Sub-linear Scalability of MQTT Clusters in Topic-based Publish-subscribe Applications

Andrea Detti, Ludovico Funari and Nicola Blefari-Melazzi

***Abstract*—Message Queuing Telemetry Transport (MQTT) is a widespread protocol for topic-based publish-subscribe architectures supporting IoT and social networks applications. MQTT brokers are logical entities that couple publishers and subscribers and play a critical role in such architectures. MQTT brokers can be implemented either as standalone servers or in a cluster configuration. Clusters of brokers increase reliability, availability and overall performance, since operations can be highly parallelized among the brokers that form the cluster. The load-balancing strategy in a cluster usually consists in connecting an incoming client to a randomly selected broker. This random-attach strategy, it is very simple, but generates a significant amount of inter-broker traffic, as we demonstrate through theoretical and experimental evaluations. Inter-broker traffic is an overhead for the system and it increases the CPU load of the brokers, compromising the scaling behavior of the whole cluster. Indeed, we found that a linear increase of the number of brokers forming a cluster does not necessarily provide an equivalent linear gain in performance, and such a scaling penalty can be surprisingly significant, in the order of 40%. To solve this issue and improve performance, we propose a novel load-balancing strategy that envisages the use of multiple MQTT sessions per client to reduce inter-broker traffic and that can be implemented by means of a greedy algorithm. We show feasibility and effectiveness of our strategy for IoT and social-network applications by means of simulations and real measurements. The resulting scaling penalty is reduced to 10%.**

***Index Terms*—Topic-based publish-subscribe, MQTT, Cluster, Scalability, IoT, social networks**

I. Introduction

The Message Queuing Telemetry Transport (MQTT) is a widespread protocol for IoT and social networks applications based on a topic-based publish-subscribe communication pattern [1]. Publishers and subscribers are coupled to each other through a logical entity called MQTT *broker*, whose failure or decrease in performance can significantly affect the entire system. For this reason, some MQTT implementations support cluster-based deployments. An MQTT cluster is a distributed system that behaves, from the user point of view, as a single logical broker, while multiple physical MQTT brokers handle the workload behind the curtains. Clusters bring about several advantages, from the increase of system reliability and availability, since the failure of a single node has a limited impact, to the achievement of better performances, since the load of the processes that handle message routing and forwarding is distributed over different nodes. In addition, the number of nodes of a cluster can be increased (scale-out) or decreased (scale-in) to better accommodate traffic demand.

Authors are with CNIT and the Department of Electronic Engineering, Univeristy of Rome "Tor Vergata", Italy, e-mail: {name.surname@uniroma2.it}

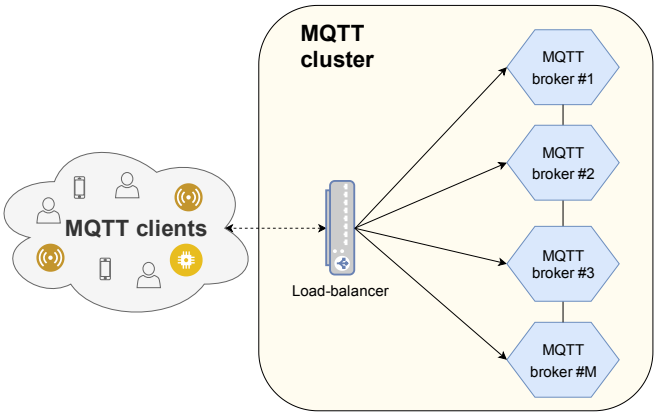

Fig. 1: Cluster example

The brokers forming the cluster run on separate physical servers or are hosted in virtual machines, preferably connected on a dedicated cloud network (Fig. 1). Clients can connect to any broker and receive messages published on any other broker of the cluster. Usually, a load-balancer node is used as a single TCP/IP entry-point of the cluster. It works at the lower TCP level to avoid possible processing bottlenecks. To establish a MQTT session, a client (publisher or subscriber) initially sets up a TCP/IP connection with the load-balancer that typically uses a destination NAT (DNAT) operation to redirect the connection to one of the internal brokers, chosen at random. This is for instance the default behavior of many MQTT implementation supporting clustering [2] [3] [4] when they use Kubernetes [5] as cloud orchestration platform. We will call this load-balancing strategy *random-attach*.

After the TCP/IP connection establishment, the client uses it to exchange MQTT packets with the broker [1]. With regard to the delivery of published messages to interested subscribers, MQTT supports 3 levels of QoS agreements between the client and the broker. QoS 0 (best effort) means that a published message is delivered "at most once" to interested subscribers, i.e. loss of the message is possible; QoS 1 means "at least one", i.e. no loss but duplicates are possible; and QoS 2 means "exactly once", i.e. neither loss nor duplicate.

As regards available implementations, a popular open-source MQTT broker is Eclipse Mosquitto [6], which however does not support clustering and can not take advantage of multi-core CPUs, as it leverages only one single thread. Another widespread message broker is RabbitMQ [7]; but,

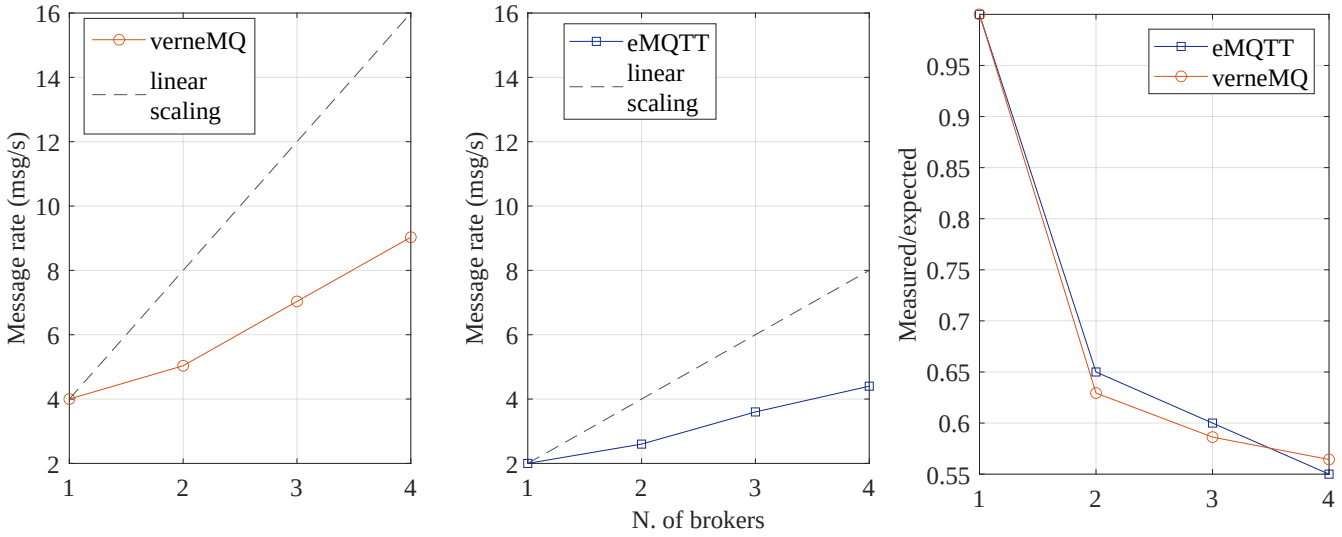

Fig. 2: Publishing rate that guarantees 2ms of message latency, in case of 1000 topics, 1 publisher and 1 subscriber per topic, VerneMQ and eMQTT brokers

it natively supports other publish/subscribe protocols than MQTT, like AMQP (Advanced Message Queuing Protocol), which are less of interest for our scenario; RabbitMQ does provide also an MQTT implementation, but, at time of writing, the latter misses important features, such as advanced quality of service support (QoS2, see above), which can be important for some applications. HiveMQ [2] supports clustering and is built in Java with scalability and enterprise-ready security in mind, even though it needs a license in order to properly work, as the evaluation version supports only 25 connections. HiveMQ is available also as open-sourced Community Edition, which however does not provide clustering capabilities. High-performance and open-source MQTT implementations that support clustering are VerneMQ [3] and eMQTT [4], both based on Erlang OTP, a very popular language in the message broker world, because it allows building distributed, highly scalable and nearly real-time messaging systems.

Thus, we made some experiments with VerneMQ and eMQTT implementations, aimed at measuring their performance. Results unveiled a similar and remarkable sub-linear scaling behavior that intrigued us to further investigate the matter. Besides, since we used two different implementations, the similarity of results has strengthened our belief that the issue is not related to a specific implementation, but it is rather due to the widely used random-attach strategy itself.

Let us better describe our experiment. By using Kubernetes [5], we have deployed different MQTT cluster configurations, ranging from a cluster with one broker up to a cluster with 4 brokers. The workload comprises 1000 topics, each one with a single publisher and a single subscriber. All publishers are equal and generate messages with a payload of 200 bytes, with MQTT QoS equal to 0 and with a message inter-time that follows a Poisson distribution. The brokers and the load-balancer run on different virtual machines with 2 CPUs each, hosted by a Microsoft Azure cloud. The publishers' and the subscribers' applications run on another virtual machine with 16 CPUs, so that the message throughput bottlenecks are, inevitably, the brokers.

We evaluated the advantage of increasing the number of brokers on the maximum rate of publications that the system can support, keeping the average message latency close to 2 ms. The related measurements are reported in Fig. 2.

To better understand these results, we have also included in the plots the values that we would have in case of a linear scaling of the performance, which implies that if we increase the number of brokers to M, the maximum publication rate increases to M times that of a single broker. For instance, in our deployment, a single VerneMQ broker can support up to 4000 messages per second, targeting 2ms of message latency. Therefore, by using two brokers, a linear scaling would imply a rate of 8000 messages per second, and so forth.

Surprisingly, the actual performance is rather far from the expected linear behaviour. The plot on the right of Fig. 2 measures such sub-linearity, by showing the ratio between the measured and the linear scaling rate. If performance scaled linearly, we should see a constant ratio equal to 1. Instead, as the cluster size increases, the ratio falls to about half of the expected value. The MQTT cluster is wasting half of the computing resources. We name this behaviour *sub-linear scalability of MQTT clusters*.

In this paper, we first study the characteristics and behaviour of the system, then we model it; after that we propose a solution to address the sub-linearity issue and we evaluate such solution.

II. Understanding the sub-linear scaling behavior

The random-attach strategy implies that publishers and subscribers of the same topic can be served by different brokers and therefore an internal exchange of messages is needed to route publications from brokers serving publishers to brokers serving subscribers. This internal traffic is an overhead

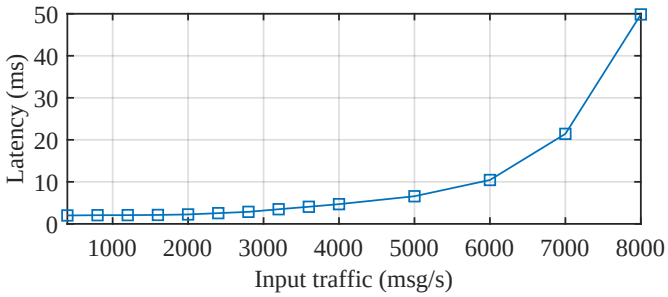

Fig. 3: Average message latency versus input traffic in case of a VerneMQ single broker; output traffic kept constant at 2000 msg/s. QoS=0

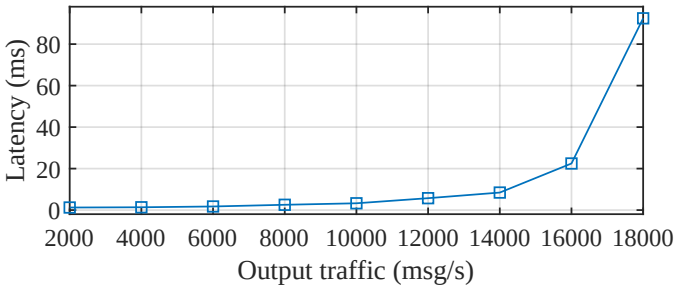

Fig. 4: Average message latency versus output traffic in case of a VerneMQ single broker; input traffic kept constant at 2000 msg/s

resulting from the clustering operation, which increases the routing and forwarding load of each broker of the cluster, giving rise to the sub-linear scaling behavior.

