Tao R, Poslad S. Delay sensitive distributed sensor data exchange for an IoT. Proc. Int. Workshop on Adaptive Security at UbiComp '13, tThe 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing, Zurich, Switzerland — September 08 - 12, 2013, Article No. 7, doi: 10.1145/2523501

Delay Sensitive Distributed Sensor Data Exchange for an IoT

Ran Tao
Queen Mary, University of London
London, United Kingdom
ran.tao@eecs.qmul.ac.uk

Stefan Poslad
Queen Mary, University of London
London, United Kingdom
s.poslad@qmul.ac.uk

ABSTRACT
Publish Subscribe Message Oriented Middleware or PSMOM is widely deployed in the IoT. This uses message brokers to enable open and distributed sensor data publishers (or pub) and sensor data subscribers (or sub) to exchange information asynchronously using loosely-coupled pub-sub mechanisms. Existing PSMOM messaging systems ignore the different delay requirements of different types of sensor data and thus may introduce unexpected delays to time critical sensor data, which results in a delay for critical decision making. To address this challenge, a delay sensitive aware workload management framework for PSMOM is proposed. The framework extends an existing state-of-the-art messaging system, PEER, by adding support for workload allocation, a Queue Depth load metric, and dynamic load thresholds, enabling end-to-end latency guarantees and avoiding unnecessary load balancing. The validation using a simulation is also presented in the paper.

Categories and Subject Descriptors
K.6.5 [**Management of Computing and Information Systems**]: Security and Protection – *Authentication, unauthorized Access.*

General Terms
Reliability, Security, Management

Keywords
Message-oriented-Middleware, Sensor data, Message Delay Sensitivity

1. INTRODUCTION
Internet of Things (IoT) is 'a world-wide network of interconnected objects uniquely addressable based on standard communication protocols' [1]. More and more environment sensor things are acting as data sources are coming online to publish data. These sensors include a range of fixed infrastructure (from buildings to city-wide regions) sensors, physical natural environment sensors and mobile sensors (including vehicles, people as sensors. Much research has focussed on the issue of supporting resilient communication for very short range sensors that are organised in various types of ad hoc wireless sensor networks and need to communicate over multi-sensor hops to reach a sensor access node in order for data to be subsequently accessible over a wide-area [3]. In contrast, many of the above sensors coming online in practice tend to be designed to communicate over wider area links to sensor access nodes and where resilient multi-hop sensor data exchange is not a requirement or limitation. The main requirement is to support highly distributed, real-time data exchange between sensor data publishers and data subscribers over both local and wide areas. The key security challenges are to support, open, heterogeneous, scalable and resilient distributed sensor data exchange in the face of faulty or malicious data publishers and subscribers and where the exchange itself may become the bottleneck. Multi-sensor data processing and fusion is a further challenge but is considered out of scope in this research.

In order to meet the requirements of data exchange in IoT, Publish Subscribe Message Oriented Middleware (PSMOM), which allows multiple clients to asynchronously exchange messages via a broker or set of brokers using a loosely-coupled pub-sub mechanism [2], is widely deployed in IoT. Much work has been done to improve the resilience and scalability of a PSMOM messaging system but this is not generally applicable to the IoT for the following reasons. First, it focuses on homogeneous message exchange or message broker models where brokers are assumed to have the same processing power and bandwidth. However, IoT tends to be heterogeneous because different system components and subsystems have varying CPU, memory, disk size and Network Bandwidth. Second, the heterogeneity of the message transmission constraints is not fully considered. Although messages have been divided into different subjects and assigned with different sizes and rates, different Quality of Service (QoS) requirements for different types of messages are ignored. This may introduce unexpected delays to time-critical data and result in a delay for decision-making.

In this paper, we propose a delay sensitive load management framework for PSMOM. This model extends the Publish/Subscribe Efficient Event Routing (PEER) framework [2] by adding a workload distribution mechanism, PEER-WD, that assigns message brokers with least utilized load capacities to clients, a Queue Depth load metric, and dynamic thresholds, to provide latency guarantees and to avoid unnecessary load balancing. The remainder of the paper is organized as follows.

UbiComp'13 Adjunct, September 8–12, 2013, Zurich, Switzerland.

We begin by discussing previous and related research, followed by an overview of the system. Next, a detailed description of the workload management framework is presented with a validation and comparison of the PEER-WD extensions to PEER. We conclude with future research directions.

2. RELATED WORK

Much research work has been done on data aggregation [4] and data dissemination [5] protocols for wireless sensor networks. A cluster-based data aggregation protocol LEACH is proposed in [6] and in [7], a linear-chain scheme is introduced to collect data in sensor networks. In [8], the author studies the transport capacity of many-to-one communication in data aggregation using a hierarchical architecture. In [5], SAFE attempts to save energy through data dissemination path sharing among multiple data sinks. In addition, LAF, a modified flooding [9], is used to disseminate data in wireless sensor networks.

