Journal of Network and Computer Applications 54 (2015) 20–32

Contents lists available at ScienceDirect

Journal of Network and Computer Applications

journal homepage: www.elsevier.com/locate/jnca

An efficient distributed max-flow algorithm for Wireless Sensor Networks

Saman Homayounnejad [a], Alireza Bagheri [a,b,*]

[a] Department of Computer Engineering and Information Technology, Amirkabir University of Technology, Tehran, Iran
[b] Department of Computer Engineering, Tehran North Branch, Islamic Azad University, Tehran, Iran

ARTICLE INFO

Article history:
Received 6 August 2014
Received in revised form
31 December 2014
Accepted 8 April 2015
Available online 22 April 2015

Keywords:
Wireless Sensor Networks
Max-flow problem
Adaptive algorithm
Distributed algorithm
Asynchronous algorithm
Near-optimal flow

ABSTRACT

The max-flow problem (MFP) is one of the most explored problems in the area of combinatorial optimization and it has a wide variety of potential applications in computer networks, especially in Wireless Sensor Networks (WSNs). In this paper, we propose a WSN-specific solution to MFP based on the well-known push-relabel method. In our solution we provide several techniques and heuristics to implement asynchronous communication, reduce message complexity, support adaptability, and satisfy soft real-time requirements. Because these heuristics are independent, we can adjust the algorithm by applying some heuristics and ignoring the others, according to the application requirements. Both theoretical analysis and simulation results show that the proposed algorithm makes a significant improvement in the case of message and time complexities in comparison with the best existing algorithms.

1. Introduction

Due to the recent advances in pervasive and ubiquitous computing, manufacturing of low-cost, tiny and unattended sensors has become feasible. These sensors are capable of measuring specific values from the environmental events and then transmit these values to a specified sink. Wireless Sensor Network (WSN) is a network consisting of a large number of such autonomous sensor nodes which cooperate together to reach some high-level purposes such as surveillance, tracking and monitoring.

The max-flow problem (MFP) is a well-known combinatorial optimization problem in which we wish to send as much flow as possible in a capacitated network between a source node s and a sink node t, without exceeding the capacity of any link. MFP arises in a wide variety of applications and in several forms, not only in the mathematics, management and engineering but also in the specific applications in WSN literature (Song et al., 2009; Bogliolo et al., 2011; Kalpakis, 2010; Al-Kofahi and Kamal, 2009; Ohara et al., 2009; Valero et al., 2012; Dandapat et al., 2010; Yantao et al., 2006).

Several algorithms have been introduced to find the max-flow value which most of them can be categorized into two main categories:

- *Augmenting path algorithms* proceed by identifying augmenting paths which are paths capable of increasing the total net-flow. Then they augment flows on these paths until the network contains no more augmenting path.
- *Push-relabel algorithms*, in which source saturates its outgoing capacities that causes to come out excess flows at neighbors of the source. Push-relabel algorithms gradually mitigate the excess flows by sending some flows from the excess nodes toward the sink, if it is possible, or send back toward the source.

Early proposed solutions for MFP were based on the augmenting path method and the concept of residual networks. The general algorithm of Ford–Fulkerson finds max-flow in $O(mnU)$ time in which U is the maximum link capacity, m depicts the number of links and n is the number of nodes (Ford and Fulkerson, 1956). In Edmonds and Karp (1972) an algorithm is proposed based on this method which terminates in $O(nm^2)$ time. A blocking flow algorithm is designed which finds the max-flow in $O(mn^2)$ time (Dinic, 1973). Since then various algorithms have been proposed based on this method but to our knowledge none of them are distributed and so they are not implementable in WSNs.

On the other hand push-relabel algorithms work locally and they have potential to become a distributed and asynchronous algorithm by some modifications (Goldberg and Tarjan, 1988). Several solutions have been published based on the push-relabel

* Corresponding author.
E-mail address: ar_bagheri@aut.ac.ir (A. Bagheri).

http://dx.doi.org/10.1016/j.jnca.2015.04.004
1084-8045/

method. Authors of Ghosh et al. (1997) have presented a self-stabilizing algorithm which finds the max-flow in almost n^2 messages in the average-case, but their method produces exponential messages in the worst-case. In Pham et al. (2005), based on the push-relabel method (Goldberg and Tarjan, 1988), a two-phase distributed algorithm has been proposed, which is asynchronous and adaptive. This algorithm finds the max-flow in $O(n^2)$ time, using $O(n^2m)$ messages, and leverages depth-first-search that is not efficient in the large-scale networks. In Avestimehr et al. (2011) the channel model and the way of deriving link capacities in wireless networks are investigated.

Recently, Orlin (2013) proposed the fastest strongly polynomial time algorithm which terminates in $O(mn)$ time and even $O(n^2/\log n)$ for the sparse networks, resolving a long standing conjecture. This algorithm which does not place in the above categories beats the fastest existing available algorithms in the literature (Goldberg and Rao, 1998; Cheriyan et al., 1996; Cherkassky and Goldberg, 1997; Chandran and Hochbaum, 2009) but it is a centralized approach like most of the other existing solutions.

There are several parallel algorithms (Shiloach and Vishkin, 1982; Anderson and Setubal, 1992; Bader and Sachdeva, 2005; Caragea and Vishkin, 2011) as well as several GPU max-flow implementations (Hussein et al., 2007; Vineet and Narayanan, 2008), unfortunately none of them could be implemented in distributed wireless ad-hoc and sensor networks. To our knowledge, existing asynchronous distributed solutions which have the potential of being implemented in WSNs are based on push-relabel algorithm (Goldberg and Tarjan, 1988; Pham et al., 2005). Therefore, the proposed algorithm is the first WSN-specific algorithm to solve MFP that is asynchronous, lightweight, adaptive and incremental. We have proposed some raw ideas previously in Homayounnejad et al. (2011a, 2011b). In this paper, we enhance and integrate them, and give more insight into the solution by transparent examples and scenarios. Moreover, we prove the correctness of some heuristics, and we analyze the complexities of some other proposed heuristics.

WSNs are distributed, dynamic, autonomous and resource-constrained networks, hence the proposed heuristics should consider these characteristics. Some applications of max-flow problem in computer networks, especially in WSNs, are life-time maximization (Song et al., 2009), routing (Bogliolo et al., 2011; Ohara et al., 2009; Soorki and Rostami, 2014), aggregation (Kalpakis, 2010), network coding (Al-Kofahi and Kamal, 2009), data redistribution in sink failures (Valero et al., 2012), bandwidth allocation (Dandapat et al., 2010), traffic management (Yantao et al., 2006), etc. As one special case, consider the following routing applications:

1. In the multipath routing, maximizing the number of edge-disjoint paths from some sources to a target could be cast to the max-flow problem in the unit-capacitated network (Ahuja et al., 2014).
2. In energy sustainable sensor networks, as long as the average workload at each node can be sustained by the average power it takes from the environment, the node can keep working for an unlimited amount of time. The power-constrained workload optimization can be modeled and solved by the max-flow problem (Bogliolo et al., 2011).
3. We hope that new QoS-aware routing protocols could be designed based on the heuristics proposed here. Because it tries to maximize the throughput of the network and so it can be combined with the effective-bandwidth theory (Wu and Negi, 2006) and statistical methods (Abdrabou and Zhuang, 2009) to propose a QoS-aware protocol which can take into account delay, reliability and bandwidth, simultaneously.

Despite the various applications of max-flow in WSNs, to our knowledge there is not any dedicated work for implementing a max-flow solution in WSNs. The proposed algorithm and its corresponding heuristics try to fill this large gap and to open a new window to WSN solutions.

The rest of the paper is organized as follows. Section 2 describes the fundamental concepts and the network model. In Section 3, the proposed algorithm and its heuristics are described in detail. The theoretical complexity analysis and the correctness proofs are included in Section 4, and simulation results are presented in Section 5. Finally, Section 6 concludes the paper.

