DynPubSub: A Peer To Peer Overlay For Topic-Based Pub/Sub Systems Deployed at the Edge

Chamseddine Bouallegue
chamseddine.bouallegue.1@ens.etsmtl.ca
Department of Software and IT Engineering
ETS Montreal / University of Quebec

Julien Gascon-Samson
julien.gascon-samson@etsmtl.ca
Department of Software and IT Engineering
ETS Montreal / University of Quebec

Abstract

There are more and more IoT devices that produce and consume and ever increasing amount of data. Publish-subscribe (Pub/Sub) is a well known paradigm that simplifies the task of exchanging messages, as it decouples the communication between the entities that emit and consume messages. While traditionally deployed in a centralized cloud-based manner, the different components of a pub/sub system can be deployed directly onto the *edge* devices, in a peer-to-peer manner, to achieve the required low latency for most IoT applications. In this poster, we propose DynPubSub, a new peer-to-peer network overlay for topic based pub/sub systems deployed at the edge. DynPubSub provides fault tolerance and scalability, and aims at minimizing the latency while respecting the constraints of the edge devices and networks.

Keywords: IoT, Edge Computing, Publish/Subscribe, Peer To Peer

ACM Reference Format:
Chamseddine Bouallegue and Julien Gascon-Samson. 2020. DynPubSub: A Peer To Peer Overlay For Topic-Based Pub/Sub Systems Deployed at the Edge. In *21st International Middleware Conference Demos and Posters (Middleware '20 Demos and Posters), December 7–11, 2020, Delft, Netherlands.*
https://doi.org/10.1145/3429358.3429373

1 Introduction

The publish/subscribe paradigm is well used across IoT applications, as it offers efficient and distributed content delivery [1] . While there are several *flavours* of publish/subscribe systems, the most common is *topic-based*, which relies on predefined *channels* in which all subscribers registered to a given topic will receive the messages published onto that topic. The MQTT (Message Queuing Telemetry Transport) protocol proposes a standardized implementation of a topic-based pub/sub messaging protocol that is reliable for connecting remote devices with minimal network bandwidth (Figure 1).

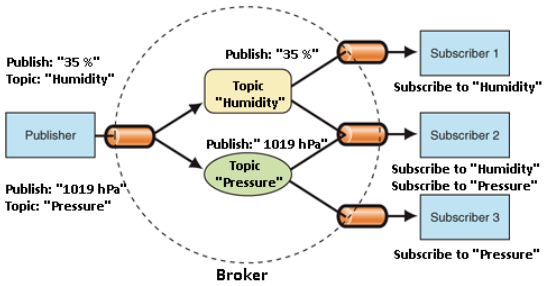

Figure 1. Pub/Sub Scenario

The infrastructure of a pub/sub system is usually composed of one or more brokers deployed in a centralized manner (e.g., in the cloud). However, given the stringent latency requirements for IoT applications, and the unreliability of some IoT network connections, there is a need for a more localized and distributed deployment of the communication infrastructure. Further, given that edge devices are getting more powerful, we envision deploying the topic-based pub/sub communication infrastructure directly onto the *edge* (IoT) devices, in a distributed, peer-to-peer, manner.

While several works have proposed peer-to-peer topic-based pub/sub systems (e.g. Tera [2], Scribe [3] which is built on top of the Pastry distributed infrastructure [4], SpiderCast [5], and PolderCast [6]), DynPubSub aims at addressing the specific challenges of the IoT context by optimizing the deployment of the infrastructure onto the edge devices. Overall, DynPubSub proposes a dynamic edge overlay partitioning approach in which the topics of the pub/sub infrastructure are dynamically (re)mapped onto the various edge devices according to the current network load and capabilities of the various edge nodes, with latency minimization as the optimization goal. Further, we provide a library that is API-compatible with the well-used Node.js MQTT library (mqtt.js), thereby allowing developers to transparently switch between a centralized pub/sub deployment and the peer-to-peer DynPubSub architecture without needing to modify their code.

