Adaptive Mediation for Data Exchange in IoT Systems

Andrew Chio[1], Georgios Bouloukakis[1], Cheng-Hsin Hsu[2], Sharad Mehrotra[1], Nalini Venkatasubramanian[1]
[1]UC Irvine, Donald Bren School of Information and Computer Science, USA
[2]Department of Computer Science, National Tsing Hua University, Taiwan
achio,gboulouk@uci.edu,chsu@cs.nthu.edu.tw,sharad,nalini@ics.uci.edu

ABSTRACT

Messaging and communication is a critical aspect of next generation Internet-of-Things (IoT) systems where interactions among devices, software systems/services and end-users is the expected mode of operation. Given the diverse and changing communication needs of entities, the data exchange interactions may assume different protocols (MQTT, CoAP, HTTP) and interaction paradigms (point to point, multicast, unicast). In this paper, we address the issue of supporting adaptive communications in IoT systems through a mediation-based architecture for data exchange. Here, components called mediators support protocol translation to bridge the heterogeneity gap. Aiming to provide a placement of mediators to nodes, we introduce an integer linear programming solution that takes as input: a set of Edge nodes, IoT devices, and networking semantics. Our proposed solution achieves adaptive placement resulting in timely interactions between IoT devices for larger topologies of IoT spaces.

CCS CONCEPTS

• **Computer systems organization** → **Heterogeneous (hybrid) systems**; • **Hardware** → **Sensor applications and deployments**; • **General and reference** → *Performance*; • **Software and its engineering** → *Message oriented middleware.*

KEYWORDS

Internet of Things, Operator Placement Problem, Protocol Mediators, Middleware

ACM Reference Format:
A. Chio, G. Bouloukakis, C. Hsu, S. Mehrotra and N. Venkatasubramanian. 2019. Adaptive Mediation for Data Exchange in IoT Systems. In *18th Workshop on Adaptive and Reflexive Middleware (ARM '19), December 9–13, 2019, Davis, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3366612.3368122

1 INTRODUCTION

Developing Internet of Things (IoT) applications requires the combination of heterogeneous IoT devices in smart spaces [3, 18]. For example, to regulate the temperature of a building according to rooms' occupancy, an application that analyzes data coming from Bluetooth beacons, WiFi access points or motion sensors can be developed. This input data can be subsequently used to estimate the occupancy level of rooms and actuate the building's HVAC appropriately. However, the involved IoT devices may employ different APIs and protocols, data formats and data information semantics. Existing interoperability solutions enable application developers to bridge IoT devices by relying on (i) service buses, proxies or gateways [8, 9, 13]; (ii) connector wrappers or mediating adapters [1, 3, 9, 13, 18]; and (iii) Cloud platforms [5, 14].

While these solutions enable data exchange, they introduce an overhead cost in terms of end-to-end delay between the involved IoT devices. In particular, developers tend to focus only on dealing with the heterogeneity problem while they ignore the deployment of the needed artifacts (gateways, mediators, etc). The most common practice is either to use a Cloud platform or to deploy the needed artifacts in the Cloud using the APIs of well known Cloud providers such as Amazon, Google, etc. Hence, IoT devices of the same space must interact with each other via the Cloud, which results in higher end-to-end delays between IoT devices. Such delays depend on the number of IoT devices, the available bandwidth, data size and frequency of messages. High delays may limit the development of dependable IoT applications, especially those that carry mission-critical information to improve public safety.

For instance, during a fire in a smart building, an emergency dispatch process is activated by sending a team of Fire Fighters (FFs) to the building. The FFs and their Incident Commander (IC) bring their own equipment (smart sensors and devices) to the site and they require up-to-date situational awareness information. Data (e.g., temperature, smoke levels and occupancy) derived from the building's IoT devices along with static information (e.g., building floor plans) can be leveraged by the stakeholders (e.g., IC, FFs, building occupants) to ensure their safety. However, advanced interoperability solutions are required to interconnect their devices with the ones in the smart building. Leveraging a Cloud platform or deploying the needed artifacts in the Cloud may result in high end-to-end delays between IoT devices. Such delays can be even higher for emergencies because the network may become highly congested and partially unavailable.