2. The notation and assumptions

A WSN is modeled as a network $G = (V, E)$ whose nodes V are sensors and E is the set of the full-duplex directed communication links. Each sensor of WSN has a unique identifier and has at least one transmitter and one receiver. We assume that the effective transceiving distance of all nodes is equal. Two nodes are neighbors and have a link between them if they are in the transceiving range of each other. Moreover, we assume that a mechanism exists for neighbor discovering and an underlying protocol to deliver a packet from a node to its one-hop neighbor, properly. Every link $(i,j) \in E$ is augmented with a nonnegative capacity c_{ij} which restricts the flow on this link (f_{ij}). Solving MFP is to find the max-flow from the source node s to the sink node t that satisfies some constraints, formulated as follows:

$$\text{maximize } |f| \tag{1}$$

subject to

$$\sum f_{ij} - \sum f_{ji} = \begin{cases} |f| & \text{for } i = s \\ 0 & \text{for all } i \in V - \{s, t\} \\ -|f| & \text{for } i = t \end{cases} \quad \text{(Mass balance constraint)} \tag{1 – a}$$

$$0 \le f_{ij} \le c_{ij} \quad \text{for all } (i,j) \in E \quad \text{(Capacity constraint)} \tag{1 – b}$$

The value of a flow f is defined as $|f| = \sum_{v \in V} f(s, v)$ that is, the total flow out of the source. For example, as one potential application, the QoS-aware routing could be proposed based on MFP. Bandwidth of the links can be modeled by link capacities, and therefore the max-flow is maximum utilization of wireless bandwidths.

Definition 1 (Residual Network (G_f)). *Residual capacity* is the maximum additional flow that can be sent from node i to node j using the links (i, j) and (j, i) and could be computed by means of capacities and flow values as $residual[u, v] = c[u, v] - f[u, v]$. The *residual network* is the network consisting links that have positive residual capacities.

Definition 2 (Height function (h)). A function $h : V \to N$ is a height function if $h[s] = n$, $h[t] = 0$, and $h[u] \le h[v] + 1$ for every residual link $(u, v) \in G_f$. Algorithm pushes flow downhill, which is from the higher node to the lower node.

Definition 3 (Generic push-relabel algorithm (GPR)). Algorithm 1 is the sequential push-relabel (Goldberg and Tarjan, 1988), which uses push and relabel operations and nodes could maintain a positive excess value during the running of the algorithm. In this algorithm, the push operation tries to send the excess flow to lower-height neighbors, if any exists. Otherwise, the relabel operation would be invoked to elevate the height and to make the push operation applicable. Throughout this paper, we

will refer to the distributed version of this method as the generic push-relabel algorithm (GPR).

Algorithm 1. Generic Push-Relabel (Sequential version).

Input: A network $G=(V, E)$ stored as an adjacency list.
Output: Max-flow which at termination is function $f: E \rightarrow N$, $f[i, j] = c[i, j] - residual[i, j]$. Therefore, the solution which is flow values could be stored as a number property of the links.
$h[source] = n$;
$h[sink] = 0$;
for (*all* $u \in V - \{s, t\}$)
  $h[v] = 0$;
for (*all* $(u, v) \in E$)
  $residual[u, v] = c[u, v]$;
for (*all links e outgoing from source*)
  saturate e;
while ($\exists v \in V$ *such that v is an active node*)
  if ($\exists (v, u) \in G_f$ *such that* (v, u) *is downhill*)
    push $min\{excess[v], residual[v, u]\}$ *units from v to u*;
    update $residual[u, v]$ *and* $residual[v, u]$ *accordingly*;
  \− ***else***
  $h[v] = 1 + min\{h[u] | residual[v, u] > 0\}$;

3. The proposed algorithm

The proposed algorithm is based on the distributed version of the push-relabel method (GPR) (Goldberg and Tarjan, 1988) and two-phase push-relable (2PPR) (Pham et al., 2005) which is itself based on GPR. The main difference between GPR and 2PPR is in their height initialization. In GPR all initial heights are zero whereas in 2PPR a distributed breadth-first-search method is utilized to compute the hop-distances from the sink. The heights are initialized with the computed hop-distances.

We propose the heuristics gradually. First we develop an asynchronous version which is exactly GPR. Second, we use several heuristics to reduce the number of control messages. Next we propose an adaptive improvement and finally we complete the algorithm by introducing an incremental version.

3.1. The asynchronous algorithm

The proposed algorithm uses three control messages to implement the push-relabel method. Whenever a sensor node wants to push a flow value to one of its neighbors, it produces a PUSH message. On the other hand, when a node increases its height via the relabel operation, informs its direct neighbors by means of HEIGHT messages. The basic idea of decentralizing the push-relabel algorithm is proposed by Pham et al. (2005).

Algorithm 2 shows the main body of the proposed algorithm which runs in the nodes. First, each node initializes its local variables and enters the listening state in which the node waits for incoming messages from the wireless channel. The only exception is source node that waits until the initialization operations finish in the other nodes; afterward it saturates every outgoing link.

Implementing the following pseudo-codes is simple. Because they only need local and one-hop information. Most important local information stored in node v are role of the node ($role_v$), current excess flow ($excess_v$), location and height (h_v). One-hop information is also simple values stored in the neighbor table. One-hop information for a typical neighbor u includes residual capacity ($residual_v[u]$), flow value ($f_u[u]$), and height ($h_v[u]$). Communicating messages are also simple messages in which first parameter is the message type and later parameters are values related to the one-hop destination.

Algorithm 2. The main body of the proposed algorithm which executes in sensor nodes – *main*().

Input: Local information of caller node, such as *role* and *excess* as well as one-hop information such as flow values and residual capacities for each neighbor.
Output: Max-flow or near-optimal flow which at termination is a function
$f: E \rightarrow N$, $f[i, j] = c[i, j] - residual[i, j]$.
initialize();
if ($role_v = SOURCE$)
  $excess_v = \sum_{u \in neighbors_v} residual_v[u]$;
  push();
while (*true*)
  listen();

Algorithm 3 shows the initialization algorithm in which height of the node and heights of its neighbors are set to their initial values. Then the source begins sending the maximum possible flow to its neighbors via the push operation.

Algorithm 3. Initialize operation – *initialize*() and $init-height(k)$.

Input: The local information of node v which are the capacities of incoming links $c[v, u]$, $role_v \in \{SOURCE, SINK, RELAY\}$ and the total number of the nodes ($n = |V|$).
Output: The node v computes the residual capacities and heights of its neighbors and this node becomes ready to run push and relabel operations.
$excess_v = 0$;
$h_v = init-height(role_v)$;
for (*all* $u \in neighbors$)
  \− ***if*** ((u, v) *is an incoming link into v in G*)
    $residual_v[u] = 0$;
  \− ***else if*** ((v, u) *is an outgoing link from v in G*)
    $residual_v[u] = c[v, u]$;
  $h_v[u] = init-height(role_u)$;
Input: $role_k \in \{SOURCE, SINK, RELAY\}$ and the total number of the nodes ($n = |V|$).
Output: An integer that is initial height of node k.
$init-height\ (role_k)$:
  \− ***if*** ($role_k = SOURCE$)
    return n; // later by SS heuristic the source height reduces to $3\sqrt{n}$.
else
    return 0; // later by GN heuristic this value will be assigned cleverly.

By definition of the height function h in the push-relabel method, initial height of the source is set to the number of network nodes (n) and other heights are initialized to zero. Later in the lightweight algorithm, we use more fashionable values for initial heights.

While a node wants to transmit some excess to its neighbors, it invokes the push operation depicted in Algorithm 4. This operation applies when u is active, $residual[u, v] > 0$, and $h[u] = h[v] + 1$.

Algorithm 4. The push operation – *push*().

Input: Local variables such as the residual capacities of the neighbors, role, height and excess. We assume that the algorithm runs at node v.
Output: A PUSH message carrying a flow value.

```
if (role_v ≠ SINK)
  while (excess_v > 0) && (∃u ∈ neighbors_v such that (residual_v[u] > 0) && (h_v[u] < h_v))
    δ = min{excess_v, residual_v[u]};
    excess_v = excess_v − δ;
    residual_v[u] = residual_v[u] − δ;
    send (PUSH, δ) to u;
```

The relabel operation applies on active node u if $h[u] \leq h[v]$ for all links $(u, v) \in G_f$. In other words, we can relabel an active node u, if u is not high-enough to push away its excess. As depicted in Algorithm 5, this operation only increases the height values so the excess and the residual capacities remain unchanged. We postpone the description of the check-canceling method to the next section but at the moment consider it as no operation.

