Installing NetPacket::*

```
gunzip NetPacket-0.03.tar.gz
tar xvf NetPacket-0.03.tar
cd NetPacket-0.03
perl Makefile.PL
make
make test
su
make install
<ctrl-D>
man NetPacket::Ethernet
perl -e 'use NetPacket::Ethernet'
```

Installing Net::Pcap

```
gunzip Net-Pcap-0.04.tar.gz
tar xvf Net-Pcap-0.04.tar
cd Net-Pcap-0.04
perl Makefile.PL
make
su
make test
make install
<ctrl-D>
man Net::Pcap
perl -e 'use Net::Pcap'
```

Installing Net::PcapUtils

```
gunzip Net-PcapUtils-0.01.tar.gz
tar xvf Net-PcapUtils-0.01.tar
cd Net-PcapUtils-0.01
perl Makefile.PL
make
su
make test
make install
<ctrl-D>
man Net::PcapUtils
perl -e 'use Net::PcapUtils'
```

Configuring Your Network Interface

```
su
/sbin/ifconfig eth0 promisc
<ctrl-D>

su
/sbin/ifconfig eth0 -promisc
<ctrl-D>
```

Building Low-Level Snooping Tools

```
#! /usr/bin/perl -w

use strict;
use Net::PcapUtils;

sub got_a_packet {
    print "Got a packet!\n";
}

my $status = Net::PcapUtils::loop( \&got_a_packet );

if ( $status )
{
    print "Net::PcapUtils::loop returned: $status\n";
}
```

loop = open + next

```
#! /usr/bin/perl -w

use strict;
use Net::PcapUtils;

sub got_a_packet {
    print "Got a packet!\n";
}

my $pkt_descriptor = Net::PcapUtils::open;

if ( !ref( $pkt_descriptor ) )
{
    print "Net::PcapUtils::open returned: $pkt_descriptor\n";
    exit;
}

while( 1 )    # i.e., forever, or until "killed" ...
{
    Net::PcapUtils::next( $pkt_descriptor );
    got_a_packet;
}
```

loop = open + next

```
my ( $packet, %header ) =
        Net::PcapUtils::next( $pkt_descriptor );
```

Optional Parameters: `loop` and `open`

```
Net::PcapUtils::loop(
    \&got_a_packet,
    SNAPLEN => 100,
    PROMISC => 1,
    TIMEOUT => 1000,
    NUMPACKETS => -1,
    FILTER => '',
    USERDATA => '',
    SAVEFILE => '',
    DEV => ''
);

Net::PcapUtils::loop(
    \&got_a_packet,
    NUMPACKETS => 1000,
    FILTER => 'ip'
);
```

Optional Parameters: The Callback Function

```
use NetPacket::Ethernet qw( :types );

    ...

sub got_a_packet {
    my ( $user_arg, $header, $packet ) = @_;

    print "Got a packet!\n\n" ;
    print "The user argument is: ", $user_arg, "\n";
    print "The header data is:\n";
    foreach my $name (sort keys %{$header})
    {
        print "    $name -> ${header}{$name}\n";
    }
    print "The packet data is:   ", $packet, "\n\n";
}
```

Ethernet Analysis

```
package XtraType;

#
# XtraType.pm - Some additional Ethernet frame types.
#

use 5.6.0;

require Exporter;

our @ISA          = qw( Exporter );

# We export all the symbols declared in this module by
# default.
our @EXPORT       = qw(
                        ETH_TYPE_NOVELL1
                        ETH_TYPE_NOVELL2
                        ETH_TYPE_REVERSE_ARP
                        ETH_TYPE_TCP_IP_COMPRESSION
                      );
```

Ethernet Analysis

```
our @EXPORT_OK    = qw(
                      );

our %EXPORT_TAGS = (
                    );

our $VERSION      = 0.01;

use constant ETH_TYPE_NOVELL1               => 0x8137;
use constant ETH_TYPE_NOVELL2               => 0x8138;
use constant ETH_TYPE_REVERSE_ARP           => 0x8035;
use constant ETH_TYPE_TCP_IP_COMPRESSION    => 0x876B;

1;
```

EtherSnooper (v0.01)

```
#! /usr/bin/perl -w

use 5.6.0;

use strict;
use Net::PcapUtils;
use NetPacket::Ethernet qw( :types );
use XtraType;
```