To deal with the heterogeneity in these emergency response scenarios, we rely on the DeXMS (*Data eXchange Mediator Synthesizer*) framework [3] that synthesizes mediators in an automated manner for bridging heterogeneous IoT devices. DeXMS does not dictate where to deploy the mediators, and the current paper studies the *mediator placement problem* which explores how adaptive mediation can be enabled in an IoT space. In particular, this is done by taking into account bandwidth, data frequency and data size constraints in order to place the synthesized mediators to a set of

ARM '19, December 9–13, 2019, Davis, CA, USA

ACM ISBN 978-1-4503-7030-1/19/12…$15.00
https://doi.org/10.1145/3366612.3368122

nodes and ensure timely data exchange between IoT devices. In particular, we propose an integer linear programming placement (ILP) solution, and compare it against other baseline algorithms– the results clearly show the efficiency of our solution. The main contributions of this paper are:

(1) Introducing the *mediator placement problem* as a formal model that takes into account network resources and IoT device attributes (Section 3).
(2) Presenting baseline heuristics based on distance and best fit decreasing bandwidth, and proposing our ILP placement algorithm for adaptive mediation between IoT devices (Section 4).
(3) Comparing the different placement approaches via their execution time and the resulted end-to-end delays for different topologies including heterogeneous IoT device interactions (Section 5).

2 MEDIATORS FOR ADAPTIVE DATA EXCHANGE

IoT devices (i.e., *Things*) employ middleware-layer protocols such as MQTT, CoAP, ZeroMQ and more, to interact with each other. These protocols support different Quality of Service (QoS) semantics; they define multiple data-serialization formats (e.g., JSON, XML, protobul, etc.) and different payloads suitable for constrained or healthy devices; and they follow different interaction paradigms such as client/server, publish/subscribe and streaming. IoT systems include heterogeneous Things employing any of the above protocols. In many cases, new heterogeneous Things may be needed to be added to an IoT system in an on-demand fashion. For instance, in the fire fighting scenario, the IoT equipment of FFs must interact with the devices in an IoT space. Hence, generic and automated solutions are required to enable data exchange in such IoT systems.

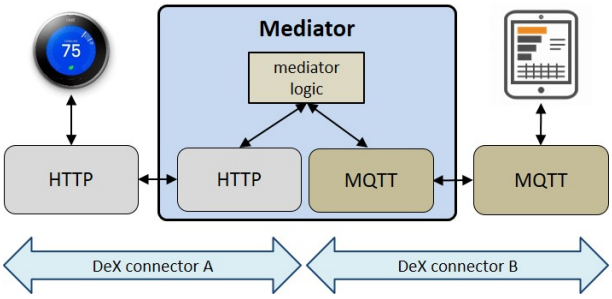

Figure 1: Enabling data exchange via mediators.

In this paper we leverage DeXMS [3] to synthesize software mediators that bridge heterogeneous Things. As depicted in Fig. 1, DeXMS relies on the Data eXchange (DeX) API, which implements `post` and `get` primitives for sending/receiving messages using existing IoT protocols such as CoAP, MQTT, XMPP, etc. In Fig. 1, the mediator converts temperature data coming from a smart thermostat (in JSON format through the HTTP protocol) to be received from a FF's dashboard (in XML format through the MQTT protocol). Considering a set of heterogeneous Things that have to interconnect with Things deployed in an IoT space, DeXMS accepts as input their input/output data representation models and synthesizes the required mediators. More details on DeXMS can be found in [3].

3 PROBLEM FORMULATION

In this section, we introduce the *mediator placement problem* which utilizes a set of Edge nodes to enable adaptive mediation for exchanging data between Things. Figure 2 shows a space where things and nodes are located. We illustrate four interactions whose mediators have already been assigned to nodes, and introduce the used notations below.

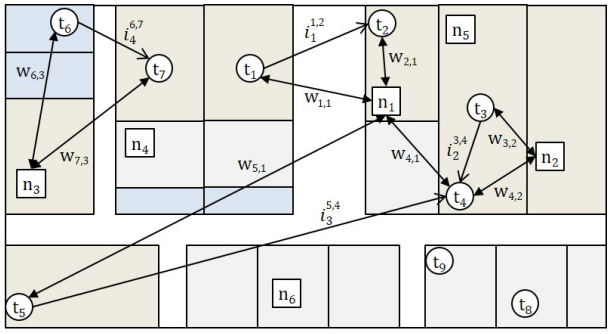

Figure 2: Heterogeneous IoT devices (*Things*) and computational resources (*Nodes*) located in an IoT space.

