2005

Wbridge Routing with Minimal Control Messages in Multi-Hop Wireless Ad-Hoc Networks

Navodaya Garepalli
WBRIDGE

ROUTING WITH MINIMAL CONTROL MESSAGES IN MULTI-HOP WIRELESS AD-HOC NETWORKS

By

NAVODAYA GAREPALLI

A Thesis submitted to the Department of Computer Science in partial fulfillment of the requirements for the degree of Master of Science

Degree Awarded:
Spring Semester, 2005
The members of the Committee approve the thesis of Navodaya Garepalli defended on Apr 18, 2005.

Kartik Gopalan  
Professor Directing Thesis

Sudhir Aggarwal  
Committee Member

Xin Yuan  
Committee Member

Zhenhai Duan  
Committee Member

The Office of Graduate Studies has verified and approved the above named committee members.
To Mom, Dad, Raj & Venu...
ACKNOWLEDGEMENTS

I would like to express my gratitude to all those who gave me the possibility to complete this research & my thesis. I am deeply indebted to my advisor Prof. Kartik Gopalan, without whose support and co-operation I wouldn’t be able to finish this thesis nor I would be able to learn new skills & the whole credit for this research should go to Prof. Kartik Gopalan.

I would also like to acknowledge all those people on NS2 users archive mailing list who have helped me in running NS2 simulations and have remained anonymous here.
# TABLE OF CONTENTS

List of Tables .......................................................... vii

List of Figures .......................................................... viii

Abstract ................................................................. xi

1. INTRODUCTION ...................................................... 1
   1.1 Motivation .................................................. 1
       1.1.1 Wireless Ad hoc Networks ....................... 1
   1.2 Problem Statement ........................................ 2
   1.3 Contributions of the thesis ............................... 2
       1.3.1 Brief Summary of WBridge ..................... 2
   1.4 Network Model and Assumptions ........................... 3
       1.4.1 Medium Access Control ....................... 3
       1.4.2 Packet Buffering ............................... 4
   1.5 Roadmap .................................................. 4
   1.6 Summary .................................................. 5

2. STATE OF THE ART IN AD-HOC ROUTING ..................... 6

3. WBRIDGE: PROTOCOL DESCRIPTION .......................... 9
   3.1 Assumptions and Notations .............................. 9
   3.2 Basic structure of a routing table ................... 10
   3.3 Setting up a basic connection ....................... 11
   3.4 Handling Silent end points ............................ 12
   3.5 Maintaining Connectivity During Mobility ........... 18
   3.6 Broadcast reception at destination ................... 18
   3.7 Null IP Packets ....................................... 18
   3.8 Detecting Duplicates ................................... 19
   3.9 Routing Loops ........................................ 20
   3.10 Suppressing Null IP Packets ....................... 23
   3.11 Route table expiry .................................. 23
   3.12 Automatic Re-learning after Node Reboot .......... 23
   3.13 Route Re-Learning Even Without Failures ........... 24
   3.14 Chapter Summary ..................................... 25
LIST OF TABLES

5.1 Average number of link connectivity changes during each 900-simulated seconds as a function of pause time .................................................. 35
5.2 Time values of different timers in WBridge ................................. 36
LIST OF FIGURES

1.1 WBridge is placed as a thin layer above Data Link layer and below IP layer . . 5
3.1 Structure of Routing table of each node . . . . . . . . . . . . . . . . . . . . . . . . . .. . . . . 10
3.2 Nodes initially without any routing tables . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
3.3 Node 0, Mac-Broadcasts the packet . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 13
3.4 Node 2 receives and Mac-Broadcasts the packet . . . . . . . . . . . . . . . . . . . . . . . 13
3.5 Node 4 receives and Mac-Broadcasts the packet . . . . . . . . . . . . . . . . . . . . . . . . 14
3.6 Node 5 receives the packet and sends an ACK back to Node 4 . . . . . . . . . . . . 14
3.7 Node 4 forwards the ACK to Node 2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 15
3.8 Node 2 forwards the ACK to Node 0, inturn Node 0 now sends the data packet 15
3.9 Node 2 forwards the data packet to Node 4 . . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
3.10 Node 4 forwards the data packet to Node 5 . . . . . . . . . . . . . . . . . . . . . . . . . . . 16
3.11 Routing loop of nodes $N_i$ to $N_k$ . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 22
3.12 Relearning by node V . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
4.1 Flowchart of a node, when a packet is to be sent out . . . . . . . . . . . . . . . . . . . 28
4.2 Flowchart of a node, when a packet is received . . . . . . . . . . . . . . . . . . . . . . . . 29
4.3 Structure of Routing table of each node . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 30
5.1 C/R vs Pause Time for 50 nodes and 30 TCP connections moving at 20m/s . . 38
5.2 C/R vs Number of TCP Connections for 50 nodes and 60 sec pause time moving 38
5.3 C/R vs Mobility speed for 50 nodes - 30 TCP connections and 60 sec pause . . 39
5.4 Number of Control Pkts vs Pause Time for 50 nodes - 30 TCP connections and 39
5.5 Number of Control Pkts vs Number of TCP Connections for 50 nodes - 60 sec 40
5.6 Number of Control Packets vs Mobility speed for 50 nodes - 30 TCP connections 41
5.7 R/S vs Pause Time for 50 nodes and 30 TCP connections moving at 20m/s . . 42
5.8 R/S vs Number of TCP connections for 50 nodes and 60 sec pause time moving at 20m/s ................................................................. 43
5.9 R/S vs Mobility speed for 50 nodes - 30 TCP connections and 60 sec pause . 43
5.10 Throughput vs Pause Time for 50 nodes - 30 TCP connections and nodes moving at 20m/s ................................................................. 44
5.11 Throughput vs Number of TCP Connections for 50 nodes - 60 sec pause time and nodes moving at 20m/s ........................................ 45
5.12 Throughput vs Mobility speed for 50 nodes - 30 TCP connections and 60 sec pause ................................................................. 45
5.13 Hop Ratio vs Pause Time for 50 nodes - 30 TCP connections and nodes moving at 20m/s ................................................................. 46
5.14 Hop Ratio vs Number of TCP Connections for 50 nodes - 60 sec pause time and nodes moving at 20m/s ........................................ 46
5.15 Hop Ratio vs Mobility speed for 50 nodes - 30 TCP connections and 60 sec pause ................................................................. 47
5.16 C/R vs Pause Time for 50 nodes and 30 UDP connections moving at 20m/s .............................................................................. 48
5.17 C/R vs Number of UDP Connections for 50 nodes and 60 sec pause time moving at 20m/s ................................................................. 49
5.18 C/R vs Mobility speed for 50 nodes - 30 UDP connections and 60 sec pause .. 49
5.19 Number of Control Pkts vs Pause Time for 50 nodes - 30 UDP connections and nodes moving at 20m/s ........................................ 50
5.20 Number of Control Pkts vs Number of UDP Connections for 50 nodes - 60 sec pause time and nodes moving at 20m/s ........................................ 50
5.21 Number of Control Packets vs Mobility speed for 50 nodes - 30 UDP connections and 60 sec pause ................................................................. 51
5.22 R/S vs Pause Time for 50 nodes and 30 UDP connections moving at 20m/s .. 52
5.23 R/S vs Number of UDP connections for 50 nodes and 60 sec pause time moving at 20m/s ................................................................. 52
5.24 R/S vs Mobility speed for 50 nodes - 30 UDP connections and 60 sec pause . . 53
5.25 Throughput vs Pause Time for 50 nodes - 30 UDP connections and nodes moving at 20m/s ................................................................. 54
5.26 Throughput vs Number of UDP Connections for 50 nodes - 60 sec pause time and nodes moving at 20m/s ........................................ 54
5.27 Throughput vs Mobility speed for 50 nodes - 30 UDP connections and 60 sec pause ................................................................. 55
5.28 Hop Ratio vs Pause Time for 50 nodes - 30 UDP connections and nodes moving at 20m/s ................................................................. 55
5.29 Hop Ratio vs Number of UDP Connections for 50 nodes - 60 sec pause time and nodes moving at 20m/s

5.30 Hop Ratio vs Mobility speed for 50 nodes - 30 UDP connections and 60 sec pause
ABSTRACT

Existing wireless ad hoc routing protocols, such as DSDV, AODV, DSR, TORA and GPSR, rely heavily upon the use of a large number of control messages to build and maintain routes between communicating mobile end-points. In this thesis, we present the design and evaluation of WBridge – an ad hoc wireless routing protocol that greatly reduces the number of control messages required to discover and maintain routes.

The WBridge protocol operates in a manner similar to the transparent bridges in the traditional wired LAN environments. Like wired bridges, W Bridges observe packets in transit and apply backward learning mechanism to transparently learn forwarding paths to the destination. Unlike wired bridges, however, W Bridges do not need to construct or maintain extensive network-wide spanning trees. Furthermore, it can be proved that transient routing loops do not last longer than a bounded time period.

