

**WiCTP: A Token-based  
Access Control  
Mechanism for Wireless  
Networks**

Raal Goff

*This report is submitted as partial fulfilment  
of the requirements for the Honours Programme of the  
School of Computer Science and Software Engineering,  
The University of Western Australia,  
2004*

# Abstract

The introduction and growth of large, metropolitan-sized wireless networks has identified a significant problem within the current 802.11 protocol specification; Hidden Nodes. A wireless network may have 2 nodes, **A** and **B**, connected to a central Access Point (AP). Node **A** is within the transmission radius of the AP, but outside the transmission radius of **B**, and vice versa. Nodes **A** and **B** are then said to be hidden from each other. Ordinarily, this does not pose a significant problem, since most nodes on a wireless network are within the transmission radius of each other, but in scenarios where all nodes are hidden it creates a significant decrease in network performance.

802.11 wireless networks use Carrier Sense Multiple Access with Collision Avoidance to determine whether they are able to send collision-free transmissions. They sense the medium to determine if any other nodes are transferring data, and if it is free, they begin their transfer. However, when nodes are hidden from each other, nodes cannot determine if the hidden nodes are transferring. Thus, a hidden node **A** may start a transfer to a common node **C** while the hidden node **B** is also transferring to **C**. This scenario results in the packets colliding at **C** and nodes **A** and **B** have to retransmit. **A** and **B** never know if the other is transferring, so this scenario repeats itself for the life of the network.

The standard way of dealing with this problem is to use the Request to Send/Clear to Send (RTS/CTS) protocol. RTS/CTS secures the medium for a node first, then transfer data, reducing the risk of collisions. A node using RTS/CTS will send a RTS packet with a preset duration and destination contained within. Any node hearing this will cease transmission for the duration contained in the packet. The destination will receive the RTS packet and reply to the sender with a CTS packet, which also contains a duration. Any nodes hearing the CTS will cease transmission for the duration in the CTS packet. Thus, the sender node is reasonably sure the medium is free for them to transfer data. Unfortunately RTS/CTS suffers from many problems in scenarios where many nodes are hidden. RTS and CTS packets often collide and require transmission, falling victim to the same problem they are attempting to solve.

To combat the hidden node problem, this study proposes a new protocol called Wireless Cyclic Token Protocol (WiCTP). Based on current token-passing access control mechanisms such as Frottle and WiCCP, it uses a token packet to control access to the medium. A master node will pass a token to each node on

the network, which enables the node to transfer for a short amount of time. Only one node is in possession of the token at once, thus only one node may transfer at a time. Since no two nodes can transfer simultaneously, collisions are eliminated and the hidden node problem is solved.

WiCTP successfully improves the stability and performance of a real wireless network where hidden nodes are present with only a small cost to overall network bandwidth and response times. Tested against standard CSMA/CA, WiCTP distributed the bandwidth effectively between all nodes on the network, whereas CSMA/CA suffered from collisions and low throughput. Under WiCTP, error rates were significantly decreased and throughput was kept at near maximum for all tests.

**Keywords:** 802.11, wireless, networks

**CR Categories:** C.2.5

# Acknowledgements

The author would like to thank the members of Leeming Wireless Network, particularly Adam Prior, Simon Prior, and Matthew Drew for their assistance in gathering results and allowing him to run several disrupting tests over many weekends. Without their help, this project would not have become a reality. The author also wishes to thank Bree Goff for her valuable assistance in proof-reading and correcting drafts.

# Contents

<b>Abstract</b>	<b>iii</b>
<b>Acknowledgements</b>	<b>iv</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Wired vs Wireless Ethernet . . . . .	2
1.2 The Hidden Node Problem . . . . .	3
1.3 Current Solutions . . . . .	5
1.4 New Solutions . . . . .	6
1.5 Previous Work on Token-Based Solutions . . . . .	8
<b>2 Wireless Cyclic Token Protocol</b>	<b>12</b>
2.1 Overview . . . . .	12
2.2 Linux Kernel Modules . . . . .	13
2.3 The New Protocol as a Linux Kernel Module . . . . .	13
2.4 Slave Operation . . . . .	13
2.5 Master Node Operation . . . . .	18
<b>3 Method</b>	<b>21</b>
3.1 Initial Tests . . . . .	24
<b>4 Results</b>	<b>27</b>
4.1 Experimental Overview . . . . .	27
4.2 TCP Throughput . . . . .	28
4.3 Ping Times . . . . .	29
4.4 Error Rates . . . . .	29
4.5 Discussion . . . . .	30

<b>5 Conclusion</b>	<b>34</b>
<b>A Original Honours Proposal</b>	<b>36</b>
<b>B Example netsync script</b>	<b>40</b>

# List of Figures

1.1	The Hidden Node Problem: Nodes <b>A</b> and <b>B</b> can both send to the Access Point ( <b>AP</b> ), but cannot send to each other. . . . .	4
1.2	WiCCP sends data to the master first, instead of straight to the destination. . . . .	10
2.1	WiCTP is implemented between the Network and Data Link layers.	14
2.2	Data and control flow of the slave nodes. . . . .	15
2.3	The PACKET_ANNOUNCE packet format. . . . .	16
2.4	The PACKET_TOKEN packet format. . . . .	16
2.5	The PACKET_ACK packet format. . . . .	17
2.6	The PACKET_TOKEN packet format when being returned to the master.	17
2.7	The PACKET_DATA packet format. . . . .	18
2.8	Data and control flow of the master node. . . . .	19
3.1	A rough layout of the Leeming Wireless Network. . . . .	22
4.1	Combined(RX + TX) TCP Throughput using CSMA/CA. . . . .	28
4.2	Combined(RX + TX) TCP Throughput using WiCTP. . . . .	29
4.3	Ping time for WiCTP. . . . .	30
4.4	Ping time for CSMA/CA. . . . .	31
4.5	Error Rates for WiCTP. . . . .	32
4.6	Error Rates for CSMA/CA. . . . .	33

## CHAPTER 1

# Introduction

Since the introduction of the Wireless Fidelity (WiFi) standard [9] in 1997, the use of wireless networks has increased dramatically. While initially costly and having a somewhat slow transfer rate of 2Mbps, later improvements of 802.11b increased the transfer rate to 11Mbps. Further improvements were made with the introduction of 802.11a and 802.11g, increasing the transfer rate further to 54Mbps. Greater market penetration and demand also saw a fall in consumer and enterprise level equipment costs, enabling WiFi to become a viable alternative for wired Ethernet (802.3) [8].

The use of a wireless, instead of a wired, network enables a greater flexibility for workstation placement in offices and the home, as no cabling is needed to connect a computer to the network. Wireless cards have also been incorporated into laptops in order to give the user greater flexibility in where they can connect to the network. These applications of the wireless standard are exactly what the protocol was designed for, allowing greater flexibility in the way computers connect to a local network.

However, some users of WiFi have used their equipment in conjunction with external antennas to extend the range of the protocol dramatically. By connecting standard 802.11 equipment to external antennas and signal amplifiers, users are able to extend their network range from a maximum of 500m, to distances greater than 10km. The 500m maximum is an effect of the poor quality of antennae supplied by consumer equipment, so changing the quality of antenna on the hardware, enables a far greater transmission radius. Surprisingly, the protocol works reasonably well in these situations, given what it was initially designed for. Unfortunately, the protocol has one major problem that prevents it from working efficiently in these situations, the Hidden Node problem.

The Hidden Node problem reduces the throughput of a wireless network so dramatically, a solution is needed to enable these extended networks to function more effectively. By using a token-passing access control mechanism, the hidden node problem can be eliminated from the network, allowing nodes to transfer at

a much more stable throughput. Unfortunately, using such a scheme has some drawbacks, such as increased network latency, but overall, a token-passing scheme results in a much more stable network.

## 1.1 Wired vs Wireless Ethernet

Wireless networks face several unique problems compared to their wired counterparts. Using wires as the physical medium allows a protocol, such as ethernet, to have dedicated wires, and consequently a dedicated medium, for transferring and receiving data. Thus, a wired ethernet network card can send and receive data simultaneously, allowing a greater transfer rate than wireless ethernet, and also allowing an individual node to determine whether other nodes are using the network while they are still transferring. Wireless protocols generally only have one medium, the radio frequency to which they are tuned. Thus they cannot determine whether other nodes are transferring if they are transferring at the same time since their own signal will override any signal received from other nodes such that they can only determine that they are transferring. Having only one medium to transfer over also limits the access control mechanisms the network can use.

Both wired, and wireless ethernet (WiFi) use Collision Sense Multiple Access (CSMA) to control access to the medium [8, 9]. Since a node on a wired network can transfer and sense the medium concurrently, it is able to use Carrier Sense Multiple Access with Collision Detection (CSMA/CD) to control access to the medium, while WiFi must use Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA). Under CSMA, a node on the network must first sense the medium to determine if any transfers are in progress. The node only proceeds to transfer data if it determines that the medium is not in use by any other node. This process occurs in both wired and wireless ethernet. The differences occur in how each handles collisions. Wired ethernet uses Collision Detection (CD) while wireless ethernet uses Collision Avoidance (CA). Under Collision Detection a node will continue to sense the network whilst transmitting. Thus, if a node detects that another node has begun transmitting on the medium while it is, the node can immediately cease its transmission and send a jamming sequence. The other transferring node will detect the jamming sequence and cease its transmission as well. Thus, the medium becomes free again after a time.

Wireless ethernet cannot sense the medium during transmission, and as such, cannot utilise Collision Detection. Under Collision Avoidance, when a node determines the medium is free, it will initialise a backoff timer to a random time and wait until it expires before sensing the medium again and transferring. If

the medium is found to be busy, the node will freeze its backoff timer until it becomes free again. The idea behind such behaviour is to avoid collisions in the scenario where they are most likely to occur, i.e when the medium becomes free. The major flaw behind using Collision Avoidance for wireless networks is that nodes may not be able to sense the entire network for activity. Nodes can, and often are, outside of the transmission range of each other, which creates problems such as long-range interference. For a transmission to interfere with another transmission, it requires significantly less signal strength than a transmission that needs to be received successfully. Thus, while two nodes may be out of range of each other, their transmissions still travel to one another. Although these transmissions are significantly weaker than successful transmissions, they can still interfere with packets and cause them to be retransmitted [16].

## 1.2 The Hidden Node Problem

In networks such as Freenets [10], nodes are often outside of each others' ranges. Each individual node cannot determine whether every node on the network is idle and not transferring any data. In these situations the Hidden Node problem becomes an important factor. The Hidden Node problem arises when one or more nodes on the network cannot determine if another node is currently transferring, because it is outside of the node's transmission range. These nodes, considered to be hidden from each other, are termed Hidden Nodes. For example, a wireless network may have two nodes, **A** and **B**, connected to a central Access Point (**AP**). Node **A** is within the transmission radius of the **AP**, but outside the transmission radius of **B**, and vice versa. Nodes **A** and **B** are then said to be hidden from one other. They cannot receive transmissions sent by each other, but can both send and receive to the **AP** (Figure 1.1). The **AP** cannot send a jamming sequence, like under CSMA/CD, because nodes **A** and **B** cannot sense the medium whilst they are transferring. Thus, nodes **A** and **B** will only discover a collision when they have finished their transfer and they receive no acknowledgement packet from the destination.

