked as global, so it is replicated to all the Sink.

Otherwise, the query has to be routed to the appropriate Sink. As the association of Taps to Sinks is a global information, any Sink knows to which Sink the required Tap is associated to, and can therefore forward the request accordingly.

V. Performance Evaluation

In this section we aim at assessing the performance of the proposed architecture and protocols. A prototypical implementation of OBELIX, including all the features presented in the previous sections, has been developed and extensively tested in a 15 nodes wireless network testbed.

A. Implementation Details

The OBELIX framework is implemented in Python, a lightweight interpreted programming language. Within a single OBELIX–monitored node, up to three distinct software processes can be active at any given time: the *sink*, the *tap*, and the *communication channel*. All nodes in the OBELIX overlay run the *communication channel* process. Nodes that are also publishers and subscribers of monitoring information will also run the *tap* and the *sink* processes, respectively. A node can implement only data dissemination functionalities by running just the *communication channel* process. An SNMP–complaint interface is made available by *communication channel* in order to provide backward compatibility with existing network management systems. Standard TCP and UDP sockets are used to enable communications among processes.

B. Experimental Settings and Configuration

The software prototype has been experimentally evaluated over a real–world IEEE 802.11–based mesh testbed consisting of 15 multi–radio mesh routers deployed across three floors of a typical office building. Mesh routers are built around three different hardware platforms, namely the PCEngines ALIX 2C2, the PCEngines WRAP 1E and the Gateworks Cambria GW2358-4. Each node is equipped with two IEEE 802.11a/b/g wireless interfaces (Atheros chipset) with RTC/CTS disabled. Routers exploit OpenWRT as operating system. Mesh connectivity is provided by the WING routing protocol [19].

C. Evaluation Methodology

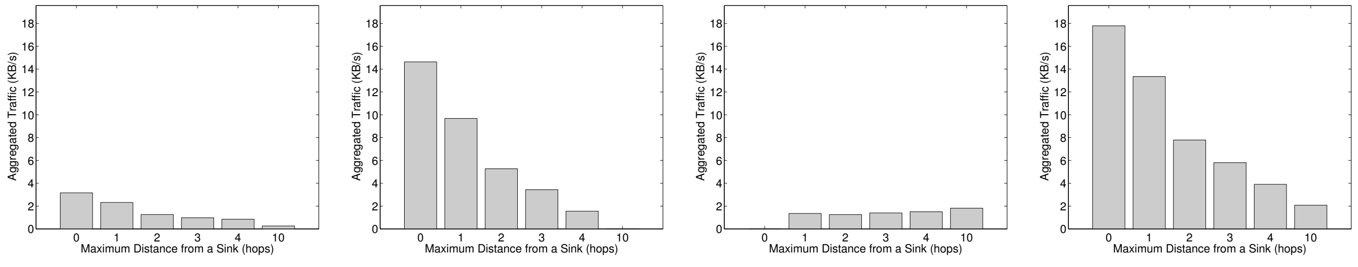
We run two types of test, aimed at estimating the traffic generated by the monitoring framework and the resilience of the system, respectively. In the experiments all messages exchanged between OBELIX entities were traced by logging the traffic handled by the *communication channel* process at each node. Log files were then collected and centrally merged and subsequently analysed.

1) Signaling Overhead: In this part we wanted to estimate the amount of signalling traffic generated by OBELIX. Each run had a duration of 4000 seconds. Different runs were performed for different values of the D_{TH} parameter, taken in the set $\{0, 1, 2, 3, 4, 10\}$. The first value corresponds to a situation in which all nodes run the Sink process. The last value forces the presence of one single Sink node.

2) Resilience: In order to assess the resilience of the system, we simulated failures of Sink nodes. At bootstrap, Sink nodes start a timer, uniformly distributed in the range $[180, 240]$ s. When the timer expires, all monitoring processes (Tap, Sink and Communication Channel) are killed. After 90 s, the Tap and Communication Channel processes are re-started, and the node enters the bootstrapping procedure. The recovery time was measured as the interval between the instant at which the Sink goes down and the instant at which all the Taps are again associated with another Sink. As the failure patterns are asynchronous, it may happen that at a given point in time multiple Sinks are failed.

