2004

APARNA : a Protocol Architecture for Resource-Constrained Networks

Kiran Komaravolu
Lehigh University

Komaravolu, Kiran

Aparna - A Protocol Architecture for Resource-Constrained Networks

January 2004

APARNA

A Protocol Architecture for Resource-Constrained Networks

by

Kiran Komaravolu

Presented to the Graduate and Research Committee
of Lehigh University
in Candidacy for the Degree of
Master of Science

in

Computer Science

Lehigh University

January 2004

This thesis is accepted and approved in partial fulfillment of the requirements for the Master of Science.

12/4/03
Date

Thesis Advisor

Chairperson of Department

Acknowledgments

I would like to take this opportunity to offer my heart felt thanks to a number of people without whose help and support this thesis would never have been possible. I must start with my advisor, Prof. Brian D. Davison. I would also like to express great gratitude to Prof. Liang Cheng and Prof. Héctor Muñoz-Avila.

I have had the pleasure of working with a number of very talented and knowledgeable people at Web Understanding, Modeling, and Evaluation lab (WUME). I would like to express special thanks to Baoning Wu and Kalyan Boggavarapu. They have been tremendously helpful. I also want to thank all my fellow students and researchers in the department.

My parents have provided me with countless opportunities for which I am eternally grateful. Nothing I can say can do justice to how I feel about their support. I must thank a good friend without whose encouragement I might never have finished. So thank you very much – Raghu.

Table of Contents

List of Figures

Abstract

The beginning is the most important part of the work.
- Plato

Most networking bottlenecks exist at the end client, as a result of the small bandwidths and loss rates of that link. Performance Enhancing Proxies (PEP) have been proposed to improve the end client performance. However existing TCP designs do not perform optimally for PEPs. The APARNA project proposes modifications at the application layer and the transport layer to improve TCP throughput for these resource-constrained networks. The suggestions include modifying the TCP congestion control algorithm, network layer modifications and header compression.

TCP/IP has been and is likely to continue for a long time as the de-facto communication protocol standard over the Internet. TCP has been designed for communication over wired networks. Essentially it assumes a reliable network link, with very little or no losses due to channel errors. But in case of communication over wireless networks, it is suboptimal. End link features such as mobility, disconnection, poor link behavior (losses) etc., are transparent above the transport layer. This means the application itself does not provide any special measures to handle end host behavior, and it should not be expected to do so. This means we still have to use just TCP/IP over the wireless networks too. This is probably a bit naive. Researchers have proposed methods to get around these problems of TCP/IP, such as the use of split connection proxies, which allow TCP connections between the end link and the server to be separated into two. This allows us to apply optimization features over the last link, some of which are discussed in this thesis.

APARNA proposes an alternative protocol stack, the lightweight protocol for weak connections (LPWC) to be used along the last link replacing TCP/IP for a client connected to the Internet through a split connection proxy. The LPWC covers the network and transport layers on the end link. In the case of a PEP, all the connections from a client end at the proxy. The length of the network is very small, about 2 to 3 hops. In such a scenario, we can improve performance over existing TCP designs.

Chapter 1.

Background and Related Work

Don't judge each day by the harvest you reap, but by the seeds you plant.
--Robert Louis Stevenson

Of late there have been many attempts to improve service to thinly connected clients. This has been boosted by the success story of wireless telephone service. There have been attempts to provide network services to hand-held wireless devices (PDA's, cellphones etc). These attempts have not seen as much success. One reason for this failure is lack of proper devices. Existing hand-held devices are not up to the mark to use Internet services. Another reason is the poor quality of service. Wireless networks will probably never be able to match wired networks in terms of bandwidth or round trip times. This raises the question as to what can be done to improve performance on wireless networks. One answer is to extract more resources out of the available channel. The idea is to use less channel bandwidth to send more information. Bandwidth in the wireless world is a precious commodity, much more than in wired networks. In wired networks bandwidth is not really an issue, if more bandwidth is needed, additional network connections could be provided. This is not possible in the wireless world. Increasing bandwidth means we need to increase the frequency spectrum, which is rarely possible. So the ideal means to improve the quality situation is to reduce bandwidth wastage, pack more information into fewer bits and try to keep the channel busy and not idle.

The ideas presented in this thesis are not just for wireless networks. They are intended to be used in all kinds of environments where bandwidth is scarce and precious. Somehow "scarce and precious" are two words associated with wireless networks. This probably explains why the

background work for this thesis is centered on solutions for wireless networks.

There have been quite a few ideas proposed and some implementations are also available, on improving performance in wireless and other networks. Some of those ideas are presented in this section. There are three broad categories where we can classify improvements to TCP over wireless links:

1. Local modifications; hide the wireless link from the TCP sender.
2. Make the sender aware that there exists a wireless link so it would realize some losses are not due to congestion
3. Modify the link-layer and make it "wireless-efficient"

We discuss these further below.

1.1 Masking Channel Losses from End Server

1.1.1 Split TCP connections

The split TCP model was first proposed by Bakre and Badrinath [I-TCP]. It has over the years been tweaked and twisted by other researchers notably [MTCP], [MOWGLI]. The goal of split TCP connections is to hide the channel losses on the wireless link from the end server. This way the loss in performance when the server reduces its congestion window on detecting a lost packet is avoided. In their original paper, Bakre and Badrinath suggested that the TCP connection be split into two (or more) TCP connections. In essence a proxy is introduced between the server and the end client. The proxy is implemented on the edge of the wireless domain. This way the server is shielded from the channel losses on the wireless link and also it does not experience the "elephant effect" of long fat networks (a large amount of time is wasted in pumping up the congestion window of the sender during the slow-start period). [Davison02] proposed a split

stack approach to client networking, wherein the client protocol stack is moved away from the client to an intermediate gateway. The application runs on the client but the networking calls are made from the gateway.

Split TCP connections found some success partly due to the fact that implementing a split connection model was rather simple (if not trivial). The problem with split connections is that they violate the end to end principle of the Internet [Saltzer84]. The end to end principle states that "functions placed at low levels of a system may be redundant or of little value when compared with the cost of providing them at that low level". A function can only be completely and correctly implemented with the knowledge and help of the applications standing at the communication endpoints. Split TCP connections violate the end to end principle by acknowledging data which the end client may not have received. Violation of this principle is a serious problem.

1.1.2 TCP Snoop

Hari Balakrishnan proposed the TCP Snoop [SNOOP] model as part of his PhD dissertation. In the TCP Snoop model, the gateway on the edge of the wireless network holds a cache of recently sent packets. Whenever the gateway detects a duplicate acknowledgment, it assumes the packet has been lost on the channel and suppresses the duplicate acknowledgment from the server. It sends out the lost packet from its own cache. This way the server never gets the lost packet and never knows that a packet has been lost. If the packet is not found in the gateway's cache, then the loss is a congestion loss and the dupack is passed onto the server. Snoop is popularly implemented these days. The drawback of snoop seems to be an apparent security flaw. If the gateway can read the transport layer headers of the packet, then so can any

other sniffing device. This flaw may not be acceptable in all cases, but it works well for the general case. Intuitively it can be guessed that the gateway could have a greater say in establishing and maintaining TCP connections if it has access to the complete packet headers, which is what I suggest in this thesis.

1.1.3 The Mowgli System

The Mowgli system [MOWGLI] is a proxy based approach to improve performance in wireless networks. The Mowgli system was designed to overcome some of the challenges posed by low bandwidth wireless wide-area links. It was primarily targeted at cellphones. Mowgli is based on the split connection philosophy. It uses split connection at all layers of the protocol stack. The basic architecture is based on split TCP connections. An application layer proxy pair may be added between a client and server, the agent (local proxy) on a mobile host and the proxy on an intermediate node that provides the mobile host with the connection to the outside world. The socket architecture on the client machine is slightly modified to support existing applications.

Mowgli also provides for an option to replace the last mile TCP/IP protocols with a proprietary protocol architecture. This protocol may implement features such as compression, unreliable link layer mode (which can be turned on or off depending on the circumstances). Unfortunately Mowgli does not comply with the end to end rule (as with most split TCP connection based ideas). This is the reason why it never really caught on. Nevertheless the Mowgli system remains one of the most important ideas in the wireless world.

1.1.4 Wireless Application Protocol (WAP)

WAP [WAPForum] was targeted at low power wireless devices such as pagers, cellphones, PDAs etc. It was specifically designed to cope with the low power, low memory, low CPU and low bandwidth constraints of these devices. The WAP model consists of a WAP client (mobile terminal), a WAP proxy, and an origin server. In a typical scenario, a WAP client sends an encoded WAP request to a WAP proxy. The WAP proxy translates the WAP request into a WWW (HTTP) request, performing the required protocol conversions, and submits this request to a standard web server on the Internet. After the web server responds to the WAP proxy, the response is encoded into a more compact binary format to decrease the size of the data over the air. This encoded response is forwarded to the WAP client.

Looking at just the network operation of WAP, it looks very similar to the design of Mowgli. And it suffers from the problems of Mowgli as well, i.e. it violates the end to end principle. Bringing an application level proxy (the WAP Proxy) into the picture violates the end to end rule. Although HTTP proxies are all not that bad, beyond HTTP, they are not of practical use. Thus the use of WAP is not much beyond HTTP. This is perhaps one reason why WAP has not really caught on despite being aggressively marketed by the cellular companies.

1.2 Sender-based Methods

There have been a few sender (the end TCP server) based methods proposed to improve the quality of service to the wireless links. These methods require the co-operation and modification of the end TCP server. These include TCP-Reno and TCP-NewReno, Selective Acknowledgments (SACK), Explicit loss notifications (ELNs).

1.2.1 SACK

Multiple packet losses from a window of data can have a catastrophic effect on TCP throughput. TCP [Postel81] uses a cumulative acknowledgment scheme in which received segments that are not at the left edge of the receive window are not acknowledged. This forces the sender to either wait a round trip time to find out about each lost packet, or to unnecessarily retransmit segments which have been correctly received [Fall95]. With the cumulative acknowledgment scheme, multiple dropped segments generally cause TCP to lose its ACK-based clock, reducing overall throughput.

