A Tunneling Approach to Routing with Unidirectional Links in Mobile Ad-Hoc Networks

Sanket Nesargi **Ravi Prakash**
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083-0688.
e-mail: {sanket,ravip}@utdallas.edu

Abstract— **Mobile ad hoc networks (MANETs) consist of a set of similar mobile nodes, communicating with each other using wireless links. As a node may not be able to directly reach every other node, a packet may need to traverse multiple wireless links from its source to its destination. Unidirectional links can occur in such networks. Several existing routing protocols implicitly assume bidirectional links when making their routing decisions. Not using unidirectional links can lead to sub-optimal routes. A tunneling solution to allow efficient operation in ad hoc networks with unidirectional links is presented. The tunneling solution uses information gathered by the routing protocol to tunnel packets from the end-point of the unidirectional link to its source. A naive implementation of tunneling could lead to loops in the system and a deluge of packets. This is because when ACKs for link layer messages are tunneled across unidirectional links, ACKs for them may end up being generated recursively. The solution presented here prevents such a packet explosion from occurring.**

Keywords: mobile ad hoc networks, routing protocols, protocol analysis and design.

I. Introduction

A mobile ad hoc network (*MANET*) is composed of a set of similar mobile nodes, communicating with each other over wireless links. The communication range of each node is finite, due to which two non-neighbor nodes need multiple hops to communicate with each other. As the mobility pattern of the nodes is often non-deterministic, the network topology is always in a state of flux and paths between node pairs may change significantly over time. Each node has to have the capability to route packets to any other node. Such networks have been studied in the past in relation to defense research, often under the name of *Packet Radio Networks*.

Significant work has been done in the development of routing protocols for these networks. Internet drafts exist for protocols such as the *Ad Hoc On Demand Distance Vector Routing (AODV)* [12], *Dynamic Source Routing (DSR)* [7], Temporally-Ordered Routing Protocol (TORA) [9], Zone Routing Protocol (ZRP) [10], etc. These algorithms have contributed towards the understanding of the routing problem and the feasible solution approaches. However, most of these (except DSR) implicitly assume the links to be “symmetric” or “bidirectional”. As explained in [14], some routing protocols may make erroneous routing decisions if they do not account for the presence of unidirectional links.

The presence of unidirectional links in the system is a manifestation of the signal propagation models in use [5]. The reasons for the presence of these links, and the impact that they have on the different network layers is analyzed in Section II.

The layer at which the unidirectional property of a link starts manifesting itself is the data-link layer. At this layer, due to the unidirectional nature of the link, acknowledgments (ACKs) cannot be sent in the reverse direction. So, none of the conventional flow control and reliability mechanisms like the sliding window protocols can be used. However, if the network is strongly connected (in the graph theoretic sense) and all nodes can reach all other nodes, connectivity information available at the network layer can be used to send the MAC layer acknowledgements as described in [3]. If this approach is adopted without care, it could potentially cause looping and packet explosion as described in Section IV-C.

The rest of the paper is organized as follows. In Section II, the reasons for the presence of the unidirectional links and their impact on routing is analyzed. Section III describes the system model and the assumptions under which the proposed solution would work. In Section IV, the tunneling approach, the motivation behind it and the proposed protocol are discussed in detail. The modifications required in the packet headers to support tunneling are presented in Section V. We analyze our approach and the effects it has on the timeout values, windows sizes, etc., in Section VI. Section VII compares and contrasts our approach with existing approaches to routing packets over unidirectional links. Section VIII presents the conclusions and the directions for future work.

II. Occurrence and Impact of Unidirectional Links

Unidirectional links arise in wireless networks due to a variety of reasons and their presence affects the performance of the network in various ways. In this section we describe the reasons for the occurrence of unidirectional links and their manifestations in ad hoc networks.