In standard WiFi set ups, nodes are mostly within transmission range of one another. A few outlying nodes may be hidden from the rest of the network, but most nodes are able to detect whether the medium is in use by other nodes. In these situations, the few hidden nodes that are present, do not impact the performance of the network considerably. However, when the WiFi network range is extended beyond its initial design, for example, in Freenets and long-range connections, nodes are commonly hidden from one another.

When all nodes are hidden from one another, each node senses the medium to

### The Hidden Node Problem

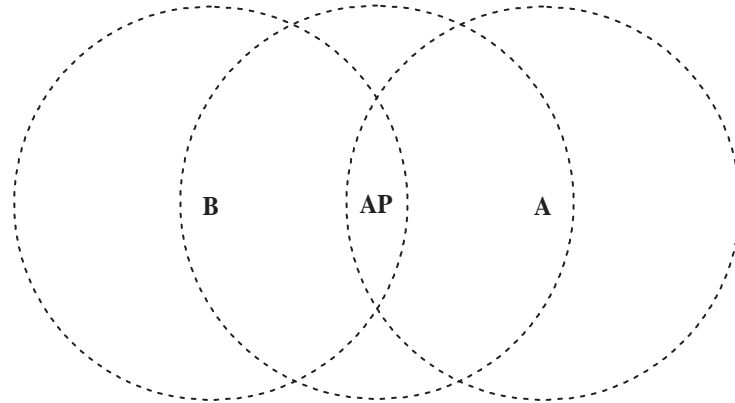


Figure 1.1: The Hidden Node Problem: Nodes **A** and **B** can both send to the Access Point (**AP**), but cannot send to each other.

determine if it is suitable to transfer, and hearing no traffic, proceeds to transfer. If no other nodes were transferring at the same time, the network would not be affected, but often, several nodes wish to transfer simultaneously. Thus, several nodes sense the medium at the same time, and see, falsely, that the medium is free. Each node transfers concurrently, and the packets then collide at the central AP. As it is impossible to recover collided packets, when a packet collides, it must be resent. Each individual node determines that its packet has collided with another, due to the lack of MAC-level acknowledgement packets, and waits before sensing the medium again and transferring their packet.

As this process continues, the number of collisions on the network increases dramatically, since each node continually believes that the medium is free, and will transfer their packet after each wait period. Each node will wait for a random time after their packet collides, but even if they do, their wait timer may end while another node is still transferring a packet. Without knowing conclusively if the medium is free, nodes still have a high chance of their packets colliding with each other. If each node continues to transfer, more collisions occur and so on, until the network degrades to a point where most protocols will fail.

Khurana et al. have presented results detailing how the presence of hidden nodes affects wireless networks [11]. When approximately 10% of nodes are within the transmission range of one another the wireless LAN can still offer reasonable performance, with a throughput of around 65%. However, when the number of hidden nodes climbs above 10%, the network performance degrades dramatically.

When the number of hidden nodes climbs to 30%, the network performance drops to only 22% throughput.

The simulation model Khurana et al. created for modelling networks with hidden nodes did not account for bit errors in the channel. Bit errors can occur in a wireless network due to signal degradation over large distances and barrier objects. Thus, the high performance drops seen in Khurana et al's simulations could be further increased due to bit errors, which are more prevalent in networks such as Freenets, where the distances involved between nodes are much greater than normal.

### 1.3 Current Solutions

Currently, the only widespread method of combating the Hidden Node problem is the Request to Send/Clear to Send (RTS/CTS) protocol. The RTS/CTS protocol allows nodes to reserve the medium for a short time so they may send their data collision free. Under RTS/CTS, each node has a preset RTS threshold. If a data packet being sent is larger than this preset threshold, the node will initiate an RTS/CTS exchange by sending a RTS packet. The RTS packet contains a duration field that is set to the time period it would take to transmit the data frame. All nodes hearing this packet set their Network Allocation Vector (NAV) to the duration in the RTS packet. Nodes wishing to transfer on the medium only do so after their NAV has expired and they physically sense that the network is free, thus any node hearing the RTS packet stops transferring for the time contained within. The destination node replies to the RTS with a CTS, which also contains duration information. All nodes hearing the CTS packet set their NAVs once again. The sender can then send the data packet since all nodes have set their NAVs for both the RTS and CTS packets. If a node did not hear the RTS packet, it is likely that they heard the CTS packet and vice versa. Thus, the sender can be reasonably sure that the network is free.

The RTS/CTS protocol works well in situations where most nodes are in close proximity, and can receive any RTS/CTS packet sent. If only one or two nodes are outside of the main group of nodes, RTS/CTS works well to combat the hidden node problem [11]. However, in situations where nodes are not within the transmission range of the receiver, and thus cannot hear the CTS packet, RTS/CTS suffers considerably [16].

The hidden node problem can also adversely affect RTS/CTS. Nodes wishing to send RTS packets will first sense the medium, and in a hidden node environment, will always determine that the medium is free. RTS packets then collide

with each other and require re-transmission in order to successfully reach their destination.

The RTS Threshold is another limiting factor on how useful the protocol can be. While it is less likely that smaller packets will collide, many small packets sent frequently will certainly cause collisions on the medium. However, RTS/CTS introduces delay in packet sending, so the RTS Threshold cannot be set too low, else the network performance will be degraded by the RTS delay.

RTS/CTS is unavailable in most consumer-grade hardware. While many enterprise-grade products include RTS/CTS implementations, consumer-grade hardware, of which most Freenets are made, do not include the protocol. RTS/CTS is thus unavailable to the networks that most require a solution. This situation is changing, with many consumer-grade hardware being sold with more features, but older hardware may not include the RTS/CTS protocol, and it is still an optional part of the 802.11 standard. RTS/CTS is still a suitable solution for some scenarios, as Khurana et al. [11] have shown, but in the case of Freenets, its effects are negligible on the overall performance of the network.

## 1.4 New Solutions

While the hidden node problem is pervasive in long-range networks that use 802.11 equipment, there has been little research into solutions to combat the problem. Of the research that does investigate solutions to hidden nodes, most study solutions which require a significant change in how the underlying MAC protocols themselves operate. Thus, they are rarely able to be implemented in current hardware solutions without risky firmware upgrades.

The Floor Acquisition Multiple Access (FAMA) protocols are two protocols that have been created specifically to alleviate the problems that hidden nodes cause. The FAMA protocols integrate a three-way Request to Send/Clear to Send (RTS/CTS) handshake with packet or carrier sensing and will compensate for performance issues associated with hidden nodes [7]. The FAMA protocol requires a station wishing to transmit to gain access to the medium before transferring. The medium is acquired by an RTS/CTS request followed by data such that if the RTS/CTS collides with other transmissions, the data will still proceed without collision. A station sends a RTS packet to the intended destination, and the receiver replies with a CTS packet that is transmitted for an interval of time such that any station that would not hear an ordinary CTS is jammed. Since the CTS is broadcast to every station, as soon as an error free CTS is received, the receiving node realises that the station to which the CTS is addressed has

acquired the medium.

There are two variations of the FAMA protocol, FAMA with Non Persistent Carrier Sensing (FAMA-NCS) and FAMA with Non Persistent Packet Sensing (FAMA-NPS) [7]. Each has slightly different RTS and CTS transmission times. In FAMA-NCS the CTS transmission length is larger than a RTS, making it the dominant packet, allowing any station, even one that has just started sending a RTS, to hear at least a partial CTS response. FAMA-NPS has an identical transmission length for both RTS and CTS, which is longer than the maximum round trip delay for the network. Any station that receives an RTS will wait for the time specified in the RTS, allowing an error free CTS to be sent.

Simulations comparing the FAMA-NCS protocol with the MACAW [6] protocol showed that the FAMA-NCS protocol offered substantial improvements in performance over MACAW, supporting the results Fullmer and Garcia-Luna-Aceves see in their analysis of the protocols [7]. The simulations showed that FAMA-NCS had both a higher throughput and a lower latency, even in the presence of hidden nodes. A discouraging factor of this solution is that it requires a re-write of the lower-levels of the networking protocols and, as such, Fullmer et al's [7] method cannot be implemented on top of existing protocols.

Other research has focused mostly on investigating how the use of directional antennae can be used to improve wireless communications. Korakis et al [13] propose a new MAC protocol that utilises an array of directional antennae to improve wireless communication and effectively remove the hidden node problem. By using an array of directional antennas their proposed protocol sends a RTS packet on each antenna consecutively until the entire area around the transmitter is covered. By taking note of which antenna RTS packets arrive on, nodes may determine the general physical layout of the network, and what nodes may be contacted via the different antennae. Thus, the receiving nodes can decide whether to defer communication along that antenna depending on a defined algorithm. This reduces the risk that two nodes will transfer along the same direction resulting in fewer collisions.

The protocol was tested by Korakis et al in simulated environments where an array of four and eight antennas were used. Under scenarios with varying node layouts the protocol caused a throughput increase of 34% where four antennas were used, and 42% where eight antennas were used compared against the standard 802.11 MAC using an omni-directional antenna [13]. Under low load conditions the protocol saw decreased performance because, in such conditions, the directional transmissions do not have any added benefit, thus the transmission overhead causes degraded performance. Under high load levels the transmission overhead is cancelled by the increase in throughput due to using the

directional antennas.

While it is a novel solution to several problems in long-range networks, it does require an array of directional antennas as well as a mechanism for switching between them. Current 802.11 standard equipment does not support this, thus it would not be possible to implement this protocol, or any protocol using an array of directional antennas, using current hardware.

## 1.5 Previous Work on Token-Based Solutions

Token-based access control protocols are not new. Other protocols that implement a similar access control mechanism include the open source projects Frottle [12], and the Wireless Central Co-ordinated Protocol (WiCCP) [5]. Both protocols create a master node that controls access to the medium by use of a token that is exchanged between nodes on the network. Client nodes will queue data packets they have to send and when they receive the token they send queued packets according to the data contained within the token. The method is simple and effectively eliminates the hidden node problem whilst increasing network stability because no two nodes can transfer at any one time.

### Frottle

Frottle is a Linux-only implementation of a token-based access control mechanism. It currently relies on the Linux kernel's iptables [2] packet filtering abilities to control access to the network, and uses the Transmission Control Protocol and Internet Protocol (TCP/IP) stack to communicate between the master and clients. It runs as a user space application and utilises the iptables QUEUE rule set to queue packets for transfer.

The Frottle package communicates between master and client nodes via TCP/IP port 999. This is a major disadvantage, since its reliance on both TCP and iptables limits its portability. Without iptables, Frottle has no effective way of controlling packet queuing, and without a TCP/IP stack, it cannot communicate between nodes. For any embedded system wishing to run Frottle, they must have both the TCP/IP stack and the iptables modules compiled into the kernel, which increases the size of the kernel dramatically. While TCP/IP would almost certainly be compiled into any networked system, iptables can increase the required size by many kilobytes, a serious concern for any embedded system.

The Frottle package also allows unrestricted access on port 999, a significant weakness. If another server runs a service on port 999, a user who is part of

the Frottle ring can circumvent the token access control and directly access the service. This can realise a dramatic drop in performance and in cases where the service is heavily used, allow the hidden node problem to re-appear. While port 999 is not used by any standard network service, a user could circumvent the access control on the network in order to achieve higher performance at the expense of other users on the network.