3.1 System and Resources Model

$T = \{t_i : i \in 1 \ldots |T|\}$ is the set of things in the system. In the sample fire fighting scenario, this set consists of the sensors that may be located in a building (e.g., temperature sensors, smoke sensors, etc.), their corresponding receivers, as well as any equipment that FFs carry before entering the burning building (e.g., heart monitor, location sensor, etc.). We assume that FFs will be assigned to some particular areas, allowing us to approximate their locations. When FFs are relocated to other areas, a new instance of the mediator placement problem needs to be solved to accommodate the FFs' mobility. We assume that a thing t_i has the following key attributes:

(1) location $l(t_i)$, which is a vector denoting the coordinates at which t_i is located. We assume that we will work in a 3D environment, and thus, the length of this vector is 3.
(2) protocol $p(t_i)$, which is a string denoting the IoT protocol (CoAP, MQTT, etc.) used by the application running on t_i. We assume that for every thing $t_i \in T$, there is only one protocol used.

$N = \{n_i : i \in 1 \ldots |N|\}$ is the set of nodes in the system. These nodes are responsible for hosting mediators to enable things to interact. In our fire fighting scenario, this set consists of any computational resources (e.g., Raspberry Pis) that are available in an IoT space to deploy software artifacts. We define the location $l(n_i)$ of a node analogously to the location of a thing.

$I = \{i_j^{kl} : j \in 1 \ldots |I|, k, l \in 1 \ldots |T|\}$ is a set of interactions that will take place in the system. An interaction i_j^{kl} denotes that interaction j consists of thing t_k sending a message to thing t_l. With the above definition, we can model a two-way interaction with i^{kl}

and i^{lk}. We also make an assumption that for each interaction $i_j^{kl} \in I$, $p(t_k) \neq p(t_l)$. Interactions will additionally have the following key attributes:

(1) message size $\lambda(i_j)$, which is a scalar denoting the size of a message sent.
(2) frequency $\gamma(i_j)$, which the number of times a message of size $\lambda(i_j)$ is sent. We let this frequency follow a Poisson process.

$W = \{w_{jk} : j \in 1 \ldots |T|, k \in 1 \ldots |N|\}$ is a graph that denotes the available bandwidth of links between Things and Nodes. In particular, w_{jk} represents the available bandwidth of the link between thing t_j and n_k. We do not include links between nodes because it is not possible for any interaction to traverse through such links.

Mediator Placement Problem. The Mediator Placement Problem can be formally stated as follows: Given a set of Things, Nodes, Interactions, and (Bandwidth of) Links, assign each mediator to a node so that the total bandwidth used over every link is less than or equal to the link's available bandwidth and the total delay is minimized.

3.2 Objective Function

Next, we describe how the total end-to-end delay is calculated to quantify the quality of a particular placement. There are two main components relevant to finding the total end-to-end delay for a given placement.

Δ_{e2e} is the end-to-end delay for the entire system. We calculate this quantity using the following equation:

$$\Delta_{e2e} = \Sigma_j \Delta_{proc}^{(j)} \tag{1}$$

which sums up the transmission delay and the propagation delay for all interactions. We note that the end-to-end delay does not include the time taken to generate mediators because we assume that such mediators have already been generated. $\Delta_{proc}^{(j)}$ is the propagation delay for interaction i_j^{kl} and a node n_m on which a mediator is placed. We calculate this value with the equation:

$$\Delta_{proc}^{(j)} = \frac{||l(t_k) - l(n_m)||}{c} + \frac{||l(n_m) - l(t_l)||}{c} \tag{2}$$

where c is the wave propagation speed, which is the speed at which data travels through a medium. That is, this formula takes the distance that the data must travel and divides it by the wave propagation speed. Notice that we assume the conversions between heterogeneous Things are relatively short, and thus the queueing and processing delays are negligible [3].

We note that the processing delay is small, since mediators are lightweight [3]. In this work, we will also assume that the queueing delay is negligible, and so we do not include it as part of the end-to-end delay. In addition, we note that the processing delay is negligible since the mediators we develop are lightweight.

3.3 Constraints

Below are the two constraints that must be satisfied for any particular placement to be valid.