Efficient transport protocols to provide reliable data delivery in sensor networks are also proposed. In [10], RMST, a transport protocol that provides guaranteed delivery is introduced. RMST is a selective NACK based protocol that can be configured for in-network caching and repair. Nodes along the route of a message cache message fragments. Some nodes perform message re-assembly, test for fragments lost, and issue repair-requests to the previous nodes in the path. In [11], an approach called Pump Slow Fetch Quickly (PSFQ) is proposed, in which each node performs a special type of message reassembly. Thus nodes can immediately forward to the next hop fragments if they are received in order. In case if they receive an out-of-order fragment, they issue a repair request, and buffer the out-of-order fragment until the missing fragment is obtained and relayed. The repair requests are answered by previous nodes in the path that use fragment caches. In [12], the author proposes a store-and-forward hop-to-hop efficient data transfer protocol, used for high-bandwidth data collection in the structural monitoring of the Golden Gate Bridge. The models above focus on the behaviour design of sensor nodes but not designed for PSMOM. In addition, all of the design above ignore the different delay requirements of different types of sensor data and thus may introduce unexpected delays to delay sensitive sensor data.

With a PSMOM system, to provide guaranteed end-to-end transmission delay of sensor data exchange, a delay aware workload management is required. In a PSMOM system, the broker workload depends upon the number and type of subscriptions served by this broker, i.e., on message size and the incoming and outgoing messages rate. Load management in a PSMOM is achieved by migrating subscriptions from overloaded brokers to ones with lesser loads.

Gupta et al [13] proposed two types of load balancing in a peer-to-peer content-based PS system [14, 15]. Load balancing is achieved by splitting the peer with the heaviest subscription load in half and propagating events to a newly joint replicated peer. Chen and Schwan [16] proposed an optimized overlay reconstruction algorithm that performs load distribution based upon CPU load. Load Balancing is triggered only when clients find a broker that is closer than its current connected broker. Subscription clustering [17, 18, 19, 20] is another solution for load balancing that partitions a set of subscriptions into a number of clusters in order to reduce the overall network traffic. These solutions above can balance the load but they are all designed for homogeneous systems.

Cheung et al [2] proposed the PEER framework that aims to overcome the above limitations for load balancing in PSMOM. Its primary target is content-based routing PSMOM but the author claims that it can also be applied to topic-based (metadata based) routing PSMOM too. In PEER, brokers have different processing capabilities and Internet links. The load of a broker is detected by periodically monitoring three middleware layer load metrics: input utilization, matching delay, and output utilization, and comparing the monitoring results of each metric with two static thresholds. Among these metrics, input utilization is determined by the quotient of the input rate (R_{input}) in messages per second over the matching rate ($R_{matching}$) in messages per second, i.e., $R_{input}/R_{matching}$; matching delay is defined as the average time (in second) spent in a broker to process matching; output utilization is defined as the quotient of the used bandwidth (BW_{used}) over the total bandwidth (BW_{total}), i.e., BW_{used}/BW_{total}. If an unbalanced load or overload is detected, a load balancing is triggered and the system migrate subscriptions from the offloading broker onto a load accepting broker while not overloading it. An evaluation of the design compared to a naive random load balancing approach shows that PEER is capable of efficiently balancing load in a heterogeneous messaging environment. However, PEER ignores the heterogeneity of system applications, such as the different end-to-end latency requirements, and therefore may trigger unnecessary load balancing if all the applications are delay tolerant or introduce unexpected delays for delay sensitive applications. In addition, it does not distinguish the uplink that is used to disseminate messages to subscribers and downlink that is used to receive messages. The difference between these may introduce different client migration priorities in a load-balancing phase. Further, there is no pre-emptive workload distribution mechanism in PEER. It therefore requires an extra work to migrate subscriber clients from one (edge) broker to another based upon the load differences.

Our work extends the PEER framework by adding support for workload allocation and more comprehensive delay sensitive aware load detection; and redesigns the load analysis and balancing mechanisms to fit the detection and distribution mechanism.

3. PEER-WD FRAMEWORK OVERVIEW

In our work, message brokers are organized into a Head-Edge Broker model (H-E model) that is motivated by the architecture adopted by Google's distributed publish/subscribe system GooPS for use in MOM deployments in real world applications [2]. Our design, PEER-WD, targets enhancements to the Head-Edge Broker model by providing delay sensitive load management supported with management agents.

3.1 Head-Edge Broker Model

The Head-Edge broker model organizes the brokers into a hierarchy structure as shown in Figure 1.

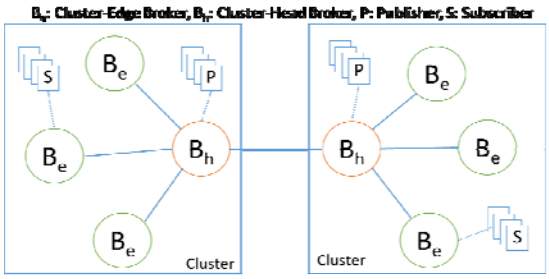

Figure 1. PEER Head-Edge Broker Model

A broker with more than one neighbour broker is referred to as a cluster-head broker (B_h), while a broker with only one neighbour broker is referred to as a cluster-edge broker (B_e). A cluster-head broker together with its connected edge brokers form a cluster. In the H-E model, publishers are only served by B_h, and subscribers are only served by $_{Be}$, so that in a cluster, messages are always routed from the B_h to B_e. Inter-cluster message dissemination is achieved by have a B_h forwarding publication messages to the B_h of all matching clusters.