A. Occurrence of Unidirectional Links

There are a few scenarios under which unidirectional links can occur

1. Some links may be unidirectional due to the *hidden terminal problem* [15]. Node *A* may be able to receive messages from node *B* due to little interference in *A*'s vicinity. However, *B* may be in the vicinity of an interfering node and, therefore, unable to receive *A*'s messages. Thus, the link between *A* and *B* is directed from *B* to *A*. In this case, this unidirectionality may be a *transient* phenomenon, which lasts only as long as the interference around *B* persists. The link would become bidirectional if the interference around *B* reduced.
2. Links may become unidirectional due to disparity between the transmission power levels of the nodes at either ends of the link. In Figure 1, the transmission range of nodes operating at a normal energy level is d. However, the energy level at a node can decrease due to reasons such as excessive local utilization or being chosen as a cluster head by routing algorithms causing more packet transmissions than normal. In such cases, the node can choose to relay packets only to nodes within a shorter range, say d'. Thus, node *A* operating at a depleted energy level, can

send and receive packets from B as the distance AB is less than d'. However, since the distance AC is greater than d', A cannot send packets to C. Node C, operating at a normal energy levels can still send packets to A as the distance AC is less than d. This results in the creation of a unidirectional link, directed from C to A. Such unidirectional links would be long-lived.

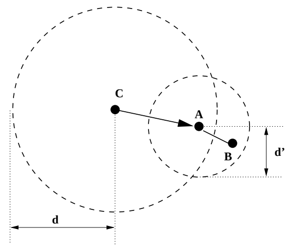

Fig. 1. Range Induced Unidirectionality

B. Impact of Unidirectional Links

The presence of unidirectional links impacts the following layers in the network protocol stack:

- *Data Link Layer*
- *Network Layer*

At the data link layer, error and flow control protocols employ acknowledgments (ACKs) for the data packets exchanged between adjacent nodes. However, due to the unidirectional nature of the links, ACKs cannot travel back to the sender. As a result, the sender would time out and assume that the link between itself and the destination is down which would result in the link not being used for further communication.

At the network layer, routing protocols like DSDV [11], AODV [12] and TORA [9] assume the links to be bidirectional. In these algorithms, information is exchanged between the neighbors to get information about the network topology. The unidirectional links cause:
1. *Knowledge Asymmetry:* In case a unidirectional link exists between nodes i and j, just because j can hear from i, j cannot assume that i can also hear from it. Node i does not hear from j at all, and could assume that no link exists between them. Thus, there is an asymmetry of knowledge between the nodes at the two ends of the unidirectional link.
2. *Routing Asymmetry:* For a unidirectional link between i and j, the path traversed to reach from i to j is different from the path required to reach i from j, leading to an asymmetry in routing.

Thus, the problems caused by the presence of unidirectional links can be summarized as
1. Some frames at the data link layer cannot be acknowledged.
2. The node into which the unidirectional link is incident cannot directly send its reachability information (as in a distance-vector based protocol) to the node from which the link is incident.

III. System Model and Assumptions

The network consists of identical nodes capable of communicating with each other across a wireless interface. Links between the nodes can be unidirectional or bidirectional. These can change their orientation depending upon interference, power levels of the nodes, etc. We assume that the network is strongly connected, $i.e.$, there exists a path from every node to every other node. The node that can transmit along the unidirectional link will henceforth be referred to as the *source* or the *head* of the link. The node towards which the unidirectional link is directed, will be known as the *sink* or *target* of the link. As shown in Figure 2, though there is a unidirectional link between B and C, an alternate path $CDEFGAB$ exists from C to B. If the unidirectional link is not considered, even the forward path between B and C would require six hops, as compared to the single hop path that exists via the unidirectional link.

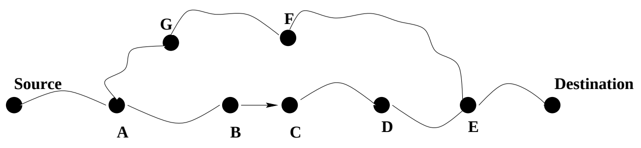

Fig. 2. Representation of Directed and Undirected Paths

IV. Tunneling Approach

We propose tunneling the link layer acknowledgment packets and the routing algorithm control packets used for topology detection. Our approach is similar to [3] proposed for traditional wired networks, with modifications made to cater to ad hoc networks. This approach solves the problem at the data link layer, using tunneling to send packets from the sink to the source. The unidirectionality of the link is made oblivious to the network and above layers. Hence, as viewed by the network layer, data, control and ACK packets adopt the same path in both directions between the source and the sink of the unidirectional link.

