Distributed Throughput Optimization for ZigBee Cluster-Tree Networks

Yu-Kai Huang[§], Ai-Chun Pang[*†], Pi-Cheng Hsiu[**], Weihua Zhuang[‡], and Pangfeng Liu[†]

Kyle.Huang@quantatw.com, acpang@csie.ntu.edu.tw, pchsiu@citi.sinica.edu.tw,
wzhuang@uwaterloo.ca, pangfeng@csie.ntu.edu.tw

[§] Quanta Research Institute, Quanta Computer
[*] Graduate Institute of Networking and Multimedia, National Taiwan University
[**] Research Center for Information Technology Innovation, Academia Sinica
[†] Department of Computer Science and Information Engineering, National Taiwan University
[‡] Department of Electrical and Computer Engineering, University of Waterloo

❖

Abstract—ZigBee, a unique communication standard designed for low-rate wireless personal area networks, has extremely low complexity, cost, and power consumption for wireless connectivity in inexpensive, portable, and mobile devices. Among the well-known ZigBee topologies, ZigBee cluster-tree is especially suitable for low-power and low-cost wireless sensor networks because it supports power saving operations and light-weight routing. In a constructed wireless sensor network, the information about some area of interest may require further investigation such that more traffic will be generated. However, the restricted routing of a ZigBee cluster-tree network may not be able to provide sufficient bandwidth for the increased traffic load, so the additional information may not be delivered successfully. In this paper, we present an adoptive-parent-based framework for a ZigBee cluster-tree network to increase bandwidth utilization without generating any extra message exchange. To optimize the throughput in the framework, we model the process as a *vertex-constraint maximum flow* problem, and develop a distributed algorithm that is fully compatible with the ZigBee standard. The optimality and convergence property of the algorithm are proved theoretically. Finally, the results of simulation experiments demonstrate the significant performance improvement achieved by the proposed framework and algorithm over existing approaches.

1 INTRODUCTION

Recent advances in wireless communications and micro-electromechanical technologies have had a strong impact on the development of wireless sensor networks (WSNs) [2]. The IEEE 802.15.4 protocol is a promising standard for WSN applications because it pays particular attention to energy efficiency and communication overheads [1]. Based on the physical (PHY) and medium access control (MAC) layers of IEEE 802.15.4, the upper-layer (including the network and application layers) specifications are defined by the ZigBee protocol stack. The applications supported by ZigBee include home automation schemes, remote control and monitoring systems, and health care devices [3].

Among the well-known ZigBee topologies, the cluster tree is especially suitable for low-power and low-cost WSNs because it supports power saving operations and light-weight routing. In the ZigBee cluster-tree topology, the power saving operation is managed by the IEEE 802.15.4 MAC superframe structure; and a light-weight tree routing protocol is enabled under a distributed address assignment policy configured by several system parameters. Although the ZigBee cluster-tree network is effective for WSNs, the topology suffers from restricted routing and poor bandwidth utilization. In a tree structure, any link failure will suspend data delivery completely and the recovery operation will incur a considerable overhead. The topology also prevents the use of many potential routing paths, which means that a considerable amount of bandwidth can not be utilized. In a constructed WSN, the information about some area of interest may require further investigation As a result, the sampling rate of the sensor nodes deployed in the area of interest will be increased, and more traffic will be generated suddenly in the network. However, as the ZigBee cluster-tree topology may not be able to provide sufficient bandwidth for the increased traffic load, the additional information may not be delivered successfully.

In the literature, there is a rich body of research on ZigBee cluster-tree networks. For example, Mišić et al. [16] [17] [18] presented master-slave and slave-slave bridge access modes for the interconnection of IEEE 802.15.4 beacon-enabled network clusters, and developed theoretical analysis to investigate their performance. Cuomo et al. [7] demonstrated the benefits of ZigBee tree routing with respect to reactive routing in typical sensor network applications. Khan et al. [10] conducted a comprehensive performance evaluation

of ZigBee cluster tree networks and provided important insights into engineering systems for developers. Koubaa et al. [13] presented an analytical model that derives worst-case end-to-end delay bounds under buffering and bandwidth requirements. Han [9] proposed an algorithm that configures cluster-tree parameters optimally and guarantees all the end-to-end deadlines of periodic real-time flows deterministically. Peng et al. [19] introduced a selection strategy for ZigBee tree and mesh routing in various applications; while Kim et al. [11] developed shortcut tree routing to reduce the transmission latency. The common drawback of the existing approaches is that they do not address the poor bandwidth utilization problem in ZigBee cluster-tree networks, so it is difficult to increase the system throughput.

On the other hand, some notable results have been reported for throughput improvement in the routing protocols of ZigBee mesh networks. For example, Gowrishankar et al. [8] conducted a complete performance evaluation of four AODV-like routing protocols in terms of the packet delivery ratio, average network delay, network throughput and normalized routing load. In addition, several ZigBee mesh routing protocols have been proposed to reduce the overhead for route discoveries and route requests, and thereby improve the end-to-end delay and packet delivery ratio [15] [14] [4] [22]. Kim et al. [12] proposed an improved AODV routing protocol to increase the throughput of ZigBee wireless networks. Unfortunately, the above models cannot be adopted in ZigBee cluster-tree networks because they make specific assumptions about the system architecture.

The beacon scheduling problem, which is unique to ZigBee cluster-tree networks, has also been well studied. Saeyoung el al. [20] proposed a slotted beacon scheduling scheme to reduce power consumption in a hierarchical tree topology; and Yamao et al. [23] presented a novel active-period assignment method to relieve the throughput bottleneck in ZigBee cluster-tree networks. Burda et al. [5] introduced a distributed beacon scheduling scheme that is suitable for sensor networks, and the robustness and performance are maintained during reconfiguration. Considering the transmission latency, Tseng et al. [21] defined a minimum delay beacon scheduling problem for quick convergecast in ZigBee tree-based WSNs, and proposed a distributed beacon scheduling scheme with a delay consideration. Although existing works can achieve significant performance improvements in ZigBee cluster-tree networks, they do not deal with the restricted routing and poor bandwidth utilization problems.

In this paper, we propose an adoptive-parent-based framework for a ZigBee cluster-tree network. Our objective is to provide more flexible routing and increase bandwidth utilization without violating the operating principles of the ZigBee cluster-tree protocol. The framework is well suited to networks in which there are sudden requirements for increased bandwidth to deliver additional information. Based on the existing cluster-tree topology, the framework allows a ZigBee node to request bandwidth from adjacent routers (called adoptive parents) as well as from its original parent router. To optimize the throughput in the framework, we model the process as a *vertex-constraint maximum flow* problem. To solve the problem, we propose a distributed algorithm that is fully compatible with the ZigBee standard. The optimality and convergence property of the algorithm are proved theoretically. The results of simulation experiments manifest the framework's significant performance improvement over the ZigBee standard.

The remainder of this paper is organized as follows. In Section 2, we present the system architecture. Section 3 describes the proposed adoptive-parent-based framework and the distributed throughput optimization algorithm. In addition, the optimality and convergence of the algorithm are demonstrated via theoretical analysis. In Section 4, we evaluate the framework's performance via extensive simulations. Section 5 contains some concluding remarks.

2 SYSTEM ARCHITECTURE

The IEEE standard, 802.15.4, defines the physical layer and medium access control sublayer for low-rate wireless personal area networks (LR-WPANs) [1]. IEEE 802.15.4 defines a superframe structure that begins by transmitting a beacon issued by a PAN coordinator. The process consists of an active portion and an inactive portion. The coordinator and devices can communicate with each other during the active period and enter a low-power phase during the inactive period. The active portion with 16 time slots is comprised of three parts: a beacon, a contention access period (CAP), and a contention free period (CFP). The beacon is transmitted by the coordinator at the beginning of slot 0, and the CAP follows immediately after the transmission. During the CAP, devices can transmit non-time-critical messages and MAC commands. In the CFP, the standard protocol provides a Guaranteed Time Slot (GTS) mechanism that ensures the devices can occupy the time slots exclusively for transmission. For those devices that desire guaranteed time slots in the next superframe's CFP, they send GTS requests to the coordinator during the current superframe's CAP. Then, the coordinator checks if there is available bandwidth in the current superframe, and determines, based on an FCFS fashion, a device list for GTS allocation in the next superframe. Finally, the GTS descriptor is included in the subsequent beacon to announce the allocation information.