Within a broker, the routing function is used to single out which are the subscribers interested in an incoming message, usually by exploiting fast matching algorithms (e.g. subscription tries). Therefore, the CPU load due to the routing directly depends on the number of messages (publications) per second received by the broker, hereafter named *input traffic*. This behavior is shown in Fig. 3, in which we plot the average message latency versus the input traffic for a VerneMQ broker, while output traffic is kept constant at 2000 msg/s. [1]. The increase in input traffic leads to an increase in latency, which rapidly grows after 6000 msg/s as the broker gets close to the saturation of its CPUs.

After the routing function is executed, the forwarding function is used to send the message to the selected subscribers, while managing their QoS. The CPU load due to the forwarding operation is a function of: (i) the number of messages per second sent out by the broker, hereafter named *output traffic*, and (ii) the QoS level. Fig. 4 shows the average message latency versus the output traffic for a VerneMQ broker, while input traffic is kept constant at 2000 msg/s [2]. The increase of the output traffic leads to a message latency increase, which rapidly grows after 14000 msg/s because the broker is running out of CPUs. Besides, comparing Fig. 3 and Fig. 4, we note that for the same amount of msg/s, the input traffic has a considerably higher impact on the message latency, highlighting that routing is a more complex function than forwarding.

As shown in Fig. 5, when a broker joins a cluster, its input and output traffic not only comes from and goes to external clients but also comes from and goes to other brokers of the cluster. In general, a broker k manages a subset of publishers

[1]The whole testbed runs in the same data-center, thus network delay is negligible. For this experiment, we varied the number of topics from 100 to 2000. Each topic is used by a different publisher, sending a message with an average rate of 4 msg/s. The number of subscribers is 2000 and each of them is interested in a single topic, which is chosen using a round-robin strategy among the topics.

[2]We used 500 publishers of different topics, sending a message with an average rate of 4 msg/s. The number of subscribers is varied from 500 to 4500 and each of them is interested in a single topic, which is chosen using a round-robin strategy.

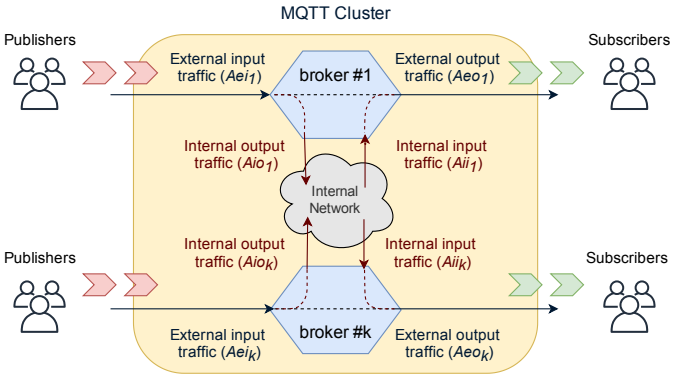

Fig. 5: Internal and external traffic of an MQTT cluster

and subscribers and deals with four types of traffic:

- External input traffic (Aei_k): the stream of messages generated by the connected publishers.
- External output traffic (Aeo_k): the stream of messages sent to the connected subscribers.
- Internal output traffic (Aio_k): the part of messages of the external input traffic forwarded to other brokers of the cluster, having interested subscribers.
- Internal input traffic (Aii_k): the stream of messages received from other brokers of the cluster, when the broker k has interested subscribers.

By using the random-attach strategy, the load-balancer fairly distributes publishers and subscribers among brokers. Thus, as the number M of brokers increases, each broker serves $1/M$ of external traffic (Aei_k, Aeo_k). However, the cluster configuration gives rise to the generation of internal traffic (Aii_k, Aio_k); as a consequence, the overall traffic handled by a broker, as well as its CPU load, actually scales out less than $1/M$, causing the sub-linear scaling behavior.

III. Analytical Model of the Random-Attach Strategy

In order to derive insights on the effect of different configurations and traffic parameters, in this section we present an analytical model for evaluating the total internal and

external traffic of an MQTT cluster that uses the random-attach strategy. We note that, since the traffic is uniformly distributed over the brokers by the load-balancer, the traffic handled by each broker is simply the total traffic divided by the number of brokers M of the cluster.

To simplify the development of the model we make the following assumptions:

- each topic has one publisher and many subscribers[3];
- publishers and subscribers are connected to a broker randomly selected among the M possible ones (random-attach);
- the number of topics (and thus of publishers) is kept constant to $Ntop$;
- the number of subscribers is kept constant to $Nsub$;
- the average rate of messages published on topic j is equal to λ_j (msg/s);
- the distribution of the topics chosen by a subscriber depends on the application scenarios; we modeled two of them, namely: *social network* and *IoT*.

As described below, the difference between the two application scenarios is mainly that in the social network case subscribers follow a constant number of topics, while in the IoT case topics are organized hierarchically and the number of topics followed by a subscriber is variable.

A. Social network scenario

This scenario is inspired by publish/subscribe social network applications like Twitter, where a subscriber is interested in different topics [8]. Accordingly, we assume that every subscriber is interested in a number of topics equal to $Nsxs$ (number of subscriptions per subscribers) and that the topic popularity follows a Zipf distribution with shape factor α, as observed in [9] [10].

An important parameter of the analytical model is the subscription-probability Ps_j that the topic j is included within the $Nsxs$ topics of a subscriber. This value is independent of the specific subscriber, because they have identical statistical behaviour. When $Nsxs = 1$, the subscription-probability is merely equal to the Zipf, i.e. $Ps_j = c/j^{\alpha}$, where c is the Zipf normalization constant. However, when $Nsxs > 1$, computational problems show up.

In the general case, the subscription-probability is equal to the so-called inclusion probability, resulting from a process of sampling without replacement, made of $Nsxs$ extractions coming from a finite population of $Ntop$ elements.

As in [11], we assume a sampling strategy working as follows: during the sampling process every extracted element is reinserted in the ballot, but whenever a duplicate appears, the current draw is rejected and is redrawn. Under this assumption, the subscription probability can be computed by modelling the extraction process as the superposition of Poisson processes. There is a process per topic, whose average frequency is equal to the topic popularity. The average time τ needed to extract $Nsxs$ different samples is the unique root of the following equation [4]:

$$\sum_{j=1}^{Ntop} (1 - e^{-(c/j^{\alpha})\tau}) = Nsxs \tag{1}$$

Consequently, the subscription probability of the j element can be approximated as the probability to have at least an arrival of the element j in τ seconds, i.e.

$$Ps_j = 1 - e^{-\lambda_j \tau} \tag{2}$$

B. IoT Scenario

This scenario is inspired by IoT applications in which topic names follow a hierarchy that is somehow shaped as the physical environment, e.g. `<roomId>/<sensorType>/<sensorId>` [13] [14]. Subscribers can be interested in the publications related to a specific topic or in those related to an aggregation of topics sharing the same name-prefix, e.g. `<roomId>/<sensorType>/#`, where # is a multi-level wildcard meaning that the subscriber will receive all messages of a topic that begins with the pattern before the # character.

We model the hierarchy of possible subscriptions with a tree having depth d and fan-out f. Fig. 6 depicts an example of the tree for $d = 3$. The first level of the tree contains the root node, which represents a subscription on every topic, i.e. on "#". The second level of the tree contains subscriptions for aggregations of topics, simply identified by name-prefixes $1, 2...f$. Finally, the third level leaves represent subscriptions for specific topics, ranging from 1/1 to f/f.

The selecting process of the subscription provides that a subscriber firstly chooses one level of the tree according to a given probability vector $Pl = Pl_1..Pl_d$, where Pl_i is the probability of selecting the ith level. Then, the subscriber randomly chooses one out of the nodes of the level. Consequently, the subscription-probability Ps_j can be readily expressed as:

$$Ps_j = \sum_{i=1}^{d} Pl_i \frac{1}{f^{i-1}} \tag{3}$$

C. External Input and Output Traffic

The total external input traffic (Aei) is simply the sum of all the message flows (λ_j) generated on each topic, i.e.

$$Aei = \sum_{k=1}^{M} Aei_k = \sum_{j=1}^{Ntop} \lambda_j \tag{4}$$

The total output traffic (Aeo) is equal to the average traffic sent to a subscriber, multiplied by the number of subscribers $Nsub$. The average traffic sent to a subscriber is the sum of

[3]MQTT makes possible to express the interest to an aggregation of topics by using a single subscription that includes wildcards like "+" and "#". This possibility can be taken into account in the model (and in the following greedy algorithm too) decomposing such subscription in many single-topic subscriptions made by the same subscriber. Moreover, the extension to many publishers per topic is not difficult. Each message stream generated by a publisher can be associated with a separate sub-topic and a subscription made on the main topic can be handled as a set of subscriptions made on all the associated sub-topics.

[4]It is worth mentioning that the same approximation is used in caching models with LRU policy and is known as *Che's approximation* [12].

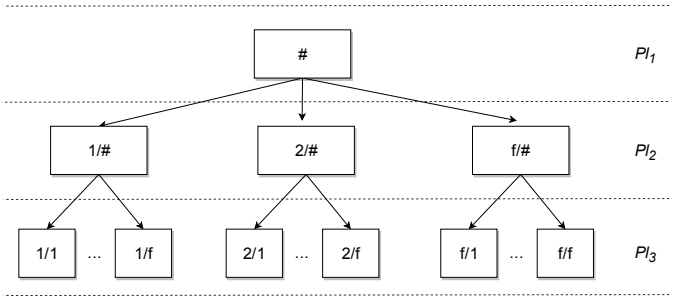

Fig. 6: Subscription tree with $d = 3$

the traffic generated on the different topics λ_j, each multiplied by the probability that the specific topic is selected by the subscriber, i.e. the Ps_j subscription-probability previously evaluated.

$$Aeo = \sum_{k=1}^{M} Aeo_k = Nsub \sum_{j=1}^{Ntop} Ps_j \lambda_j \tag{5}$$

Now let us turn our attention to the computation of the internal traffic. The total internal input traffic Aii is equal to the total internal output traffic Aio because there is no traffic loss within the cluster. Consequently, we can compute only the internal input traffic and simply call it internal traffic Ai.

We note that a broker internally receives the messages of topic j when: i) it has at least a subscriber of topic j connected to it, and ii) the publisher of the topic j is connected to a different broker. Therefore, the total internal input traffic Aii, as well as the output one Aio, can be written as:

$$Ai = Aii = Aio = \sum_{h=1}^{M} \sum_{j=1}^{Ntop} Pbs_{h,j} \left(1 - \frac{1}{M}\right) \lambda_j = (M-1) \sum_{j=1}^{Ntop} Pbs_j \lambda_j \tag{6}$$

where, the value $Pbs_{h,j}$ is the probability that the hth broker has at least a connected subscriber that is interested in the topic j and $(1 - 1/M)$ is the probability that the publisher of the topic j is not connected to the hth broker, since we are assuming a single publisher per topic. Since every subscriber is connected to a random broker with the same probability, then $Pbs_{h,j}$ is independent of the specific broker h, i.e. $Pbs_{h,j} = Pbs_j$.