3.2 Management Agent

To manage the workload for H-E model, a management agent (MA) is allocated for each broker. The MA belonging to the head broker is called HMA while the one belonging to the edge broker is named EMA. Both HMA and EMA consist of an Overlay Manager (HOM for the Head and EOM for the Edge respectively), a Load Detector (HLD and ELD), and a Load Analyser (HLA and ELA).

HOM receives the broker allocation request from all the clients in the cluster and assign brokers to the clients according to the client's source (a publisher or a subscriber), the availability of the broker, and the distribution status of existing clients. In addition, when load balancing is triggered, HOM notifies selected clients to migrate from original brokers to the new load-accepting brokers. What's more, HOM interacts with EMA to update the load information of edge brokers; and interacts with HOM of other clusters to share the cluster-based load information. **EOM** updates the load status of the edge broker to HOM and receives the load status of other edge brokers in the same cluster from HOM. In addition, when load balancing is triggered, EOM updates the available selected subscriptions to the HOM. Both HOM and EOM work with its relevant load analysers to generate an offloading client list that contains the clients to be migrated from the overloaded broker to the load-accepting broker when load balancing is required.

HLD and **ELD** detects the load status, e.g., as a set of fuzzy states, LOW, HIGH, and OVERLOAD, of the relevant broker, i.e., HLD monitors the head broker and ELD monitors edge brokers. Although the authors in [2] claim that the head broker is less likely to be overloaded since it does no matching work for subscribers, the head broker can become overloaded when it reaches its maximum network capacity whilst exchanging messages. So HLD monitors the network bandwidth used by the head broker and reports its status to HOM when its load state changes. ELD does similar work but it needs to monitor all the load metrics (see section IV.B) and report this to the EOM. In addition, to get the dynamic threshold, ELD periodically detects the transmission latency between the edge broker and head broker, between subscribers and edge brokers, and request HLD to detect the transmission latency between publishers and head brokers.

HLA and **ELA** analyse the load distribution for clients, e.g., the Internet usage of individual client, store the observations into a table and passes this to the relevant OM. In addition, the clients in the overloaded broker are prioritized for offloading when its load metric exceeds its threshold otherwise making the broker become overloaded.

4. LOAD MANAGEMENT FRAMEWORK

In this design, the workload management framework consists of a workload distribution phase, a load detection phase, and a load-balancing phase. In the workload distribution phase, HOM allocates brokers to each new subscriber based upon the load status of the edge brokers and the distribution of existing subscribers. In the load detection phase, the load of the broker is periodically detected and the change of the load status is updated and sent to its Overlay Manager. During the load balancing phase, a three step offloading strategy is adopted, i.e., locating the load accepting broker(s), selecting subscriptions, and migrating the selected subscriptions from the overloaded broker to the load accepting broker(s).

4.1 Workload Distribution

In practice, it is very important to avoid the OVERLOAD problem by optimizing the workload distribution beforehand. In this workload management framework, the workload distribution process is designed using the following principles:

- First, subscribers of the same topic are allocated to the same broker to avoid extra Network Bandwidth usage, as same messages are no longer routed to different edge brokers.
- Second, topics that are highly positively correlated are allocated to different brokers [5] as they may increase the load much more than non-correlated topics, which results in a sudden increase in broker load.
- Third, new subscriber clients are allocated to brokers that have the least utilized load capacity that is computed from all the load metrics.

4.2 Load Detection

To accurately detect the load status of a broker, the load metric and related thresholds need to be clarified. In the H-E broker model, brokers are classified into a cluster-head broker and cluster-edge broker, and different load metrics and thresholds are allocated to the different types of brokers.

4.2.1 Load Metrics for Cluster Head Broker

The main tasks of a B_h are: to route messages from publishers and B_h of other clusters to the B_e that serves matched subscribers; to route messages from publishers to the B_h of another clusters that serve matched subscribers. As claimed in [2], a B_h is less likely to be overloaded for doing the matching work as no subscribers connect to it. Therefore, the load status of a B_h is mainly affected by the network bandwidth usage. Table 1 lists the load metrics used for cluster-head broker.

Table 1. Load Metrics for Head Broker

Metric	Expression
Downlink Bandwidth Utilization	Input-Rate / Downlink-Bandwidth
Uplink Bandwidth	Output-Rate / Uplink-

Utilization	Bandwidth

4.2.2 Load Metrics for Cluster Edge Broker

As opposed to B_h, B_e serves all subscribers; it therefore does much more matching work. So the load costs for matching needs to be monitored. In addition, since a guaranteed end-to-end transmission delay is required, a Queue Depth metric that measures the number of messages waiting in the output queue and reflects the message waiting time in a broker is introduced. Table 2 lists the load metrics used for cluster edge broker.

