Symbiot: Congestion-driven Multi-resource Fairness for Multi-User Sensor Networks

Yad Tahir*, Shusen Yang†, Usman Adeel*, Julie McCann*
*Imperial College London, †University of Liverpool
{yst11, u.adeel09, j.mccann}@imperial.ac.uk, shusen.yang@liverpool.ac.uk

Abstract—In this paper, we study the problem of multi-resource fairness in multi-user sensor networks with heterogeneous and time-varying resources. Particularly we focus on data gathering applications run on Wireless Sensor Networks (WSNs) or Internet of Things (IoT) in which users require to run a serious of sensing operations with various resource requirements. We consider both the resource demands of sensing tasks, and data forwarding tasks needed to establish multi-hop relay communications. By exploiting graph theory, queueing theory and the notion of dominant resource shares, we develop Symbiot, a light-weight, distributed algorithm that ensures multi-resource fairness between these users. With Symbiot, nodes can independently schedule its resources while maintaining network-level resource fairness through observing traffic congestion levels. Large-scale simulations based Contiki OS and Cooja network emulator show the effectiveness of Symbiot in adaptively utilizing available resources and reducing average completion times.

I. INTRODUCTION

Wireless Sensor Networks (WSNs) and Internet of Things (IoT) [1] are evolving towards interconnected, sensing and processing infrastructures that are expected to provide services for multiple concurrent users. One of the main reasons of having such shared enterprise deployments is to maximize the return on investment while minimizing operating costs.

Such multi-user networks are almost invariably heterogeneous in terms of their hardware components, offered resource capacities as well as user demands. Various types of sensors can be attached to nodes. The assorted capacities of some resource types, such as bandwidth, can be highly dynamic and time-varying [2], [3]. In addition, diversity is found when users demand specific resources. For instance, a user may require the execution of a memory-heavy task on nodes with temperature sensors, while another user may need to compute a bandwidth-hungry task on nodes having humidity and light sensors.

Resource sharing in any multi-user sensor network is a key issue for one primary reason. The rapid growth of traffic and computation requirements are often not matching the growth and expansion of overall network capacity. Hence, the limited network resources should be distributed fairly between users [4], [5]. Accounting for diversity across node resources and user requirements presents an increased challenge to schedulers for fair provisioning of resources. Getting this right is fundamental to next-generation WSN and IoT systems.

This paper concerns how the available network resources, including those that are time-varying, should be allocated between competing users. In particular, we consider the problem of fair resource allocation for data gathering applications in WSNs and IoT.

A. Motivation

Both resource and demand heterogeneity have an impact on the efficient and fair allocation of resources to users. Current max-min-based resource allocation schemes such as [6] do not deal well with both heterogeneous resources and user demands in multi-resource systems. Utilizing multi-resource fairness schemes [4], [5] such as Dominant Resource Fairness (DRF) [4] have become a hot topic in both computation economics communities and cloud computing. Given users with heterogeneous demands, these approaches achieve more efficient allocation than single-resource fairness schemes.

However, these approaches are limited when it comes to providing distributed scheduling that can adapt to time-varying resources. They assume that the system has a single, centralized scheduler that is aware about the resource capacities of all the system components. This means in a network with dynamic resources, nodes have to repeatedly report their available resource capacities to the scheduler. This is neither scalable nor resilient, thus impractical for large-scale networks. Furthermore, these approaches do not address multi-hop nature on which the network relies. When a user executes a task on a particular node, the output has to be forwarded to a gateway device using multi-hop communications. This creates indirect resource consumptions from the user. To ensure having fair multi-resource scheduling, it is crucial to consider both direct and indirect resource demands in multi-hop networks.
To understand the complexity of achieving multi-resource fairness in multi-user sensor networks, consider a typical sensing application shown in Fig. 1. The network has three users \( u_1 \), \( u_2 \), and \( u_3 \) with different sensing requirements expressed as SQL-like queries for convenience. Two cases need to be analyzed here: first, based on the sensing requirements, \( u_1 \) and \( u_2 \) do not compete on resources. Each of them requires a different set of nodes. However the employed multi-hop communications make \( u_1 \) and \( u_2 \) indirectly related to each other. They now compete for the shared resources of relay nodes (nodes A and B). The second case is that if \( u_1 \) changed its requirements and asked for all the resources available in all nodes with temperature or light sensors, surely this would be an unfair application shown in Fig. 1. The network has three users \( u_1 \) and \( u_2 \), but \( u_3 \) is certainly not. Second, given each user submit sensing tasks to the network, what is the maximum number of tasks that each user can execute while the resource consumptions between related users remain fair?