Frottle does not contain auto-detection of master nodes. Thus, each node must be set up individually, and have at least one network interface of the same subnet as the master in order to exchange control packets. This severely limits the flexibility of the protocol, as a protocol with auto-detection of master nodes and no reliance on TCP/IP would effectively allow multiple subnets on the same access point. This would increase the number of possible network set-ups many times over.

While Frottle has been available since August 2003, development of the protocol seems to have ceased, and presently, no new version has been released. The results of a Frottle implementation on a network are also largely anecdotal, with no solid results available for comparison against the standard CSMA/CA access control mechanism. This, combined with its numerous implementation disadvantages, leads users to seek alternatives to Frottle.

## The Wireless Central Co-ordinated Protocol

The Wireless Central Co-ordinated Protocol (WiCCP) is one such alternative. This package, unlike Frottle, is a kernel-level implementation of a cyclic token passing protocol. It functions in much the same way as Frottle, queuing data packets until it receives a token that contains data on how many packets the node may send before returning the token. WiCCP is implemented as a Linux Kernel Module (LKM) and thus has no reliance on a TCP/IP stack or iptables. WiCCP addresses most of the problems associated with Frottle. It has no reliance on TCP/IP or iptables and automatically discovers master nodes and associates with them. WiCCP also encapsulates higher-level protocols in its own custom protocol, ensuring that any higher-level protocol can be used concurrently. Since it uses its own low-level protocol to communicate, masters and clients need not be on the same network subnet. Nodes only need to be connected to the same access point in order to receive the benefits of the WiCCP access control. This allows far greater flexibility in network set-ups than any Frottle set-up could achieve.

However, WiCCP is not perfect. Anecdotal evidence suggests that WiCCP may be much slower than Frottle, and have a higher level of packet loss in low quality connections. This could be due to WiCCP's management of packet trans-

fer, and the way it associates with new nodes. WiCCP nodes will encapsulate queued data in a WiCCP packet header and send it directly to the master node, disregarding its original recipient. When the packet reaches the master node, it then forwards the packet to the correct node. The purpose of such a mechanism seems counter-intuitive, as it should decrease network performance.

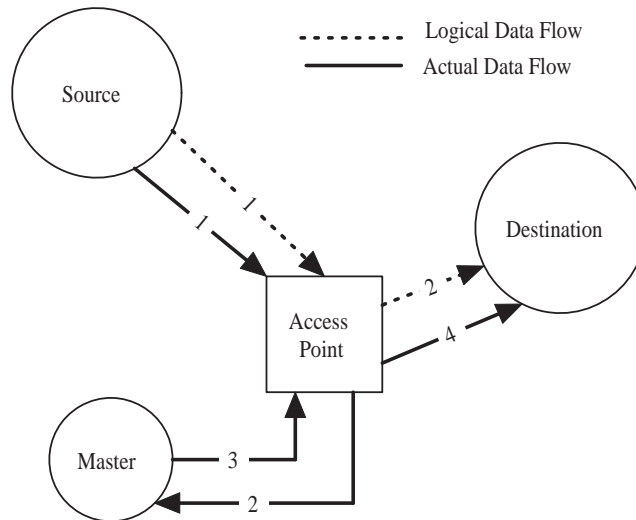


Figure 1.2: WiCCP sends data to the master first, instead of straight to the destination.

Problems also arise under WiCCP if the token is ever lost due to transmission errors, a large flaw in the implementation. While hidden nodes are commonly the reason most packets collide in long-range networks, they certainly do not cause all transmission errors. When the Signal to Noise Ratio (SNR) of the connection is low, transmissions are often lost due to absorption by barrier objects such as trees and walls. WiFi transmissions can also be corrupted by interference from other signals produced from devices running on the 2.4GHz spectrum, such as mobile phones. While these other transmissions do not directly interfere with the WiFi standards, enough anecdotal evidence has been compiled to show that they do create reception problems.

Reception problems can cause the token to be lost in transmission, and without proper checks or timeouts, the master does not know whether the node has received it. WiCCP takes advantage of the lower level driver's retransmission protocol and MAC level acknowledgement frames that are part of the standard, but these will eventually exhaust their retransmission algorithm such that the packet will be discarded, and the WiCCP package has no way of knowing when.

Thus, in a low SNR environment, arguably the environment where such a solution is most needed, WiCCP could still be unstable due to loss of tokens.

Node-master associations are inefficient under WiCCP. WiCCP specifies that at given intervals (the specification suggests 5 seconds), the master sends a Free Contention Period (FCP) packet. This packet is a signal for any un-associated nodes to send their association packet to the master. Thus, any node wishing to associate with a master has to wait for the FCP to arrive before being able to join the network. Depending on the implementation of the protocol, any time period could be used. A large time could mean nodes are waiting too long to become associated with nodes, and a short time would mean more network time is lost waiting for new nodes.

As with Frottle, development of WiCCP seems to have ceased. The latest release available as of October 2004 is version 0.5, available as both a Linux kernel module and a Windows XP driver. Thus, it is more attractive to general users as Linux is not required to utilise the protocol. Unfortunately, the Windows driver only supports client mode, but this is still preferable to Frottle, which has many more dependencies.

WiCCP has greater flexibility when designing a network, while Frottle seems to have a greater level of performance. Both adequately eliminate the hidden node problem, but issues with token loss remain, particularly in WiCCP. Each protocol has unique problems, which provide valuable insights into possible pitfalls in the WiCTP design.

## CHAPTER 2

# Wireless Cyclic Token Protocol

## 2.1 Overview

The proposed protocol implements a token-passing access control mechanism, similar to the mechanism implemented by both Frottle and WiCCP, called the Wireless Cyclic Token Protocol (WiCTP). WiCTP works by passing a token to each node on the network. Only the node in possession of the token may transfer data, allowing only one node to transfer at any one time, thus eliminating collisions entirely. Each node on the network is classed as either a master or a slave. The master node has knowledge about every other node on the network, but each slave only knows about the master. The master node controls all access to the network, since it handles the distribution of the token to each node.

This allows the master node vast flexibility in its control of the network. It can re-order the passing of the token to better utilise the network, or give preferential treatment to certain nodes, allowing the master to ensure that nodes that normally would be relegated to low performance receive a fair share of network bandwidth. While not yet implemented, WiCTP supports the integration of different Quality of Service (QoS) algorithms.

When a node connects to the wireless network, it sends a packet announcing its presence to the network. When the master receives this packet, it replies to the node with data about the master to which it has been associated, and makes the node a slave. The master adds the new slave's information to its list of current slaves and places it in the queue for receiving the token. When the slave receives the token, it checks to see if the token is from the master to which it has been associated. If so, it will check for any data that needs to be sent, and examine the token to see how many data packets it may send as allocated by the master. The slave will then send its queue of data packets to the recipients. When the slave has no more data packets to send, or it has sent the maximum number of packets, it sends the token back to the master. Since only the slave with the token may transmit data, the hidden node problem is eliminated. No

two nodes can transfer at the same time, thus collisions are eliminated entirely.

## 2.2 Linux Kernel Modules

Linux Kernel Modules (LKMs) are modular pieces of code that can be loaded and unloaded, into and out of the Linux kernel without needing to restart the Linux machine. A Linux feature since version 1.2, they are now a well-used aspect of the operating system. Most Linux kernel features can be built as LKMs, reducing the kernel size and allowing greater kernel configuration flexibility, without the need for rebooting or recompiling the kernel itself.

There are many benefits of using a LKM for a new piece of kernel code. A LKM can be loaded and unloaded, and does not require a kernel recompile when new versions are tested. One version can be compiled, loaded, tested and unloaded, then recompiled with changes and loaded again, all without requiring the machine to reboot. It is also easier to change loading parameters and can even be debugged with a debugger such as gdb. Such flexibility would not be possible if the code were changed within the kernel, since a kernel recompile, and more importantly, a reboot, would be required each time the code was altered. As such, building a LKM was the most suitable method of implementing the new protocol in Linux.

## 2.3 The New Protocol as a Linux Kernel Module

By creating the new protocol implementation as a LKM, the code path dealing with the new protocol could be implemented wherever was most suitable. In this case, a new layer was inserted between the network layer and the data link layer (Figure 3.1). This allowed the current wireless or Ethernet driver to be utilised, but the new protocol could intercept data link layer packets and alter them before being sent, essentially making the protocol totally transparent to any higher layer. This transparency was an important goal, since it allows higher-level protocols to be used with the new protocol.

## 2.4 Slave Operation

As described previously, nodes operating under the new protocol are classified as either masters or slaves. Each network has only one master, which controls

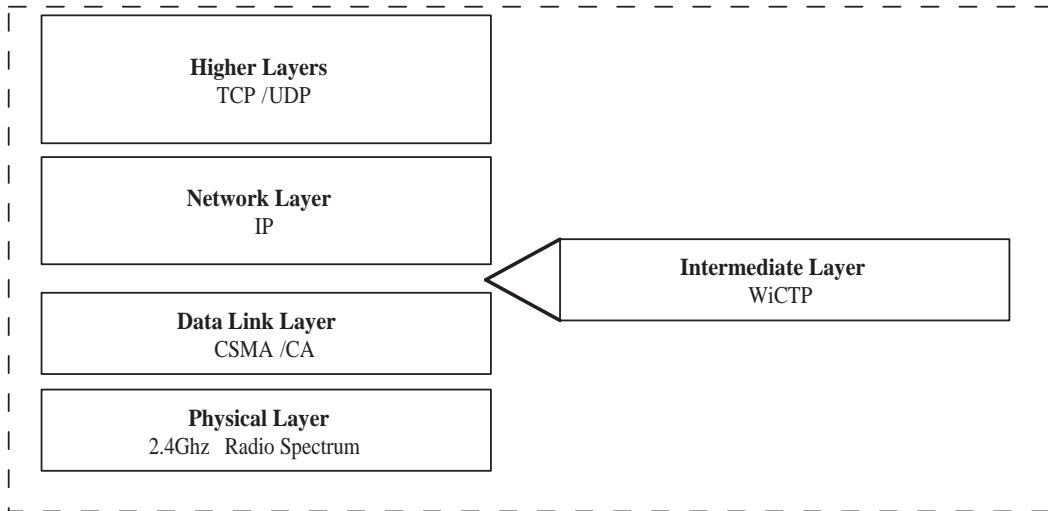


Figure 2.1: WiCTP is implemented between the Network and Data Link layers.

the token packet. The implementation of the new protocol for this project has been built as a LKM and resides between the Data Link Layer and the Network Layer. The LKM creates a new interface, named `wethx` that attaches to an already existing interface, where `x` is the interface number (e.g, attaching to `eth2` would create an interface called `weth2`).

The new interface operates like any other interface, except that any packets sent across the new interface are encapsulated in the new protocol. Like any other interface, the new interface can be associated with an IPv4, IPv6, or any other protocol address. Aliases and other such functions will also work with the new interface. Unfortunately, if the original interface to which the LKM was bound goes down, the new interface will also go down, i.e, if `eth1` were to go down (perhaps with `ifconfig eth1 down`), `weth1` would also go down and be removed as a viable network interface for any user-space programs.