Table 2. Load Metrics for Edge Broker

Metric	Expression
Downlink Bandwidth Utilization	Input-Rate / Downlink-Bandwidth
Matching Utilization	Input-Rate / Matching-Rate
Uplink Bandwidth Utilization	Output-Rate / Uplink-Bandwidth
Queue Depth	No. of Messages waiting in each Output Queue

4.2.3 Threshold Determination

Two thresholds are introduced for each metric to describe the load status of a broker. A lower threshold (TH_{low}) indicates whether or not a broker is available to accept more loads; while a higher threshold (TH_{high}) indicates whether or not any load shifting is required. Based upon the two load thresholds, the load status of a broker is divided into LOW LOAD, HIGH LOAD, and OVERLOAD. The relationship between the threshold and the load state is defined in.

Table 3. Load State & Threshold

Condition	State
(All the metrics) $< TH_{low}$	LOW LOAD
$TH_{low} <$ *(Any metric) & (All the metrics)* $< TH_{high}$	HIGH LOAD
$TH_{high} <$ *(Any metric)*	OVERLOAD

A high value for the HIGH LOAD threshold (e.g., 99% CPU Utilization) makes fuller use of system resources. However, a broker can become overloaded before it can do any offloading. The magnitude of the difference between the lower and higher threshold controls the efficiency of load balancing and the level of the load imbalance between brokers. For example, a small difference, e.g., 1%, reduces the load imbalance between brokers but makes brokers more likely to enter the OVERLOAD state from the HIGH LOAD state which may result in endless load balancing cycles [2]. In addition, based upon whether or not the load metrics are affected by the delay sensitivity of the messages, the load metrics are divided into two groups and assigned with different thresholds.

Both uplink usage and downlink usage for B_h are set with static thresholds, i.e., TH_{low} = 0.9 and TH_{high} = 0.95. The same thresholds are applied to the downlink bandwidth utilization and the matching utilization for the edge broker. These values are retrieved from the threshold defined for PEER [2]. The uplink usage and the Queue Depth metric of a B_e is considered separately as it affects the time of messages waiting in the broker. In this design, only TH_{low} is assigned to the uplink bandwidth utilization metric of the edge broker as it is only used to indicate whether or not the broker is available for more loads; and only TH_{high} is set for the Queue Depth metric that is used to trigger load balancing with latency guarantees. The value of TH_{low} for the uplink bandwidth utilization of the edge broker is set the same as others, e.g., 0.9; while the value of TH_{high} for Queue Depth of edge broker is calculated based upon the end-to-end latency requirements for different topics of individual subscribers, the transmission delays, and the time a message spends in broker. The following procedure shows the steps of determining the dynamic TH_{high} for Queue Depth metric.

4.2.4 Transmission Time Detection

The end-to-end latency requirement for subscriber "s" on topic "T" is denoted as $t_{s,T}$. The practical end-to-end latency is calculated as the sum of the total transmission time (t_{s,T_trans}), the total time spent in broker (t_{s,T_broker}). With the H-E model, the total transmission time is obtained based upon the transmission time from publishers to B_h ($t_{s,T\_p\text{-}h}$), from B_h to B_h of matching clusters ($t_{s,T\_h\text{-}h}$), from B_h to B_e of the matched subscribers ($t_{s,T\_h\text{-}e}$), and from B_e to subscribers ($t_{s,T\_h\text{-}s}$), i.e., $t_{s,T\_trans} = t_{s,T\_p\text{-}h} + t_{s,T\_h\text{-}h} + t_{s,T\_h\text{-}e} + t_{s,T\_e\text{-}s}$. For the case that publisher clients on the same topic are served by different clusters, the transmission time obtained for different publishers may have different values since the time cost from publishers to B_h and from B_h to B_h may be different. In our design, the maximum value from all the obtained transmission time is selected, denoted as $t_{s,T\_trans\text{-}sel}$.

4.2.5 Time in Broker Detection

The total time spent in broker (t_{s,T_broker}) consists of the time spent in B_h that serves the publisher (t_{s,T_h}), the time spent in the remote B_h belonging to the matched clusters ($t_{s,T\_remote\text{-}h}$), the B_e that serves the matched subscribers ($t_{s,T\text{-}e}$), i.e., $t_{s,T\_broker} = t_{s,T\_h} + t_{s,T\_remote\text{-}h} + t_{s,T\_e}$. For each broker, the time cost is the sum of the arrival time ($t_{s,T\_arrival}$), departure time ($t_{s,T\_departure}$), the matching time ($t_{s,T\_matching}$) and the time waiting in the queue ($t_{s,T\_waiting}$). Both the arrival and departure time is determined by the size of the message and the uplink/downlink bandwidth whilst the matching time is mainly affected by the number of filters in the matching process. The waiting time in a broker is determined by the number of messages waiting in the queue and the message output rate.