ZigBee [3], which is based on the IEEE 802.15.4 standard, defines the network (NWK) layer and the application layer (APL) in the protocol stack. There

are three types of device in a ZigBee network: a coordinator, a router, and an end device. A ZigBee network is comprised of a ZigBee coordinator and multiple ZigBee routers/end-devices. The coordinator provides the initialization, maintenance, and control functions for the network. The router has a forwarding capability to route sensed data to a sink node. The end device lacks such a forwarding capability. ZigBee supports three kinds of network topology, namely, star, cluster-tree, and mesh topologies. In a star network, multiple ZigBee end devices connect directly to the ZigBee coordinator. For cluster-tree and mesh networks, communications can be conducted in a multi-hop fashion through ZigBee routers. In a cluster-tree network, each ZigBee router with its surrounding devices is regarded as a respective cluster, and each cluster operates individually as a star network, as shown in Figure 1. In this paper, we assume that sensed data in ZigBee cluster-tree networks is delivered by the GTS mechanism because a high delivery ratio can be guaranteed [13].
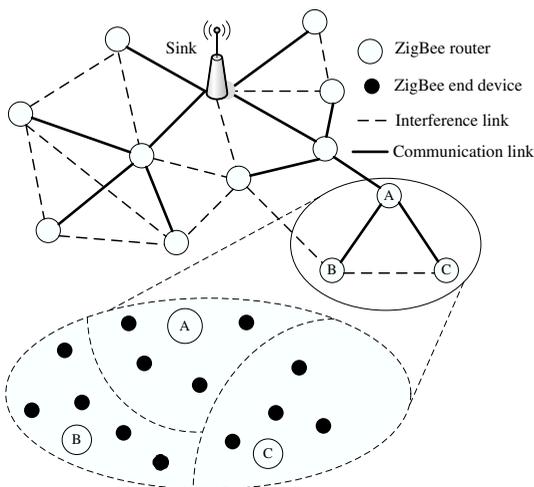

Fig. 1. The ZigBee cluster-tree network

Although the ZigBee cluster tree provides an effective solution for low-power and low-cost wireless sensor networking, the rigidness of the topology makes it vulnerable to link failures. To resolve such problems, we propose an adoptive-parent-based framework for a ZigBee cluster-tree network. The framework provides more flexible routing and increases bandwidth utilization without violating the operating principles of the ZigBee protocol. Under the framework, when a ZigBee router suddenly initiates data transmissions that need much more bandwidth than usual, the router is allowed to request bandwidth from adjacent routers (called adoptive parents) as well as from its original parent router. Additional routing paths are established through the transmission links to the adoptive parents in order to increase the bandwidth between the *source* and the *sink* and thereby satisfy the sudden requirement for extra bandwidth.

3 DISTRIBUTED THROUGHPUT OPTIMIZATION IN ADOPTIVE-PARENT-BASED FRAMEWORK

To realize the concept of adoptive parents in Zig-Bee cluster-tree networks, we utilize the concept of distributed throughput optimization in an adoptive-parent-based framework. In the following, we begin by formulating the throughput maximization problem as a *vertex-constraint maximum flow* problem in Section 3.1; and then propose a distributed algorithm to resolve the problem in Section 3.2. Finally, we provide a theoretical analysis of the optimality and convergence property of the distributed algorithm in Section 3.3.

3.1 Problem Formulation

A vertex-constraint flow network can be formulated as a directed graph $G = (V, E)$, where V represents the routers in the network and E represents the possible communication links between pairs of routers. In a traditional flow network, each edge has a non-negative capacity. By contrast, in a vertex-constraint flow network, each vertex $u \in V$ is associated with a non-negative capacity, denoted by $\hat{c}(v) \geq 0$, which represents the GTS capacity of the router. Each directed edge (u, v) is associated with an implicit capacity $c(u, v) = \infty$ if $(u, v) \in E$; otherwise, if $(u, v) \notin E$, then $c(u, v) = 0$. For each flow, two vertices are distinguished in the network: a source s and a sink t, where s is the sender of the data that requires additional bandwidth and t is the data receiver.

A *flow* in a vertex-constraint flow network G with respect to a source s and a sink t is a real-value function $f : V \times V \to R$ that satisfies the following three properties:

- *Capacity constraint*:
  $\sum_{u \in V} \{f(u, v) | \ f(u, v) > 0\} \leq \hat{c}(v), \forall v \in V$.
- *Skew symmetry*:
  $f(u, v) = -f(v, u), \forall u, v \in V$.
- *Flow conservation*:
  $\sum_{u \in V} f(u, v) = 0, \forall v \in V - \{s, t\}$.

The quantity $f(u, v)$, which can be positive, zero, or negative, is called the *net flow* from vertex u to vertex v. The capacity constraint, which relates to a router's physical resource usage, stipulates that the net flow passing through the router must not exceed its capacity. Because of the skew symmetry property, the net flow from one vertex to another vertex is the negative of the net flow in the opposite direction; and because of flow conservation, the total net flow into a vertex, except the source or sink is equal to zero. The value of a flow f is defined as the total net flow into the sink, i.e., $\sum_{u \in V} f(u, t)$. A maximum flow is a flow of *maximum value*. In the *vertex-constraint maximum flow* problem, given a vertex-constraint flow network G with source s and sink t, the objective is to find a maximum flow f from s to t in G.

3.2 A Distributed Algorithm

As mentioned previously, we propose a distributed algorithm to resolve the *vertex-constraint maximum flow* problem. Specifically, we revise the *push-relabel method* [6], which underlies many of the asymptotically fastest algorithms used to solve traditional network-flow problems. The proposed algorithm, called the *pull-push-relabel* (PPR) algorithm, is designed to adapt to a ZigBee cluster-tree network of a certain scale. In the following, we present our algorithm and demonstrate its optimality and convergence.

First, we define some notations and terms. Given a vertex-constraint flow network $G = (V, E)$ with source s and sink t, let f be a flow in G, and let a vertex $v \in V$. The amount of extra flow that can be added to v before exceeding the capacity $\hat{c}(v)$ is called the *residual capacity* of v, and is given by

$$\hat{c}_f(v) = \hat{c}(v) - \sum_{u \in V} \{f(u, v) | f(u, v) > 0\}.$$

Similarly, for any pair of vertices $u, v \in V$, the residual capacity of (u, v) is the extra flow that can be moved from u to v before exceeding the capacity $c(u, v)$, denoted as

$$c_f(u, v) = c(u, v) - f(u, v).$$

Note that $c(u, v)$ is either ∞ or 0 depending on whether (u, v) is in E. Hence, $c_f(u, v) = \infty$ if $(u, v) \in E$; otherwise, $c_f(u, v) = -f(u, v)$. The *residual network* of G induced by f is $G_f = (V, E_f)$, where E_f is the *residual edge set* defined as follows:

$$E_f = \left\{ (u, v) \, \middle| \, \begin{array}{ll} \hat{c}_f(v) > 0, & \text{if } c(u, v) = \infty, \\ c_f(u, v) > 0, & \text{Otherwise.} \end{array} \right\}$$

That is, a residual edge (u, v) is included in E_f if it can admit a positive net flow from u to v. Note that (u, v) may be a residual edge in E_f even if it is not an edge in E. A *preflow* is a function $f : V \times V \to R$ that satisfies the capacity constraint, skew symmetry, and the following relaxation of flow conservation rule:

$$\sum_{u \in V} f(u, v) \geq 0, \forall v \in V - \{s\}.$$

That is, the total net flow into a vertex v, except the source, can be greater than zero. The total net flow into v is called its excess flow, denoted as $e(v) = \sum_{u \in V} f(u, v)$. In addition, a vertex $v \in V - \{s, t\}$ is said to be *overflowing* if $e(v) > 0$. Let f be a preflow in G. A function $h : V \to N$ is a *height function* for f if $h(s) = |V|$, $h(t) = 0$, and $h(u) \leq h(v) + 1$ for every residual edge $(u, v) \in E_f$. Intuitively, the height of a vertex determines the direction of force imposed on the flow; that is, a flow moves downwards from a higher vertex to a lower vertex.

We now explain the rationale behind the algorithm. Throughout the algorithm, the height of the source s is fixed at $|V|$, and the height of the sink t is fixed at 0. The height of every other vertex starts at 0 and increases over time. Initially, the vertices are lower than the source, so they tend to pull the flow[1] downwards from s toward t; however, any "over-pulled" flow may be *pushed* back upwards towards s eventually. During the algorithm's execution, if the excess flow of a vertex can not be pushed or pulled in either direction, the vertex *relabels* itself to increase its height. Eventually, all the flow that can possibly get through to the sink has arrived there. Vertices tend to push the over-pulled flow back to the source by increasing their height so that they are above the fixed-height source. Once any over-pulled flow has been removed from the vertices, the flow is deemed to be maximum flow.

