Identifying Deadlock Situations in Mobile Ad Hoc Networks

P. Varaprasada Rao

Associate Professor

Dept. of Computer Science & Engineering

Gokaraju Rangaraju Institute of Engineering & Technology

Hyderabad, India

ABSTRACT

Many challenges have been facing in Mobile Ad hoc networking due to frequent changes in the network topology and the lack of resources. Now a day's a lots of research is going on to support QoS in the Internet and other networks, although they are not sufficient for mobile Ad hoc networks and still QoS support for such networks remains an open problem. In this paper, a new scheme has been proposed for achieving QoS in terms of packet delivery. The proposed method adopts the snapshot algorithm of distributed systems to store information and the same will be forwarded to destination using dynamic linking. The performance of the proposed method is assessed through its low processing overhead and loop freedom.

Keywords: Deadlock, Snapshot Algorithm,MANET, QoS.

1. INTRODUCTION

Collection of mobile devices equipped with interfaces and networking capability are collectively called as mobile ad hoc wireless networks. Ad hoc can be mobile, stand alone or networked. Such type of devices can communicate with another node within their region or outside their region by multi hop techniques and each mobile node operates not only as a host but also as a router, forwarding packets for other mobile nodes in the network that may not be within direct wireless transmission range of each other. Each node participates in an ad hoc routing protocol that allows it to discover "multi-hop" paths through the network to any other node [5].

A mobile ad hoc network is also called MANET. The main characteristic of MANET strictly depends upon both wireless link nature and node mobility features. Basically this includes dynamic topology, bandwidth, energy constraints, security limitations and lack of infrastructure [2]. MANET is viewed as suitable systems which can support some specific applications as virtual classrooms, military communications, emergency search and rescue operations, data acquisition in hostile environments, communications set up in Exhibitions, conferences and meetings, in battle field among soldiers to coordinate defence or attack, at airport terminals for workers to share files etc. Several routing protocols for ad hoc networks have been proposed as DSR and AODV. Major emphasis has been on shortest routes in all these protocols in response whenever break occurs.

In this paper a new technique is proposed to avoiding deadlock situation between nodes based on snapshot algorithm. Due to the frequent changes in network topology and the lack of the network resources both in the wireless medium and in the mobile nodes, mobile ad hoc networking becomes a challenging task [1]. Effect of this technique a MANET is free from deadlock situation.

2. CLASSIFICATION OF ROUTING PROTOCOLS

A routing protocol is needed to send packets from source node to destination node. A routing protocol has to find a route for packet delivery and make the packet delivered to the correct destination [3]. Routing Protocols have been a QOS based Routing for Ad Hoc Mobile networks. Routing Protocols in Ad Hoc Networks can be categorized into two types:

2.1 Proactive Protocols

In Proactive or Table Driven routing protocols each node maintains one or more tables containing routing information to every other node in the network. All nodes keep on updating these tables to maintain latest view of the network. Some of the famous table driven or proactive protocols are: DBF [4], GSR [5].

2.2 Reactive Protocols

In Reactive or On Demand routing protocols, routes are created as and when required. When a transmission occurs from source to destination, it invokes the route discovery procedure. The route remains valid till destination is achieved or until the route is no longer needed. Some famous on demand routing protocols are: DSR [6], AODV [8].

3. SNAPSHOT ALGORITHM

The proposed scheme takes care of on detecting deadlock situation between source node and destination node based on snapshot algorithm. The Snapshot algorithm helps to MANET to detect deadlock based on maintaining state information of presenting nodes in the conversation.

The goal of this algorithm is to record a set of processes of nodes and channels states for a set of processes p_i, i=1, 2,…, N such that, even though the combination of recorded processes may have occurred at the same time, the recorded global state is consistent.

The assumptions of algorithm:

- ✓ Neither process of nodes norchannels fail.
- ✓ All channels are uni directional.
- ✓ FIFO based services are provided channels.
- ✓ The graph of channels and processes are strongly connected.
- ✓ Any processes may initiate a global snapshot at any time.