It now remains to evaluate Pbs_j, which can be written as the probability of having at least a subscriber of the topic j when the broker has k subscribers of any topic, i.e. $1-(1-Ps_j)^k$, weighted for the probability that the broker has k subscribers out of the $Nsub$ possible ones (binomial distribution B), in formulas:

$$Pbs_j = \sum_{k=1}^{Nsub} B(k, Nsub, \frac{1}{M})(1-(1-Ps_j)^k) \quad \text{where, } B(k,n,p) = \binom{n}{k} p^k (1-p)^{n-k} \tag{7}$$

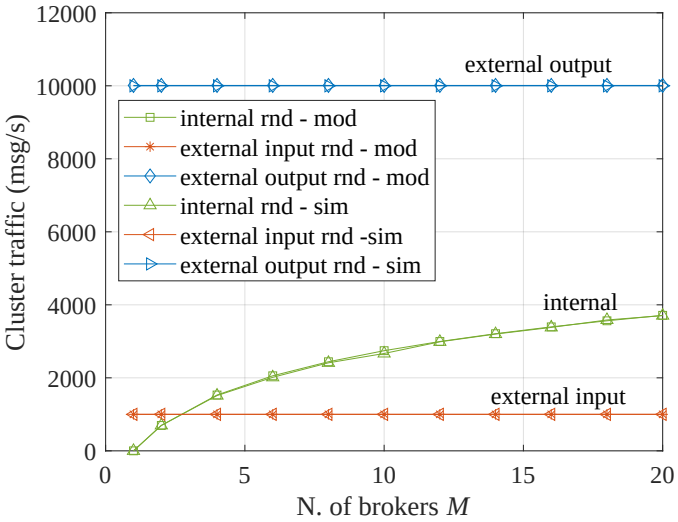

Fig. 7: Cluster traffic vs. cluster size for the social network scenario, random-attach (rnd), $Ntop = 1000, Nsub = 1000, Nsxs = 10, \lambda_j = 1, \alpha = 1.13$

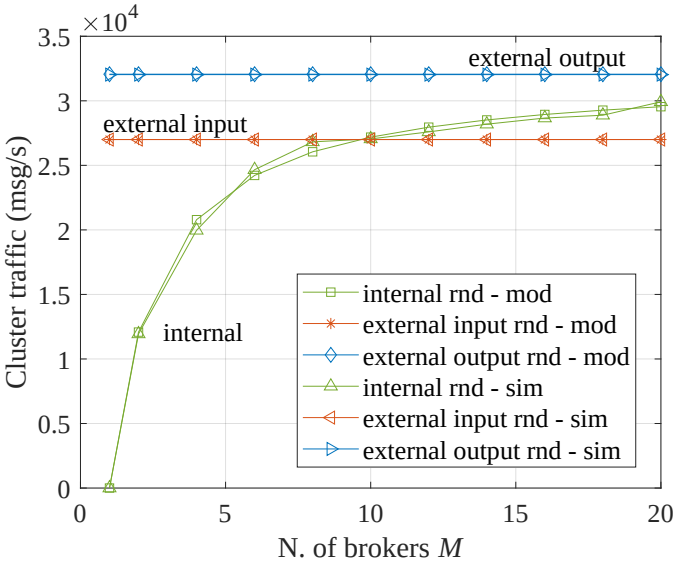

Fig. 8: Cluster traffic vs. cluster size for the IoT scenario, random-attach (rnd), $Ntop = 1000, Nsub = 1000, \lambda_j = 1, d = 4, Pl = [0, 0.03, 0.3, 0.67]$

Preliminary results

Fig. 7 and Fig. 8 show the cluster traffic versus the number of brokers M for the social network and IoT scenarios, respectively. For evaluating the external input traffic we used Eq. 4, for the external output traffic we used Eq. 5 and for the internal traffic we used Eq. 6. The figures also include the results that we obtained by means of a MATLAB simulator that reproduces the configurations used in the analytical model. The simulation and the theoretical results are basically overlapping, thereby confirming the validity of the analytical approach. As foreseen in Sec. II, the increase in the number of brokers results in an increase of the internal traffic, which consumes the brokers' processing resources, causing the sub-

linear scaling behavior.

IV. Multi-Session Best-Matching Strategy

To improve the cluster scalability, we have to limit the generation of internal traffic. To this end, the load-balancer should try to put publishers and subscribers of the same topics together, on the same brokers, thus avoiding inter-broker internal traffic. However, two issues make it difficult to achieve a perfect grouping as well as zero internal traffic:

1) Subscribers are usually interested in more than one topic.
2) External input and output traffic should be as much balanced as possible among brokers, as in the case of the random-attach strategy.

For example, let us look at the initial configuration in Fig. 9 (left) where we have a publisher and a subscriber of the topic "tennis" served by broker #1 and also a publisher and a subscriber of the topic "baseball" served by broker #2. We have a perfect grouping of homologous publishers and subscribers and therefore no internal traffic. Now, suppose that a new subscriber comes into the system and is interested in both tennis and baseball. Independently of the chosen broker for the new incoming subscriber, a new internal traffic flow is activated. For example, in Fig. 9 (middle) we have chosen broker #2 and this choice implies the generation of internal traffic that transports tennis publications from broker #1 to #2.

A possible way out to bring the internal traffic back to zero would seem to migrate the tennis publisher and subscriber from broker #1 to broker #2. However, such migration would face serious feasibility issues: MQTT is a stateful protocol, thus the MQTT context, and the underlying TCP/IP connection, cannot be moved from a broker to another in a seamless way. Moreover, even if the run-time migration of clients were possible, we would have to take into account that brokers must be fairly loaded, in order to benefit from the advantages of clustering. This fairness requirement restricts the option of migrating clients but also the choice of the broker to which connect a new incoming client.

In the following section, we propose a greedy load-balancing strategy, used to choose which broker has to serve an incoming client: publisher or subscriber. The strategy is aimed at reducing the internal traffic without having to resort to the migration solution and also addresses the aforementioned fairness issue. The technical feasibility of the strategy will be discussed at the end of the section.

We define Ta_k as the set of *active* topics of the kth broker, where a topic is active if there exists at least a subscriber or a publisher of that topic on the broker. Assuming that an incoming client is interested in a set of topics Tc out of the possible $Ntop$ ones, Tc_j is the jth topic of the set Tc and the size of the set is indicated as "len(Tc)".

Without loss of generality, we assume that the client is either a publisher or a subscriber. If the client is a subscriber, the increase of ΔAi of the internal traffic, resulting from its connection to broker k, can be written as:

$$\Delta Ai = \sum_{j=1}^{\text{len}(Tc)} \lambda_{Tc_j} I(Tc_j, Ta_k) \tag{8}$$

$$I(Tc_j, Ta_k) = \begin{cases} 1, & Tc_j \notin Ta_k \text{ and publisher of } Tc_j \text{ exists} \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

because, if the topic Tc_j is not active on the kth broker ($Tc_j \notin Ta_k$), and the publisher of the topic is connected to the cluster, then the broker starts receiving the related publications at a rate equal to λ_{Tc_j}. Otherwise, no additional internal traffic is generated.

If instead the client is a publisher, the internal traffic increase ΔAi resulting from its connection to broker k can be written as:

$$\Delta Ai = \sum_{j=1}^{\text{len}(Tc)} \sum_{h=1, h\neq k}^{M} \lambda_{Tc_j} I(Tc_j, Ta_h) \tag{10}$$

because, after the connection of the client, the publications of the jth topic of Tc are transferred to every other broker having at least a subscriber, i.e. having the topic active. If the client is both a subscriber and a publisher we can use Eq. 8 for the set of subscribed topics and Eq. 10 for the topics for which the client is a publisher.

The equations 8 and 10 initially drove us to design a *best-matching* greedy strategy, which follows the minimization in Eq. 11. Simply, when a new client arrives, the algorithm connects it to the broker that provides the lowest increase in internal traffic. However, to ensure a fair balance of external traffic among brokers, not all brokers can be selected. Indeed, the set of candidates is made up of brokers whose incoming and outgoing external traffics are equal to the fairly share values of Aei/M and Aeo/M, respectively, except for a tolerance fairness factor $\gamma \geq 1$. The greater γ, the smaller the fairness level.

$$\begin{aligned} \min_k \quad & \Delta Ai_k \\ \text{s.t.} \quad & k \in \{1..M\} \\ & Aei_k \leq \gamma \; Aei/M \\ & Aeo_k \leq \gamma \; Aeo/M \end{aligned} \tag{11}$$

In Fig. 10 we plot the amount of internal traffic, obtained with simulations, in case of the social network scenario for the random-attach (rnd) and best-matching (bm) strategies. In the left graph we have considered subscribers interested in a single topic ($Nsxs = 1$). In this case, the best-matching strategy is able to efficiently group homologous publishers and subscribers, thus achieving a valuable reduction of internal traffic compared to the classical random-attach strategy. However, problems arise when considering the case of subscribers interested in more than one topic. The graph on the right shows the results we get when we consider subscribers interested in 40 topics ($Nsxs = 40$). We see that the advantages brought in by the best-matching strategy are strongly reduced. This is simply due to the aforementioned difficulty of efficiently grouping publishers and subscribers in case of multiple subscriptions per subscriber, while ensuring a good level of fairness.

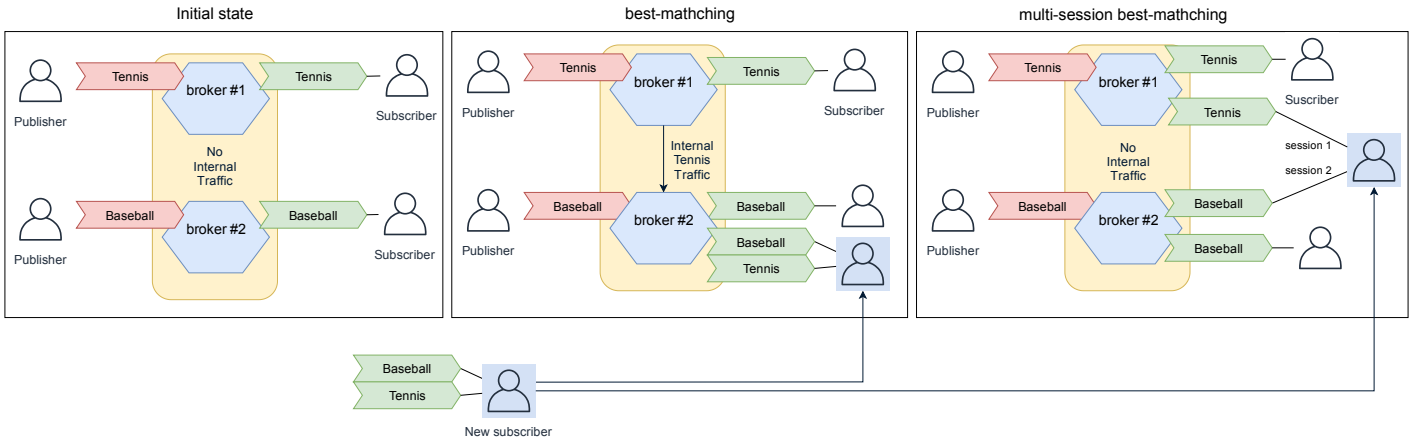

Fig. 9: New subscriber joining a cluster, best-matching (middle) and multi-session best-matching (right) algorithms