Procedure 1 PULL(u, v)

Applicability: $e(v) > 0$, $c(v, u) = \infty$, $\hat{c}_f(u) > 0$, and $h(v) = h(u) + 1$
Action: u pulls $\delta = \min(e(v), \hat{c}_f(u))$ units of flow from v

  1: $f(v, u) \leftarrow f(v, u) + \delta$
  2: $f(u, v) \leftarrow -f(v, u)$
  3: $e(v) \leftarrow e(v) - \delta$
  4: $e(u) \leftarrow e(u) + \delta$

Procedure 2 PUSH(u,v)

Applicability: $e(u) > 0$, $c(u, v) \neq \infty$, $c_f(u, v) > 0$, and $h(u) = h(v) + 1$
Action: u pushes $\delta = \min(e(u), c_f(u, v))$ units of flow to v

  1: $f(u, v) \leftarrow f(u, v) + \delta$
  2: $f(v, u) \leftarrow -f(u, v)$
  3: $e(u) \leftarrow e(u) - \delta$
  4: $e(v) \leftarrow e(v) + \delta$

In a PULL(u, v) operation, a lower vertex u pulls the flow of a higher vertex v downwards to itself. The operation can only be implemented if all the following conditions are satisfied:

- v is overflowing, i.e, $e(v) > 0$;
- there is an edge from v to u in G, i.e, $c(v, u) = \infty$;
- the residual capability of u is positive, i.e., $\hat{c}_f(u) > 0$;
- u is lower than v by 1, i.e., $h(v) = h(u) + 1$.

If the conditions are satisfied, u determines the amount of flow, $\delta = \min(e(v), \hat{c}_f(u))$, which can be pulled from v to u without causing the excess flow of v to become negative or the capability of u to be exceeded. To ensure that f remains a preflow after the

---

1. Note that flow represents bandwidth rather than data packets. There is no intention to send data packets back and forth.

operation, the flow is moved from v to u by updating f and e accordingly (see Lines 1-4 of Procedure 1).

In a PUSH(u, v) operation, a higher vertex u pushes the over-pulled flow back to a lower vertex v along the edge $(v, u) \in G$. The operation can only be implemented if all the following conditions are satisfied:

- u is overflowing, i.e, $e(u) > 0$;
- there is no edge from u to v in G, i.e., $c(u, v) \neq \infty$;
- there is a positive net flow on (v, u), i.e., $c_f(u, v) > 0$;
- u is higher than v by one, i.e., $h(u) = h(v) + 1$.

Note that, because there is no edge from u to v in G, v cannot pull the flow from u; therefore, u pushes the over-pulled flow back to v. When PUSH(u, v) is applicable, u determines the amount of flow, $\delta = \min(e(u), c_f(u, v))$, which u can push back to v without causing the excess flow of u or the net flow on (v, u) to become negative. Similarly, we move the flow from u to v by updating f and e (see Lines 1-4 of Procedure 2) so that f remains a preflow after the operation.

Procedure 3 RELABEL(u)

Applicability: $e(u) > 0$, and $(u, v) \in E_f \Rightarrow h(u) \leq h(v)$, $\forall v \in V$

Action: u increases its height to $1 + \min\{h(v)|(u, v) \in E_f\}$

1: $h(u) \leftarrow 1 + \min\{h(v)|(u, v) \in E_f\}$

A RELABEL(u) operation enables a vertex u to increase its height. It is only applicable if the following two conditions are satisfied:

- u is overflowing, i.e, $e(u) > 0$;
- $(u, v) \in E_f$ implies that $h(u) \leq h(v)$ for all vertices $v \in V$.

In other words, an overflowing vertex u needs to be relabeled if, for every vertex v for which there is residual capacity from u to v (i.e., $(u, v) \in E_f$), the flow cannot be pushed or pulled from u to v. When RELABEL(u) is applicable, u determines the minimum height, $1 + \min\{h(v)|(u, v) \in E_f\}$, which it requires so that the excess flow can be pushed or pulled out of the vertex afterwards.

Procedure 4 PULL-PUSH-RELABEL(u)

1: **for all** $v \in Adj(u)$ **do**
2:     PULL(u,v)
3: **if** u cannot be pulled by any other vertex **then**
4:     **for all** $v \in Adj(u)$ **do**
5:         PUSH(u,v)
6:     RELABEL(u)

PULL-PUSH-RELABEL(u) is a compound operation in which a vertex u performs the three basic operations consecutively. Initially, u performs a pull operation to pull flows from each of its adjacent

vertices if applicable. Then, if u cannot be pulled by any adjacent vertex, it performs a push operation to push any over-pulled flow back to each of its adjacent vertices if applicable. After these operations, u may still be overflowing, and there may be residual capability from u to v; however, the flow cannot be pushed or pulled from u to v because of the vertices' heights. In this case, u relabels itself to increase its height.

The proposed algorithm applies the compound operation so that the vertices can manipulate the initial preflow until no overflowing vertex exists. Eventually, the flow from the source to the sink will reach the maximum. An initial preflow is created by the following subroutine:

Procedure 5 INIT(u)

1: **if** u is source s **then**
2:     $h(u) \leftarrow |V|$
3:     **for all** $v \in Adj(u)$ **do**
4:         $h(v) \leftarrow 0$
5:         $e(v) \leftarrow \hat{c}(v)$
6:         $f(u, v) \leftarrow \hat{c}(v)$
7:         $f(v, u) \leftarrow -\hat{c}(v)$
8: **else if** $u \notin Adj(s)$ **then**
9:     $h(u) \leftarrow 0$
10:     $e(u) \leftarrow 0$
11:     **for all** $v \in Adj(u)$ **do**
12:         $f(u, v) \leftarrow 0$
13:         $f(v, u) \leftarrow 0$

INIT(u) is a subroutine whereby every vertex $u \in V$ initializes itself so as to create an initial preflow in G. On completion of the initialization step, we should have an initial preflow in which every adjacent vertex of s is filled to its maximum capacity, and none of the other vertices carries any flow. Therefore, for the source s, the height $h(s)$ is set as $|V|$; for each adjacent vertex v, the height $h(v)$ is set as 0; and the excess flow $e(v)$ is set as $\hat{c}(v)$. The net flows into and out of s are also updated accordingly. For every other vertex u, the height $h(u)$ and excess flow $e(u)$ are set to 0, and the net flows into and out of the vertex are also set to 0.

Algorithm 1 PPR

1: **for all** $u \in V$ **do**
2:     INIT(u)
3: **while** there exists any overflowing vertex **do**
4:     **for all** $u \in V$ **do**
5:         PULL-PUSH-RELABEL(u)

The steps of the proposed pull-push-relabel (PPR) algorithm are shown in Algorithm 1. After a preflow is initialized, if any vertices are overflowing, Algorithm PPR repeatedly applies a process in which

all vertices perform, in any order, the PULL-PUSH-RELABEL operation sequentially. The algorithm terminates only when there are no more overflowing vertices.

For ease of presentation, Algorithm PPR is presented in a centralized manner. In Appendix II, we explain how the algorithm can be implemented in a distributed manner by utilizing the innate mechanisms of ZigBee networks. In addition, to speed up the algorithm's convergence, we introduce the concept of *parallel subsets*. Vertices in the same parallel subset have neither direct links nor common neighbors. Consequently, the parallel subsets execute the algorithm sequentially, but the vertices in each parallel subset operate simultaneously so that the algorithm's correctness is ensured. Since the ZigBee architecture implicitly enables the formation of parallel subsets, parallel executions can be achieved seamlessly, and all of the algorithm's operations can be completed without any extra message exchange.

3.3 The Properties of Algorithm PPR

In this sub-section, we consider two essential properties of the PPR algorithm, namely, the optimality of throughput maximization and the convergence of the algorithm. Given a vertex-constraint flow network $G = (V, E)$, let f be a preflow in G and let h be any height function for f. To prove the algorithm's optimality, we derive the following lemmas. Due to the page limit, the proofs are omitted, and readers can be referred to Appendix I for the details.

Lemma 1: During the execution of Algorithm PPR, if a vertex $u \in V$ is overflowing, u performs a push or a relabel operation; otherwise, a pull operation is performed on u.

Lemma 2: Whenever a vertex u performs a relabel operation on itself, its height $h(u)$ increases by at least 1.

Lemma 3: If h is initialized as a height function, then it remains a height function throughout Algorithm PPR.

Lemma 4: Let f be a preflow in G and let h be a height function of f. Then, there will not be a path from the source s to the sink t in the residual network G_f.

Lemma 5: The *vertex-constraint maximum flow* problem can be reduced to the traditional maximum flow problem[2].

Theorem 1: (Optimality of Algorithm PPR)
When Algorithm PPR terminates, the preflow f is a maximum flow from the source s to the sink t in G.

---

2. This lemma does not imply that a distributed algorithm designed for the traditional maximum flow problem can be applied to derive a maximum flow in a given vertex-constraint flow network, because the network topology after the reduction is different from the given one.