Algorithm 5. The relabel operation – *relabel*().

Input: Local variables such as the residual capacities of the neighbors, role and height.
Output: Some HEIGHT messages carrying the new height value of the caller node.

```
if ( role_v ≠ SOURCE ) && ( role_v ≠ SINK )
  check−canceling();
  h = min_{u ∈ neighbors_v, residual_v[u] > 0}{h_v[u]} + 1;
  h_v = h;
  \− tfor (all u ∈ neighbours_v)
    Send (HEIGHT, h_v) to u;
  push();
```

Algorithm 6 shows the reaction of nodes upon receiving PUSH and HEIGHT messages. When a node v receives a HEIGHT message form node w, updates the height of w in its neighbors table ($h_v[w]$). On the other hand, when one node receives a PUSH message, accepts the excess flow if it can, otherwise, it sends back a REFUSE message to the sender. After that, the sender rollbacks the push impact and recovers the previous state. For example to illustrate the importance of REFUSE message in the algorithm, consider the scenario shown in Fig. 1. Node A with $h_A = 2$ sends a PUSH message to node B with $h_B = 1$. Node B before receiving the message, via relabel operation increases its height to 3 and sends HEIGHT message to node A. In this situation node A has performed an invalid push to node B that is from a lower-height node to the higher one. Fortunately, this turns out not to be a serious problem. To handle this inconsistency, REFUSE message emerges. In the abovementioned scenario, when node B detects an invalid push, produces a REFUSE message, and sends back this message to the pusher node A.

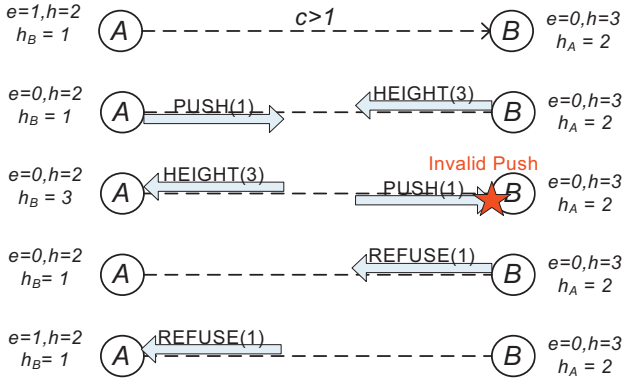

Fig. 1. Invalid push and REFUSE message.

Algorithm 6. Nodes reaction upon receiving messages from the network – *listen*().

Input: An incoming message from the channel with w as its sender and δ as its data. Moreover, the node uses local variables such as residual capacities, excess value and the height.
Output: Depends on the incoming message and the state of the node. It might be an update in local variables or calling other operations to propagate data through new messages.

```
On receiving PUSH message :
  \− tif (h_v[w] > h_v)
    residual_v[w] = residual_v[w] + δ;
    excess_v = excess_v + δ;
      \− tif (role_v ≠ SINK)
        \− tpush();
        \− tif (excess_v > 0)
          \− trelabel();
  \− telse
    send (REFUSE, δ) to w;
On receiving REFUSE message :
  excess_v = excess_v + δ;
  residual_v[w] = residual_v[w] + δ;
    push);
    \− tif (excess_v > 0)
      \− trelabel();
On receiving HEIGHT message :
  h_v[w] = δ;
```

3.2. The lightweight algorithm

The abovementioned asynchronous algorithm is the asynchronous version of the generic push-relabel (GPR) algorithm which finds the max-flow value with $O(n^3)$ messages but an algorithm with a cubic message complexity is too costly to be practical in WSNs, except the smallest networks. In the case of large-scale wireless networks like WSNs, finding near-optimal flow (NOF) with linear or even quadratic message overhead is much better than finding the precise max-flow with cubic order of messages. Hence, it is invaluable to trade a little accuracy over a significant number of messages.

The lightweight algorithm is a combination of six independent heuristics that some of them could be disabled according to the network conditions and limitations. Heuristics are as follows:

- *Gradient-Network* (*GN*)*:* Initializing height of the nodes to make an initial gradient toward the sink.
- *Smart-Push* (*SP*)*:* Pushing into more appropriate nodes to accelerate the algorithm convergence and finding shorter flow paths in the network.
- *Stunt-Source* (*SS*)*:* Reducing the source height to restrain the network from overwhelming by the trapping flow values.
- *Temporal Canceling* (*TC*)*:* Canceling the aged flows and sending them back to the source.
- *Spatial Canceling* (*SC*)*:* Canceling the faraway flows.
- *Hop-Counting Canceling* (*HC*): Canceling the much oscillated flows.

GN and SP do not impair the accuracy but other heuristics may affect the optimality. This algorithm could be adjusted by applying or not applying these heuristics. For example, if the location information in the sensor nodes is not precise, it would be better to use TC and HC rather than SC. In Section 5, we will show that the inaccuracy is very small and negligible.

3.2.1. Gradient Network

In the GPR algorithm all relay nodes initialize their heights with zero whereas in 2PPR initial height of a node is its hop-distance from the sink node. These initial heights make a gradient in nodes toward the sink which accelerates the convergence of the push-relabel method but calculating the hop-distances is very costly. We also suggest assigning heights with values in the way they make a gradient downhill to the sink in a simple operation and by means of the locations of the nodes. The initial height of a node u can be computed by the ratio of the distance between u and t to the communication radius r. Figure 2 shows the impact of the GN heuristic on the initial heights. The numbers written in the nodes are their initial heights and for the sake of simplicity, link capacities are eliminated. Simulation results illustrate that this initial gradient accelerates the movement of the flow toward the sink.

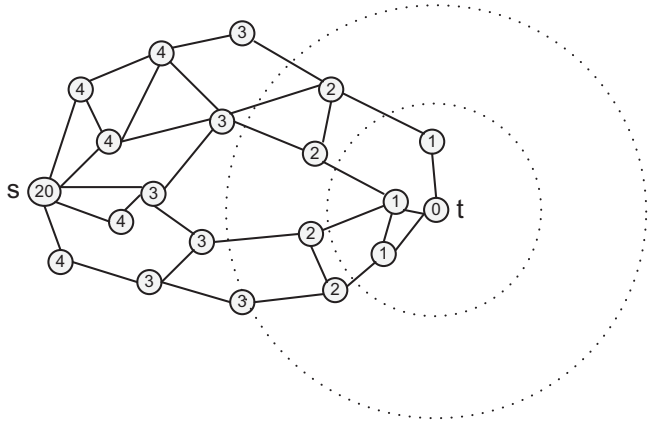

Fig. 2. The Gradient Network (GN) heuristic.

Lemma 1 shows that such an initialization of height values maintains the height function properties in GPR (Definition 2), and so GPR algorithm which is equipped with the GN heuristic works correctly and finds the exact max-flow value.

3.2.2. Smart Push

In most applications of MFP, in addition to the max-flow value, the flow paths are important too. For example, it is better almost in all applications to send flows via shorter paths or via shorter link distances (less number of hops). Unfortunately, most of the existing solutions for MFP only consider the worst-case complexities. Here we introduce some heuristics which could take into account such a path-quality objective.

In the GPR method, when one node decides to push, selects one downhill neighbor randomly and without considering any priority. However, if we use location of the nodes and apply the idea of greedy next-hop selection, then we could obtain better flow paths. For example to the purpose of shortening the paths, it suffices to push flow to the neighbors that are closer to sink.

Figure 3 illustratively compares the flow paths of the GPR method and the SP heuristic in one example network. In addition, applying the SP heuristic will cause to a little reduction in the communication overhead. Because transferring the flow from shorter paths will cause to faster convergence to the result.

This heuristic does not influence the worst-case analysis of the GPR. Hence, it works correct and does not affect the optimality.

