A Novel Fault Tolerant Protocol for Information Propagation in Sensor Networks

Bidyut Gupta, Shahram Rahimi, and Arun Kumar

Southern Illinois University, Carbondale, IL 62901 USA

Summary
In this paper we have proposed a fault-tolerant protocol for fast and robust propagation of information in sensor networks. The proposed approach has been shown to offer better bandwidth utilization as well as faster delivery of information in sensor networks than some noted existing works. Fault-tolerance offers robust data delivery to the sink by the deployed sensor nodes, where the sink sensor serves as the gateway node to the external environment.

Keywords:
Sensor networks, Source, Sink, Fault-tolerant routing

Introduction

Sensors are designed to detect events, collect data and are equipped with data processing and communication capabilities. The development of network of low-cost, low-power, multifunctional sensors have received increasing attention [1]. The sensing circuitry senses the events in its vicinity related to the application to be dealt with and the environment surrounding the sensor and transforms them into an electric signal for transmission. The research in the area of sensor networks is fast increasing and so are its use in many different applications from general engineering applications to improving the safety and efficiency of traffic, agricultural and environmental monitoring applications and also many other*s* like civil engineering, military applications and health monitoring, surgery, etc [2]. The limitations in power transmission, energy and computing power make the sensor networks different from other wireless adhoc or mesh networks. There are many other factors which are to be considered when designing the routing protocol such as the deployment of the nodes in the network [7], the data delivery models, the nodes capabilities. For instance, some sensors can be homogenous and others heterogeneous, i.e., some nodes are of different processing capabilities.

There are three different classes of the protocols which are:

1) Data-centric protocols in which the sink queries to certain regions and waits for the data from the sensors located in the selected regions [5],
2) Hierarchical protocols which help in solving the scalability issues by reducing the load on the nodes by the use of tired architecture and it also assists in efficient maintaining of the energy consumption,
3) Location-based routing which makes use of the location information i.e., it deals with the applications where the location of the sensor nodes is known and using this information in routing makes such types of protocols more reliable.

A sink besides being a sensor node similar to the other sensor nodes in the network, has the added capability of communicating with the outside network like laptop, Desktop, Base Station where all the data are collected [3] and further processed. An *Event* in a sensor network is defined as any change in the current conditions where these sensors are deployed. The sensor in the vicinity of the location where the event occurs gathers the data, stores it and forwards it to the sink node. The protocol presented in this paper comes under the data-centric protocol. In our proposed algorithm, we find a path between a sink and an Event-source without much delay. The first redundant path (i.e., the second best path) can be used as a fault-tolerant path. The protocol should work well for a network of any number of sensor nodes and it will also handle the problems due to the neighboring node failures or the link failures. The deployment of the nodes in this protocol is random keeping in consideration that the nodes are within the transmission range of each other and all the nodes have the same processing capabilities making it a homogeneous network.

This paper is organized as follows: in Section 2 we explain in brief the related works in the data-centric routing protocols and we state our problem formulation. In Section 3, a clear idea of our approach is given with an example. In Section 4, we have presented our fault tolerant routing protocol and its performance. Section 5 draws the conclusion.

2. Related works

Directed Diffusion [4] is a data-centric approach where the sink node broadcasts the data expected and the details it wants to know, to the other sensor nodes and this request is further propagated by other nodes. For this

purpose, the method uses a control packet called interest [4]. The sink broadcasts the interest in the form of a query to its neighbors; the query has in it the attribute-value pairs based upon the properties of the objects or the surroundings being sensed. The attribute-pair values are dependent on the applications the sensors are being used for and are defined based on name of the objects, interval, duration, geographical area, etc.

The data sensed by the sensor nodes (present in their cache) are compared with those arriving in the interest, i.e., the expected data are compared with the collected data. Besides, the interest packet also contains a gradient field [4]. A *gradient* is the weight calculated between two adjacent nodes and is characterized by data rate or duration time, etc. A brief introduction of this protocol is as follows; the query is broadcasted by the sink node to its neighboring nodes. An intermediate node say, X receives the interest from a node Y. Node X, before forwarding this interest to its other neighbors, adds the gradient between X and Y to the interest packet. In this way, when the interest packet finally reaches the source, the source gets the total weight (summation of all the gradients in the gradient field) of the path that this packet has traversed through from the sink. The source node waits for a certain amount of time before which many such interests may reach it. But the source selects a path (the best path) based on the cumulative gradient. In other words the source node waits to collect information (total weight) of several paths to the sink. The path with the minimum total gradient is considered to be the best path. Then the collected information is delivered along this path to the sink. Detailed explanation of its propagation is not important for our work in this paper. However, one note worthy point is that the source may have to wait for a considerable amount of time to determine the best path; hence it delays the data delivery.