Whether the new interface is used to send data or not is usually determined by the routing table of the computer, but there may be instances where the user-level program explicitly specifies the new interface as the network interface to be used. As far as any user-space program is concerned, the new interface is simply another Ethernet interface available for use.

When the slave LKM code is loaded into the kernel of a slave node, it initialises the new interface and associates it with the Ethernet device that was specified by the module loader. If none was specified, the interface `eth0` is used as the default. The LKM also registers several packet handlers for the following packets:

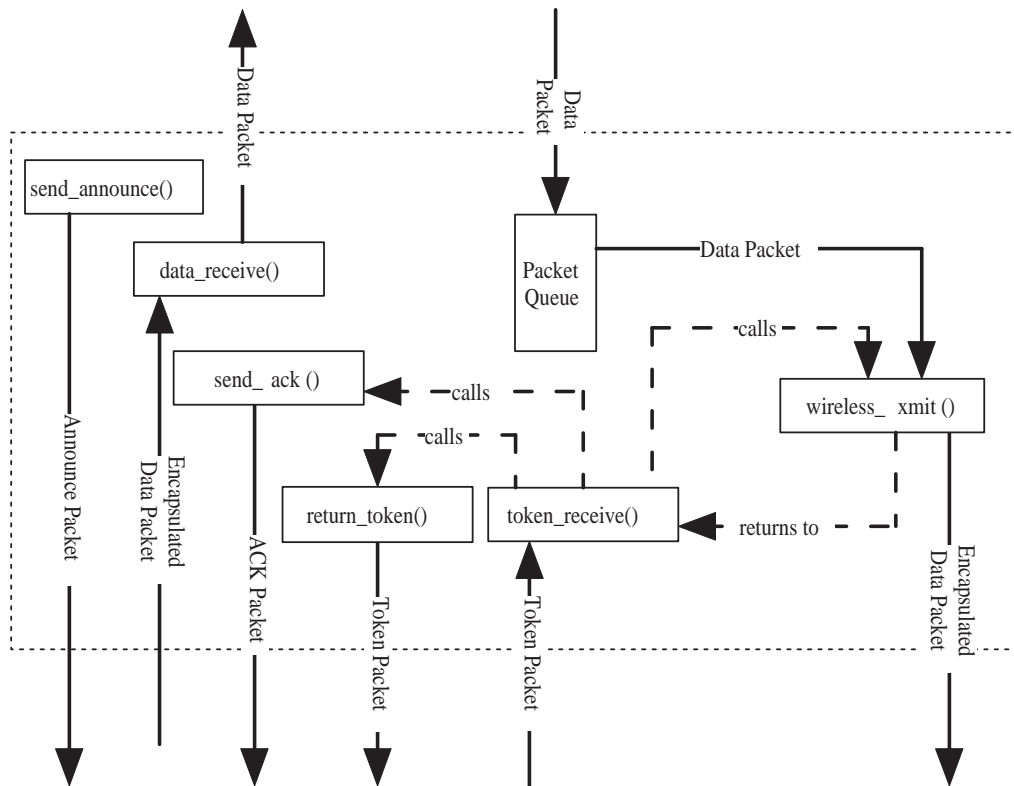


Figure 2.2: Data and control flow of the slave nodes.

PACKET\_ACK: 0xAA01

PACKET\_TOKEN: 0xAA02

PACKET\_DATA: 0xAA03

Ordinarily, no packets with these identifiers would be sent along the network, since all common protocols, such as IPv4, IPv6, and AppleTalk, do not use these identifiers. Using packet handlers in this manner allows the LKM to encapsulate the lower level protocols such as IPv4 and IPX, and also allows the protocol to be transparent to all higher-level protocols. Once the packet handlers are installed, any packets arriving with these identifiers are sent to the LKM code for handling. Currently, the slave code simply removes the protocol header, and sends the packet up to the higher levels for processing (Figure 2.2).

Once the packet handlers have been installed, the slave module will search for a master node by sending PACKET\_ANNOUNCE packets. PACKET\_ANNOUNCE packets are 60 bytes long and in the format shown in Figure 2.3.

Destination	Source	Protocol	Data
FF:FF:FF:FF:FF	Slave MAC Address	0xAA00	46 * 0x00

Figure 2.3: The PACKET\_ANNOUNCE packet format.

The PACKET\_ANNOUNCE packet is simply a broadcast packet containing the slave node's MAC address. The master only requires the slave's MAC address initially, so there is no reason to add any additional information. When it receives a PACKET\_ANNOUNCE, the master will add a new slave's MAC address to its list of slaves (Figure 2.8). The master does not send any acknowledgement to the slave indicating its announcement has been successful. A slave only realises it has become part of the token ring when it receives its first token. The PACKET\_TOKEN packet format is shown in Figure 2.4.

Destination	Source	Protocol	Data		
Slave MAC Address	Master MAC Address	0xAA02	ID	Transfer Amount	Number of Nodes

Figure 2.4: The PACKET\_TOKEN packet format.

The token contains the receiver's MAC address (destination), the master's MAC address (source), the PACKET\_TOKEN identifier (0xAA02), a group identifier, a data transfer amount, and the number of nodes currently in the group. The group identifier allows a slave to distinguish between two different masters should a situation involving conflicting masters arise. The token identifier also allows the master to ensure it is only sending one token at a time.

The transfer amount value holds the number of packets a slave is allowed to send in this period, which can be variable, depending on whether the master is implementing any quality of service algorithms. The slave can transfer until this value is reached, or it runs out of data to send. The number of nodes value is the current number of nodes in the ring. This enables the slave to calculate a rough estimate for when it will receive the token again. This allows the wireless card to utilise its energy saving mode until it is due to receive the token again. It can also be used to determine if the master node has crashed. If the slave node does not receive the token again within the calculated time, it starts the process again in order to associate with a new master.

Once a slave has received a token, it checks the ID provided against the last ID it received. If the ID has changed, the node knows the previous token was lost somewhere, and the network could be temporarily unstable. After checking the ID, the slave checks its data queue to see if there are any packets that require delivery. If there are packets to deliver, the slave sends a `PACKET_ACK` to the master, and then starts transferring the packets (Figure 2.2). The format of a `PACKET_ACK` is shown in Figure 2.5.

Destination	Source	Protocol	Data
Slave MAC Address	Master MAC Address	0xAA01	ID

Figure 2.5: The `PACKET_ACK` packet format.

The `PACKET_ACK` packet signals to the master node that the slave has data to transfer, and the master should alter its timeout values to wait for it to send all its data and return the token. If the slave has no data in its queue, has finished transferring all its data, or has reached the maximum number of data packets to send, it will return the token to the master node in the format described in Figure 2.6.

Destination	Source	Protocol	Data		
Master MAC Address	Slave MAC Address	0xAA02	ID	Packets Remaining	NULL

Figure 2.6: The `PACKET_TOKEN` packet format when being returned to the master.

The format is the same as any other token, except that the slave stores how many packets are remaining in its queue in the Transfer Amount field, and sends null data for the rest of the packet. The master node can use the Packets Remaining value to determine those nodes that have heavier traffic requirements and thus require the token more often than others.

When a data packet is sent through the slave's new interface, it is caught by the LKM code and processed. Under the Linux 2.4 series kernel, network packets are stored in an `sk_buff` structure that allows editing of headers and data, as well as modification of protocol identifiers. Any packet delivered to the LKM is encapsulated in a `PACKET_DATA` packet and stored in a queue of packets to wait until the LKM receives the token (Figure 2.2). The `PACKET_DATA` format is as shown in Figure 2.7.

Destination	Source	Protocol	Data
Destination MAC Address	Slave MAC Address	0xAA03	ORIGINAL PACKET

Figure 2.7: The `PACKET_DATA` packet format.

The LKM simply adds a new header so that the packet is handled by the destination's LKM, rather than by the kernel itself.

## 2.5 Master Node Operation

The master node operation (Figure 2.8) is similar to a slave node with some notable exceptions. The master node LKM creates a new interface like the slave LKM, but it registers more packet handlers, a list of which follows:

`PACKET_ANNOUNCE: 0xAA00`

`PACKET_ACK: 0xAA01`

`PACKET_TOKEN: 0xAA02`

`PACKET_DATA: 0xAA03`

The master node registers `PACKET_ACK` and `PACKET_ANNOUNCE` in addition to `PACKET_TOKEN` and `PACKET_DATA` registered with a slave.

Once a master has initialised its new network interface and packet handlers, it will initialise the ring ID and create an empty list of nodes to send the token to. It will then add itself to the list, so that it may transfer through the interface immediately. When a master receives a `PACKET_ANNOUNCE` packet, it will extract the source MAC address from the data packet and add the node to the nodes list (Figure 2.8). It also initialises some additional data fields such as node status and whether the node has packets waiting to be sent. Since the `PACKET_ANNOUNCE` packet does not hold this information, the master assumes that it has data to send.

When a master sends the token to a slave node, it first initialises timers so that if the token is lost, the master can recover. The master will initialise both a token timer and an acknowledgement timer. Since the expected response time for the acknowledgement is smaller than the expected time to receive the token back,

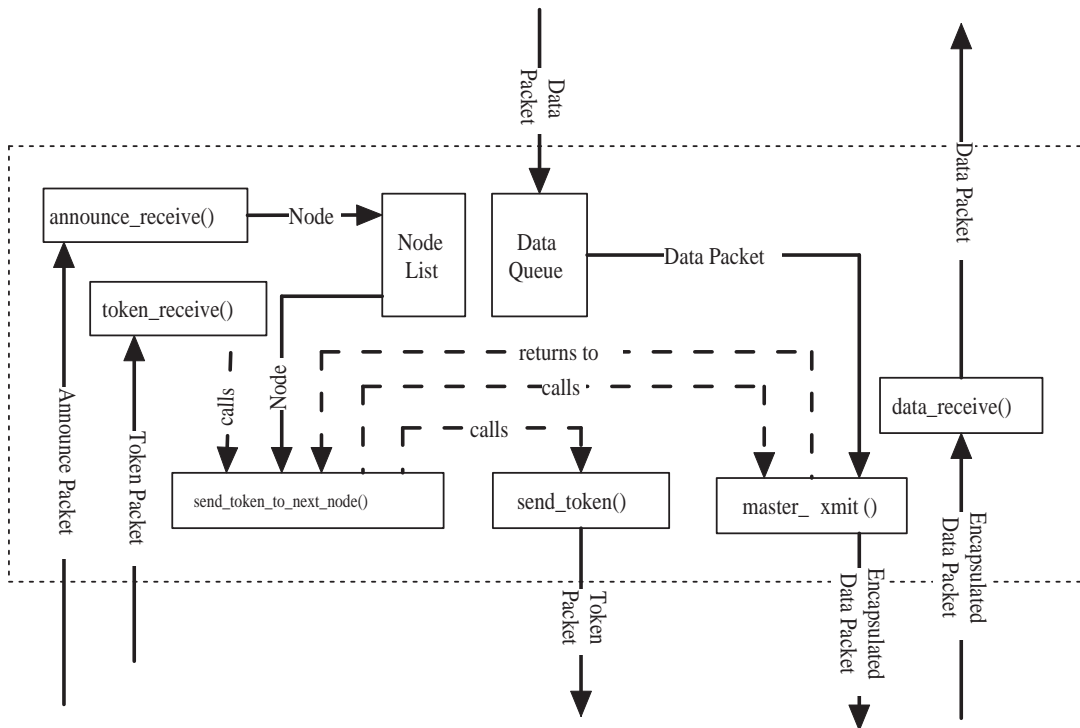


Figure 2.8: Data and control flow of the master node.

the acknowledgement timer is smaller than the token timer. Once the timers are initialised, the master will send the token to the slave node with the current ring ID. If the slave node replies with either a `PACKET_ACK` or a `PACKET_TOKEN`, the acknowledgement timer is disabled. The token timer is only disabled if a `PACKET_TOKEN` is received.