3.2.3. Stunt-Source heuristic

The main idea of this heuristic is to reduce the initial height of the source. As we will show in the empirical experiments, the most resource consuming situation in the push-relabel algorithms is whenever the flow traps in the network i.e. there is some excess flow in the network which has not any way to the sink. This trapped flow oscillates between sensor nodes until active nodes elevate their heights to some values more than n. Then they can push the trapped excess back to the source. Implementing Stunt-Source heuristic is quite simple. In general, it suffices to assign $K < n$ to the initial height of the source. Furthermore, we can assign a value to the height of the source that is not dependent to n, e.g. the height of source can be a function of nodes density and the distance between source and sink. This method has three advantages: first—as we will show—by choosing more carefully the source height, we can reduce the lower bound of the worst-case complexities, in the cost of sucrifing a little flow. Second, source does not need to change its height after addition or removal of nodes to/from the network. Finally, according to the application one can trade the flow value over the algorithm overheads by choosing a lesser height value. We intentionally choose $K = c\sqrt{n}$ for reducing message complexity in worst-case analysis, in which C is a constant (see Theorem 1).

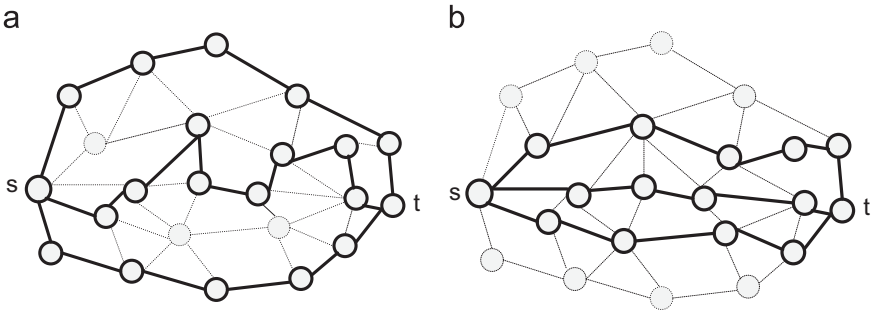

Fig. 3. Flow paths in GPR vs. SP heuristic.

3.2.4. Temporal Canceling (TC)

As mentioned before, canceling is a technique to discover trapped flows and redirect them back into the source. There is not an explicit method to discover the trapped flows, but we can guess it using flow's life-time or its distance from the sink. The TC-enabled node sends a PUSH message back to the source, when its life-time becomes more than a specified threshold. To implement such a canceling, each node must be equipped by a timer and each PUSH message must be augmented with a timestamp. When a flow moves in the network its timestamp accumulates queuing delays. Whenever the timestamp exceeds the specified threshold it should be canceled back to the source. The threshold could be computed as a coefficient of round-trip-time between the source and sink.

In general, utilizing canceling heuristics imposes PUSH message to convey an id-sequence that indicates the loop-free path from the source to the current node. As we will show in Lemma 2 such path always exists. When a node distinguishes a flow as trapped flow, it generates a CANCEL message, augments it with reversed id-sequence and the flow value in PUSH message and sends it back toward the source to withdraw its flow value.

Algorithm 7. The canceling operation – $check-canceling()$.

Input: Local variables such as the residual capacities of the neighbors, role, height and excess. We assume that the algorithm runs at node v.
Output: A CANCEL message carrying a flow value.
$isTrapped = examinTrappedFlow(PUSH)$;
if $(isTrapped \;\&\&\; role_v \neq SOURCE)$
  while $(there\ is\ an\ unhandled\ PUSH\ message\ containing\ trapped\ flow)$.
    $\delta = the\ flow\ value\ of\ the\ PUSH\ message$;
    $id-seq = the\ reversed\ id-sequence\ of\ the\ PUSH\ message.$
    $u = remove\ the\ last\ id\ from\ the\ id-seq$;
    $excess_v = excess_v - \delta$;
    $send\ (CANCEL, \delta, id-seq)\ to\ u$;

Algorithm 8. Nodes reaction upon receiving the CANCEL message.

Input: An incoming message from the channel with w as its sender and δ as its data. Moreover, the node uses local variables such as residual capacities, excess value and the height.
Output: Some changes in local variables and another CANCEL message.
On receiving CANCEL message :
if $(role_v \neq SOURCE)$
  $residual_v[w] = residual_v[w] + \delta$;
  $u = remove\ the\ last\ id\ from\ the\ id-seq$;
  $send\ (CANCEL, \delta, id-seq)\ to\ u$;

Algorithm 7 and Algorithm 8 depict the canceling method. These two methods emerge in all canceling heuristics. The only difference is in the way they mark a flow as the trapped flow.

3.2.5. Spatial Canceling (SC)

This heuristic is similar to TC. It differs in the strategy to mark a flow as trapped. It cancels a flow when its corresponding PUSH message becomes far away from the sink more than a specified threshold. For example, the threshold value could be a coefficient of the distance between the source and the sink. Another spatial strategy can be augmenting a PUSH message with a new field which maintains an accumulated distance it moves so far. Again, when this value extends to a specific threshold, the node cancels the flow.

3.2.6. Hop-Counting Canceling (HC)

This heuristic appends hop-count information to the PUSH message. Once a PUSH message moves between two nodes, its hop-count will be incremented by one. Once more, when this counter reaches a specified threshold, the flow should be canceled. The threshold can be computed by the ratio of the distance between the source and the sink, to the average hop-distance. In uniform distributed sensors in a 2-dimentional terrain this threshold can be a coefficient of the square root of the number of the network nodes. Combining this heuristic with SS heuristic, contributes to the asymptotical reduction in the worst-case message complexity of the existing solutions proposed for max-flow problem (see Theorem 1).

3.3. The adaptive algorithm

In this section, we will introduce two new operations, pull and pushback, which are used to adapt the max-flow in the network when capacities change.

Whenever the capacity of the link (u, v) increases, u invokes pull operation to transmit more flow, if possible. Similarly, once the capacity of link (u, v) decreases, it triggers the pushback operation. The pull and the pushback operations are depicted in Algorithm 9 and Algorithm 10, respectively. These operations use PULL and PUSHBACK messages, which are augmented with the id-sequence, if the algorithm has enabled canceling heuristics.

Algorithm 9. $pull()$.

Input: Local variables.
Output: A PULL message carrying an integer height value.
if $(residual_v[u] = 0)$
  $h_v = init-height(v)$;
for $(all\ \ w \in neighbors_v)$
  if $((residual_v[v] > 0)$
    $send\ (PULL, h_v)\ to\ w$;

The pull operation tries to obtain more flow from the neighbors. This operation is useful when the capacity of a link increases, a new link established or a node joints the network. In contrary, the pushback operation tries to cease some flow and redirect it from neighbors' residual capacities. If it is not possible, the operation would decrease the total net-flow. This operation is useful when the capacity of a link decreases, or a link or a node is omitted from the network.

Algorithm 10. The pushback operation – *pushback*().

Input: Local variables and $\Delta_{v,u}$ which is the amount of reduction in $c[v, u]$.
Output: A PUSHBACK message carrying two values, one is for flow difference and the other is for the height value of the node.

```
if (residual_v[u] ≥ Δ_{v,u})
  residual_v[u] = residual_v[u] − Δ_{v,u};
else
  x = Δ_{v,u} − residualv[u];
  h_v = init − height(v);
  send (PUSHBACK, x, h_v) to u;
  residual_v[u] = 0;
  excess_v = x;
  push();
```

Algorithm 11 shows the reaction of nodes upon receiving PULL and PUSHBACK messages produced by their corresponding operations.

Algorithm 11. Reaction of nodes upon receiving PULL and PUSHBACK messages from the network – *listen*().

Input: An incoming message from the channel with w as its sender and δ as its data. Moreover, the node uses local variables such as residual capacities, excess value and the height.
Output: Depends on the incoming message and the state of the node. It may be an update in local variables or calling other operations to propagate data through new PULL or PUSHBACK messages.

```
On receiving PULL message :
  h_v[w] = h;
  \− tif (excess_v > 0)
    push();
  \− tif (incoming flow of v equals outgoing flow from v)
    break;
  \− tif (incoming links of v are not saturated)
    h_v = init − height(v);
    \− tfor (all w ∈ neighbors_v)
      \− tif ((residual_w[v] > 0)
        send (PULL, h_v) to w;
On receiving PUSHBACK message :
  residual_v[w] = residual_v[w] − x;
  h_v[w] = h;
  h_v = init − height(v);
  \− tif (role_v = SINK)
    excess_v = excess_v − x;
    \− tfor (all u ∈ neighbors_v)
      \− tif (residual_v[u] > 0)
        tmp = min(r_v[u], x);
        residual_v[u] = residual_v[u] − tmp;
        send (PUSHBACK, tmp, h_v) to u;
        x = x − tmp;
      \− tif (x = 0)
        break;
```

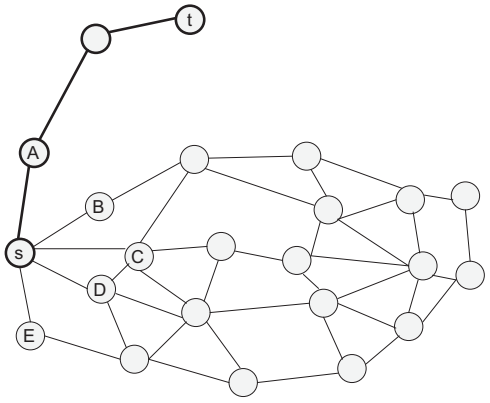

Fig. 4. The network with open end paths.