A slight variation of the Directed Diffusion method is the Gradient-Based Routing [6], where the *number of hops* is noted when the interest is propagated through the network. It was proposed to simplify the original Directed-diffusion. In this method, each node has the knowledge of the *minimum number of hops* from itself to the sink. This parameter is called the *height of the node*. In this work [6], the difference between a node's height and that of its neighbor has been considered as the gradient of the link between those nodes.

Problem formulation:
In this work, we have proposed a protocol that reduces the delay (to the minimum) at the source sensor node during the determination of the best (minimum) path between a sink and a source sensor node. This ensures the faster delivery of data compared to the works reported in [4], [6], where the proposed algorithms take time to decide the minimum path because the source needs time to gather several requests via different paths. Besides, the control information added to the interest packet may cause inefficient utilization of bandwidth. Extra information like the minimum number of hops is necessary at each node in [6], to determine the best possible path for data delivery to a sink. Besides, the fault- tolerant property of our proposed approach also offers robust routing.

3. An Outline of the Proposed Approach

We have proposed a fault- tolerant approach for the propagation of the interest (which we call as 'request' packet) and the reply packet between a source and a sink.

Initially the Sink broadcasts a Request packet to all its neighboring nodes which is propagated right through, till the Source is reached. This Request may refer to some physical properties of the surroundings and also the kind of data expected. The parameters of this Request are decided by the Sink node and for every such request, the sink has the Timer to know the expiry limit. In our approach, when the request is broadcast, each sensor node appends its own id and deletes the previously appended node id (from which this node has received the current request) to the current path so that the source node and the other intermediate nodes know which node to send the reply packet next.

Thus all the nodes receiving the request will propagate it further to their immediate neighbors. Now the Source node will wait till its TTL (Time-to-Live) expires. This TTL is the timer maintained in the source node and is explained further. Its value is checked at the source node, and it will wait to receive the request only till the time in this timer expires. When the source node receives the first request, it replies to the sink via the neighbor from which it first received the request packet. In this way, the source comes to learn about the best (minimum) path as soon as it gets the first request. This makes the path discovery fast. Each node knows the next node to send the reply packet, towards the direction of the sink node.

Before its timer expires, if the source receives the second request packet, it takes note of the node from which it has received this request. This gives the information to the source node about the second path which can later be used (if necessary) to achieve fault-tolerant delivery of data.

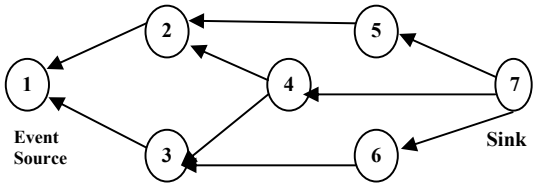

Fig. 1 Request propagation from Sink to source

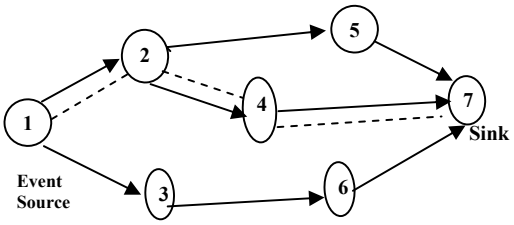

Fig. 2 Propagation of path 1-2-4-7

In Fig. 1, the sink broadcasts the Request to its neighbors and the subsequent nodes follow suite. Without any loss of generality, let us consider that 1 is the node which is the one nearest to the event-location and hence considered as the source. 7 is the sink broadcasting the request packet and this packet is propagated till it reaches 1 (source sensor). In Fig. 2, we assume that the source node 1 gets the first request from node 2 and so sends its reply to node 2. This process is continued till the sink is reached. It is noted that the roles of the nodes are reversed while the reply packet is sent from the source to the sink. Every node maintains a table to know the immediate neighbors information corresponding to the different paths passing through it. Now node 2 sends the reply packet to its neighboring node 4 through this available information (assuming that 7-4-2-1 is the best path) and this procedure is followed at every sensor node along the path till the sink is reached.