Selective Acknowledgment [SACK] [RFC2018] is a strategy which corrects this behavior in the face of multiple dropped segments. With selective acknowledgments, the data receiver can inform the sender about all segments that have arrived successfully, so the sender need retransmit only the segments that have actually been lost. The drawback with SACK is that it needs support at both client and server ends. This makes SACK difficult to implement.

1.2.2 Explicit Loss Notifications

Explicit Loss Notification (ELN) [Balakrishnan97] is a mechanism by which the reason for the loss of a packet can be communicated to the TCP sender. In particular, it provides a way by which senders can be informed that a loss happened because of reasons unrelated to network congestion (e.g. due to wireless bit errors), so that sender retransmissions can be decoupled from congestion control. If the receiver or a base station knows for sure that the loss of a segment was not due to congestion, it sets the ELN bit in the TCP header and propagates it to the source. In the situation at hand, this ELN message is sent as part of the same connection (and not in a separate way, using ICMP for instance). This simplifies the sender implementation as it receives messages

in-band. ELN is a general concept that has applications in a wide variety of wireless topologies.

The snoop agent running at the base station monitors all TCP segments that arrive over the wireless link. However, it does not cache any TCP segments since it does not perform any retransmissions. Rather, it keeps track of holes in the sequence space as it receives data segments, where a hole is a missing interval in the sequence space. These holes correspond to segments that have been lost over the wireless link. However, it could also be the case that the packet was lost due to congestion at the base station. To avoid wrongly marking a congestion hole as having been due to a wireless loss, it only adds a hole to the list of holes when the number of packets queued on the base station's input interface is not close to the maximum queue length.

When ACKs, especially duplicate ACKs, arrive from the receiver, the agent at the base station consults its list of holes. It sets the ELN bit on the ACK if it corresponds to a segment in the list before forwarding it to the data sender. It also cleans up all holes with sequence numbers smaller than the current ACK, since they correspond to segments that have been successfully received by the receiver. When the sender receives an ACK with ELN information in it, it retransmits the next segment, but does not take any congestion control actions. The sender also makes sure that each segment is retransmitted at most once during the course of a single round-trip, as the snoop agent would flag an ELN for each duplicate ACK following a loss.

1.2.3 TCP Reno and TCP NewReno

TCP Reno added a fast recovery feature to traditional the TCP (Tahoe) version. With fast recovery, TCP detects a lost packet by detecting duplicate acknowledgments. When the third duplicate ACK is received, the Reno TCP transmitter sets slow start threshold (ssthresh) to one half of the current congestion window (cwnd) and retransmits the missing segment (fast

retransmission). The cwnd is then set to ssthresh plus three segments (one segment per each duplicate ACK that has already been received). The congestion window is then increased by one segment on reception of each duplicate ACK which continues to arrive after fast-retransmission. This allows the transmitter to send new data when cwnd is increased beyond the value of the cwnd before the fast-retransmission. When an ACK arrives which acknowledges all outstanding data sent before the duplicate ACKs were received, the cwnd is set to ssthresh so that the transmitter slows down the transmission rate and enters the linear increase phase. This way the recovery is much faster than with Tahoe, where slow start is invoked after the lost packet is retransmitted.

NewReno brings about a few small changes to traditional Reno algorithm. An optimal ssthresh is calculated when the connection is about to be made. The optimal ssthresh is calculated by determining the available bandwidth on the channel using the packet pair algorithm. If two or more segments have been lost from the transmitted data (window), the fast retransmission and fast recovery algorithms will not be able to recover the losses without waiting for retransmission time out. NewReno overcomes this problem by introducing the concept of a fast retransmission phase, which starts on detection of a packet loss (receiving 3 duplicate ACKs) and ends when the receiver acknowledges reception of all data transmitted at the start of the fast retransmission phase. If more than one packet is missing within the same window, a retransmission only recovers the first lost packet from the window. The receiver then ACKs reception of the retransmitted segment and all following segments up to the next lost segment. This ACK is called a "partial ACK", because it has not acknowledged all the packets which were transmitted prior to the start of the current fast retransmission phase. The transmitter assumes reception of a partial ACK during the fast retransmission phase as an indication that another packet has been lost

within the window and retransmits that packet immediately to prevent expiry of the retransmission timer. NewReno sets the cwnd to one segment on reception of 3 duplicate ACKs (i.e. when entering the Fast Retransmission Phase) and unacknowledged data are retransmitted using the slow start algorithm. The transmitter is also allowed to transmit a new data packet on receiving 2 duplicate ACKs. While the transmitter is in the fast retransmission phase, it continues to retransmit packets using slow start until all packets have been recovered (without starting a new retransmission phase for partial ACKs). Although this modification may cause unnecessary retransmissions, it reduces transmitter time outs and efficiently recovers multiple packet loss using partial ACKs. Reno and NewReno algorithms make it easier to cope with and recover from packet losses. These algorithms are still designed for congestion losses. They do not cope well with heavy channel losses are those seen in an unreliable medium. All simulations and comparisons in this thesis are made with TCP Reno.

The problem with sender based schemes is that they cannot be depended upon. Sender based schemes require widespread deployment to work effectively (there are more clients than servers). Widespread deployment is usually (almost) impossible.

1.3 Link Layer methods

Another area for improving performance under wireless networks is to use link layer schemes. Schemes such as low level retransmissions are implemented in most link layer protocols (e.g 802.11) to reduce the number of losses seen by the upper layer. This way the losses seen by the TCP sender are significantly reduced. Other methods such as FEC/ARQ [Chockalingam99] and link layer retransmissions [Wong99] have been proposed.

The advantage of link layer methods is that they can be implemented almost

independently of the other methods applied to the upper layers. The disadvantage is that changes in the link layer are expensive to implement. The whole hardware will have to be modified to include any modifications. Of course many of these changes may be implemented in an incremental manner. The methods suggested in this thesis may be implemented alongside any other link layer mechanisms.

Chapter 2.

TCP Drawbacks

Nature always sides with the hidden flaw.
Murphy's Ninth Law

Transmission Control Protocol (TCP) [RFC793] is widely used transport layer protocol. If Internet Protocol is the heart of the Internet, TCP is the soul. Despite many inherent drawbacks, TCP has been successfully implemented over many networks and it continues to be and most probably will be for the foreseeable future, the most popular transport layer solution for the Internet.

TCP has traditionally been designed to operate under a wired environment. This means the expected losses due to the channel errors are very small. TCP's main job is to counter the congestion and congestion related losses. TCP uses the slow start and congestion avoidance algorithms [RFC2001] to determine its optimal sending rate. TCP starts is transmission in the slow start phase, where it increases its congestion window at an exponential rate per round trip time. When the congestion window reaches its maximum possible value, i.e the channel is filled, and the loss event occurs, the congestion window is halved and TCP enters the congestion avoidance phase. In the congestion avoidance phase the congestion window increases at a very slow rate, about one segment per round trip time[1]. This leads to TCP's congestion window size varying like a sawtooth. This saw-toothed nature of TCP flows does not allow for maximum utilization of a channel. A single TCP flow may not be fully able to use a channel. For better channel utilization, multiple TCP flows must be used. Even with the case of multiple TCP flows,

1. Actually the congestion window increases at the rate of segsize*segsize/CWND, where segsize is the segment size and CWND is the congestion window size.

the problem of global synchronization exists. All the TCP flows through a channel may back-off simultaneously.

This phenomenon may be acceptable in a wired network. But in case of a wireless network, the cost of bandwidth is much more than that in a wired network. It is not an appropriate thing to under utilize this bandwidth in such a manner.

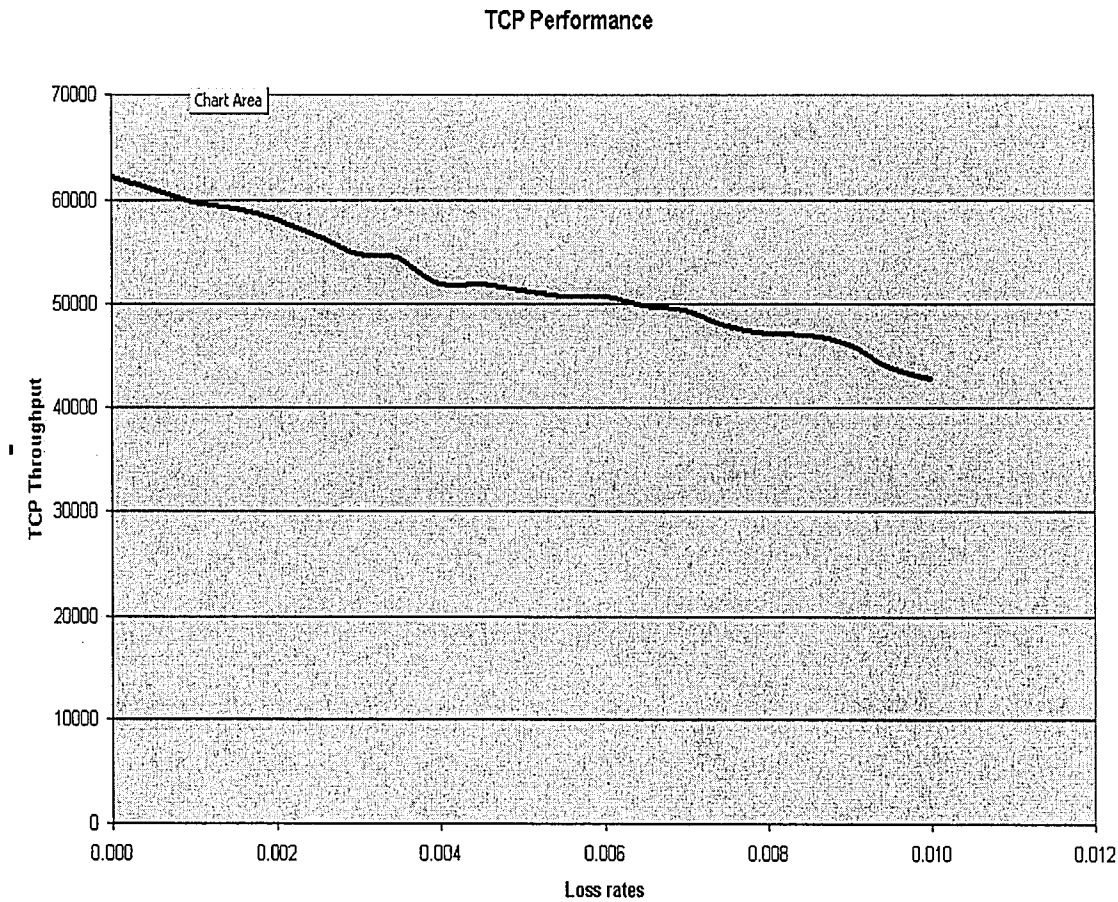

Figure 2.1: TCP Performance

TCP does not perform well in networks with a high channel loss rate. TCP incorrectly assumes channel losses to be those due to congestion and unnecessarily reduces its congestion

window and slows down its sending rate. Figure 2.1 depicts how channel losses affect TCP throughput. In some cases with a channel loss of 1%, TCP throughput came down by 40%. Channel losses are particularly high in case of wireless networks, making TCP unsuitable for wireless networks. The above figure is the result of a channel simulation. The channel bandwidth was set to 500 Kb/sec and the round trip time to 200 msec.