A. Motivation

The main motivation in adopting the tunneling approach is to provide transparency for
1. Link-Layer ACKs
2. Exchange of network-layer routing information between neighbors
The presence of the unidirectional link can be detected only by the sink, and the source is unaware of its existence. Hence, if the routing protocol needs to make use of this link, information about its existence must be propagated to its source. This routing information typically consists of the reachability information of the sink, which can be used by the source to update the reachability information in its routing table. In case this information is not tunneled across, the source would not come to know about the existence of the link and not use it in its routing decisions. Also, when data flows across this link, link-layer ACKs need to be exchanged for flow and error control. This is facilitated by tunneling the ACKs back to the source.

Also, in case of a unidirectional link, the sink gets routing updates from the source. These updates are of little use to the sink as the source cannot be reached directly along this link. Hence, care has to be taken to avoid using these routing updates at the sink. As the knowledge about the unidirectionality of the link is maintained both at the data link and the network layers, the latter can ignore all routing update packets from the source to the sink.

B. Solution

The solution consists of three parts
1. *Detection of unidirectional links*
2. *Tunneling routing information*
3. *Tunneling acknowledgments for data packets*

B.1 Detection of Unidirectional Links

At the network layer, each node broadcasts a message *hello* (similar to the AODV *Hello* message) once every `hello_interval` milliseconds. This is a message with a TTL (time to live) of 1 so that it is only exchanged between neighbors. In this message, each node broadcasts the list of the neighbors it is currently receiving *hello* messages from. This is used by the nodes to build the neighborhood information. In our approach, we additionally use it to detect the presence of unidirectional links. When a node receives a *hello* message from a neighbor, and itself is not present in the neighbor's neighborhood list information, the node can conclude that it is the sink of a unidirectional link. This is similar to the method used in [1] to gain neighborhood information which is maintained at the network layer. The link layer, too, is notified of this attribute of the link. However, there is no way for the source to know about the directionality of this link. This information has to be conveyed by the sink to the source.

Thus, to convey the existence of the unidirectional link to its source a *link_inform* message is tunneled from the sink node to the source. When this packet is received at the source, the network layer and the data link layer make a note of the unidirectional property of the link. This packet also carries an expiration timeout value, which is the time for which the link is considered active. This packet has to be tunneled every `link_inform_timeout` milliseconds, to prevent the link from being marked inactive. After this packet is received, both the source and the sink of the unidirectional link are aware of its existence.

B.2 The Tunneling Approach

The approach adopted to tunnel the information is the same for all types of packets. As the network is assumed to be strongly connected, there exists a path from the sink to the source of a unidirectional link. This path is used to tunnel route updates and link layer acknowledgements from the sink of the unidirectional link to its source. For this purpose, information is encapsulated within a new network layer packet destined for the source of the unidirectional link.

Our approach reduces the overhead traffic on intermediate nodes by using *selective tunneling*. It does not tunnel all packets from the destination to the source, but tunnels only ACKs and routing information packets. The compromise made to achieve this is that the approach is not entirely transparent to the network layer. However, these modifications are still transparent to the routing module, thereby allowing any routing protocol to be used. Both the data link and network layer are aware of the unidirectionality of the link. The advantage of this approach is that routing protocols can make use of the unidirectional link to transmit data as well as control information in the forward direction.

In Figure 3 packets from A to B are sent along the unidirectional link. The ACKs for these packets and the routing information packets to be sent to A are encapsulated in a network layer header, with source as B and destination as A. These packets are then routed to A based on the information gathered by the routing protocol being used. Upon reception at A, the outer header is stripped off and the inner one examined. If the packet is a link layer ACK, then it is pushed down to the link layer for further processing. In case it is a packet containing routing information, it is passed to the routing module for processing. This determination of whether the tunneled packet is an ACK or a routing packet, is done on the basis of an option set in the IP header. The details of this are provided in Section V.

An important issue that needs to be considered in this approach is that looping and packet explosion should be avoided at all costs. The cause and impact of the same are discussed in Section IV-C. Also, B should not use routing updates received from A to update its own routing table as it cannot reach A via this link.

Tunneling of Routing Information: Once the existence of the unidirectional link has been established, routing information from the sink to the source has to be tunneled as the source is ignorant of the destinations that can be reached optimally via the sink. In a protocol like DSDV, the reachability information is periodically broadcast to the neighbors and the network topology diffused through the network. In such a scenario, to make the source aware of the paths, the reachability information of the sink has to be tunneled.