The WBridge protocol requires no control messages to construct and maintain routes for bidirectional traffic (such as TCP connections) and minimal data-like control messages for one-way traffic (such as streaming UDP). Additional advantages of WBridge protocol include low-latency route setup, network throughput comparable to state-of-the-art routing protocols, no need for promiscuous-mode operation of wireless interface and elimination of ARP request/reply messages.
CHAPTER 1

INTRODUCTION

1.1 Motivation

Wireless networks have emerged as one of the rapidly growing communication technologies. Wireless devices are usually the combination of computational capabilities integrated with networking capabilities among the devices which are free to move.

1.1.1 Wireless Ad hoc Networks

Nodes in a wireless Ad hoc network can move freely and still expect to communicate with their peers. Usually nodes have a wireless transmitter/receiver attached to them, and communicate through the links with neighbors who are in the range of that transmitter/receiver.

Ad hoc networks are infrastructure-less networks, which means that there is no central base station connected to the wired network which manages wireless connectivity. These networks are cost effective and very easy to setup compared to conventional networks.

Some of the applications of Wireless Ad hoc networks:

1. Military combat situations, where the setup time should be minimized and there should be no single point of failure.

2. Disaster recovery, where the entire existing infrastructure has collapsed and setting up a new network in a moment’s notice is high-priority.

3. Online Mobile games, where the users desire to play games on the go over a network.

4. Business environments, where the employees can cooperate outside the office.
1.2 Problem Statement

Existing routing protocols heavily rely on the control messages to route the packets. As we will show later in Section 5.3 the number of control messages in existing protocols (like DSDV[1], AODV[2], DSR[3] etc) are almost comparable to the number of data packets.

Most of the existing routing protocols use ARP protocol to find out the MAC/hardware address of their communicating endpoint.

In this thesis, we tackle the problem of whether it is possible to design ad hoc routing protocols that incur low control message overhead and at the same time maintain a high network throughput.

We propose a new ad hoc routing protocol called WBridge, which incurs no control message overhead for bidirectional traffic and minimal overhead for unidirectional traffic. In addition, WBridge doesn’t use ARP protocol, instead it extracts information from the packet that was received. Similarly, WBridge nodes do not use any Hello messages to discover the neighbors.

As a bottom line, the “control messages are inherent in the existing routing protocols”. This is one of the primary reasons which we couldn’t digest about the existing routing protocols and we have come up with a new routing protocol, WBridge.

1.3 Contributions of the thesis

In this thesis, we design and evaluate a simple ad-hoc wireless routing protocol, called WBridge, that does not rely on the use of control messages. We investigate the advantages and disadvantages of eliminating control message overhead in order to achieve greater protocol simplicity and show that WBridge indeed provides better or competitive performance in comparison to existing routing protocols on a number of performance metrics.

1.3.1 Brief Summary of WBridge

The basic idea behind WBridge is inspired by the concept of transparent bridging that has long been relied upon in traditional wired LANs. Transparent bridges apply the principle of backward learning to learn and maintain routes to different nodes in the LAN. In its simplest form, a transparent bridge looks at each incoming packet and maps the Layer-2
source address A of the packet to the input port P on which the packet arrives. Later, when other packets destined to address A arrive at the bridge, they are forwarded over the output port P, under the assumption that address A must be reachable in the direction of port P. Thus transparent bridges learn routes to various destinations on-the-fly by observing the incoming direction of data packets in transit. The routing process is completely transparent to end-nodes because no route setup messages need to be exchanged between sources and destinations. In order to prevent routing loops, a distributed algorithm constructs and maintains a spanning tree over all bridges in the LAN, which also forms the primary source of routing overhead.

The primary motivation behind WBridge has been to combine the on-demand nature of current ad-hoc routing protocols with the simplicity of backward learning in traditional wired-LAN bridges. On one hand, WBridge eliminates the protocol complexity that arises in the processing of numerous control messages in current ad-hoc routing protocols. On the other hand, WBridge also avoids the extensive spanning-tree construction and maintenance overhead of wired-LAN bridges. WBridge is completely decentralized, self-starting, self-organizing and scales to a large number of nodes. It adapts quickly to dynamic network conditions and mobility in a transparent manner. Furthermore, we show that transient routing loops, that may arise under certain pathological scenarios, last for a provably short and bounded duration and do not significantly impact the routing performance.

Advantages of WBridge include low-latency route setup, minimal/no control packets, no need for promiscuous-mode operation of wireless interface, no periodic updates, no hello messages, and elimination of ARP request/reply messages.

1.4 Network Model and Assumptions

Our network model is based upon a collection of possibly mobile nodes aggregated together in an ad hoc network, i.e. without any infrastructure support such as base stations.

1.4.1 Medium Access Control

We assume that the link layer uses the IEEE 802.11 standard Medium Access Control (MAC) protocol Distributed Coordination Function (DCF). Although WBridge works with
other Medium Access Control protocols, all we expect from MAC layer is that each unicast packet should be preceded by some mechanism like Request-To-Send/Clear-To-Send (RTS/CTS) and correctly received unicast packet should be followed by ACK from the receiver to the Sender. The RTS-CTS mechanism helps prevent the hidden terminal problem. Similarly, ACKS ensure basic link-level reliability among neighboring nodes.

1.4.2 Packet Buffering

We also assume that every node is capable of buffering at least 50 packets in the interface queue and manage the queue in the drop-tail fashion. The above parameters are taken from the Monarch Project’s[4] Wireless extension to Network Simulator NS2[5].

Figure 1.1 shows that the WBridge routing protocol is positioned as a thin layer in the network protocol stack between the Layer-3 (Network/IP layer) and Layer-2 (Mac-layer). Although forwarding is based on destination IP address, the WBridge routing is slightly different from the form traditional IP routing. As we will see in Chapter 3, the next hop for the destination IP in the WBridge routing table is the layer-2 Mac address of the next hop rather than its IP address. WBridges nodes do not care about the IP address to Mac address mapping of other nodes - they only need to recognize their own IP address when they see one in a packet and they need to know the IP address of the destination where they want to send the packet to.

1.5 Roadmap

Chapter 2, compares WBridge protocol with other existing routing protocols and makes the case for why we have come up with this new routing protocol.

Chapter 3, discusses the technical details of the protocol, such as how the route is setup and maintained. We also discuss the mechanism for detecting duplicate packets & handling routing loops.

Chapter 4, focuses on the internal implementation details of WBridge, including the existing tools that we used, and the different data structures, timers etc.

In Chapter 5, we compare the performance of WBridge with AODV and DSR algorithms.
Figure 1.1. WBridge is placed as a thin layer above Data Link layer and below IP layer

And finally in Chapter 6, we conclude the thesis with a summary of contributions and future enhancements that can make WBridge more practical and more efficient.

1.6 Summary

Ad-hoc networks have a wide-range of applications ranging from simple communication to time critical/real time applications. WBridge a new wireless ad hoc routing protocol, which gets the name from wired LAN Bridges, where the bridges have a self learning mechanism to automatically learn link-level routes. WBridge has many advantages over other existing ad hoc routing protocols. In the following chapters, we present the technical details of WBridge.
CHAPTER 2

STATE OF THE ART IN AD-HOC ROUTING

Existing Ad-hoc routing protocols like DSDV, AODV, DSR, GPSR and TORA, rely heavily upon the use of control messages to build and maintain routes between communicating mobile end points.

**DSDV** *Destination-Sequenced Distance Vector (DSDV)*\([1]\) algorithm is basically an extension to Bellman-Ford mechanism\([6]\). Nodes using DSDV, send network wide transmissions of their routing tables with a sequence number. Whenever they update their routing tables they increment the sequence number and again broadcast. This creates network wide broadcasts and this is effective in ad-hoc networks only for small populations of mobile nodes as the control message overhead grows as \(O(n^2)\)[2].

**AODV** *Ad-hoc On-Demand Distance Vector (AODV)*\([2]\) routing protocol is based upon distance vector, and uses destination sequence numbers to determine the freshness of routes. It is a on-demand routing protocol, so nodes don’t send out messages unless it needs a route. Nodes broadcast a RREQ packet whenever they want to find out a route to a destination, if an intermediate node receives a RREQ packet and it has an active route to that destination it replies with a RREP packet, which is unicast back, or the destination might reply back with the RREP message. Every intermediate node that receives a RREQ message, stores the route back to the originator so that RREP can be unicasted back to the originator. Every route expires after some predetermined period of time and sending a packet via a route will reset the associated expiry time[6]. Whenever there is a route failure, the node which detects the failure sends back an RERR message to the Source and the Source initiates a route discovery process again.