TCP performs poorly on LFNs (Long Fat Networks) ([Ubik03]). This is because the window size of the sender may not be large enough to fill the pipe. Since the window size field in the TCP header is only 16 bits, the maximum window size can be at most 65,535 bytes. Also because of the large size of the pipe, it may take a significantly larger amount of time to fill the pipe.

TCP/IP has a large header size (40 bytes). Assuming the packet to of the standard TCP segment size (536 bytes). The TCP/IP header along with the link layer header (10 byte header in case of ethernet) account for almost 10% of the payload size. Thus 10% of the bandwidth is used in just moving about packets. This large size appears unnecessary, given the expensive nature of end links.

These drawbacks make TCP a non-solution for wide area wireless networks. The light weight protocol proposed in this thesis attempts to solve many of these problems affecting wireless last mile links.

Chapter 3

A LightWeight Protocol for Weakly-Connected Clients

3.1 Network-layer behavior of LPWC

The light-weight protocol (LPWC) that connects the end client with the proxy, which in turn connects to the end server using a traditional TCP/IP connection. As described earlier, the problem with split connection proxies is that they violate the end to end principle [Saltzer84] of the Internet. What is actually needed here is a NAT-like gateway, which does not violate the end to end semantics, but also provides the required functionality.

This functionality could be provided using the TCP-Splice technique proposed by Bhagwat, et al. [SPLICE]. TCP-splicing provides an end-to-end acknowledgment scheme.

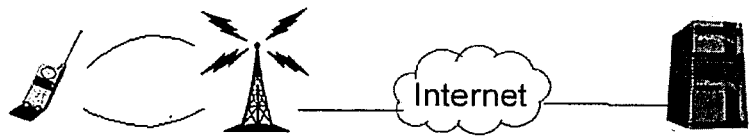

Figure 3.1: Connection topology

This chapter describes the network layer behavior of LPWC and how packets are routed through the network. The topology of a client-server setup using LPWC is as in the in the figure

described above.

The packets flowing between the client and the gateway are the LPWC packets. These packets have smaller header sizes (the 3 byte LPWC header). The packets flowing out of the gateway, to the server are the standard IP packets. The gateway performs the job of translating between LPWC headers to the IP headers and vice-versa. This functionality is quite similar to the network address translation (NAT) [RFC1631] feature commonly provided by routers. NAT is commonly used to share one public IP address among multiple clients.

The LPWC packets arriving at the gateway are translated into the IP packets. The gateway maintains the status of each connection and knows the source IP and destination IP, sequence numbers, etc, of the next packet to be sent on this TCP connection. The gateway manipulates the header and inserts the new IP headers and sends the packet on its way. Similarly, when the gateway receives a packet, it determines to which connection it belongs, and similarly manipulates the header and converts the IP header into the LPWC header and sends the packet to the client. LPWC can be implemented alongside a NAT implementation, thereby obtaining the benefits of both a NAT gateway as well as LPWC.

3.1.1 Overview of LPWC network layer behavior

LPWC's network layer issues are almost completely handled by the gateway and are to be implemented as a stub router function. The addresses used by the stub domain are not important. The link layer addresses itself could be used. One of our primary assumptions is that the clients are directly connected to the router. No routing information is needed (or is already available) to transfer the packets from the client node to the gateway.

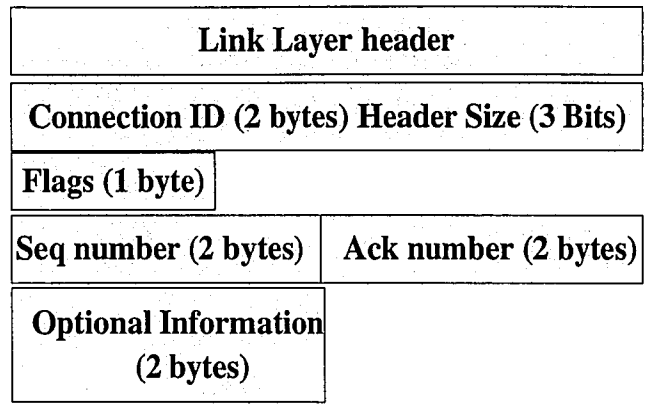

Figure 3.2: Data packet format

LPWC packet structure is shown in figure 3.2. Whenever a new connection is established between a client and a remote server, the gateway assigns an unique connection-ID is assigned to it. The connection-ID is 2 bytes long, which means there can be 64000 connections flowing through a gateway. The gateway maintains state information regarding each connection flow, just as in case of a NAT. The transformed port and IP-addresses are stored along with the connection-ID.

3.1.2 Various Aspects of the Protocol

Address Space

The client IP address seen by the outside world is that of the gateway it is using to communicate. The internal address the client uses to communicate with the gateway is the link layer address (MAC address). Since the client-gateway communication is over a single hop, we do not need anything other than the link layer address. This way another inefficiency of the Internet, having to use two unique addresses per machine, is eliminated.

Header Manipulations

In addition to manipulating the TCP and IP header fields, it is the Gateway's responsibility to calculate the TCP and IP checksums. The client does send a two byte link layer checksum while sending out its frame. The gateway uses this to verify that the packet has been correctly received. After manipulating the TCP and IP headers the gateway will have to recalculate the checksums of both headers and insert them.

If an ICMP message is passed through the gateway, it may require two address modifications and three checksum modifications. This is because most ICMP messages contain part of the original IP packet in the body. Therefore, for the gateway to be completely transparent to the host, the IP address of the IP header embedded in the data part of the ICMP packet must be modified, the checksum field of the same IP header must correspondingly be modified, and the ICMP header checksum must be modified to reflect the changes to the IP header and checksum in the ICMP body. Furthermore, the normal IP header must also be modified as already described.

It is not entirely clear if the IP header information in the ICMP part of the body really need to be modified. This depends on whether or not any host code actually looks at this IP header information. Indeed, it may be useful to provide the exact header seen by the router or host that issued the ICMP message to aid in debugging. In any event, no modifications are needed for the echo and time stamp messages, and NAT should never need to handle a Redirect message. SNMP messages could be modified, but it is even more dubious than for ICMP messages that it will be necessary.

3.2 Gateway Discovery protocol

The gateway discovery protocol works much like Mobile IP. Gateways periodically announce their presence with advertisement packets. Clients read these advertisements and know to which subnet they are connected to. The hardware address sent along with the gateway advertisements is used by the client to send outgoing packets. When a node detaches from one network and attaches to another one, there will be a handoff problem. Handoffs are a difficult task in any split connection environment. Some techniques used to alleviate this problem are described here.

3.2.1 Setting up a connection

The section describes the steps followed by the client to setup a connection with a remote server. The client needs to discover its proxy to help setup connections with the outside world. The "gateway discovery protocol" is first used to determine the gateway. Once the gateway is known, it can be used to connect to the outside world.

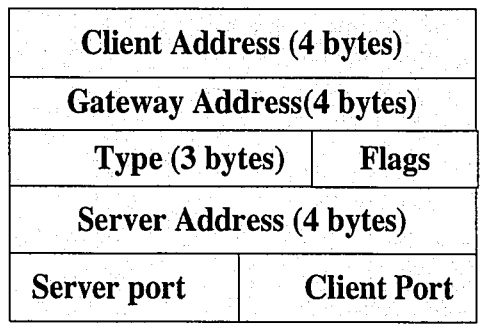

Figure 3.3: Connection request packet format

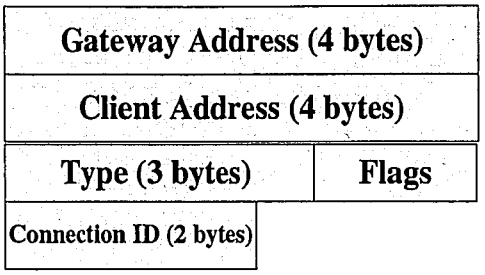

Figure 3.4: Connection response packet format

Once the gateway is known, the client must first request a connection-ID specifying its port number before starting a transaction. In the connection-ID request, the client specifies the remote server address and the remote port number. The client sends a connection request packet to the gateway to start the connection. Connection request and response packet structure is shown in figure 3.3 and figure 3.4.

When the connection request is sent to the gateway, it reserves some buffer space for the client's packets on this connection. From now on, the gateway knows what to do with the response of a particular packet sent by the client, or a packet received from the server on this connection. The client now simply sends a normal SYN packet to the gateway which passes it on to the server and the connection is established in normal TCP way. The connection requests have the SYN bit turned on in the LPWC packet's flags. The client must also include information regarding the remote server address with which it wants to connect.

3.2.2 Handoff problem

Handoffs are needed when a mobile node moves from one cell to another. Clients are connected to the home base station by the light-weight protocol. There exist TCP connections belonging to the client from the base station to servers elsewhere.

This might present a peculiar problem, since the new base station may not be ready to serve a new client, or worse the new base station may not be supporting the new protocol. One would expect the network architecture to be suitably designed to handle such changing conditions. The following section describes how the protocol handles such situations and ensures the client remains connected.