3.4. The incremental algorithm

Up to present, the proposed algorithm is asynchronous, lightweight and adaptive but still suffers from two significant drawbacks.

1. Most applications of WSN are sensitive to long delays but still the proposed algorithm is slow in some situation. However, in most cases a temporal partial solution can be utilized by the application.
2. The algorithm does not have any appropriate termination detection mechanism i.e. the relay nodes and especially the source do not know once the algorithm terminates and what are the flow paths. Fortunately, the sink node has this information and it can inform other nodes by using the id-sequence.

For example, consider a scenario depicted in Fig. 4. For the sake of simplicity, suppose that all capacities are unit. At the beginning source sends five PUSH messages to its neighbors. One of the PUSH messages reaches the sink rapidly, and the other PUSH messages will come back to the source after a long time. Consider the routing application. In this scenario, before receiving returned PUSH, source is unable to decide where to stream its data.

To this end we apply a standard algorithmic design technique: we will develop an incremental revision of the algorithm which elaborates the last part of the algorithm. Sink informs the source about received flows using the CONFIRM message. This strategy gives us a partial solution, instantly.

Suppose that every sensor node has a unique identifier. As well as canceling operations, we modify the PUSH message and its corresponding operation to carry id-sequence of the loop-free path from the source into the current node. Therefore, whenever a PUSH message reaches the sink node, it knows the backward path to the source. Hence the sink produces a CONFIRM message and launches this message into the path distinguished by the reversed id-sequence. All the internal nodes write the path information to their neighbor tables, truncate their identifiers from id-sequence and forward the CONFIRM message to the next node. For this purpose push operation should be modified according to the pseudocodes shown in Algorithm 12 and Algorithm 13.

Algorithm 12. The push operation which is smart and incremental.

Input. Local variables and $id-seq$ which holds the path of excess value from *source* to the current node.
Output. A PUSH message that carries an flow value and $id-seq$.
if $(role_v \neq SINK)$
$\backslash -$ t***while*** $(excess_v > 0)$ && $(\exists u \in neighbors_v \ such\ that\ (residual_v[u] > 0)$ && $(h_v[u] < h_v))$
$\delta = min\{excess_v, residual_v[u]\}$;
$excess_v = excess_v - \delta$;
$pushing-set = \{\forall u \in neighbors_v \ such\ that\ (residual_v[u] > 0)$ && $(hv[u] < hv)\}$;
$x = nearest\ node\ to\ the\ sink\ in\ the\ pushing-set$;
$residual_v[x] = residual_v[x] - \delta$;
$append\ v\ to\ id-seq$;
$remove\ cycles\ from\ id-seq$;
$send\ (PUSH, \delta, id-seq)\ to\ x$;

As mentioned before, it is likely to have loops in id-sequence so the algorithm should prune them. Since the maximum hop-distance in WSNs is not very large ($O(n)$ in the worst-case, and $O(\sqrt{n})$ in the average-case of the uniform distributed sensors in the 2-dimentional terrain), so the process of cycle removing will be done in a negligible time and the size of PUSH and CONFIRM messages does not increase dramatically. A little modification in the listen operation suffices to handle the cycle removing.

Algorithm 13. Reaction of the nodes upon receiving CONFIRM messages – *listen*().

Input: An incoming message from the channel with w as its sender, and δ and $id-seq$ as its data. Moreover, the node uses local variables such as residual capacities, excess value and height.
Output: Depends on the incoming message and the state of the node. It is quite likely to reproduce another CONFIRM message.
On receiving PUSH message :
$\backslash -$ t***if*** $(h_v[w] > h_v)$
$residual_v[w] = residual_v[w] + \delta$;
$excess_v = excess_v + \delta$;
$\backslash -$ t***if*** $(role_v \neq SINK)$
$push()$;
$\backslash -$ t***if*** $(excess_v > 0)$
$relabel()$;
$\backslash -$ t***else***
$pathID = assign\ a\ unique\ identifier\ to\ the\ new\ detected\ path$;
$send\ (CONFIRM, \delta, id-seq, pathID)\ to\ w$;
$\backslash -$ t***else***
$send\ (REFUSE, \delta, id-seq)\ to\ w$;
On receiving CONFIRM message :
$\backslash -$ t***if*** $(role_v = SOURCE)$
$add\ the\ new\ end-to-end\ path\ to\ the\ path\ table$;
$\backslash -$ t***else***
$w = remove\ last\ node\ from\ id-seq$;
$send\ (CONFIRM, \delta, id-seq, pathID)\ to\ w$;

By this modification, the proposed algorithm becomes an incremental algorithm, in which at any time the confirmed flows are the partial solution of the MFP. Soft real-time applications with no strict time constraints can use these partial solutions and terminate the algorithm whenever they decide with a fairly effective solution. Lemma 9 shows that the confirmed path is stable until the algorithm terminates, hence it can be used as a real-time partial solution.

In addition, we can enhance the quality of the flow paths by another modification which will be described by a simple example shown in Fig. 5. Suppose that in this network all capacities are one and after a while sink receives two distinct PUSH messages along the solid and dashed paths depicted in Fig. 5(a). In the favor of id-sequences, sink can discern the better alternative paths for these two paths as shown in Fig. 5(b). In this case, it suffices to send two CANCEL-like messages along these two paths and nodes change their tables according to the changes. The algorithm of this enhancement in the internal nodes is similar to the algorithm of handling the CANCEL messages. Furthermore, the sink can optimize the flow paths by using a substring matching algorithm. It suffices to find the maximum common substring between the reversed id-sequence of the last PUSH message and the id-sequences of the previous flow paths. We can split two flow paths which have any common substring by eliminating the substring, and then remerge four split sections in cross-over fashion generates the shorter paths.

4. The correctness and the complexities

In this section we analyze the correctness and the complexity of the proposed algorithm. We will prove the following claims:

- The GN heuristic maintains the attribute h as a height function. Therefore applying this heuristic does not affect the optimality of the solution (Lemma 1).
- Canceling heuristics at node v use an id-sequence in the PUSH message to store the path from the source to the node v. This path always exists (Lemma 2).
- Applying SS and HC heuristics simultaneously, by choosing parameters carefully, reduces the number of messages from $O(nm^2)$ to $O(nm)$ (Lemmas 2 through 8 and Theorem 1).
- The real-time version of the algorithm uses confirmed paths as partial solutions. These paths remain unchanged during the execution of the algorithm (Lemma 9).

To model the behavior of the proposed algorithm, we made some assumptions which are usually valid in WSNs.

(A1) *Bidirectional links with a single and error-free physical channel.* Like most of the algorithms for WSNs, we assume that all wireless links are bidirectional and the underlying wireless media has a mechanism to deliver packets to their destinations without any error.

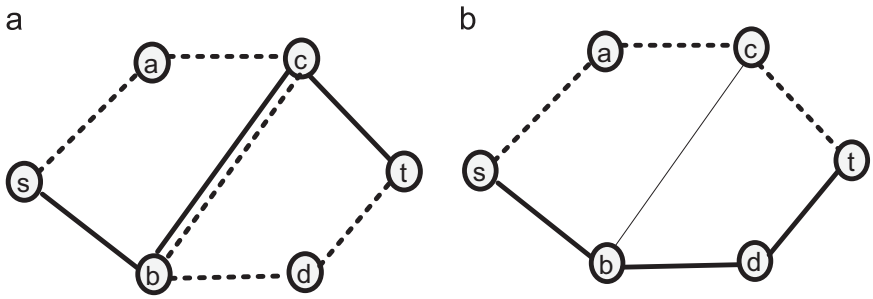

Fig. 5. Flow paths in the original confirmed path vs. shortened paths. (a) The original confirmed paths. Solid path: s-b-c-t; dashed path: s-a-c-b-d-t; reversed dashed path: t-d-b-c-a-s. (b) The shortened paths. Solid path: s-b-d-t; dashed path: s-a-c-t.