(1) Bandwidth Constraint. Given a placement, for every link, comprised of a Thing and a Node, its used bandwidth must be less than or equal to its available bandwidth. Given an interaction i_j^{kl} and node n_m to place its corresponding mediator, i_j will consume $\lambda(i_j) \times \gamma(i_j)$ bandwidth on the two links from t_k to n_m and from t_l to n_m. This constraint can be checked efficiently with the algorithm given in Algorithm 1.
(2) Injective Constraint. Given a placement, for every interaction, there must be only one node assigned to that interaction.

Algorithm 1: Check Bandwidth Constraints

Input: A placement P, of Interactions I to Nodes N, graph W
Output: True if the bandwidth constraint is satisfied for P, otherwise False

```
for t_u ← T do
    for n_v ← N do
        bandwidthUsed ← 0;
        for i_j^kl ← I do
            if u = k ∧ v = P[j] then
                bandwidthUsed ←
                  bandwidthUsed + λ(i_j) * γ(i_j);
        if bandwidthUsed > w_uv then
            return False;
    return True;
```

4 MEDIATOR PLACEMENT ALGORITHMS

In this section, we propose several algorithms to solve the mediator placement problem.

4.1 Baseline Algorithms

Best Fit Decreasing Bandwidth Placement. This heuristic algorithm places mediators to nodes only when the bandwidth constraint would not be violated, but selects a node such that the bandwidth *wasted* by the placement is minimized. We consider interactions in decreasing order of their data rates and place mediators on nodes that would saturate more bandwidths earlier.

First, we define the waste of a mediator assignment to node n_m for interaction i_j^{kl} as a difference of two sums, which we will refer to as SUM1 and SUM2. SUM1 is the sum of the available bandwidths from t_k to n_m and from t_l to n_m, which can be denoted with $w_{km} + w_{lm}$. SUM2 is the sum of the bandwidths used by all other interactions sending a message from t_k to n_m and from t_l to n_m, added to the data rate used by i_j. We can denote SUM2 using the formula: $U_{km} + U_{lm} + 2 \times \lambda(i_j) \times \gamma(i_j)$, where U represents the total bandwidths used by all other interactions. We note that since there are two links involved in every interaction, we must multiply the data rate by two to properly count the total bandwidth used. Collectively, the waste produced by placing on some particular node n_m is described with: $w_{km} + w_{lm} - (U_{km} + U_{lm} + 2 \times \lambda(i_j) \times \gamma(i_j))$. We give the psuedocode of this algorithm in Algorithm 2.

Greedy Distance Placement Algorithm. This is a heuristic algorithm that places mediators to nodes such that the bandwidth constraints on all the links are not violated, and the distance travelled by the data is minimized. That is, for every interaction, we

Algorithm 2: Best Fit Decreasing Bandwidth Placement

Input: set of Things T, set of Nodes N, set of Interactions I, graph W
Output: Placement P, or NULL
$P \leftarrow$ an empty map;
$U \leftarrow$ an isomorphic graph to W, with all edge weights = 0;
for $i_j^{kl} \leftarrow I$, *sorted in decreasing data rate order* **do**
  $c \leftarrow NULL$, the node to place a mediator for i_j;
  $bandwidthUsed \leftarrow \infty$, the bandwidth used by placing a mediator on c
  $A \leftarrow \emptyset$ be an empty set of nodes;
  for $n_m \leftarrow N$ **do**
    if $(U_{km}+\lambda(i_j)*\gamma(i_j) \leq W_{km}) \wedge (U_{lm}+\lambda(i_j)*\gamma(i_j) \leq W_{lm})$ **then**
      add n_m to A;
  if $A = \emptyset$ **then**
    return NULL;
  for $n_m \leftarrow A$ **do**
    $b \leftarrow w_{km} + w_{lm} - (U_{km} + U_{lm} + 2 * \lambda(i_j) * \gamma(i_j))$
    if $b < bandwidthUsed$ **then**
      $c \leftarrow m$;
      $bandwidthUsed \leftarrow b$;
  $P[j] \leftarrow m$;
  $U_{kc} = U_{kc} + \lambda(i_J) * \gamma(i_j)$;
  $U_{lc} = U_{lc} + \lambda(i_J) * \gamma(i_j)$;