EtherSnooper (v0.01)

```
our %type_totals = ();

our %type_desc = (
        0x0800 => 'IPv4',
        0x0806 => 'ARP',
        0x809B => 'AppleTalk',
        0x814C => 'SNMP',
        0x86DD => 'IPv6',
        0x880B => 'PPP',
        0x8137 => 'NOVELL1',
        0x8138 => 'NOVELL2',
        0x8035 => 'RARP',
        0x876B => 'TCP/IPc'
    );

our $num_packets = 0;
```

EtherSnooper (v0.01)

```
sub got_a_packet {
    my ( $user_arg, $header, $packet ) = @_;

    my $frame = NetPacket::Ethernet->decode( $packet );

    $type_totals{ $frame->{type} }++;

    $num_packets++;
}
```

EtherSnooper (v0.01)

```
my $status = Net::PcapUtils::loop(
                \&got_a_packet,
                NUMPACKETS => 1000
             );

if ( $status )
{
    print "Net::PcapUtils::loop returned: $status\n";
}
else
{
    display_results;
}
```

EtherSnooper (v0.01)

```
sub display_results {
    print "$num_packets frames processed.\n\n";

    foreach my $etype ( sort keys %type_desc )
    {
        print "$type_desc{$etype} generated ";
        if ( exists $type_totals{$etype} )
        {
            print "$type_totals{$etype} packets.\n";
        }
        else
        {
            print "no packets.\n";
        }
    }
}
```

EtherSnooper (v0.01)

```
print "\n\nRaw statistics:\n\n";
print "frame-type -> frequency\n\n";
foreach my $e_total ( sort keys %type_totals )
{
    printf "%lx -> %d\n", $e_total, $type_totals{$e_total};
}
```

EtherSnooper (v0.01)

```
if ($frame->{type} < 1501)
{
    $type_totals{ 1500 }++;
}
else
{
    $type_totals{ $frame->{type} }++;
}

print "\nNon Ethernet II (DIX) frames generated";
print " $type_totals{1500} packets.\n";
```

EtherSnooper (v0.02)

```
my $minute = 3;
my $now = time;
my $then = $now + (60 * $minute);

while ( ($now = time) < $then )
{
    ; # Do whatever you want to do here.
}
```

EtherSnooper (v0.02)

```
my $minute = 3;
my $now = time;
my $then = $now + (60 * $minute);

while ( ($now = time) < $then )
{
    my $status = Net::PcapUtils::loop(
                    \&got_a_packet,
                    NUMPACKETS => 1
                );
}
```

EtherSnooper (v0.02)

```
#! /usr/bin/perl -w

use 5.6.0;
use strict;
use Net::PcapUtils;
use NetPacket::Ethernet qw( :types );
use XtraType;

our %type_totals = ();
our %type_desc = (
        0x0800 => 'IPv4',
        0x0806 => 'ARP',
        0x809B => 'AppleTalk',
        0x814C => 'SNMP',
        0x86DD => 'IPv6',
        0x880B => 'PPP',
        0x8137 => 'NOVELL1',
        0x8138 => 'NOVELL2',
        0x8035 => 'RARP',
        0x876B => 'TCP/IPc'
  );
our $num_packets = 0;
```

EtherSnooper (v0.02)

```
sub got_a_packet {
    my $packet = shift;

    my $frame = NetPacket::Ethernet->decode( $packet );

    if ( $frame->{type} < 1501 )
    {
        $type_totals{ 1500 }++;
    }
    else
    {
        $type_totals{ $frame->{type} }++;
    }

    $num_packets++;
}
```

EtherSnooper (v0.02)

```
sub display_results {
    print "$num_packets processed.\n\n";

    foreach my $etype ( sort keys %type_desc )
    {
        print "$type_desc{$etype} generated ";
        if ( exists $type_totals{$etype} )
        {
            print "$type_totals{$etype} packets.\n";
        }
        else
        {
            print "no packets.\n";
        }
    }
    print "\nNon Ethernet II (DIX) frames generated";
    print " $type_totals{1500} packets.\n";
}

my $pkt_descriptor = Net::PcapUtils::open;
```