4. A Fault-Tolerant Protocol

4.1 Relevant data structures

The following data structures are used in our approach:

Source Table (ST):

This table is maintained in the source node in the path and it consists of 3 parameters. *Sink ID* indicates the sink from which the request has been broadcast. *Next Node ID* gives the physical id of immediate neighbor which forwarded the request packet to the source sensor node. *TTL* is the *Time-to-Live*, the time for which the source node waits for the request originating from the sink. This means the source keeps the data for a time equal to TTL.

Sink ID	Next Node ID	TTL

Sink Navigation Table (SNT):

This table is maintained by the intermediate nodes in the path (between the source and the sink). *Sink ID*, again, indicates the sink from which the request has been broadcast. *Next Node ID* gives the physical id of its immediate neighbor which has forwarded the request packet to the current sensor node. The values of the next node id and the previous node id are interchanged if a failure along a path during reply packet propagation has to be communicated to the source node. Moreover, it is necessary to keep track of the *previous node id*, during the reply packet propagation, because when an unreachable node along a path has to be notified to the source node, this parameter is made use of.

Sink ID	Previous Node ID	Next Node ID

Request Packet ($P_{request}$):

Request packet is propagated from node to node. The first parameter is *flag* and if the value of this single byte is 0, it indicates a request packet $P_{request}$. $Timer_R$ is a preset counter, i.e., the approximate upper bound before which the sink expects a reply for the current request. *Sink ID* indicates the sink from which the request has been broadcast. *Next Node ID* gives the physical id of the immediate neighbor which forwarded the request packet to the current sensor node. *Request Description* is the field that gives the properties or the parameters of the object that sink looks forward to collect information about. $Timer_R$ is decremented in each intermediate node along the path to the source and also at the source sensor.

flag = 0	$Timer_R$	Sink ID	Next Node ID	Request description

Reply Packet (P_{reply}):

This packet will be sent along the best path found to the sink. Reply packet has five parameters. The first parameter is *flag* and if the value of this single byte is 1, it indicates a reply packet, P_{reply}, with data collected. D_{coll} has the data collected by the source $Timer_R$ is decremented in each intermediate node along the path to the sink. The parameters *Sink ID* and *Next Node ID* have the same roles as described in $P_{request}$ (but the direction is reversed).

flag = 1	$Timer_R$	D_{coll}	Sink ID	Next Node ID

Fault Tolerant Path Packet (P_{FTP}):

We assume that a second best path is cached at the source node. This is also a reply packet meant for data delivery, but it is noteworthy that the path followed by this reply packet is different than the one followed by P_{reply} and is used as a Fault Tolerant path between the sink and the source. The first parameter flag = 2, indicates that this is a

reply packet sent via the fault tolerant path along with the data collected. It also contains the Sink ID and the next node id to reach the sink and the $Timer_R$.

flag = 2	$Timer_R$	D_{coll}	Sink ID	Next Node ID

Node Unreachable packet (P_{NUR})

This is a packet sent by an intermediate node to abort any further data delivery if it finds that the next node along the path to sink is unreachable. This is further propagated to the source. Here the flag is set to 3. The packet is sent to the previous node and this is continued till the source is reached. $Timer_R$ is also sent in this packet.

flag = 3	Previous Node ID	$Timer_R$

4.2 An illustration

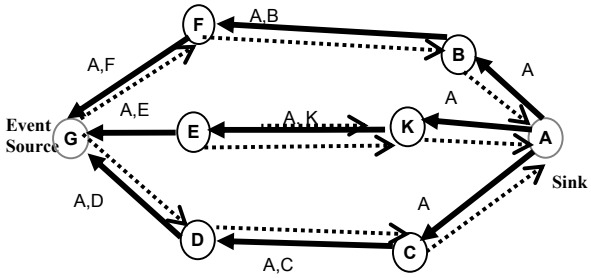

Fig. 3 Multiple path formation between the Source and the Sink of our approach

Let us consider the sensor network of as shown in Fig. 3: The sensor node G is the source and node A is the sink broadcasting the request to its neighboring sensor nodes B, K and C. These nodes will broadcast the request further to their neighbors. In this case, B, K and C have only one neighbor each. So node B forwards the request to F; node K forwards to E and node C forwards to D. F, E and D propagate the request to their only neighbor, which in this case is the source node G.