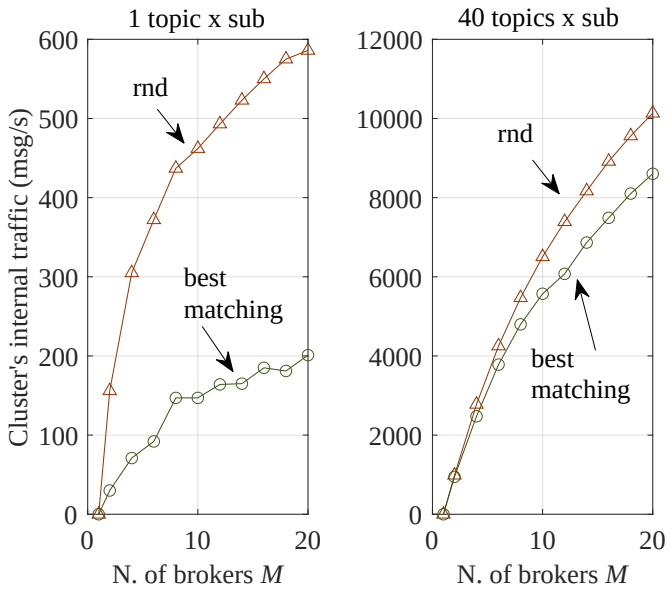

Fig. 10: Internal cluster traffic vs. cluster size for the social network scenario, best-matching algorithm, $Ntop = 1000, Nsub = 1000, Nsxs = 1$ (left) and $Nsxs = 40$ (rigth), $\lambda_j = 1, \alpha = 1.13, \gamma = 1.1$

In order to improve the reduction of internal traffic, we believe that it is necessary to expand the range of optimization options at the load-balancer's disposal, i.e. its decision space. Accordingly, we propose to introduce a new dimension into the game: the number of MQTT sessions a client can establish with the cluster. A client usually establishes a single session (and TCP/IP connection) with the cluster. To reduce internal traffic, we force the client to establish multiple sessions, but limiting their number to $Nses$. Each session will be used for a subset of the topics Tc and will be connected by the load-balancer to a different broker of the cluster. To showcase what we can do with multiple sessions, we start by noting that the configuration in Fig. 9 (in the middle) is a possible result of the best-matching strategy that uses just one session: new internal traffic is anyway generated. But if we simply increase the allowed number of sessions to two, then we can connect the client as shown in the right part of Fig. 9. The first session is used for the subscription to the tennis topic and is connected to broker #1. The second session is used for the subscription to the baseball topic and is connected to broker #2. Consequently, we obtain no increase in internal traffic.

To demonstrate the potential of introducing multiple sessions per client, we measure the number of optimization opportunities the load balancer gains by increasing the number of sessions $Nses$. We can make an analytical and a simulation-based evaluation as follows:

- analytically, we evaluate the number of possibilities the load-balancer would have by not considering the constraints on fair sharing of external traffic in Eq. 11. This computation provides an upper-bound of the real value;
- by using simulations, we measure the real number of possibilities considering a load-balancer that makes decisions taking into account of the constraints in Eq. 11 with $\gamma = 1.1$.

As far as the analytical evaluation is concerned, with only one session, the decision space is simply made of M options. In fact, for an incoming client, the load-balancer can only choose which is the broker to which the client must be connected to, among the M possible ones. With multiple sessions, the load-balancer can instead decide:

- which is the combination of at most $Nses$ brokers (one per session) to use among the M available in the cluster,
- which subset of topics Tc to assign to each session/broker.

It follows that the number of possible choices is equal to the number of ways of partitioning, with permutations, a set of len(Tc) elements into i "buckets", with $1 \leq i \leq Nses$, with these buckets extracted from a bucket pool of size M. Eq. 12 is the result of these choices, where $\left\{ {j \atop i} \right\}$ are the Stirling numbers of the second kind.

$$\sum_{i=1}^{Nses} \binom{M}{i} \left\{ {\text{len}(Tc) \atop i} \right\} i! \tag{12}$$

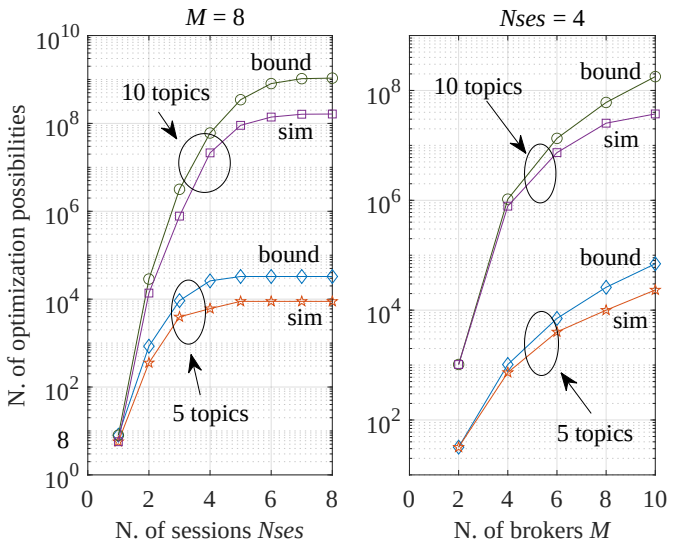

Fig. 11: Upper-bound and real value of the number of optimization possibilities available at the load-balancer in case of multiple sessions versus the number of session (left) and the cluster size (right), social-network scenario, $Ntop = 1000, Nsub = 1000, \alpha = 1.13, \gamma = 1.1$

For a cluster of 8 brokers ($M = 8$), Fig. 11 shows the number of possibilities for the load-balancer optimization, as a function of the number of $Nses$ sessions, on the left part of the figure, and of the number of brokers in the cluster on the right part. We consider the case of subscribers interested in 5 and 10 topics and plot both the upper-bound resulting from Eq. 12 and the values calculated by simulations, which take into account the fairness constraint in Eq. 11. Looking at the results on the left, we can appreciate the great expansion of the decision set when the number of sessions increases. We go from 8 possibilities in the conventional case of 1 session per client up to 10^8. We also observe that the upper-bound values are quite similar to the simulation results. This means that the fairness constraint of Eq. 11 does not actually limit the extension of the decision space. Another interesting property of the proposed approach is its ability to provide more optimization opportunities just when an optimization is more necessary to reduce internal traffic growth, for example when the number of subscribed topics per subscriber increases or when there is a greater number of brokers in the cluster (see the right part of Fig. 11).

We were looking for a new domain through which the load-balancer could expand its optimization space. The results presented show that the number of sessions per client can be one of these domains. Now, we need to figure out an optimization algorithm in this framework. The goal of such algorithm can be easily expressed in words as follows: for an incoming client, find a set of at most $Nses$ brokers and the subsets of client topics to assign to each of them in order to minimize the increase in internal traffic, while ensuring the fair share of external traffic as in Eq. 11.

We were not able to find an analytical, closed, solution for such a minimization problem and consequently we designed the sub-optimal greedy algorithm 1, named Multi-Session Best-Matching strategy. The algorithm is recursive: in each iteration, it finds first the next broker to use and then the set of client topics to assign to it. When $Nses$ brokers have been used or all client topics have been assigned to the selected brokers, the iterations end. The next broker (k) to be used is found by solving the minimization in Eq. 11 by using, as input, the set of client topics not yet assigned to any session (Tna). The subset of topic $Tbest$ of Tna to be assigned to that broker k are those that provide the minimum traffic increase [5]. This computation is made on a per-topic base by using Eq. 10 for a publisher client, or Eq. 8 for a subscriber client.

Algorithm 1: Multi-Session Best-Matching strategy

```
/* Tc, topics of client interest                    */
/* Tcb_k, topics of the client assigned
   to a session connected to the
   broker k                                         */
/* Uset set of used brokers                         */
/* Tna, topic of the client not yet
   assigned to any broker                           */
/* Nses, max number of sessions per
   client                                           */
/*                                                  */
i = 1 , Tna = Tc , Tcb_k = {} , Uset = {}
while len(Tna) > 0 do
    k = solution of Eq. 11 using Tna rather than Tc
    Tbest = subset of Tna topics providing minimum
     internal traffic increase when handled by broker k
    Tcb_k = Tcb_k ⋃ Tbest
    Tna = Tna \ Tbest
    Uset = Uset ⋃ k
    if len(Uset) == Nses then
        Tcb_k = Tcb_k ⋃ Tna
        Tna = {}
    end
end
```

We now verify the performance of the multi-session best-matching greedy algorithm in two small experiments by comparing its results with the ones obtained by a brute force algorithm that test all possible combinations and thus reaches optimal results. Larger tests were computational unfeasible for the brute force algorithm because of the hugeness of the decision space that is in the order of Eq. 12. Fig. 12 shows the amount of internal traffic in these two experiments versus the number of sessions in the left part of the figure, and the number of brokers on the right. Our greedy algorithm is sub-optimal because its internal traffic is slightly greater than the optimal one, but it is rather close to the optimum.

[5]This procedure assigns to the broker the client topics that are active and therefore do not produce any traffic increase, or only the topic that produces the minimum traffic increase. The remaining client topics will be assigned in a new iteration round, thus calculating Eq. 11 with a different set of Tna remaining topics, since for this new set the best broker may be different.

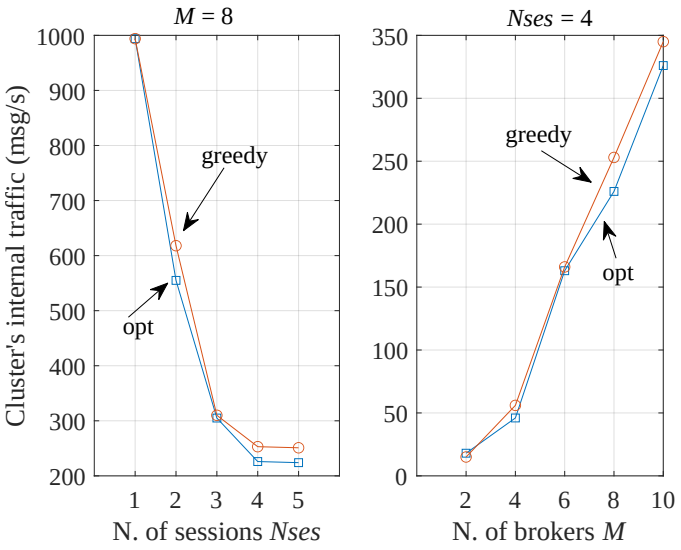

Fig. 12: Internal cluster traffic vs. number of sessions $Nses$ (left) and cluster size M (right) for the social network scenario; comparison between the multi-session best-matching greedy algorithm and the optimal solution, for $Ntop = 1000, Nsub = 1000, Nsxs = 5, M = 8$ (left) and $Nses = 4$ (rigth), $\lambda_j = 1, \alpha = 1.13, \gamma = 1.1$

A. Implementation issues

The multi-session best-matching algorithm requires the knowledge of the topics of interest of a client. The implementation of any topic-aware load-balancer, such as ours, is not an easy task. In particular, it is necessary to know which topics a client is interested in, before setting up the eventual connection with the internal broker(s); to this end, the load-balancer should be the end-point of the MQTT session, acting like a proxy, to decode MQTT messages such as PUBLISH, SUBSCRIBE, etc. But, in doing so, the load-balancer would surely become a severe system bottleneck, thus vanishing the horizontal scaling advantages of the cluster. Therefore, we have to face a challenge: how can we connect clients to the right brokers without closing the MQTT session on the load-balancer?