The algorithm is defined two rules, the *marker sending rule* and the *marker receiving rule*.

Algorithm

Marker receiving rule for process P_i

On P_i's receipt of a marker message over channel C:

if(P_i has not yet recorded its state)

it records its process state now;

records the state of c as the empty set;

turns on recording of messages arriving over other incoming channels;

else

Pi records the state of c as the set of messages it has received over c since it saved its state.

endif

Marker sending rule for process P_i

After P_i has recorded its state, for each outgoing channel C;P_i sends one marker message over C

(before it sends any other messages over C).

The marker sending rule obligates processes to send a marker after they have recorded their state. The marker receiving rule obligates a process that has not recorded its state to do so. In that case, this is the first marker that it has received. It notes which messages subsequently arrive on the other incoming channels. When a process that has already saved its state receives a marker (on another channel), it records the state of that channel as the set of messages it received on it's since it saved its state.

Any process may begin the algorithm at any time. It acts as though it has received a marker and follows the marker receiving rule. Thus it records its state and begins to record messages arriving over all its incoming channels, several processes may initiate recording concurrently in this way.

Illustration of the algorithm is for a system for a system of two process, p1 and p2 connected by two unidirectional channels, c1 and c2.

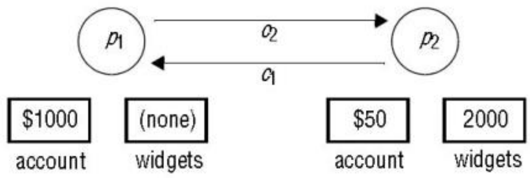

Figure 1: Two processes and their initial states

The two processes trade in 'widgets'.Process *p*1 sends orders for widgets over *c*2 to *p*2 , enclosing payment at the rate of $10per widget. Sometime later, process *p*2 sends widgets along channel *c*1 to *p*1 . Process *p*2 has already received an order for five widgets, which it will shortly dispatch to *p*1 .

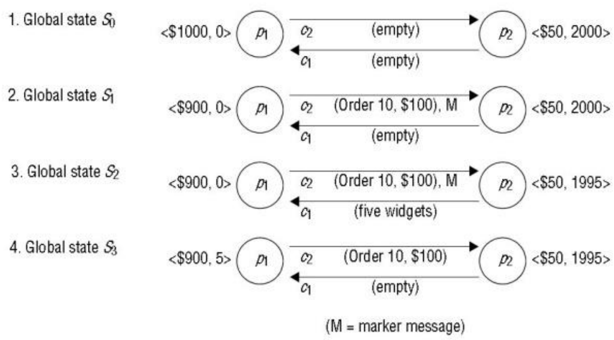

Figure 2: The execution of the processes

The above diagram shows an execution of the system while the state is recorded. Process*p*1 records its state in the actual global state *S*0, when the state of *p*1 is <$1000, 0>.Following the marker sending rule, process *p*1 then emits a marker message over itsoutgoing channel *c*2 before it sends the next application-level message: (Order 10,$100), over channel *c*2 . The system enters actual global state *S*1.

Before *p*2 receives the marker, it emits an application message (five widgets) over*c*1 in response to *p*1 's previous order, yielding a new actual global state *S*2 .

Now process *p*1 receives *p*2 's message (five widgets), and *p*2 receives themarker. Following the marker receiving rule, *p*2 records its state as <$50, 1995> andthat of channel *c*2 as the empty sequence. Following the marker sending rule, it sends amarker message over *c*1.

When process *p*1 receives *p*2 's marker message, it records the state of channel*c*1 as the single message (five widgets) that it received after it first recorded its state.The final actual global state is *S*3.

The final recorded state is *p*1 : <$1000, 0>; *p*2 : <$50, 1995>; *c*1 : <(fivewidgets)>; *c*2 : <>. Note that this state differs from all the global states through whichthe system actually passed.