A possible optimization would involve tunneling of only those routes which would be of benefit to the source. This would help in reducing the control traffic on the links. As the sink knows about the routing table at the source (via its routing updates), it can compute the destinations for which routing through it would be more efficient. Information about only these nodes is propagated to the source. This, however, compromises the transparency of the tunneling approach to the routing protocol in use. Also, the stability of the routing information may be in question if tunneled updates do not make it to the source of the unidirectional link in time. This is analyzed in Section VI.

These *link_route_update* packets would be instrumental in letting the source know about the possible routes through the sink. The exact content of these packets would depend upon the routing protocol in use, and what information needs to be sent across. At the receiving end, these packets are used by the source to update its routing table.

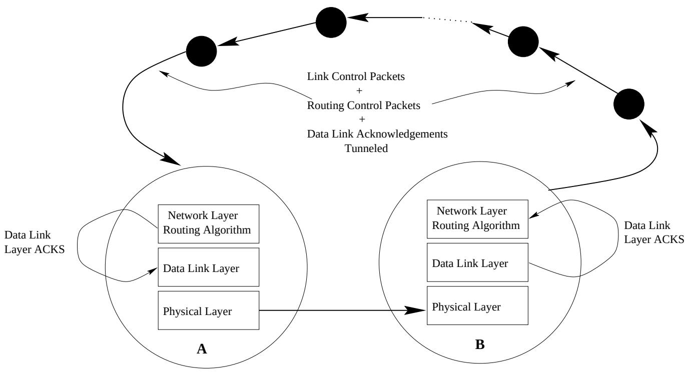

Fig. 3. Tunneling of Packets

```
on_hello_interval_timeout()
{
 neighbor_list = build_neighbor_list()
 hello = create_hello_packet()
 broadcast_packet(hello)
}

on_hello_message(hello)
{
 hello_src = get_source_addr(hello)
 hello_nbr_list[hello_src] = get_nbr_list(hello)
 local_addr = get_local_addr()
 if (is_present_in_list(hello_nbr_list[hello_src]
                             local_addr))
     return
 set_unidirectional_sink(hello_src, local_addr)
 link_inform = create_link_inform(hello_src,
                             local_addr)
 tunnel_message(link_inform, hello_src)
}

on_link_inform(link_inform)
{
 local_addr = get_local_addr()
 link_inform_source = get_source_addr(link_inform)
 set_unidirectional_source(local_addr,
                               link_inform_source)
}
```

Fig. 4. Pseudo-code for Unidirectional Link Detection

Link Layer

```
// received from network layer
send_packet(packet)
{
 sink_mac_addr = get_sink_mac_addr(packet, arp_cache)
 local_mac_addr = get_local_mac_addr()
 outgoing_packet = replicate_packet(packet)
 // check if it is link layer info (which was tunneled)
 if (is_link_layer_ack(get_payload(packet)) {
   cancel_ack_timer(packet)
   return
 }
 if (is_tunneled_packet(packet))
   set_link_layer_tunnel_flag(outgoing_packet)
 if(is_link_unidirectional(local_mac_addr, sink_mac_addr))
   // don't ensure reliability
 else {
   // normal reliability mechanism
   start_ack_timer(packet)
 }
 forward_to_physical_layer(outgoing_packet)
}

// received from physical layer
recv_packet(packet)
{
 src_mac_addr = get_src_mac_addr(packet)
 local_mac_addr = get_local_mac_addr()
 if (is_link_unidirectional(src_mac_addr, local_mac_addr)) {
    if (is_tunneled_packet(packet))
       // generate no ack
    else
       tunnel_ack(packet, local_mac_addr, src_mac_addr)
 } else
    generate_ack(packet, local_mac_addr, src_mac_addr)
 forward_to_network_layer(packet)
}
```

Network Layer

```
// received from transport layer or for fowarding
send_packet(packet)
{
 outgoing_packet = replicate_packet(packet)
 if (is_tunneled_packet(packet))
    set_network_layer_tunnel_flags(outgoing_packet)
 forward_to_link_layer(outgoing_packet)
}

// received from data-link layer
recv_packet(packet)
{
 sink_addr = get_sink_addr(packet)
 src_addr = get_src_addr(packet)
 local_addr = get_local_addr()
 // handle only packets for this node, else forward
 if (sink_addr != local_addr)
    forward_to_next_hop(packet)
 if (! is_tunneled_packet(packet))
    forward_to_transport_layer(packet)
 // received a tunneled packet
 tunnel_payload = get_tunnel_payload(packet)
 if (is_tunneled_ack(tunnel_payload))
    send_to_link_layer(tunnel_payload)
 if (is_routing_info(tunnel_payload) &&
      !is_link_unidirectional(src_addr, sink_addr))
    send_to_routing_module(tunnel_payload)
}
```