A possible idea that can be simply developed as a software library to be used on the client side is the following. We divide the load-balancing operations in two planes: (i) a control-plane that decides which are the $Nses$ brokers to be used for the different sessions and the related topics to assign to each of them; (ii) a data-plane that sends and receives topic messages from these brokers/sessions. Usually, a load-balancer implements both planes. In our proposal the control-plane remains in the hands of the cluster, but we move the data-plane to the client's side. Before connecting to the cluster, the client contacts the load-balancer control-plane in the cluster (e.g. through a REST interface) and tells it what topics is interested in. The controller applies the greedy algorithm and sends back a map containing, for each topic, the IP address and TCP port of the broker to be used. In practice, the controller behaves like a DNS for MQTT topics. Consequently, the client autonomously establishes the MQTT sessions and uses them for topic messages as indicated by the control-plane.

This solution does not require any modification of the MQTT standard. If changes to MQTT were possible, then other solutions could be explored. For example, we could introduce an MQTT Redirect message, to be used by the cluster to request the redirection of some topics to a specific broker in the cluster.

It is worth noting that, by allowing more sessions per client, we are increasing the number of TCP/IP connections that a single broker and a client has to handle; this may seem a serious disadvantage. On the broker side, we have measured experimentally that this is not a practical problem with current technology and we will present related results in the performance evaluation section. Roughly speaking, on a Linux system, the maximum number of TCP/IP connections is related to the maximum number of file descriptors, usually more than 300K, and thus we have an abundant space to support many connections. In addition, the memory footprint of a TCP/IP connection is negligible with modern memory chips, because the default read/write buffers are in the order of 16KB and this size can be reduced or increased by the kernel if necessary. On the client side, increasing the number of sessions may be unfeasible for constrained devices. However, the results will show that the proposed strategy appreciably improves performance already from 4 sessions/connections per client. Therefore, even constrained devices such as Raspberry PI and Arduino should not have any implementation problems.

Finally, we observe that having multiple sessions obviously increases the initial setup time, because the client needs to i) interact with the load-balancer control plane, as described above, and to ii) establish more than one MQTT session followed by the related subscription. This latter component linearly increases with the number of sessions in case of single-threaded implementation. In our testbed, whole running in the same data-center, we measured about 3.5 ms per session/subscription, mainly due to client/broker processing. Consequently, because in MQTT applications a session is usually maintained for a long period (hours, days) a setup overhead in the order of milliseconds is reasonably negligible.

V. Performance analysis

In this section, we evaluate the scaling performance of an MQTT cluster whose load-balancer uses either the random-attach or the multi-session best-matching strategy. Some performance parameters we consider are the routing and forwarding overheads, defined as the increase of the related processing load due to internal traffic, compared to the ideal case of no internal traffic. The higher the overhead, the lower the scaling performance, i.e. the advantage of increasing the number of brokers in the cluster.

We have seen that the routing load is proportional to the rate of messages received by brokers, i.e. $Aei + Aii$, and the forwarding load is proportional to the rate of messages sent by brokers, i.e. $Aeo + Aio$. Consequently, by using Eq. 6, the

routing h_{rt} and forwarding h_{fw} overheads can be written as:

$$h_{rt} = 1 + \frac{Ai}{Aei}$$
$$h_{fw} = 1 + \frac{Ai}{Aeo} \quad (13)$$

For example, a value of $h_{rt} = 1.5$ corresponds to a cluster whose internal traffic increases the forwarding load by 50% compared to the ideal case of no internal traffic.

A. Social Network Scenario

First of all, we analyze cluster performance for social network applications and we consider a value of 1.13 for the Zipf parameter α [9].

In Fig. 13 we can see how the system responds to a scale-out of the number M of brokers in the cluster. In this figure and in the next ones, we plot the simulation results of the random-attach strategy (rnd) with a solid line without markers and the analytical results with markers only. For the multi-session best-matching strategy (greedy), we show the simulation results obtained by using up to 4 sessions per client ($Nses = 4$).

As a general comment, having more brokers corresponds to an increase in the overhead. For a given topic, the spreading of interested subscribers among the different brokers becomes higher and this increases the internal traffic needed to bring them such publications. The routing overhead is much higher than the forwarding one because each subscriber is interested in 10 topics, in the considered configuration. In turns, the external output traffic (Aeo) is 10 times greater than the input traffic (Aei) and thus the impact of internal traffic (Ai) is greater on the routing overhead rather than on the forwarding one (see Eq. 13).

In the case of a random-attach strategy, the overhead reaches a discouraging value of about 4, for a cluster of 20 brokers. Using the multi-session best-matching strategy, the overhead is practically halved. This implies, for example, that a cluster using the greedy algorithm is theoretically capable of supporting a double publication rate with the same performance, e.g. the same average message latency.

Similar considerations motivate the cluster's response to the growth in the number of subscribers ($Nsub$) and the number of subscriptions per subscribers ($Nsxs$), as shown in Fig. 14 and Fig. 15 for a cluster with 8 brokers. However, it should be noted that, compared to Fig. 13, in these cases the forwarding overhead decreases slightly because the increase in $Nsub$ or $Nsxs$ leads to a higher growth in external output traffic (Aeo) than the growth in internal traffic (Ai). This is not the case for the routing overhead because the number of publishers, and therefore the external input traffic (Aio), remains constant while the internal traffic (Ai) grows.

For a cluster of 8 brokers, Fig. 16 shows that by increasing the number of sessions, the greedy algorithm is more and more able to reduce both routing and forwarding overhead, thus showing that the number of sessions is an effective tool to address the problem of sub-linear scaling of MQTT clusters. The random-attach strategy has constant results because it always uses a single session.

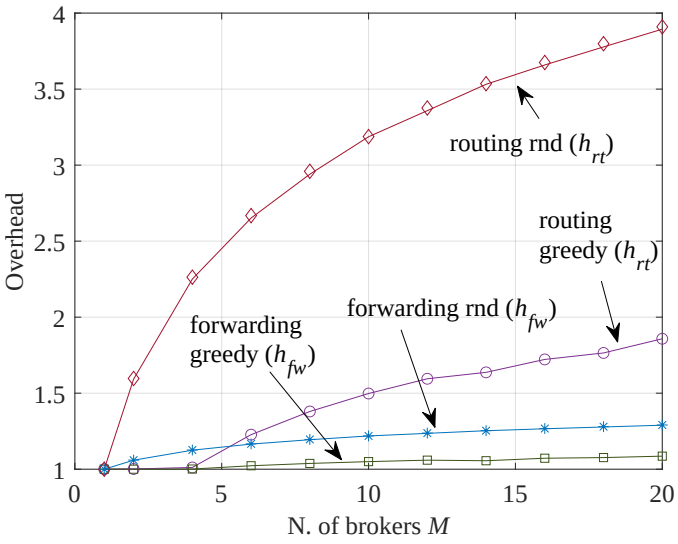

Fig. 13: Routing and forwarding overhead vs. cluster size M for the social network scenario, random-attach (rnd) and multi-session best-matching (greedy) strategies, $Ntop = 5000, Nsub = 5000, Nsxs = 10, \lambda_j = 1, \alpha = 1.13, Nses = 4, \gamma = 1.1$

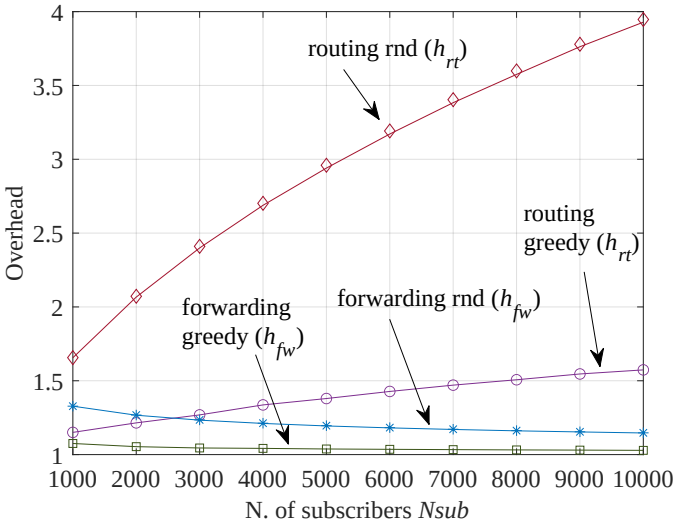

Fig. 14: Routing and forwarding overhead vs. number of subscribers $Nsub$ for the social network scenario, random-attach (rnd) and multi-session best-matching (greedy) strategies, $Ntop = 5000, Nsxs = 10, \lambda_j = 1, \alpha = 1.13, Nses = 4, M = 8, \gamma = 1.1$

Fig. 17 shows the overhead by varying the popularity parameter α of the Zipf distribution, which is used by subscribers to select the topics of interest. When α increases, subscribers focus on fewer topics, thus reducing the number of used topics in the cluster and the resulting inter-broker traffic that causes the overhead.

Finally, to evaluate the fair sharing of the load among the brokers of the cluster, we used the well-known Jain's index metric (J), applied to the sum (l_k) of input and output traffics,

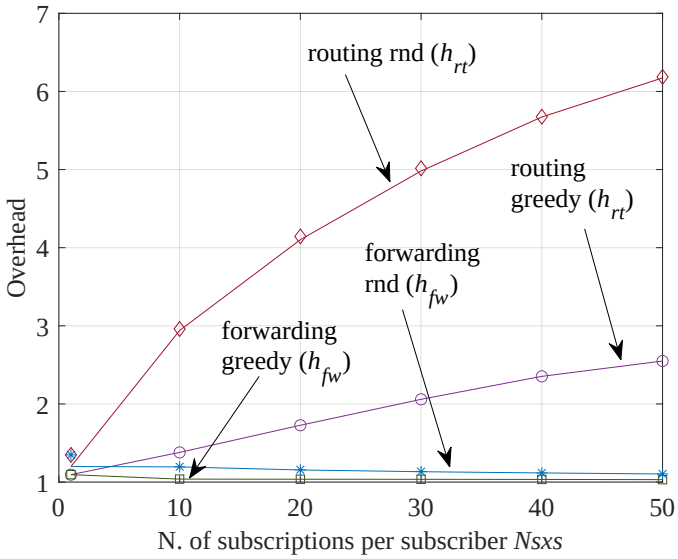

Fig. 15: Routing and forwarding overhead vs. number of subscription per subscribers $Nsxs$ for the social network scenario, random-attach (rnd) and multi-session best-matching (greedy) strategies, $Ntop = 5000, Nsub = 5000, \lambda_j = 1, \alpha = 1.13, Nses = 4, M = 8, \gamma = 1.1$

