ORIGINAL ARTICLE

A Big Data and Machine Learning Approach for Network Monitoring and Security

Leonardo Maccari[1] | **Andrea Passerini**[1]

[1]Dept. of Information Engineering and Computer Science, University of Trento, Italy

Correspondence
Leonardo Maccari
Email: leonardo.maccari@unitn.it

Funding information
This work has been partially funded by the European Commission, H2020-ICT-2015 Programme, Grant Number 688768 'netCommons' (Network Infrastructure as Commons)

In the last decade the performances of 802.11 (Wi-Fi) devices skyrocketed. Today it is possible to realize gigabit wireless links spanning across kilometers at a fraction of the cost of the wired equivalent. In the same period, mesh network evolved from being experimental tools confined into university labs, to systems running in several real world scenarios. Mesh networks can now provide city-wide coverage and can compete on the market of Internet access. Yet, being wireless distributed networks, mesh networks are still hard to maintain and monitor. This paper explains how today we can perform monitoring, anomaly detection and root cause analysis in mesh networks using Big Data techniques. It first describes the architecture of a modern mesh network, it justifies the use of Big Data techniques and provides a design for the storage and analysis of Big Data produced by a large-scale mesh network. While proposing a generic infrastructure, we focus on its application in the security domain.

1 | INTRODUCTION

Wireless mesh networks have attracted a large attention from ICT researchers in the past decade, but today, they are not on the edge of the research agenda. This is due to some disappointment after the high initial expectations on this technology, but also to the simple fact that in some fields this technology is now mature enough; it become a product. Two cases are generally mentioned as positive examples for the application of mesh networks, bottom-up networks and industrial applications.

The first case deals with Community Networks (CNs), networks built with a bottom-up approach by communities of people that set-up their Internet infrastructure generally (but not only) to overcome digital divide. As the 802.11 technology has achieved a quantum leap in the last decade (from 60 Mb/s to 1Gb/s for roughly the same price, headed to 100Gb/s [1]) CNs today can give connectivity to thousands of people with just one mesh network [2, 3]. Internet access with mesh networks is becoming economically viable also in non-market failure areas. Small ISPs are starting to use this technology[1] which seems to be a competitive model even against other established options.

The second case is given by the industrial field, in which companies like Tropos Networks (now part of the ABB service company) use mesh network to monitor industrial infrastructure in remote, mainly disconnected areas[2]. In section 3 we describe a modern mesh node (a composition of high-speed wireless devices, as opposed to the still common perception of the single broadcast radio that was introduced in the early 2000s) and explain how a mesh network with recent technology (802.11ac, or even 802.11ad) can serve thousands of people.

What did not change in the last decade are some of the drawbacks of using a mesh network, mainly, the complexity of its management and its vulnerability to external factors. A mesh network made of outdoor nodes is susceptible to many sources of disruption, physical damage, power outages, physical obstruction, security attacks due to the shared media etc. Moreover, every node not only produces and receives data, but also routes data from other nodes, so that it is paramount to use proper security means to prevent tampering and privacy intrusion. As a result, we have networks that can easily achieve an aggregated throughput in the order of Gb/s, with a very dynamic and still challenging management model.

One of the most important network functions is monitoring, and specifically anomaly detection and root-cause analysis [4]. In a hierarchical wired network under the control of a single manager, there are many instruments that can be used for this purpose. In a wireless mesh network instead, this is still an open field.

This paper proposes the use of Big Data techniques to perform anomaly detection and root cause analysis in wireless mesh networks. It elaborates on the possibility of performing data collection on mesh nodes, and transporting data from nodes to one, or more, network monitors. These monitors will reconstruct the stream of data and analyse it with current state-of-the-art Big Data techniques, in order to understand the cause of a certain anomaly, which can be due to a large number of apparently uncorrelated events.

We explain how Big Data technology can be used to gather and process data, how data can be organized to better represent the scenarios, and what are the best scientific knowledge we own in order to solve them, primarily based on Machine Learning approaches. We describe how data from the network can be exported to a remote cluster based on Apache Spark, which is one of the most popular open source Big Data platforms available. We show how this technology fits the purpose of monitoring a high-performance mesh network. We show also that coupling a remote cluster with a Big Data approach is necessary in order to achieve scalability and performance, even in a distributed mesh network that generally does not fit with a centralized approach. While we make examples of specific machine learning techniques that can be applied to this domain, it is out of the scope of this paper to contribute with detailed proposals in this field. The goal of this paper is to motivate the use of Big Data techniques for mesh networks monitoring, and to provide an architecture that can be used to achieve this goal.

2 | STATE OF THE ART

The relationship between Big Data and network performance is generally explored to understand what is the best way to have networks that support Big Data applications [5, 6].

[1]See `https://www.common.net/` in the USA or `http://www.roonet.it/2017/07/17/common-net-internet-ultraveloce-a-kilometro-zero/` in Italy, for instance

[2]See `https://new.abb.com/network-management/communication-networks/wireless-networks`

Few are the examples of works that investigate how Big Data techniques can be used to perform network monitoring or root cause analysis, directly targeted to the improvement of the network infrastructure. Among them Cui et al. and Trevisan et al. [7, 8] explore how Big Data can be used to perform traffic engineering and classify specific kinds of traffic when using Software Defined Networking (SDN). SDN assume to have a centralized controller that takes decision at the management plane, therefore, coupling SDN with Big Data techniques to take automated decisions is a natural next step. Yet, SDN is used in data centers but in a mesh context it is still not a practical approach, because there is no actual separation between the data and the management plane. A mistake in the management plane can isolate a part of the network that is physically separated from the rest, and require manual intervention from a human operator. Some mesh network results exist but generally assume an always working and reachable network [9], which may not be the case in a general purpose mesh network. Big Data techniques have been proposed to gather data from the access network, especially mobile cellular networks [10, 11], but these techniques do not apply to the management of the network itself.

Root Cause Analysis (RCA) is a well studied subject, in particular when coupled to network monitoring and data logging. When a system is monitored there are several sources of information one can use for cause analysis, spanning from network measures [12] down to daemon logs [13]. To perform such analysis there is a need for a large amount of data which in the past was approached in several ways, for instance, with local processing [14] or with distributed data collection [15]. Yet, cross-layer analysis is generally hard to carry-on especially in multi-domain networks [16].

In the security domain Big Data already play an important role. Intrusion detection is one typical application [17, 18] botnet detection is another [19] as well as DDoS mitigation [20]. None of the existing approaches deal with network monitoring.

Finally, architectures for Big Data collection and evaluation have been proposed in various areas, but are specific to those areas [21, 22, 23].

3 | A MODERN WIRELESS MESH NETWORK ARCHITECTURE

In the scientific literature, mesh networks are typically imagined as a collection of nodes equipped with omnidirectional antennas at a distance of a few hundred meters. Modern mesh networks are much more complex than that, as they use specialized hardware and some configurations that make them much more robust and performing. In this section we describe a mesh node for residential connections as used in many community networks [2].

A mesh node is generally split in two parts, one outdoor, generally mounted on a roof or on a terrace and another inside the house of the end-user. The outdoor part is made of a mount pole, where multiple wireless devices are mounted. These devices are full-fledged wireless routers designed for the outdoor, they are powered over Ethernet and equipped with wireless radios and antennas. When using the 802.11n/ac standard they work on ISM frequencies, but other proprietary frequencies can be used. Recently, the first outdoor mm wavelength devices working on the 60GHz band hit the market. Depending on the technology, and on the kind of antenna they use, these devices can support up to (nominally) Gbit/s traffic and their price is very low: an 802.11ac device for multi-km links or a Gb/s device for links in the range of the hundreds of meters can be bought for less than 70$.

As said, the devices are powered over Ethernet, and here there are two options. The first option is to have power directly on the roof. This happens in many common situations, but when it is not possible, the power needs to be carried from the user's house. Bringing Ethernet cables from the indoor to the outdoor may be tricky (conduits need to be present and free), so generally, one cable that encapsulates an Ethernet cable and a standard power cable is used. This is the solution depicted in fig. 1, in which the power source is indoor and only a single cable reaches the outdoor.

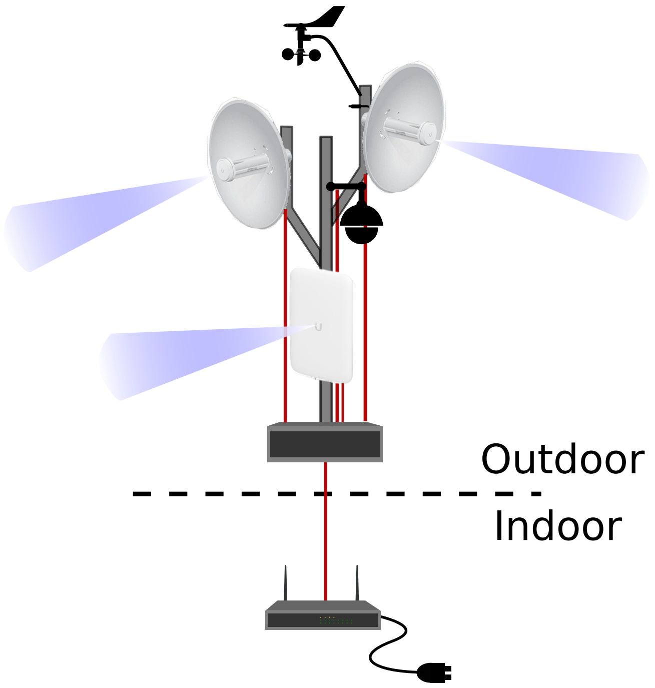


FIGURE 1 An example of a modern mesh node.