Fig. 5. Pseudo-code for Tunneling

B.3 Tunneling of Acknowledgments

The data packets are sent across the wireless link from the source to the sink triggering link layer acknowledgments to the source. The flow and error control mechanisms for these packets are implemented using sliding window protocols [16], which transmit a window of packets and then wait for acknowledgments. If the acknowledgments are not received within a timeout period, the lost packets are retransmitted. *In the proposed solution, as the link layer is aware of the unidirectional nature of the link, timeout values are increased.* This is to accommodate the latency incurred by ACKs having to travel across multiple hops to reach the source. The size of the window could also be increased correspondingly. An analysis of the changes needed in these values is presented in Section VI.

Thus, network layer messages referred to as *link_tunnel_acks* are generated. These messages encapsulate the link acknowledgments and deliver them to the source. At the source, these messages are decapsulated, identified as link layer ACKs (as described in Section V and then passed to the data link layer to ensure proper functioning of the sliding window protocols.

Tunneling of ACKs can potentially result in looping of packets and explosion of network traffic. A solution to this problem is described next.

C. Loop Freedom

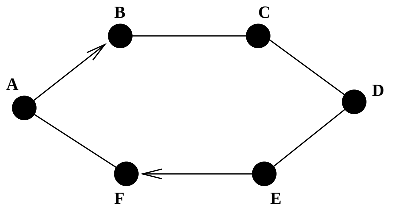

Fig. 6. Multiple Unidirectional Links

If the path across which data link layer ACKs are tunneled also contains unidirectional links, additional ACKs may get generated for these ACKs which are being tunneled. As shown in Figure 6, when a packet is being sent from node A to node B across the unidirectional link AB, packets containing ACKs from B to A have to be tunneled via $BCDEFA$. This path also contains a unidirectional link EF. When an ACK traverses across the link EF, node F may treat it as an ordinary link layer message for which a link layer ACK needs to be sent to E. Such an ACK will be tunneled via $FABCDE$. As this packet traverses from A to B, at the link layer, ACKs again need to be tunneled from B to A. Hence, an explosion of ACKs would result and clog the network which is already resource poor.

To avoid this problem, *we do not generate link layer acknowledgments for tunneled packets received on an incoming unidirectional link*. Thus, when a data link layer at F receives an encapsulated packet from E, it does not generate any acknowledgment for it. As a result, packets can be tunneled across paths which themselves contain unidirectional links, without any looping problems. In the network shown in Figure 6, the tunneled ACKs from B to A across $BCDEFA$, would not generate any further ACKs, avoiding the ACK explosion problem entirely. To avoid ACK generation for tunneled packets, certain modifications need to be made to the data link as well as the network layer headers. These modifications and the handling of tunneled packets is described in Section V.

V. Packet Header Modifications to Support Tunneling

In this section, we describe the modifications made to the packet headers to support the tunneling approach and avoid ACK generation for tunneled packets. The link layer at the intermediate nodes should be aware that the packet being processed by it is actually a tunneled packet. If the headers of the tunneled packet are not modified, the link layer would have to look into the payload of each packet passing through it to determine if it is a tunneled packet. For the network topology depicted in Figure 6, the view of a tunneled ACK from B to A at F would be as shown in Figure 7. At the data link layer of F, the packet looks as shown in Figure 7. In order to determine that the packet is a tunneled one, the data link layer of F would have to look past the data link and the network layer headers of the packet, into its payload and search for an ACK packet, or a routing information packet. This overhead at the data link layer, for each packet traversing through it, is unacceptable and violates the principles of protocol layering.