Algorithm 3: Greedy Distance Placement

Input: set of Things T, set of Nodes N, set of Interactions I, graph W
Output: Placement P, or NULL
$P \leftarrow$ an empty map;
$U \leftarrow$ an isomorphic graph to W, with all edge weights = 0;
for $i_j^{kl} \leftarrow I$ **do**
  $c \leftarrow NULL$, the closest node to t_k and t_l;
  $totalDist \leftarrow \infty$, the distance from t_k to c to t_l;
  $A \leftarrow \emptyset$ be an empty set of nodes;
  for $n_m \leftarrow N$ **do**
    if $(U_{km}+\lambda(i_j)*\gamma(i_j) \leq W_{km}) \wedge (U_{lm}+\lambda(i_j)*\gamma(i_j) \leq W_{lm})$ **then**
      add n_m to A;
  if $A = \emptyset$ **then**
    return NULL;
  for $n_m \leftarrow A$ **do**
    $d \leftarrow dist(l(t_k), l(n_m)) + dist(l(n_m), l(t_l))$;
    if $d < totalDist$ **then**
      $c \leftarrow m$;
      $totalDist \leftarrow d$;
  $P[j] \leftarrow m$;
  $U_{kc} = U_{kc} + \lambda(i_J) * \gamma(i_j)$;
  $U_{lc} = U_{lc} + \lambda(i_J) * \gamma(i_j)$;

place a mediator to the closest node by distance such that the constraints are met. Pseudocode is provided in Algorithm 3.

4.2 Integer Linear Programming Placement

We define a decision variable X, which is a matrix of size $|I| \times |N|$, such that an entry $X_{jk} = 1$ iff interaction i_j uses node n_k, otherwise 0. To compute the total bandwidth between thing t_i and node n_j, we sum the data rates of all interactions that are both: (1) using t_i as part of the interaction, and (2) using n_j to host the corresponding mediator. In order to express whether some thing t_i is part of an interaction, we introduce a boolean matrix V with size $|I| \times |T|$, whose entries $V_{ji} = 1$ iff thing t_i is a part of interaction i_j, otherwise 0. Thus, we formulate the bandwidth constraints as:

$$\forall t_p \forall n_q \Sigma_{i_j} V_{jp} \times X_{jq} \times \lambda(i_j) \times \gamma(i_j) \leq w_{pq}.$$

Since an entry $V_{jp} = 1$ iff t_p is a part of i_j, the links we take a summation of must include at least t_p as one endpoint. Likewise, since an entry $X_{jq} = 1$ iff n_q will be used to host a mediator for i_j, this same link must have n_q as another endpoint. We multiply these two entries together to ensure that both must be true. In such a case, we are allowed to add the bandwidth used by i_j to the summation. Otherwise, one of these entries must be 0, which will cause the summation to stay the same.

The second constraint we consider is that a mediator is assigned exactly one node. We formulate this constraint as follows:

$$\forall i_j \Sigma_{n_i} X_{ji} = 1.$$

Suppose that for some interaction $i_j, \Sigma_{n_i} X_{ji} \neq 1$. In the case that the sum is equal to 0, there must not have been any nodes assigned to host this mediator, thus creating an invalid placement. Similarly, in the case that the sum is greater than 1, there must be at least 2 nodes that were assigned to host this mediator, which is also an invalid placement. Thus, the sum of X_{ji} over all nodes n_i must be 1. With the above derivation, our ILP problem is formally written as:

$$\begin{array}{ll} \text{minimize} & \Delta_{e2e} \\ \text{subject to} & \forall t_p \forall n_q \Sigma_{i_j} V_{jp} \times X_{jq} \times \lambda(i_j) \times \gamma(i_j) \leq w_{pq} \\ & \forall i_j \Sigma_{n_i} X_{ji} = 1. \end{array}$$

5 EXPERIMENTS

In this section, we will describe the experiments used to quantify both the running time and quality of solutions produced by the presented algorithms. We use the random placement algorithm as a baseline. Additionally, we assume that if an algorithm is unable to place a mediator to a node such that all constraints specified in Section 3 are satisfied, the algorithm will report failure, and the attempt is not recorded.