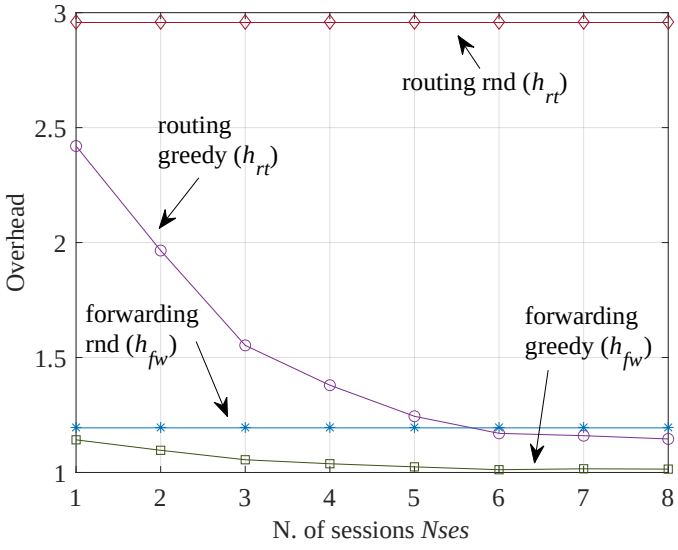

Fig. 16: Routing and forwarding overhead vs. number of sessions ($Nses$) for the social network scenario, random-attach (rnd) and multi-session best-matching (greedy) strategies, $Ntop = 5000, Nsub = 5000, Nsxs = 10, \lambda_j = 1, \alpha = 1.13, M = 8, \gamma = 1.1$

which has been considered to be a cumulative measurement of routing and forwarding load (Eq. 14).

$$l_k = Aei_k + Aeo_k + Aii_k + Aio_k$$
$$J = \frac{\left(\sum_{k=1}^{M} l_k\right)^2}{M \sum_{k=1}^{M} l_k^2} \quad (14)$$

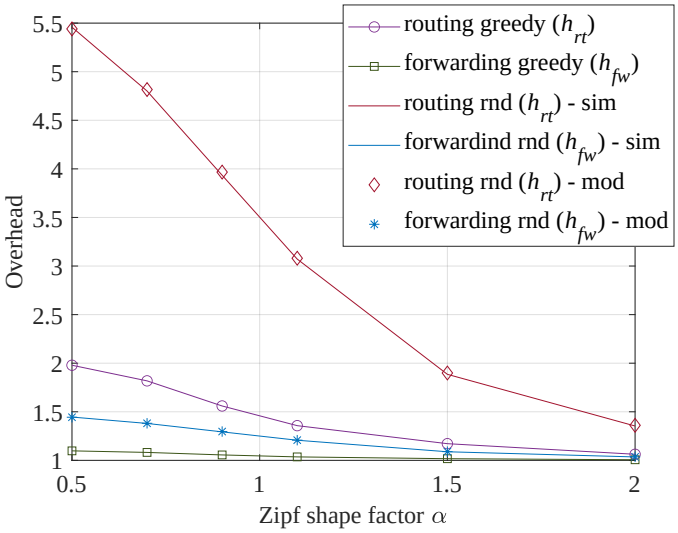

Fig. 17: Routing and forwarding overhead vs. Zipf shape parameter α for the social network scenario, random-attach (rnd) and multi-session best-matching (greedy) strategies, $Ntop = 5000, Nsub = 5000, \lambda_j = 1, Nsxs = 10, Nses = 4, M = 8, \gamma = 1.1$

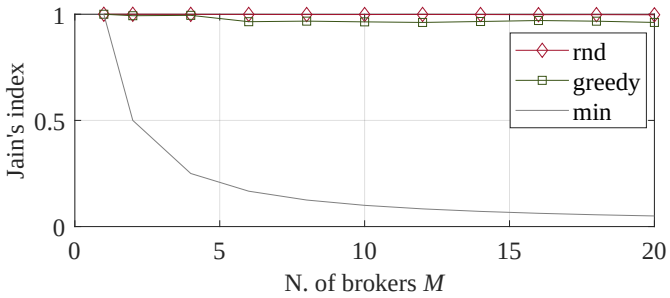

Fig. 18: Jain's index size for the social network scenario vs. cluster size, random-attach (rnd) and multi-session best-matching (greedy) strategies, $Ntop = 5000, Nsub = 5000, Nsxs = 10, \lambda_j = 1, \alpha = 1.13, Nses = 4, \gamma = 1.1$

Fig. 18 shows the Jain's index obtained by varying the number of brokers of the cluster as in Fig. 13. The maximum value of the index is 1 and it means perfect fairness. The minimum value is $1/M$ and it means the worst possible level of fairness, i.e. a configuration for which all the processing load is handled by a single broker, while the other brokers in the cluster are inactive. Fig. 18 includes the minimum value and we note that the greedy algorithm provides levels of fairness very similar to those offered by the random-attach strategy. The random-attach strategy has a Jain's index of about 0.99 regardless of the number of brokers. The Jain's index of the greedy algorithm reaches about 0.96 as the number of brokers increases.

B. IoT Scenario

In this section we report the results relevant to the IoT scenario described in Sec. III-B. Let us consider the Pl_i probability used by subscribers to choose the ith level of the

subscription tree (Fig. 6), given in Eq. 15; we assume: firstly, $Pl_1 = 0$ to avoid the presence of subscribers interested in every topic; secondly, a Zipf distribution for the remaining levels, because it allows to simply evaluate the impact of different Pl configurations by varying the α parameter of the Zipf. To simulate scenarios where subscribers interested in specific topics (e.g. `<roomId>/<sensorType>/<sensorId>`) are more than those interested in topic aggregates (e.g. `<roomId>/#`), the popularity ranking has a reverse order with respect to the tree levels, i.e. $Pl_i \geq Pl_{i-i}$.

$$Pl_i = \begin{cases} \frac{\frac{1}{(d-i+1)^\alpha}}{\sum_{j=1}^{d-1} \frac{1}{j^\alpha}}, & \text{if } i \geq 1 \\ 0, & \text{otherwise} \end{cases} \quad (15)$$

For example, for $\alpha = 1.5$ and a tree with depth $d = 4$, we have $Pl = [0, 0.122, 0.222, 0.656]$. Reducing $alpha$ to 0.5, we get $Pl = [0, 0.258, 0.293, 0.449]$, which is a condition with more subscribers interested in aggregates of topics.

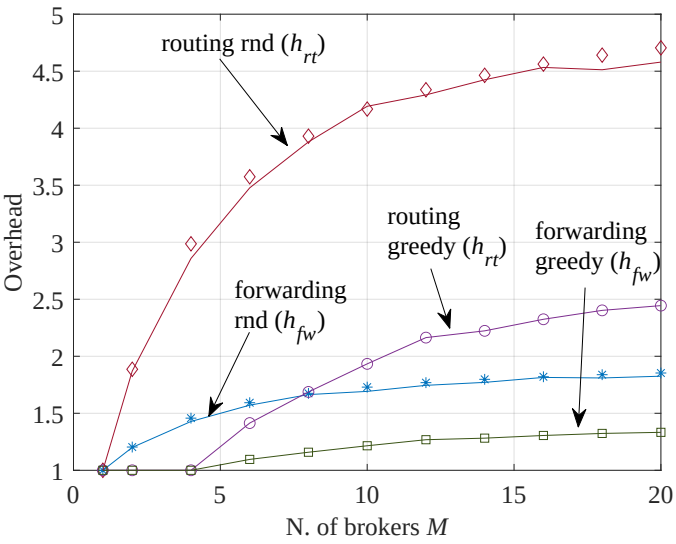

Fig. 19: Routing and forwarding overhead vs. cluster size M for the IoT scenario, random-attach (rnd) and multi-session best-matching (greedy) strategies, $Ntop = 5000, Nsub = 5000, d = 4, \alpha = 1.5, f = 30, \lambda_j = 1, Nses = 4, \gamma = 1.1$

In Fig. 19 and Fig. 20, we plot the overhead as a function of the cluster size M and of the number of sessions $Nses$, respectively. As in the social network scenario, also in the IoT one the overhead increases with the cluster size and the greedy algorithm performs better than the random one, especially as the number of sessions increases. Finally, Fig. 21 shows that the overhead, especially the routing one, decreases by increasing α. This is motivated by the fact that for low values of α there are more subscribers interested in aggregates of topics and this means that the average number of topics subscribed by each subscriber is higher, as well as the internal traffic. The greater $alpha$, the lower the number of subscribers interested in aggregates of topics as well as the internal traffic.

We conclude the section by reporting some general insights on the behavior of overheads that we have obtained from the analysis of social network and IoT scenarios:

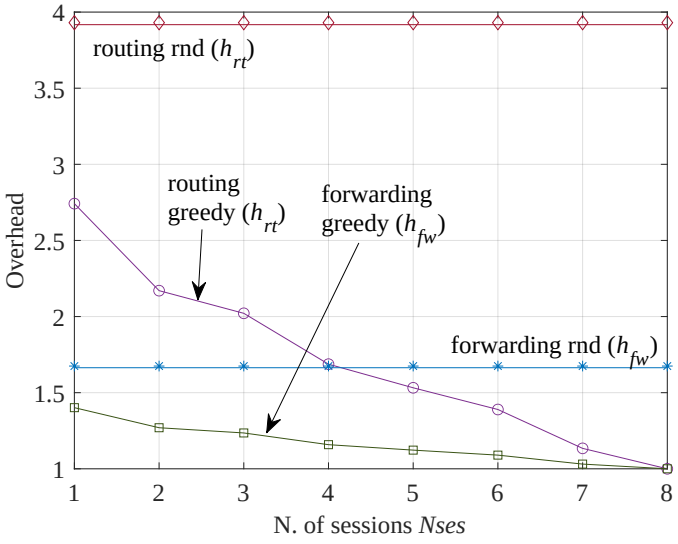

Fig. 20: Routing and forwarding overhead vs. number of sessions $Nses$ for the IoT scenario, random-attach (rnd) and multi-session best-matching (greedy) strategies, $Ntop = 5000, Nsub = 5000, d = 4, \alpha = 1.5, f = 30, \lambda_j = 1, M = 8, \gamma = 1.1$

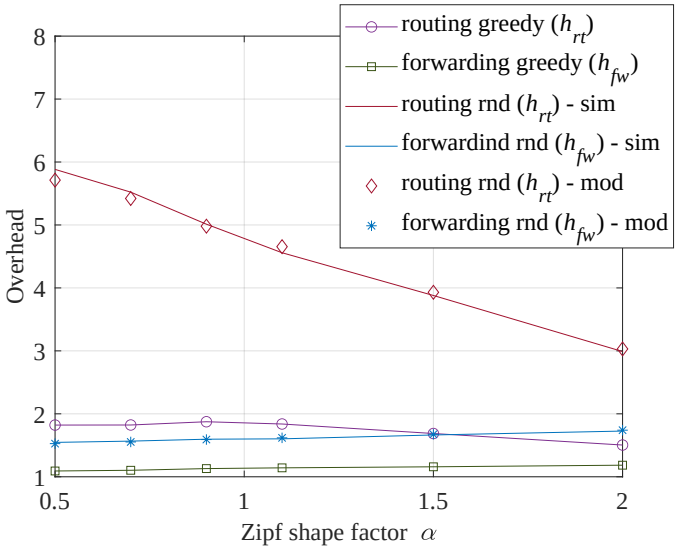

Fig. 21: Routing and forwarding overhead vs. Zipf shape parameter α for the IoT scenario, random-attach (rnd) and multi-session best-matching (greedy) strategies, $Ntop = 5000, Nsub = 5000, d = 4, M = 8, f = 30, \lambda_j = 1, Nses = 4, \gamma = 1.1$

1) The increase in the number of brokers corresponds to an increase in routing and forwarding overheads.
2) The amount of routing and forwarding overhead is inversely proportional to the input and output external traffic, respectively, which depend on the application scenario. For example, in the case of a few publishers and many subscribers we have low input and high output external traffic, therefore high routing overhead and low forwarding overhead.