**DSR** *Dynamic Source Routing (DSR)*\([3]\) is a on-demand source routing protocol, where the source finds the route by initially broadcasting out a RREQ packet, every intermediate
node appends its own id to the packet and rebroadcasts, this process continues until the packet reaches the destination or if any intermediate node has a route to the destination. Every Source node stores the complete route to the destination. The destination or the intermediate node has a complete route back to the initiator and can unicast back the route.

Route is maintained by using link level acknowledgments, if any node in the route determines that a node is unreachable, then it sends back a route error (RERR), message to the sender, and the sender initiates the route discovery process again. The nodes using DSR, should work in promiscuous mode.

**GPSR** *Greedy Perimeter Stateless Routing (GPSR)*[7] is a position based static routing, where nodes do a local broadcast of their positions and their ID’s (can be mac addresses). Whenever a node needs to forward a packet, it sends it to the nearest neighbor greedily using right hand perimeter rule.

The nodes using GPSR, should know their positions using GPS or other techniques and every node should work in promiscuous mode. GPSR also has a requirement that the number of neighbors within a nodes range must be substantially less than total number of nodes in the network.

**TORA** *Temporally Ordered Routing Algorithm (TORA)* [8] is on-demand distributed routing protocol based on “link reversal” algorithm. Route optimality is considered to be of secondary importance and longer routes are often used to avoid the overhead of discovering newer routes.

When a node needs a route to a particular destination, it broadcasts a QUERY packet containing the address of the destination for which it requires a route. This propagates until it reaches either the destination, or an intermediate node having a route to the destination. The recipient of the QUERY then broadcasts an UPDATE packet listing its height with respect to the destination. As this propagates through the network, each node that receives the UPDATE sets its height to a value greater than the height of the neighbor from which the UPDATE was received. When a node discovers that a route is no longer valid it adjusts its height so that it is a local maximum with respect
to its neighbors and transmits an UPDATE packet. When a node detects a network partition, it generates a CLEAR packet that resets routing state and removes invalid routes from the network[4].

TORA is layered on top of IMEP, the Internet MANET Encapsulation Protocol, which is required to provide reliable in-order delivery of all routing control messages from a node to each of its neighbors, plus notification to the routing protocol whenever a link to one of its neighbors is broken or created. For link status sensing and maintaining a list of a node’s neighbors, each IMEP node periodically transmits a BEACON (or “BEACON-equivalent”) packet, which is answered by each node hearing it with a HELLO (or “HELLO-equivalent”) packet[4].

DSDV nodes broadcast its routing tables throughout the network and both AODV & DSDV use “HELLO” messages to discover the neighbors which is locally broadcasted within one hop, periodically. While DSR sounds optimal in case of control messages, DSR nodes should work in promiscuous mode. For GPSR every node should know its position before hand using GPS or other techniques and nodes should work in promiscuous mode. One problem that arises with nodes working in promiscuous mode is that for every packet thats being “heard”, the CPU receives an interrupt and should process that packet. So, if 90% of the packets listened in the promiscous mode turns out to be useless then every node wastes a lot of processing power, which is highly important incase of mobile devices. TORA requires reliable in-order delivery of control messages and it should have some mechanism to detect the links with the neighbors.

Every protocol described above has one or more disadvantages that we hope WBridge will overcomes.
CHAPTER 3

WBRIDGE: PROTOCOL DESCRIPTION

WBridge protocol is completely on-demand Ad-hoc routing protocol, in which nodes dynamically learn routes by applying the concept of backward learning from data packets in transit. WBridge does not rely upon any control messages for route construction and maintenance, thus avoiding the control message overhead and processing complexity. One important consequence of the absence of control messages is that the WBridge protocol operations at each node are fully independent. Each node making completely local routing decisions that require no co-ordination whatsoever among the intermediate nodes but lead to globally efficient packet delivery to destination.

3.1 Assumptions and Notations

We assume that all the nodes in the network who wish to communicate are also able to forward the packets of other nodes. The nodes should be able to communicate bidirectionally, and for every packet that is sent at the link level, if the receiver receives it, will send a link level acknowledgment back, except in cases of broadcast (where the packet is not specific to one receiver).

WBridge nodes have two levels of identity. One is the link level (layer-2) or MAC address, other network level (layer-3) or IP address, which a node uses to identify and establish communication sessions with any node (not necessarily a neighbor). A node does not need to have any prior knowledge of the identity, location or connectivity of other nodes in the network except of course, the layer-3 IP address identity of nodes with which it wishes to communicate. We assume that there exists a MAC level broadcast where the destination MAC-Address is 0xfffffffffff which is received by all the immediate neighbors of that node.

Any number of nodes in the network can be mobile and can begin moving without any notification. Like, other Wireless routing protocols, WBridge supports rapid rates of
mobility, but we assume that nodes do not move so rapidly as to render broadcast of every
data packet as the only feasible routing alternative.

We assume that all the nodes in the network are reachable through some other node in the
network, if a node is unreachable through any other node in the network, then it is
isolated and the network is partitioned. In this case any routing protocol including WBridge
cannot deliver the packets to those unreachable destinations or nodes in the other partition.

Notations:
Node’s are represented by UPPER CASE letters, like X,A,S or D...

$S_{IP}$ is the source IP address

$D_{IP}$ is the Destination IP address

$S_{MAC}$ is the source MAC Address

$D_{MAC}$ is the destination MAC Address

(Here $S_{MAC}$ and $D_{MAC}$ changes at every hop that is traversed by the packet).

Packets are represented by $p_1,p_2,p_3,...$ (lower case)

3.2 Basic structure of a routing table

The basic structure of a WBridge node’s routing table is shown in Figure 3.1 Here for
every destination IP address there exists corresponding entries like valid flag, next_hop, age,
etc. valid flag represents whether this entry is valid or not. next_hop is the Mac address of
the next hop where this packet has to be forwarded for that corresponding destination IP
address. age represents the time until this entry has to be stored in the routing table, after
the age expires the routing table entry is deleted. The other entries in the routing table will
be discussed as they are encountered in the following discussion.

3.3 Setting up a basic connection

If a source node S doesn’t already have a route to a destination D, it MAC-broadcasts the
first data packet p1 (lets assume that D is not the immediate neighbor of S). p1 is received
by all immediate neighbors of S. Next, every immediate neighbor tries to learn every detail
from this packet received. First they create a routing table entry for S\textsubscript{IP} (if there doesn’t
exist already) and they enter the next hop as the S\textsubscript{MAC} of the p1 (which basically says
that in-order to reach S\textsubscript{IP}, one should forward the packet to S\textsubscript{MAC}). Then the immediate
neighbors MAC-broadcast the packet again. If any of the immediate neighbors already have
a routing entry for D\textsubscript{IP} then instead of MAC-broadcasting again, these neighbors would
unicast the packet to the corresponding next hop according to their routing table. This
process gets repeated until the packet reaches the D\textsubscript{IP} or D. Once the packet reaches the
destination D all the intermediate nodes and D know how to get to S.

Now, there are two cases: one is either D is an active end point which responds back (like
in case of TCP connections) or D might be a silent end point (like in case of unidirectional
UDP connections). Let’s, for now assume that D responds back with packet p2. D doesn’t
broadcast p2, because it knows the next hop to get to S, so p2 is forwarded to intermediate
node that has sent p1 to D, and intermediate node forwards according to its routing table,
and so on... this process repeats until the packet reaches the S. Once p2 reaches S, there is
a route setup between S and D, because all the intermediate nodes and S know how to get
to D and also D and all intermediate nodes know how to get to S. In section 3.4 we will see
how to handle the silent end points.

Besides finding next hop in the initial setup every node, also finds out about the preferred
next hops. Basically whenever there is a broadcast, a node receives the same packet
(duplicates)[More about duplicate detection in Section 3.8], from many sources besides
next hop. Instead of just discarding these duplicates, WBridge tries to extract some
information like what is the $S_{IP}$, and $S_{MAC}$, if the $S_{MAC}$ doesn’t match the next_hop of that $S_{IP}$ then WBridge enters this $S_{MAC}$ into the preferred next hops list. WBridge nodes use this preferred next hops in case of route failures, when the routing to next_hop fails they try to send the packets to preferred next hops instead of broadcasting.