(A2) *Sparse topology.* The coverage area of an omnidirectional antenna with range r is generally modeled by a circle of radius r. Therefore, the Euclidean distance of the two end-points of a link is not more than r. Moreover, we suppose that the topology of the network is controlled to be sparse or planar.

(A3) *Location-awareness.* To enable the location-aware heuristics the sensor nodes must have a mechanism to identify their locations.

The following lemma shows that the GN heuristic works correctly and finds the exact max-flow value.

Lemma 1. *After execution of GN-enabled initialization operation in all nodes, the attribute h is a height function.*

Proof. Let $dist(u, v)$ be the Euclidean distance between nodes u and v, and r be the radio range described in (**A2**). Let $init[u]$ be the initial value of the $h[u]$ just after execution of the init-height operation. Assume that there is a link between nodes u and v.

$$\begin{aligned} &init[u]-init[v] \\ &= dist(u,sink)/r - dist(v,sink)/r \\ &< (dist(u,sink)-dist(v,sink))/r \\ &< dist(u,v)/r \ \text{(By triangle inequality)} \\ &\le 1 \ \text{(Since there is link between } u \text{ and } v) \end{aligned}$$

After execution of the init-height function at all nodes, $h[u]$ is equal to $init[u]$ for all $u \in V$ so $h[u] \le h[v] + 1$ is hold, proving the lemma. □

Lemmas 2 through 8 conclude the Theorem 1 which shows that using SS and HC heuristics simultaneously leads to an asymptotical decrease in the worst-case complexity of messages, from $O(n^2m)$ to $O(nm)$, and so by the Assumption (**A2**) the worst-case message complexity of the algorithm is $O(n^2)$. Furthermore, Lemma 2 justifies the canceling heuristics. The following proofs are based on previously developed schemes and techniques in Ahuja et al. (2014) and Cormen et al. (2009).

Lemma 2. *At any stage of the proposed algorithm, for each node i with positive excess, there exists a directed path form node i to the source (s) in the residual network.*

Proof. This lemma is a direct result of the flow decomposition theory which is a well-known proposition in the network flow literature. We sketch the proof below, for more details see Ahuja et al. (2014) and Cormen et al. (2009).

We show how to decompose any flow path into a path and cycle flows. Suppose that s has saturated its neighbors so s has negative flow balance and some link (s, i_1) carries a positive flow. If i_1 is an excess node, we stop; otherwise, the mass balance constraint (1-a) of node i_1 implies that some other link (i_1, i_2) carries positive flow. We repeat this manner until we encounter an excess node or we revisit a previously visited node. In either case the path or the cycle consists solely of links with positive flow.

Notice that at any stage of the algorithm, $e(s) \le 0$ and $e(i) \ge 0$ for all $i \in N - \{s\}$. By the abovementioned argument we can decompose any net-flow with respect to the original network G into nonnegative flows along (1) paths from node s to node t, (2) paths from node s to excess nodes, and (3) flows around directed cycles. Let i be an excess node relative to the net-flow in G. Note that the paths from node s to node t and the flows around cycles do not contribute to the excess at node i. Therefore, the flow decomposition of the net-flow must contain a path P from node s to node i.

The residual network contains the reversal of P (P with the orientation of each link reversed), so there is a directed path from node i to node s.

To be more precise, in the following lemmas we make another assumption.

(A4) The parameters of the heuristics. In the proposed algorithm, we use the SS heuristic with $h(s) = c_1\sqrt{n}$, and the HC heuristic with hop-count threshold equal to $c_2\sqrt{n}$, simultaneously, in which c_1 and c_2 are constants and $c = c_1 + c_2$.

Lemma 3. *By (**A4**), for each node $i \in N$, $h(i) < c\sqrt{n}$.*

Proof. The algorithm relabels a node i when it has a positive excess, so by Lemma 2 the residual network contains a path P of length at most $c_2\sqrt{n}$ from node i to node s. The fact that $h(s) = c_1\sqrt{n}$ and that $h(j) \le h(k) + 1$ for every link (j, k) in the path P implies that $h(i) \le h(s) + |P| < (c_1 + c_2)\sqrt{n} = c\sqrt{n}$.

Lemma 4. *Each height value increases at most $c\sqrt{n}$ times during the execution of the algorithm. Consequently, the total number of relabel operations is at most $cn^{3/2}$.*

Proof. This result follows directly from Lemma 3.

Lemma 5. *At most $O(m\sqrt{n})$ HEIGHT messages will be produced during one run of the proposed algorithm.*

Proof. By Lemma 3 the height of every node u_i increases at most $O(\sqrt{n})$ times and as shown in Algorithm 5 after applying every relabel operation, the node informs all neighbors about the new height value, so that the total number of HEIGHT messages is $\sum_{i=1}^{n} \deg(u_i).O(\sqrt{n}) = O(m\sqrt{n})$.

In the following, by saturating push (sat-push) we mean a push operation if it consumes all the residual capacity of its outgoing link, otherwise we call it as no saturating (nosat-push) push.###

Lemma 6. *The proposed algorithm performs at most $c\sqrt{n}m$ sat-pushes throughout the execution.*

Proof. It is not hard to show that between two consecutive saturations of a link, the height of both endpoints must increase

Table 1
The simulation configuration.

Terrain	A disk with radius = 100 m.
Number of nodes (n)	30–130.
Node distribution	Uniform.
Net. model	Unit Disc Graph
Radio range	30–50 m.
Topology	Maximal planar network – Delaunay triangulation without long edges.
Link capacities	Random number picked from 1 through 5.
Source and sink	One source and one sink in the opposite sides of the disk.
Iteration per config.	1000
Source height in SS	$3\sqrt{n}$

by at least 2 units (Cormen et al., 2009). Therefore, the total number of link saturations would be $(c\sqrt{n})(m)/2$.

Lemma 7. *The proposed algorithm performs $O(nm)$ nosat-pushes.*

Proof. We adopt a potential function from Ahuja et al. (2014) and Cormen et al. (2009) to bound the number of nosat-pushes. We have restated some details intentionally.

Let I denote the set of active nodes. Consider the potential function $\varphi = \sum_{i \in I} h(i)$. Since $|I| < n$, and $h(i) < c\sqrt{n}$ for all $i \in I$ (Lemma 3), the initial value of φ (after the initialization operation) is at most $cn^{3/2}$. At the termination of the algorithm, φ is zero. During the push (or Smart-Push) and relabel operation, one of the following three cases must apply:

- The node is unable to identify a link on which it can push some flow, so it triggers relabel operation and its height value increases by $\varepsilon \geq 1$ units. This operation increases φ by at most ε units. Since the total increase in $h(i)$ for each node i throughout the execution of the algorithm is bounded by $c\sqrt{n}$ (Lemma 3), the total increase in φ due to increases in height values is bounded by $cn^{3/2}$. On the other hand, when the node can push some flow to at least one of its neighbors, it performs a sat-push or a nosat-push which are considered in the two following cases.
- A sat-push on link (i, j) might create a new excess at node j, thereby increasing the number of active nodes by 1, and increasing φ by $h(j)$, which could be as much as $c\sqrt{n}$ per sat-push, and so by Lemma 6 increasing could be $(c\sqrt{n})(c\sqrt{n}m) = c^2nm$ over all the sat-pushes.
- A nosat-push on link (i, j) does not increase $|I|$. However, the nosat-push will decrease φ by $h(i)$ since i becomes inactive, but it simultaneously increases φ by $h(j) = h(i) - 1$ if the push causes node j to become active, the total decrease in φ being of value 1. If node j was active before the push, φ decreases by an amount of $h(i)$. Consequently, net decrease in φ is at least 1 unit per nosat-push.