Next, we prove that the algorithm always terminates within some finite time. To normalize the algorithm's running time, we devise a specific time unit, called a pass, which represents an iteration where every vertex performs the PULL-PUSH-RELABEL operation (i.e., Procedure 4) once.

Lemma 6: For any overflowing vertex u, there must exist a path from u to the source s in the residual network G_f.

Lemma 7: During the execution of Algorithm PPR, $h(u) \leq 2|V| - 1$ always holds for any vertex $u \in V$.

Lemma 8: During the execution of Algorithm PPR, any vertex performs at most $2|V| - 1$ relabel operations.

Lemma 9: During the execution of Algorithm PPR, at least one relabel operation is performed during each pass.

Theorem 2: (Convergence of Algorithm PPR)
Algorithm PPR always terminates within $2|V|^2$ passes.

4 SIMULATIONS AND NUMERICAL RESULTS

In this section, we demonstrate the capability of the proposed adoptive-parent-based framework via a series of simulation experiments. The topology of the cluster tree under investigation is based on the ZigBee specification [3], i.e., it is assumed that each device associates with the parent router at the lowest depth. The tree construction parameters, Rm and Cm, are both set at 5, and Lm is set at 10. The beacon scheduling methodology proposed in [21], which is designed for low latency, is adopted in the experiments. Based on the ZigBee specification, the superframe parameters, BO and SO, are set at 8 and 3 respectively. In the experiments, 100 and 400 ZigBee routers are distributed in a $100 \times 100 \ m^2$ area and a $200 \times 200 \ m^2$ area respectively. The ZigBee coordinator is placed in the center of the square and serves as the data sink to collect the sensed information in the network. In our simulations, we adopted the same channel model as used in the NS-2 simulator. Based on the two-ray ground propagation model, the transmission power and receiving power threshold are set to achieve a transmission range of 20m.

The deployed ZigBee routers represent the clusters that cover different sensing regions. We assume that the deepest cluster covers the region of interest that yields additional information suddenly. The region of interest has a traffic load with a mean that ranges from $4 - 8$ packets per second, while the remaining areas have a light average traffic load of 0.01 packets per second. The packet inter-arrival times follow an exponential distribution. In the proposed adoptive-parent-based framework, multiple adoptive parents can be associated with a router. However, in the experiment, each ancestor router of the cluster of interest only

associates with one additional adoptive parent. As a result, we can observe the performance improvement over the original approach conservatively. In the performance evaluation, we analyze the throughput and transmission latency of the packets generated in the region of interest. Specifically, the throughput metric used in the experiments is normalized as a fraction of the achieved throughput over the ideal throughput (i.e., without any packet loss).
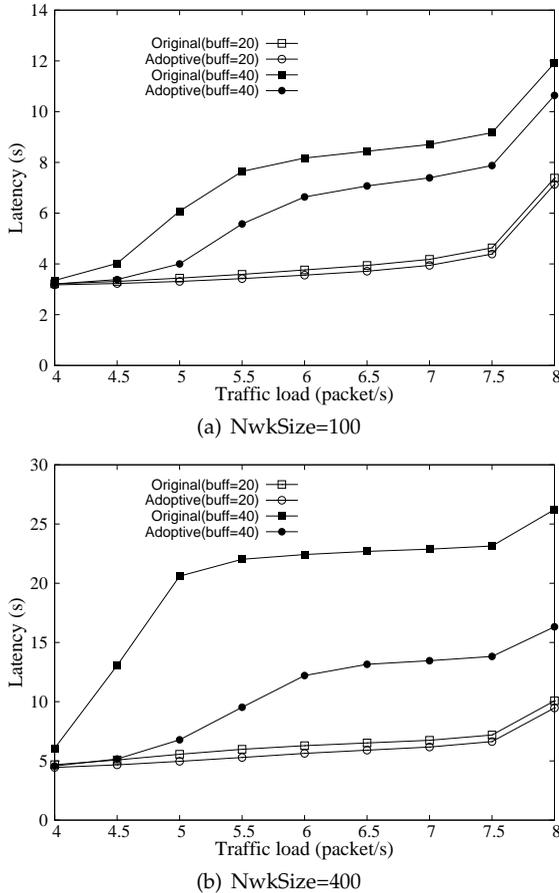

(a) NwkSize=100

(b) NwkSize=400

Fig. 2. The latency versus various traffic loads: (a) a network of 100 routers; (b) a network of 400 routers.

The following experiments demonstrate the performance improvement achieved by the adoptive-parent-based framework. Routers are placed at random in the experiments, and numerical results are averaged over the outcomes of 100 random placements. The comparison baseline is the original approach that adopts the tree topology without adoptive parents. Specifically, Algorithm PPR is executed in both the original and adoptive-parent approaches to maximize their throughputs. Figure 2 shows the effect of the traffic load of the cluster of interest on the latency with two different buffer sizes, 20 packets and 40 packets, at the routers. When the network has 100 routers, Figure 2(a) shows that the proposed framework's latency is lower than that of the original approach especially when the buffer size is 40 packets. This

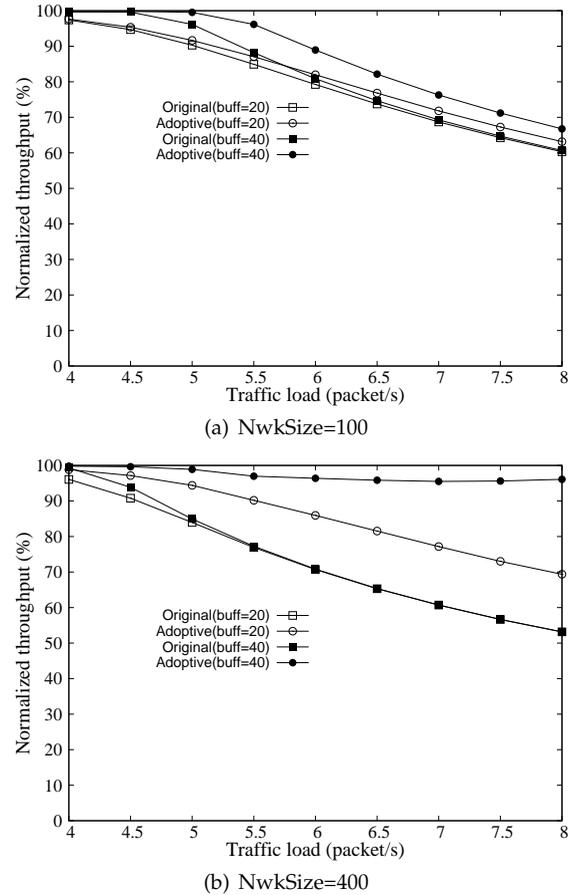(a) NwkSize=100

(b) NwkSize=400

Fig. 3. The throughput versus various traffic loads: (a) a network of 100 routers; (b) a network of 400 routers.

result also indicates that a large buffer causes longer latency, since more packets are delayed. When the network is larger, the performance improvement is much more significant, as shown in Figure 2(b). The performance improvement is as high as 400% in this experimental setting. We notice that for the uppermost curve in Figure 2(b), when the traffic load is less than 4.5 packets/s, the buffer space is sufficient to hold arriving packets. In this case, the latency increases linearly as the traffic load increases due to the queuing delay. For the traffic load is between 4.5 and 7.5 packets/s, the significant packet drops resulting from buffer overflows cause that the latency increases slowly. When the traffic load is even heavier (i.e., large than 7.5 packets/s), the network tends to be overloaded such that latency increases rapidly.

Figure 3 shows the effect of the traffic load on the throughput for the network settings in Figure 2. The normalized throughput decreases as the traffic load increases as shown in Figure 3(a). Notice that the increase of the traffic load indeed results in the increase of the achieved throughput and ideal throughput. However, due to queuing packet drops, the gap between the achieved throughput and the ideal throughput increases as the traffic load increases.

The improvement achieved by the proposed approach is not significant when the network size is small; however, it is significant when the network is large, as shown in Figure 3(b). For a buffer size of 40 packets, the normalized throughput can reach almost 100% under the proposed adoptive-parent framework. With a sufficient buffer space, the influence of queuing drops on the achieved throughput is insignificant so that the normalized throughput of the proposed framework is hardly affected by the increased traffic load. However, the original approach only can achieve 50%. The proposed framework performs better in a larger network because additional paths are established by more adoptive parents. Moreover, the increased buffer size can enhance the throughput in the proposed framework significantly. In contrast, the original approach is hardly affected by the size of buffers in the routers.