If either timer expires, the master will recover the token. If the acknowledgement timer expires, this informs the master that the slave node to which it was addressed has died, or has been unable to send acknowledgements to the master. To recover, the master will mark the slave node as dead in its node list and increment the ring ID. The master will then ignore this node as it moves through the node list. If the token timer expires, this informs the master that the slave node has either died, or been somehow delayed in its processing of data packets. In this case the master will mark the node as potentially dead in its node list and increment the ring ID. The master will continue to send this potentially dead node tokens since another timeout will mark the node as either dead if it fails to return the acknowledgement, or alive if it returns the token. If the node returns an acknowledgement packet, and then does not return the token for a second

time, causing another token timeout, the node is marked as dead.

The ring ID is incremented when a timeout occurs so that the master will not become confused if the slave node resumes normal operations. If the master had sent the token to a slave node, and the slave node became delayed in its processing of the packet, timeouts would occur on the master node. The master would then send a new token to the next node in the list. If the initial slave node then processed its token packet, it would send its data and then attempt to return the token to the master node. Incrementing the ring ID enables the master to determine that the slave token was simply delayed instead of dead, and enables the master to alter the node list accordingly. The master is also able to determine that the token returned was old, and thus should not be forwarded again to other nodes.

When a master node receives a returned token from a slave node, it will check the source MAC address to ensure it came from one of its known nodes. If so, and the ring ID contained in the packet matches the current ring ID, the master will disable both the acknowledgement and the token timers. After extracting the number of packets queued on the slave node, the master will store the value in the node information. The master will then examine its node list to determine where to send the token next, initialise timers, and send the token again.

## CHAPTER 3

# Method

Initially, WiCTP was to be implemented in a simulated environment such as the *ns-2* network simulator [4], or the cNet simulator [14]. Implementing the protocol under either of these simulators would have enabled a large number of tests in degrees of variability that cannot be generated in a real world situation. However, finding an appropriate simulator ultimately proved unsuccessful. The two most likely simulators, *ns-2* and cNet, were both inappropriate for such a protocol. *ns-2* did not support the level of layer abstraction required, whereas cNet did not include robust wireless simulation code. While either of the simulators could have been modified to support such additions, coding them into the programs would have been overly onerous. The lack of a suitably robust simulator, and a fortuitous agreement with the owners of Leeming Wireless Network (LWN) allowed real-world tests to be conducted. This has the added benefit of examining the protocol's effectiveness in a real-world situation, where weather, interference, and other factors interfere with the performance of 802.11b equipment.

In order to test the new protocol's effectiveness against the standard CSMA/CA access control method, it was loaded onto several computers that were part of LWN. LWN is a community wireless network situated in Leeming, with client nodes connecting from surrounding suburbs. LWN consists of approximately six wireless nodes connected to a central access point attached to a 14dB omnidirectional antenna joined to a 10-metre mast. The client nodes each connect using a 24dB directional antenna and client wireless card. The network uses the 802.11b WiFi standard to communicate, providing the network with a maximum theoretical throughput of 1300KBps and a realistic maximum throughput of approximately 500KBps. Each node card has a maximum transmission power of 15dBm (31.62mW). Each node on the network is anywhere between 100m to 2km from the access point. All nodes are hidden from each other, making the network ideal for testing the new protocol.

Agreements with the owners of each node only allowed for the use of up to 5 nodes in each experiment. While this may seem a low number of nodes to test the viability of the new protocol, prior experience has indicated that a wireless

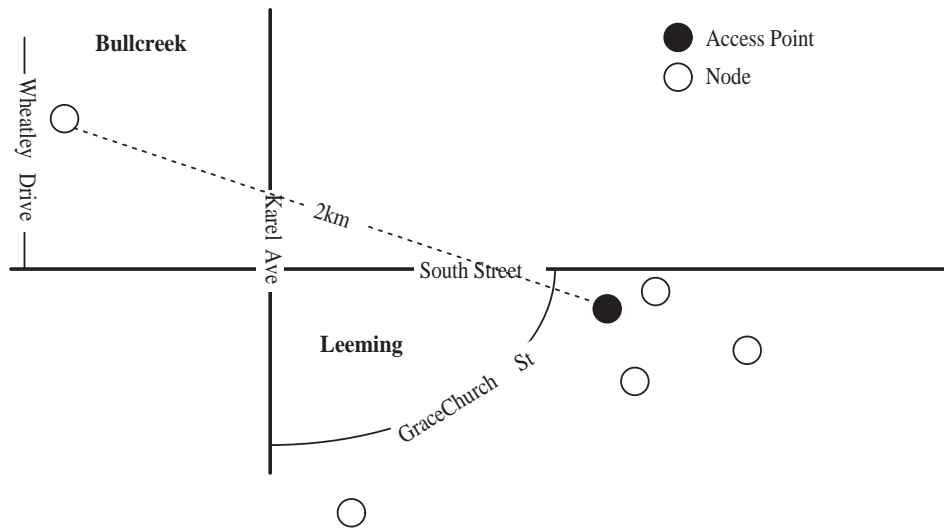


Figure 3.1: A rough layout of the Leeming Wireless Network.

network running CSMA/CA will not scale above four nodes if all nodes on the network are hidden from one another. The number of collisions and retries far exceeds the tolerance CSMA/CA allows. A five node network should demonstrate how effective the new protocol is compared to standard CSMA/CA, since the new protocol will scale to at least this number of nodes, and provide increased performance over CSMA/CA for any number of nodes.

Performance of the new protocol was measured by the transfer rate and round-trip times the protocol can sustain whilst a variable number of nodes transfer simultaneously. The experiments initialised the network with one node and gradually began transfers between nodes using a TCP/IP connection. While the nodes were transferring, transfer rates and ping times were recorded until the experiment stopped. In this way the load on the network was gradually increased to determine how well each protocol scaled as an increasing number of nodes started to transfer between one another.

In order for each node to maximise transfer rates, the network benchmark tool netperf [3] was run on each node. Netperf is a network benchmark suite that provides tests for both unidirectional throughput and end-to-end latency. It is able to test a variety of protocols such as TCP, UDP, and Unix Domain Sockets. For the purposes of testing WiCTP, only TCP and UDP support was compiled into the packages distributed to each node. As TCP and UDP are the primary network protocols used today, and are also the protocols end-users will be most likely to use on their wireless network, they were the two protocols chosen for

this study.

The netperf package contains two programs, netperf and netserver. netperf has several command line options including the test duration, the target host, and the test type. In the package distributed to the client nodes, netperf had four tests available to it;

- TCP\_STREAM
- UDP\_STREAM
- TCP\_RR
- UDP\_RR

The netserver program runs on the host that netperf wishes to test against. Its only function is to listen for a netperf client connection in order to start a test. For a netperf test to succeed, it requires a netserver to connect to, thus, each client node ran a netserver for the duration of any test.

Since WiCTP was implemented as a LKM, each client node needed to run Linux in order to carry out the tests. Unfortunately, some nodes on LWN were running Microsoft Windows, and thus a solution was required whereby the operating system could be changed quickly and easily, and only for the duration of the tests. Knoppix, a Linux distribution [1], provided a simple solution to this problem. Knoppix boots and runs off a single CD. It does not need write-access to any hard-drives currently in the system, and for this reason is perfectly suited to the required purpose. By customising a Knoppix distribution, the software available on the CD could be altered to include the WiCTP kernel module, netperf, and any other software package required for the tests.

A modified Knoppix CD was created which removed several large software packages that were included with the standard CD, and included the WiCTP LKM, netperf, and several other set-up scripts and tools needed to synchronise nodes. This CD was burnt and distributed to each of the client nodes running windows so that they would all be able to run Linux for the period of the tests.

Nodes already running a Linux 2.4 distribution did not require the CD. While these machines would not be running the same kernel version as the Knoppix CD machines, the WiCTP and netperf versions were standardised, which allowed accurate tests to be recorded. While the kernel versions were not standardised, the variability between machines would be present in any real-world set-up and would give more precise results, similar to those recorded by an end-user running the protocol on their network.

## 3.1 Initial Tests

The machines connected to LWN were required to start their tests at similar times in order for useful results to be gathered. In initial tests, a Network Time Protocol (NTP) server was used to synchronise the clocks on each of the machines, and a scheduled task inserted, using `cron` [15], to start the test at a specific time. Each node would then start the tests within moments of one another, outputting results once the tests were complete.

Unfortunately this method was not well suited to the task. Since the nodes were remote, the tests had to be set-up and clocks synchronised at several locations throughout Leeming surrounding suburbs. The Knoppix distribution CD also had no remote shell software such as `ssh`. Thus, once the tests were set-up at each location, it was only when the results were physically collated at the end of the experiment that the fate of the tests were known. Several times when this method was used, the tests were set-up and run, only to find the tests had failed to start at the allotted time. While some tests worked using this method, ultimately a more effective way of synchronising the nodes was required, as each set of experiments took at least two hours.

In order to synchronise individual nodes properly, a custom program, `netsync`, was written that allowed a central client to control the actions of each individual node. `Netsync` is able to connect to a number of nodes and read a set of scripted actions to send to each client, which will return the results of the action. Using `netsync`, the actions of the clients can be changed remotely, allowing several tests to be run in one sitting, and restarted if a problem is identified.

`Netsync` has two programs, a client and a server. The server is run on each of the wireless nodes, allowing the central client to connect and send actions to each client individually. On start-up the server loads a configuration file containing several set-up options including the location of the `netperf` binaries, the location of the `WiCTP` LKM, the wireless Ethernet interface to use, and paths to several scripts that gather various results.

In order to account for the different Linux distributions, and the small differences that occur in the standard tools that accompany them, the server called shell scripts to obtain results, rather than directly obtaining the results itself. Scripts to obtain the receiving transfer rate of a network interface, the transmitting transfer rate of a network interface, the average ping time to a host, and the packet statistics for a network interface were included with `netsync`. Using scripts allowed minor changes to be made, if the programs included with different Linux distributions provided slightly different output.

The `netsync` client allowed the actions of each of the wireless nodes to be

co-ordinated in an easily re-configurable fashion. When run, the client will load a file containing the actions for each node on the network. The client will send the action to a specific node and output the results to a file indexed by time. This allows netsync great flexibility as it can initiate netperf tests between nodes and gradually increase the network load.

The netsync client has several commands that it can send to any netsync server. All commands are blocking, meaning the client will stop execution until the server returns a result.

