IPCoreL

An Intuitive Packet Injection Programming Language

Programming Language Proprosal
By
Phillip Duane Douglas, Jr.

Introduction

IPCoreL is a programming language that will be geared towards newcomers to the world of computer networking. Many languages already exist that have had libraries appended to the architecture after their inception. While these languages are more than capable of providing tools to create custom network packets, or datagrams, the method in which this task is done can be daunting and difficult. The goal of IPCoreL is to provide novices, "noobs," the field with a networking tool capable of providing the same functionality as proven programming languages, yet doing so in a less time-consuming and hair-pulling manner.

The concept of IPCoreL is simple, provide a small, intuitive programming language that will assists beginners in network programming. The construction of datagrams in IPCoreL will encompass two of the layers within the OSI model. These layers are the Network (Layer 3) and Transport (Layer 4). IPCoreL will provide methods to create fields of an IPv4 and/or IPv6 packet that will include an IP header, a TCP/UDP header, and a field for the payload of the packet containing information described by the programmer, the data. Figure 1 illustrates the IPv4 and IPv6 header structure, while Figure 2 illustrates the header structures of TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) headers.

In addition to creating packets, IPCoreL will provide functionality to calculate the various measurements involved with networking, i.e. throughput, delay, jitter, etc., based on the number of packets desired for transmission, the size of the packets, number of hops and latency of the conceptual network. The intention of IPCoreL is for users of the programming language to learn the innards of IP packets and networking while creating and performing calculations on the generated packets. A method for transmission of generated packets will be included in IPCorel to allow programmers to view the traversal of their packets through their network via packet capture programs such as Wireshark.

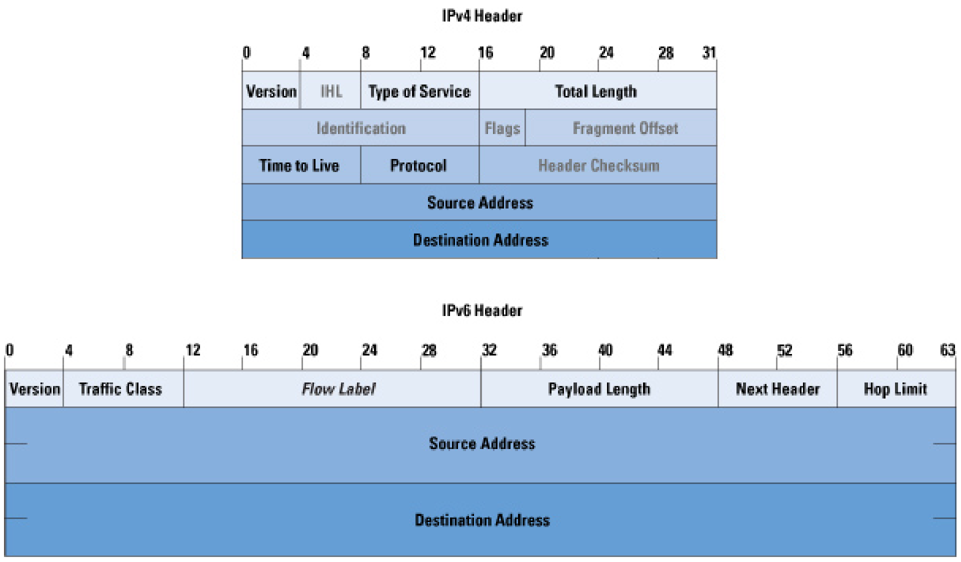
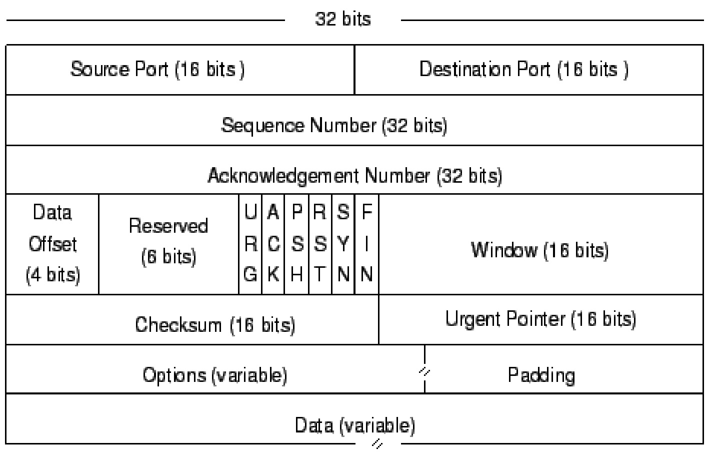
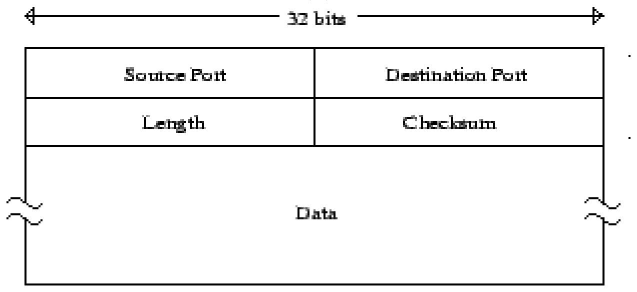