**B. Contributions**

We present the study of the fair resource allocation problem in network with multiple, time-varying resource types. We summarize the contributions of this paper as follows:

1. Section II establishes a set of system models to formally describe multi-user WSNs and IoT. By exploiting graph theory, we rigorously define the relationship between users in terms of their direct and indirect resource requirements.
2. Section III proposes a new extension to max-min fairness that exploits graph theory and dominant resource sharing and light sensors. Let a node \( n \) execute two types of tasks: sensing and data forwarding. Each user \( u \in U \) can run sensing tasks periodically on nodes. We assume that a sensing task requires no more than one time slot to finish. We also assume that the sensing tasks are delay-tolerant, which is realistic as many monitoring applications are reasonably delay tolerant.

Performing sensing tasks will generate data packets to be collected by a gateway \( h \in H \). To achieve this, nodes execute data forwarding tasks. The data can be transmitted directly if \( h \in N_u \), or indirectly in a multi-hop fashion if \( h \notin N_u \). The underlying routing layer determines the forwarding policy. In this paper, we only consider networks running in a fixed routing scenario, i.e., the route of each data flow is pre-determined and fixed during the time of interest. We also assume both sensing and forwarding tasks are indivisible.

In the network, each node maintains data packets through queues (data buffers). Let \( Q_n(t) \) denote the queue existing in node \( n \) to hold the data packets generated by user \( u \). Let \( Q_n^u(t) \) be the queue backlog (or queue length) of the queue \( Q_n(t) \) at time slot \( t \). The queue dynamics of \( Q_n^u(t) \) are defined as follows:

\[
0 \leq Q_n^u(t) \leq MaxQ_n^u \quad Q_n^u(t+1) = |Q_n^u(t) - f_{u,n}^t(t)|_{+} + r_{u,n}(t) + f_{u,n}^t(t) \quad (1)
\]

Where the operator \(| . |_{+} \) represents \( max(0, .) \) and \( MaxQ_n^u > 0 \) is the maximum queue length (i.e. allocated data buffer size) of \( Q_n^u \). \( r_{u,n}(t) \) is the output of performing sensing tasks in node \( n \) for user \( u \) at time slot \( t \). \( f_{u,n}^{in}(t) \) and \( f_{u,n}^{out}(t) \) are the incoming and outgoing traffic of node \( n \) for user \( u \), respectively. It is worth noting that the queue length of any gateway always equals to zero.

**B. Node Resources**

Each node \( n \in N \) offers \( K \) types of resources for users such as computing (i.e. MCU), memory, bandwidth, and light sensors. Let a \( K \)-dimensional vector \( c_{n}(t) = (c_{n,1}(t), ..., c_{n,K}(t)) \) represents node’s resource capacities at time slot \( t \), where each entry \( c_{n,k}(t) \) \( 1 \leq k \leq K \) represents the resource capacity of \( k \)th resource of \( n \) at \( t \). Let \( C(t) \) be a \( |N| \times K \)-dimensional matrix representing the resource capacity for the whole network at \( t \).

In the rest of this paper, we will use a tuple \((n,k), n \in N, 1 \leq k \leq K \) to refer to a specific resource.
C. User Resource Requirements

Let \( \hat{R}^u \) be a \( |\mathcal{N}| \times K \)-dimensional matrix in which the entry \( \hat{R}^u_{n,k} \) represents the requirement from \((n,k)\) to perform a sensing task by user \(u\). Let \( R^u\) be a \( |\mathcal{N}| \times K\)-dimensional matrix in which the entry \( R^u_{n,k}\) is the amount of resource needs from \((n,k)\) to perform a data forwarding task on \(G_n^u\).