This can be just one Ethernet cable with PoE, but the limited maximum power restricts the amount of devices that can be installed, so a better solution is to use two different cables entangled. In the outdoor the cables enter into a splitter, which can also be coupled with an UPS. Note that this device is technically an outdoor gigabit switch (again, the market offers devices for less than 100$). Once power is on the roof, the node can be integrated with other devices. For instance, a surveillance cam can be used both for security reasons, but also to do remote monitoring. In fact, outdoor nodes are subject to failures, wind can move the devices and something that impede the communication (i.e. another antenna) can be placed nearby. Cams helps to monitor the state of the node remotely and troubleshoot such situation. A weather station can also be placed, in order to monitor the surrounding conditions (wind, temperature, humidity). Again, these parameters can be used to gather insights on the state of the node, especially when working on mm wavelength that are influenced by weather conditions.

The intelligence of the node generally resides in the home router, it's the home router that runs a routing algorithm, and routes packets from one device to another. For this reason it is paramount that the devices support gigabit Ethernet, as the cable going to the roof needs to route messages and it would otherwise be a bottleneck. The intelligence is placed in the home router because it is the device that is easier to intervene on, with a simple reboot, a reset or a full re-flash if needed. The user can perform these actions by himself, and does not need to go on the roof to do basic procedures.

4 | DATA COLLECTION

Given the description of a full-fledged wireless mesh network we made in section 3, in this section we provide a rough estimation of the amount of monitoring traffic that this network may produce. For monitoring traffic we intend the traffic that is needed to monitor the behaviour of the network. Following the typical layering of networks, for each layer we describe the kind of data and the data rate we expect to have.

For this estimation we need to do some assumptions on the composition and the size of the network:

- The network is made of N nodes with $N \in [100, 1000]$ and we assume that there is one gateway every 100 nodes. We have analysed networks of comparable size, so we think this is a realistic choice [2]
- Every node is made of d wireless devices used for the backbone links.
- Every node serves H hosts that access the Internet through that node. Every node concurrently forwards C transport-layer connections (such as TCP connections) active at the same time, generated from these hosts.
- Every node routes (i.e. it receives on one device and reforwards on another device) P packets per second (which, at the optimistic size of 1500B per packet produces an average of B b/s).

As it is common in networking protocols, in order to quantify the bit-rate of the generated traffic we assume 4 Bytes are used to transmit a float value [24]. The next sections use these parameters to introduce formulas that we later on use to evaluate the overall data rate of the monitoring traffic, in order to assess if it is actually reasonable to treat this problem with a Big Data approach. For each layer we introduce the base parameters, and some optional parameters (marked with an asterisk) that could produce even higher data rates. For each parameter we estimate its size and the sampling frequency necessary to use it effectively. We also add some examples of how how the data from a each layer can be used to address a specific problem in the network, with special attention to security issues.

4.0.1 | Physical Parameters

This section deals with the control traffic that is generated by the sensors that monitor the proper functioning of the node. Among the components that need to be monitored we mention:

- Power management and power consumption with PoE. One sample per device with a frequency of 10Hz. It is important to have frequent samples in order to correlate the consumption with the traffic rate, which can be extremely volatile.
- Physical connections (Ethernet ports). One sample per device, once per second. Contains the negotiated bit rate, which can change if the cable is damaged, or the ports have defects.
- UPS signals. UPS monitors the line tension and current both in input and output, the battery charge and the line frequency. Again, this needs frequent logging to visualize spikes (10Hz).
- Physical orientation of the devices. Devices can have compasses that sample the orientation of a directional antenna. This is key to understand if for some reason the antenna was moved from its original position (due to physical damage or wind). We expect two samples per antenna, once per second
- Weather Station*. Can be useful for reporting environmental parameters like outdoor temperature, humidity, pressure, wind speed, in order to correlate eventual failures with these parameters. We expect one sample per each parameter, once per minute.
- Security Cam*. This is useful for two reasons, the first is the obvious perimeter protection, the second is because

Description	Data Size (Bytes)	Frequency (Hz)
PoE	$d \times 4$	10
Ports	$d \times 1$	1
UPS	6×4	10
Direction	$d \times 8$	1
Meteo*	12	1/60
Camera*	$500K$	1

TABLE 1 The estimated control traffic generated by the physical monitoring system.

Description	Data Size (Bytes)	Frequency (Hz)
Tx/Rx Pow. and MCS	$4 \times 2 + 2$	$2 \times P$

TABLE 2 The estimated control traffic generated by the communication physical communication physical layer.

when the operating frequency is in the mm wave length range, the weather conditions alter the performance of the links.

An example application of these parameters is the following. There are conditions in which a wrong configuration of the power source may cause devices to periodically reboot. For instance, if the power supply is not correctly dimensioned, the devices may reboot when there is a traffic peak that raises the power consumption beyond the available power. Professional devices may reboot in a few seconds, and the routing tables may be repaired quickly when the device boots again. This would produce a loss of packets for some seconds that would kill the performance of the link and possibly introduce route flapping (the phenomena in which the routing layer “flaps” the next hop from one node to another for a set of destinations), for no evident reason. Monitoring power consumption and traffic may help to correlate the two events (traffic peak and power outage) to detect this problem.

Looking at table 1 we can estimate the traffic data rate (in Bytes per second, excluding optional entries) as in eq. (1):

$$D_{phy} = 10 \times (d \times 4) + d + (6 \times 4) \times 10 + d \times 8 \tag{1}$$

4.0.2 | Communication Physical Layer

The communication physical layer of modern wireless protocols is extremely complex. If we want to shrink it to the minimum (excluding details on MIMO and beamforming) for every forwarded packet, a received and a transmitted power value is generated, plus the negotiated bit-rate (MCS).

An example application of these parameters is when the link suddenly degrades because of a new source of interference in the same frequency, which is hard to track if not generated by the same communication protocol. We can estimate the traffic using the data in table 2 and eq. (2):

Description	Data Size (Bytes)	Frequency (Hz)
802.11e queues	7×4	10
Clients & SA	$H \times 128$	1/10
ARP Tables	$d + H \times 10$	1
MAC Packets*	6×2	$1/P$

TABLE 3 The estimated control traffic generated by the MAC layer.

$$D_{comm\_phy} = (4 \times 2 + 2) \times P \tag{2}$$

4.0.3 | Media Access Control Layer

The MAC layer of the wireless protocol takes care of access to the wireless media and again, MAC layers of real devices are getting more and more complex. Moreover, MAC layer performs many more tasks than packet scheduling, for instance, it is responsible of client authentication.

For the sake of simplicity we consider only standard features of 802.11, and among the many data sources we consider two of them: i) the fill level of queues in the MAC layer (7 queues according to IEEE 802.11e, sampled at 10Hz) ii) the number and kind of clients that have an active security association (128B at 0.1Hz). Additionally, we consider also ARP tables, which bridge the MAC and IP layer and are very important to detect ARP spoofing attacks (10B per entry, at least $d + H$ entries, at 1Hz).

An example application of these parameters is the case in which two different applications have a very different performance on the same path, probably due to some misconfiguration in the priority queues. This may require balancing the QoS parameters on the link. Also, a security warning may be raised when a node has too many connected hosts or some hosts are connected to multiple nodes.

Looking at table 3 we can estimate the traffic data rate as in eq. (3):

$$D_{MAC} = 7 \times 4 \times 10 + H \times 128 \times 0.1 + (d + H) \times 10 \tag{3}$$

To have an even finer logging, each received packet can be logged together with its source MAC address and destination MAC address, which is considered in table 3 as an optional source of control traffic.

4.0.4 | IP Layer

Monitoring the routing layer is a very difficult activity. Historically, debugging the routing protocol has been done using applications like `ping` and `traceroute`, but this does not help to debug past conditions. One approach that can be used to detect (temporary) loops or route flapping is to dump the routing table of every node very frequently, and to collect the dumped routing tables, in order to be able to reconstruct the network situation at some moment in the past, and emulate the path that a packet would have traversed at any point in time. For this to happen we need to dump the routing tables of a node (every entry made of one destination IP, one next hop IP both for IPv4 and IPv6,

Description	Data Size (Bytes)	Frequency (Hz)
Routing Table	$N \times (8 + 32 + 4)$	2

TABLE 4 The estimated traffic generated by the routing layer.

Description	Data Size (Bytes)	Frequency (Hz)
Connections	$C \times 50 \times (4 \times 2 + 2)$	1

TABLE 5 The estimated control traffic generated by the transport layer.

plus some link quality metric value). Sampling the routing table every half a second is the minimum sufficient to spot temporary loops. Since every node should have a routing table entry for every network attached to every other node, we have table 4 and eq. (4):

$$D_{routing} = N \times (8 + 32 + 4) \times 2 \tag{4}$$

4.0.5 | Transport Layer

The router activities related to the transport layer are Network Address Translation (NAT) and firewalling for all the connections generated and destined to the hosts of the node. It is impossible to give an average number of running connections per host, as the usage of wireless networks is extremely variegate [25]. We know that several operating systems and appliances (such as MS Windows and Office) open several connections towards their servers, and keep them up for all the up-time of the user session. Moreover, each browser tab (or phone apps) keeps at least 2/3 connections alive, and other Internet-based applications are running in the background (antivirus, mail agent etc). As a rule of thumb we consider every user to generate about 50 active connections. For each connection two IP addresses, two TCP ports and some state information must be kept (not only TCP state, but also bit-rate generated, match on some filtering rules etc). We consider one sample per second sufficient.