Included are the screen-shots from Nam (Network Animator)[5], where node 0 wishes to communicate with node 5. Figure 3.2 shows the initial setup without any routing information. Figure 3.3 shows node 0 broadcasting and all the consequent figures from Figure 3.4 to 3.10, show the route setup

3.4 Handling Silent end points

On one hand it is reasonable to expect that most real-world network applications would engage in bidirectional communication in which both sides will generate either application-level data and control packets, or transport-level control packets. For instance, the use of most popular TCP protocol would, at a minimum involve at least one end transmitting data and the other end responding back with TCP acknowledgments. On the other hand, it is also possible that in some rare instances (such as with some UDP applications), the end-points
Figure 3.3. Node 0, Mac-Broadcasts the packet

Figure 3.4. Node 2 receives and Mac-Broadcasts the packet
Figure 3.5. Node 4 receives and Mac-Broadcasts the packet

Figure 3.6. Node 5 receives the packet and sends an ACK back to Node 4
Figure 3.7. Node 4 forwards the ACK to Node 2

Figure 3.8. Node 2 forwards the ACK to Node 0, in turn Node 0 now sends the data packet to Node 2 to forward it to Node 5
Figure 3.9. Node 2 forwards the data packet to Node 4

Figure 3.10. Node 4 forwards the data packet to Node 5
may not necessarily indulge in bidirectional communication or, even if they do, one of the 
communicating end-points remains largely silent for long periods of time. In such cases, the 
backward learning mechanism described above would consistently maintain network routing 
information for only the active sender end-point and not for the passive receiver end-point. 
This creates an undesirable situation in which all data packets sent to the passive receiver are 
transmitted via broadcast, due to absence of network routing information for the receiver. 
To rectify this situation, the WBridge protocol layer at the receiver node D monitors whether 
packets from S are being received for some \textit{ACTIVITY\_INTERVAL} duration (say 80\% of 
maximum route expiration time) without the application at D sending any data back to S. 
In this case, WBridge protocol layer at D creates a dummy IP packet with zero byte data 
payload and transmits it to S. As the dummy IP packet from \textit{D} to \textit{S} is routed through 
the network, intermediate nodes learn the routing information for node D. This ensures that 
future data packets from S to D are unicast rather than broadcasted through the network. 
Since the dummy IP packet is not directed towards any specific application at S, it gets 
silently discarded by the WBridge layer once it reaches S. Once the communication session 
ends, and WBridge layer at D observes that S is not sending any more data, dummy IP 
packets are not generated as well.

While the dummy IP packet could be technically considered as a control packet, careful 
examination reveals this is not quite the case. First, the dummy IP packet is processed 
and forwarded just like any other data packet through the network. It does not require 
any form of special processing by the intermediate nodes, except at the time of its creation 
at the passive communication end-point. Secondly, this packet is created only under the 
exceptional circumstance when one of communication end-points does not generate its own 
data for a certain period of time. Without the dummy IP packet, communication can still 
take place, albeit in an inefficient manner.

Current usage of the Internet is dominated by transmission control protocol (TCP) 
traffic such as remote terminal (e.g., Telnet), FTP, Web traffic, and electronic mail (e.g., 
SMTP). These TCP sources constitute 90\% of all traffic\cite{9}, if the same pattern of 
connections occur in Wireless Ad-hoc networks then there are very few end points which are 
passive/silent, hence very few null IP packets. More about Null IP packets is described in 
Section 3.7.
3.5 Maintaining Connectivity During Mobility

As nodes are free to move, the connections associated with these nodes might be broken if there is any mobility. As with other routing protocols WBridge does not distinguish between the mobility of source node or intermediate node or the destination node, all the nodes have the same mechanism. Usually a mobility is detected if you don’t get an ACK back for the packet you have sent or if you don’t get CTS for the RTS requested after some maximum retries. As there are no control messages, if one node detects that there is a failure in the routing, it cannot notify the source/sender about the failure. It is the responsibility of this node to deliver the data packets to the destination. The node tries to send the data packets to the preferred next hops. If this is successful, then it sets the next hop as the preferred next hop and keeps forwarding the packets. If transmissions to none of the preferred next hops is successful then it broadcasts the packets. After some time, packets from the sender will cease to arrive at the intermediate node. We will see why this happens later in the following Section 3.6.

3.6 Broadcast reception at destination

Whenever there is a route failure, the packets are broadcasted and the destination D receives the broadcasted packets, D finds that the packet has been received as broadcast, and figures out that there was a route failure. So, in order to allow the route to settle, D sends out a packet to the source S, so that the intermediate nodes learn about the routing information for D. For this, the WBridge on node D waits for the application to respond back to the received broadcasted packet for a short period of time. If the application does not respond back then WBridge at node D sends out a Null IP packet to S, assuming that the application is a silent end point.

3.7 Null IP Packets

As described above we send a dummy IP packet, which we call as “Null IP Pkt”. These Null IP packets are used in cases of silent end points where the destination is a passive receiver and doesn’t send back any return packets. The basic structure of a Null IP packet is the IP header, without any payload \((S_{IP},D_{IP})\) and the Mac-header is attached
at the link level at every node that's been traversed. So, on a total the Null IP packet is 64 bytes (including the Mac-header), which is sent in cases of silent end points for every (0.8*ROUTE_EXPIRY_TIME) period.

**Here we try to find out how many Null IP packets are sent out for 50 connections:**

Let's assume that there are 50 connections among 50 nodes, and ROUTE_EXPIRY_TIME is 5 seconds. As the ROUTE_EXPIRY_TIME is 5 seconds, every 4 seconds (0.8*ROUTE_EXPIRY_TIME) a Null IP packet is sent out for each silent end point connection. 1 packet every 4 seconds i.e., 0.25 Null IP packet is sent out every second per silent end point connection. By using the above study in [9] that 90% constitute the TCP traffic and 10% constitute the UDP traffic there are around 5 silent end point connections.

So, on the total we have 5 * 0.25 * 64 bytes/sec = 80 bytes/sec control overhead network wide. Later on, in Section 3.10, we will tell how to suppress Null IP packets, in cases where there are multiple connections between the same source and destination. So, on an overall the control message overhead turns out to be significantly less than other routing protocols.

### 3.8 Detecting Duplicates

Wired-LAN bridges construct and maintain a network-wide spanning tree in order to prevent exponential proliferation of duplicate packets that may arise due to broadcasts or transient routing loops. The primary reason for this conservative routing via spanning tree is that link layer Ethernet header does not carry any hop count or TTL information, that wired-LAN bridges can use. However, a spanning-tree approach is neither feasible nor advisable in wireless ad hoc networks due to the far higher degree of non-determinism in wireless transmission as well as higher degree of node mobility. In fact, restrictions imposed by spanning trees based routing are increasingly being viewed unfavorably even in wired environments.

WBridge nodes do not construct or maintain any spanning trees. Instead they damp the propagation of duplicate packets by relying on the IP Identification and fragment offset fields carried inside the IP header of every packet.

The simplest form of broadcast and duplicate damping mechanism is as follows. Corresponding to each IP address $S_{IP}$ in its routing table, a WBridge node remembers the largest IP Identification value, and corresponding fragment offset values, that it has seen in packets
with $S_{IP}$ in their source IP field. The WBridge node then discards any packet from $S_{IP}$
whose IP Identification value is smaller than the largest recorded IP Identification value
from the same source. Since IP Identification field is 16-bits long, the recorded values wrap
around upon reaching a value of 64K. Thus a duplicate copy of a packet that arrives at a
node will carry either a smaller IP Identification value than the one already recorded, or in
case they are equal, a fragment offset value that has already been seen by the node. Such a
packet will be discarded.

The danger with the above simple scheme is that occasionally re-ordered packets might
be mistakenly identified as duplicates and dropped. For instance, a packet $p_1$ from source
$S$ may take a long route to reach an intermediate node $X$, whereas a later packet $p_2$ from
$S$ may reach $X$ earlier via a shorter route. Consequently packet $p_1$ would get dropped as
it carries a smaller Identification value. In order to mitigate this situation, every WBridge
node remembers the the identification and offset values of last $k$ (say 10) packets seen from
each source node $S$. A packet is dropped only if its value is among the $k$ remembered values
or is smaller than all of them. This permits re-ordering of packets to be tolerated within a
window of $k$ packets.

While one may be tempted to equate the use of identification field in WBridge protocol to
the use of destination sequence numbers in AODV, they are not at all the same. Destination
sequence numbers in AODV are used to maintain freshness of routes and to prevent the
formation of routing loops. On the other hand, the identification fields in WBridges are
merely used to prevent propagation of duplicate data packets. In fact, a duplicate damping
mechanism similar to one described here would be required in any forwarding mechanism
that does not rely on a spanning tree.

### 3.9 Routing Loops

One side-effect of not relying on a spanning tree is that transient routing loops may arise
for some destinations when network topology is changing. We define a *complete transient
loop* for a destination $D$ as one in which each node in the loop has a valid routing entry
towards the next hop node in the loop. On the other hand, a *partial transient loop* for a
destination $D$ is one in which at least one node in the loop does not have a valid routing
entry for D. Consequently, such a node would broadcast the packets destined to D and the
ext node in the loop picks up the broadcast packet and recirculates it around the loop.
Algorithms such as AODV incorporate intricate mechanisms (based on destination sequence
numbers) to guarantee that complete transient loops are not formed at all times. On the
other hand, partial transient loops are impossible to avoid whenever any intermediate node
needs to broadcast a data packet to destination.