The gross resource requirements of user \(u\) is represented as a \( |\mathcal{N}| \times K\) matrix \( R^u\), where:
\[
R^u_{n,k} = \hat{R}^u_{n,k} + \tilde{R}^u_{n,k} \tag{2}
\]

Consider \( D_u \) to represent the set of strictly positive resource demands of user \(u\):
\[
D_u := \{(n,k) : R^u_{n,k} > 0, \ n \in \mathcal{N}, \ 1 \leq k \leq K\} \tag{3}
\]

D. Topological User Dependency

This section presents a graph model to characterize the dependency of all users in \(\mathcal{U}\), with respect to resource requirements.

**Definition 1.** [User Dependency Graph]. We can use an undirected graph \(G(\mathcal{U}, \mathcal{V})\) to represent the dependency of all users, where
\[
\mathcal{V} := \{(u,v) : u, v \in \mathcal{U}, \ D_u \cap D_v \neq \emptyset\}
\]
is the set of the links in the graph. Each link \((u,v) \in \mathcal{V}\) indicates that two users \(u\) and \(v\) share some resources.

**Definition 2.** [User Dependent Cluster (UDC)]. For a given user dependence graph \(G(\mathcal{U}, \mathcal{V})\), a User Dependent Cluster (UDC) \(C \subseteq \mathcal{U}\) is defined as the set of users in a connected component\(^1\) of \(G(\mathcal{U}, \mathcal{V})\).

Define \(\mathcal{U}_{n,k}\) as the set of users that have a strictly positive demand of resource \((n,k)\), \( n \in \mathcal{N}, 1 \leq k \leq K\), i.e.
\[
\mathcal{U}_{n,k} := \{u : R^u_{n,k} > 0, \ u \in \mathcal{U}\} \tag{4}
\]

It can be seen that two users \(u_1\) and \(u_2\) in a UDC \(C\) may not share common resources, meaning that they are not directly dependent on each other. However the resource requirements of \(u_1\) would affect that of all neighbors \(v \in \mathcal{U}_{n,k}, \ (n,k) \in D_{u_1}\). By repeating the above process, the requirement of \(u_1\) would affect all users in \(C\), including \(u_2\). This implies that all users in a UDC are directly or indirectly dependent in terms of resource requirements.

E. Objective

The objective of this paper is to develop a distributed, fair scheduling algorithm to allocate resources for sensing and data forwarding tasks for the users of the network. In particular for every time slot \(t\), it is required to distributedly compute two \(|\mathcal{N}| \times |\mathcal{U}|\)-dimensional matrices: \(\hat{X}(t)\) and \(\tilde{X}(t)\), where the entries \(\hat{X}_{u,n}(t)\) and \(\tilde{X}_{u,n}(t)\) are the number of sensing and forwarding tasks performed by user \(u\) in node \(n\), respectively.

The network resources allocated to user \(u\) at time slot \(t\) is defined as a \(|\mathcal{N}| \times K\)-dimensional matrix \(A^u(t, \tilde{X}(t))\) where each entry \(A^u_{n,k}(t)\) represents the amount of resource \((n,k)\) allocated to user \(u\) at time slot \(t\):
\[
A^u_{n,k}(t) = \hat{X}_{u,n}(t)\hat{R}^u_{n,k} + \tilde{X}_{u,n}(t)\tilde{R}^u_{n,k} \tag{5}
\]

**Definition 3.** [Feasible Resource Allocation]. A resource allocation matrix \(A^u(t, \tilde{X}(t))\) is feasible if the following condition is satisfied:
\[
\sum_{u \in \mathcal{U}} A^u(t, \tilde{X}(t)) \leq C(t) \tag{6}
\]

where \(\leq\) means entry-wise smaller than or equal to.

III. SYMBIOT

Let \(d_{u,n,k}(t) = R^u_{n,k}/C_{n,k}(t)\) be the resource demand share of user \(u\) for resource \((n,k)\) at slot \(t\). Let \(d^\text{max}_{u,n}(t)\) denote the node-wise dominant demand share of user \(u\) in node \(n\).
\[
d^\text{max}_{u,n}(t) = \max_{1 \leq k \leq K} d_{u,n,k}(t) \tag{7}
\]

In Symbiot we normalize the resource demand shares of users in respect to their node-wise dominant demand share as follow:
\[
d_{u,k}(t) = d_{u,n,k}(t)/d^\text{max}_{u,n}(t) \tag{8}
\]