An example application of these parameters is the detection of Denial of Service attacks, in which a client starts generating a large number of connections towards a specific victim. Note that this in a wireless mesh network can quickly escalate to saturating one link, or one path to a gateway. In this case the routing protocol will make its decision, and, detecting that one gateway is having bad performance, it may switch to another gateway. As soon as this happens the available bandwidth in the new gateway will drop, while the old one will improve, which may introduce route flapping. In general, "flashcrowd" situations are hard to manage and in a mesh network they can introduce network-wide instability.

Looking at table 5 we can estimate the following traffic as in eq. (5):

$$D_{tra} = C \times 50 \times (4 \times 2 + 2) \tag{5}$$

4.0.6 | Security Alerts

In the security domain there are plenty of applications that perform monitoring of networks and application. Among the most classical (and least invasive ones) are traffic sniffers that try to detect misbehaviour at various levels, for instance:

- The wireless intrusion detection/prevention systems generally listen for suspicious activities, for instance, new APs that use the same name of the current one, storms of packets of some kind, or brute force attempts on authentication protocols. At the same time they can signal the presence in the user network of devices configured with insecure parameters.
- At MAC layer suspicious activities can be flagged as ARP spoofing attacks, that are still feasible today.
- At network and transport layer we still have SYN floods, reset attacks, DNS spoofing, and attacks to the routing protocols that can be detected using monitors.
- Finally, at the application layer there are today very sophisticated appliances to monitor entire systems. An host IDS monitors in real time the behaviour of an user, and raises flags if the user is trying to install packages that are known to be malicious, or if it is attacked by worms and viruses. These appliances are generally targeted for real time monitoring in corporate networks.

Many of these systems exist, for instance Palo Alto Magnifier analyses traffic and, after a period of training is able to raise alerts on suspicious activities [26]. Of course these appliances collect all the traffic from the gateway and analyse it, but are only partially useful to understand where inside a mesh network some suspicious activity started. Other host-level systems collect logs from any host and apply Big Data techniques on top of them, for instance, the ELK stack (that stands for Elasticsearch, Logstash and Kibana) is frequently used by security firms.

The data rate for this kind of data is impossible to predict, as every vendor is different from another and it is impossible to guess how much traffic they can generate. For this reason, we don't focus on this layer and just consider the traffic described in previous sections. Indeed, the architecture we propose will use also the data coming from these security monitors.

4.1 | Data-set Size

If we look at the data described so far, we can estimate the bit-rate for control data generated by one node as the sum:

$$D = D_{phy} + D_{comm\_phy} + D_{MAC} + D_{routing} + D_{tra} \tag{6}$$

To plot this equation we need to add some further consideration in order to have some realistic values for P. First of all we recall that we expect to have $N \in [100, 1000]$, and every node to have in average 3 devices. In the case these devices can sustain a nominal 1Gb/s throughput, we consider a realistic medium efficiency of 60% and a maximum occupation of each link of 80%. This yields roughly 500Mb/s per link. If every gateway has 3 devices, then one gateway can realistically sustain 1.5Gb/s. We assumed one gateway per 100 nodes, which yields a maximum of 15Mb/s per node. This is the *minimum* guaranteed bandwidth per node, and as such, it is orders of magnitude higher than what a typical ADSL provides. Commercial ISPs tend to oversell their capacity with a contention ratio that can reach even 50 times the minimum bandwidth, so we consider our scenario more than credible.

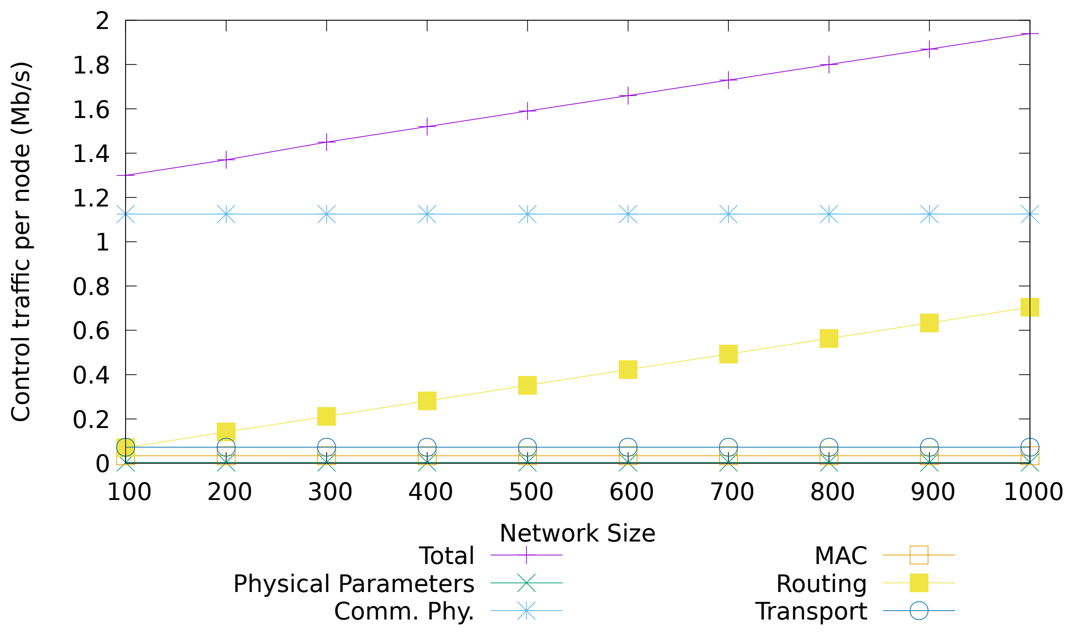


FIGURE 2 Expected control traffic data rate generated by one node.

Note that if $N = 1000$ and we have one gateway per 100 nodes, the network is still one large network, with the advantage of offering many gateways per node, and thus an increased resilience. Yet, to reach a gateway the traffic from node n_i will have to go through a network of roughly 100 nodes, and not through a network of 1000 nodes. How many hops are necessary to reach a gateway in average? We imposed that every node has in average d radios, and practical considerations limit the maximum number of devices to a number close to d. Thus, we expect the network to behave similarly to a random Erdos graph [27], and not as a long-tailed graph in which some nodes have a degree orders of magnitude higher than others. A well known result for Erdos graphs tell that the average distance between two nodes is given by $\lceil log_d(N) \rceil = \lceil log_d(100) \rceil = 5$. If every node produces at most 15Mb/s this traffic must be delivered on a 5-hop path, that sums up to 75Mb/s of traffic generated on the network by each node. In the fortunate case in which all packets are 1500B long, this yields $P \simeq 7000$. We have now all the parameters to plot eq. (6) as shown in fig. 2.

Figure 2 shows that the largest part of the control traffic is generated by the communication phy. layer and the routing layer, with the first one depending only on P and the second one growing with the number of hosts. The total data rate ranges between 1.3 and 1.94 Mb/s. Figure 3 reports the total control data rate per second generated by all the nodes in the network, which eventually scales more than linearly with the network size. This result shows that even with a relatively small network of 100 nodes, we can generate roughly 7.8Gb per minute, and 468Gb per hour. These values skyrocket to 116Gb per minute and 6.9Tb per hour in a network with 1000 nodes. In practice, the only way to treat such a large data-set is with the use of Big Data techniques. Consider that in this estimation we excluded some factors that could severely increase the data rate:

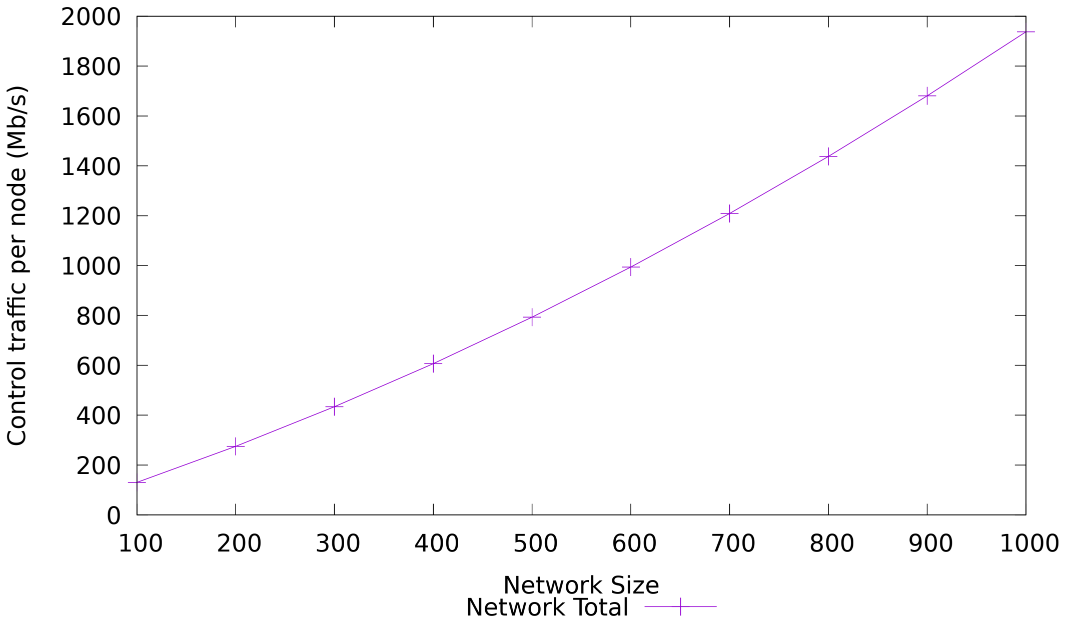


FIGURE 3 Expected control traffic data rate generated by the whole network.