In Fig. 3, consider that G receives the first request from node A through the sensor nodes B and F. Before the TTL expires, node G also receives the same request through some other path, say via the nodes K and E, the information about which is stored in its buffer. Node G sends the collected data to the node (in a reply packet) through which the first request packet is received. If node A (the Sink) does not receive any reply packet till its Timer expires, it rebroadcasts the request. In Fig. 3, the Dotted lines indicate all the possible paths between the Source and the Sink in the given network of nodes. This approach can be easily carried out since the next node information is known to each node.

For example, in Fig. 3, consider *G-F-B-A* to be the best (minimum) path. Then G will know immediately that its next hop will be F from the information available from the $P_{request}$ packet, where, Sink ID =*A* and Next Node ID = *F* and similarly node F will know about its connection details to the node B. Thus G starts the data delivery to the sink through a P_{reply} packet. Before the TTL at G expires, assume that another $P_{request}$ reaches here along the nodes K and E. This can be used as the Fault-tolerant path. We have also considered a case, i.e., say node B is unreachable from the sensor node F and hence node F is not able to transmit further via this path. So node F sends a Node Unreachable packet (P_{NUR}) to its previous node (here, the source G), in which, it sets the flag to 3 to indicate that the sink cannot be reached via this path, and decrements the $Timer_R$ before sending this packet. Once the source G gets this packet, it makes use of the fault-tolerant for data delivery through the reply packet P_{FTP} to node E, setting the flag = 2, to indicate to the sink A that this reply is done along the fault tolerant path. E further unicasts this to K and then the reply finally reaches the sink A.

4.3 The proposed protocol

Let us consider a network of n nodes being represented as $N_1,N_2,N_3,....,N_n$. If the i^{th} node N_i becomes a sink, it will also be denoted by S_i, and if the i^{th} node becomes a source node, it will also be denoted by N_{source}, where $1 \le i \le n$. The request packet $P_{request}$ is broadcast by the sink node S_i to all its neighbors N_j and all these nodes in turn forward this packet to their neighbors. This packet ultimately reaches the Source node N_{source}, which unicasts the reply packet P_{reply} to the immediate neighbor *prev_node_id* from where the source received the request. As long as the nodes along the path are reachable, this procedure is repeated till P_{reply} is delivered to the sink S_i. Let *flag* be a single byte indicating the packet type. Also we have considered the fault-tolerant delivery of data, i.e., if the reply packet does not reach the sink, because of any failure along the best path source come to know about it through the intermediate nodes. It then sends a reply packet P_{FTP} along the fault-tolerant path. Our algorithm also mentions the case in which the fault-tolerant path fails and the imminent responsibility of the source sensor node, which stops any further delivery of data to the sink node. The responsibilities at the sink node, the intermediate nodes and the source node are stated in Fig. 4.

The paths cached in the source sensor node may be useful for the future data gathering. For instance, let X be a source sensor node providing data to the sink Y. If later, X needs to contact Y (because of whatever reason), it can do that following the minimum (best) possible path, or if necessary following the second best path.

At any Sink node

```
Set the timer, TimerR;        // internal timer for the sink and a timer for the traveling packets.
data_received = false;        // a Boolean variable to know if a reply packet is received for the request.
Broadcast the Prequest to the neighbors;    // will be executed iff the sink has any new or pending request
if (timer does not expire)
        if (data_received  = true)    //  a reply packet is received.
            Forward the data received to the Base station;
else rebroadcast the Prequest
```

At any source node:-

```
while (TimerR does not expire)
  if (TTL ≠ 0)
     if (flag =0)   // a request packet
        if (data_described = data collected) // this is the first request packet. This has come via the best path.
           Nsource  =  Nj;
           Decrement TimerR;
           Store the best path in the cache;
           Set flag =1;
           Unicast the reply with data to the next node;      // this reply is done along the best path.
        // Now source waits for a time 't' to get a same request again.
           if ( same request is received again)
              decrement TimerR;
              set flag = 2;
              Store the second path in cache;   // a fault tolerant path is found.
     if (flag =3)    // some intermediate node along a path is not reachable
        if ( Source already unicast the data along the fault-tolerant path)
           Source stops the reply propagation to its neighbors; // no more reply packet is sent.
        else
           decrement TimerR;
           Set flag =2;
           Unicast the reply with data along the second path;     // this is the fault tolerant path
     if (TTL = 0)
        Set flag = 1;
        Data_collected = null;
        Decrement TimerR;
        Unicast the reply packet to sink; // this notifies the sink that the path is still available, but the source TTL has expired.
```