We summarize the above mentioned facts. The initial value of φ is at most $Cn^{3/2}$ and the maximum possible increase in φ is $cn^{3/2} + c^2nm$. Each nosat-push decreases φ by at least 1 unit and φ always remains nonnegative. Consequently, the algorithm performs at most $cn^{3/2} + cn^{3/2} + c^2nm = O(nm)$ nosat-pushes.

Lemma 8. *At most $O(nm)$ PUSH and CONFIRM messages will be produced during the execution of the proposed algorithm.*

Proof. It is obvious that the number of CONFIRM messages is not more than the number of PUSH messages so the above result follows directly from Lemma 6 and Lemma 7.

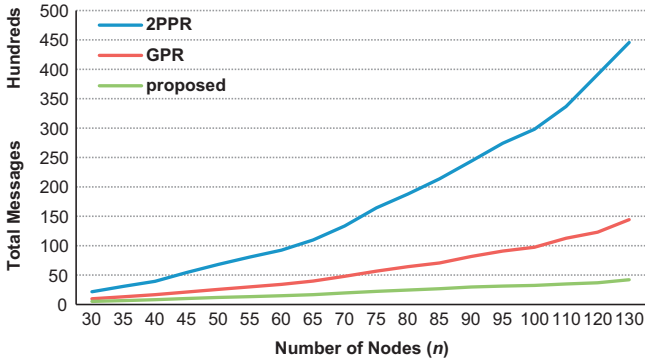

Fig. 6. Communication cost of 2PPR, GPR and the proposed algorithm.

Theorem 1. *By (**A4**), the proposed algorithm terminates with $O(nm)$ messages, and by (**A2**), algorithm terminates with $O(n^2)$ messages.*

Proof. This result follows directly from Lemmas 5 and 8.

Note that applying SS and HC heuristics has the cost of losing optimality of the flow value. However, as we will see in the next section, the solution is near-enough to the optimal max-flow in random generated networks.

Lemma 9. *In the incremental version of the algorithm, the confirmed flows remain unchanged, which means the incremental version of the algorithm works properly.*

Proof. Let $residual[p_i, p_{i+1}]$ be the residual capacity of the link (p_i, p_{i+1}). We know that the proposed algorithm is based on the GPR method which is an iterative algorithm. In each iteration, a path $P = p_0p_1 \ldots p_k$ will be discovered and the value $\sum_{i=0}^{k-1} residual[p_i, p_{i+1}]$ will be augmented to the total net-flow and this value will be removed from the residual capacities of the links of P. After this iteration, the residual network will be updated and the problem is the same but with the degraded residual network as its input. Therefore, the confirmed flows become eliminated from the problem and they remain unchanged in the solution.

5. Simulation results

In this section we first compare message complexities of the proposed algorithm and some naïve distributed versions of existing solutions. Then we get through the time needed for every algorithm to get finished. Afterward, we illustrate the interesting results about near-optimality of the proposed algorithm. Finally, we adjust some network factors such as nodes density and link capacities to ensure that improvements exist in most simulation situations.

Table 2
The ratio between communication costs (%).

#Nodes (n)	30	40	50	60	70	80	90	100	110	120	130
Proposed/GPR	53.3	48.8	46.8	43.3	41.2	38.2	36.4	33.4	10.2	9.5	9.4
Proposed/2PPR	24.2	20.8	17.8	16.1	14.8	13.1	12.2	10.9	30.9	300	29.2

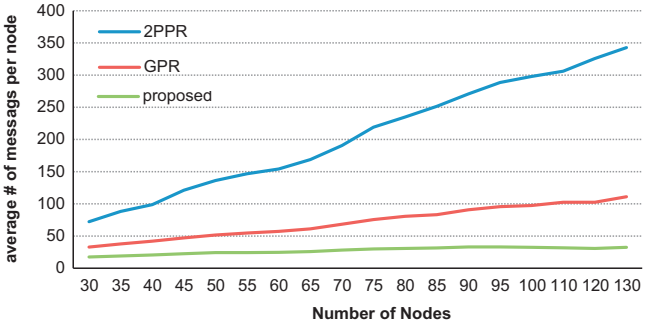

Fig. 7. Normalized communication cost (cost per node).

To evaluate the applicability of the proposed algorithm, we have used the configuration shown in Table 1. For each random generated network, three algorithms are applied:

- Two-phase push-relabel algorithm (2PPR) proposed in Pham et al. (2005). This algorithm in the first phase initiates heights of nodes with their hop-distance from sink by means of asynchronous breadth first search (flooding) started from the sink. The second phase is exactly as GPR as follows.
- Generic push-relabel (GPR) which is asynchronous version of Goldberg and Tarjan (1988) and can be adopted easily by removing the first phase of 2PPR (Pham et al., 2005) and assigning zero to the heights of the internal nodes.
- Proposed algorithm in which Stunt Source (SS), Gradient Network (GN), and Smart Push (SP) heuristics are applied simultaneously.

Figure 6 shows the number of produced messages in the algorithms. The vertical axis shows the number of messages in hundreds and the horizontal axis shows the number of nodes in the network.

As depicted the proposed algorithm reduces the message overhead significantly. Table 2 gets more insight into the impact of the reduction in the communication cost. Numbers in the rows indicate ratio of the produced messages in the proposed algorithm over the produced messages in GPR and 2PPR. As a result, we can say in larger networks, the proposed algorithm reduces the number of messages up to 91% in compare with 2PGR, and reduces it up to 71% in compare with GPR.

Figure 7 illustrates the scalability of the algorithm in compare with 2PPR and GPR. In simulation, when the network becomes larger and larger (in the case of number of nodes), the average number of messages produced in the proposed algorithm oscillates around 35 messages per node and seems to be converged around this value. In contrast, average number of the messages produced by 2PPR and GPR grows up almost linearly.

From simulation results, we also found out that the ratio between time complexities in the algorithms is very similar to the average communication cost ratios depicted in Fig. 7. It is evident that the consumed time is direct coefficient of the number of rounds in which the algorithm produces messages in its nodes simultaneously.

We also realized that the most effective heuristic in reducing number of messages is SS that reduces number of messages around 5 times stronger than GN and even more in comparison with the other heuristics.

Table 3
The impact of capacities.

Capacities	2PPR	GPR	Proposed
Unit	90.9	36.2	16.0
Random integer picked up between 1 and 10	137.8	58.7	25.6
Random integer picked up between 1 and 100	249.1	122.5	46.2
Random integer	270.3	146.1	50.6

The most attractive result is the effect of the proposed algorithm on the optimality. In the above simulation setting which source height is set to $3\sqrt{n}$ instead of n, the reduction in optimality is very small and almost negligible. In average, in every 1000 rounds of simulation, 993.4 rounds terminate in the exact max-flow. Even in 0.0066% of configurations with near-optimal solution, the computed flow values are more than 92% of the max-flow value, in average. All in all, by considering the dynamism and uncertainty of Wireless Sensor Networks, the deviation from optimal solution could be neglected.

We defined the sum of link lengths whose flows are positive as the path length criteria. Simulation results for 30 through 130 nodes show that in average the path length criteria in the proposed algorithm which uses SP heuristic is 8.7% smaller in compare with GPR and 2PPR algorithm. Furthermore, this heuristic makes a slightly reduction in the number of messages, around 3.5%.

Finally we investigate sensitivity of performance on capacities. We repeated a simulation in a network with 50 nodes in four different edge capacities: unit capacities, random integers between 1 and 10, random integers between 1 and 100, and fully random integers. Simulation results illustrate that push-relabel methods in a network in which capacities are picked up from a little set, produce lesser control messages. As depicted in Table 3, push-relabel method in the unit-capacitated network produces the least number of messages whereas fully random-capacitated network produces most number of messages. Number of control messages in the later network is around four times more than in the former network. We figured out there is not any meaningful relation between the edge capacities and the solution accuracy.

6. Conclusion

The max-flow problem has many applications in Wireless Sensor Networks (WSNs). Although, this problem is well-studied but the existing algorithms to solve this problem suffer from high overheads and inadaptability which make them not applicable in WSNs. In this paper the max-flow problem has been restated in a way that the objective is not only finding the optimal max-flow but also reducing message and time overheads and being adaptive to network changes are some other objectives. The algorithm consists of several heuristics which make it asynchronous, light-weight, adaptive and real-time.