- We did not consider the video of surveillance cams, which can generate at at least 0.5Mb/s each (depending on the desired quality).
- We did not include any security appliance.
- We could have included a more detailed logging at many layers. For instance we could include per-packet logging, both at the MAC layer (including the last line of table 3) but even at the IP layer.

The fact that Figure 3 is more than linear introduces another issue that we need to take into account, that is, the horizontal scalability of the computing resources we need for the monitoring system. A mesh network grows in a loosely planned way and has acceptable performance as long as the ratio of nodes per gateway is upper bounded. If a new end-user joins the network, the topology changes, and that new node can become the gateway to reach a set of new nodes. Networks can also merge or split, based on the addition or removal of nodes. These dynamics are generally slow and it is the task of the network manager to maintain a certain level of graph connectivity in order for the graph to be k-connected (with k being a design choice), but indeed, under controlled circumstances the topology can change abruptly. For instance at some point, the administrator can turn on a few nodes merging two separate meshes in only one network. When this happens, the dimension of the data set grows more than linearly, and thus, the sum of the resources used to monitor the two initial networks are not sufficient anymore. Apache Spark, coupled with Apache Hadoop is able to offer the necessary horizontal scalability in order to follow evolution of the network topology. Note that Spark can be coupled with several other Big Data platforms, Hadoop, Mesos Kubernetes, to name a few. Along the paper we refer to Hadoop but the same considerations are valid for an other choice.

Finally, it is fundamental to observe the difference between the size of the data-set and the amount of control data transmitted over the network, which is much lower. First of all, in the rest of the paper we describe two approaches, one that collects data with a very high granularity and can create large data sets, another one in which data are averaged on time windows. This second method can be used to perform high-level analysis if the capacity of the links does not make it possible to adopt the first method. Second, whatever the collection method used, the majority of the data that we described so far are slowly changing: for instance, routing tables are going to change relatively slowly during normal operations. This means that data can be cached and compressed and the amount of data that is effectively transmitted on the network from one node to the Big Data cluster is way smaller that what shown in fig. 3. Note also that this approach naturally adapts to network conditions, if the routing tables do not change (so no anomaly can be recognized at the routing layer) data can be strongly compressed. When the routing tables vary with high frequency instead, that is the probably the case in which an anomaly could happen, and the bit-rate of the control traffic will increase. Nevertheless, when reconstructed, the data-set will inflate to the size we estimated and still needs to be treated with Big Data techniques.

5 | DATA FORMAT

We envision that every layer in each node will produce a single unstructured data stream, which will then be collected and analysed together by the Spark cluster. The cluster will be responsible for organizing data in a way that makes analysis feasible, both in real time and off-line. The natural representation of a network data-set is a graph-based one, which we imagine to be possible at two levels: in an aggregated form, better suited for fast in-line analysis and anomaly detection, and with the full data-set, imagined for post-processing.

Figure 4 presents a possible way of organising the data-set. On the left there is the physical network, with each node generating a set of data streams, one for each parameter we identified. Once the streams are transmitted to the cluster we consider two ways in which the cluster can organize them.

In one case, the streams are divided in chunks, each chunk corresponding to a time-window. Each chunk is averaged and timestamped and contributes to create a data-graph, each data layer corresponds to a separate graph with different characteristics. For instance, the physical monitoring data is generally confined to the internal representation of a node, and has no notion of edges between nodes[3]. In the picture we represented it as a layer, but it could actually be embedded as attributes for the nodes in the other layers. At the physical layer instead, each node in the graph is a radio, that is connected with a directed edge with other radios from which it receives packets. At the MAC layer, each node is a MAC address, and each edge represents a relationship at the MAC layer (like AP/client, ad-hoc link, or any other kind of link that 802.11 can provide). At the IP layer each node is an IP and an edge is present between two IPs if one has received traffic from the other and they are at one hop distance. A graph can also be defined at the transport layer, using hosts as nodes and adding edges if there is an ongoing traffic flow between them. Note that while the nodes of the graph at a certain level correspond to a group of nodes at the lower level (a network host is made of several IPs, to which correspond MAC addresses, to which correspond radios…) there is not a hierarchical relationship between the edges at various levels. A radio in a node may receive packets from neighbors for which there is no link at the MAC layer. Similarly, the presence of a possible link at the MAC layer does not necessarily imply a possible “link” at the routing layer. Finally, the fact that there is a link at the routing layer does not necessarily mean that at the transport layer there is an ongoing communication between two nodes. The data graphs are then different, even if they have similarities.

[3]Note that we use the term *edge* when we refer to an edge in graph used in the data-set, and we use the term *link* when we refer to some kind of connection in the real network, at any layer.

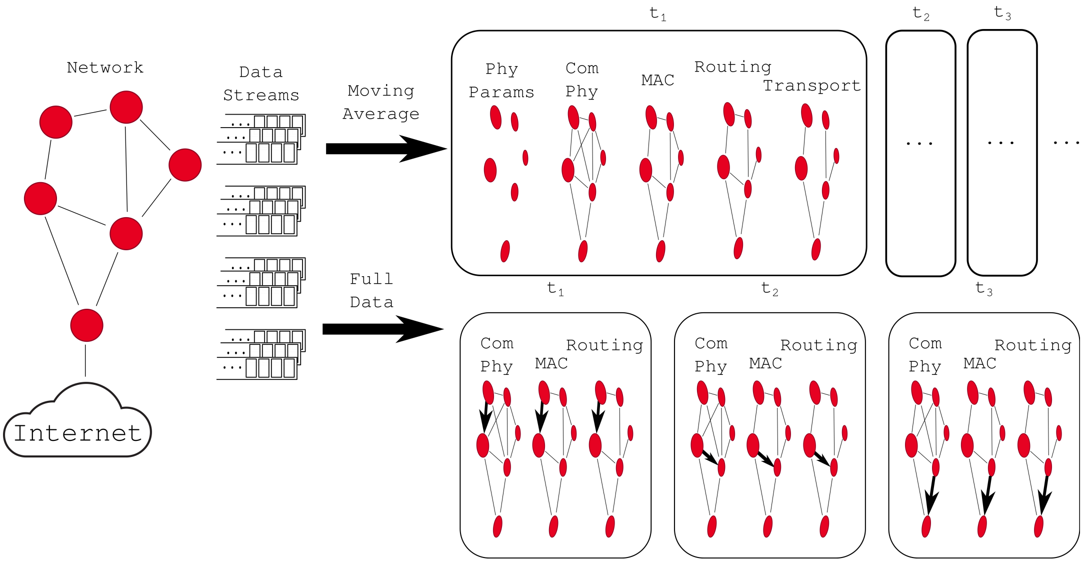


FIGURE 4 An example for the data format. From left to right: physical network, buffers of data to be used in two ways: aggregated graphs based on average values (top) and exact graphs representing all the traffic exchange between nodes (bottom).

In this kind of representation, a large part of the information is lost, because each node in each layer includes only some aggregate value derived from the original ones. Yet this representation can be used for anomaly detection at a coarse level. A more fine-grained way of inspecting the data is to use the raw data and correlate the events. An example is to track all the data related to a specific packet going from its source to its destination. The packet is identified by addresses at every layer so at every hop it is possible to correlate data from one layer to another, and the links used on its path in the network can be annotated by all the relevant metrics. This kind of analysis, aggregated by flows or by source-destination couples can be used to debug local anomalies that are impossible to detect using aggregate averages.

5.1 | Usage Examples

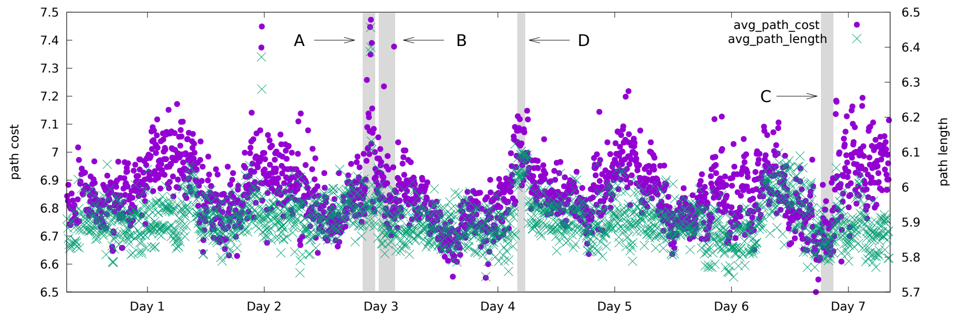
We give two examples of potential use of the data-sets described in the previous section, in order to highlight their difference. Consider the case in which a source of interference is activated close to a node. This will decrease the signal-to-noise ratio measured on some link l, which will produce an anomaly in the communication physical layer. This information will take some time to be transported at the MAC layer, as the MAC layer will detect loss of packets, and will gradually decrease the bit-rate for packets sent on the link in order to use a more robust encoding that can stand the new (lower) SNR. This may take hundreds of milliseconds, depending on how many packets are sent on l. The routing layer will instead take seconds (possibly tens of seconds, depending on its configuration) to change the value of the routing metric for l. At some point, the routing layer may reconfigure the routing tables in order to route around l and use an alternative path for the existing connections. This may saturate some other link and impact the traffic of an user that experiences a poor service. The important thing to note is that l and the node used by the user that experiences the malfunction could be in two distant locations, so that inferring a causal relationship between the two events could be all but easy for a human operator. Since this is a key element to understand the complexity of the management of a mesh network, in section 6 we use real data from an existing network to describe how complex such task can be. This chain of events can be detected in real-time and countermeasures can be taken before the user experience is impacted. For instance, the interfered node could decide to jump to another frequency that is less crowded than the current one. For this to happen we need anomaly detection algorithms, that not only detect a problem (a link is saturated) but, given a sufficiently large time window, are able to perform root cause analysis in order to tell what is the first anomaly that initiated the chain of events.