EtherSnooper (v0.02)

```
if ( !ref( $pkt_descriptor ) )
{
    print "Net::PcapUtils::open returned: $pkt_descriptor\n";
    exit;
}

my $minute = 3;  my $now = time;
my $then = $now + (60 * $minute);
my ( $next_packet, %next_header );

while ( ($now = time) < $then )
{
    ( $next_packet, %next_header ) =
       Net::PcapUtils::next( $pkt_descriptor );
    got_a_packet( $next_packet );
}
display_results;
```

EtherSnooper (v0.03)

```
our %src_hosts = (); our %dest_hosts = ();
    ...
sub got_a_packet {
    my $packet = shift;

    my $frame = NetPacket::Ethernet->decode( $packet );

    if ( $frame->{type} < 1501 )
    {
        $type_totals{ 1500 }++;
    }
    else
    {
        $type_totals{ $frame->{type} }++;
    }

    $src_hosts{ $frame->{src_mac} }++;
    $dest_hosts{ $frame->{dest_mac} }++;
    $num_packets++;
}
```

EtherSnooper (v0.03)

```
sub display_results {
    print "$num_packets frames processed.\n\n";

    my $busiest_mac = 0;
    my $busiest_count = 0;

    print "The busiest hosts were:\n\n";

    while ( my ( $host, $count ) = each %src_hosts )
    {
        if ( $count > $busiest_count )
        {
            $busiest_mac = $host;
            $busiest_count = $count;
        }
    }
```

EtherSnooper (v0.03)

```
    print "Source: $busiest_mac with ";
    print "$busiest_count frames\n";

    $busiest_mac = 0;
    $busiest_count = 0;

    while ( my ( $host, $count ) = each %dest_hosts )
    {
        if ( $count > $busiest_count )
        {
            $busiest_mac = $host;
            $busiest_count = $count;
        }
    }

    print "Destination: $busiest_mac with ";
    print "$busiest_count frames\n";
}
```

EtherSnooper (v0.03)

```
sub display_results {
    print "$num_packets frames processed.\n\n";

    foreach my $etype ( sort keys %type_desc )
    {
        print "$type_desc{$etype} generated ";
        if ( exists $type_totals{$etype} )
        {
            print "$type_totals{$etype} packets.\n";
        }
        else
        {
            print "no packets.\n";
        }
    }
```

EtherSnooper (v0.03)

```
    print "The host statistics are:\n\nSources:\n\n";

    foreach my $host (sort keys %src_hosts )
    {
        print "Host: $host, Count: $src_hosts{$host}.\n";
    }

    print "\nDestinations:\n\n";

    foreach my $host (sort keys %dest_hosts )
    {
        print "Host: $host, Count: $dest_hosts{$host}.\n";
    }
}
```

Displaying IP Addresses

```
our %e2ip = ();

    ...

sub got_a_packet {
    my $packet = shift;

    my $frame = NetPacket::Ethernet->decode( $packet );

    if ($frame->{type} < 1501 )
    {
        $type_totals{ 1500 }++;
    }
    else
    {
        $type_totals{ $frame->{type} }++;
    }

    $src_hosts{ $frame->{src_mac} }++;
    $dest_hosts{ $frame->{dest_mac} }++;
```

Displaying IP Addresses

```
    if ( $frame->{type} == NetPacket::Ethernet::ETH_TYPE_IP )
    {
        my $ip_datagram = NetPacket::IP->decode(
            NetPacket::Ethernet::eth_strip( $packet ) );

        $e2ip{ $frame->{src_mac} }  = $ip_datagram->{src_ip};
        $e2ip{ $frame->{dest_mac} } = $ip_datagram->{dest_ip};
    }

    $num_packets++;
}
```

Displaying IP Addresses

```
use NetPacket::Ethernet qw( :types :strip );
use NetPacket::IP;

    ...

sub display_results {
    print "$num_packets frames processed.\n\n";

    foreach my $etype ( sort keys %type_desc )
    {
        print "$type_desc{$etype} generated ";
        if ( exists $type_totals{$etype} )
        {
            print "$type_totals{$etype} packets.\n"
        }
        else
        {
            print "no packets.\n";
        }
    }
```

Displaying IP Addresses

```
print "\nNon Ethernet II (DIX) frames generated";
print " $type_totals{1500} packets.\n";

print "\nThe host statistics are:\n\nSources:\n\n";

foreach my $host (sort keys %src_hosts )
{
    if ( exists $e2ip{$host} )
    {
        print "Host: $host ($e2ip{$host}), ";
        print "Count: $src_hosts{$host}.\n";
    }
    else
    {
        print "Host: $host, Count: $src_hosts{$host}.\n";
    }
}
```