4.2.6 Dynamic Threshold Calculation

With the end-to-end transmission delay, the maximum time that a message can spend in the output queue of broker B_e ($t_{s,T\text{-}e}$) can be determined as $t_{s,T} - t_{s,T\_trans\text{-}sel} - t_{s,T\_h} - t_{s,T\_remote\text{-}h} - t_{s,T\_arrival\text{-}e} - t_{s,T\_matching\text{-}e} - t_{s,T\_departure\text{-}e}$. The value may be changed in different time due the change of transmission time, matching time and arrival/departure time. This maximum waiting time in the message broker is used to compute the higher threshold for Queue Depth metric for subscriber "s" on topic "T", i.e., the value of Queue Depth at a time t_i ($QD_{s,T}(t_i)$) must follow the condition defined in (1), where $\lambda_{s,T}(t_{i+1})$ and $\mu_{s,T}(t_{i+1})$ are the predicted message input rate and output rate in message/s for time t_{i+1}, and t_{LB} is the average time cost for load balancing that is mainly affected by the notification message transmission time from HOM to subscribers, e.g., from milliseconds to seconds, and the analysis time, e.g., in milliseconds.

$$\frac{QD_{s,T}(t_i) + [\lambda_{s,T}(t_{i+1}) - \mu_{s,T}(t_{i+1})]}{\mu_{s,T}(t_{i+1})} \le t_{s,T-e} - t_{LB} \quad (1)$$

Therefore the higher threshold for Queue Depth metric at time t_i ($TH_{s,T}(t_i)$) is found with (2).

$$TH_{s,T}(t_i) = \left(t_{s,T-e} - t_{LB}\right) * \mu_{s,T}(t_{i+1}) - [\lambda_{s,T}(t_{i+1}) - \mu_{s,T}(t_{i+1})] \quad (2)$$

4.3 Load Analysis

Load analysis is invoked when a broker is in the OVERLOAD state. A load analyser estimates and profiles the load distribution for individual clients served by the broker. In addition, it prioritizes the offloading clients according to their overloaded load metrics and stored this into the prioritised client list.

4.3.1 Load Estimation

Both ELA and HLA computes the network bandwidth usage for individual clients based upon the message exchange rate and the bandwidth, e.g., the uplink usage of edge broker for subscriber "s" on topic "T" is computed as the message output rate ($\mu_{s,T}$) / uplink bandwidth. In addition, ELA estimates the matching utilization and records the Queue Depth for each subscriber on each topic.

4.3.2 Priorities Offloading Client

In our design, the clients of the same topic are recognized as a bundle in the offloading process, i.e., they are either migrated together to the load-accepting broker or kept together in the overloaded broker. Only if the load-accepting broker cannot accept any bundle of clients, these clients are dealt with separately.

In the head broker, the publishers of different topics can be categorized into 4 groups: the publishers that only have remote subscribers, i.e., subscribers in neighbour domain (P_r), the publishers that have both local and remote subscribers ($P_{r\text{-}l}$), the publishers that only have local subscribers (Pl), and the publishers that have no subscribers (P_n). So if the broker is in a downlink bandwidth overload state, the priority of all the publishers are $P_n > P_r > P_{r\text{-}l} > P_l$; while if it is an uplink overload state, the priority relationship becomes $P_r > P_{r\text{-}l} > P_l > P_n$. The difference between the two is the location of P_n, because migrating publishers with no subscribers cannot reduce the uplink bandwidth utilization but only reduce the downlink bandwidth utilization.

In each edge broker, similar to the equivalent situation with head brokers, the subscribers on different topics can be categorized into S_r, $S_{r\text{-}l}$, S_l and S_n. In addition, for the Queue Depth metric, as it does not relate to where the publishers locate, the subscribers are categorized into three groups: subscribers without messages waiting in the queue (S_{empty}), subscribers with messages waiting in the queue but not overloaded ($S_{w\text{-}no}$), and subscribers for which the QD metric is overloaded ($S_{overload}$). In all the groups above, the subscribers are ordered based upon its allowed waiting time, i.e., the larger the waiting time, the higher the priority. The relationship between subscribers is defined in Table 4.

Table 4. Priorities Subscribers in Edge Broker

Overload Metric	Priority
Downlink Bandwidth Utilization	$S_r > S_{r\text{-}l} > S_l > S_n$.
Uplink Bandwidth Utilization	
Matching Utilization	$S_n > S_r = S_{r\text{-}l} = S_l$
Queue Depth	$S_{empty} > S_{w\text{-}no} > S_{overload}$

4.4 Load Balancing

After the load analysis process, load balancing takes place. As described in PEER, if a head broker becomes overloaded, load balancing happens between head brokers in different clusters by migrating publishers from an overloaded head broker to head brokers with lesser loads. If instead, the edge broker becomes overloaded, the load balancing takes place within a local cluster first and only if there is no available local load accepting broker, i.e., no broker is in the LOW LOAD state, or the available load accepting brokers have less actual load capacity than that required by the overloaded broker to recover from OVERLOAD state, is inter-domain load balancing invoked. All the load balancing processes follow a similar three-step offloading strategy, i.e., Load Accepting Broker Locating, Client Selection, and Client Migration. In this paper, intra-domain load balancing between edge brokers is described below as an example.

4.4.1 Load Accepting Broker Locating

The EOM of an overloaded broker checks the load state of brokers in the same domain to locate brokers in a LOW load state and sends a load balancing request to a HOM with the candidate broker ID(s). The HOM records whenever a broker is in a load-balancing phase and sends requests to all candidate brokers. The EOMs of these candidate brokers reports the value of the all the load metrics to the HOM. And when HOM receives this information, it will in turn forward to the requesting EOM.