3.2.3 Handoff Algorithm

The client, while negotiating with its home base station (HS) initially, obtains address information about the HS and stores it. Each base station usually sends out advertisement beacons periodically to inform new clients about the services it offers. This is specified by mobile IP [RFC2002]. The new foreign base station (FS) may or may not support the light-weight protocol. It however is expected to route TCP/IP packets and provide support for mobile IP. The mobile host (MH) on reading one of these beacons discovers it is out of the reach of HS and sends the information of the HS to the FA. The HS meanwhile does not receive any acknowledgment packets from the MH and consequently does not send any acknowledgments to the server. It as usual buffers one window's worth of data.

In case the the FS does not support the new protocol, the client will generate TCP packets, which will be routed to the new FS via the HS just as in the case of Mobile IP. We will still be suffering from the triangle routing problem and we cannot expect major gains by using the protocol.

If the FS supports the new protocol then the client first informs the FS about its HS. The HS is then informed of the new location of the MH. Some authentication mechanism is needed for credibility. Also the FS's credentials must be verified before initiating any transactions with it.

The addressing scheme for LPWC is similar to a NAT scheme. There exists a TCP connection between HS:portnum and server and a LPWC connection between MH:port and HS. The server sees the HS:portnum to be the address of the client. When the client moves to a new FS, a LPWC connection is established between the FS and the MH. The HS:portnum packets which are received on the HS, are forwarded to the FS. Its the responsibility of the FS to send out the ACKs for these packets.

3.2.4 Shifting Gateways

The client maintains a list of possible gateways it can use to connect to the outer world by looking at the list of advertisements it has received. An advertisement is valid only for a particular period of time, specified by the TTL field in the advertisement. The client may choose its gateway or use different gateways for different connections, from this list. We assume the client is using a particular gateway g1 for a connection c1. In the middle of a transfer lets assume the client moves out of the range of g1 and into the range of g2.

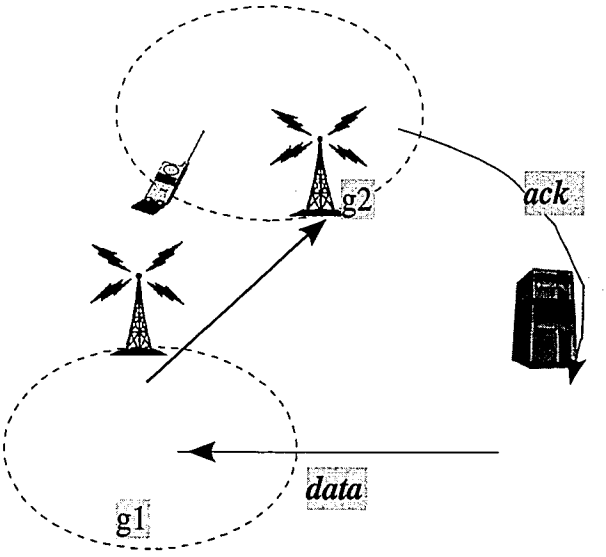

Figure 3.5. Triangle Routing

The client detects the presence of g2 by receiving an advertisement from it. The client detects that g1 is no longer accessible after the TTL of g1's advertisement expires. Ideally we would want g1's connection data and buffers to be somehow moved to g2. This way the connection could move on seamlessly. However since the outside world only sees the IP address of the gateway, a change in the gateway would mean a change in IP address. Thus the end machine with whom the client was communicating would send its packets to the older gateway g1 only.

A solution to this situation is to use g1's IP address for existing connections and use g2's address for any new connections being made. After the client discovers that it is no longer in the range of g1, it sends a message to g1 via g2, informing it of the new location of the client and asking it to re-route all packets bound to g2. The client also informs g2 of its current connections. When g1 receives a packet bound for our client, it changes the destination IP of this packet and sends it to g2. g2 looks at the sender-port and destination-port combination and determines that the packet is bound for this client. The packet header is compressed and sent to the client. All outgoing packets on this connection sent by the client are sent directly to the end machine by g2 itself. Since g2 knows the end IP address it can route the packets by itself. The triangle routing scenario exists in this proposed model as well. Figure 3.6 below describes advertisement packet formats.

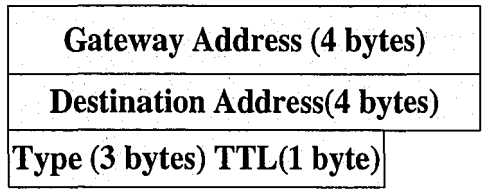

Figure 3.6: Advertisement Packet Format

The destination address is the broadcast address.

3.3 Transport Level Behavior

LPWC is not a multi layer protocol. Its made of a single layer, but spans multiple layers. In addition to moving data from and to the client, LPWC is also responsible for handling packet retransmissions in the event of packet losses. The gateway is responsible for storing recently seen packets in a cache. In previous chapters TCP's inability to handle channel losses properly and the problems that arise out of this have been described. This section describes the loss handling characteristics of LPWC and how it helps TCP maintain a good throughput rate.

3.3.1 Detecting Packet Loss

The gateway maintains the status of each connection that passes through it. The most recent acknowledgment that has been sent by the client is stored. When a packet does not reach the client, the client detects this loss by sending a duplicate acknowledgment, i.e., it resends the acknowledgment for the last packet it received in sequence. The gateway can detect these duplicate acknowledgments by looking at the current acknowledgment frame and the last acknowledgment it received. If both have the same sequence numbers then some data has been lost. The sequence of bytes that have been lost can been found by looking at the sequence numbers of the two acknowledgment packets. Duplicate acknowledgments are the most common method of detecting packet losses. In some circumstances though, the gateway may not receive an acknowledgment for a considerable period. A timeout is used to detect packet losses in such a case. The timeout maybe a calculated using the round trip times. A simple standard, such as four RTTs could be used as a timeout period. If two timeouts occur consecutively, the gateway assumes the client is unreachable and does not attempt to retransmit again. In any case, if the acknowledgments are not flowing towards the TCP sender, it by default times out and slows

down the sending rate (which is the correct thing to do). The gateway controls the TCP flow by adjusting the advertisement window value so that the sender times out only when the client is unable to receive or transmit.

3.3.2 Hiding Packet Losses

The gateway already maintains a cache of recently seen packets. The gateway maintains the cache as a sequence of bytes. It also records the sequence number of the first byte of the cache and the current size of the cache. On detecting a duplicate acknowledgment, the gateway builds a packet or possibly multiple packets in case more than one packets were lost and sends them to the client. These duplicate acknowledgments are discarded and not sent to the TCP sender. This way the packet losses due to the channel bit errors are hidden from the TCP sender. As mentioned in the previous section, the gateway also makes a retransmission in the case of a timeout and ignores a second timeout.

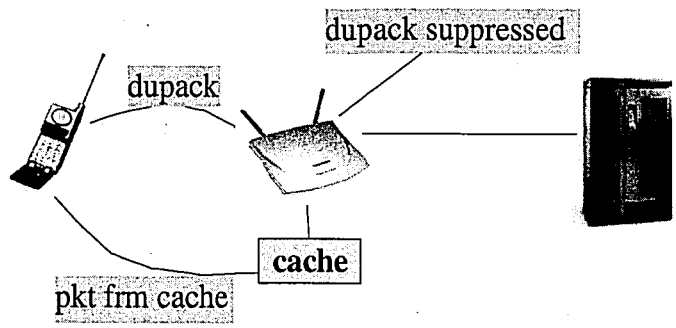

Figure 3.7: Gateway Cache

3.3.3 Gateway Cache

The gateway maintains a cache of recently seen data packets sent to the client. This cache is necessary to hide the packet loss from the TCP sender. All packets which have not yet been

acknowledged by the client are stored in this cache. The size of the cache needed (to maintain incoming packets) will be equal to the maximum possible size of the TCP sender (since the maximum possible unacknowledged data at any time is equal the the window size of the TCP sender). The gateway does not maintain a cache for the out going acknowledgment packets sent by the client. Determining the maximum possible window size is a bit complicated. Ideally the maximum window size would be (Channel Bandwidth * RTT). The bandwidth and round trip times are determined during the connection setup period. However both these values keep changing dynamically throughout the lifetime of the connection. When the client negotiates with the gateway to request buffer space, the bandwidth and the round trip times are calculated using the packet pair algorithm [Lai01]. This way the gateway knows the maximum buffer size that needs to be allocated to a particular TCP connection that flows to a client. The cache itself is maintained as a byte queue of a particular size. When data arrives on a full queue, the front of the queue is pushed out making space for the newly arrived data. The data which has been removed has already been acknowledged by the client.

Chapter 4

Simulation Setup

The protocol proposed in this thesis was tested on a simulation testbed. The testbed was made using the NS network simulator. NS (version 2) is an object-oriented, discrete event driven network simulator developed at UC Berkeley written in C++ and OTcl. NS is primarily useful for simulating local and wide area networks. NS is an event driven network simulator, that simulates variety of IP networks. It implements network protocols such as TCP and UPD, traffic source behavior such as FTP, Telnet, Web, router queue management mechanism such as Drop Tail, RED and CBQ, routing algorithms such as Dijkstra, and more. NS also implements multicasting and some of the MAC layer protocols for LAN simulations. NS is now a part of the VINT project that develops tools for simulation results display, analysis and converters that convert network topologies generated by well-known generators to NS formats. Currently, NS (version 2) written in C++ and OTcl (Tcl script language with object-oriented extensions developed at MIT) is available.

4.1 NS Architecture

NS is written not only in OTcl but in C++ also. For efficiency reason, NS separates the data path implementation from control path implementations. In order to reduce packet and event processing time (not simulation time), the event scheduler and the basic network component objects in the data path are written and compiled using C++. These compiled objects are made available to the OTcl interpreter through an OTcl linkage that creates a matching OTcl object for

each of the C++ objects and makes the control functions and the configurable variables specified by the C++ object act as member functions and member variables of the corresponding OTcl object. In this way, the controls of the C++ objects are given to OTcl. It is also possible to add member functions and variables to a C++ linked OTcl object. The objects in C++ that do not need to be controlled in a simulation or internally used by another object do not need to be linked to OTcl. Likewise, an object (not in the data path) can be entirely implemented in OTcl. Figure 4.1 shows an object hierarchy example in C++ and OTcl. One thing to note in the figure is that for C++ objects that have an OTcl linkage forming a hierarchy, there is a matching OTcl object hierarchy very similar to that of C++.