Displaying IP Addresses

```
    print "\nDestinations:\n\n";

    foreach my $host (sort keys %dest_hosts )
    {
        if ( exists $e2ip{$host} )
        {
            print "Host: $host ($e2ip{$host}), ";
            print "Count: $dest_hosts{$host}.\n";
        }
        else
        {
            print "Host: $host, Count: $dest_hosts{$host}.\n";
        }
    }
}
```

Displaying IP Addresses

```
my $pkt_descriptor = Net::PcapUtils::open;

    ...

my $pkt_descriptor =
    Net::PcapUtils::open( FILTER => 'ip' );
```

EtherSnooper (v0.05)

```
#! /usr/bin/perl -w

use 5.6.0;

use integer;
use strict;

use Net::PcapUtils;
use NetPacket::Ethernet qw( :strip );
use NetPacket::IP;

our %ttl_totals = ();

our $num_datagrams = 0;
```

EtherSnooper (v0.05)

```
sub got_a_packet {
    my $packet = shift;

    my $ip_datagram = NetPacket::IP->decode(
       NetPacket::Ethernet::eth_strip( $packet ) );

    $ttl_totals{ $ip_datagram->{ttl} }++;

    $num_datagrams++;
}
```

EtherSnooper (v0.05)

```
sub display_results {
    print "$num_datagrams datagrams processed.\n\n";

    my $min_ttl = 256;
    my $max_ttl = 0;
    my $average_ttl = 0;

    while ( my ( $ttl_key, $ttl_value ) = each %ttl_totals )
    {
        if ( $ttl_key < $min_ttl )
        {
            $min_ttl = $ttl_key;
        }
        if ( $ttl_key > $max_ttl )
        {
            $max_ttl = $ttl_key;
        }
        $average_ttl =
            $average_ttl + ( $ttl_key * $ttl_value );
    }
```

EtherSnooper (v0.05)

```
    $average_ttl = ( $average_ttl / $num_datagrams );

    print "Minimum TTL value: $min_ttl\n";
    print "Maximum TTL value: $max_ttl\n";
    print "Average TTL value: $average_ttl\n\n";

    print "TTL distribution analysis:\n\n";

    foreach my $ttlkey ( sort {$a <=> $b} keys %ttl_totals )
    {
        print "TTL: $ttlkey, ";
        print "frequency: $ttl_totals{$ttlkey}.\n";
    }
}
```

EtherSnooper (v0.05)

```
my $pkt_descriptor =
        Net::PcapUtils::open( FILTER => 'ip' );

if ( !ref( $pkt_descriptor ) )
{
    print "Net::PcapUtils::open returned: $pkt_descriptor\n";
    exit;
}

my $minute = 3;  my $now = time;
my $then = $now + (60 * $minute);

my ( $next_packet, %next_header );

while ( ($now = time) < $then )
{
    ( $next_packet, %next_header ) =
           Net::PcapUtils::next( $pkt_descriptor );
    got_a_packet( $next_packet );
}

display_results;
```

EtherSnooper (v0.06)

```
#! /usr/bin/perl -w

use 5.6.0;

use strict;
use Net::PcapUtils;
use NetPacket::Ethernet qw( :strip );
use NetPacket::IP qw( :protos ) ;

our %ip_type_totals = ();

our %ip_type_desc = {
        0   => 'IP',
        1   => 'ICMP',
        2   => 'IGMP',
        4   => 'IP/IP',
        6   => 'TCP',
        17  => 'UDP'
    };

our $num_datagrams = 0;
```

EtherSnooper (v0.06)

```
sub got_a_packet {
    my $packet = shift;

    my $ip_datagram = NetPacket::IP->decode(
        NetPacket::Ethernet::eth_strip( $packet ) );

    $ip_type_totals{ $ip_datagram->{proto} }++;

    $num_datagrams++;
}
```

EtherSnooper (v0.06)

```
sub display_results {
    print "$num_datagrams processed.\n\n";

    foreach my $iptype ( sort keys %ip_type_desc )
    {
        print "$ip_type_desc{$iptype} generated ";
        if ( exists $ip_type_totals{$iptype} )
        {
            print "$ip_type_totals{$iptype} datagrams.\n";
        }
        else
        {
            print "no datagrams.\n";
        }
    }
}

my $pkt_descriptor =
        Net::PcapUtils::open( FILTER => 'ip' );
```