Middleware '20 Demos and Posters, December 7–11, 2020, Delft, Netherlands

ACM ISBN 978-1-4503-8202-1/20/12.
https://doi.org/10.1145/3429358.3429373

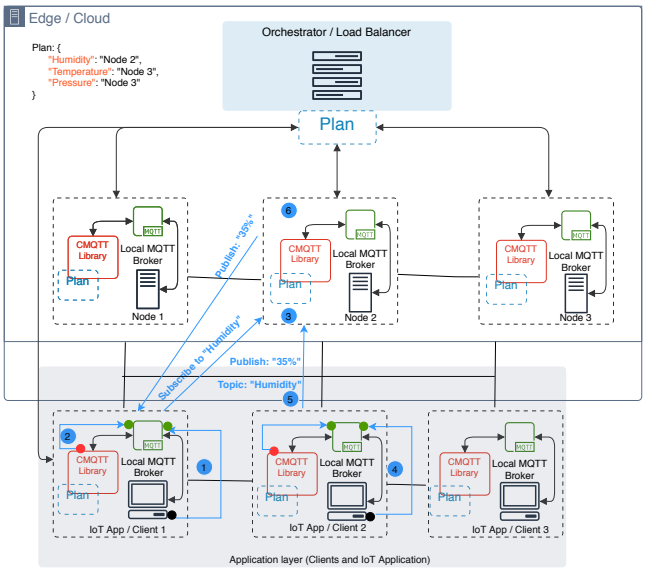

Figure 2. Architecture and System Data Flow

2 System Model and Architecture

Our architecture (Figure 2) is composed of three layers: (1) the application layer, which comprises IoT client applications ; (2) the edge peer to peer overlay layer, which interconnect the system nodes and route the messages, and (3) the orchestrator layer, which gathers real-time performance data from the edge layer and dynamically rebalances the load by remapping the pub/sub topics onto the edge layer. We assume that the edge layer infrastructure (2) can be deployed onto the same nodes as the application layer (1), although this is not a requirement.

The orchestrator periodically receives aggregated data from the edge layer and periodically generates a plan which minimizes the average latency while respecting the bandwidth constraints of the edge nodes. The plan specifies which edge node should be managing each topic defined in the pub/sub service. The `cmqtt.js` client-side library handles the use of the plan to transparently dispatch publication and subscription requests.

Figure 2 also illustrates the flow of messages for a client performing a subscription and publication, using the example scenario depicted in Figure 1. Considering the example plan shown in Figure 2, (1) Client 1 subscribes to topic "Humidity", its local MQTT broker accepts the subscription. (2) The local MQTT broker looks at the plan via the "CMQTT" library. (3) Local MQTT broker transmits the subscription to "node 2" as it is responsible for topic "Humidity". (4) Client 2 publishes a message on topic "Humidity", its local MQTT broker accepts the publication. (5) The local MQTT broker consults the plan via the "CMQTT" library and transmit the message to "node 2". (6) Finally, the local MQTT broker of the "node 2" publishes the message from client 2 to the client 1 (subscriber).

3 Implementation

As mentioned, we aim at providing full API compatibility with the Node.js MQTT library (mqtt.js), so that DynPubSub can be used as a simple drop-in replacement. Listing 1 depict a simple pub/sub client that corresponds to Figure 1.

Listing 1. Publish/Subscribe Sample Code

```
1  var cmqtt=require("../cmqtt");
2  //imports the library
3  var client=cmqtt.connect("mqtt://192.168.1.15:1883");
4  //connects to the orchestrator
5  client.on('connect',function(){});
6  //uses connect callback
7  function publishHumidity(){
8      let hum=35;
9      client.publish("Humidity",hum.toString()+'%');
10 }
11 client.on('connect',function(){
12     client.subscribe("Humidity");
13     //subscribes to topic "Humidity"
14 });
15 client.on('message',function(topic,message){
16     console.log(message.toString());
17 });
```

4 Evaluation Strategy

We plan to evaluate our system on a high number of (Raspberry Pi) and test the performance of the proposed approach in terms of scalability, fault tolerance and latency.

5 Conclusion

In this paper, we presented a global overview of DynPubSub, a peer to peer network overlay for topic-based pub/sub systems deployed at the edge, which aims at reducing the latency while considering the bandwidth as major constraint.

References

[1] Kato D. Kunieda K. Yamada K. Michiardi P Elkhiyaoui, K. A scalable interest-oriented peer-to-peer pub/sub network. *2009 IEEE Ninth International Conference on Peer-to-Peer Computing*, pages 204–211, 2009.

[2] Vivien Quéma Roberto Baldoni, Roberto Beraldi. Tera: topic-based event routing for peer-to-peer architectures. *DEBS 2007*, pages 2–13, 2007.

[3] M. Castro ; P. Druschel ; A.-M. Kermarrec ; A.I.T. Rowstron. Scribe: a large-scale and decentralized application-level multicast infrastructure. *IEEE Journal on Selected Areas in Communications*, 20(8):1489–1499, 2002.

[4] A. Rowstron and P. Druschel. Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems. *Middleware 2001*, 2001.

[5] Y. Tock G. Chockler, R. Melamed and R.Vitenbergl. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. *In DEBS, 2007*, 2007.

[6] Roman Vitenberg Spyros Voulgaris Vinay Setty, Maarten Steen. Poldercast: Fast, robust, and scalable architecture for p2p topic-based pub/sub. *Middleware 2012*, pages 271–291, 2012.