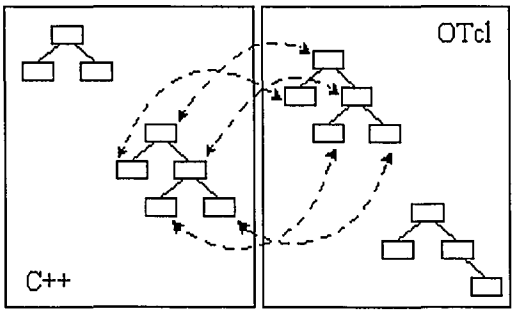

Figure 4.1: C++ and OTcl duality.

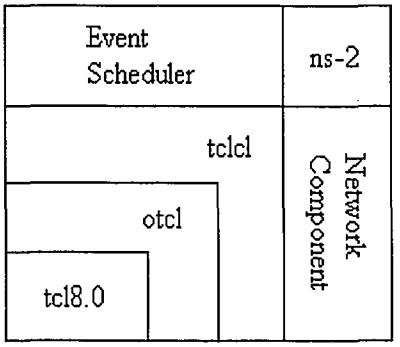

Figure 4.2: Architectural View of NS.

Figure 4.2 shows the general architecture of NS. In this figure a general user (not an NS developer) can be thought of standing at the left bottom corner, designing and running simulations in Tcl using the simulator objects in the OTcl library. The event schedulers and most of the network components are implemented in C++ and available to OTcl through an OTcl linkage that is implemented using Tcl. The whole thing together makes NS, which is a OO extended Tcl interpreter with network simulator libraries.

This section briefly examined the general structure and architecture of NS. At this point, one might be wondering about how to obtain NS simulation results. As shown in Figure 4.1, when a simulation is finished, NS produces one or more text-based output files that contain detailed simulation data, if specified to do so in the input Tcl (or more specifically, OTcl) script. The data can be used for simulation analysis (two simulation result analysis examples are presented in later sections) or as an input to a graphical simulation display tool called Network Animator (NAM), that is developed as a part of VINT project. NAM has a nice graphical user interface similar to that of a CD player (play, fast forward, rewind, pause and so on), and also has a display speed controller. Furthermore, it can graphically present information such as throughput and number of packet drops at each link, although the graphical information cannot be used for accurate simulation analysis. Another reason to use the NS simulator is ease with which one can extend it. Users can write custom C/C++ modules for experimental protocols and attach them to NS. Protocols can be added at the layers 2 ,3 or 4.

4.2 NS Modifications

For simulating the proposed protocol, I implemented a custom sender module and a custom receiver module. The sender and receiver could be controlled from the Tcl script. They

do not send out any packets automatically unless explicitly commanded to do so by the Tcl script. The way the protocol packets are sent and received is controlled from the Tcl Script. The actual packets sent along the wireless link are UDP packets with their size modified. The acknowledgments along the path are UDP packets as well. I used ftp at the application layer since I was testing for bulk transfers.

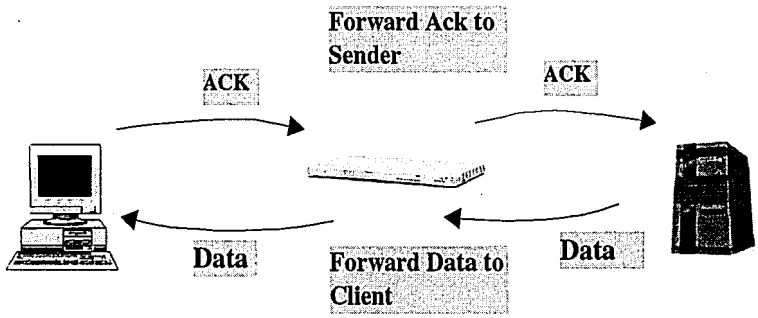

Figure 4.3: End to End Acknowledgment Scheme

The gateway which is the sender (to the client) takes care of retransmitting lost packets. A TCP (Reno TCP) connection is opened between the gateway and the end server. The acknowledgments on this connection are not immediately sent. The acknowledgment is sent only after the ack for the packet sent to the end client is received at the gateway, thereby maintaining an end to end acknowledgment scheme.

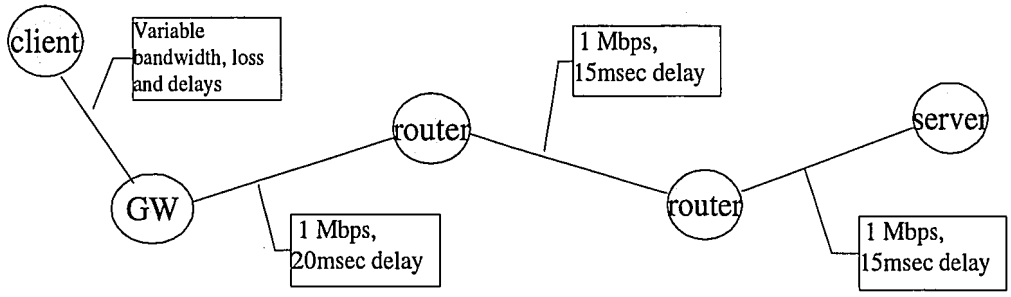

Figure 4.4: Simulation topology

We used a simple network topology for our simulations. The client is directly connected to the wireless gateway. Packets are routed from the gateway to the server via two intermediate routers. Figure 4.4 describes the topology and parameters used.

We experimented variable bandwidths from 5 Kbps to 1Mbps on the wireless links and loss rates of 0 to 2%. One way delay was varied between 50 to 500 msec.

Chapter 5

Performance Comparisons

In this chapter we compare TCP and the light weight protocol's performance. As described in earlier chapters, TCP's flaws with regards to wireless networks are:

1. Underutilization of available bandwidth.
2. Poor performance under lossy channels.
3. Poor performance in networks with a high round trip time. Wireless channels typically have higher round trip times.
4. Large header sizes.

I have not addressed the first issue in this thesis. Since we need an end to end acknowledgment scheme, it is still left to the TCP sender to determine the appropriate sending rate. This is probably still an open issue although there has been at least one suggestion made to avoid the TCP sawtooth. The TCP TEAR (TCP Emulation At Receivers) [TEAR] project has suggested that feedback from the receiver could be used to determine the appropriate receiving rate. TEAR receivers calculate a fair receiving rate which is sent back to the senders. There is a congestion window maintained at the receiver end that is modified similarly to the sender's congestion window. TEAR receiver estimates from the arriving packets when TCP would change its congestion window size. To address the issue of TCP sawtooth, TEAR averages the rate over an epoch, this is the period of time between two consecutive rate reductions. To prevent further rate changes caused due to noise or loss patterns, a smooth rate is calculated over a number of epochs for a final rate. This rate is sent to the sender. The rate calculated is the the maximum

possible channel throughput. TCP TEAR techniques can be easily implemented along with the light weight protocol. There are no special adjustments required to put TEAR into place.

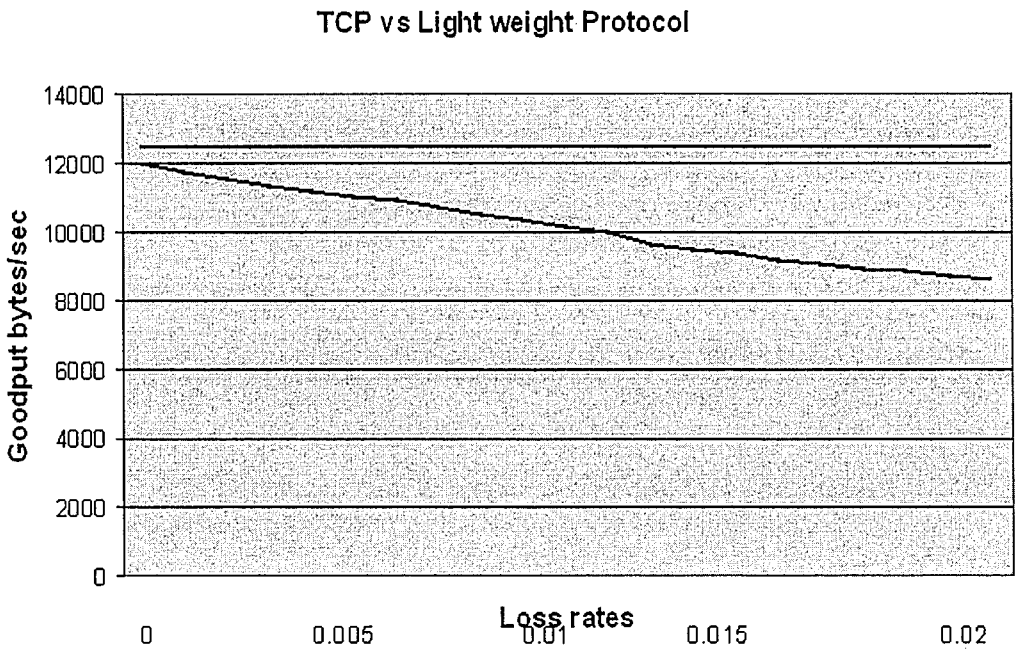

Figure 5.1: Performance comparisons

TCP does not perform well under lossy channels. TCP assumes channel losses to be caused by congestion and unnecessarily reduces its congestion window thereby reducing the effective throughput rates. LPWC handles this issue by hiding channel errors from the sender. The gateway holds a buffer of recently seen packets. Packets in this buffer are those which have not yet been acknowledged by the client. The maximum size of this buffer is same as the maximum size of the congestion window. The gateway detects channel losses by looking at duplicate acknowledgments. Duplicate acknowledgments are not sent to the sender. The lost packet is sent again by the gateway to the client. This way the sender never notices the lost

packets. Part of the bandwidth is lost due to the channel losses. I have experimented with loss rates of 2% packet loss. In my experiments I have found LPWC to demonstrate almost no change in throughput due to the effects of channel losses. No change was found even due to increased loss rates. This shows that TCP did not completely utilize the available bandwidth. Figure 5.1 compares the goodput values of TCP and LPWC under increasing loss rates. Goodput refers to the measurement of actual data successfully transmitted from the sender to receiver.