For each user \(u\) in a UDC \(C\), define UDC-wise dominant demand share as
\[
d^\text{max}_u(t) = \max_{(n,k) \in D(C)} d^\text{max}_{u,n}(t), \ u \in C \tag{9}
\]
where
\[
D(C) := \bigcup_{u \in C} D_u
\]
denotes the set of all resources used by all users in \(C\). For a given \(d^\text{max}_u(t)\), we denote the node \(n^* = \arg \max_n d^\text{max}_{u,n}(t)\) as the bottleneck node for user \(u\) in \(C\). We define the UDC-wise dominant share for user \(u\) in \(C\) as:
\[
d^\text{max}_u(t)(\hat{X}_{u,n^*}(t) + \tilde{X}_{u,n^*}(t)) \tag{10}
\]

A. Algorithm

The basic idea of Symbiot is to implement a distributed, light-weight algorithm to approximate the max-min fairness for the UDC-wise dominant shares for each UDC.

Because the algorithm is distributed, every node \(n \in \mathcal{N}\) computes resource scheduling locally. However, we utilize network traffic congestion to collect feedback from the bottleneck nodes existing in UDCs. The pseudo code of Symbiot is summarized in Fig. 2 and Fig. 3.

In node \(n\) at the beginning of time slot \(t\), all users that use \(n\) are considered to be unsaturated (line 01 in Fig. 2) and stored in the unsaturated user set \(\mathcal{U}_u\). Symbiot allocates resources for the unsaturated users through multiple iterations.
The pseudo code of Symbiot is given below:

**Variables:**
- $t$: current time slot for the node
- $n$: the current node
- $\mathcal{U}_{us}$: the set of unsaturated users
- $\mathcal{U}_{ut}$: the set of users that performed a task in an iteration
- $B$: a $|\mathcal{U}| \times K$ matrix holding resource budgets for an iteration
- $\hat{A}$: a $|\mathcal{U}| \times K$ matrix holding allocated resources for an iteration.

**Input:**
- $\mathcal{R}_n := \{\hat{R}_n, k \mid \forall u \in \mathcal{U}, 1 \leq k \leq |K\}$
- $\mathcal{R}_u := \{\hat{R}_u, k \mid \forall u \in \mathcal{U}, 1 \leq k \leq |K\}$

**Output:**
- $\overline{X}_{u,n}(t), \overline{X}_{u,n}(t) \forall u \in \mathcal{U}$

**Main Algorithm:**
1. $\mathcal{U}_{us} \leftarrow \{u \in \mathcal{U} \mid \hat{A}_{u,k} \forall k\};$ // $\mathcal{U}_{us}$ defined in Eq. 4
2. while $\mathcal{U}_{us}(t) \neq \emptyset$ do
3. $\hat{A} \leftarrow 0, \mathcal{U}_{ut} \leftarrow \emptyset;$ //reset the parameters for this iteration
4. compute $d_{u,k}(t), \forall k, u \in \mathcal{U}_{us};$ //based on Eq. 8
5. $b \leftarrow 1/\max \sum_{\forall k \in \mathcal{U}_{us}, u} d_{u,k}(t);$ //node-wise dominant demand share maximization
6. for all $u \in \mathcal{U}_{us}(t)$
7. $\mathcal{B}_{u,k} \leftarrow d_{u,k}(t)\hat{c}_{u,k}(t)b, \forall k;$ //assign resource budgets
8. if containLocalPacket($Q_n^*(t)$) = FALSE
9. $[r \mathcal{A}_{u,n}] \leftarrow$ sensing $(u, \hat{A}, \mathcal{U}_{ut}, \mathcal{R}_n, B);$ //sensing and storing data in $Q_n^*$
10. $\overline{X}_{u,n}(t) \leftarrow \overline{X}_{u,n}(t) + r;$
11. end if
12. $r \leftarrow 1;
13. $\mathcal{U}_{us} \leftarrow \mathcal{U}_{us} \cap \mathcal{U}_{ut};$ // update unsaturated users
14. while $r = 1$
15. $[r \mathcal{A}_{u,n}] \leftarrow$ dataForwarding $(u, \hat{A}, \mathcal{U}_{ut}, \mathcal{R}_n, B);$ //data forwarding operations are traffic-aware
16. $\overline{X}_{u,n}(t) \leftarrow \overline{X}_{u,n}(t) + r;$
17. end while
18. end for
19. $c_{u,k}(t) \leftarrow c_{u,k}(t) - \sum_{u \in \mathcal{U}_{us}} \hat{A}_{u,k}\forall k;$ //update available resources based on actual consumptions
20. $\mathcal{U}_{us} \leftarrow \mathcal{U}_{us} \cap \mathcal{U}_{ut};$ // update unsaturated users
21. end while