A second example is a security-related one. Imagine that a user was able to gain administrator privileges on the node that connects him to the network. This of course should not be possible, the user should be somehow in control of his home router, but not to the level of interfering with other people's traffic. In this case, he exploited a zero-day attack to the router software and he is now able to perform a man-in-the-middle attack, deflecting some traffic to his PC and then injecting the traffic in the network again. The user is not interested in all the traffic, but will actually try to intercept only traffic that contains sensitive information. A typical example is that the attacker is performing man-in-the-middle attacks only on websites that have a badly configured certificate (unfortunately there are still many of them). Users already receive a warning when they connect to the website and they still use it. The attacker can exploit this bug setting-up a proxy that intercepts the HTTPS connection, gives a fake certificate, and steals the user credentials. The result of this is that for only some connections, for no apparent reason the performance will drop, as the traffic is diverted, passed to the attacker, analysed and injected back in the network. This may take tens/hundreds of milliseconds, which is not enough for the users to note. Even the first approach we described, with aggregated data will not detect this event. If instead the traffic flow can be fully reconstructed, this event can be detected. In fact, following the sequence of packets in the network and inspecting the data produced by all the layers, it would be clear that even if at all layers the quality of the links as reported by the graph edges is high (routing quality, MAC bit rate, queues occupation etc.), one specific forwarding node introduces an unexpected latency for some kind of connections. This would raise a warning, which could be confirmed by analysing the physical parameters, which will tell that everything is OK (i.e. no broken Ethernet ports or cables), and finally raise an alarm. Of course to perform this kind of check, we need to be able to trace packets going through the network at every hop and correlate packets at all layers.

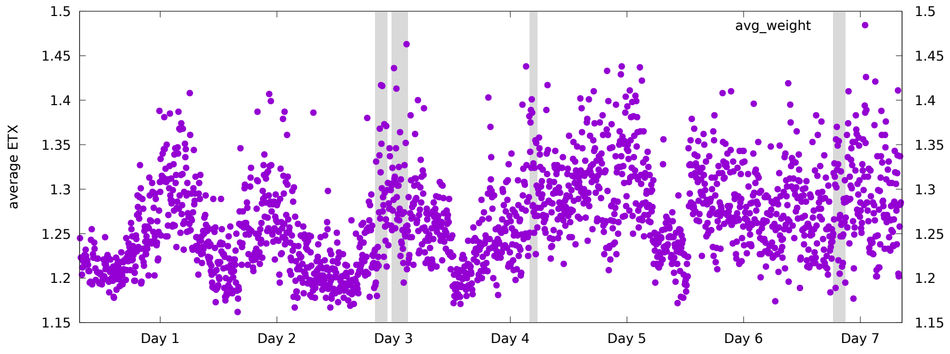
6 | DATA-BASED MESH MONITORING: TESTS AND LIMITATIONS

In this section we describe an initial experience in using available data from existing networks, together with graph-based analysis to perform anomaly detection. Our goal is to outline that monitoring and performing RCA on mesh networks with existent technology is a challenging task even when the sources of data are just a small subset of the ones described so far. Two problems clearly arise: the computational power needed to extract meaningful data and the need of human intervention to extract some useful information from available data. Based on these premises, section 7 presents some Machine Learning research directions to tackle the second problem, and section 8 introduces a big-data architecture to scale up to the necessary computational power.

In 2014 we monitored the behaviour of 3 large-scale mesh networks made of hundreds of nodes [2]. Among the data we extracted there is a sequence of network graphs that we obtained from the routing protocol. The three networks we monitored all use the well known OLSR (Optimized Link State Routing) protocol, which, being a link-state protocol, generates enough information for each node to reconstruct the whole network topology. OLSR annotates



(a) The average path cost and path length for every shortest path in the ninux network, for every graph.



(b) The average number of nodes in the network.

FIGURE 5 The average path cost, length and average number of nodes in the network.

each link with the ETX metric, that expresses the average number of transmission attempts for each successfully delivered packet [28]. An ETX value of 1 means that the link always delivers all the packets, an ETX value of 2 means that the link drops half of the packets. Note that ETX does not take into account the re-transmission mechanism of 802.11, so it greatly overestimate loss, neither it takes into consideration the available bit-rate. Still it is a very easy to implement, and thus widely used metric. For one week, every 5 minutes we dumped the network topology annotated with the ETX value per link. The whole data-set has been used for several publications [29, 30] and is available online[4]. For this paper we use only one network, the “ninux” network of Rome, made of about 130 nodes.

One way to assess the conditions of the network is to consider the cost of every path going from every possible source node n_i to every destination node n_j. ETX is an additive metric, so that the cost of a path is the sum of the costs on the edges that compose it, the higher the cost, the worse is the path. The minimum value of ETX is 1, so that for every hop, at least an unitary cost is added. If a path length is made of x hops, the minimum ETX cost is x.

Figure 5a shows the average path length and path cost for every network sample. The purple dots (the average path cost) have a clearly defined circadian trend, due to the fact that when the network activity is high the links become more congested and their ETX increases. This is confirmed by fig. 5b that shows the average value of ETX per edge per graph, which stays between 1.15 and 1.5 showing that the network performs in average pretty well.

[4]See https://zenodo.org/record/1218746

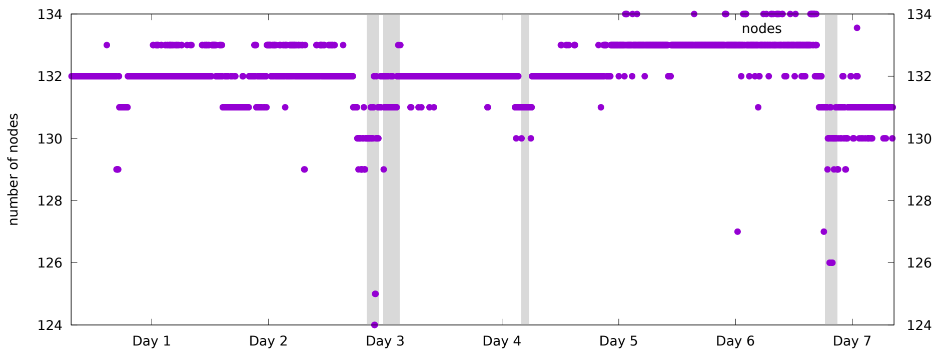


FIGURE 6 The average number of nodes in the network.

The path length shows a similar circadian trend but with a smaller variation, which is, again, expected. In fact, every time the ETX of a link increases the protocol may, or may not, abandon a certain path in favour of another one. The average path length increases only if the new path is longer than the old one.

We marked four specific time-windows labeled A,B,C and D in fig. 5a that help us explain an anomalous behaviour. In A and B there are some outliers in the ETX graph (fig. 5b), which produce outliers in the average path cost. One may easily conclude that in a moment in which some links were highly congested the average path cost was significantly impacted. Actually, the two situations are not the same; at time A the average path length has some outliers too, while at time B it doesn't and it is interesting to understand why.

Figure 6 shows the number of nodes in the network, which oscillates around the value of 132 during the whole week. Consider that ninux is a network created with a bottom-up approach from a group of volunteers, so that it is expected that some nodes could be unreliable. This happens normally at the fringes of the network, for some leaf nodes that 'come and go', since they are connected with unstable links. Having 4-5 nodes that appear and disappear in a network made of more than 130 nodes is acceptable for the community. At time A, the number of nodes drops to its minimum of the whole week, which does not happen at time B, so the number of nodes may be related to the outlier in path length. Yet if we look at time C, the number of nodes drops down again, but the path length and cost does not raise significantly.

In order to add another ingredient to the analysis, we compute the betweenness centrality of all the nodes in the network. Betweenness is a graph metric that represents a score of importance for every node in the graph: the betweenness of n_i represents the fraction of shortest paths between any couple of nodes in the network that pass through n_i [31].[5] Among the nodes that disappear at time A we identify two nodes (with identifiers 7276 and 7387) whose betweenness is 0.15 and 0.10. Not only they are not leaf nodes (whose betweenness is 0), but they are "in between" a large number of couples of nodes. Their breakage forces the protocol to choose alternative paths that make the average path length tangibly higher.

Figure 7 reports the state of the two nodes for the whole week and shows that for the large majority of time the nodes are On. At time A both nodes are Off, and it is the only moment in the week when this situation occurs. At time D node 7276 is Off and node 7387 is On, and in fact, we can see in fig. 5a that again there is an increase in the average path. There is a third point in which node 7276 is off but due to the density of points in fig. 5a the impact on the average path length is not noticeable.