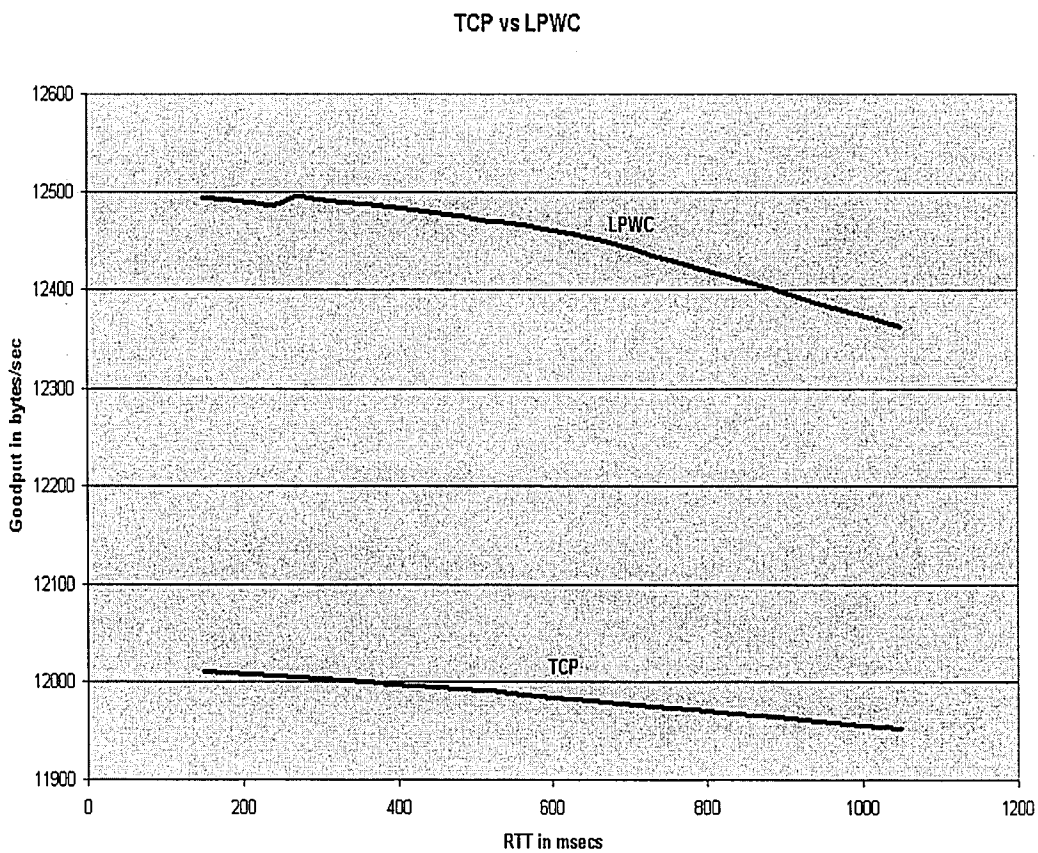

Figure 5.2: Performance comparisons

The next issue with TCP is coping with long round trip times. Long round trip times are undesirable for two reasons. Large windows are needed at the sender end to completely utilize

the available bandwidth. Large round trip times also mean slow start will be much slower. The time taken for TCP to reach its full speed sending rate will be higher than in those networks with smaller round trip times. Figure 5.2 compares the effective good put of TCP and LPWC under identical network conditions with no channel losses.

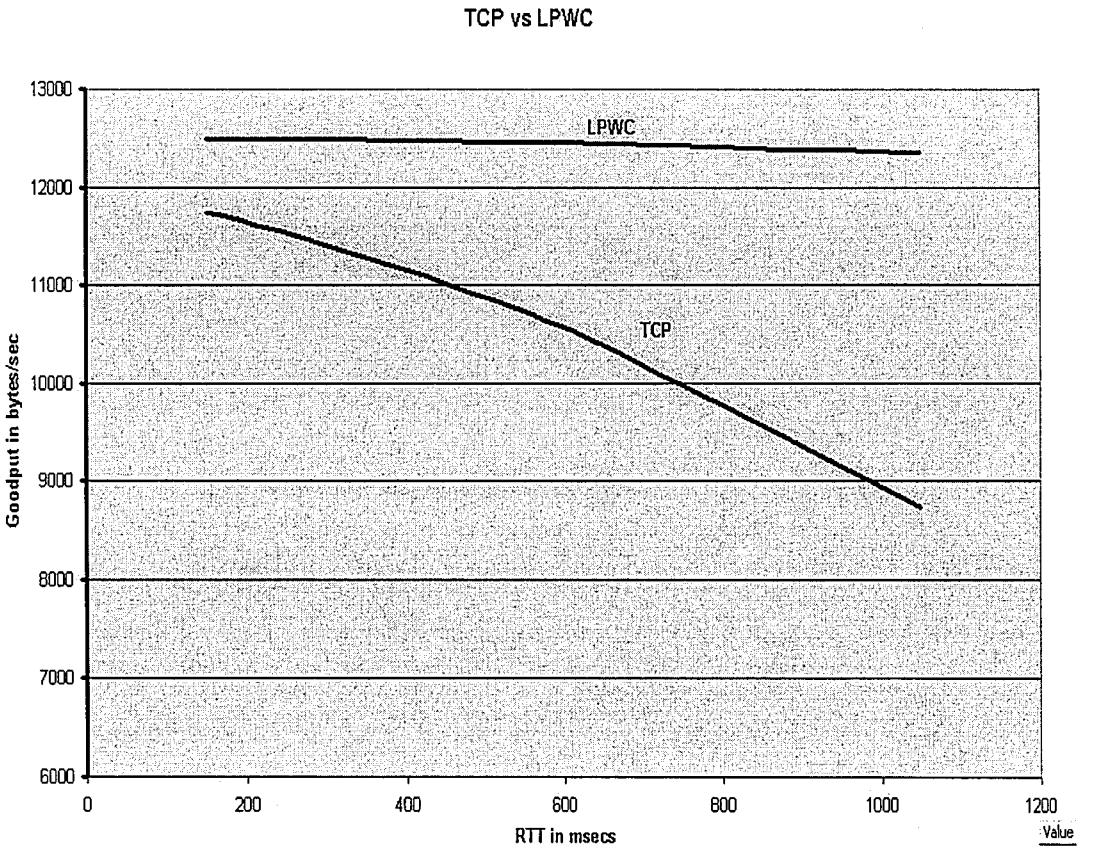

Figure 5.3: Performance comparisons

The effect of high RTTs is prolonged in lossy networks. If a packet loss causes TCP to slip back into a slow start phase, a lot more time is wasted while increasing TCP's congestion window. Hiding the packet losses from the end server in this cases helps a lot in improving the end throughput. Hiding the packet losses would help prevent the sender from slipping into a slow

start mode. The comparison between LPWC and TCP under link with a high round trip time and with small amount of channel losses (0.05%) is shown in figure 5.3.

Figure 5.4 compares the performance difference between TCP and LPWC with increasing bandwidth. We find that the difference between TCP and LPWC increases with increasing bandwidth. Figure 5.5 shows the performance difference as a percentage value. LPWC performs much better than TCP at higher bandwidths because it is able to completely utilize the available channel bandwidth.