5 CONCLUDING REMARKS

In a constructed WSN, information about an area of interest may be required for further investigation, which means that more traffic can be generated. However, the restricted routing and poor bandwidth utilization in a ZigBee cluster-tree network cannot provide sufficient bandwidth for the increased traffic load, so the additional information cannot be delivered successfully. In this paper, we have proposed an adoptive-parent-based framework for a ZigBee cluster-tree network to increase the bandwidth utilization without incurring any extra message exchange. Under the framework, a throughput maximization problem, called the *vertex-constraint maximum flow* problem, is formulated, and a distributed algorithm that is fully compatible with the ZigBee standard is proposed. The theoretical analysis proves that the proposed algorithm can provide an optimal solution, and the results of simulation experiments demonstrate a significant performance improvement over the original approach.

REFERENCES

[1] 802.15.4-2003 IEEE Standard for Information Technology-Part 15.4: wireless medium access control (MAC) and physical layer (PHY) specifications for low-rate wireless personal area networks (LR-WPANs). 2003.

[2] I. F. Akyildiz, Y. Sankarasubramaniam W. Su, and E. Cayirci. A Survey on Sensor Networks. In *IEEE Comm. Magazine*, Vol. 40, pp. 102-114, Aug. 2002.

[3] ZigBee Alliance. ZigBee Specifications. Dec. 2006.

[4] A. Bhatia and P. Kaushik. A Cluster Based Minimum Battery Cost AODV Routing Using Multipath Route for ZigBee. In *Proc. IEEE International Conference on Networks (ICON)*, December 2008.

[5] R. Burda and C. Wietfeld. A Distributed and Autonomous Beacon Scheduling Algorithm for IEEE 802.15.4/ZigBee Networks. In *Proc. IEEE Internatonal Conference on Mobile Adhoc and Sensor Systems (MASS)*, October 2007.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[7] F. Cuomo, S. Della Luna, U. Monaco, and F. Melodia. Routing in ZigBee: Benefits from Exploiting the IEEE 802.15.4 Association Tree. In *Proc. IEEE International Conference on Communications (ICC)*, June 2007.

[8] S. Gowrishankar, S.K. Sarkar, and T.G. Basavaraju. Performance Analysis of AODV, AODVUU, AOMDV and RAODV over IEEE 802.15.4 in Wireless Sensor Networks. In *Proc. IEEE International Conference on Computer Science and Information Technology (ICCSIT)*, August 2009.

[9] J. Han. Global Optimization of ZigBee Parameters for End-to-End Deadline Guarantee of Real-Time Data. In *IEEE Sensor Journal*, Vol. 9, No. 5, pp. 512-514, May 2009.

[10] S.A. Khan and F.A. Khan. Performance Analysis of a ZigBee Beacon Enable Cluster Tree Network. In *Proc. International Conference on Electrical Engineering (ICEE)*, April 2009.

[11] T. Kim, D. Kim, N. Park, S. Yoo, and T.S. Lopez. Shortcut Tree Routing in ZigBee Networks. In *Proc. International Symposium on Wireless Pervasive Computing (ISWPC)*, Febrary 2007.

[12] Y.D. Kim and I.Y. Moon. Improved AODV Routing Protocol for Wireless Sensor Network based on ZigBee. In *Proc. International Conference on Advanced Communication Technology (ICACT)*, February 2009.

[13] A. Koubaa, M. Alves, and E. Tovar. Modeling and Worst-Case Dimensioning of Cluster-Tree Wireless Sensor Networks. In *IEEE International Real-Time Systems Symposium (RTSS)*, December 2006.

[14] K. K. Lee, S. H. Kim, Y. S. Choi, and H. S. Park. A Mesh Routing Protocol using Cluster Label in the ZigBee Network. In *Proc. IEEE International Conference on Mobile Adhoc and Sensor Systems (MASS)*, October 2006.

[15] K. K. Lee, S. H. Kim, and H. S. Park. Cluster Label-based ZigBee Routing Protocol with High Scalability. In *Proc. International Conference on Systems and Networks Communications (ICSNC)*, August 2007.

[16] J. Mišić. Algorithm for Equalization of Cluster Lifetimes in a Multi-Level Beacon Enabled 802.15.4 Sensor Network. In *Computer Networks*, Vol. 51, pp. 3252-3264, 2007.

[17] J. Mišić. Analysis of Slave-Slave Bridging in IEEE 802.15.4 Beacon-Enabled Networks. In *IEEE Transactions on Vehicular Technology*, Vol. 57, No. 3, pp. 1846-1864, May 2008.

[18] J. Mišić and C. J. Fung. The Impact of Master-Slave Bridge Access Mode on the Performance of Multi-cluster 802.15.4 Network. In *Computer Networks*, Vol. 51, pp. 2411-2449, 2007.

[19] R. Peng, M. Sun, and Y. Zou. ZigBee Routing Selection Strategy Based on Data Services and Energy-Balanced ZigBee Routing. In *Proc. IEEE Asia-Pacific Conference on Services Computing (APSCC)*, December 2006.

[20] A. Saeyoung, J. Cho, and S. An. Slotted Beacon Scheduling Using ZigBee Cskip Mechanism. In *Proc. International Conference on Sensor Technologies and Applications*, August 2008.

[21] Y.-C. Tseng and M.-S. Pan. Quick Convergecast in ZigBee Beacon-Enabled Tree-Based Wireless Sensor Networks. In *Computer Communications*, Vol. 31, No. 5, pp. 999-1011, Mar. 2008.

[22] X. Xu, D. Yuan, and J. Wan. An Enhanced Routing Protocol for ZigBee/IEEE 802.15.4 Wireless Networks. In *Proc. International Conference on Future Generation Communication and Networking*, December 2008.

[23] Y. Yamao and S. Takagishi. Time Shift Grouping Access in IEEE 802.15.4 MAC Beacon Mode for Layered-Tree Networks. In *Proc. IEEE Consumer Communications and Networking Conference (CCNC)*, January 2008.

Distributed Throughput Optimization for ZigBee Cluster-Tree Networks (Appendix)

———————————— ❖ ————————————

I THE PROPERTIES OF ALGORITHM PPR

In this appendix, we present the detailed proofs for the optimality of throughput maximization and the convergence of the PPR algorithm.

Lemma 1: During the execution of Algorithm PPR, if a vertex $u \in V$ is overflowing, u performs a push or a relabel operation; otherwise, a pull operation is performed on u.

Proof: For any edge $(u, v) \in E_f$, we have $h(u) \leq h(v) + 1$ since h is a height function of f. First, we consider the case where $h(u) = h(v) + 1$. If $c(u, v) = \infty$, we have $\hat{c}_f(v) > 0$ because $(u, v) \in E_f$. Then, Procedure 1 is satisfied, so v performs a pull operation on u. If $c(u, v) \neq \infty$, then $c_f(u, v) > 0$ because $(u, v) \in E_f$, and Procedure 2 is satisfied. Therefore, u performs a push operation on v. If neither of the two operations is performed, we have $h(u) < h(v) + 1$ such that Procedure 3 is satisfied, which leads u to relabel itself. □

Lemma 2: Whenever a vertex u performs a relabel operation on itself, its height $h(u)$ increases by at least 1.

Proof: Let us consider the time instant immediately before u is relabeled. At that point, we should have $h(u) \leq h(v)$ if $(u, v) \in E_f$ for any vertex v, which implies that $h(u) < 1 + \min\{h(v) \, | \forall(u, v) \in E_f\}$. Since $h(u) = 1 + \min\{h(v) \, | \forall(u, v) \in E_f\}$ immediately after the relabeling operation, the height is increased by at least 1. □

Lemma 3: If h is initialized as a height function, then it remains a height function throughout Algorithm PPR.

Proof: This lemma is proved by induction on the number of operations performed. Initially, h is a height function. We show that h remains a height function after one more operation. Let us consider a vertex u that performs the operation. If the operation is a relabeling operation, then, for all $v \in V$ where $(u, v) \in E_f$, we have $h(u) \leq h(v)$ before the operation and $h(u) \leq h(v) + 1$ afterwards. However, by Lemma 2, for all $w \in V$, where $(w, u) \in E_f$, we have $h(w) \leq h(u) + 1$ before the operation and $h(w) < h(u) + 1$ afterwards. Thus, a relabeling operation leaves h as the height function.

Next, we consider the case where u performs a pull or push operation on v. Because edge (u, v) is in E_f but (v, u) is not, after one of the two operations, (v, u) could be added to E_f or (u, v) could be removed from E_f. If (v, u) is added, we would have $h(v) = h(u) - 1$ before the operation. Since a pull or push operation never changes the height of vertices, we have $h(v) < h(u) + 1$ after the operation, so h remains a height function. On the other hand, if (u, v) is removed from E_f, h will remain a height function. This is because a height function is only associated with the edges in E_f, so its removal will not violate its function in the remainder of the algorithm. □

Lemma 4: Let f be a preflow in G and let h be a height function of f. Then, there will not be a path from the source s to the sink t in the residual network G_f.