D. Results and Comments

In order to get a qualitative insight into the behaviour of the overall system, we first reported the signalling traffic generated as a function of time for a specific run of our experiments. We considered the case of $D_{TH} = 3$ hops. The results are plotted in Fig. 4. In the plot we can identify the contributions of the various types of traffic. First, there is an initial traffic burst, which is due to initial `TAP_UPDATE` messages. At the beginning, indeed, Taps send the complete set of data to the Sink they are associated to. In the subsequent updates, only the values that changed are sent, leading to a much lower traffic. After the initial burst,



(a) Sink Hello (b) Sink Update (c) Tap Update (d) Total

Fig. 6: Average traffic generated by OBELIX as a function of the D_{TH} parameter value grouped by message type.

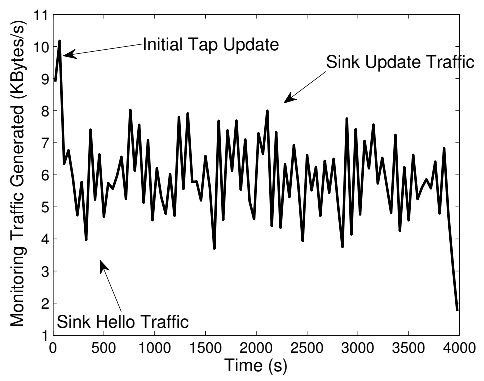


Fig. 4: Traffic generated by the monitoring framework as a function of time, $D_{TH} = 3$.

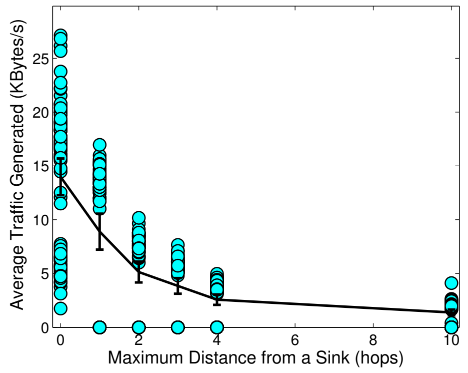


Fig. 5: Average traffic generated by OBELIX as a function of D_{TH}. Circles indicate values averaged over 60s intervals, continuous line indicates the mean value and the 95% confidence intervals.

the traffic shows a floor, which is mostly due to SINK_HELLO messages. The spikes that are present are due to the replication of information among sinks (SINK_UPDATE messages). The behaviour is similar for all other cases considered. (Of course, for the runs where only one sink was present no SINK_UPDATE traffic was present.)

In order to assess the impact of the number of sinks, we varied the D_{TH} parameter value. The results, reported in terms of average traffic generated, are reported in Fig. 5. Out of the 4000 s of log files, we removed the first 200 s (in order to get rid of the initial Tap update traffic) and the last 200 s (to avoid issues related to the flushing of pending messages). For each run, the traffic generated over intervals of 60 s was considered. The measured values are reported in the figure as circles. We also computed the mean value and the 95% confidence interval, and reported them in the figure as continuous line. As it may be seen, the average traffic generated tends to decrease with the value of D_{TH}. In this case, indeed, the number of sinks decreases, together with the need for replicating data across sinks. The variation is hence mostly due to the reduction of the SINK_UPDATE traffic. With values of $D_{TH} \geq 2$, the average traffic generated by the monitoring framework is below 10 KB/s. This shows the ability of the proposed solutions to make an efficient use of the wireless medium by effectively limiting the overhead traffic associated to the monitoring infrastructure. Further, as the number of sinks (which inversely relates to the value of D_{TH}) is an indicator of the robustness of the system, the graph can be used to understand the robustness/performance tradeoff offered by our solution.