Figure 1. IPv4 and IPv6 Header Structure



(a)



(b)

Figure 2. TCP(a) and UDP(b) Header Structures

<u>IPCoreL Example Program</u>

The program below is a small representation of an IPCoreL program when creating an IPv4 and

IPv6 header, then performing throughput calculations on a stream of packets and then transmits them across a network.

```
Header ip = ip4hdr();      //creates an empty IPv4 header
int vers, tclass, flabel, pload, nexthdr,hop, latency, win_size, thrput, num_pkts;
string srcaddr = “192.168.10.2”;
string dstaddr = “192.168.10.3”;
vers = 6;
tclass = 0;
flabel = 0;
pload = 8;
nexthdr = 0;
latency = 15;
win_size = 1280;
thrput = ( win_size * 8 ) / latency;
Header ip6 =  ip6hdr(vers, tclass, flabel, pload, nexthdr);
Packet ip4_pkt = pack(ip4);                  //creates an empty IPv4 packet
Packet ip6_pkt = pack(ip6, “Test”);          //creates an IPv6 packet with “Test” being
                                             representing //the data of the packet
sock(ip4_pkt, num_pkts);                     //transmits a user-defined amount of IPv4 packets
sock(ip6_pkt, num_pkts);                     //transmits a user-defined amount of IPv6 packets
```

<u>IPCoreL Syntax</u>

type -> int | string | float | Header | Packet | bitval
stmt -> expr

|

function -> **ip4hdr** (*expr* (single or multiple)) ; {*stmt.n* = new *IPv4hdr*(*expr.n*)}

| **ip6hdr** (*expr* (single or multiple)); { *stmt.n* = new *IPv4hdr*(*expr.n*)}

| **udphdr** (*expr* (single or multiple)); { *stmt.n* = new *UDPhdr(expr.n*)}

| **tcphdr** (*expr* (single or multiple)); { *stmt.n* = new TCPhdr(*expr.n*)}

| **pack** (*function, function, function, function, expr*); { *stmt.n = new Pack*(*function.n, function.n, function.n, function.n, expr.n*)}

| **sock** (*function)* { *stmt.n = new Sock*(*function.n)*}

expr -> *expr + factor*

| *expr – factor*

| *expr * factor*

| *expr / factor*

| *factor*

factor -> (expr)

| *digit*

| *letter*

| *punctuation*

|

digit -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

letter -> ‘_’ | ‘a’ | ‘A’ | ‘b’ | ‘B’ | ’c’ | ‘C’ | ‘d’ | ‘D’ | ‘e’ | ‘E’ | ‘f’ | ‘F’ | ‘g’ | ’G’ | ‘h’ | ‘H’ | ‘i’ | ‘I’ | ‘j’ | ‘J’ | ‘k’ | ‘K’ | ‘l’ | ‘L’ | ‘m’ | ‘M’ | ‘n’ | ‘N’ | ‘o’ | ‘O’ | ‘p’ | ‘P’ | ‘q’ | ‘Q’ | ‘r’ | ‘R’ | ‘s’ | ‘S’ | ‘t’ | ‘T’ | ‘u’ | ‘U’ | ‘v’ | ‘V’ | ’w’ | ‘W’ | ‘x’ | ‘X’ | ‘y’ | ‘Y’ | ‘z’ | ‘Z’

punctuation -> ‘;’ | ‘(‘ | ‘)’ | ‘,’

<u>Meanings</u>

expr: consists of numerical operations that can be performed
bitval: will provide bit-level representation type
Header: represents header type for declarative statements
Packet: represents packet type for declarative statements

functions: consist of procedure that creates headers, packets, and sockets for transmission of packets

- **ip4hdr** – a function that creates an IPv4 header with a single parameter or multiple parameters describing the fields of the header established by the programmer when the function is called.
- **ip6hdr** – a function that creates an IPv6 header with a single parameter or multiple parameters describing the fields of the header established by the programmer when the function is called.
- **udphdr** - a function that creates a UDP header with a single parameter or multiple parameters describing the fields of the header established by the programmer when the function is called.
- **tcphdr** – a function that creates a TCP header with a single parameter or multiple parameters describing the fields of the header established by the programmer when the function is called
- **pack** – a function that creates a IP packet with the headers created by the programmer when the function is called

- **sock –** a function that opens a socket and establishes the type of transmission that will take place over the socket. Programmer provides the packet created as a parameter for the function when calling it.