Proof: We prove this lemma by contradiction. Suppose there exists a path $p = (v_0, v_1, \cdots, v_k)$ from the source s to the sink t in G_f, where $v_0 = s$ and $v_k = t$. Since p is a simple path, $k < |V|$. For $i = 0, 1, \cdots, k-1$, edge $(v_i, v_{i+1}) \in E_f$, and $h(v_i) \leq h(v_{i+1}) + 1$ since h is a height function. The summation of the k inequalities yields $h(s) \leq h(t) + k$. Note that $h(t) = 0$ because the sink t never performs any relabeling operations. Thus, we have $h(s) \leq k < |V|$, which contradicts the fact that $h(s) = |V|$ in a height function. □

Lemma 5: The *vertex-constraint maximum flow* problem can be reduced to the traditional maximum flow problem.

Proof: Given a vertex-constraint flow network $G = (V, E)$ with capacity $\hat{c}(v), \, \forall \, v \in V$, we construct a traditional flow network $G' = (V', E')$ as follows. First, every $u \in V$ is replaced by two vertices u_I and u_O connected by an edge (u_I, u_O), where u_I

is connected to the incoming edges of u and u_O is connected to the outgoing edges of u. That is, $V' = \{u_I, u_O | \ \forall u \in V\}$ and $E' = \{(u_I, u_O)| \ \forall u \in V\} \bigcup \{(v, u_I)| \ \forall (v, u) \in E\} \bigcup \{(u_O, v)| \ \forall (u, v) \in E\}$. Then, the capacity of the edge (u_I, u_O) is set as $\hat{c}(u)$, $\forall u \in V$, and any other edge in G' is assigned the same capacity as its corresponding edge in G. Therefore, finding a maximum flow f for G is equivalent to finding a maximum flow f for G', and vice versa. □

Theorem 1: (Optimality of Algorithm PPR) When Algorithm PPR terminates, the preflow f is a maximum flow from the source s to the sink t in G.

Proof: The initial procedure makes f a preflow. When the algorithm terminates, each vertex $u \in V - \{s, t\}$ must have $e(u) = 0$ by Lemmas 1 and 3, which means that the preflow f is a flow. By Lemma 4, there is no path from s to t in the residual network G_f; and by Lemma 5, there is no path from s to t in the residual network G'_f (of G' constructed based on G). The max-flow min-cut theorem states that a flow is a maximum flow if there is no path from the source to the sink in the residual network. Therefore, f is a maximum flow for G' and also a maximum flow for G, which proves that the proposed algorithm can achieve a maximum flow in a vertex-constraint flow network. □

Lemma 6: For any overflowing vertex u, there must exist a path from u to the source s in the residual network G_f.

Proof: Let U be the set of vertices achievable by u in G_f and let $\overline{U} = V - U$. We claim that $f(w, v) \leq 0$ for any pair of vertices $w \in \overline{U}$ and $v \in U$. The claim can be verified by contradiction as follows: if $f(w, v) > 0$, then $f(v, w) < 0$, which implies that there exists an edge $(v, w) \in E_f$ and, consequently, there is a path from u to w via v. It is a contradiction because w is in $\overline{U}$. Thus, we have

$$
\begin{aligned}
\sum_{v \in U} e(v) &= \sum_{w \in V, \ v \in U} f(w, v) \\
&= \sum_{w \in \overline{U}, \ v \in U} f(w, v) + \sum_{w, \ v \in U} f(w, v) \\
&= \sum_{w \in \overline{U}, \ v \in U} f(w, v) \\
&\leq 0.
\end{aligned}
$$

Note that $\sum_{u, \ v \in U} f(u, v) = 0$ because of the skew symmetry property of a flow. Now, suppose for the sake of contradiction that a path does not exist from u to s, i.e., $s \in \overline{U}$. Because any excess flow is non-negative for all vertices in $V - \{s\}$ and $U \subseteq V - \{s\}$, we have $e(v) = 0$ for all vertices $v \in U$. Since $u \in U$, we have $e(u) = 0$, which contradicts the fact that u is overflowing. Therefore, s must be in U. □

Lemma 7: During the execution of Algorithm PPR, $h(u) \leq 2|V| - 1$ always holds for any vertex $u \in V$.

Proof: This lemma holds for the source s and the sink t because, at the outset, $h(s) = |V|$ and $h(t) = 0$ and they do not change during the algorithm's execution. For any vertex $u \in V - \{s, t\}$, $h(u)$ is initialized as 0. We only consider the relabeling operation that may change the height of u. Each time that u is relabeled, u is overflowing; thus, according to Lemma 6, there must exist a path p from u to s in G_f. Let $p = (v_0, v_1, \cdots, v_k)$, where $v_0 = u$, $v_k = s$, and $k \leq |V| - 1$. For $i = 0, 1, \cdots, k - 1$, we have $(v_i, v_{i+1}) \in E_f$, and $h(v_i) \leq h(v_{i+1}) + 1$ according to Lemma 3. The summation of the k inequalities yields $h(u) = h(v_0) \leq h(v_k) + k \leq h(s) + (|V| - 1) = 2|V| - 1$. □

Lemma 8: During the execution of Algorithm PPR, any vertex performs at most $2|V| - 1$ relabel operations.

Proof: Neither the source nor the sink performs relabeling operations. For any vertex $u \in V - \{s, t\}$, $h(u)$ is set at 0 initially; then, by Lemma 7, it is increased to at most $2|V| - 1$. This implies that u performs at most $2|V| - 1$ relabeling operations based on Lemma 2. □

Lemma 9: During the execution of Algorithm PPR, at least one relabel operation is performed during each pass.

Proof: For any pass during the algorithm's execution, suppose for the sake of contradiction that no vertex performs any relabeling operation; that is, the vertices heights remain the same during the pass. Thus, there must be some overflowing vertices before the algorithm terminates (see Line 3 of Algorithm 1). Let us examine the shortest vertex u that is overflowing. When implementing a PULL-PUSH-RELABEL operation, u first performs pull operations on all of its adjacent vertices if applicable. Then, because u is not higher than any overflowing vertex, it cannot be pulled upwards by any other vertex due to the difference in height (see Line 3 of Procedure 4). Thus, u performs push operations on all of its adjacent vertices if applicable and then relabels itself (see Line 6 of Procedure 4). Note that pull operations can not be performed on u by other vertices and u can not perform push operations at this point. Therefore, by Lemma 1, the relabeling operation must be applicable and performed, which leads to a contradiction. □

Theorem 2: (Convergence of Algorithm PPR) Algorithm PPR always terminates within $2|V|^2$ passes.

Proof: By Lemma 8, each vertex $u \in V - \{s, t\}$ performs at most $2|V| - 1$ relabeling operations. Thus, the total number of relabeling operations performed is at most $(2|V| - 1)(|V| - 2) < 2|V|^2$. By Lemma 9,

at least one relabeling operation is performed during each pass. Therefore, there are at most $2|V|^2$ passes. $\square$

II IMPLEMENTATION OF ADOPTIVE-PARENT-BASED FRAMEWORK

ZigBee supports three kinds of network topology, namely, star, cluster-tree, and mesh topologies. In a star network, multiple ZigBee end devices connect directly to the ZigBee coordinator. For cluster-tree and mesh networks, communications can be conducted in a multi-hop fashion through ZigBee routers. In a cluster-tree network, each ZigBee router with its surrounding devices is regarded as a respective cluster, and each cluster operates individually as a star network. Compared to mesh networking, the cluster-tree is especially suitable for low-power WSNs because it supports the superframe structure, which handles the power saving operation in the IEEE 802.15.4 protocol. In a ZigBee cluster-tree network, the superframe structure is based on the concept of time-division duplex. More specifically, each router selects an active portion as its outgoing superframe, and the active portion selected by its parent acts as the router's incoming superframe. The router is expected to transmit a beacon to its child routers via its outgoing superframe while receiving a beacon from its parent router in its incoming superframe. The selection of a beacon starting slot by each router is called beacon scheduling.

A very light-weight routing protocol without routing table maintenance is also supported in cluster-tree networks. Before forming a cluster-tree topology, three system configuration parameters must be determined: the maximum number of children of a router (Cm), the maximum number of child routers of a router (Rm), and the depth of the network (Lm). Under the topology configuration, an appropriate distributed address assignment policy is executed to enable the light-weight tree routing. With the three parameters, a parent device can determine unique 16-bit network addresses for its child devices locally. Utilizing the pre-allocation design concept, the light-weight tree routing protocol is the reverse of the distributed address assignment. Therefore, the routing mechanism in a ZigBee cluster-tree network can be implemented by a simple calculation without any routing overhead.