4.4.2 Client Selection

After the load accepting broker is located, EOM of the overloaded broker prioritises the candidate brokers based upon the value of the load metric that overloads the broker, i.e., the broker with the lowest load metric value has the highest priority to accept the load. In addition, EOM retrieves the clients from the prioritised client list (see section 4.3.2). It estimates the load influence on the load accepting broker based upon on all the load metrics. For example, for the uplink BW usage, the influence is estimated as the (input rate of the client / the uplink BW of the load accepting broker). This means that if clients are migrated to the load-accepting broker, the uplink usage will be increased by this amount. So for the case that the client does not overload the load-accepting broker, the client is selected and put in an offloading list. The selection process continues until the estimated load status of the overload broker is not OVERLOAD any more. The offloading list is then sent to the HOM. HOM notifies the EOMs of the selected edge brokers to be in a load-balancing phase.

4.4.3 Client Migration

In last step, HOM sends messages to all the clients that are in the offloading list, asking them to start a message exchange via the load-accepting broker(s). All the clients then set up connection(s) to the load accepting broker(s) and drop the connection to the offloading broker except for subscribers that have messages waiting in the queue. In this case, the subscribers will drop the connection only when all the messages waiting in the overloaded broker are received. In addition, a message is sent by each client to HOM to confirm the completion of the migration process. HOM counts the number of clients that have completed the migration away from the overloaded broker. There is also a default timeout for the migration so that the load-balancing phase can stop even if some clients fail, and fail to restart their message exchange, during the migration. When all the clients complete the migration or the waiting time has timed out, the HOM notifies all

the EOMs involved in the load-balancing phase that the load balancing is complete.

5. VALIDATION

The PEER-WD framework is validated by comparing its load management scheme to that designed for the PEER framework. In this paper, local load balancing, caused by the full utilization of the uplink bandwidth, is used as an example scenario to evaluate. In this experiment, four edge brokers (B_0, B_1, B_2, and B_3) are connected to one cluster-head broker (B_h). They form a star topology, which forms a messaging bus to exchange information for sensor data. The deployment is illustrated in Figure 2.

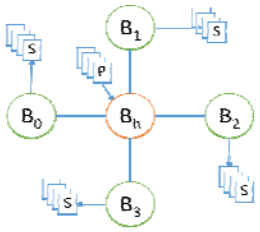

Figure 2. Broker Deployment in Experiment

The simulation environment specification is listed in Table 5. For each broker, the uplink bandwidth and downlink bandwidth is fixed at the same value during the experiment. Hence, the transmission latency between brokers does not change, e.g., is set to 0.1s. In addition, we assume that the client to broker transmission latency is also static during the experiment, e.g., 0.2s.

In the experiment, we assume that there are 15 different types of sensor data are exchanged through the system, denoted as 15 different message topics. For each topic, there are multiple publisher sensor nodes (PSN) that publish sensor data to the brokers and multiple subscriber sensor nodes (SSN) that subscribe to the matched sensor data via the brokers.

Table 5. Simulation Experiment Specification

Broker ID	Specifications		
	CPU (MHz)	***Memory (MB)***	***Bandwidth (Mbps)***
B_h	2000	64	20
B_0	800	32	6.5
B_1	1500	32	8
B_2	1300	64	5
B_3	1000	64	8

The number of publisher sensor nodes and subscriber sensor nodes are randomly generated. The reason to use a random number is to allow the broker loads to be varied in different experiments to improve the validation. In addition, we assume that subscribers of different topics have different end-to-end latency requirements but the subscribers of the same topic have the same latency requirement. The average message size depends on the topic, e.g., varies from 200 Byte to 1KB. Table 6 gives an example of how topics are specified in one experiment.

Table 6. Topic Specifications in one Experiment

Topic ID	No. of PSN	No. of SSN	Latency Req. (s)	Msg Size (Byte)
1	1	8	1.8	200
2	2	2	1.7	800
3	5	1	1.6	1000
4	4	2	1.5	400
5	3	3	1.4	200
6	1	1	1.3	400
7	2	5	1.2	300
8	2	7	1.1	400
9	5	2	1.0	500
10	4	4	0.9	200
11	1	5	30	600
12	3	2	60	400
13	1	6	40	200
14	2	3	50	300
15	4	5	100	200

Based upon the latency requirements specified in Table 6, we then compute the maximum time that a message can be held in a broker, e.g., for topic 2, $t_{topic2\text{-}broker} = 1.7 - 0.2 - 0.1 = 1.4s$. These maximum waiting times in the broker are used in an experiment to determine the high threshold for the Queue Depth metric.

At the beginning of each experiment, all brokers are started simultaneously with the MAs. When all the brokers are started, all PSNs register and connect to the cluster head broker (B_h), and MAs start to measure the load status of a broker and the broker-to-broker transmission delays. After that, the experiment is divided into three phases: client distribution, equilibrium and message burst simulation combined with load balancing. In the client distribution phase, 1s – 15s, SSNs of the each topic are registered and assigned with available edge brokers at second intervals. In the equilibrium phase, 15s - 29s, both PSNs and SSNs are running. There are no message bursts and no clients join or leave. In the message burst simulation and load balancing phase, at 30s, a burst that simulates a sensor data flood is generated by doubling the sampling rate (publishing rate) of 7 sensor types (topics), e.g., topic 2, 4, 6,…, 12, 14; from 31s to the end of the experiment at 75S, load balancing will be triggered if any load metric exceeds its higher threshold.