EtherSnooper (v0.06)

```
if ( !ref( $pkt_descriptor ) )
{
    print "Net::PcapUtils::open returned: $pkt_descriptor\n";
    exit;
}

my $minute = 3;  my $now = time;
my $then = $now + (60 * $minute);
my ( $next_packet, %next_header );

while ( ($now = time) < $then )
{
    ( $next_packet, %next_header ) =
           Net::PcapUtils::next( $pkt_descriptor );
    got_a_packet( $next_packet );
}
display_results;
```

Preparing to Snoop UDP

```
#! /usr/bin/perl -w

use 5.6.0;

use strict;
use Net::PcapUtils;
use NetPacket::Ethernet qw( :strip );
use NetPacket::IP qw( :strip ) ;
use NetPacket::UDP;
```

Preparing to Snoop UDP

```
my $udp_datagram = NetPacket::UDP->decode(
    NetPacket::IP::ip_strip(
        NetPacket::Ethernet::eth_strip( $packet ) ) );

    ...

my $pkt_descriptor = Net::PcapUtils::open(
                         FILTER => 'udp',
                         SNAPLEN => 1500
                     );
```

Preparing to Snoop TCP

```
#! /usr/bin/perl -w

use 5.6.0;

use strict;
use Net::PcapUtils;
use NetPacket::Ethernet qw( :strip );
use NetPacket::IP qw( :strip ) ;
use NetPacket::TCP;
```

Preparing to Snoop TCP

```
my $tcp_segment = NetPacket::TCP->decode(
    NetPacket::IP::ip_strip(
        NetPacket::Ethernet::eth_strip( $packet ) ) );

    ...

my $pkt_descriptor = Net::PcapUtils::open(
                         FILTER => 'tcp',
                         SNAPLEN => 1500
                     );
```

The TCP and UDP Gotcha!

```
sub got_a_packet {
    my $packet = shift;

    my $ip_datagram = NetPacket::IP->decode(
        NetPacket::Ethernet::eth_strip( $packet ) );

    $num_datagrams++;

    if ($ip_datagram->{foffset} > 0)
    {
        $num_fragments++;
        exit;
    }

    $ip_type_totals{ $ip_datagram->{proto} }++;
}
```

Application Traffic Monitoring

```
$udp_datagram->{src_port}
$udp_datagram->{dest_port}

    ...

$tcp_segment->{src_port}
$tcp_segment->{dest_port}
```

Application Traffic Monitoring

```
        ...
To the extent possible, these same port assignments
are used with the UDP [RFC768].

The range for assigned ports managed by the IANA is 0-1023.
        ...
vettcp          78/tcp    vettcp
vettcp          78/udp    vettcp
#               Christopher Leong <leong@kolmod.mlo.dec.com>
finger          79/tcp    Finger
finger          79/udp    Finger
#               David Zimmerman <dpz@RUTGERS.EDU>
http            80/tcp    World Wide Web HTTP
http            80/udp    World Wide Web HTTP
www             80/tcp    World Wide Web HTTP
www             80/udp    World Wide Web HTTP
www-http        80/tcp    World Wide Web HTTP
www-http        80/udp    World Wide Web HTTP
#               Tim Berners-Lee <timbl@W3.org>
hosts2-ns       81/tcp    HOSTS2 Name Server
hosts2-ns       81/udp    HOSTS2 Name Server
        ...
```

Application Traffic Monitoring

```
#! /usr/bin/perl

# Process the IANA 'port-numbers' file so that we just
# have the well-known ports (i.e., < 1024) for the TCP
# and UDP protocols.  Two new files are created:
#
#     well-known-udp  - well-known ports for TCP.
#     well-known-tcp  - well-known ports for UDP.

open PORTNUMS, 'port-numbers' or die "Ooops: $!";

open PROCESSED_TCP, '>well-known-tcp' or die "Ooops: $!";
open PROCESSED_UDP, '>well-known-udp' or die "Ooops: $!";
```