**LATENCY** tells the specified node to ping an IP address for a specific time, and return the average ping time to the central client. Since the network tests are likely to take up most, if not all, of the network bandwidth, this operation always takes seven seconds. This limit is imposed because ICMP is not a guaranteed protocol. Return times are variable, or the packets could never return due to network load, so if the limit were not there, the time taken for the command would be unknown, and potentially never return control to the program. The server will block for seven seconds while it waits for the script to return the result of the pings, but should the pings be lost or the script take longer than seven seconds to execute, the server will return, allowing the client to continue script execution.

**GET\_RXTX** obtains the amount of data sent and received by the server on a given network interface when the command is received, and again after a specific time interval. The server will return four values, the amount of data, in bytes, sent when the command was received, the amount of data, in bytes, received when the command was received, the amount of data sent, in bytes, X seconds after the command was received, and the amount of data received, in bytes, X seconds after the command was received, where X is a number of seconds, specified by the client. These values allow the client to calculate the average transfer rate of the node at a specific time. This command is blocking, and will always take X seconds to execute.

**GET\_RXTX\_NW** obtains the number of bytes sent and received from a given node without waiting. The server will return the values as soon as it receives the command. This allows the client to obtain the values from multiple nodes at the same time. Thus, a client can obtain the average transfer rates of several nodes by calling **GET\_RXTX\_NW** for the nodes, then waiting a set number of seconds before obtaining the values again. This records better results, since the transfer values are taken almost simultaneously, rather than separated by several seconds.

`GET_PACKS` obtains the packet output of the network interface configuration tool `ifconfig`. It will return the number of packets sent, packets received, error packets, dropped packets, and overrun packets.

`TEST` will make the server start a `netperf` client to connect to the specified IP and run a `TCP_STREAM` test for the specified number of seconds.

`LOAD_NODE` will make the server load the WiCTP slave node LKM.

`LOAD_MASTER` will make the server load the WiCTP master node LKM.

`WAIT` is the only command that is not sent to the servers. It causes the client to stop executing the script for a set number of seconds before resuming script execution.

With a combination of these commands, a script can be created that will load the WiCTP module, start several `netperf` tests to other nodes and obtain the transfer rates and ping times between nodes. The tests can be started one after another, gathering transfer rates and ping times as more nodes start to transfer. Thus, it gives greater flexibility over the original method of machine synchronisation, which would have provided only a single average. One potential pitfall of this method is that, should the network load become too high, the central client's commands may not reach the recipient, causing it to block indefinitely, but since the commands are both compact, and `netsync` is using TCP to communicate, this problem should be overcome. A sample `netsync` script is available in Appendix B.

The need for `netsync` also identified another problem with the original method of client node set-up. Initially, all the software was loaded onto the Knoppix CD, burnt, and then distributed to each node. However, since both `netsync` and WiCTP were under constant development, the Knoppix CDs needed constant software updates. Instead of constantly re-burning and wasting CDs, a set-up script was created which would configure the machine to connect to LWN and download the latest distribution of `netsync`, `netperf`, and WiCTP. This distribution archive included another script that automated the installation of these three packages, allowing tests to be set-up more rapidly than previous arrangements.

## CHAPTER 4

# Results

### 4.1 Experimental Overview

The receiving (RX) and transferring (TX) transfer rates of each node were recorded. The results indicate that two nodes are transferring at 700KB/sec each, a figure far above the theoretical maximum of the network (Figures 4.2 and 4.1). However, as an example, node one may be sending data to node two at 350KB/sec, and node two may be sending data to node one at 350KB/sec. Thus, when the sum of both RX and TX rates are calculated, both nodes have a total transfer rate of 700KB/sec.

The nodes used for the tests displayed in Figures 4.1 and 4.2 were not all hidden from one another. Unfortunately due to unforeseen problems, some members of LWN became unable to participate in the experiments, thus, the set-up was as follows:

**Node One:** The central access point

**Node Two:** A laptop

**Node Three:** A laptop

**Node Four:** A node from LWN, approximately 2km from the Access Point

This set-up used netsync to gradually increase the network load and return statistics summarising the nodes' performance. In this study Node One started to transfer to Node Two. At 100 seconds, Node Two began transferring back to Node One. At approximately 200 seconds, Node Three would start transferring to Node Four, and then at 400 seconds, Node Four would transfer back to Node Three. Results were gathered as the network load increased.

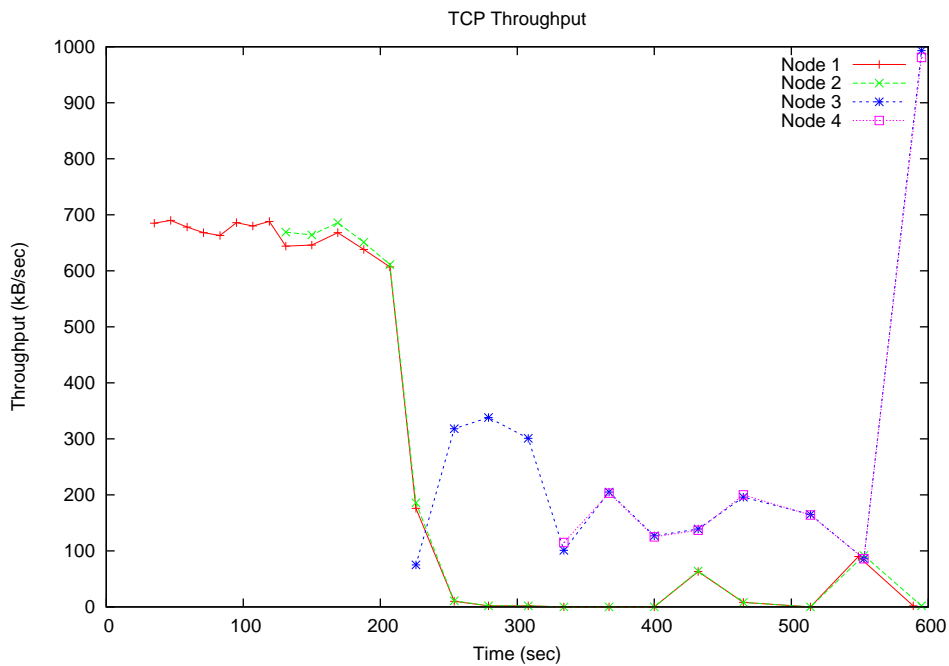


Figure 4.1: Combined(RX + TX) TCP Throughput using CSMA/CA.

## 4.2 TCP Throughput

Nodes two and four were not hidden from one another, but this has not effected the results. Figure 4.1 demonstrates how CSMA/CA performs in a four-node network. Nodes one and two sustain a high level of throughput, until nodes three(220secs) and four(330secs) begin to transfer, significantly decreasing the throughput of each node. This demonstrates how hidden nodes decrease the network's effectiveness.

Figure 4.2 shows how WiCTP performed under the same parameters as Figure 4.1. Clearly, WiCTP provides consistency in network performance. Nodes one and two sustain maximum transfer rates of up to 600KB/sec until nodes three and four begin transferring. At that point, the transfer rate of nodes one and two drop, allowing the new nodes to begin transferring, and sharing the bandwidth between all nodes.

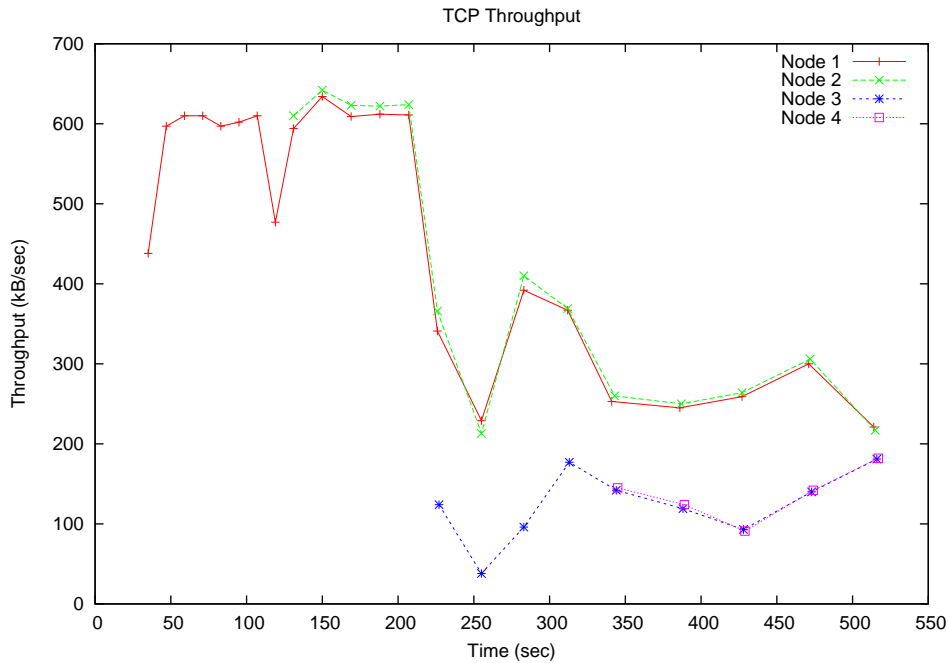


Figure 4.2: Combined(RX + TX) TCP Throughput using WiCTP.

### 4.3 Ping Times

ICMP Ping reply time gradually increased for each node as the network load increased (Figures 4.3 and 4.4). Figure 4.3 shows the ping times for the four nodes under WiCTP. The ping time gradually increases as more data is transferred. Figure 4.4 shows the ping times under CSMA/CA, which also increase as more data is transferred. Clearly, WiCTP has greater ping times than CSMA/CA, with times in excess of 2000ms under the greatest load. Even the lowest WiCTP ping time, approximately 100ms, is very high, however the CSMA/CA test that it has similar ping times. Under a higher load the CSMA/CA protocol clearly has a lower ping time, with a maximum of 500ms, compared to WiCTP's 2000ms.

### 4.4 Error Rates

The error rates for each node running WiCTP and CSMA/CA are shown in Figures 4.5 and 4.6 respectively. An error was either a corrupt packet or a collision, but due to the nature of how the data was gathered, determining which was impossible. Figure 4.5 displays the error rates for the experiment when

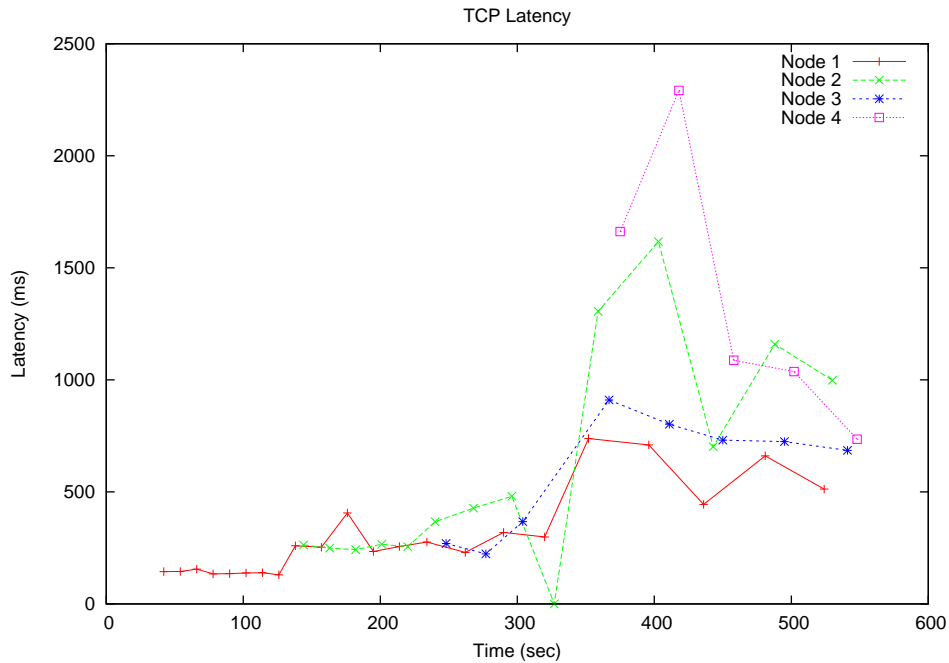


Figure 4.3: Ping time for WiCTP.

running WiCTP while Figure 4.6 shows the error rate when the nodes are using CSMA/CA. Each of the scenarios shows an increase in errors as more nodes start to transfer simultaneously, however it is clear that under WiCTP, the nodes have significantly fewer transfer errors than under CSMA/CA. As more nodes begin to transfer the error rate increases dramatically from negligible amounts of close to 0 errors/sec to 12 errors/sec CSMA/CA. Under WiCTP the error rate is kept at negligible levels, and has a maximum of 0.35 errors/sec even under the highest load.