WBridge routing does not worry about avoiding transient loops at all costs. Rather, if
and when transient loops are indeed formed, their impact on routing is highly minimized.
This is due to three factors.

1. Since WBridge nodes perform duplicate packet detection (Section 3.8), packets will not
traverse a loop more than once. Rather a packet would be discarded once it revisits
the node through which it originally entered the loop.
2. Successive WBridge nodes decrement the Time-to-live (TTL) field in each packet and
discard them when TTL reaches zero.
3. WBridge guarantees that complete transient loops last for less than short bounded
time interval. We prove this claim later in the section.

While (1) and (2) guarantee that packets do not proliferate unchecked even when complete
or partial transient loops are formed, (3) guarantees that complete transient loops do not last
forever. Also, manipulating the TTL field inside the IP header is all right from the layering
perspective as well since WBridge in fact performs routing based on layer-3 destination IP
address. We now prove that complete transient loops last for bounded duration.

Theorem 1: Lifetime of a complete transient loop for destination D is less than or equal
to $MAX\_ROUTE\_LIFETIME$.

Proof: Consider the complete loop in Figure 3.11 for destination D with k nodes $N_1$ to
$N_k$, where the next hop for destination D at $N_i$ is node $N_{i+1}$. By definition, node D itself
cannot be part of the complete loop and hence is outside the loop. Furthermore, in order for
node $N_i$ to maintain $N_{i+1}$ as its next hop entry for D, $N_{i+1}$ must forward a packet with source
IP address $D_{IP}$ to $N_i$ within $MAX\_ROUTE\_LIFETIME$ interval of its learning the next
hop. Two cases arise. In the first case, node D does not send a packet through any node in
the complete loop for `MAX_ROUTE_LIFETIME` duration. Thus \( N_{i+1} \) cannot forward a packet with \( D_{IP} \) as source IP address to \( N_i \). Consequently \( N_i \)'s routing entry for D will expire and become invalid, thus breaking the complete loop within `MAX_ROUTE_LIFETIME` interval. In the second case, assume that D sends a packet which enters the loop through node \( N_i \). Since node D is outside the loop, the packet from D must enter the loop from some external node \( N_e \). Without loss of generality, we assume that \( N_e \) is a neighbor of node \( N_{i+1} \) that we are presently considering. Thus node D’s packet arrives at \( N_{i+1} \) from \( N_e \). However, \( N_e \) is different from \( N_{i+2} \) which is the current next hop for D at node \( N_{i+1} \) (since \( N_e \) is outside the loop). Thus, as described in 3.13, \( N_{i+1} \) will switch to maintenance mode to re-learn its next hop for D by invalidating its routing entry for D via \( N_{i+2} \). This too would break the complete loop within `MAX_ROUTE_LIFETIME` interval.

Once a complete loop is broken, it may get converted into a partial loop in which at least one of the nodes in the loop broadcasts the packets destined to D. However the partial loop also lasts only till the broadcasting node re-learns a new route to destination D. Though the duration of re-learning interval would depend upon network conditions and topology, it is expected to be short if the broadcasting node lies along the path of data packets arriving
from D. Furthermore, D is guaranteed to generate either a data packet or a Null IP packet within a certain ACTIVITY_INTERVAL, as described in Section 3.4.

3.10 Suppressing Null IP Packets

In cases where there are multiple connections among the same source and destination, and if one of the node happens to be the silent end point, then we could take advantage of the other connection’s packets instead of sending out Null IP packets.

For instance, lets say there happens to be a connection C1 between Node X and Node Y, where X is the active end point and Y is the silent/passive end point, and simultaneously lets say there happens to be an other connection C2 between X and Y, where both X and Y are active end points or X is the silent/passive end point and Y is the active end point.

As WBridge is placed on top of Mac-layer and below IP layer, WBridge can actually look at all the packets to and from this node. Whenever it sees that a packet going from Y to X, and it has a Null IP packet to send to X, it can suppress that Null IP packet, because the other end is receiving the packets from this side and the network and destination knows about the routing information of this node.

3.11 Route table expiry

Every routing table entry has an age period of MAX_ROUTE_LIFETIME. Whenever a packet is received, the corresponding age of the source IP address entry in the Routing table is refreshed. If a node doesn’t receive any packets corresponding to the routing entry for $S_{IP}$ for MAX_ROUTE_LIFETIME period then the routing entry is deleted from the Routing table.

3.12 Automatic Re-learning after Node Reboot

A failure or reboot of a WBridge node X may induce neighboring nodes to re-learn their routes that passed through X. However, once X reboots and comes back up, it does not need any re-synchronization with its neighbors. Rather it can immediately start learning and participating in the routing process again. This is in contrast to AODV in which a node
Figure 3.12. Relearning by node V

is forced to stay silent for a period of time after reboot in order to flush the routing state of its neighbors[2].

3.13 Route Re-Learning Even Without Failures

So far we have considered the route re-learning process at a node X that detects a communication failure with one of its immediate neighbors Y. However, network failures, mobility, or simply appearance of a better route can also induce re-learning at nodes that do not experience any communication failure with their valid next hop route. For instance, node V in the Figure 3.12 has a valid next hop neighbor W for destination D. However, at some point in its operation, node V might receive non-duplicate first-time packets sent by D via another neighbor K and not via W. This is an indication to V about a possible failure, mobility, or presence of a better route to D and V tries to re-learn its route to D. At this point, node V invalidates its current routing entry for D that goes through W. Subsequently, just as immediate link failures were handled above, V observes all the non-duplicate packets sent by D that arrive through its various neighbors (W, I, J and K) for MAX LIFETIME
interval after invalidation. At the end of the observation period, it selects that neighbor from which it receives the latest non-duplicate packet sent by D as the next hop for D.

3.14 Chapter Summary

Unlike other routing protocols, WBridge doesn’t use any control messages for bidirectional connections. For a basic route setup it broadcasts the initial data packet and expects the route to be formed by itself, as once the other end sends the data or Null IP packet back, the route is setup both ways. As TCP sources constitute 90% of all traffic in the internet, assuming the same connection pattern exists in Wireless Ad-hoc networks we expect to have a very few silent end points, thus resulting in minimal Null IP packets. Even though, if the network consists of completely, silent endpoints we prove in the performance evaluation that WBridge has very little control overhead in comparison to other popular routing protocols, DSR and AODV.
CHAPTER 4

WBRIDGE SIMULATION

4.1 Used NS2.27/NS2.28

We used discrete event Network Simulator(NS2)[5] on Linux/Unix platform to implement the WBridge protocol and compare its performance with two popular routing algorithms – AODV and DSR. NS2 began as a variant of the REAL network simulator in 1989 and has evolved substantially over the past few years. In 1995 NS development was supported by DARPA through the VINT project at LBL, Xerox PARC, UCB, and USC/ISI. Currently NS2 development is support through DARPA with SAMAN and through NSF with CONSER, both in collaboration with other researchers including ACIRI.

One of the primary reasons why we choose the Network Simulator NS2 is that it is widely used and accepted. A number of prior works compare the performance of different Wireless routing protocols, based on NS2, so we thought it to be appropriate to evaluate WBridge using NS2.

One can get the Network simulator from
http://www.isi.edu/nsnam/dist/ns-allinone-2.28.tar.gz
and the source code of WBridge from
http://www.cs.fsu.edu/~kartik/WBridge/code.tar.gz

We created a separate protocol in NS2 Wireless section, and a new packet type, which we’ve used as NULL IP Packet, with no data but just the IP header. There are instructions and necessary files in tar.gz file provided to add WBridge in NS2. If you have any questions or problems installing this, you can contact the email provided in the installation instructions file.
4.2 Mac layer modifications

In NS2 wireless simulations, one can see that, the packets that are sent out, are being processed by the corresponding routing protocol. But when a packet is received by a node from outside, the routing protocol is not notified and the protocol has no idea whether there was a packet from the other side of the connection. Also, we couldn’t find any other mechanism that let the routing protocol be notified, so we’ve modified the Mac layer of the nodes, and implemented the routing mechanism in the Mac Layer itself (This is not necessary, if WBridge knows about the packets that is being received). So, whenever a packet is being sent out or received, the mac layer gets the packet and it decides the next hop (in case if it is be routed), Null IP sequence number (in case if it is a Null Ip packet being originated), it modifies the routing table according to the received packets details and so on.

4.3 Flow Chart for Senddown

“A picture is worth a thousand words”, so here we provide a big picture of our implementation in Figure 4.1 what a node does, when a packet is to be sent out (either to be forwarded or originated).