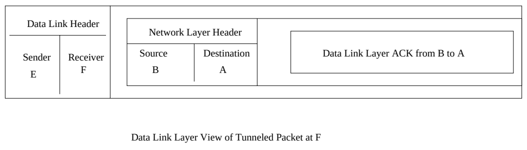
Data Link Header
Sender E
Receiver F
Network Layer Header
Source B
Destination A
Data Link Layer ACK from B to A
Data Link Layer View of Tunneled Packet at F

Fig. 7. Packet View at Network and Data Link Layers of Intermediate Nodes

The ACKs for the packets are generated at the data link layer and later passed to the routing module of the network layer. Hence, both these layers should know that the packet is a tunneled one, and replicate that information into the outgoing packets. The packet headers at the data link as well as the network layer need to be modified to support a flag indicating whether a packet is a tunneled one or not. At the network layer, this flag can be added as an option to the IP header, beyond the 20 byte standard header. If IEEE 802.11 [5] is being used at the MAC and physical layers, this flag could be added as one of the options in the Frame Control field. In addition, after stripping the outer header, the packet is either forwarded to the routing module or the link layer. Link layer ACKs are forwarded to the link layer and routing updates to the routing module. The network layer at the receiving node should be able to distinguish between these two types of packets easily, without having to go into the payload to determine the type. This is accomplished by adding an option in the network layer header which indicates whether a tunneled packet is a routing update or a data link layer acknowledgement. The value of this flag is used to determine the module which receives the packet.

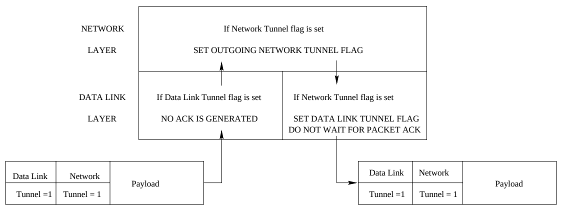
NETWORK
LAYER
If Network Tunnel flag is set
SET OUTGOING NETWORK TUNNEL FLAG
DATA LINK
LAYER
If Data Link Tunnel flag is set
NO ACK IS GENERATED
If Network Tunnel flag is set
SET DATA LINK TUNNEL FLAG
DO NOT WAIT FOR PACKET ACK
Data Link
Tunnel =1
Network
Tunnel = 1
Payload
Data Link
Tunnel =1
Network
Tunnel = 1
Payload

Fig. 8. `tunnel` Flag Processing

The processing of a tunneled packet is shown in Figure 8. On packet reception at the data link layer of an intermediate node, it is seen that the packet being processed has the `tunnel` flag set in its data link header. This suppresses ACK generation for this packet, *if it has been received on an incoming unidirectional link*. Thereafter, the packet is handed over to the network layer. As the IP header size is greater than 20 bytes, options are checked to determine if the packet is a tunneled one. In case it is, this flag is replicated in the IP header of the outgoing packet before it is dispatched to the data link layer (if the processing node is not the final destination). The option indicating whether that packet is a tunneled ACK or routing update, is also replicated into the header of the outgoing packet. The data link layer replicates the `tunnel` flags set in the network layer into appropriate fields in the data link header. It also ensures that *if an outbound tunneled packet is leaving on a unidirectional interface*, ACKs for it are not awaited. Thus, this packet is excluded from the sliding window protocols ensuring reliability at data link layer. *The source does not await ACKs for tunneled packets, but ACKs for data packets are waited for*.

This ensures that ACK flooding is stemmed without excessive overheads.

VI. Analysis of the Tunneling Approach

In this section we analyze the proposed approach based on the effect it has on the size of the link layer sliding windows and the timers associated with packets. The impact on the frequency of routing updates in both pro-active and reactive algorithms is also analyzed.

A. Impact on Link Layer Windows and Timers

Link layer employs sliding window protocols to ensure reliability of packet delivery between adjacent nodes. When this scheme is applied across a unidirectional link using tunneling to propagate ACKs along the return path, propagation delays in the forward and reverse directions may vary considerably. This asymmetry in propagation times should be considered in determining the values of the link layer windows and associated timers.

We propose to use the round trip time (RTT) of the packet in determining the windows sizes and timer values. We employ an approach similar to the one adopted for TCP in [6]. Tunneling of packets is used to simulate a single link over multiple hops in the reverse direction and in some sense is similar to a TCP connection as it has end to end semantics.