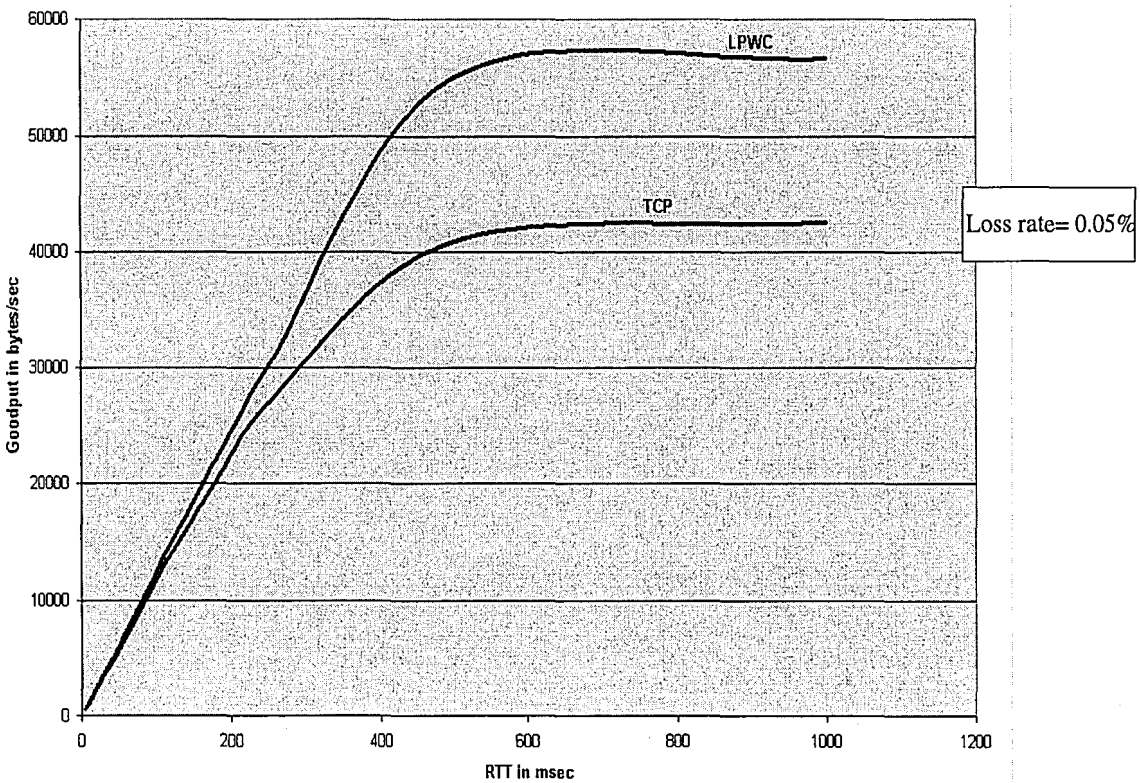

Figure 5.4: Performance comparisons

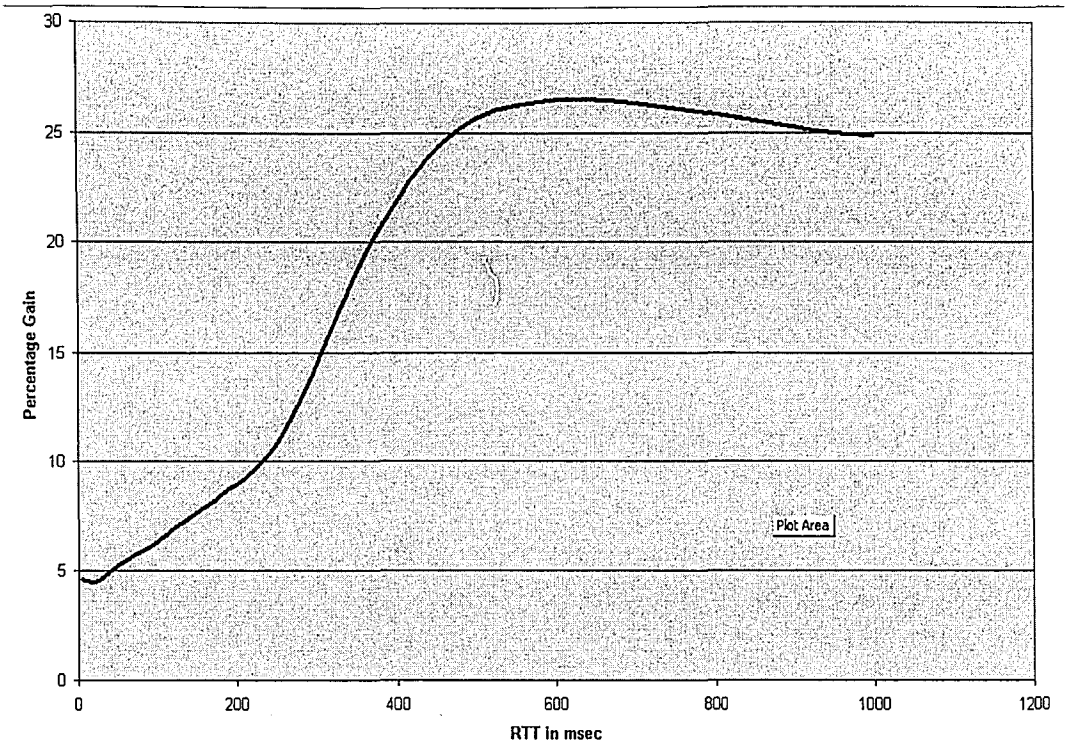

Figure 5.5: Performance comparisons

The last issue tackled in this thesis is that of header sizes. The proposed framework allows for the usage of a header compression scheme on the link between the client and the gateway. This thesis suggested a simple algorithm which brought down the header from 40 bytes large to about 10 bytes. Although developing a header compression scheme was not a goal of this thesis, More importantly the goal of the thesis is to design a framework that allows for the usage of a header compression scheme. The framework also simplifies the process of assigning addresses to clients by using the link layer address to be used on the network level as well.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

This thesis identifies three reasons for TCP's performance degradation in a low bandwidth network and methods to cope up with the performance loss caused by each of these three reasons. Earlier works based on split connection models ([M-TCP], [I-TCP], [MOWGLI] etc.) violate the end to end principle of the Internet. This thesis proposed a split connection model that conforms to the end to end principle. The proposed protocol framework also incorporates features such as header compression and link layer data compression.

6.1.1 Excessive packet losses caused due to bit errors in wireless networks.

TCP's performance in many wireless networks suffers due to bit error induced packet losses, which usually occur in bursts due to the nature of wireless channels. These error induced packet losses are misinterpreted by the TCP sender, and are assumed to be caused by network congestion. This causes the TCP sender to invoke its congestion control algorithms and reduce its transmission window size, and often causes long timeouts which keep the channel idle for long periods. Thus bit errors degrade TCP's performance by a significant amount.

TCP's unsuitability for networks with high loss rates and/or low bandwidths has been demonstrated in this thesis (as well as in other works such as [SNOOP], [M-TCP], [MOWGLI], [I-TCP]). TCP's inherent design flaws make it unsuitable for low reliability networks. A new protocol has been proposed to replace TCP at the last mile link. The protocol has been designed for a small scale induction and does not require major changes in the existing network setup. The

new protocol aims to reduce header sizes, improve effective throughput by masking channel losses and at the same time maintain the end to end semantics of the connection. The new protocol shows up well in simulation against TCP. A throughput improvement of up to 30% was found. The new protocol uses the available bandwidth in a more efficient manner than TCP.

6.1.2 Large header sizes of TCP/IP protocol stack.

The TCP/IP protocol stack needs at least 40 bytes to make up the packet header. This is not acceptable for low bandwidth networks (CDPD, GSM etc.). The header itself makes up a sizable part of the packet. By having a 40 byte header in addition to a link layer header, a sizable chunk of the packet is devoted to headers itself. This thesis proposes the use of header compression to be implemented along with this light weight protocol. Chapter 3 suggested a few simple methods which reduce the header sizes by about 30 bytes and bring down the transport/network layer header to about 3 bytes. More complex techniques could be put in place to bring down the header size to an even smaller number.

6.1.3 Underutilization of the wireless channel.

This is an issue which has not been thoroughly addressed by researchers in the academic world. The saw tooth pattern of TCP throughput does not allow it to completely utilize available bandwidth. Ideally we would like TCP throughput to maintain a constant optimal throughput rate that is the maximum possible throughput that the channel allows in the given circumstances. In order to completely fill up the channel, a number of TCP connections should be open and running at the same instance. Sivakumar, Bailey and Grossman used this idea of using multiple parallel connections to stripe the same data transfer. They developed the Psockets library

[PSockets], a TCP socket library for high speed networks. By using LPWC on the last link, we are preventing underutilization by a significant degree, but since the sender is still traditional TCP, the "saw tooth" effect is still in place and a part of the bandwidth is being left unused. One solution proposed was the TCP TEAR method. TCP TEAR uses feedback from the receiver to calculate the congestion window size. The feedback obtained is used to set the optimal congestion window size. TCP TEAR however requires widespread deployment for it to work correctly. TCP TEAR techniques can be incorporated into LPWC quite easily. The gateway reads the packet patterns and provides feedback to the sender. The packet format of LPWC will not need any modifications. The task of optimal rate calculation is delegated to the gateway. However sending feedback from the receiver may not always be possible. Sender end modifications will be needed for such a system to work correctly. Receiver feedback may not be the best way to achieve complete bandwidth utilization. It is just a suggestion that this thesis makes. Future research may bring out possibilities that are unknown now.

6.2 Future Work

6.2.1 Header Compression Algorithms

There are a couple of areas where the performance of the light weight protocol can be improved. The header compression algorithm described in this thesis is a very simple and straight forward algorithm. The gains that are obtained by using header compression can be further improved. Better header compression algorithms have been proposed in [Jiao00] [Auge99] and [Liao01]. By modifying one of these algorithms to suit the protocol, we can expect further gains. The available bandwidth on the wireless link is not yet being completely utilized. By using LPWC on the last link, we are preventing underutilization by a significant degree, but since the

sender is still traditional TCP, the "saw tooth" effect is still in place and a part of the bandwidth is being left unused. Solutions proposed by TCP TEAR can be incorporated into LPWC quite easily. A proper means is still required to completely utilize the channel bandwidth.

6.2.2 Security

Another issue with the protocol is regarding security. With its current design and packet format, it is difficult to encrypt packet headers. Although the application may use its own data encryption method with any problems, encrypting transport and network layer headers is difficult with the current model. An end to end encryption system like IPSec [RFC2401], that would encrypt the headers as well as the data, cannot be put in place with the current network model. We could have a system where the packets are encrypted from the client to the gateway, decrypted there, and encrypted again to be sent to the TCP sender. Although this is not an end to end encryption method, it should work well enough to provide a secure method of data transmission.

6.2.3 Link Layer Data Compression.

Bandwidth in wireless networks will always be comparatively low. One must attempt to send more information using less bandwidth. It would of interest to consider the possibility of using data compression on the link layer between the client and the wireless gateway. A standard dictionary based algorithm such as the Liv Zempel-Welch [LZW] may be used to compress the data that is being sent by the client. The gateway and the client would be required to agree on a particular dictionary or alternatively a dictionary is built on the fly as the data is being transmitted. The compression algorithm would be part of the link layer. The payload of the link

layer frame would be compressed. In [RFC1962] Rand and Novel proposed a compression protocol for point to point links. A similar scheme could be adapted to suit the proposed protocol.

6.3 Summary

Split connection models have been shown (both in this thesis and elsewhere) to be a useful technique to overcome the performance degradation of TCP over lossy channels. This thesis presents a the split connection model to improve TCP performance at the last-mile. Unlike other techniques, the proposed approach uses end to end acknowledgment scheme and adheres to the end to end principle. The proposed framework also incorporates features such as protocol header compression and link layer data compression. In addition to describing the protocol architecture, this thesis has presented the results of simulation to estimate some of the performance improvements possible under various scenarios. This framework can provide benefits even with limited deployment, and can be implemented easily and in an incremental fashion.

Appendix

Packet types and their Codes

A.1 Advertisement Beacon:

Packet code 0x8001

Advertisement Beacons contain the hardware address of a gateway willing to service clients in that region. Clients receiving a beacon may store it and use this gateway to communicate with the outside world. A client may simultaneously utilize the services of multiple gateways. Each beacon has a particular TTL, after the TTL expires the client must assume it is not in the range of a particular gateway unless it receives a fresh advertisement.