[5]This definition is valid in graphs in which there is only one shortest path between two nodes.

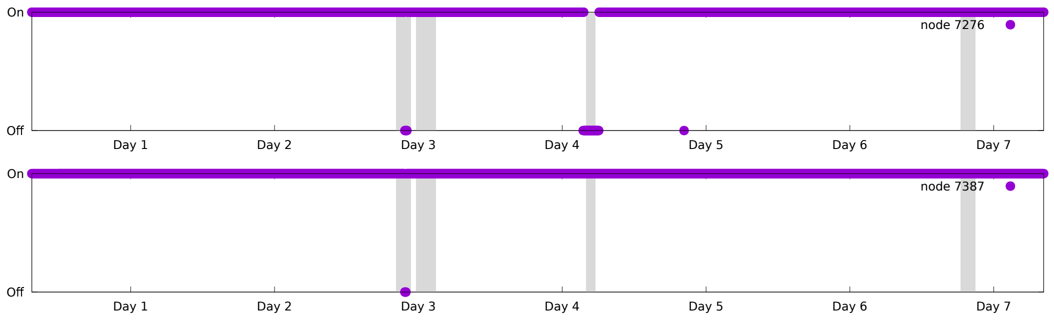


FIGURE 7 The On/Off state of nodes 7267 and 7387.

Summing up, we can conclude that the anomaly we encountered was most probably due to the contemporary failure of two nodes with high centrality. We note that the failure of 7276 happens in two moments of peak of traffic in the network (two moments in which the circadian pattern is close to the maximum). Failure could be related to a power outage that happens when the node is under stress and its CPU routes thousands of packets per second, but this is just a wild speculation. In practice our analysis stops here since we have no data to try to asses the root cause of this anomaly.

Three factors emerge clearly from this analysis. The first is that even with a limited amount of data extracted from the routing daemon we can already derive many interesting insights on the network behavior. The second is that in order to interpret the data we have to apply several layers of computation. For instance, betweenness calculation with the fastest algorithm available has a complexity of $O(nm + n^2 \log n)$ in a network with n nodes and m weighted edges [31]. On a modern CPU it took about 20 minutes to do all the computations needed to plot the graphs we have shown so far. Indeed, if this must be done in real time in a network with thousands of nodes for tens of graphs per second, we need to use Big Data solutions. The third factor is that finding the RCA of the anomaly was a process that required human intervention, and an interesting question is if this kind of activity could be replaced using a Machine Learning approach.

7 | MACHINE LEARNING FOR ANOMALY DETECTION AND RCA

The two themes that we focus on are anomaly detection and RCA, and each one deserves to be analysed separately in the next two sections.

7.1 | Anomaly Detection

Anomaly detection is the key technique to raise alarms when a system is misbehaving, and warnings when the problem still did not materialize but is giving the first signs of its presence. Anomaly detection for security applications is a well known research topic, with many connections to Machine Learning [32].

A first issue to approach is the problem of the classification of an anomaly, which in the case of an ISP can be actually done using user's feedback. When the network is not working properly, users will try to contact the operator and raise warnings. When this happens, a first-level customer support opens a ticket, based on the description of the problem given by the user. As the home router today is generally installed by the operator, and not owned by the client, the operator can even automatically perform remote operations on the router in order to acquire more data (assuming the router is reachable). In some sense, this relaxes the problem of classification because the typical sequence of tiers in the technical support (generally split in three levels of intervention) already produces enough information to identify network issues. Given this background of labeled information, supervised Machine Learning techniques can be used. In case there is a lack of examples to train the system, we can imagine two complementary directions to address this problem. On the one hand, the set of examples containing identified anomalies can be augmented with simulated runs, where one can observe the effect of arbitrary changes in the network topology over the network performance. On the other hand one can train a system to predict measures, like the shortest path length between pairs of nodes, which act as proxies of potential problems but are expensive to compute at run-time. In this case the anomaly detection works on data of a coarser grain, and thus requires less precision, but does not itself flags anomalies, it just triggers the use of other techniques that perform a finer-grained analysis, which could be based on statistical analysis or predetermined rules.

Once the corpus of examples is given, in our specific case we have to narrow the set of available learning techniques to the ones that can be applied to a time-varying sequence of (multidimensional) graphs.

Anomaly detection on time varying graphs is a studied subject [33] with several possible approaches. Among the relevant techniques we mention *Feature-based events* that try to identify the situations in which some of the key feature of the graph changed [34]. One typical approach is to use some distance between graphs (like Error Correcting Graph Matching, or Hamming distance for the adjacency matrix) and flag an anomaly when some threshold is reached. This approach can start in a localized manner (i.e. identifying small continuous regions of the graph that behave abnormally [35]), and then proceed by observing how the anomaly expands and creates temporary (vanishing) correlations with other detected anomalies [36]. Machine Learning plays a key role in speeding up this process since some of the available tools (like Recurrent Neural Networks or Dynamic Bayesian Networks) are well suited to model time sequences of graphs [37]. Graph kernels [38, 39, 40] and graph neural networks [41, 42] are natural candidates for their ability to extract relevant features from networked data. These techniques can be complemented with graph mining approaches [43, 44], in order to identify patterns.

Note also that the set of defined anomalies is not static, it changes and evolves with time, so the anomaly detection engine needs to follow this evolution. In this sense, techniques like Association Rule Mining have been successfully applied to anomaly detection [45].

7.2 | Root Cause Analysis

The next step, after identifying an anomaly is finding the root cause that generated it. Again, Machine Learning is extremely useful in this situation, as in essence, RCA is basically a specific case of inference, in which a sequence of causality relationships must be identified. Bayesian Networks are a natural candidate to perform inference, and Case-Based Reasoning is another valuable technique that helps to exploit previous expert knowledge on the domain. Both techniques have been proposed for RCA [4, 46].

Figure 8 represents a possible structure for a Bayesian network representing a toy communication network (we limit the analysis to the Phy, MAC and IP layer for clarity). In the figure we show how in each node the physical layer influences the MAC, which in turn influences the IP. This is an accurate description, as the link quality determines which neighbors are admitted at MAC layer, which in turn influences the routing tables at the IP layer. IP layers influence each other on different nodes, as the routing protocols exchange packets to create the routing tables. In our approach, the influence of the IP layer of node A towards the IP layer of node B follows the shortest path from B to A and thus, the Bayesian network is a tree (or a DAG in case of multipath). The random variable associated to

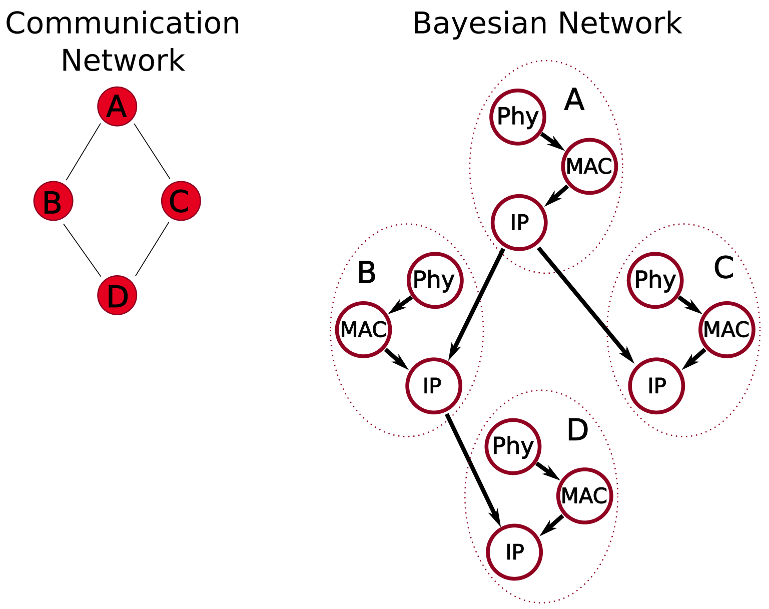


FIGURE 8 An example communication network (left) with a possible configuration for a Bayesian network used for RCA (right).

each node of the Bayesian network represents a metric computed on the node in the graph of the corresponding layer. For instance, let us call $G_{Phy}[i]$ the Com. Phy graph represented in the upper part of fig. 4, sampled at time t_i. Let the random variable A_{Phy} associated to the Phy layer of host A in the Bayesian network be an estimation of the sum of the link quality between A and its neighbors in G_{Phy} at time t_i (the weighted degree of node A in $G_{Phy}[i]$). If we consider a time window w then the distribution of A_{Phy} is a time-varying continue function derived from all the snapshots of $G_{Phy}[i]$ with $t - w < t_i \leq t$. With a similar process we can define the variables for the other layers of node A in the Bayesian network (A_{MAC} and A_{IP}) and for all other nodes. It is clear that the neighborhood at a certain layer in the network stack is influenced by the neighborhood at the lower layer, but this influence is probabilistic, as we can not deterministically predict how one layer influences the other. The Bayesian network can be trained during normal operation of the network deriving the right configuration for the edge weights.