The initial value of RTT is $(1 + D)t$ where D is the diameter of the network, and t is the propagation delay in sending a packet of length L to an adjacent neighbor. Thereafter, its value adapts according to the equation

$$RTT = \alpha RTT + (1 - \alpha) RTT_{last}, \quad \alpha < 1$$

where RTT_{last} is the RTT of the last packet acknowledged. α is a constant chosen to set the importance of history on the RTT estimate.

Thus, in such a scenario, if the rate of data flow across a unidirectional link is d bits/sec, the timer values and window sizes would be

$$\textit{Timeout Duration} = \beta \times \textit{RTT} \text{ seconds}$$

This is the same as the approach adopted in the initial versions of TCP.

$$\textit{Window Size} = \omega \times \frac{RTT \times d}{L} \text{ packets}$$

Here ω is a constant lesser than 1. Any value of ω larger than 1 would overload the network. This value has to ensure that the link does not remain idle while acknowledgements for previously transmitted packets are awaited. It also should ensure that the number of packets awaiting acknowledgements is not too high, as these would clog up the network in case the packet drop rate (due to errors) is high (as is typically the case in wireless networks).

B. Impact on Routing Updates

When routing updates are exchanged between adjacent nodes, it is assumed that when the same is received at the target node, the information contained in them is still valid and can be used to route packets. If a link is stable for ϕ seconds, it can be used to determine the frequency and freshness of routing updates in both pro-active and reactive routing protocols. ϕ is an estimate of how long a particular link would exist, based on the signal strength, fading characteristics, interference and other physical layer attributes *at the time of sampling*. This could be computed by making an estimate of how long it would take for the signal strength to fall below the levels needed for transmission.

When a routing update has to be sent across a bidirectional link, it would take time t to get there. This information is valid for $\phi - t$ seconds after it reaches its target node. To ensure that the routing updates are "fresh" and up to date when they reach their destinations, they should be generated at a rate greater than $\frac{1}{\phi - t}$. However, when the updates are sent upstream over a unidirectional link, information needs to be tunneled from the sink to the source which takes δ seconds to reach the source. Thus, the information, when received at the source, is fresh only for $\phi - \delta$ seconds. Thus, to ensure freshness, the frequency of routing updates has to be increased to at least $\frac{1}{\phi - \delta}$.

The generation of routing updates depends upon the type of routing protocol in use. These routing protocols could be either pro-active (such as DSDV [11]) where the topology information is periodically exchanged, or reactive (such as AODV[12] or DSR[7]). In reactive protocols, route computation is performed only in response to a request to route a packet from a source to a destination.

In the case of pro-active routing protocols, the frequency of route updates is increased to $\frac{1}{\phi - \delta}$, as stated earlier. An adaptive value for δ can be estimated as $RTT - t$ as it changes according to network stability. The estimates of RTT are maintained for every unidirectional interface present on the node. When a node has no unidirectional interfaces attached to it, it could be generating routing updates at a rate slightly higher than $\frac{1}{\phi - t}$. However, when any of the incoming interfaces on the node become unidirectional, the rate of routing update generation is increased to $\frac{1}{\phi - RTT + t}$. The rate of routing update generation is increased across all links, and not just the unidirectional ones to maintain the transparency of the routing protocols to the tunneling approach adopted.

In reactive protocols, route discovery packets are sent to determine the path to a destination in response to a data packet arriving for it. The responses to these discovery packets may need to be tunneled in case the original packets were received over a unidirectional link. It would make sense to tunnel the packets only if only if $\phi - RTT + t$ has a value significantly larger than t so that it can be used for at least some data transmission. When the sink determines that information carried by the response would expire before it can reach the source of the unidirectional link, the packet is not propagated. The source would assume that a route via this link was not discovered and resort to some other route.

VII. Comparison with Existing Solutions

In [14], the knowledge about the unidirectionality of links is maintained and propagated by exchanging messages containing information about the nodes that can be reached from a particular node as well as the nodes a particular node can be reached from. If there are n nodes in the network, the size of each message exchanged is $O(n^2)$. This increased control messaging overhead in a resource poor wireless network could prove crucial. The main advantage is that since the topology information is up to date, there is no latency involved in making routing decisions.

In [2], unidirectional links are detected and an inclusive cycle containing them is formed. This is used to route information from the sink to the source. However, it requires two rounds of message exchange between the sink and the source across the inclusive cycle before the path can be used. Our approach, though more stringent in assuming a strongly connected topology, does not incur this latency as the packets can be tunneled whenever required, without waiting for inclusive cycle establishment.