## 4.5 Discussion

WiCTP is able to add significant stability to an existing network as shown by the recorded results. As predicted, WiCTP decreases the error rate of each network node significantly (Figures 4.5 and 4.6). Decreasing the error rate from approximately 12 errors/sec to just 0.35 errors/sec created a more stable network. While the software installed on each node could not determine the exact reason for the errors, it is possible that the errors under WiCTP were due to external factors such as transmission losses and corrupted packets. This

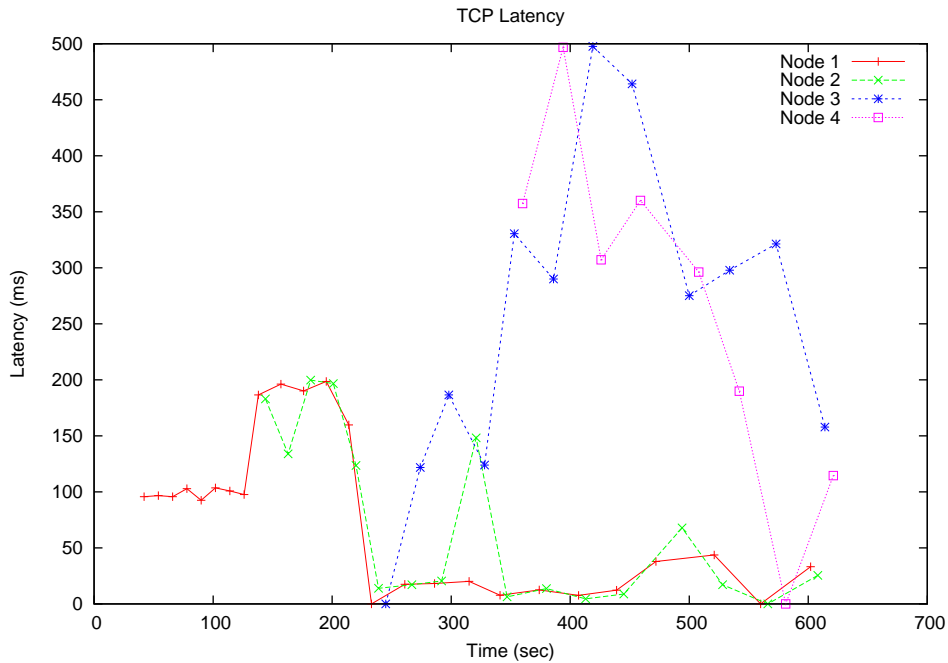


Figure 4.4: Ping time for CSMA/CA.

suggests the protocol would have a greater effect if the network were set-up in a more stable environment whereby antennas were aimed correctly and no barrier objects were impeding the signal. Even so, the error rates under WiCTP were still lower than those under CSMA/CA, regardless of whether external factors are considered. While transmission losses would also be present under CSMA/CA, their contribution to the overall error rate is negligible, as most errors would be from collisions. This can be reasoned since it is very unlikely that the number of packets lost due to reasons other than collisions would suddenly rise to such a high level when the protocol is changed. Indeed, WiCTP runs on top of CSMA/CA, so it is reasonable to assume that if packets were corrupted simply because of the use of CSMA/CA, WiCTP would also suffer a much higher error rate than is seen.

WiCTP network throughput, while not higher than CSMA/CA, is more efficiently shared between the nodes. While CSMA/CA had a greater maximum throughput rate of just under 700KB/sec, it quickly degraded as more nodes began to transfer. While it could sustain this high transfer rate when two nodes were transferring, the introduction of a third node caused the network quality to drop significantly (Figure 4.1). After the third node began transferring, the throughput of nodes one and two decreased to next to zero, whilst nodes three

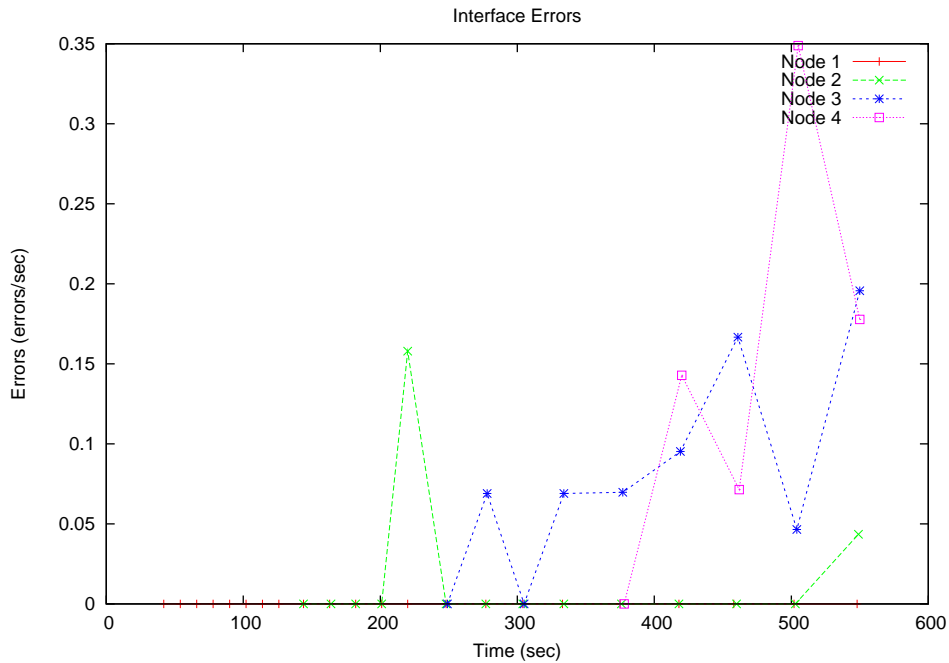


Figure 4.5: Error Rates for WiCTP.

and four transferred at only a fraction of the maximum bandwidth. Clearly, the sum of all throughput on the network does not reach the maximum throughput possible. In comparison, under WiCTP the network performance is consistently close to 600KB/sec at all times. Unlike CSMA/CA, when nodes three and four begin to transfer, the network bandwidth was efficiently shared between all four nodes, whilst nodes one and two continued to transfer at a high amount. While it seems that nodes three and four were not receiving an equal share of the bandwidth, since they had significantly lower transfer rates than nodes one and two, this was mostly due to node four's low SNR and link quality. If node four had a greater link quality the bandwidth allocation for nodes three and four would have been comparable to that of the other nodes.

The ICMP ping times for CSMA/CA are clearly more desirable than those seen under WiCTP. While it was expected that the ping times under WiCTP would be higher, the large difference between the two different access mechanisms was not. The ping times for CSMA/CA were expected to be as high as 2000ms, or even timing out, due to the high network load, whilst WiCTP was expected to give ping times of approximately 500ms due to the inherent delay associated with a token-based access mechanism. There are several possible reasons the ping times for WiCTP are so high, however none are conclusive without further

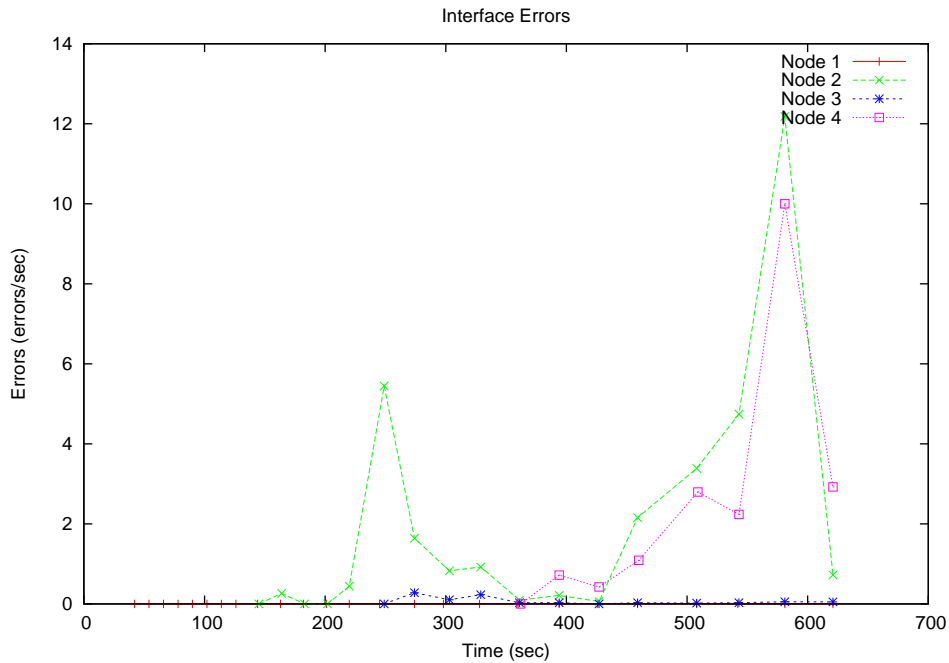


Figure 4.6: Error Rates for CSMA/CA.

testing. One possibility is that, due to CSMA/CA's low transfer rates, there is more available bandwidth and thus ping packets are more readily able to pass between nodes. Examining Figure 4.1 and 4.4 together supports this, since the nodes with the lowest transfer rates (nodes one and two) also have the lowest ping times, whilst those nodes with higher transfer rates (nodes three and four) have higher ping times. In contrast, WiCTP always keeps used bandwidth close to the maximum and ping packets are placed at the end of the data queue. This means that a ping time will essentially be the time it takes for a packet to reach the front of the queue, be sent, and then return to the sender. With netperf continually trying to send data as fast as possible, it is unlikely that a ping packet would ever manage to enter the queue in front of the other data packets. This, combined with the inherent wait for the token could all contribute to the high ping times seen under WiCTP.