We further analysed the composition of the traffic generated, classifying it on the basis of the type of message. Results are reported in Fig. 6. The vast majority of the traffic is generated by SINK_UPDATE messages, which are used for data replication and dissemination. The amount of traffic generated by TAP_UPDATE messages is not correlated with the D_{TH} parameter value, in that those messages are always generated by each Tap and sent to the closest Sink (except in the extreme case of $D_{TH} = 0$ where every node is a Sink). Finally, the traffic carrying SINK_HELLO messages decreases with D_{TH} parameter. Indeed, the higher the number of hops, the lower the number of Sinks in the network and thus the amount of SINK_HELLO messages generated by the beaconing infrastructure. The conclusion that these results allows us to draw is twofold. On the one hand, the amount of overhead generated by the monitoring overlay can be precisely controlled by the network administrator by appropriately setting the D_{TH} parameter. On the other hand, the overall amount of traffic generated by the monitoring overlay ranges from roughly 19 KBytes/s in the worst case ($D_{TH} = 0$) down to 2.5 KBytes/s when only one Sink is present ($D_{TH} = 10$).

Concerning the resilience tests, we first report in Fig. 7 a plot of the number of Sinks present in the network at any time instant for the case $D_{TH} = 3$. It can be noticed that, with the settings chosen, in a number of time instants no Sinks are present in the network. As such, the case represented here could well be considered a worst-case for a real system, where node's failures are expected to be rather rare events.

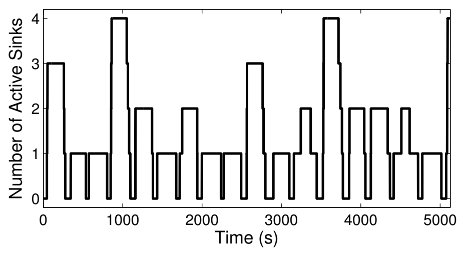


Fig. 7: Number of Sinks active in the network as a function of time, $D_{TH} = 3$ hops.

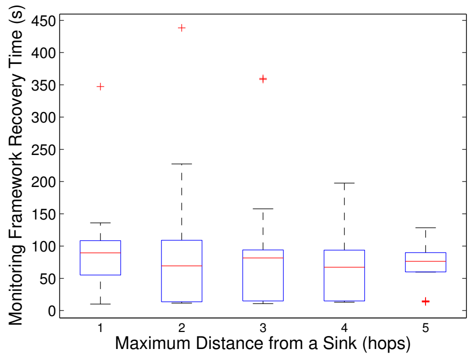


Fig. 8: Box plot of the recovery time as a function of the D_{TH} parameter value.

The results in terms of monitoring framework recovery time are reported, using a box plot[3], in Fig. 8 for various values of the parameter D_{TH}. As it may be seen, the recovery time turns out to be rather insensitive to the value taken by D_{TH}. The presence of extremely large values (300 s or more) in some runs is due to situations in which all the Sinks in the network go off-line at the same time. In such a case all Taps in the network try to associate to the other entries in their list of known Sinks. Each attempt implies the expiration of a timeout of 15 s. After getting through all known Sinks without getting a positive reply, a random back-off (uniformly distributed in $[0, 80]$ s) is generated before promoting to the Sink state. The joint effect of these timers is the cause of such high values for the recovery time. The setup of timers' value clearly represents a critical issue for OBELIX efficiency. Using lower values would speed up the recovery process, but may lead to instabilities in the network bootstrap phase.

VI. Conclusion

In this paper we have introduced an architecture and a set of protocols for building a distributed network monitoring framework. Design choices have been made to accommodate the peculiarities of wireless multi-hop networks, in terms of adaptivity, robustness and efficiency requirements. The proposed framework has been prototyped and experimentally evaluated on a 15–nodes wireless mesh network. Results show that the framework generates a limited amount of traffic, and that the system is able to consistently recover from node failures. The results can also be used to gain insight into the traffic overhead/robustness trade-off inherently present in our design (by means of properly tuning the D_{TH} parameter).

Directions for future research include the adoption of a more information–centric approach, whereby the whole monitoring framework becomes address-agnostic, and the use of optimised mechanisms for handling the replication of global monitoring information across sinks, leveraging —in a cross-layer perspective— knowledge on the underlying wireless technology employed.

[3]Bottom and top of the box represent the lower and upper quartile, respectively. The band within the box represents the median. The ends of the whiskers represent the most extreme data points not considered outliers. Outliers, represented with a cross, are identified as points whose distance from the lower/upper quartile exceeds 1.5 times the inter-quantile range.