The reason to set time slots to these values is to make the simulation results highlight the changes in each stage. The experiment can be easily expanded by 1) adding more brokers, PSNs and SSNs; 2) increasing the time interval for each phase; 3) generating more sensor data bursts. The duration of these phases is also configurable.

In the example experiment with the specifications in Table 6, load balancing is triggered when the uplink bandwidth is fully utilized and the queue depth reaches its higher threshold. Figure 3 shows the results for the uplink bandwidth utilization % (y) against time in second (x). When the value of the uplink bandwidth utilization exceeds 100%, there are messages waiting in the output queue of the broker, e.g., for broker b1, after the burst at 30s, the output queue is built up.

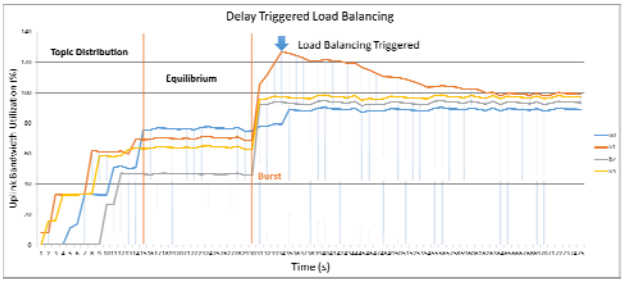

Figure 3. Simulation Result for Uplink Bandwidth Utilization

After the workload distribution, SSNs of different topics have setup connection to one of the edge brokers. Table 7 shows the topic distribution status. The topic ID of each broker can be read from Figure 3 according to the inflection points of each broker.

Table 7. Topic Distribution

Broker ID	Topic ID List
B_0	5, 6, 7, 11, 15
B_1	1, 3, 8, 14
B_2	10, 12
B_3	2, 4, 9, 13

After the burst, e.g., at 30s, the publishing rate of topic 2, 4, 6,…, 12, 14 are doubled, for broker b1 , the output queue starts to build up as the uplink bandwidth utilization exceeds 100%; 4s after the burst (34s), the queue depth value of topic 8 exceeds the TH_{high}, and thus load balancing is triggered. Figure 4 shows the Queue Depth, i.e., number of messages in the output queue, for topic 8 in in broker b1.

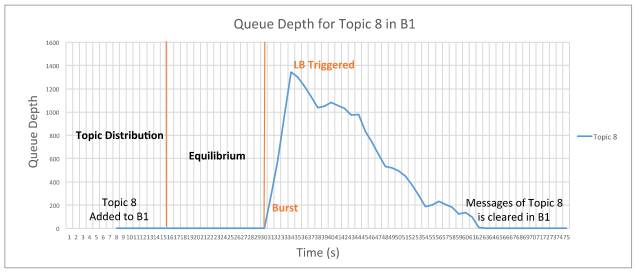

Figure 4. Queue Depth for Topic 8 in broker b1

Topic 1 in b1 is then selected and migrated to broker b0. Therefore broker b1 has more bandwidth to clear the messages for topic 8 in the queue (from 34s – 62s, a balancing stage). After 62s, the messages queue for topic 8 in broker b1 is removed. The uplink bandwidth utilizations for all the brokers are below 100%.

When the same simulation is applied using PEER load balancing mechanism, the results are shown in Figure 5.

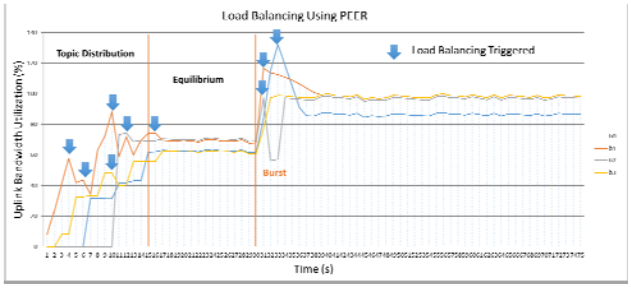

Figure 5. Simulation Result for Uplink Bandwidth Utilization using PEER

In the distribution phase (1s-15s) of PEER, all the SSNs are initially connected to broker b1 and migrated to other brokers (e.g., b2) based upon the load differences (as there is no work distribution mechanism in PEER). In addition, after a burst (30s), as the delay requirements for topics are ignored, unnecessary load balancing takes place between broker b0 and b2 (at time 31s) that results in an additional load balancing to balance the two at time 33s. Comparing Figure 3 to Figure 5, the differences indicate that our proposed delay-aware load balancing method leads to more effective workload distribution, and can avoid unnecessary load balancing as the delay requirements are considered.

6. CONCLUSION AND FUTURE WORK

In this paper, we propose a delay sensitive aware PSMOM solution, PEER-WD, to support open and distributed sensor data exchange. In order to provide resilient sensor data exchange in the face of QoS restrictions, a delay sensitive workload management framework, PEER-WD that extends a state-of-the-art PEER framework [2] is presented. In addition, an intra-cluster load management example is presented and compared to PEER. The results show that the proposed framework is aware of the delay requirements, and has the potential to efficiently support sensor data exchange in the face of QoS restrictions.