Application Traffic Monitoring

```
while ($_ = <PORTNUMS>)
{
    next if /Unassigned/;
    next if /Reserved/;
    next if /^#/;

    next unless m[/(udp|tcp)];

    m[(\d+)/(udp|tcp)];

    print PROCESSED_TCP $_ if ($1 < 1024) and m[/tcp] ;
    print PROCESSED_UDP $_ if ($1 < 1024) and m[/udp];
}

close PROCESSED_TCP;
close PROCESSED_UDP;

close PORTNUMS;
```

Application Traffic Monitoring

```
my %tcp_desc = ();

open WELLKNOWN_TCP, 'well-known-tcp' or die "Ooops: $!";

while ($_ = <WELLKNOWN_TCP>)
{
    chomp;

    m[(\d+)/tcp];

    $tcp_desc{$1} = $_;
}

close WELLKNOWN_TCP;
```

Application Traffic Monitoring

```
print "$tcp_desc{69}\n";
print "$tcp_desc{80}\n";
print "$tcp_desc{110}\n";
print "$tcp_desc{434}\n";

    ...

tftp             69/tcp    Trivial File Transfer
www-http         80/tcp    World Wide Web HTTP
pop3            110/tcp    Post Office Protocol - Version 3
mobileip-agent  434/tcp    MobileIP-Agent
```

EtherSnooper (v0.07)

```
#! /usr/bin/perl -w

use 5.6.0;

use strict;
use Net::PcapUtils;
use NetPacket::Ethernet qw( :strip );
use NetPacket::IP qw( :strip ) ;
use NetPacket::TCP;

our %tcp_proto_totals = ();

our $num_datagrams = 0;
```

EtherSnooper (v0.07)

```
sub got_a_packet {
    my $packet = shift;

    my $tcp_segment = NetPacket::TCP->decode(
        NetPacket::IP::ip_strip(
            NetPacket::Ethernet::eth_strip( $packet ) ) );

    if ($tcp_segment->{src_port} < 1024)
    {
        $tcp_proto_totals{ $tcp_segment->{src_port} }++;
    }
    if ($tcp_segment->{dest_port} < 1024)
    {
        $tcp_proto_totals{ $tcp_segment->{dest_port} }++;
    }

    $num_datagrams++;
}
```

EtherSnooper (v0.07)

```
sub display_results {
    print "$num_datagrams segments processed.\n\n";

    my %tcp_desc = ();

    open WELLKNOWN_TCP, 'well-known-tcp' or die "Ooops: $!";

    while ($_ = <WELLKNOWN_TCP>)
    {
        chomp;

        m[(\d+)/tcp];

        $tcp_desc{$1} = $_;
    }

    close WELLKNOWN_TCP;
```

EtherSnooper (v0.07)

```
    foreach my $port (sort keys %tcp_proto_totals)
    {
        print "Port: $port, ";
        print "generated $tcp_proto_totals{$port} segments.\n";
        print "Protocol-port: $tcp_desc{$port}.";
        print "\n\n";
    }
}
```

EtherSnooper (v0.07)

```
my $pkt_descriptor = Net::PcapUtils::open(
                         FILTER => 'tcp',
                         SNAPLEN => 120
                     );

if ( !ref( $pkt_descriptor ) )
{
    print "Net::PcapUtils::open returned: $pkt_descriptor\n";
    exit;
}

my $minute = 3;
my $now = time;
my $then = $now + (60 * $minute);
```

EtherSnooper (v0.07)

```
my ( $next_packet, %next_header );

while ( ($now = time) < $then )
{
    ( $next_packet, %next_header ) =
       Net::PcapUtils::next( $pkt_descriptor );
    got_a_packet( $next_packet );
}

display_results;
```

EtherSnooper (v0.07)

```
my $filter_address = '149.153.100.23';
    ...
got_a_packet( $filter_address, $next_packet );
    ...
sub got_a_packet {
    my ($ip_addr, $packet) = @_;

    my $ip_datagram = NetPacket::IP->decode(
        NetPacket::Ethernet::eth_strip( $packet ) );

    if ( ($ip_datagram->{src_ip} eq $ip_addr) or
          ($ip_datagram->{dest_ip} eq $ip_addr)
    {
        my $tcp_segment =
            NetPacket::TCP->decode( $ip_datagram->{data} );

        $tcp_proto_totals{ $tcp_segment->{src_port} }++;
        $tcp_proto_totals{ $tcp_segment->{dest_port} }++;

        $num_datagrams++;
    }
}
```