Termination of the snapshot algorithm • We assume that a process that has received amarker message records its state within a finite time and sends marker messages overeach outgoing channel within a finite time (even when it no longer needs to sendapplication messages over these channels). If there is a path of communication channelsand processes from a process *pi* to a process $p_j(j \neq i)$, then it is clear on theseassumptions that *pj* will record its state a finite time after *pi* recorded its state. Sincewe are assuming the graph of processes and channels to be strongly connected, it follows that all processes will have recorded their states and the states of incoming channels afinite time after some process initially records its state.

Characterizing the observed state • The snapshot algorithm selects a cut from the historyof the execution. The cut, and therefore the state recorded by this algorithm, isconsistent. To see this, let *ei* and *ej* be events occurring at *pi* and *pj* , respectively, suchthat *ei*→ *ej* . We assert that if *ej* is in the cut, then *ei* is in the cut. That is, if *ej* occurredbefore *pj* recorded its state, then *ei* must have occurred before *pi* recorded its state.This is obvious if the two processes are the same, so we shall assume that $j \neq i$. Assume,for the moment, the opposite of what we wish to prove: that *pi* recorded its state before*ei* occurred. Consider the sequence of *H* messages *m*1, *m*2,…, *mH*((*H* ≥ 1), giving riseto the relation *ei*→*ej* . By FIFO ordering over the channels that these messagestraverse, and by the marker sending and receiving rules, a marker message would havereached *pj* ahead of each of *m*1,*m*2,…,*mH* . By the marker receiving rule, *pj* wouldtherefore have recorded its state before the event *ej* . This contradicts our assumptionthat *ej* is in the cut, and we are done.

We may further establish a reachability relation between the observed global stateand the initial and final global states when the algorithm runs. Let *Sys*= *e*0,*e*1,….bethe linearization of the system as it executed (where two events occurred at exactly thesame time, we order them according to process identifiers). Let *Sinit* be the global stateimmediately before the first process recorded its state; let *Sfinal* be the global state whenthe snapshot algorithm terminates, immediately after the last state-recording action; andlet *Ssnap* be the recorded global state.

We shall find a permutation of *Sys*, $Sys' = e'_0, e'_1, e'_2, \ldots$ such that all three states*Sinit* , *Ssnap* and *Sfinal* occur in *Sys*' , *Ssnap* is reachable from *Sinit* in *Sys'* , and *Sfinal*is reachable from *Ssnap* in *Sys'*.

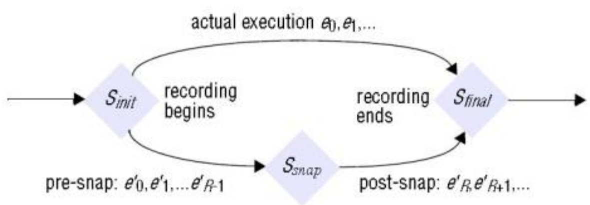

Figure 3: Reachability between states in the snapshot algorithm

The above diagram shows this situation, in which the upperlinearization is *Sys* and the lower linearization is *Sys'*.

We derive *Sys'* from *Sys* by first categorizing all events in *Sys* as *pre-snap*events or *post-snap* events. A pre-snap event at process *pi* is one that occurred at *pi*before it recorded its state; all other events are post-snap events. It is important tounderstand that a post-snap event may occur before a pre-snap event in *Sys* , if the eventsoccur at different processes. (Of course, no post-snap event may occur before a pre-snapevent at the same process.)

We shall show how we may order all pre-snap events before post-snap events toobtain *Sys'*. Suppose that $e_j$is a post-snap event at one process, and e_{j+1} is a pre-snap event at a different process. It cannot be that $e_j \rightarrow e_{j+1}$ for then these two events wouldbe the sending and receiving of a message, respectively. A marker message would haveto have preceded the message, making the reception of the message a post-snap event,but by assumption e_{j+1} is a pre-snap event. We may therefore swap the two eventswithout violating the happened-before relation (that is, the resultant sequence of eventsremains a linearization). The swap does not introduce new process states, since we donot alter the order in which events occur at any individual process.