4.4 Flow Chart for Sendup

Figure 4.2, shows the flowchart, when a node receives a packet,

4.5 Various data structures & Timers

The various data structures that we used to implement the WBridge are, For each node:

• Routing Table (Used STL’s map data structure)

• Timers
  – Route expiry timer
  – Null IP timer
Figure 4.1. Flowchart of a node, when a packet is to be sent out
Figure 4.2. Flowchart of a node, when a packet is received
Figure 4.3. Structure of Routing table of each node

- Initial phase timer
- Mobility Fault timer
- Buffer check timer

4.5.1 Routing Table

For convenience, we show the basic structure of a routing table again in Figure 4.3, which is the same Figure 3.1. Here we discuss in detail every field of the routing table entry.

We have used a C++ STL’s map to map the the IP address to the structure with the contents valid_flag, next_hop, age, latest_next_hop, num_hops, buffer, Mobility_fault_timer, pref_next_hops.

- **valid_flag:**
  A valid flag is used to denote whether this routing entry is valid or not. If this is set to true then the routing entry is valid, else the routing entry is invalid.

- **next_hop:**
  next_hop is basically the mac address of the neighbor node which can forward the packet to that IP address.

- **age:**
  age is the age of the routing entry in seconds, once the age expires the routing table
entry is deleted from the routing table. RTable_Timer checks once every 100ms, to see
whether the routing table entry is expired or not. If it is expired then the packets in
the buffer are sent out and the entry is deleted from the routing table. This one is set
to 5 seconds, which we found to be optimal.

- **latest_next_hop:**
  latest_next_hop is used to detect the mobility scenarios. latest_next_hop is the
  neighbors mac address from where a new packet (in the on going connection) is received.
  If latest_next_hop is not the same as next_hop then the corresponding IP address node
  has moved and thats the reason we’ve received the packet from latest_next_hop instead
  of next_hop. In this case, we fire up a Mobility_fault_timer which after sometime decides
  the next_hop.

- **num_hops:**
  This field is basically used to denote the number of hops to the corresponding IP
  address. Right now, we are not using this field in the routing decisions.

- **buffer:**
  We use a buffer to store the packets to that corresponding IP address, until the route
  gets valid or route table expires. Once the route gets valid, we send out all the packets
  in the buffer to the valid next_hop. In-order to prevent a number of broadcasts we use
  this mechanism.

- **Mobility_fault_Timer:**
  Once we detect that there is a mobility (by comparing the nexthop and the latest
  nexthop), this timer is fired up. This timer sets up the latest nexthop to the nexthop.
  So, after certain period of time whatever is the latest nexthop will be the next hop to
  that destination IP address. The reason why we wait for certain time is, when a node
  is moving, there is no point in setting up the next_hop to the neighbor who forwarded,
  because once the node moves again, we need to change it. So, we wait until the node
  settles and then we decide the next_hop.

- **pref_next_hops:**
  Whenever a route is being setup, the initial packet is being broadcasted. So, a node
receives a packet from many nodes that are on the side of the source node. In this case we store the first five mac addresses of the neighbors from whom we have received the packet. We store this information because in case of mobility, we expect that getting to that node might be possible through any of these five neighbors. So, incase we couldn’t get to a node through the nexthop, we use these pref_next_hops in a round robin fashion, until one of them is a success. In case, if we couldn’t get to a destination through any of these five nodes, then we set that routing entry to invalid.

4.5.2 Timers

Timers play an important role, when it comes to performance and optimization. Here is a detailed description of timers used by WBridge.

- **Route Expiry Timer:**
  Each routing table entry has an age, and once this age expires this entry will be deleted. Route expiry timer is fired up every 100ms, and decrements the age of an entry by 100ms. If the age is negative the entry is deleted, before deleting the entry if there are any packets in the buffer, then they are sent out. If the route is valid, they are sent to the nexthop, if route is invalid it tries to send it to pref_next_hops, if none of them is a success, it discards the packets.

- **Null IP Timer:**
  To keep a route up-to-date, both the ends of the connection should communicate actively. If one of the ends is silent (like in case of UDP), then once in a while to keep the route up-to-date the WBridge layer sends out a Null IP packet (around 64 bytes packet). This also helps the other end know that this side node is up and running. In order to implement this mechanism, we have two flags, a sent and a recv flag. Whenever a packet is being sent or received the appropriate flags are set to that corresponding node. Null IP Timer checks periodically the recv flag, if this flag is set and sent flag is not set then it sends out a Null IP packet. In case, if it decides to send out a Null IP packet and if there are any data packets in the buffer corresponding to that destination, then those data packets are sent out instead of the Null IP packet.
• **Initial Phase Timer:**
  In the initial phase of a connection, upon receiving the first packet, a routing table entry is created for that packet's source IP address, if there doesn’t exist one. So at this point, the destination node receives the packets from many neighbors, as it is being broadcasted. Initial phase timer is fired up after the routing table entry is created and it sets nexthop to the latest nexthop.

• **Buffer Check Timer:**
  Whenever a route gets invalid, the first few packets are broadcasted and the remaining packets are buffered. Buffer Check Timer, checks the buffer periodically. For each routing entry, if the route gets valid and there are packets in the buffer to that corresponding destination, the packets are sent out, through that valid nexthop.

### 4.6 Chapter Summary

In this chapter, we understood the internal details of WBridge, and the way it is implemented. Flow chart presented provides greater details about the flow structure and how a packet is being processed, and what parameters are changed. Timers play a critical role, when it comes to performance and optimization. The next chapter presents the performance evaluation of WBridge compared to AODV and DSR.
CHAPTER 5

PERFORMANCE EVALUATION

In the previous chapter, we have seen how WBridge was implemented in NS2. In this chapter we compare the performance of WBridge routing protocol with that of famous AODV & DSR routing protocols in NS2 simulator.

5.1 Evaluation Setup

Simulations were run for networks of size 50 nodes by varying the number of connections from 10 to 100 and varying the speed of the nodes from 0 to 20 meters/sec. We have setup a rectangular field of 1500m x 300m, where the nodes are free to move randomly and the connections are completely random and a packet can originate from any node in the network to any other node. Each simulation is run for 900-simulated seconds. The setup described here is quite similar to the one in the performance comparison paper of AODV and DSR[10].

5.1.1 What exactly is a control message?

WBridge protocol does not have any control messages, ARP requests/replies, hello messages but it has a special data packet termed as Null IP packet. In these simulations, we count the Null IP packets as the control packets for WBridge routing protocol, as other routing protocols do not have this special packet. For AODV & DSR, we count the RREQ, RREP, RERR messages, ARP requests/replies, hello messages as the control overhead. For these simulations we have turned ON the Link Layer Detection. This mechanism suppresses the transmission of “Hello” messages in AODV in NS2 (actually this is the default mechanism for wireless simulations in NS2). As, one can see this a very fair consideration of the control messages, because WBridge does not use any of these packets and so in comparison these are routing control overhead.
Table 5.1. Average number of link connectivity changes during each 900-simulated seconds as a function of pause time

<table>
<thead>
<tr>
<th>Pause Time</th>
<th># of Connectivity Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>898</td>
</tr>
<tr>
<td>30</td>
<td>908</td>
</tr>
<tr>
<td>60</td>
<td>792</td>
</tr>
<tr>
<td>120</td>
<td>732</td>
</tr>
<tr>
<td>300</td>
<td>512</td>
</tr>
<tr>
<td>600</td>
<td>245</td>
</tr>
<tr>
<td>900</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pause Time</th>
<th># of Connectivity Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1m/s</td>
<td></td>
</tr>
<tr>
<td>20m/s</td>
<td></td>
</tr>
</tbody>
</table>

5.1.2 Cmu Mobility Scenarios & CBR connection patterns

We have used the CMU’s mobility scenarios & Constant Bit Rate (CBR) connection patterns from Monarch Project[4] which is available online at [11]. Table 5.1 shows a summary of the number of link connectivity changes in CMU scenarios.

Similarly, we have used the CMU’s CBR connection patterns, where the connections are completely random and we have set each packet size to be 64bytes. 4 packets/sec is the speed with which these CBR connections send out the packets for each connection. We vary the number of connections from 10 to 100, so the number of packets/sec varies from 40 packets/sec to 400 packets/sec for 900 simulated seconds and terminates exactly at 900 simulated seconds, without a cool down period.