**Remark 1.** Both sensing and data forwarding operations are traffic-aware. Even if a user has the required budget to perform a task, Symbiot will not allocate resources if traffic congestion is observed for that user. In line 01 in Fig. 3, sensing tasks check whether the queue of the current node is not full. Line 09 in Fig. 3 ensures data forwarding tasks are not performed if the queue of the next-hop node is not full.

**Remark 2.** In a bottleneck node, the data queues of some users will be full. A bottleneck node $n^*$ in a UDC will limit the budget of one or more users as they are seen to be locally unfair with respect to other users. This means in the node $n^*$, the $f_{u,n^*} < f_{u,n}$ for some users. Based on the queue dynamics defined in Eq. 1, the queue levels of some $Q_n^*$ will gradually increase until it reaches the corresponding $\max Q_n^*$. This results in affecting the sensing operations performed on these nodes. This process repeats itself throughout the UDC and will affect all nodes that generating and sending data packet for user $u$ through the $n^*$.

Fig. 2. Symbiot - Main Algorithm

Fig. 3. Symbiot - Sensing and Data Forwarding functions

(lines 02-21 in Fig. 2). In each iteration Symbiot finds a budget $b$ that maximizes the equalized node-wise dominant demand shares of these users (lines 04-05 in Fig. 2). This computation also ensures that Symbiot adheres to the capacity constraint introduced in the Definition 3.

Then Symbiot assigns each unsaturated user a resource budget for every resource type $k$ (line 07 in Fig. 2). The given resource budgets can be utilized for sensing tasks, data forwarding tasks or both. For sensing tasks, Symbiot first checks whether the data queue of the user contains a local packet (i.e. generated by $n$). If no local packet found, Symbiot executes the sensing function. It is worth noting that in the pseudo code of Symbiot, we assume the data queues of the users follow the First-In-First-Out (FIFO) scheduling. Symbiot utilizes the remaining budget for data forwarding tasks by calling the dataForwarding function. As shown in Fig. 3, both sensing and dataForwarding update the resource consumptions and active users set $\mathcal{U}_{ut}$ for the current iteration. These functions returns 1 if the sensing or data forwarding tasks is actually executed, otherwise 0. Based on the feedback given in the lines 09-10 and 15-16 in Fig. 2, Symbiot updates the $\overline{X}_{u,n}(t)$ and $\overline{X}_{u,n}(t)$ outputs for all unsaturated users. Then the remaining resource capacities are adjusted according to the resource consumptions of this iteration. In line 20 in Fig. 2, Symbiot removes saturated users from $\mathcal{U}_{us}$. We define a saturated user as a user who did not perform any task, whether sensing or data forwarding, in an iteration. This process continues until all users become saturated.
IV. EVALUATION

The performance of Symbiot was evaluated through conducting a set of simulation on Cooja. We implemented Symbiot on top of Contiki OS [7] and its IPv6 stack. The implementation provides a set of public APIs to allow users to express their sensing and data forwarding requirements. Internally it uses UDP messages and IPv6 to transfer data packets from nodes to gateways. We use ICMPv6 to periodically broadcast the queue lengths of the users between nodes. We established a 100-node network with 5 gateways and 95 nodes in a 600m×600m area. The network topology of our experiments is shown in Fig. 4. The network uses CSMA and RPL with the default parameter settings for the MAC and routing layers, respectively. We set RPL to use the ETX Objective Function. The nodes employ IEEE 802.15.4 transceiver, CC2420 with transmission range of 50m. Based on the experimental studies in [8], the bandwidth capacity of the nodes is set to be 160 40-bytes packets per second. The nodes in our network uses MSP430F1611 microcontroller with 10KB RAM. The network offers two types of sensors: temperature and humidity with the resource capacities of 1000 and 2000 readings per second, respectively. We randomly chose 40 nodes to offer temperature sensing and another 40 nodes for humidity sensing. In all simulations the duration of a time slot was set as one second and MaxQ$^u_n$ for all users in all nodes were equal to 70.