At any intermediate node

```
  while (TimerR does not expire)
     if (flag =0)  // data request packet
           prev_node_id = Nj-1;
           decrement TimerR;
          // Check if all the neighbors (Next nodes) are reachable
            if  (reachable)
              Broadcast the request packet
            else    // one of the neighbors is unreachable
              Stop propagating to that neighbor; // request is broadcast to all other neighbors that are reachable
     else if (flag =1)  // reply packet with data along the first path.
           // Check if the next node is reachable
           next node id = previous node id; // the direction of propagation is reversed during the reply packet, so node ids are set accordingly.
           if (reachable)
              Decrement TimerR;
              Unicast the reply packet to the next node;
           else      // the next node is unreachable
              Set flag =3;
              Decrement TimerR;
              Unicast to the previous node;
     else if (flag =2)    // reply packet with data along the fault-tolerant path
           next node id = previous node id; // the direction of propagation is reversed during the reply packet, so node ids are set accordingly.
           // Check if the next node is reachable
           if (reachable)
              Decrement TimerR;
              Unicast the reply packet to the next node; // this is sent with just the path.
           else      // the next node is unreachable
              Set flag =3;
              Decrement TimerR;
              Unicast to the previous node;  // this packet will finally reach the source.
```

Fig. 4 Algorithms for the sink node, the intermediate nodes and the source node

At any source node:-

```
while (Timer_R does not expire)
  if (TTL ≠ 0)
      if (flag =0)   // a request packet
        if (data_described = data collected) // this is the first request packet. This has come via the best path.
          N_source = N_j;
          Decrement Timer_R;
          Store the best path in the cache;
          Set flag =1;
          Unicast the reply with data to the next node;// this reply is done along the best path
        // Now source waits for a time 't' to get a same request again.
          if ( same request is received again)
            decrement Timer_R;
            set flag = 4; // to indicate that this reply packet will not include the collected data.
            Store the second path in cache;   // a fault tolerant path is found.
            Unicast P_SPN to the next node;  // this reply is done along the fault- tolerant path.
      if (flag =3)    // some intermediate node along a path is not reachable
        if ( Source already unicast the data along the fault-tolerant path)
          Source stops the reply propagation to its neighbors; // no more reply packet is sent.
        else
          decrement Timer_R;
          Set flag =2;
          Unicast the reply with data along the second path;      // this is the fault tolerant path
  if (TTL = 0)
      Set flag = 1;
      D_coll = null;
      Decrement Timer_R; // to make sure that this reply packet is not started after the timer in sink expires.
      Unicast the reply packet to sink; // this notifies the sink that the path is still available, but  the source TTL has expired.
```

At any intermediate node

```
  while (Timer_R does not expire)
      if (flag =0)  // data request packet
          prev_node_id = N_j-1;
          decrement Timer_R;
        // Check if all the neighbors (Next nodes) are reachable
          if  (reachable)
            Broadcast the request packet
          else    // one of the neighbors is unreachable
            Stop propagating to that neighbor; // request is broadcast to all other neighbors that are reachable.
      else if (flag =1)  // reply packet with data along the first path.
          // Check if the next node is reachable
          next node id = previous node id; // the direction of propagation is reversed during the reply packet, so node ids are set correctly.
          if (reachable)
            Decrement Timer_R;
            Unicast the reply packet to the next node;
          else      // the next node is unreachable
            Set flag =3;
            Decrement Timer_R;
            Unicast to the previous node;
      else if (flag =2)    // reply packet with data along the fault-tolerant path
          next node id = previous node id; // the direction of propagation is reversed during the reply packet, so node ids are set correctly.
          // Check if the next node is reachable
          if (reachable)
            Decrement Timer_R;
            Unicast the reply packet to the next node;
          else      // the next node is unreachable
            Set flag =3;
            Decrement Timer_R;
            Unicast to the previous node;  // this packet will finally reach the source.
      else if (flag =3)    // P_NUR is received because the path to sink, via the next node is no longer available.
          Decrement Timer_R;
          Unicast the packet to the previous node;
          next node id = previous node id; // the direction is reversed during this packet propagation, so node ids are set correctly.
      else if (flag =4)    // reply packet without data along the fault-tolerant path
          // Check if thenext node is reachable
          if (reachable)
            Decrement Timer_R;
            Unicast P_SPN to the next node; // this is sent with just the path and without the collected data.
          else      // the next node is unreachable
            Set flag =3;
            Decrement Timer_R;
            Unicast to the previous node;  // this packet will finally reach the source.
```