Experimental Setup. We consider two classes of topologies: (1) TOPOLOGY1 is a simple case consisting of 10 things and 10 nodes and (2) TOPOLOGY2 is a more complex case consisting of 100 things and 10 nodes. In both topologies, all locations $l(t_i)$ and $l(n_j)$ are chosen from the range $[0, 1000]$ (meters). The interactions are generated randomly by selecting two things with different IoT protocols (thereby requiring a mediator). We select the data rate randomly between $[0.8, 2]$, and the bandwidth available is dependent upon the total number of things in the topology.

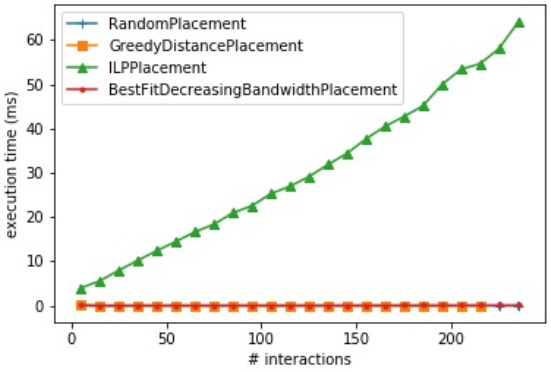

Figure 3: Execution time for TOPOLOGY1

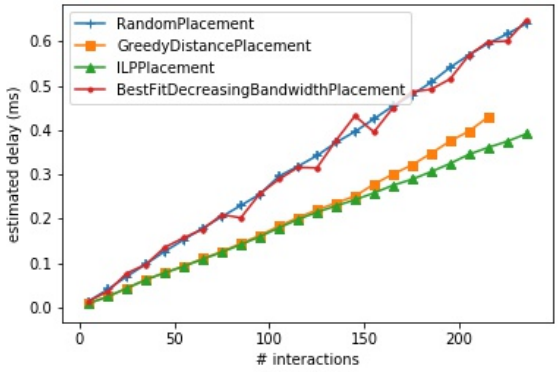

Figure 4: Estimated total delay TOPOLOGY1.

We record two performance measures: (1) the execution time and (2) the estimated response time (Equation 1), while varying the number of interactions in a given topology from 5 to 245 in increments of 10 and use the timeit module [1] built into python. Each experiment was repeated ten times for each interaction size, and the minimum amount of time taken was recorded (as advised in the documentation for the timeit module). After timing each algorithm on a topology, we record the quality of the solution produced using the estimated response time.

We note that although we do not explicitly test the extreme case, where all mediators are deployed to one node, it is easy to reason that such a placement will introduce a single point of failure, along with significantly increasing the total end-to-end delay.

Results Analyzing the experiments done on TOPOLOGY1 first, we can see that the ILP solution took the longest to run, and all other algorithms had negligible runtimes. In particular, Figure 3 shows that the runtime of the ILP grows linearly with the number of interactions. Connecting this with Figure 4, we can see that the ILP has the best estimated response time, followed by the greedy placement, and then finally the random and best fit decreasing algorithms. We note that while the greedy algorithm could find nearly optimal solutions, as the number of interactions increases, the difference between the greedy placement and the ILP placement widens, and at a certain point, the greedy algorithm simply fails to place mediators. This failure to place mediators is caused by not considering the bandwidth remaining on links. Thus, the other two algorithms are able to place a larger number of mediators because they evenly distribute the mediators.

TOPOLOGY2 considers a case with a larger number of things; we record the execution time and the estimated response time in Figures 5 and 6. Similar to TOPOLOGY1, we can see that the execution time of the ILP is much larger than the execution time of the other algorithms. We connect this to the estimated response time in Figure 6, where once again, the ILP solution outperforms the others. We note that the other three algorithms were not able to find solutions for larger numbers of interactions. In particular, the greedy algorithm failed first, as is expected, since it does not consider the remaining bandwidth on each link. On the other hand, the best fit decreasing bandwidth algorithm which does take into

[1] https://docs.python.org/3.7/library/timeit.html

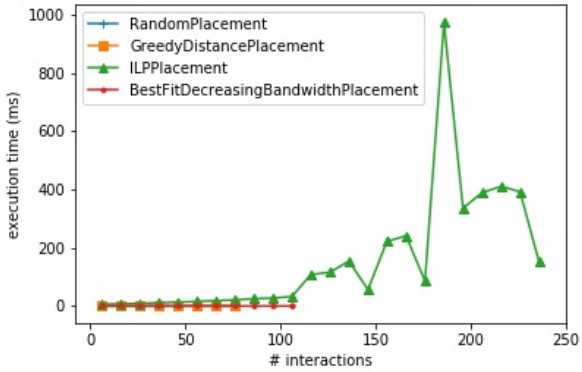

Figure 5: Execution time for TOPOLOGY2.