We continue swapping pairs of adjacent events in this way as necessary until wehave ordered all pre-snap events $e'_0, e'_1, e'_2, \ldots e'_{R-1}$ prior to all post-snap eventse$e'_R, e'_{R+1}, e'_{R+2}, \ldots$with *Sys'* the resulting execution. For each process, the set ofevents in $e'_0, e'_1 e'_2, \ldots, e'_{R-1}$ that occurred at it is exactly the set of events that itexperienced before it recorded its state. Therefore the state of each process at that point,and the state of the communication channels, is that of the global state *Ssnap* recordedby the algorithm. We have disturbed neither of the state's*Sinit* or *Sfinal* with which thelinearization begins and ends. So we have established the reachability relationship.

Stability and the reachability of the observed state • The reachability property of thesnapshot algorithm is useful for detecting stable predicates. In general, any non-stablepredicate we establish as being *True* in the state *Ssnap* may or may not have been *True*in the actual execution whose global state we recorded. However, if a stable predicate is*True* in the state *Ssnap* then we may conclude that the predicate is *True* in the state *Sfinal* ,since by definition a stable predicate that is *True* of a state *S* is also *True* of any statereachable from *S*. Similarly, if the predicate evaluates to *False* for *Ssnap* , then it mustalso be *False* for *Sinit* .

Global states

The examplesof distributed garbage collection, deadlock detection, termination detection anddebugging:

Distributed garbage collection: An object is considered to be garbage if there are nolonger any references to it anywhere in the distributed system. The memory taken upby that object can be reclaimed once it is known to be garbage. To check that anobject is garbage, we must verify that there are no references to it anywhere in thesystem.

Distributed deadlock detection: A distributed deadlock occurs when each of acollection of processes waits for another process to send it a message, and wherethere is a cycle in the graph of this 'waits-for' relationship.

Distributed termination detection: The problem here is how to detect that adistributed algorithm has terminated. Detecting termination is a problem that soundsdeceptively easy to solve: it seems at first only necessary to test whether each processhas halted. To see that this is not so, consider a distributed algorithm executed by twoprocesses $p1$ and $p2$, each of which may request values from the other.Instantaneously, we may find that a process is either active or passive – a passiveprocess is not engaged in any activity of its own but is prepared to respond with avalue requested by the other. Suppose we discover that $p1$ is passive and that $p2$ is passive. To see that we may not conclude that the algorithm hasterminated, consider the following scenario: when we tested $p1$ for passivity, amessage was on its way from $p2$, which became passive immediately after sendingit. On receipt of the message, $p1$ became active again – after we had found it to bepassive. The algorithm had not terminated.

Distributed debugging: Distributed systems are complex to debug, and care needs to be taken in establishing what occurred during the execution.For example, suppose Smith has written an application in which each process $p_i$contains a variable x_i($i = 1, 2,\ldots,N$). The variables change as the programexecutes, but they are required always to be within a value δ of one another.Unfortunately, there is a bug in the program, and Smith suspects that under certaincircumstances $|xi - xj| > \delta$ for some i and j, breaking her consistency constraints. Herproblem is that this relationship must be evaluated for values of the variables thatoccur at the same time.

Each of the problems above has specific solutions tailored to it; but they all illustrate theneed to observe a global state, and so motivate a general approach.

It is possible in principle to observe the succession of states of an individual process, butthe question of how to ascertain a global state of the system – the state of the collectionof processes – is much harder to address.The essential problem is the absence of global time. If all processes had perfectlysynchronized clocks, then we could agree on a time at which each process would recordits state – the result would be an actual global state of the system. From the collection ofprocess states we could tell, for example, whether the processes were deadlocked. Butwe cannot achieve perfect clock synchronization, so this method is not available to us.So we might ask whether we can assemble a meaningful global state from localstates recorded at different real times.