5.1.3 Our own TCP Connection Patterns

CMU’s monarch project does not test the performance of these routing protocols based on TCP connections. So, we have used the same connection patterns of the CMU’s CBR but have changed it to TCP instead of UDP and to FTP instead of CBR. Basically, the connection from node and to node remains the same, but the connection parameters change to FTP on top of TCP.
<table>
<thead>
<tr>
<th>Timer</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ROUTE_EXPIRY_TIMER</td>
<td>5 sec</td>
</tr>
<tr>
<td>NULL_IP_TIMER</td>
<td>4 sec</td>
</tr>
<tr>
<td>MOBILITY_FAULT_TIMER</td>
<td>100 milli.sec</td>
</tr>
<tr>
<td>BUFFER_CHECK_TIMER</td>
<td>50 milli.sec</td>
</tr>
<tr>
<td>INITIAL_PHASE_TIMER</td>
<td>50 milli.sec</td>
</tr>
</tbody>
</table>

### 5.2 Different timers time period

Initially we did not know what to set the timers to, so we’ve been experimenting with the timers like ROUTE_EXPIRY_TIMER and NULL_IP_TIMER. Obviously, NULL_IP_TIMER should be less than ROUTE_EXPIRY_TIMER because you want to send a Null IP packet before the route expires. We found that 5 seconds for ROUTE_EXPIRY_TIMER and 4 seconds for NULL_IP_TIMER gave the best results in terms of minimizing the number of Null IP packets and maximizing the R/S factor. Here is a basic table which shows the R/S and Null IP packets for different ROUTE_EXPIRY_TIMER and NULL_IP_TIMER.

### 5.3 Performance evaluation results

We have measured factors like R/S, C/R, throughput, number of control packets and hop ratio for WBridge, AODV and DSR by varying the pause times and number of connections. We have varied the number of connections from 10 to 100 and Pause Times from 0 seconds (Highly mobile) to 900 seconds (static). For each pause time we have run 5 simulations and the results shown are the average of the 5 simulations.

- **R/S** is the ratio of number of packets received by the destination to that of number of packets sent by the source. It shows the fraction of packets that is received at the other end. If subtracted from 1, it represents the fraction of packets that are lost in transmit. The more this factor, the better the routing protocol, the smaller the worse.

- **C/R** shows the ratio of number of control packets to the number of data packets received by destination. It represents the number of control packets for each packet that is received. The smaller this factor, the better the routing protocol, the more the worse.

- **Throughput** is the number of bytes per second received by the destination. This does not include the control packets or the packets that are dropped.
Number of control packets represents the total number of control messages required to deliver the data packets. The definition of control message is given in Section 5.1.1. The control packets are counted on a hop-to-hop basis rather than on end-to-end basis. Let's say, if a control packet traverses 6 hops to get to destination, then this is counted as 6 control messages instead of one.

Hop Ratio is the ratio of the number of hops the packet has traversed to that of the optimal route, the less this factor the better the routing protocol, the more the worse.

Although in the real world there exists a combination of TCP traffic and UDP traffic in the network, here we present two cases. The case with active end points on both sides (TCP traffic) and the case where there is a silent end point on one side (UDP traffic).

5.3.1 TCP traffic

We have 3 parameters to vary, one is Pause Time (the time period that a node stays before or after moving), number of connections (which actually varies the congestion in the network, more connections more congestion and vice versa) and mobility speed (the speed with which the mobile node moves, or in other terms the rate at which the links change). While we vary one term, we keep the other two constant.

5.3.1.1 C/R

Figures from 5.1 to 5.3 shows the C/R value by varying one parameter at a time. C/R is the ratio of number of control packets to the number of data packets received by destination. In Figure 5.1, as WBridge has less control packets in comparison to data packets received, the C/R value stays low and in this case it is below 0.2 for all the pause times. DSR and AODV has more control packets for lower pause times when compared to the number of data packets received, and for larger pause times they have less number of control messages in comparison to data packets. The curve of WBridge is almost a straight line because the number of control packets in WBridge routing protocol remains the same as they are sent out every NULL_IP_TIMER time period, irrespective of pause times.

In Figure 5.2, we vary the number of TCP connections for 60 sec pause time and 20m/s mobility speed. Obviously, for less number of connections one has less control packets and
more control packets for higher number of connections. As WBridge has very less control packets in comparison to AODV and DSR, C/R remains low.

As one can see that, for C/R WBridge outperforms AODV and DSR. Except in case of higher pause times, where DSR outperforms WBridge by a factor of 0.05 (approx).

### 5.3.1.2 Control Packets

Control packets represent the overhead of the routing protocol in delivering the data packets, the less the control packets the better the routing protocol. Figures from 5.4 to 5.6 represent the control overhead of WBridge in comparison to AODV and DSR.

In Figure 5.4, one can see that WBridge’s number of control packets remains significantly lower than AODV and DSR. For 0 pause times, WBridge uses around 20K control packets, AODV uses around 90K control packets and DSR uses around 120K control packets. So, in comparison WBridge uses 4.5 times lower control packets than AODV and 6 times lower control packets than DSR. As the number of control messages remains the same in WBridge (because of the constant NULL_IP_TIMER), the curve of WBridge is almost a straight line. DSR at higher pause times like 900 sec, outperforms WBridge by a factor of around 2K
Figure 5.2. C/R vs Number of TCP Connections for 50 nodes and 60 sec pause time moving at 20m/s

Figure 5.3. C/R vs Mobility speed for 50 nodes - 30 TCP connections and 60 sec pause
control packets. So, WBridge outperforms AODV in case of control packets and outperforms DSR for most of the times except at higher pause times where DSR outperforms WBridge by a very low factor.

In Figure 5.5, we vary the number of TCP connections from 10 to 150, for a fixed 60 sec pause time and fixed mobility speed of 20m/s. One can see that number of control packets remains pretty low than AODV and DSR. As the number of connections increases, AODV uses more control packets than DSR and for 150 connections, AODV uses around 220K control packets and DSR uses around 125K control packets, while WBridge uses around 25K control packets.

5.3.1.3 R/S

The figures from Figure 5.7 to 5.9, show the R/S value by varying the pause times, number of connections and mobility speed. $R/S$ is the ratio of number of packets received by the destination to that of number of packets sent by the source. For all the graphs of R/S shown here, DSR outperforms WBridge and AODV, while WBridge outperforms AODV. The difference in R/S between DSR and WBridge is around 0.01, which is significantly low.
**Figure 5.5.** Number of Control Pkts vs Number of TCP Connections for 50 nodes - 60 sec pause time and nodes moving at 20m/s

**Figure 5.6.** Number of Control Packets vs Mobility speed for 50 nodes - 30 TCP connections and 60 sec pause
In Figure 5.7, as the pause time increases, i.e., the network becomes more static the R/S approaches 1 (no packet loss).

In Figure 5.8, we vary the number of TCP connections for 60 sec pause time and 20 m/s mobility speed. As, the number of connections increases the congestion in the network increases and more packets collide and we have more packet loss, so the R/S drops as the number of connections increases. As, in the previous case, DSR outperforms WBridge and AODV, while WBridge outperforms AODV. DSR outperforms WBridge by a factor of 0.01, which is significantly low.

5.3.1.4 Throughput

Throughput is computed by using the formula \( \frac{\text{Number of data pkts recv} \times \text{Packet size}}{\text{Simulation time}} \). Here, the number of data packets received are the number of data packets received by the destinations and not intermediate nodes. Figures from 5.10 to 5.12 represent the throughput of WBridge in comparison to AODV and DSR. As one can see that WBridge has less throughput than AODV and DSR, there are two reasons for this. WBridge has the mechanism of buffering the packets until that route gets valid after broadcasting a packet, which delays sending out
Figure 5.8. R/S vs Number of TCP connections for 50 nodes and 60 sec pause time moving at 20m/s

Figure 5.9. R/S vs Mobility speed for 50 nodes - 30 TCP connections and 60 sec pause
the packet for some time and in turn effects the throughput on the whole. Also, WBridge has the mechanism of Mac broadcasting the packet which in some cases has packet collisions and we get more packet loss, resulting in less throughput. In Figure 5.10, we vary the pause times for 30 TCP connections and 20m/s mobility speed. As, the pause time increases the network becomes static and we can route more packets with out failures. So, the throughput increases with the pause time.

Clearly, WBridge is being outperformed by AODV and DSR by a factor of around 15 - 25%.

5.3.1.5 Hop Ratio

The Figures from 5.13 to 5.15 shows the hop ratio versus pause time, number of connections and mobility speed. As one can see that WBridge doesn’t differ much from other routing protocols in case of hop ratio.
Figure 5.11. Throughput vs Number of TCP Connections for 50 nodes - 60 sec pause time and nodes moving at 20m/s

Figure 5.12. Throughput vs Mobility speed for 50 nodes - 30 TCP connections and 60 sec pause
Figure 5.13. Hop Ratio vs Pause Time for 50 nodes - 30 TCP connections and nodes moving at 20m/s

Figure 5.14. Hop Ratio vs Number of TCP Connections for 50 nodes - 60 sec pause time and nodes moving at 20m/s
Figure 5.15. Hop Ratio vs Mobility speed for 50 nodes - 30 TCP connections and 60 sec pause

5.3.2 UDP traffic

The same scenarios and connection patterns are used as for TCP, but the connection type is UDP and the application type is CBR, actually these are the original files provided by CMU.