3) The overhead is also inversely proportional to the skeweness of the topic popularity distribution. It is greater in the case of subscribers whose interests are evenly distributed over the available topics and it is lower in case of subscribers interested in a few popular topics.
4) The multi-session best-matching strategy reduces routing and forwarding overheads, achieving a good level of fairness. The reduction in overheads increases as the number of sessions increases.

C. Measurements on a real implementation

We conclude the performance evaluation by presenting real measurements made with VerneMQ running on our Kubernetes cluster, using an MQTT Benchmarking Tool [15], comparing the (default) random-attach strategy of VerneMQ with the proposed multi-session best-matching strategy. Differently from Fig. 2 in which we had a benchmark scenario simply made of 1000 publisher/subscriber couples, here we consider a social network scenario with 1000 publishers and 100 subscribers, each one interested in 10 topics.

The left plot in Fig. 22 shows, as a function of the number of brokers M, the maximum publication rate supported by the cluster such that the average message latency remains lower than 10ms; also included is the ideal linear scaling behavior. The middle plot shows the normalized value of the maximum publication rate with respect to the ideal linear scaling. The greedy algorithm is quite better than the default random one, as it supports a greater publishing rate. The central graph shows that for 4 brokers the random algorithm has a performance that is only 67% percent of the ideal one. The greedy algorithm reaches 95%. The down-up behavior of the greedy strategy in the central plot is due to the fact that we are using a maximum number of sessions $Nses = 4$, but when the number of brokers is less than 4, the number of sessions exploited is actually equal to the number of brokers, i.e. $real\ Nses = \min(M, 4)$. Therefore, by increasing the number of brokers we are practically increasing the number of sessions exploited as well, thus improving the optimization possibilities. After 2 brokers, this improvement is so great to overcome the worsening due to the increase in internal traffic with the number of brokers.

The plot on the right in Fig. 22 shows the average message latency with respect to the publication rate, in the case of a cluster composed by 4 brokers. The random-attach strategy has a sudden increase in message latency around 35 msg/s because brokers run out of CPUs, whereas the greedy one is able to limit the message latency up to 50 msg/s, thus demonstrating a better efficiency in handling MQTT traffic.

Fig. 23 describes the resource consumed by a VerneMQ broker in a cluster of 4 brokers in terms of CPU, memory and network traffic. The resources have been measured during a test in which we first send traffic (35 msg/s) by using the random-attach strategy, then paused and finally send again the traffic but using the greedy algorithm with 4 sessions. The graph on the right shows the traffic reduction provided by the greedy algorithm compared to the default random one. The graph on the left shows that this traffic reduction results in lower CPU usage which, in turn, leads to lower message latency (see the right Fig. 23 for 35 msg/s). Finally, the memory consumption reported in the central plot of Fig. 23 reveals that the memory growth due to the increased number of sessions used by the greedy algorithm is limited to 20 Mbytes, with respect to a base level of about 500 Mbytes relevant to the random algorithm.

Finally, we point out that the presented results are obtained by using a Poisson traffic model, for which message inter-times follows an Exponential distribution. This distribution simulates the behavior of *time-driven* publishing applications. We obtained similar performances by using also a LogNormal distribution, with a coefficient of variation equal to 4 and 8 (for Poisson it is equal to 1). The LogNormal distribution generates more bursty traffic with respect to the Poisson one and, therefore, better simulates the behavior of *event-driven* publishing applications [16]. The burstiness increases as the coefficient of variation increases. The Poisson/LogNormal results are similar because what affects the performance of the MQTT cluster is the mix of traffic generated by all publishers and the burstiness of the aggregated traffic fades by mixing so many publishers. Therefore, the considerations we made apply to both time-driven and event-driven publishing applications.

VI. Related Work

Publish/subscribe systems are used and studied for a wide range of applications [17]. Several proposals suggest the use of publish/subscribe systems specifically tailored to IoT and sensor applications ([18], [19], [20], [21]).

Sensor Andrew [20] is an infrastructure for Internet-scale sensing and actuation across a range of heterogeneous devices for data dissemination. MQTT-S [19] is a stripped-down version of MQTT, optimized for the small frame sizes in sensor networks, which are typically under 128 bytes. Gateways often found in sensor networks are used to translate MQTT-S to ordinary MQTT and forward messages to upstream brokers.

The OpenIoT project [22], instead, uses an ecosystem for mobile crowdsensing applications which relies on the Cloud-based Publish/Subscribe [18], a specifically designed content-based publish/subscribe with the ability to have mobile brokers, i.e. mobile publish/subscribe-enabled gateways. The OpenIoT project has compared their system to MQTT, but only considering qualitative metrics and messaging overhead, mostly in a protocol level analysis.

Zhang et al. [21] propose not to rely on the cloud for messaging, but instead to use a fully distributed system of message routers to deliver messages across a global IoT overlay network offering a publish/subscribe interface. These works, therefore, emphasize the need for publish/subscribe integration in sensor applications. Still, none of them focus on or provides a performance analysis of the internal traffic, nor address the scaling consequences caused by the internal traffic, which inevitably affect the cluster setup.

Previous work has been done in the field of performance evaluation of publish/subscribe systems ([23], [24]). However, comparisons of different systems in the literature usually do not consider the specific requirements, use-cases or traffic

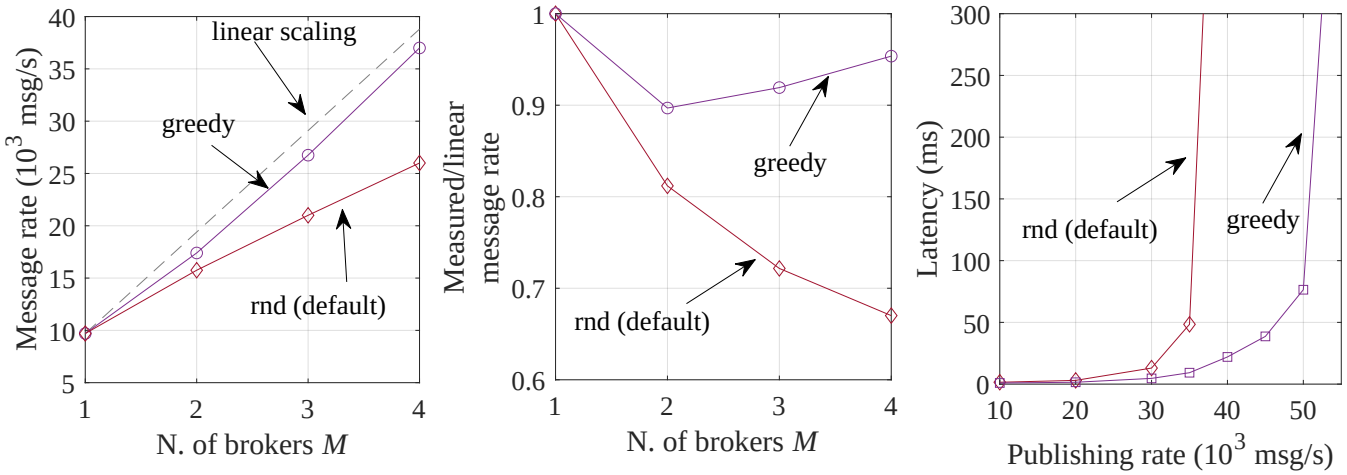

Fig. 22: VerneMQ performance for the social network scenario, default VerneMQ strategy (rnd) and multi-session best-matching (greedy) strategy. Maximum message rate for message latency $\leq$ 10ms vs. cluster size (left, middle), message latency vs. publishing rate for a cluster with 4 brokers (right). $Ntop = 1000, Nsub = 100, Nsxs = 10, \alpha = 1.13, Nses = 4, \gamma = 1.1$.

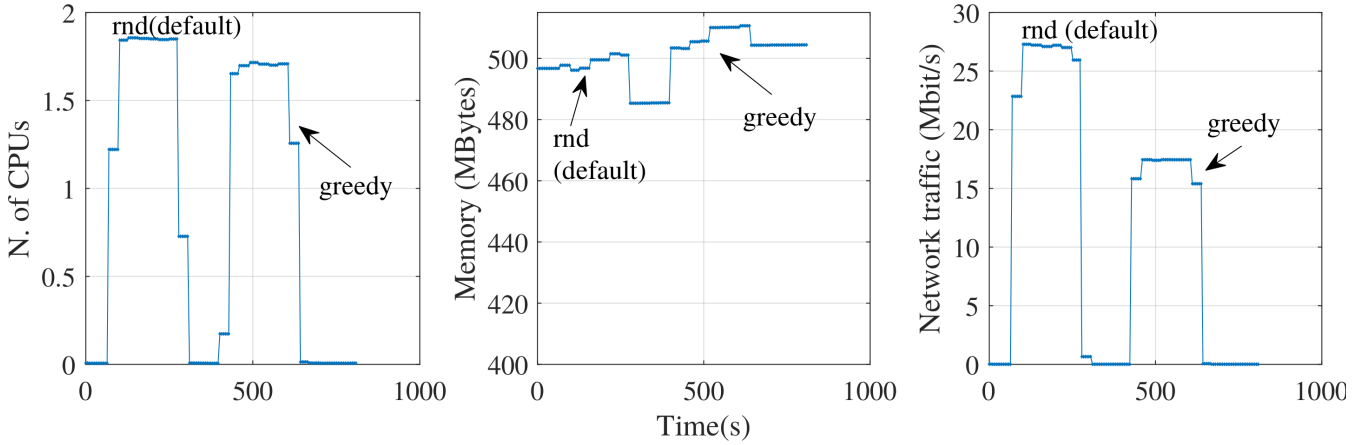

Fig. 23: VerneMQ performance in case of social network scenario versus time. Default VerneMQ strategy (rnd) and multi-session best-matching (greedy) strategies. Message rate 35 msg/s, $Ntop = 1000, Nsub = 100, Nsxs = 10, \alpha = 1.13, Nses = 4, \gamma = 1.1$.

patterns relevant to the phenomena at hand in this paper. In [23], a good overview of works on performance evaluation of publish/subscribe systems is given. In [24], the authors give a generic publish/subscribe benchmark based on scenarios in the field of logistics.

Recent studies focused on the concept of scalability, specifically on scaling the architecture to hundreds of IoT devices, concentrating their effort in building a broker with these capabilities ([25], [26], [27]). The authors of [25], designed a stateless broker that decouples the networking functionality of the broker from the state information it needs to maintain. EMMA [26], is an edge-enabled publish/subscribe middleware that addresses the challenges of the strict quality of service (QoS) requirements imposed by many applications that cannot be satisfied only by a cloud-based solution. In fact the goal of this middleware is to migrate MQTT clients to brokers, offloading the burden to a broker which is located in proximity of the clients in order to optimize QoS and reduce end-to-end latencies. Lastly, [27] examines the main research challenges to scale up a publish/subscribe architecture for upcoming IoT applications in 5G networks.

While these fundamental researches are essential for understanding the basic concepts and techniques required for building large scale publish/subscribe systems, they did not work and did not unveil the MQTT sub-linear scaling issue we have focused on this paper.