References

[1] I. Akyildiz, X. Wang, and W. Wang, "Wireless mesh networks: a survey," *Elsevier Computer Networks*, vol. 47, no. 4, pp. 445 – 487, Mar. 2005.

[2] G. Pavlou, "On the evolution of management approaches, framework and protocols: A historical perspective," *Journal of Network and Systems Management*, vol. 15, pp. 425–445, 2007.

[3] R. Mortier and E. Kiciman, "Autonomic network management: Some pragmatic considerations," in *Proc. of ACM SIGCOMMM*, Pisa, Italy, 2006.

[4] S. Dobson, S. G. Denazis, A. Fernández, D. Gaïti, E. Gelenbe, F. Massacci, P. Nixon, F. Saffre, N. Schmidt, and F. Zambonelli, "A survey of autonomic communications," *ACM Trans. Auton. Adapt. Systems*, vol. 1, no. 2, pp. 223–259, 2006.

[5] J. N. de Souza and J. Strassner, "Self-organization and self-management in communications as applied to autonomic networks," *Computer Communications*, vol. 31, no. 13, pp. 2935–2936, 2008.

[6] A. Gonzalez Prieto, D. Dudkowski, C. Meirosu, C. Mingardi, G. Nunzi, M. Brunner, and R. Stadler, "Decentralized in-network management for the Future Internet," in *Proc. of IEEE ICC – Communications Workshops*, Dresden, Germany, 2009, pp. 1–5.

[7] J. Case, M. Fedor, M. Schoffstall, and J. Davin, "A simple network management protocol," IETF RFC 1157, May 1990, http://www.ietf.org/rfc/rfc1157.txt.

[8] W. Stallings, "Snmp and snmpv2: the infrastructure for network management," *IEEE Communications Magazine*, vol. 36, no. 3, pp. 37 –43, mar. 1998.

[9] L. Li, M. Thottan, B. Yao, and S. Paul, "Distributed network monitoring with bounded link utilization in IP networks," in *Proc. of IEEE INFOCOM*, San Francisco, CA, USA, 2003.

[10] Y. Bejerano and R. Rastogi, "Robust monitoring of link delays and faults in IP networks," in *Proc. of IEEE INFOCOM*, San Francisco, CA, USA, 2003.

[11] K. Suh and Y. Guo, "Locating network monitors: Complexity, heuristics and coverage," in *Proc. of IEEE INFOCOM*, Miami, FL, USA, 2005.

[12] S. Jamin, C. Jin, Y. Jin, D. Raz, Y. Shavitt, and L. Zhang, "On the placement of Internet instrumentation," in *Proc. of IEEE INFOCOM*, Tel–Aviv, Israel, 2000.

[13] J. Postel, "Internet control message protocol," IETF RFC 0792, Sep. 1981, http://www.ietf.org/rfc/rfc0792.txt.

[14] R. Enns, "Netconf configuration protocol," IETF RFC 4741, Dec. 2006, http://www.ietf.org/rfc/rfc4741.txt.

[15] M. Montemurro and D. Stanley, "Control And Provisioning of Wireless Access Points (CAPWAP)," IETF RFC 5415, Mar. 2009, http://www.ietf.org/rfc/rfc5415.txt.

[16] K. Ramachandran, E. M. Belding-Royer, and K. C. Almeroth, "DAMON: A distributed architecture for monitoring multi-hop mobile networks," in *Proc. of IEEE SECON*, Santa Clara, California, USA, 2004.

[17] R. Raghavendra, P. Acharya, E. Belding, and K. Almeroth, "Antler: A multi-tiered approach to automated wireless network management," in *IEEE INFOCOM Workshops*, Phoenix, AZ, USA, 2008.

[18] S. Schuetz, K. Zimmermann, G. Nunzi, S. Schmid, and M. Brunner, "Autonomic and decentralized management of wireless access networks," *IEEE Trans. Netw. and Serv. Management*, vol. 4, no. 2, pp. 96 –106, Sept. 2007.

[19] R. Riggio, T. Rasheed, S. Testi, F. Granelli, and I. Chlamtac, "Interference and traffic aware channel assignment in wifibased wireless mesh networks," *Elsevier Ad Hoc Networks*, vol. 36, no. 3, pp. 37 –43, mar. 1998.