## CHAPTER 5

# Conclusion

The results recorded from this study comparing WiCTP and CSMA/CA are promising. The results show that, as expected, a wireless network utilising WiCTP obtains a greater level of stability than that of a network using standard CSMA/CA. Error rates were significantly decreased under WiCTP compared to CSMA/CA, and while the CSMA/CA error rates were not as high as initially predicted, they nonetheless made a noticeable difference in network performance. Throughput was also more stable throughout the tests under WiCTP. Rather than giving one or two nodes a significant portion of the bandwidth, while other nodes suffered, WiCTP managed to successfully share the bandwidth between nodes, creating a much more stable network environment and an overall higher network utilisation at all times. While ICMP ping results for WiCTP were not as expected, the protocol still performed well under increasingly difficult scenarios and the low pings under CSMA/CA could be due to lower bandwidth utilisation, as detailed above.

Comparing WiCTP against RTS/CTS and other protocols that attempt to improve wireless network stability is difficult without running each protocol on the same network and under similar conditions. While RTS/CTS does provide a measurable increase in network stability and performance when there are hidden nodes present, there are still significant problems that prevent the protocol from reaching above a certain threshold. Each of the protocols that attempt to improve network performance have their own strengths and weaknesses, and are suited to different situations. RTS/CTS is widespread and will work well under conditions where there are few hidden nodes whereas token based solutions such as WiCTP and Frottle will work best in situations where all nodes are hidden.

While WiCTP is certainly not an end-all solution to the hidden node problem, it certainly alleviates the problem and returns much needed stability to a wireless network that suffers from hidden nodes. Freenets, and other long-distance networks, can easily implement the protocol over existing CSMA/CA without the need for risky firmware upgrades or expensive hardware replacement. Newer standards and protocols in development by the IEEE and other third-parties may

eliminate the hidden node problem completely, but for existing hardware, particularly 802.11b-based equipment, WiCTP is a cheap and effective mechanism for alleviating the problems associated with hidden nodes.

## APPENDIX A

# Original Honours Proposal

## Background

Since their introduction several years ago, wireless networks have increased in both popularity and power. The 802.11 [9] range of wireless standards are the most widely used standards for creating wireless networks in both the home and office. Each of these 4 standards, 802.11, 802.11a, 802.11b and 802.11g, vary in maximum range and speed, from 2Mbits/sec (802.11) to 54Mbits/sec (802.11a). External antennas and power amplifiers, significantly increase the range of the networks, leading to the rise of metropolitan networks and *Freenets* [10]. However, the creation of *Freenets* using these standards has identified problems with the use of the Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA) access control method. The 802.11 range of standards were created with the assumption that most, if not all, nodes on the wireless network would be within range of all other nodes on the network, as they are in most standard wireless arrangements. To avoid collisions, a node listens on the medium, and if any nodes are transferring packets, the node waits for a random interval, then tries again. This method is suitable only when all nodes are within listening range of one another, not when network range has been significantly increased, as is the case in *Freenets*. In this scenario the nodes cannot sense other transfers, so the node falsely believes it can transfer. This, in turn, leads to a dramatic increase in collisions when several nodes transfer simultaneously, decreasing network quality substantially. This problem is termed the hidden node problem. The Request to Send/Clear to Send (RTS/CTS) protocol is the standard method for solving the hidden node problem. This method has several problems, with RTS requests colliding with other RTS packets. The number of packets on the medium increases, as does the number of collisions. A solution to the hidden node problem will increase the performance of metropolitan networks, and potentially increase the performance of standard wireless networks.

Several groups have investigated this problem. Frottle [12] and the Wireless

Central Coordinated Protocol (WiCCP) [5] are two packages that combat the hidden node problem by passing a token to each node. When a node has the token it can transfer data, removing collisions and increasing network quality. Both groups have improved network quality and speed. While much of the data showing improvement is anecdotal, evidence suggests that performance increases are achieved with a token-based solution.

## Aim

S. Khurana et al [11] has shown that the performance of a wireless network drops dramatically as the number of hidden nodes increases. RTS/CTS can improve the performance somewhat, but it increases the number of packets on the medium and is not available in most consumer grade hardware.

The purpose of this project is to determine whether performance gains can be found when a token-based access control method is implemented on top of the current CSMA/CA Media Access Controller (MAC). Implementing this access control method should improve network quality significantly since it will only allow one node to transfer at a time. This additional access control layer will eliminate most, if not all, collisions in a static wireless environment, and give a better quality of service (QoS) to nodes that, currently, have difficulty in sustaining a reliable link.

This project aims to create a token passing protocol for wireless networks that has a performance level comparable with Frottle, but also has the functionality of the WiCCP package. The WiCCP package has a greater level of functionality since it is implemented at kernel level. This enables it to run regardless of what protocols are allowed on the network unlike Frottle, which requires a TCP/IP stack in order to send and receive control packets. This project also aims to create a more robust protocol that can handle missing tokens, which is a current issue with WiCCP. An adaptive algorithm could also be implemented in the protocol to give more transfer time to those nodes that have a higher bandwidth requirement. To cut down energy usage, the protocol will also place the wireless nodes into a low power mode when the nodes do not have possession of the token. Clearly, this will only be of use to networks where all nodes only have traffic destined for outside of their network, but it will provide a drop in power consumption for these scenarios. The resulting protocol should provide wireless networks with a high level of QoS and eliminate the hidden node problem completely.

## Method

The protocol will be implemented in a Linux Kernel Module and loaded onto a network of hidden wireless nodes. Tests will then be performed to benchmark the network using common metrics such as network latency and maximum throughput. The same tests will be performed on the same wireless nodes using the default CSMA/CA access mechanism and the results will be compared. This will give a comprehensive comparison between the two access mechanisms running in a real world scenario.

The projected timetable is as follows:

**Week 3-6:** Design access control protocol.

**Week 7-10:** Implement access control protocol.

**Week 11-14:** Run network tests.

## Software and Hardware Requirements

The requirements for this project are quite large. Up to six computers, each with 802.11b cards will be used to test the protocol in a real scenario where all nodes are hidden from each other.

## Progress

Results were initially going to be gathered by running the protocol on a network simulator, but it has since become more suitable to implement the protocol in a Linux Kernel Module and gather results from real world tests. The kernel module is mostly completed, with only a few features left to implement. An algorithm for giving more transfer time to nodes that require it still needs to be implemented, as does placing the wireless card into a low power mode when it relinquishes the token. However, early indicators suggest that giving more transfer time to nodes may not improve performance very much, especially on TCP dominant networks. It is thought that the TCP window size does not grow large enough to warrant giving more time to an individual node, but this has yet to be investigated fully.

Preliminary tests on a wireless network consisting of 2 hidden nodes show a significant increase in performance over standard CSMA. When nodes **U** and **D** open simultaneous TCP streams to each other, the average transfer rate grows to

190Kb/sec under the new protocol compared to only 49Kb/sec under standard CSMA. The network latency also grows under the new protocol, from an average of 6ms under CSMA to an average of 9ms. It has yet to be seen how well the protocol scales, but these initial results look promising.

## APPENDIX B

# Example `netsync` script

A simple `netsync` example. While it does not display all the functionality of `netsync`, the general idea of the program is demonstrated.

```
NNODES 2

CONNECT 0 10.6.10.10
CONNECT 1 10.6.10.98

#start first stream, sample every 10 seconds over 5 second interval
CMD 0 TEST TCP_STREAM 10.6.10.98 200
SAMPLE 0 0 0 5
SAMPLE 0 1 0 5
LATENCY 0 10.6.10.98 5

SAMPLE 0 0 0 5
SAMPLE 0 1 0 5
LATENCY 0 10.6.10.98 5

SAMPLE 0 0 0 5
SAMPLE 0 1 0 5
LATENCY 0 10.6.10.98 5

SAMPLE 0 0 0 5
SAMPLE 0 1 0 5
LATENCY 0 10.6.10.98 5

SAMPLE 0 0 0 5
SAMPLE 0 1 0 5
LATENCY 0 10.6.10.98 5

SAMPLE 0 0 0 5
```

```
SAMPLE 0 1 0 5
LATENCY 0 10.6.10.98 5

WAIT 10
#start second stream, sample etc
CMD 1 TEST TCP_STREAM 10.6.10.10 100
SAMPLE 0 0 0 5
SAMPLE 1 0 0 5
SAMPLE 0 1 0 5
SAMPLE 1 1 0 5
LATENCY 0 10.6.10.98 5
LATENCY 1 10.6.10.10 5

SAMPLE 0 0 0 5
SAMPLE 1 0 0 5
SAMPLE 0 1 0 5
SAMPLE 1 1 0 5
LATENCY 0 10.6.10.98 5
LATENCY 1 10.6.10.10 5

SAMPLE 0 0 0 5
SAMPLE 1 0 0 5
SAMPLE 0 1 0 5
SAMPLE 1 1 0 5
LATENCY 0 10.6.10.98 5
LATENCY 1 10.6.10.10 5

WAIT 10

CMD 0 EXIT
CMD 1 EXIT
```

# Bibliography

- [1] Knoppix linux live cd. <http://www.knoppix.org/>, November 2004.
- [2] netfilter/iptables. <http://www.netfilter.org/>, November 2004.
- [3] Netperf. <http://www.netperf.org/>, November 2004.
- [4] The network simulator - ns-2. <http://www.isi.edu/nsnam/ns/>, November 2004.
- [5] Wireless central coordinated protocol. <http://patraswireless.net/software.html>, November 2004.
- [6] Vaduvur Bharghavan, Alan J. Demers, Scott Shenker, and Lixia Zhang. MACAW: A media access protocol for wireless LANs. In *SIGCOMM*, pages 212–225, 1994.
- [7] Chane L. Fullmer and J. J. Garcia-Luna-Aceves. Solutions to hidden terminal problems in wireless networks. In *Proceedings of the ACM SIGCOMM '97 conference on Applications, technologies, architectures, and protocols for computer communication*, pages 39–49. ACM Press, 1997.
- [8] IEEE. *IEEE 802.3: IEEE standards for local area networks: carrier sense multiple access with collision detection (CSMA/CD) access method and physical layer specifications*. IEEE, 1985.
- [9] IEEE. *IEEE 802.11 Wireless LAN medium access control (MAC) and physical layer (PHY) specifications*. IEEE, 1997.
- [10] Jason Jones. e3 :: blogging the wireless freenet. <http://www.e3.com.au>, November 2004.
- [11] S. Khurana, A. Kahol, and A. P. Jayasumana. Effect of hidden terminals on the performance of IEEE 802.11 MAC protocol. In *Local Computer Networks, 1998. LCN '98. Proceedings., 23rd Annual Conference on*, pages 12–20, 1998.
- [12] Chris King. Frottle: Packet scheduling and QoS for wireless networks. <http://frottle.sf.net/>, November 2004.

- [13] Thanasis Korakis, Gentian Jakllari, and Leandros Tassiulas. A mac protocol for full exploitation of directional antennas in ad-hoc wireless networks. In *Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing*, pages 98–107. ACM Press, 2003.
- [14] Chris McDonald. The cnet network simulator. <http://www.csse.uwa.edu.au/cnet/>, November 2004.
- [15] Paul Vixie. crontab(5). <http://www.rt.com/man/crontab.5.html>, 1994.
- [16] Kaixin Xu, Mario Gerla, and Sang Bae. How effective is the IEEE 802.11 RTS/CTS handshake in ad hoc networks? In *IEEE Global Communications Conference (GLOBECOM02)*, volume 1, pages 72–76, Nov 2002.