account bandwidth, does not do well in terms of the response time; the tradeoff is that it is able to place a larger number of mediators compared to the greedy and random algorithms. An additional interesting point to make in Figure 6 is the relationship between the random and the best fit decreasing algorithms. Although their execution times are roughly the same, the best fit decreasing algorithm is able to place a larger number of mediators. This is caused by the way the two algorithms select nodes for placement, and the amount of bandwidth available on links. Since the best fit decreasing algorithm attempts to minimize the amount of "wasted" bandwidth, it is able to place more mediators before failing.

6 RELATED WORKS

Much of elated work in the operator placement problem has studied the context of in-netowrk query processing and WSN, where efficient placement is the key criteria; these algorithms differ in their initial assumption and optimization goals.

One large class of proposed algorithms start with an assumption of an operator tree.. Under this assumption, Bonfils et. al. [2] introduces a neighbor exploration strategy that iteratively moves operators until they find a local minimum. Lu et. al. [11] instead offers a distributed heuristic-based algorithm, which first individually places operators to nodes, and then later attempts to modify them

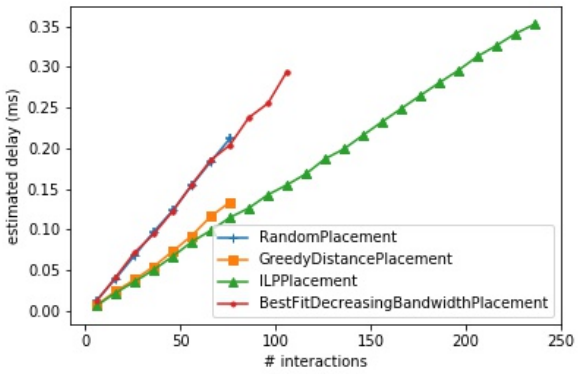

Figure 6: Estimated total delay TOPOLOGY2.

as needed. On the other hand, Srivastava et. al. [15] adds another assumption: that all operators are filters, which they propose a greedy and (separate) polytime algorithm. Ying et. al. [17] proposed a distributed algorithm inspired by the Bellman-Ford algorithm More recently, Tzaritas et. al. [16] presented an algorithm similar to [2], and focused on operator migration.

Another assumption often made in the context of operator placement is the node resource bottleneck. In [7], Chatzimilioudis et. al. proposed a hybrid algorithm utilizing both dynamic programming and heuristics. Hong et. al. [10] proposed a greedy algorithm based on scarcest resource, shortest path, and early feature extraction.

However, not all algorithms avoid bandwidth constraints. Pietzuch et. al. [12] alternatively proposed an algorithm that utilized an n-dimensional cost space based on data rates and network latency.

We acknowledge that the work published by Cardellini et. al. [6] is closest to our own. In their approach, they formulate the operator placement problem with regards to both nodes and operators graphs, along with providing constraints for both node resources as well as link resources. We note that one of the key differences between our work and their work is that each interaction produces data on their own based on a message size and frequency of sending.

It is evident that many papers present solutions for the operator placement problem under strong assumptions that may not hold true for more complex systems. From operator trees to node resource bottlenecks, we differentiate our work by focusing on links and bandwidth over node resources, and also modeling interactions in detail using attributes like message size and frequency.

7 CONCLUSIONS

To support adaptive mediation in IoT systems, the heterogeneity gap must be bridged via a mediation based-architecture. Components called wrappers or mediators support such bridging. This paper describes the *mediator placement problem* in which a set of mediators must be placed on a set of nodes based on IoT device attributes and network semantics. We devise an adaptive placement of mediators through an integer linear programming algorithm.

In future work we intend to incorporate the effect of queuing to our proposed solution, which is crucial in constrained networks. We will use queueing theoretic models [4]. as input to our placement techniques to more accurately quantify the performance of IoT interactions; such interactions may need to handle intermittently connected things while satisfying timely delivery. We expect to represent interactions with directed acyclic graphs (DAG), which more accurately model a more complex setting of sensors / virtual sensors. Ultimately, we plan to extend the mediator placement problem based on information semantics produced by IoT devices.