At one hand, some of the proposed heuristics find the near-optimal flow with little fraction of messages needed for the other existing algorithms, whereas near-optimal flow is almost same as the optimal max-flow value. Stunt Source and canceling heuristics

Table 4
Summary of the proposed techniques and heuristics.

Heuristic, technique or algorithm	Location-aware	Appends id-sequence to messages	Related messages[a]	Asynchronous	Lightweight	Adaptive	Real-time	Intensity of reduction in the # of the control messages in compare with Goldberg and Tarjan (1988) and Pham et al. (2005).	Optimality[b]	Path quality
Gradient Network (GN)	✓			✓	✓			Medium	MF	
Smart Push (SP)	✓		PSH, RFS	✓	✓			Little	MF	✓
Stunt Source (SS)			HIT	✓	✓			Drastic	NOF	
Temporal Canceling (TC)		✓	CAN	✓	✓		✓		NOF	
Spatial Canceling (SC)	✓	✓	CAN	✓	✓		✓		NOF	
Hop-Count Canceling (HC)		✓	CAN	✓	✓		✓		NOF	
Pull			PUL	✓		✓			MF	
Pushback			PBK	✓		✓			MF	
Incremental		✓	CFM	✓			✓		MF	
Enhanced Incremental		✓	CFM, CAN	✓			✓		MF	✓
The proposed algorithm	✓		PSH, HIT, RFS, CAN, PUL, PBK, CFM	✓	✓	✓	✓	Drastic	NOF	

[a] PSH=PUSH, RFS=REFUSE, HIT=HEIGHT, CAN=CNACEL, PUL=PULL, PBK=PUSHBACK, and CFM=CONFRIM.
[b] MF means optimal max-flow and NOF means near-optimal flow.

terminate in such a near-optimal solution. At the other hand, some other heuristics such as Gradient Network and Smart Push ensure the optimality but they use the location of nodes to obtain the problem objectives, so that these heuristics are applicable in location-aware networks.

All the proposed heuristics and methods are independent, thus we can use some of them and eliminate some others to configure an application-specific algorithm even in other wireless networks. Table 4 summarizes the proposed contributions.

It worth mentioning that this algorithm most of the heuristics could easily be extended to apply in multi-source/multi-sink networks. But some others still need more research. For instance, SS, SP, TC, SC, and HC could easily be deployed in such a network. However, GN and incremental version need a little more strive to get adopted with multi-source and multi-sink networks, respectively.

We hope that the proposed algorithm would open a new window to the novel algorithms and protocols in several issues such as energy optimization, network coding, routing, aggregation and multimedia streaming.

References

Abdrabou A, Zhuang W. Statistical QoS routing for IEEE 802.11 multihop ad hoc networks. IEEE Trans Wirel Commun 2009;8(3):1542–52.

Ahuja RK, Magnanti TL, Orlin JB. Network flows: theory, algorithms, and applications. England: Pearson Education Limited; 2014.

Al-Kofahi OM, Kamal A. Network coding-based protection of many-to-one wireless flows. IEEE J Sel Areas Commun 2009;27(5):797–813.

Anderson RJ, Setubal J. On the parallel implementation of Goldberg's maximum flow algorithm. In: Proceedings of 4th ACM SPAA; 1992. p. 168–77.

Avestimehr AS, Diggavi SN, Tse DNC. Wireless network information flow: a deterministic approach. IEEE Trans Inf Theory 2011;57(4):1872–905.

Bader D, Sachdeva V. A cache-aware parallel implementation of the push-relabel network flow algorithm and experimental evaluation of the gap relabeling heuristic. In: Proceedings of the 18th ISCA international conference on parallel and distributed computing systems; 2005.

Bogliolo A, Delpriori S, Lattanzi E, Seraghiti A. Self-adapting maximum flow routing for autonomous wireless sensor networks. Clust Comput 2011;14(1):1–14.

Caragea GC, Vishkin U. Brief announcement: better speedups for parallel max-flow. In: Proceedings of the 23rd ACM symposium on parallelism in algorithms and architectures; 2011. p. 131–4.

Chandran B, Hochbaum D. A computational study of the pseudo-flow and push-relabel algorithms for the maximum flow problem. Oper Res 2009;57(2):358–76.

Cheriyan J, Hagerup T, Mehlhorn K. An O(n3)-time maximum-flow algorithm. SIAM J Comput 1996;45:1144–70.

Cherkassky B, Goldberg A. On implementing the push-relabel method for the maximum flow problem. Algorithmica 1997;19(4):390–410.

Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. 3rd ed. New York: The MIT press; 2009.

Dandapat SK, Mitra B, Ganguly N, Choudhury RR. Fair bandwidth allocation in wireless mobile environment using max-flow. In: Proceedings of the international conference on high performance computing (HiPC); 2010. p. 1–10.

Dinic EA. Excess scaling and transportation problems. Issledovaniya po Dikretanoi Matematike. Moskva: Nauka; 1277–80.

Edmonds J, Karp RM. Theoretical improvements in algorithmic efficiency for network flow problems. J ACM 1972;19(2):248–64.

Ford LR, Fulkerson DR. Maximal flow through a network. Can J Math 1956;8:399–404.

Ghosh S, Gupta A, Pemmaraju SV. A self-stabilizing algorithm for the maximum flow problem. Distrib Comput 1997;10(4):167–80.

Goldberg AV, Rao S. Beyond the flow decomposition barrier. J ACM 1998;45:783–97.

Goldberg AV, Tarjan RE. A new approach to the maximum-flow problem. J ACM 1988;35(4):921–40.

Homayounnejad S, Bagheri A and Ghebleh A. Near-optimal flow in wireless sensor networks. In: Proceedings of the 34th international convention, MIPRO. Opatija (Croatia): IEEE; 2011a. p. 434–9.

Homayounnejad S, Bagheri A and Ghebleh A. AAA: asynchronous adaptive algorithm to solve max-flow problem in wireless sensor networks. In: Proceedings of the 34th international convention, MIPRO. Opatija (Croatia): IEEE; 2011b. p. 440–5.

Hussein M, Varshney A, Davis L. On implementing graph cuts on cuda. In: Proceedings of the first workshop on general purpose processing on graphics processing units (GPGPU); 2007.

Kalpakis K. Everywhere sparse approximately optimal minimum energy data gathering and aggregation in sensor networks. ACM Trans Sens Netw 2010;7(1):1550–4859 9.

Ohara Y, Imahori S and Van Meter R. MARA: maximum alternative routing algorithm. In: Proceedings of IEEE INFOCOM; 2009. p. 298–306.

Orlin JB. Max flows in O (nm) time, or better. In: Proceedings of the forty-fifth annual ACM symposium on theory of computation, STOC; 2013. p. 765–74.

Pham TL, Lavallee I, Bui M and Do SH. A distributed algorithm for the maximum flow problem. In: Proceedings of the 4th international symposium on parallel and distributed computing, ISPDC; 2005. p. 131–8.

Shiloach Y, Vishkin U. An O($n2\log n$) parallel max-flow algorithm. J Algorithms 1982;3(2):128–46.

Song WZ, Wang WZ, Moaveni Nejad K, Li XY. Lifetime maximized cluster association in two tiered wireless sensor networks. Wirel Commun Mob Comput 2009;9(3):325–34.

Soorki MN, Rostami H. Label switched protocol routing with guaranteed bandwidth and end to end path delay in MPLS networks. J Netw Comput Appl 2014;42:21–38.

Valero M, Xu M, Mancuso N, Song WZ, Beyah R. EDR2: a sink failure resilient approach for WSNs. In: Proceedings of IEEE international conference on communication (ICC); 2012. p. 616–21.
Vineet V, Narayanan P. Cuda cuts: fast graph cuts on the gpu. In: Proceedings of IEEE computer society conference on computer vision and pattern recognition workshops (CVPR); 2008. p. 1–8.
Wu D, Negi R. Effective capacity-based quality of service measures for wireless networks. ACM Mob Netw Appl 2006;11(1):91–9.
Yantao P, Lu X, Zhu P. DTM: a distributed traffic management algorithm for sensor networks. Int J Comput Sci Netw Secur 2006;6(8A):208.