The approach in [13] is based on using *incremental source routing* by iteratively building the routing tree at each node. A link is advertised only if it is being used by the nodes at its ends. This could potentially lead to longer routes if links along the shortest path are not being used by the nodes harboring them.

The protocols presented above are complete routing protocols. On the other hand, the approach presented by us is a framework, involving network stack modifications at the data link and network layer, enabling various routing protocols to take advantage of the existing unidirectional links.

VIII. Conclusion and Future Work

In this paper, we have presented an approach to make use of unidirectional links in an ad hoc network. This approach uses a tunneling approach to propagate information across the unidirectional link. It makes the unidirectional link behave like a pipe with full data and acknowledgment flow in one direction, but only acknowledgment and routing update flow in the other direction). It utilizes only **O(**n**)** for information propagation. We have explained the tunneling approach and the information that is exchanged and maintained at both the network and the data link layer to support its operation.

We propose to perform a detailed simulation analysis of the protocol proposed, and the performance benefits achieved by using the same. The simulation would determine the likelihood of occurrence of unidirectional links, and then analyze the impact of the same. We intend to compare the performance of algorithms making use of the tunneling approach with those not making use of it, and analyze the improvements in terms of hop count reduction, throughput benefits etc.

References

[1] D. J. Baker and A. Ephremides. The Architectural Organization of a Mobile Radio Network via a Distributed Algorithm. *IEEE Transactions on Communications*, pages 1694–1701, November 1981.

[2] Lichun Bao and J. J. Garcia-Luna-Aceves. Link-state Routing in Networks with Unidirectional Links. In *Eight International Conference on Computer Communications and Networks*, pages 358–363, 1999.

[3] E. Duros W. Dabbous H. Izumiyama N. Fujii and Y. Zhang. A Link Layer Tunneling Mechanism for Unidirectional Links. Network Working Group Internet Draft, June 1999. Work in Progress.

[4] J. J. Garcia-Luna-Aceves and J. Behrens. Distributed Scalable Routing Based on Vectors of Link States. *IEEE Journal on Selected Areas of Communications*, October 1997.

[5] IEEE. *IEEE Draft Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer(PHY) Specification*, d2.0 edition, July 1995.

[6] V. Jacobson. Congestion Avoidance and Control. In *Proceedings of SIGCOMM '88*, August 1988.

[7] D. B. Johnson and D. A. Maltz. Dynamic Source Routing in Ad-Hoc Wireless Networks. T. Imielinski and H. Korth, editors, Mobile Computing, 1996. Kluwer Academic Publishers.

[8] J. Macker and S.C.(chairs). Mobile Ad-Hoc Networks (manet), 1997. http://www.ietf.org/html.charters/manet-charter.html.

[9] V. D. Park and M. Scott Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *IEEE Conference on Computer Communications (Infocom '97)*, 1997.

[10] M. R. Pearlman and Z. J. Haas. Determining the Optimal Configuration for the Zone Routing Protocol. *IEEE Journal on Selected Areas in Communications*, 17(8):1395 – 1414, August 1999.

[11] C. Perkins and P. Bhagwat. Routing over Mulitihop Wireless Network of Mobile Computers. In *SIGCOMM '94: Computer Communications Review*, pages 234–244, October 1994.

[12] Charles Perkins and Elizabeth Royer. Ad Hoc On-Demand Distance Vector Routing. In *2nd IEEE Workshop on Selected Areas in Communication*, pages 90–100, February 1999.

[13] Carlos Pomalaza-Raez. A Distributed Routing Algorithm for Multihop Packet Radio Networks with Uni- and Bi-Directional Links. *IEEE Transactions on Vehicular Computing*, 44(3):579 – 585, August 1995.

[14] R. Prakash. Unidirectional Links Prove Costly in Wireless Ad Hoc Networks. In *Proceedings of the Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications (DIAL-M'99)*, pages 15–22, August 1999.

[15] A. Demers S. Shenker, V. Bhargavan and L. Zhang. MACAW: A Media Access Protocol for Wireless LANs. In *ACM SigComm '94*, September 1994.

[16] A. S. Tanenbaum. *Computer Networks*. Prentice Hall, Englewood Cliffs, 3rd edition edition, 1996.