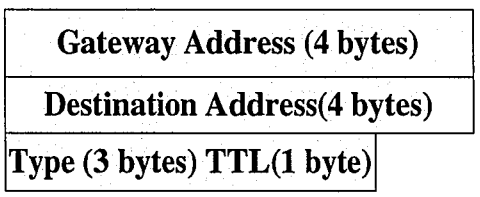

Figure A.1: Advertisement Packet Format

A.2 Connection Requests/ Responses:

Packet code 0x8002

Connection requests are sent by clients to the gateway asking for a new channel to be set up. Client specify the destination IP address and port number in the connection request. On receiving a request, the gateway allocates some buffer space (of a window's worth) for this connection. The gateway returns a connection ID to the client. All subsequent data packets sent

by the client should be marked with this connection ID.

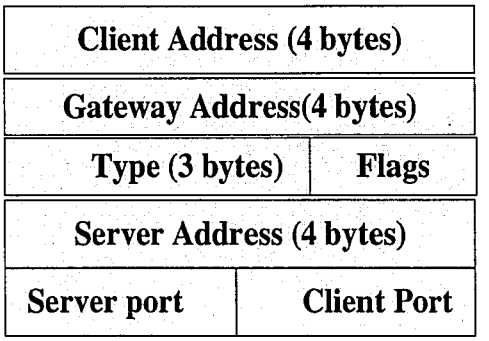

Figure A.2: Connection request packet format

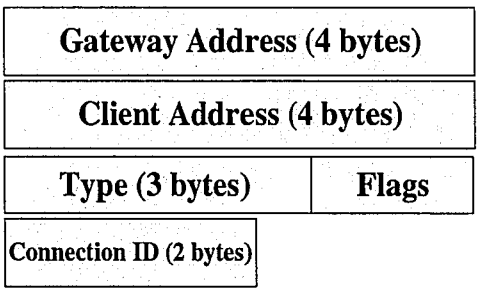

Figure A.3: Connection response packet format

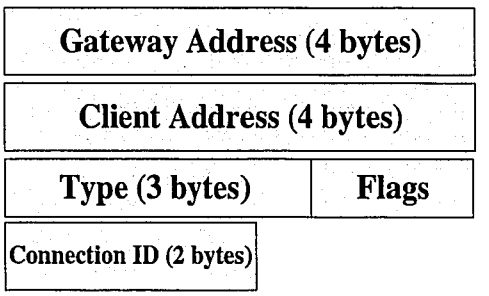

Figure A.4: Connection release packet format

A.3 Data Packets

These are the normal TCP packets sent out of the subnet. Its the gateway's responsibility to

convert them from LPWC format to TCP/IP format. The packet format for an ethernet network has been shown here.

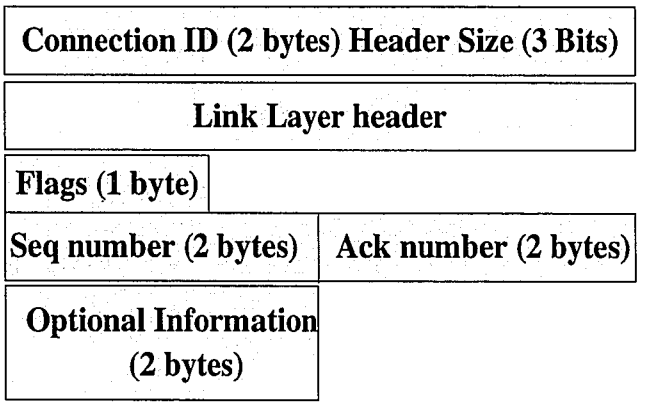

Figure A.5: Data packet format

References

If I have seen further it is by standing on the shoulders of Giants
-Isaac Newton

[Auge99]: Anna Calveras-Augé, Miquel Arnau-Osorio, and Josep Paradells-Aspas.
A Controlled Overhead for TCP/IP Header Compression Algorithm over Wireless Links. The 11th International Conference on Wireless Communications. Calgary. Alberta. Canada. July 1999.

[Chockalingam99]: A. Chockalingam, M. Zorzi, and V. Tralli.
Wireless TCP performance with link layer FEC/ARQ.
Proceedings of the IEEE International Conference on Communications.
Pages 1212- 1216. June 1999.

[Compression]: Jean-Loup Gailey and Mark Nelson.
The Data Compression Book. M&T Books, New York, NY 1995

[Davison02]: Brian Davison, Kiran Komaravolu, and Baoning Wu.
A Split Stack Approach to Mobility-Providing Performance-Enhancing Proxies Technical Report LU-CSE-02-012, Department of Computer Science and Engineering, Lehigh University. November 2002.

[Duska97]: Bradley M. Duska, David Marwood, and Michael J. Feeley.
The Measured Access Characteristics of World-Wide-Web Client Proxy Caches.
Proceedings of the USENIX Symposium on Internet Technologies and Systems.
December 1997.

[I-TCP]: A. Bakre and B. Badrinath.
I-TCP: Indirect TCP for Mobile Hosts.
Proc. 15th Intl. Conference on Distributed Computing Systems (ICDCS), Vancouver, Canada, May 1995.

[Jiao00]: Changli Jiao, Loren Schwiebert, and Golden Richard.
Adaptive Header Compression for Wireless Networks.
Proceedings of the 26th Conference on Local Computer Networks (LCN), pages 377-378, November 2001, Tampa, Florida.

[Lai01]: Kevin Lai and Mary Baker.
Nettimer: A Tool for Measuring Bottleneck Link Bandwidth
Proc. 3rd USENIX Symposium on Internet Technologies and Systems (San Francisco CA, March 2001) 122-133

[Liao01]: H. Liao, Q. Zhang, W. Zhu, and Y.-Q. Zhang,
A robust TCP/IP header compression scheme for wireless networks.
IEEE International Conference on 3G wireless and Beyond (3Gwireless'01),
June, 2001, San Francisco, CA, USA.

[LZ77]: J. Ziv and A. Lempel.
A Universal Algorithm for Sequential Data Compression.
IEEE Transactions on Information Theory, Vol. IT-23, No. 3, May 1977, pp. 337-343

[LZW]: Terry A. Welch.
A Technique for High Performance Data Compression.
IEEE Computer, 17(6), June 1984, pp. 8-19

[NS]: The Network Simulator
www.isi.edu/nsnam/ns

[NsManual]: The Network Simulator Manual.
http://www.isi.edu/nsnam/ns/doc/index.html

[NsTutorial]: Jae Chung and Mark Claypool.
Ns by Example.
http://nile.wpi.edu/NS/

[M-TCP]: K. Brown and S. Singh.
M-TCP: TCP for Mobile Cellular Networks.
ACM Computer Communication Review, 1997.

[MOWGLI]: M. Kojo, K. Raatikainen, and T. Alanko
Connecting Mobile Workstations to the Internet over a Digital Cellular
Telephone Network.
Proceedings of the Mobidata Workshop, Rutgers University, NJ, Nov 1994.

[MSOCKS]: David Maltz, and Pravin Bhagwat.
MSOCKS: An architecture for transport layer mobility
Proceedings of INFOCOM '98, April 1998, San Francisco, CA.

[PSockets]: H. Sivakumar, S. Bailey, and R. Grossman.
PSockets: The Case for Application-level Network Striping for Data Intensive
Applications using High Speed Wide Area Networks.
Proceedings of 2000 ACM/IEEE conference on Super Computing September 2000.

[RFC1144]: Van Jacobson.
Request for Comments: Compressing TCP/IP Headers for Low Speed Internet
Connections. February 1990.

[RFC1962]: D. Rand.
Request for Comments: The PPP Compression Control Protocol (CCP). June 1996

[RFC 2001]: Richard W. Stevens.
Request for Comments: TCP Slow Start, Congestion Avoidance, Fast Retransmit, and Fast Recovery Algorithms. January 1997

[RFC2002]: C. Perkins.
Request for Comments: IP Mobility Support. October 2001.

[RFC2401]: S. Kent and R. Atkinson.
Request for Comments: Security Architecture for the Internet Protocol. November 1998.

[RFC2507]: M. Degermark, B. Nordgren, and S. Pink.
Request for Comments: IP Header Compression. February 1999.

[RFC 3243]: L.-E. Jonsson.
Request for Comments: RObust Header Compression (ROHC): Requirements and Assumptions for 0-byte IP/UDP/RTP Compression. April 2002.

[RFC793]: Jon Postel.
Request for Comments: Transmission Control Protocol. September 1981

[SNOOP]: Hari Balakrishnan.
Challenges to Reliable Data Transport over Heterogeneous Wireless Networks. University of California at Berkeley, Department of Electrical Engineering and Computer Sciences. August 1998.

[TEAR]: Injong Rhee, Volkan Ozdemir, and Yung Yi.
TEAR: TCP emulation at receivers -- flow control for multimedia streaming. Technical Report, Department of Computer Science, NCSU. 2000.

[Ubik03]: Sven Ubik and Pamel Cimbal.
Achieving Reliable High Performance in LFNs.
Proceedings of Terena Networking Conference, Zagreb Croatia. May 2003

[WAPForum]: Open Mobile Alliance.
http://www.wapforum.org/

[Wong99]: J. W. K. Wong, and V. C. M. Leung.
Improving end-to-end performance of TCP using link-layer retransmissions over mobile internetworks.
Proceedings of the IEEE International Conference on Communications.
Pages 324-328. June 1999.

Vita

Kiran K. Komaravolu was born in Nellore, India, in 1979. In June 2001 he obtained a baccalaureate degree in computer science and engineering from Osmania University in Hyderabad. He entered the graduate program at Lehigh University, Bethlehem, PA in the Fall of 2001. He held assistantships with the Department of Physics (Aug 2001 - Dec 2001) and with the Department of Computer Science and Engineering (Jan 2002 - Aug 2003).

In the spring of 2001 he joined Prof. Brian Davison's Web Understanding, Modeling and Evaluation Group. This group nurtured his interest in computer networks, particularly in transport and network layer issues, QOS, and multimedia networks.

From Jan 2002 through Aug 2003 his research focused on protocols for resource constrained networks. He completed his Masters program in Computer Science in December 2003, under the technical guidance of Prof. Brian Davison. He has also published a technical report:

> *A Split Stack Approach to Mobility-Providing Performance-Enhancing Proxies* (with Brian D. Davison and Baoning Wu). Technical Report LU-CSE-02-012, Dept. of Computer Science and Engineering, Lehigh University, November 2002.

END OF TITLE