In the reset of the paper, we will use the vector $⟨r_1$ packets, $r_2$ bytes of RAM, $r_3$ temperature readings, $r_4$ humidity readings$⟩$ to represent the resource requirements of offered resource types in the experiments.

A. Dynamic Resource Allocation

In our first experiment, we show how Symbiot dynamically allocates resources between users in a system with time-varying resource requirements and capacities. Fig. 5 shows the average allocated network resources and UDC-wise shares for each users as a function of time.

In the beginning, the network has two users $u_1$ and $u_2$. $u_1$ has the resource requirements of $⟨0, 1, 2, 0⟩$ for sensing and $⟨1, 20, 0, 0⟩$ for data forwarding. Similarly, $u_2$ needs $⟨0, 10, 0, 20⟩$ for sensing and $⟨2, 50, 0, 0⟩$ for data forwarding operations. As seen in Fig. 5, the UDC-wise dominant shares for both users are equal to 50% as bandwidth is the most demanded resource for both users in bottleneck nodes. At slot 300, $u_2$ changes his/her resource requirements making it more memory demanding. $u_2$ now demands $⟨0, 2048, 200, 0⟩$ for sensing and $⟨9, 512, 0, 0⟩$ for data forwarding. Symbiot dynamically adjusts the resource allocation for both users. The UDC-wise dominant shares increased to around 60% as the most required resource for $u_1$ is bandwidth while for $u_2$ now it becomes RAM. At time slot 600, $u_3$ enters the network.
network with the resource requirements of $\langle 0, 200, 20, 20 \rangle$ for sensing and $\langle 4, 200, 0, 0 \rangle$ for data forwarding. Symbiot lowers the resource allocation for both $u_1$ and $u_2$ to accommodate the resource demands for the new users. As $u_2$ enters the network, the nodes in beginning tries to perform as much sensing as possible until bottleneck nodes are found causing traffic congestion. This will result in gradually lowering the sensing rate as shown between slot 600 and 700. At slot 900 when $u_2$ leaves the network, Symbiot allocates the released resources from $u_2$ to $u_1$ and $u_3$. At time slot 1900, we intentionally reduce the bandwidth capacities of all nodes by 50%. Symbiot automatically adjusts the allocations for both $u_1$ and $u_3$.

B. Symbiot vs Alternative Multi-Resource Allocation

To compare the performance of Symbiot with another multi-resource allocation algorithm, we implemented a naive version of distributed DRF. Here, each node runs the progressive filling of DRF [4]. However, the sensing and data forwarding tasks are not traffic-aware. If any node receives data packets more than $MaxQ_n$, the node will drop the packets. In these experiments, we measured the completion times needed to perform sensing tasks on all the nodes required by the users.

Fig. 6 presents the reductions of average completion times in networks with different numbers of gateways. Here, the network has two users with sensing requirements of $\langle 0, 1, 2, 0 \rangle$ and $\langle 0, 10, 0, 20 \rangle$ and forwarding requirements of $\langle 1, 20, 0, 0 \rangle$ and $\langle 2, 50, 0, 0 \rangle$. Fig. 7 shows Symbiot outperforms the implemented distributed DRF in networks with different numbers of users. The used topology had 5 gateways and the sensing and forwarding requirements were randomly generated.

VI. Conclusion

This paper addresses the problem of multi-resource fairness between users in data gathering applications in WSNs and IoT. Here, nodes can offer multiple types of resources with heterogeneous, time-varying capacities. Users can have different requirements on various resource types. We explain why the current state-of-the-art falls short when it comes to achieve multi-resource fairness in such networks. By utilizing graph theory, queuing theory and dominant resource sharing, we propose Symbiot, a lightweight, distributed algorithm as a solution to this problem. Our simulations based on Cooja network emulator shows the practical performance of Symbiot.

REFERENCES