Although the ZigBee cluster tree provides an effective solution for low-power and low-cost wireless sensor networking, the rigidness of the topology makes it vulnerable to link failures. It also prevents the use of many potential transmission paths, which means that a considerable amount of bandwidth can not be utilized. To resolve such problems, we propose an adoptive-parent-based framework for a ZigBee cluster-tree network. In this appendix, we describe the implementation of the proposed adoptive-parent-based framework. First, we discuss the method used to determine adoptive parents locally. Then, we demonstrate that Algorithm PPR is compatible with ZigBee networks. Because of the low-power and low-cost characteristics of ZigBee, the proposed framework can be implemented without incurring any extra message exchange.

II.1 Choosing Adoptive Parents

In the proposed adoptive-parent-based framework, routers are chosen as adoptive parents based on three criteria: *GTS capacity*, *node depth*, and *path similarity*. The *GTS capacity* indicates a router's available GTS slots. This criterion is to ensure that the allocation of additional bandwidth by adoptive parents for a region of interest will not affect the QoS of on-going data deliveries. The *node depth* represents the length of the uplink path from a router to the sink (i.e., the PAN coordinator). The *path similarity* is introduced specifically for ZigBee cluster-tree networks to indicate the degree of overlap between the uplink paths from the original parent router and from an adoptive-parent router candidate to the sink. A small amount of path similarity or overlap implies that the candidate can provide more bandwidth if extra traffic is generated suddenly. Before defining the *path similarity* criterion, we introduce a router's *ancestor list*, i.e., its ancestor routers. Let ψ_i denote the ancestor list of router i, and let $d(k)$ denote the depth of router k. The path similarity s_{ij} of routers i and j is defined by

$$s_{ij} = max\{d(k)|k \in \psi_i, \psi_j\}.$$

In fact, s_{ij} is the depth of the nearest common ancestor of routers i and j. It also represents the number of common ancestors between routers i and j along their uplink paths to the sink.

The adoptive-parent router candidates must satisfy the following two requirements:

1) They must have available GTS capacity;
2) Their depth must be lower than that of the original parent router.

Note that the depth requirement prevents the new path from being longer than the original path so that unexpectedly longer latency can not occur. Then, an adoptive parent is chosen from the candidates based on the following two priorities:

1) The lowest depth;
2) The smallest path similarity with the original parent.

Next, we describe the implementation of the adoptive-parent determination framework in ZigBee. The *GTS capacity* and *node depth* can be accessed directly because the information is attached to the beacon frames. In the remainder of this appendix, we focus on how the *path similarity* is derived. The process involves comparing the ancestor lists of the
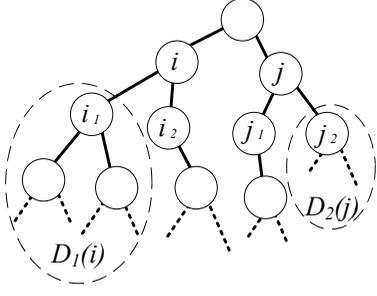
Fig. 1. An example of i_n and $D_n(i)$

original parent router and the adoptive-parent router candidates. As mentioned previously, the ZigBee cluster tree utilizes three configuration parameters, Cm, Rm and Lm to form the topology. With the three parameters, a parent router can also determine the unique network addresses of its child devices. Each router of depth d calculates its $Cskip(d)$ to determine the number of network addresses allocated to each router-capable child device. The function $Cskip(d)$ is calculated as follows:

$$Cskip(d) = \begin{cases} 1 + Cm \cdot (Lm - d - 1), & \text{if } Rm = 1, \\ 1 + \dfrac{Cm \cdot (1 - Rm^{Lm-d-1})}{1 - Rm}, & \text{Otherwise.} \end{cases}$$
(1)

Then, a distributed address assignment can be executed based on $Cskip(d)$. For a router i, i_n denotes its n-th router-capable child, and $D_n(i)$ denotes the set of descendant routers that reside in the branch of i_n. Figure 1 shows an example of i_n and $D_n(i)$. Given the network address $a(i)$ for i with the depth d, the network address of i_n defined in the ZigBee specification is

$$a(i_n) = a(i) + 1 + (n-1)Cskip(d). \quad (2)$$

Consider a router i with depth d. Based on (2), the network address range for any router k in $D_n(i)$ can be derived as follows.

Lemma 10:
$a(i) + 1 + (n-1)Cskip(d) \le a(k) < a(i) + 1 + n \cdot Cskip(d)$, $\forall k \in D_n(i)$.

Proof: First, we prove the left part, i.e., $a(i) + 1 + (n-1)Cskip(d) \le a(k)$. Let $d(k)$ denote the depth of the router k. It is obvious that $a(y) > a(z)$ if $d(y) > d(z)$, where $y, z \in D_n(i)$. For all routers in $D_n(i)$, the network address $a(k)$ is the minimum if $d(k) = d + 1$. Since k is the n-th router-capable child of i, by (2), we have $a(k) = a(i) + 1 + (n-1) \cdot Cskip(d)$. For the right part, i.e., $a(k) < a(i) + 1 + n \cdot Cskip(d)$, since $Cskip(d)$ decides the number of addresses allocated to every router-capable child device of i, $A(k)$ has the maximum value $a(k) < a(i) + 1 + n \cdot Cskip(d)$. $\square$

For a descendant $k \in D_n(i)$, the address relation between i and its n-th child router is presented as follows.

Lemma 11:

$$a(i_n) = a(i) + 1 + \left\lfloor \frac{a(k) - a(i) - 1}{Cskip(d)} \right\rfloor \cdot Cskip(d), \forall k \in D_n(i).$$

Proof: By rewriting Lemma 10, we have $n - 1 \le [a(k) - a(i) - 1]/Cskip(d) < n$. Therefore, $n = \lfloor [a(k) - a(i) - 1]/Cskip(d) \rfloor + 1$. Moreover, by (2), we have $a(i_n) = a(i) + 1 + (n-1)Cskip(d) = a(i) + 1 + \left\lfloor \dfrac{a(k) - a(i) - 1}{Cskip(d)} \right\rfloor \cdot Cskip(d)$. $\square$

Theorem 3: For any router k, its ancestor list ψ_k can be derived from the address $a(k)$.

Proof: In ZigBee, the address of the ZigBee coordinator is known by all devices in the network. Given that i is the coordinator and $k \in D_n(i)$, $a(i_n)$ can be derived by Lemma 11. By replacing i with i_n in a series of steps, the ancestor list ψ_k of the router k can be determined recursively. $\square$

When the framework is implemented in ZigBee networks, each router attaches its network address to the beacon frames. Consequently, devices can overhear the beacon frames of surrounding routers and compute the corresponding ancestor lists. Then, they can derive the *path similarity* of the adoptive-parent candidates, such that no extra message exchange occurs.

II.2 The Implementation of Algorithm PPR

Next, we discuss the implementation of Algorithm PPR in ZigBee cluster-tree networks.

- Implicit message exchange:
  In the proposed framework, the communications between routers utilize the GTS mechanism to ensure transmission reliability. The amount of flow on each link can be represented by the allocated GTS slots. Based on the ZigBee specification, the GTS allocation information is embedded in beacon frames. The two parameters used in Algorithm PPR, i.e., the height and excess flow, are attached in the reserved field of the beacon frames. Therefore, the information exchange needed by Algorithm PPR can be achieved implicitly by beacon frames transmitting/receiving between neighboring routers.

- Tracking multiple beacon frames:
  In the adoptive-parent-based framework, a router tracks multiple beacon frames from its original parent and adoptive-parent candidates to acquire the information used in Algorithm PPR. Since ZigBee regulates the outgoing superframes for the routers so that direct/indirect interference is blocked, the router can track multiple beacon frames.

- PUSH/PULL/RELABEL operations:

When a device needs to perform a PULL operation, it must first ensure that it has sufficient GTS bandwidth. Then the pulling device selects the devices that satisfy the following pull conditions: (1) the devices are the (adoptive) children of the pulling device; (2) the devices are overflowing; and (3) the height of the selected devices is larger than that of the pulling device by one. The pulling device can then utilize the GTS allocation information of the selected devices to determine the amount of flow it can pull. Next, the pulling device issues new GTS allocation embedded in a beacon frame to perform the PULL operation implicitly.

For the PUSH operation, the pushing device first ensures that it is overflowing. Similar to the PULL operation, it selects those devices that satisfy the following push conditions: (1) the devices are the (adoptive) children of the pushing device; and (2) the height of the selected devices is lower than that of the pushing device by 1. The pushing device then utilizes the GTS allocation of the selected devices to determine the amount of flow it should pull and issues a beacon frame to broadcast the new GTS allocation.

For the RELABEL operation, the relabeling device must first ensure that it is overflowing, and its height must be lower than that of the devices which are its (adoptive) parents or (adoptive) children. The relabeling device can then update its height based on its (adoptive) parents and (adoptive) children.