Consider the following application example: link D-C has a very low quality, so its cost is very high and thus node D never uses C to reach A. The quality of link B-A is higher than D-C, meaning that its average cost is strongly lower, but, it is a bit shaky, meaning that its variance is high. In normal conditions the Bayesian network in fig. 8 will show that D_{IP} is influenced by a chain of factors, including A_{Phy}, which frequently changes and produces changes on the path to C. At time t_f link B-A fails, and D is forced to use the shortest path D-C-A. This impacts the average path cost (as discussed in section 6) and raises an alarm. Without our architecture the network manager can not understand the cause of the alarm, and will focus on the path D-C-A. With our architecture he can query the Bayesian networks for $t < t_f$, and understand that the anomaly was caused by the failure of the link B-A, as in the past A_{Phy} was a regular cause of influence on D_{IP}. Of course, the conclusion seems straightforward in this toy example, but in a large network our architecture can detect the root cause of an anomaly which would be impossible to detect by manual inspection. Note that using such an approach we can capture only the effects happening on the shortest path tree rooted in node

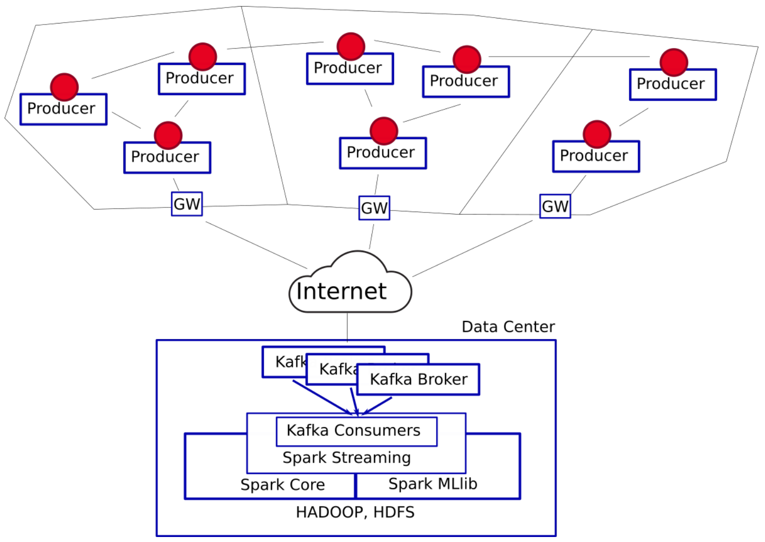


FIGURE 9 The overall architecture.

A. A different tree rooted in each of the nodes should be constructed, and the joint effects of all these trees could be studied to analyze the whole network.

Of course this model introduces some simplifications, the most evident one being that in real networks there are influence loops; for instance, Phy and MAC layers of different nodes influence each other. Treating influence in a network with loops can not be done with a Bayesian network, and we leave to future works the study of a more general approach. Furthermore, in order to explicitly bias our analysis towards causal relationships, we will investigate the recent literature on causal models, causal inference and counterfactual reasoning [47].

8 | SYSTEM ARCHITECTURE

The architecture we propose is based on cutting-edge research, and state of the art technology for Big Data analysis. Figure 9 depicts the components we envision and how they interact. The figure represents an example network, split in areas, in each area reside the nodes that use the same gateway to reach the Internet. At the core of the architecture there is Apache Spark. Spark is an open source Big Data tool that includes many of the functions that we need, specifically, the collection of streams of data and their analysis using Machine Learning techniques.

At the core of Spark there is a programming abstraction known as Resilient Distributed Datasets (RDDs) that make it possible to analyse data as they arrive from the stream, as well as to analyze batches of past data. RDDs support in-memory data storage in a distributed cluster in a fault-tolerant way, parallelised on a number of nodes in the cluster. RDDs support Transformation (creating new RDDs processing existing ones) and Actions, that do not modify the stream. In our architecture every kind of data is a separate stream. To generate streams from raw data there are multiple options, one widely used open source option is Kafka. Kafka is a distributed system made of "producers", "brokers" and "consumers". Producers generate the data streams, which are basically a sequence of values with an associated time. Kafka allows to produce streams divided in "topics", each topic can be split in "partitions". In our design we imagine that each node will produce a topic, tagged with a unique identifier of the node, and each topic will be split in one partition per kind of data (one for each line in the tables defined in section 4). Each partition is sent to one broker, brokers are in charge of storing the data and delivering it to the consumers. Brokers can maintain the stream of data for a configurable period (a "retention" period). This period can be set to any time window, and Kafka is designed to be fault tolerant, every broker can replicate the partitions it receives to a number of slaves so that in case of failure of a broker a slave becomes master and the system keeps working. The time needed for serving the data to consumers is not affected by the retention period (and thus, by the size of the stored data).

In our design we imagine to have one broker per gateway. Every node in the network is a producer of data and sends data to the corresponding broker, whose identifier is derived in some way from the gateway the node is currently using. If for some reason the node changes its default gateway, it will also change the corresponding broker. This way, we naturally implement a load balancing strategy based on the network organization at the routing layer. Brokers reside in the data center, and guarantee the necessary redundancy, which is transparent to producers and consumers. Spark will play the role of a Kafka consumer, and pull messages from brokers. We can imagine multiple consumers processing the streams in different, concurring ways. Multiple consumers will, for instance, process all the topics, and realize the graph representations that we explained in section 5, with one consumer per layer (in the case of aggregated graphs) or one consumer per function (in the case of graphs with detailed information). In the first case, as an example, this requires to receive the data in real time, average the series with some moving average function, and parse each layer of information in order to reconstruct the corresponding network graph.

Spark supports graph data-format using the GraphX library, each graph can be annotated with the relevant data in order to be analysed with Machine Learning algorithms. Operators and functions on graphs are allowed in GraphX, so that known metrics (e.g. PageRank) can be extracted from each graph. The algorithms can be run in parallel and relevant features extracted can be saved in a RDD, to be fed to machine learning algorithms in order to perform anomaly detection as explained in section 7. Spark currently does not natively support saving complex graphX structures, but in our case the dimension of the annotated graph is not as huge as in other Spark applications, we can keep in the system memory thousands of annotated graphs made of hundreds of nodes each. Graphs can be dumped in a text representation and reloaded in the future if needed. It is fundamental to guarantee horizontal scalability, and Spark can be used in a Hadoop cluster architecture, so that it can exploit the features of Hadoop without replicating them, especially the distributed filesystem and the resource manager that schedules tasks across nodes in the cluster. Resources can dynamically be added to the Hadoop cluster at any time, hence scaling the resources of the Monitor System dynamically. This is a key feature we need in our scenario since, as we explained in section 4 the size of the data-set and the complexity of the elaboration can change abruptly.

Finally, Spark comes with the very powerful MLlib, a library that provides state of the art algorithms for machine learning techniques. Among them, Spark integrates some of the algorithms we mentioned in section 7, such as Bayesian networks and decision trees.

Figure 10 reports an example tool chain for the data analysis. As described so far, the Kafka Broker contains the topics divided in partitions, each topic corresponds to a node. In this case we take the MAC layer reporting all the packets received by a node, with the corresponding source MAC address. Spark collects the stream from all the nodes, and elaborates each stream producing a network graph that interconnects all nodes (each node representing a MAC address) with nodes from which it received packets. Each graph is created from packets sent in a short time-window (e.g. 1s). Graphs are elaborated with GraphX, which exploits Spark to perform parallel computation of graph features (or node features). Each feature is saved in a RDD which contains its time-evolution. Such RDD is then passed to MLlib that uses some classification strategy to define outliers (in the figure we mention Support Vector Machine, which is supported by MLlib, but it could be any other machine learning primitive).

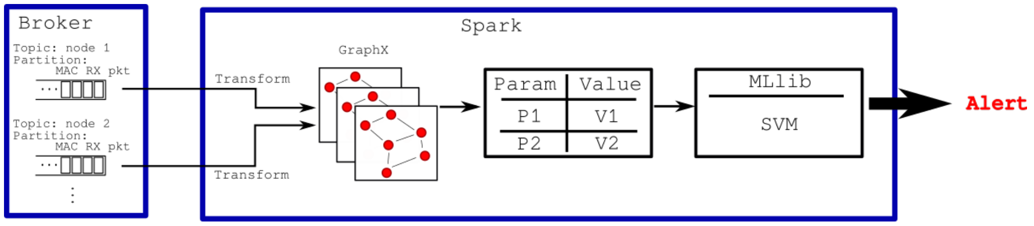


FIGURE 10 An example toolchain for data processing.

9 | CONCLUSIONS

Mesh networks are a mature technology which today offers new challenges, not much in the physical and MAC layer as it was studied in the past decade, but in the upper layers and in the management strategy. Large networks made of hundreds of even thousands of nodes can be set-up and their monitoring, especially for security reasons, is a daunting task. In this paper we explained why, and how Big Data tools and Machine Learning can be used to monitor, perform anomaly detection, and also RCA on large-scale wireless mesh networks. We proposed an architecture using state-of-the-art technology which can help solving critical situations, some of which we exemplified. We consider this architecture practically viable, and we plan to start its evaluation as a next step of this research.

references

[1] Ghasempour Y, da Silva CRCM, Cordeiro C, Knightly EW. IEEE 802.11ay: Next-Generation 60 GHz Communication for 100 Gb/s Wi-Fi. IEEE Communications Magazine 2017 DECEMBER;55(12):186–192.

[2] Maccari L, Lo Cigno R. A week in the life of three large Wireless Community Networks. Ad Hoc Networks 2015;24, Part B:175–190.

[3] Baig R, Roca R, Navarro L, Freitag F. Guifi.Net: A Network Infrastructure Commons. In: ACM International Conference on Information and Communication Technologies and Development, ACMDev; 2015. .

[4] Gonzalez JMN, Jimenez JA, Lopez JCD, Parada G HA. Root Cause Analysis of Network Failures Using Machine Learning and Summarization Techniques. IEEE Communications Magazine 2017;55(9):126–131.