ACKNOWLEDGEMENTS

This work was supported by: NIST Award # 60NANB18D241 and the DARPA agreement # FA8750-16-2-0021.

REFERENCES

[1] A. Bennaceur and V. Issarny. 2015. Automated synthesis of mediators to support component interoperability. *IEEE Transactions on Software Engineering* 41, 3 (2015), 221–240.
[2] B.J. Bonfils and P. Bonnet. 2004. Adaptive and decentralized operator placement for in-network query processing. *Telecommunication Systems* (2004), 389–409.
[3] G. Bouloukakis, N. Georgantas, P. Ntumba, and V. Issarny. 2019. Automated Synthesis of Mediators for Middleware-layer Protocol Interoperability in the IoT. *Future Generation Computer Systems* 101 (2019), 1271 – 1294.
[4] G. Bouloukakis, A. Kattepur, N. Georgantas, and V. Issarny. 2018. Queueing Network Modeling Patterns for Reliable and Unreliable Publish/Subscribe Protocols. In *ACM MobiQuitous, New York, USA*.
[5] A. Bröring, S. Schmid, C. Schindhelm, A. Khelil, S. Käbisch, D. Kramer, D. Le Phuoc, J. Mitic, D. Anicic, and E. Teniente. 2017. Enabling IoT ecosystems through platform interoperability. *IEEE software* (2017), 54–61.
[6] V. Cardellini, V. Grassi, F. Lo Presti, and M. Nardelli. 2016. Optimal operator placement for distributed stream processing applications. In *ACM DEBS*. 69–80.
[7] G. Chatzimilioudis, Huseyin Hakkoymaz, Nikos Mamoulis, and Dimitrios Gunopulos. 2009. Operator placement for snapshot multi-predicate queries in wireless sensor networks. In *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*. IEEE, 21–30.
[8] M. Collina, G.E. Corazza, and A. Vanelli-Coralli. 2012. Introducing the QEST broker: Scaling the IoT by bridging MQTT and REST. In *IEEE PIMRC*. 36–41.
[9] H. Derhamy, J. Eliasson, and J. Delsing. 2017. IoT interoperability On demand and low latency transparent multiprotocol translator. *IEEE Internet of Things Journal* (2017), 1754–1763.
[10] H. Hong, P. Tsai, M.Y.S. Cheng, A.and Uddin, N. Venkatasubramanian, and C. Hsu. 2017. Supporting internet-of-things analytics in a fog computing platform. In *IEEE CloudCom*. 138–145.
[11] Z. Lu, Y. Wen, R. Fan, S. Tan, and J. Biswas. 2013. Toward efficient distributed algorithms for in-network binary operator tree placement in wireless sensor networks. *IEEE Journal on Selected Areas in Communications* (2013), 743–755.
[12] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. 2006. Network-aware operator placement for stream-processing systems. In *IEEE ICDE*. 49–49.
[13] F.M. Roth, C. Becker, G. Vega, and P. Lalanda. 2018. XWAREâĂŤA customizable interoperability framework for pervasive computing systems. *Pervasive and Mobile Computing* (2018), 13–30.
[14] S. Soursos, I.P. Žarko, P. Zwickl, I. Gojmerac, G. Bianchi, and G. Carrozzo. 2016. Towards the cross-domain interoperability of IoT platforms. In *IEEE EuCNC*. 398–402.
[15] U. Srivastava, K. Munagala, and J. Widom. 2005. Operator placement for in-network stream query processing. In *ACM SIGMOD-SIGACT-SIGART*. 250–258.
[16] N. Tziritas, T. Loukopoulos, S. Khan, and C. Xu. 2015. Distributed algorithms for the operator placement problem. *IEEE Transactions on Computational Social Systems* (2015), 182–196.
[17] L. Ying, Z. Liu, D. Towsley, and C. Xia. 2008. Distributed operator placement and data caching in large-scale sensor networks. In *IEEE INFOCOM*. 977–985.
[18] R. Yus, G. Bouloukakis, S. Mehrotra, and N. Venkatasubramanian. 2019. Abstracting Interactions with IoT Devices Towards a Semantic Vision of Smart Spaces. In *BuildSys '19, New York, USA*.