The answer is a qualified 'yes', but in order to seethis we must first introduce some definitions.

Let us return to our general system$\wp$of N processes pi ($i = 1, 2, \ldots ,N$), whoseexecution we wish to study. We said above that a series of events occurs at each process,and that we may characterize the execution of each process by its history:

$$history(p_i)=h_i=<e^i_0,e^i_1,e^i_2, \ldots\ldots>$$

Similarly, we may consider any finite prefix of the process's history:

$$H^i_0=<e_i^0,e_i^1, \ldots , e_i^k>$$

Each event either is an internal action of the process (for example, the updating of oneof its variables), or is the sending or receipt of a message over the communicationchannels that connect the processes.

In principle, we can record what occurred in$\wp$'s execution. Each process canrecord the events that take place there, and the succession of states it passes through. Wedenote by $s_i^k$the state of process $p_i$immediately before the kth event occurs, so that s_i^0 is the initial state of pi. We noted in the examples above that the state of thecommunication channels is sometimes relevant. Rather than introducing a new type ofstate, we make the processes record the sending or receipt of all messages as part of theirstate. If we find that process *pi* has recorded that it sent a message *m* to process$p_j(i \neq j)$, then by examining whether *pj* has received that message we can infer whetheror not *m* is part of the state of the channel between *pi* and *pj*.

We can also form the *global history* of℘as the union of the individual processhistories:

$$H = h_0 \cup h_1 \cup \ldots \cup h_{N-1}$$

Mathematically, we can take any set of states of the individual processes to form a globalstate$S = (s1, s2, \ldots, sN)$. But which global states are meaningful – that is, whichprocess states could have occurred at the same time? A global state corresponds to initialprefixes of the individual process histories. A *cut* of the system's execution is a subsetof its global history that is a union of prefixes of process histories:

$$C = h_1^{c1} \cup h_2^{c2} \cup, \ldots, \cup h_N^{cN}$$

The state *si* in the global state *S* corresponding to the cut *C* is that of *pi* immediatelyafter the last event processed by *pi* in the cut – e_i^{ci}($i = 1,2, \ldots, N$). The set of events$\{e_i^{ci}: i = 1, 2, \ldots, N\}$ is called the *frontier* of the cut [7].

4. PROPOSED METHOD

The routing protocols provide a route but not detect a deadlock situation. The snapshot algorithm detects a deadlock situation between source and destination.

The channels in Ad-Hoc networks will be added dynamically. Even though, the processes identified by "snapshot" algorithm will be forwarded through, by linking the channels dynamically.

The snapshot algorithm detects the deadlock situation in between nodes and the information maintained in separate file similar to log file. By verifying this by file one can assess and avoids the sending information to other node in the channel.

5. CONCLUSION

A new scheme has been proposes to detect deadlock situation between source and destination. This can be incorporated effectively in MANETs to improve low processing overhead and loop freedom.

6. REFERENCES

[1] National Science foundation, "Research priorities in Wireless and mobile networking", Available at www.cise.nsf.gov.

[2] E.M. Royer and C.K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks". IEEE Personal Communications, pages 46–55, April 1999.

[3] Andrew TanenBaum, "Computer Networks", Prentice Hall, New Jersey, 2002.

[4] Tsu-Wei Chen, Mario Gerla, "Global State Routing: A New Routing Scheme for Ad-hoc Wireless Networks"Proceedings IEEE ICC 1998.

[5] S. Murthy, J.J. Garcia-Luna-Aceves, "An Efficient Routing Protocol for Wireless Networks", ACM Mobile Networks and App. Journal, Special Issue on Routing in Mobile Communication Networks, Oct. 1996, pp. 183-97.

[6] Navid Nikaein, Christina Bonnet, "Dynamic Routing algorithm", available at Institute Eurecom,Navid.Nikaein@eurocom.fr .

[7] George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair, "Distributed Systems Concepts and Design", 5th Edition,Pearson, 2012.

[8] NS notes and documentation www.isi.edu/vint.