- Parallel set:
  In the implementation, each router device receives the beacon frames of its neighbors, and executes the PULL-PUSH-RELABEL procedure before issuing a new beacon frame. The devices with the same start time for outgoing superframes implicitly form a parallel set used in Algorithm PPR, since they do not have direct links or common neighbors according to the beacon scheduling rule. Therefore, the devices with the same start time for outgoing superframes can perform the operations simultaneously.

- Pass:
  Recall that a pass represents an iteration in which every vertex performs the PULL-PUSH-RELABEL procedure once. Since each device executes the procedure before issuing a beacon frame, and the beacon interval adopted in all devices is the same, a pass is equal to a beacon interval. Consequently, Algorithm PPR can be implemented to operate in a distributed manner by utilizing the mechanisms of ZigBee networks.
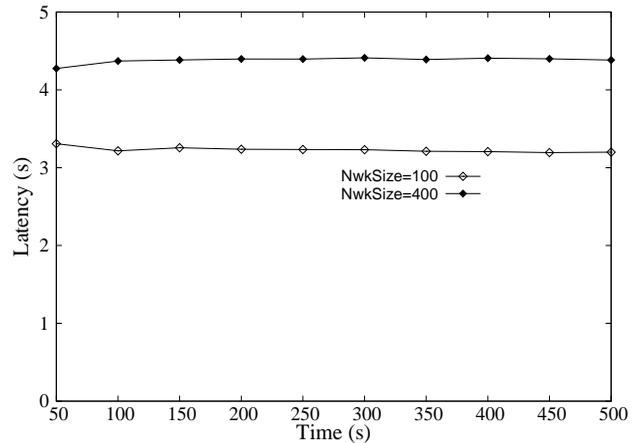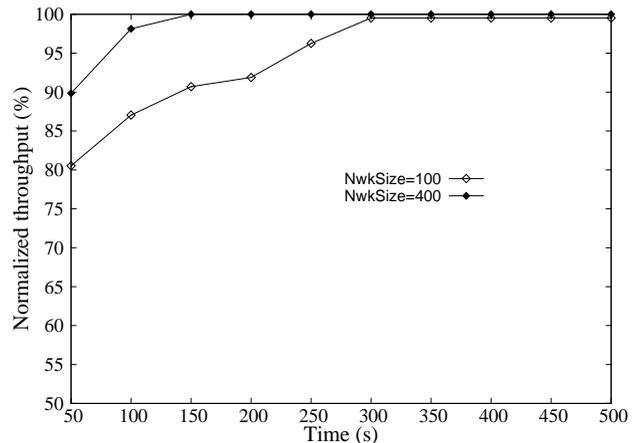
Fig. 2. The convergence of Algorithm PPR: (a) latency; (b) throughput.

III ADDITIONAL SIMULATIONS AND NUMERICAL RESULTS

Figure 2 shows the convergence performance of the proposed Algorithm PPR. Without loss of generality, routers are placed in a grid manner in this experiment, and the traffic load of the region of interest is set at 4 packets per second. Figure 2 (a) shows the transmission latency. We observe that the latency is not affected during the convergence period of the algorithm. Figure 2 (b) shows that the throughput can reach 100% in less than 5 minutes. The throughput in a larger network increases more quickly than that in a smaller network. Since more adoptive parents can be selected in a large-scale network, the increased bandwidth benefits the operations of Algorithm PPR.

To further investigate the effects of the buffer size on the system performance, we consider buffers ranging in size from 10 packets to 100 packets. In addition, the traffic loads of the region of interest are set at 4 and 8 packets per second. Figure 3(a) shows how the transmission latency changes with the buffer size for a network of 100 routers. We observe that the buffer
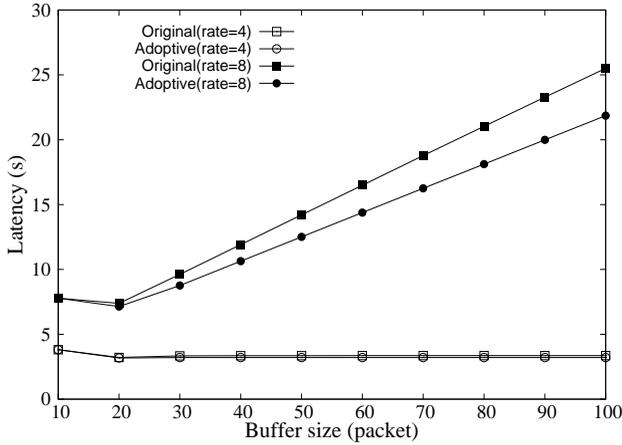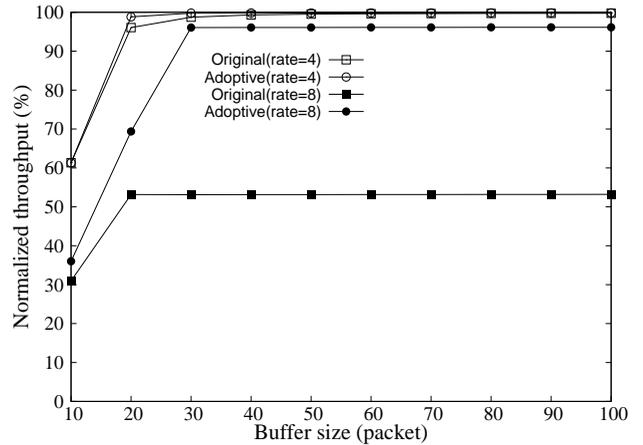
(a) NwkSize=100
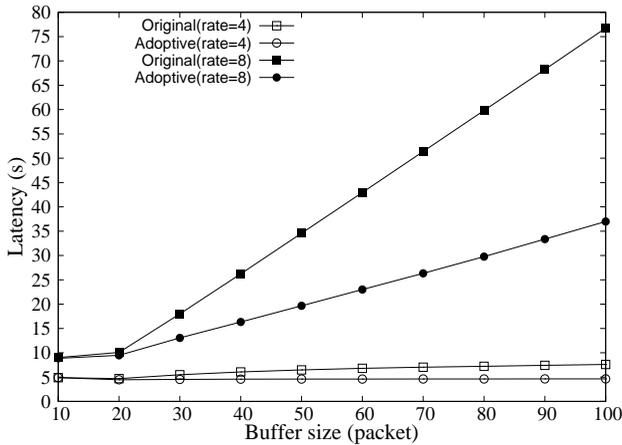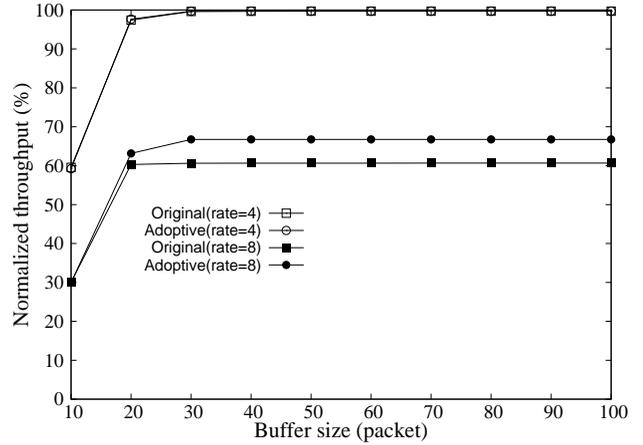
(b) NwkSize=400

Fig. 3. The latency versus various buffer sizes: (a) a network of 100 routers; (b) a network of 400 routers.

(a) NwkSize=100

(b) NwkSize=400

Fig. 4. The throughput versus various buffer sizes: (a) a network of 100 routers; (b) a network of 400 routers.

size does not affect the latency significantly when the traffic load is 4 packets per second, since the bandwidth is sufficient for this size of load. On the other hand, the latency grows as the buffer size increases when the traffic load is 8 packets per second, since the bandwidth is not sufficient for the heavier traffic load. As shown in Figure 3(b), when the number of routers is 400, the proposed framework outperforms the original approach by a substantial margin. This is because of the extra bandwidth provided by the additional adoptive parents.

Figure 4 evaluates the throughput versus the buffer size. For a network with 100 routers, Figure 4(a) shows that the achieved throughput is only 60 − 70% when the traffic load is 8 packets per second. This is because the bandwidth is not sufficient for the high traffic load. When the network is larger, Figure 4(b) shows that the proposed approach can achieve more than 95% throughput when the buffer size is greater than 30 packets. This is almost a 200% improvement over the original approach. In this figure, we notice that the normalized throughput increases rapidly when the

buffer size increases from 10 to 20. The reason for this phenomenon is that a small buffer size leads to large amount of packet loss due to queuing drops. In this case, the increasing of the buffer size significantly benefits the normalized throughput performance. When the buffer size is larger than a threshold (e.g., 20 packets), the effect of the buffer size on normalized throughput is insignificant.