The framework is implemented with Apache Qpid, an open source AMQP based MOM product. In the future, real sensor data will be used to evaluate the framework in a WAN based setting.

7. ACKNOWLEDGMENTS

This work is supported in part by, the EU FP7 funded project TRIDEC (FP7-258723-TRIDEC), by a PhD studentship at Queen Mary University of London.

8. REFERENCES

[1] INFSO D.4 Networked Enterprise & RFID INFSO G.2 Micro & Nanosystems in Co-operation with the Working Group RFID of the ETP EPOSS. 2008. Internet of Things in 2020, Roadmap for the Future, Version 1.1.

[2] Cheung, A. K. Y., and Jacobsen, H-A. 2010. Load Balancing Content-based Publish/Subscribe Systems. ACM Transactions on Computer Systems (ACM TOCS), 28, 4, Article 9.

[3] Cheung, C.T., Tse, C.K., and Lau, F.C.M. 2011. A Delay-Aware Data Collection Network Structure for Wireless Sensor Networks, IEEE Sensors Journal, 11, 3, 699 – 710.

[4] Rajagopalan, R., and Varshney, P. K. 2006. Data-aggregation techniques in sensor networks: a survey, IEEE Communication Survey & Tutorials, 8, 4, 48 – 63.

[5] Kim, S., Son, S.H., Stankovic, J.A., Li, S., Choi, Y. 2003. SAFE: A Data Dissemination Protocol for Periodic Updates in Sensor Networks. In 23rd International Conference on Distributed Computing Systems Workshops (ICDCSW), 228 – 234.

[6] Heinzelman, W.R., Chandrakasan, A., and Balakrishnan, H. 2000. Energy-Efficient Communication Protocol for Wireless Microsensor Networks. In 33rd Hawaii International Conference on System Sciences (HICSS), 8, 8020 – 8020.

[7] Lindsey, S., and Raghavendra, C.S. 2002. PEGASIS: Power Efficient Gathering in Sensor Information Systems. In: Proceedings of the IEEE Aerospace Conference, 1125 – 1130.

[8] Kahn, J.M., Katz, R.H., and Pister, K.S.J. 1999. Next century challenges: mobile networking for “Smart Dust. In: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking, 271 – 278.

[9] Sabbineni, H., and Chakrabarty, K. 2005. Location-Aided Flooding: An Energy-Efficient Data Dissemination Protocol for Wireless Sensor Networks. IEEE Transactions on Computers, 54, 1, 36 – 46.

[10] Stann, F., and Heidemann, J. 2003. RMST: Reliable Data Transport in Sensor Networks. 1st IEEE International Workshop on Sensor Net Protocols and Applications (SNPA), 102 – 112.

[11] Wan, C.Y., Campbell, A.T., and Krishnamurthy L. 2002. PSFQ: A Reliable Transport Protocol for Wireless Sensor Networks. First Workshop on Sensor Networks and Applications (WSNA), 1 – 11.

[12] Kim, S. 2003. Structure Monitoring using Wireless Sensor Networks. CS294-1 Deeply Embedded Network Systems class project.

[13] Gupta, A., Sahin, O. D., Agrawal, D., and Abbadi, A. E. 2004. Meghdoot: Content-based Publish/Subscribe over P2P Network. 5th ACM/IFIP/USENIX International Conference on Middleware (Middleware), 254 – 273.

[14] Aekaterinidis, I., and Triantafillou, P. 2006. PastryStrings: A Comprehensive Content-Based Publish/Subscribe DHT Network. 26th IEEE International Conference on Distributed Computing Systems (ICDCS), 23 – 23.

[15] Zhu, Y. 2007. Ferry: A P2P-Based Architecture for Content-Based Publish/Subscribe Services. IEEE Transactions on Parallel and Distributed Systems (TPDS), 18, 5, 672 – 685.

[16] Chen, Y., and Schwan, K. 2005. Opportunistic Overlays: Efficient Content Delivery in Mobile Ad Hoc Networks. 6th ACM/IFIP/USENIX International Conference on Middleware (Middleware), 354 – 374.

[17] Casalicchio, E., and Morabito, F. 2007. Distributed Subscriptions Clustering with Limited Knowledge Sharing for Content-Based Publish/Subscribe Systems. 6th IEEE International Symposium on Network Computing and Applications (NCA), 105 – 112.

[18] Riabov, A., Liu, Z., Wolf, J. L., Yu, P. S., and Zhang, L. 2002. Clustering Algorithms for Content-Based Publication-Subscription Systems. 22nd International Conference on Distributed Computing Systems (ICDCS), 133 – 142.

[19] Riabov, A., Liu, Z., Wolf, J. L., Yu, P. S., and Zhang, L. 2003. New Algorithms for Content-Based Publication-Subscription Systems. 23rd International Conference on Distributed Computing Systems (ICDCS), 678 – 686.

[20] Wong, T., Katz, R. H., and McCanne, S. 2000. An Evaluation of Preference Clustering in Large-scale Multicast Applications. 9th IEEE International Conference on Computer Communications (INFOCOM), 451 – 460.