[5] Yi X, Liu F, Liu J, Jin H. Building a network highway for big data: architecture and challenges. IEEE Network 2014 July;28(4):5–13.

[6] Yu S, Liu M, Dou W, Liu X, Zhou S. Networking for Big Data: A Survey. IEEE Communications Surveys Tutorials 2017 Firstquarter;19(1):531–549.

[7] Cui L, Yu FR, Yan Q. When big data meets software-defined networking: SDN for big data and big data for SDN. IEEE Network 2016 January;30(1):58–65.

[8] Trevisan M, Drago I, Mellia M, Song HH, Baldi M. AWESoME: Big Data for Automatic Web Service Management in SDN. IEEE Transactions on Network and Service Management 2018 March;15(1):13–26.

[9] Yaghoubi F, Furdek M, Rostami A, Öhlén P, Wosinska L. Consistency-aware Weather Disruption-tolerant Routing in SDN-based Wireless Mesh Networks. IEEE Transactions on Network and Service Management 2018;p. 1–1.

[10] Cheng X, Fang L, Yang L, Cui S. Mobile Big Data: The Fuel for Data-Driven Wireless. IEEE Internet of Things Journal 2017 Oct;4(5):1489–1516.

[11] He Y, Yu FR, Zhao N, Yin H, Yao H, Qiu RC. Big Data Analytics in Mobile Cellular Networks. IEEE Access 2016;4:1985–1996.

[12] Hanemann A, Boote JW, Boyd EL, Durand J, Kudarimoti L, Łapacz R, et al. PerfSONAR: A Service Oriented Architecture for Multi-domain Network Monitoring. In: Service-Oriented Computing - ICSOC 2005 Berlin, Heidelberg: Springer Berlin Heidelberg; 2005. p. 241–254.

[13] Kobayashi S, Otomo K, Fukuda K, Esaki H. Mining Causality of Network Events in Log Data. IEEE Transactions on Network and Service Management 2018 March;15(1):53–67.

[14] Aceto G, Botta A, Pescape A, Westphal C. Efficient Storage and Processing of High-Volume Network Monitoring Data. IEEE Transactions on Network and Service Management 2013 June;10(2):162–175.

[15] Wuhib F, Dam M, Stadler R, Clem A. Robust monitoring of network-wide aggregates through gossiping. IEEE Transactions on Network and Service Management 2009 June;6(2):95–109.

[16] Zhang Y, Debroy S, Calyam P. Network-Wide Anomaly Event Detection and Diagnosis With perfSONAR. IEEE Transactions on Network and Service Management 2016 Sept;13(3):666–680.

[17] Tan Z, Nagar UT, He X, Nanda P, Liu RP, Wang S, et al. Enhancing Big Data Security with Collaborative Intrusion Detection. IEEE Cloud Computing 2014 Sept;1(3):27–33.

[18] Zuech R, Khoshgoftaar TM, Wald R. Intrusion detection and Big Heterogeneous Data: a Survey. Journal of Big Data 2015 Feb;2(1):3. https://doi.org/10.1186/s40537-015-0013-4.

[19] Singh K, Chandra Guntuku S, Thakur A, Hota C. Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests. Information Sciences 2014;278:488 – 497. http://www.sciencedirect.com/science/article/pii/S0020025514003570.

[20] Yan Q, Huang W. A DDoS Detection and Mitigation System Framework Based on Spark and SDN. In: Qiu M, editor. Smart Computing and Communication Cham: Springer International Publishing; 2017. p. 350–358.

[21] Marchal S, Jiang X, State R, Engel T. A Big Data Architecture for Large Scale Security Monitoring. In: 2014 IEEE International Congress on Big Data; 2014. p. 56–63.

[22] Sun Z, Chen F, Chi M, Zhu Y. A Spark-Based Big Data Platform for Massive Remote Sensing Data Processing. In: Data Science Springer International Publishing; 2015. p. 120–126.

[23] Dutta R, Li C, Smith D, Das A, Aryal J. Big Data Architecture for Environmental Analytics. In: Environmental Software Systems. Infrastructures, Services and Applications Cham: Springer International Publishing; 2015. p. 578–588.

[24] Eisler M. RFC 4506, XDR: External data representation standard; 2006.

[25] Biswas S, Bicket J, Wong E, Musaloiu-e R, Bhartia A, Aguayo D. Large-scale measurements of wireless network behavior. In: ACM SIGCOMM Computer Communication Review, vol. 45 ACM; 2015. p. 153–165.

[26] Networks PA, MAGNIFIER: Rapidly hunt down and stop threats with cloud-delivered behavioral analytics and machine learning;. https://www.paloaltonetworks.com/apps/pan/public/downloadResource?pagePath=/content/pan/en_US/resources/datasheets/magnifier.

[27] Erdős P, Rényi A. On random graphs. Publicationes Mathematicae 1959;6.

[28] De Couto DSJ, Aguayo D, Bicket J, Morris R. 'A High-throughput Path Metric for Multi-hop Wireless Routing'. Springer Wireless Networks July 2005;11(4):419–434.

[29] Maccari L, Maischberger M, Cigno RL. Where have all the MPRs gone? On the optimal selection of Multi-Point Relays. Ad Hoc Networks 2018;77:69–83.

[30] Maccari L, Lo Cigno R. Improving Routing Convergence with Centrality: Design and Implementation of Pop-Routing. Accepted for publication in IEEE Transactions on Networking 2018;`https://ans.disi.unitn.it/users/maccari/assets/files/bibliography/Maccari2018Improving.pdf`.

[31] Brandes U. A Faster Algorithm for Betweenness Centrality. J of Mathematical Sociology 2001;25(2):163–177.

[32] Buczak AL, Guven E. A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection. IEEE Communications Surveys Tutorials 2016 Secondquarter;18(2):1153–1176.

[33] Akoglu L, Tong H, Koutra D. Graph based anomaly detection and description: a survey. Data Mining and Knowledge Discovery 2015 May;29(3):626–688. `https://doi.org/10.1007/s10618-014-0365-y`.

[34] Bunke H, Dickinson PJ, Kraetzl M, Wallis WD. A graph-theoretic approach to enterprise network dynamics, vol. 24. Springer Science & Business Media; 2007.

[35] Mongiovi M, Bogdanov P, Ranca R, Papalexakis EE, Faloutsos C, Singh AK. Netspot: Spotting significant anomalous regions on dynamic networks. In: Proceedings of the 2013 SIAM International Conference on Data Mining SIAM; 2013. p. 28–36.

[36] Cheng W, Ni J, Zhang K, Chen H, Jiang G, Shi Y, et al. Ranking Causal Anomalies for System Fault Diagnosis via Temporal and Dynamical Analysis on Vanishing Correlations. ACM Trans Knowl Discov Data 2017 Jun;11(4):40:1–40:28. `http://doi.acm.org/10.1145/3046946`.

[37] Hsu D. Anomaly Detection on Graph Time Series. arXiv preprint arXiv:170802975 2017;.

[38] Vishwanathan SVN, Schraudolph NN, Kondor R, Borgwardt KM. Graph Kernels. J Mach Learn Res 2010 Aug;11:1201–1242. `http://dl.acm.org/citation.cfm?id=1756006.1859891`.

[39] Shervashidze N, Schweitzer P, van Leeuwen EJ, Mehlhorn K, Borgwardt KM. Weisfeiler-Lehman Graph Kernels. J Mach Learn Res 2011 Nov;12:2539–2561. `http://dl.acm.org/citation.cfm?id=1953048.2078187`.

[40] Yanardag P, Vishwanathan SVN. Deep Graph Kernels. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining KDD '15, New York, NY, USA: ACM; 2015. p. 1365–1374. `http://doi.acm.org/10.1145/2783258.2783417`.

[41] Scarselli F, Gori M, Tsoi AC, Hagenbuchner M, Monfardini G. The Graph Neural Network Model. IEEE Transactions on Neural Networks 2009 Jan;20(1):61–80.

[42] Niepert M, Ahmed M, Kutzkov K. Learning Convolutional Neural Networks for Graphs. In: Balcan MF, Weinberger KQ, editors. Proceedings of The 33rd International Conference on Machine Learning, vol. 48 of Proceedings of Machine Learning Research New York, New York, USA: PMLR; 2016. p. 2014–2023. `http://proceedings.mlr.press/v48/niepert16.html`.

[43] Chakrabarti D, Faloutsos C. Graph Mining: Laws, Generators, and Algorithms. ACM Comput Surv 2006 Jun;38(1). `http://doi.acm.org/10.1145/1132952.1132954`.

[44] Cook DJ, Holder LB. Mining Graph Data. USA: John Wiley &#38; Sons, Inc.; 2006.

[45] Treinen JJ, Thurimella R. A Framework for the Application of Association Rule Mining in Large Intrusion Detection Infrastructures. In: 9th International Symposium on Recent Advances in Intrusion Detection (RAID) Hamburg, Germany; 2006. .

[46] Bennacer L, Amirat Y, Chibani A, Mellouk A, Ciavaglia L. Self-Diagnosis Technique for Virtual Private Networks Combining Bayesian Networks and Case-Based Reasoning. IEEE Transactions on Automation Science and Engineering 2015 Jan;12(1):354–366.

[47] Pearl J. Causality: Models, Reasoning and Inference. 2nd ed. New York, NY, USA: Cambridge University Press; 2009.