Fig. 5 Enhanced Algorithms for the intermediate nodes and the source node

4.4 Further enhancement

The protocol can be enhanced in such a way that the sink knows the information about both the paths. To achieve this, the source node, after sending P_{reply} packet to the sink following the best path, must send another reply packet along the fault-tolerant path, without including the data collected. In this way, sink will have the information about both the paths. Later if the sink needs similar kind of data, it can contact the source first via the best path, thus avoiding the broadcast of a request. If this path is faulty, then the sink can use the fault-tolerant path to get the data from the source. Our proposed protocol has been expanded to realize this enhancement and the modified protocol uses few additional data structures. There are some minor modifications in the responsibilities of the Source node and the intermediate nodes and this has been shown in Fig. 5.

4.5 Comparison with [4] and [6]

Our protocol offers the fastest determination of the shortest path compared to the works in [4], and [6], since we use flooding. Besides, our protocol is fault-tolerant unlike the works in [4], and [6]. We have also incorporated the idea of aborting data delivery as soon as any node along a path to a sink discovers that the sink is not reachable. It offers efficient use of the wireless bandwidth and energy of the sensors nodes.

5. Conclusion

In this paper, we have presented a Fault-tolerant routing protocol for information propagation in sensor networks. The proposed protocol finds the best possible path between a source sensor and a sink sensor in the fastest possible manner (because of the flooding), unlike in the works [4], [6], where the source waits for some time to determine a best (minimum) path. Observe that in our work, the source is not involved in any kind of computation, which is not true in [4], [6]. We have also used very effective data structures for path discovery as well as for making our protocol fault-tolerant. For this purpose, the packet needs to have only the sink, source, next node id and timer value. The use of this small amount of data structures offers good utilization of the limited wireless bandwidth available in sensor networks. Thus the advantages of our proposed protocols make them efficient, robust, and suitable for the sensor networks.

References

[1] R. Min, et al., "Low Power Wireless Sensor Networks", *Proceedings of International Conference on VLSI Design*, Bangalore, India, January 2001.

[2] Mohammad Ilyas and Imad Mahgoub, *Handbook of Sensor Networks: Compact wireless and wired sensing systems*, CRC press.

[3] W. Heinzelman, “Application specific protocol architectures for wireless networks”, *PhD Thesis*, MIT, 2000.

[4] C. Intanagonwiwat, R. Govindan and D. Estrin, "Directed diffusion: A scalable and robust communication paradigm for sensor networks", *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom'00),* pp. 56-67, Boston, MA, August 2000.

[5] K. Akkaya and M. Younis, “A survey of routing protocols on Wireless Sensor Networks”, *Ad Hoc Network Journal,* pp. 325-349, 2005.

[6] C. Schurgers and M.B. Srivastava, “Energy efficient routing in wireless sensor networks”, *Proceedings on Communications for Network-Centric Operations: Creating the Information Force,* McLean, pp. 356-361, VA, 2001.

[7] I. F. Akyildiz et al., “Wireless sensor networks: a survey”, *Computer Networks*, Vol. 38, pp. 393- 422, March 2002.

Bidyut Gupta received his PhD in Computer Science and his MTech degree in Electronics Engineering from the University of Calcutta, India. Currently, he is a professor of computer science and the graduate director for Computer Science department at the Southern Illinois University Carbondale.

Shahram Rahimi received his PhD in Scientific Computing and his MS degree in Computer Science from the University of Southern Mississippi in 1998 and 2002 respectively, and his BS from National University of Iran (Tehran) in 1992. Currently, he is an assistant professor at Southern

Illinois University and the Editor-in-Chief of the
International Journal of Computational Intelligence
Theory and Practic