5.3.2.1 C/R

The following figures from 5.16 to 5.18, show C/R parameter by varying the pause times, number of UDP connections and mobility speeds.

As shown in Figure 5.16, AODV has control packets almost 1.5 times that of received data packets, i.e., more control packets than data packets. Although, DSR has 0.8 C/R (approx), for 0 sec pause time, it reduces as the pause time increases. WBridge at any given point of time has C/R less than 0.2.

In Figure 5.17, we vary the number of UDP connections and determine C/R for 60 sec pause time and mobility speed of 20 m/s. As the number of connections increases, the C/R
value keeps falling for WBridge, because for more number of connections, suppression of Null IP packets takes place and C/R value drops.

WBridge outperforms AODV and DSR in case of C/R.

### 5.3.2.2 Control Packets

Control packets represent the overhead of the routing protocol in delivering the data packets. The following graphs from Figure 5.19 to Figure 5.21 show a detailed study of the number of control packets used by WBridge, AODV and DSR.

In Figure 5.19, AODV uses around 140K control packets for 0 sec pause time, whereas DSR uses around 75K control packets and WBridge uses around 12K control packets, which is more than 10 times less than AODV and more than 6 times less than DSR.

In Figure 5.20 we can see that, although all routing protocols use fewer control messages for lower number of connections, the main difference arises when the number of connections increases. For 150 connections AODV uses around 320K control packets and DSR uses around 180K control packets, while WBridge uses less than 20K control packets, which is extremely less than AODV and DSR.
Figure 5.17. C/R vs Number of UDP Connections for 50 nodes and 60 sec pause time moving at 20m/s

Figure 5.18. C/R vs Mobility speed for 50 nodes - 30 UDP connections and 60 sec pause
Figure 5.19. Number of Control Pkts vs Pause Time for 50 nodes - 30 UDP connections and nodes moving at 20m/s

Figure 5.20. Number of Control Pkts vs Number of UDP Connections for 50 nodes - 60 sec pause time and nodes moving at 20m/s
5.3.2.3 R/S

The following graphs from Figure 5.22 to Figure 5.24 show the R/S factor of WBridge, AODV and DSR by varying the Pause Times, number of connections and mobility speeds.

In Figure 5.22, WBridge has R/S factor lesser than AODV and DSR for less pause times. As pause time increases, WBridge approaches AODV. The main reason that can be attributed to this is, WBridge has the mechanism of Mac Broadcasting packets, so for very less pause times and for very high number of connections there happens to be a state where there are more route failures and more packets will be Mac Broadcasted which in turn get collided and will not be received.

In Figure 5.23, although WBridge and DSR give almost the same performance factor R/S, WBridge and DSR give up to 10% less R/S than AODV.

5.3.2.4 Throughput

Figures from 5.25 to 5.27 show the throughput of WBridge in comparison with AODV and DSR.
Figure 5.22. R/S vs Pause Time for 50 nodes and 30 UDP connections moving at 20m/s

Figure 5.23. R/S vs Number of UDP connections for 50 nodes and 60 sec pause time moving at 20m/s
Figure 5.24. R/S vs Mobility speed for 50 nodes - 30 UDP connections and 60 sec pause

The Figure  5.25, reflects the R/S graph  5.22 for 30 connections because in CBR, the send rate or the number of packets sent is the same. So R/S and throughput basically shows the same factor, number of packets received or R factor of R/S.

In Figure  5.26, the throughput of WBridge is comparable to that of DSR. For less number of connections all the routing protocols give almost the same throughput, but as the number of connections increases AODV outperforms DSR and WBridge.

In Figure  5.27, as the mobility speed increases the throughput gets dropped because of more link changes in shorter period of time. In this case, WBridge is being outperformed by AODV and DSR, by a factor of around 10- 20%.

5.3.2.5 Hop Ratio

The Figures from 5.28 to 5.30 shows the hop ratio versus pause time, number of connections and mobility speed.

Clearly, the difference between the hop ratio of WBridge differs by a factor of around 3% to 10% to that of AODV.
Figure 5.25. Throughput vs Pause Time for 50 nodes - 30 UDP connections and nodes moving at 20m/s

Figure 5.26. Throughput vs Number of UDP Connections for 50 nodes - 60 sec pause time and nodes moving at 20m/s
Figure 5.27. Throughput vs Mobility speed for 50 nodes - 30 UDP connections and 60 sec pause

Figure 5.28. Hop Ratio vs Pause Time for 50 nodes - 30 UDP connections and nodes moving at 20m/s
Figure 5.29. Hop Ratio vs Number of UDP Connections for 50 nodes - 60 sec pause time and nodes moving at 20m/s

Figure 5.30. Hop Ratio vs Mobility speed for 50 nodes - 30 UDP connections and 60 sec pause
CHAPTER 6

FUTURE WORK

A number of research issues remain to be addressed in WBridge protocol. Here we outline some of the salient issues. There needs to be a lot of work done to make WBridge practical and implement in the real world.

6.1 Making WBridge work for Multicast

Multicast is basically a one-to-many communication paradigm. One of the primary things that need to be done for WBridge is make it work for Multicast by using either data packets or Null IP packets and without introducing any new packet types. One might think of adding fields to the routing table, so that when a node/router sees that specific multicast address, it forwards the packets to those nodes.

6.2 Making it more efficient

Definitely, WBridge can perform far better than what we have right now, we could make it work more efficient in terms of control packets (for UDP like connections), and/or increasing R/S. As one can see that the control packets are being sent out every 0.8*ROUTE_EXPIRY_TIME time period. Even though the route stays the same like in case, even if there is no mobility, we send out a Null IP packet. So, if we can detect some mechanism to prevent sending this Null IP packets in case of static networks, we could reduce the number of Null IP packets, making WBridge more efficient.

6.3 Making timers adaptable to the network

Timers play a critical role in the performance of any routing protocol. One might observe that for some definite time period of some timer one might get good performance
on parameter X, and other time period might give good performance on parameter Y. So, instead of setting these timer’s time periods manually, the network and the nodes should figure out for themselves the optimal time period and should fine tune automatically. This is not just for WBridge, this should be done for all the Ad-hoc routing protocols, so that the timers are self adaptable.

One of the ways this can be done is by keeping track of the number of times (say, k times) the route has failed over some past time period say X sec. The average pause time of the nodes can be approximated to $X/k$ sec. If we already know the timers time period before hand, which gives maximum performance for the obtained pause time, we can set the timers to those values and route accordingly.

6.4 Extending WBridge to realistic 3-dimensional scenarios with obstacles in the radio path

Right now WBridge is simulated in 2 dimensional scenarios. In the real practical world the situation is different, we have a 3-dimensional scenario, obstacles which damp the transmitting radio waves, there would be disturbances in the radio path etc,. Although the basic mechanism of WBridge routing may work, we may encounter different problems which we haven’t expected. So, in order to make WBridge practical it should be extended to realistic 3-dimensional scenarios with obstacles in the radio path.

6.5 Summary

Although, WBridge in its current form provides good performance, there are a number of avenues for further work, still there needs to be a lot of work to be done in order to make it fully functional. There exists a scope for lot of research & development in WBridge and definitely in the field of Ad-hoc networks.
CHAPTER 7

CONCLUSION

WBridge is a simple, effective & intelligible Multihop Ad-hoc Wireless Routing Protocol, where there are very few control messages and offers many advantages over other famous routing protocols. WBridge extracts all possible details from a received packet, and with this information it can self sufficiently decide on the routing decisions. Each node making completely local routing decisions that require no co-ordination whatsoever among the intermediate nodes but lead to globally efficient packet deliver to destination.

WBridge works efficiently for small to very large and static to highly mobile networks. Although, it might perform little worse in case of UDP like connections in parameters like R/S, Throughput, this is not the common case. Current usage of the Internet is dominated by transmission control protocol (TCP) traffic such as remote terminal (e.g., Telnet), FTP, Web traffic, and electronic mail (e.g., SMTP). These TCP sources constitute 90 percent of all traffic[9], if same connection pattern exists in Wireless Ad hoc networks, WBridge will be a very efficient routing protocol in terms of control packets.

We expect that further development of the protocol will take place and implementing this protocol in the real world will be accomplished.
REFERENCES


BIOGRAPHICAL SKETCH

Navodaya Garepalli

Navodaya Garepalli (garepall@cs.fsu.edu), is a Master’s student in Computer Science at Florida State University. He received his B.Tech degree in Computer Science & Engineering from Jyothishmathi Institute of Technology & Science (JITS), Affiliated to Jawaharlal Nehru Technological University (JNTU), Hyderabad, India in 2002. His research interests are in the areas of Wireless Ad-hoc networks, Communication Protocols, Operating Systems. He is a member of Florida State University’s chapter of Upsilon Pi Epsilon (UPE), a computer science honor society.