We conclude by mentioning that, in addition to MQTT, publish/subscribe systems can be based on other standard protocols, including AMQP [28], HTTP and CoAP [29]. AMQP can support more publish-subscribe patterns than the topic-based of MQTT, however, its implementation is more complex and has a higher overhead. HTTP-based systems use an HTTP server, which exposes a REST API through which it is possible to publish/receive messages, subscribe, etc. A subscription also includes a *notification URI* where the client wants to be connected to receive related publications. Being

based on the REST paradigm, each action requires the creation and release of an HTTP session and this increases the message latency compared to other solutions. CoAP is similar to HTTP but is faster than HTTP being based on UDP, is better designed for constrained devices and is capable of operating in scenarios with and without broker. Compared to MQTT, CoAP shows better performance in case of high packet loss rates because the underlying TCP used by MQTT is not suitable in these scenarios [30]. For IoT application, AMQP is not widely used because of its complexity, but many IoT platforms, such as ETSI oneM2M [31] and OMA NGSI [32] support HTTP, CoAP and MQTT.

VII. Conclusions

The use of MQTT broker clusters offers an undeniable advantage in terms of availability, reliability and scalability. However, as far as scalability is concerned, the expectation that the addition of resources will contribute to the same amount of additional capacity can be largely disappointed.

Many clustered applications scale out much better than MQTT, because they are simply client-server. MQTT, on the other hand, is designed for client-to-client applications, supported by a server mediation. Clients are connected to different servers in the cluster and therefore internal server-to-server traffic is generated to create the client-to-client path. This extra internal traffic, not usually existing in client-server applications, is a form of clustering overhead that leads the cluster to have a sub-linear scaling behavior, meaning that an increase in the number of brokers does not results in an equivalent improvement of application performance.

But how much sub-linear? It strongly depends on the application scenario. One must consider that each topic can at most generate $M-1$ internal publication flows, from the broker serving the publisher to all other brokers that may have subscribers. Therefore, if each topic has a small number of subscribers, such an amount of internal traffic is similar to that of the external traffic and sub-linearity can be significant; for example, it can waste 30-40% of resources, as in our experiments. If there are few topics used by many subscribers, the amount of internal traffic is relatively low compared to external traffic and sub-linearity is still present, but in a less accentuated way.

Designing a load-balancer that restores the linear scalability of the cluster may be hard, because subscribers are interested in more than one topic, because we still want to have a fair load distribution between brokers and because MQTT is stateful. Our proposal to address the problem is to distribute clients' subscriptions over multiple MQTT sessions. This opportunity drastically increases the optimization possibilities available to the load-balancer, which, together with our greedy algorithm, allows the cluster to scale almost linearly. An undeniable disadvantage is the increased complexity on both the broker and client side. As a result, future works could focus on finding other possible domains in addition to sessions' one, which could also increase the decision space of the load-balancer, but possibly providing a less complex implementation.

Acknowledgment

This work is supported in part by the H2020 EU-JP Fed4IoT project (www.fed4iot.org, EU contract n. 814918) and by the Italian MIUR PRIN Liquid_Edge project. The document reflects only the authors' view, European Commission, Japanese MIC and Italian MIUR are not responsible for any use that may be made of the information it contains

Andrea Detti is a professor of Wireless Networks and Cloud Computing at the University of Rome Tor Vergata. His research activity spans on different topics in the area of computer networks and copes with framework design, analytical modeling, performance evaluation through simulation and testbed. He is co-author of more than 80 papers on journals and conference proceedings, and participated to several EU funded projects with coordination and research roles. Currently is the research area is focused on Cloud Computing and IoT. (http://netgroup.uniroma2.it/people/faculties/andrea-detti/)

Ludovico Funari Ludovico Funari is a researcher at the University of Rome Tor Vergata. He received the master's degree in "ICT And Internet Engineering" in October 2019. His research activity includes IoT, Cloud and Edge computing. He has worked as a CNIT (Italian National Inter-University Consortium for Telecommunications) researcher for the UE H2020 "Fed4IoT" project. He is currently working for the "Liquid Edge Computing Based on Distributed Machine Learning and Millimetre-Wave Radio Access" research project.

Nicola Blefari Melazzi Nicola Blefari-Melazzi (http://blefari.eln.uniroma2.it/) is a full Professor of Telecommunications at the University of Roma Tor Vergata, where he served as Chair of the PhD program in Telecommunications Engineering, Chair of the undergraduate and graduate programs in Telecommunications Engineering and Chair of the Department of Electronic Engineering. He is currently the Director of CNIT (National Inter-University Consortium for Telecommunications, http://www.cnit.it/), a non-profit Consortium among 37 Italian Universities. More than 1,300 people, belonging to the participating universities, collaborate with CNIT, while the number of own-employees is more than 100. His research projects have been funded by Italian Ministries, by the Italian National Research Council, by major companies (e.g., Ericsson, Telecom Italia), by the ESA and by the EU. He has participated in 31 EU projects, playing the role of project coordinator for seven of them. He has been an elected member of the 5G Public Private Partnership association (https://5g-ppp.eu/), a 1.4 Billion Euro initiative established to create the next generation of networks. He evaluated many research proposals and projects in EU programs and served as TPC member, TPC Chair, General Chair and Steering Committee Chair for IEEE Conferences and guest editor for IEEE Journals. He is an area editor for Elseviers Computer Networks. He is author/co-author of about 240 papers. His research interests lie in the performance evaluation, design and control of telecommunications networks.

REFERENCES

[1] "Mqtt version 5.0," OASIS, Tech. Rep. [Online]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html

[2] HiveMQ. Reliable data movement for connected devices. [Online]. Available: https://www.hivemq.com/

[3] VerneMQ. Clustering mqtt for high availability and scalability. [Online]. Available: https://vernemq.com

[4] eMQTT. The massively scalable mqtt broker for iot and mobile applications. [Online]. Available: http://emqtt.io/

[5] Kubernetes (k8s): Production-grade container orchestration. [Online]. Available: https://kubernetes.io/

[6] Mosquitto. An open source mqtt broker. [Online]. Available: https://mosquitto.org/

[7] Rabbitmq. [Online]. Available: https://www.rabbitmq.com/

[8] H. Kwak, C. Lee, H. Park, and S. Moon, "What is twitter, a social network or a news media?" in *Proceedings of the 19th international conference on World wide web*. AcM, 2010, pp. 591–600.

[9] H. Liu, V. Ramasubramanian, and E. G. Sirer, "Client behavior and feed characteristics of rss, a publish-subscribe system for web micronews," in *Proceedings of the 5th ACM SIGCOMM conference on Internet Measurement*. USENIX Association, 2005, pp. 3–3.

[10] V. Setty, G. Kreitz, R. Vitenberg, M. Van Steen, G. Urdaneta, and S. Gimåker, "The hidden pub/sub of spotify," in *Proceedings of the 7th ACM international conference on Distributed event-based systems*. ACM, 2013, pp. 231–240.

[11] Y. Yu *et al.*, "On the inclusion probabilities in some unequal probability sampling plans without replacement," *Bernoulli*, vol. 18, no. 1, pp. 279–289, 2012.

[12] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: Modeling, design and experimental results," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 7, pp. 1305–1314, 2002.

[13] "Benchmark of mqtt servers," Scalagent, Tech. Rep., 2015. [Online]. Available: http://www.scalagent.com/IMG/pdf/Benchmark_MQTT_servers-v1-1.pdf

[14] "Mqtt topics & best practices - mqtt essentials: Part 5," HiveMQ, Tech. Rep.

[15] Mqtt benchmarking tool. [Online]. Available: http://netgroup.uniroma2.it/Andrea_Detti/papers/journals/mqtt_bench-master.zip

[16] N. Nikaein, M. Laner, K. Zhou, P. Svoboda, D. Drajic, M. Popovic, and S. Krco, "Simple traffic modeling framework for machine type communication," in *ISWCS 2013; The Tenth International Symposium on Wireless Communication Systems*. VDE, 2013, pp. 1–5.

[17] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec, "The many faces of publish/subscribe," *ACM computing surveys (CSUR)*, vol. 35, no. 2, pp. 114–131, 2003.

[18] A. Antonic, K. Roankovic, M. Marjanovic, K. Pripuic *et al.*, "A mobile crowdsensing ecosystem enabled by a cloud-based publish/subscribe middleware," in *2014 International Conference on Future Internet of Things and Cloud*. IEEE, 2014, pp. 107–114.

[19] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-sa publish/subscribe protocol for wireless sensor networks," in *2008 3rd International Conference on Communication Systems Software and Middleware and Workshops (COMSWARE'08)*. IEEE, 2008, pp. 791–798.

[20] A. Rowe, M. E. Berges, G. Bhatia, E. Goldman, R. Rajkumar, J. H. Garrett, J. M. Moura, and L. Soibelman, "Sensor andrew: Large-scale campus-wide sensing and actuation," *IBM Journal of Research and Development*, vol. 55, no. 1.2, pp. 6–1, 2011.

[21] B. Zhang, N. Mor, J. Kolb, D. S. Chan, K. Lutz, E. Allman, J. Wawrzynek, E. Lee, and J. Kubiatowicz, "The cloud is not enough: Saving iot from the cloud," in *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*. Santa Clara, CA: USENIX Association, Jul. 2015.

[22] J. Soldatos, N. Kefalakis, M. Hauswirth, M. Serrano, J.-P. Calbimonte, M. Riahi, K. Aberer, P. P. Jayaraman, A. Zaslavsky, I. P. Žarko *et al.*, "Openiot: Open source internet-of-things in the cloud," in *Interoperability and open-source solutions for the internet of things*. Springer, 2015, pp. 13–25.

[23] K. Sachs, *Performance modeling and benchmarking of event-based systems*. Sierke, 2011.

[24] K. Sachs, S. Kounev, J. Bacon, and A. Buchmann, "Performance evaluation of message-oriented middleware using the specjms2007 benchmark," *Performance Evaluation*, vol. 66, no. 8, pp. 410–434, 2009.

[25] S. Sen and A. Balasubramanian, "A highly resilient and scalable broker architecture for iot applications," in *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*. IEEE, 2018, pp. 336–341.

[26] T. Rausch, S. Nastic, and S. Dustdar, "Emma: Distributed qos-aware mqtt middleware for edge computing applications," in *2018 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2018, pp. 191–197.

[27] A. E. Redondi, A. Arcia-Moret, and P. Manzoni, "Towards a scaled iot pub/sub architecture for 5g networks: the case of multiaccess edge computing," *arXiv preprint arXiv:1902.07022*, 2019.

[28] S. Vinoski, "Advanced message queuing protocol," *IEEE Internet Computing*, vol. 10, no. 6, pp. 87–89, 2006.

[29] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," Internet Requests for Comments, RFC Editor, RFC 7252, June 2014. [Online]. Available: http://www.rfc-editor.org/rfc/rfc7252.txt

[30] D. Thangavel, X. Ma, A. Valera, H.-X. Tan, and C. K.-Y. Tan, "Performance evaluation of mqtt and coap via a common middleware," in *2014 IEEE ninth international conference on intelligent sensors, sensor networks and information processing (ISSNIP)*. IEEE, 2014, pp. 1–6.

[31] S. K. Datta, A. Gyrard, C. Bonnet, and K. Boudaoud, "onem2m architecture based user centric iot application development," in *2015 3rd International Conference on Future Internet of Things and Cloud*. IEEE, 2015, pp. 100–107.

[32] O. M. Alliance, "Ngsi context management," *OMA, OMA-TS-NGSI Context Management-V1 0*, 2012.