Wireshark
(aka Ethereal)

Aamer Akhter / aa@cisco.com

ECMD, cisco Systems

What is Wireshark

- Free Open Source Network Protocol Analyzer
- Multi-platform: Runs on Windows, Linux, Solaris, NetBSD, FreeBSD
- CLI as well as Graphical display
- 100’s of protocols supported

Acknowledgements

- Gerald Combs, creator, lead developer – 1998
- Guy Harris
- Gilbert Ramirez
- Many, many contributors

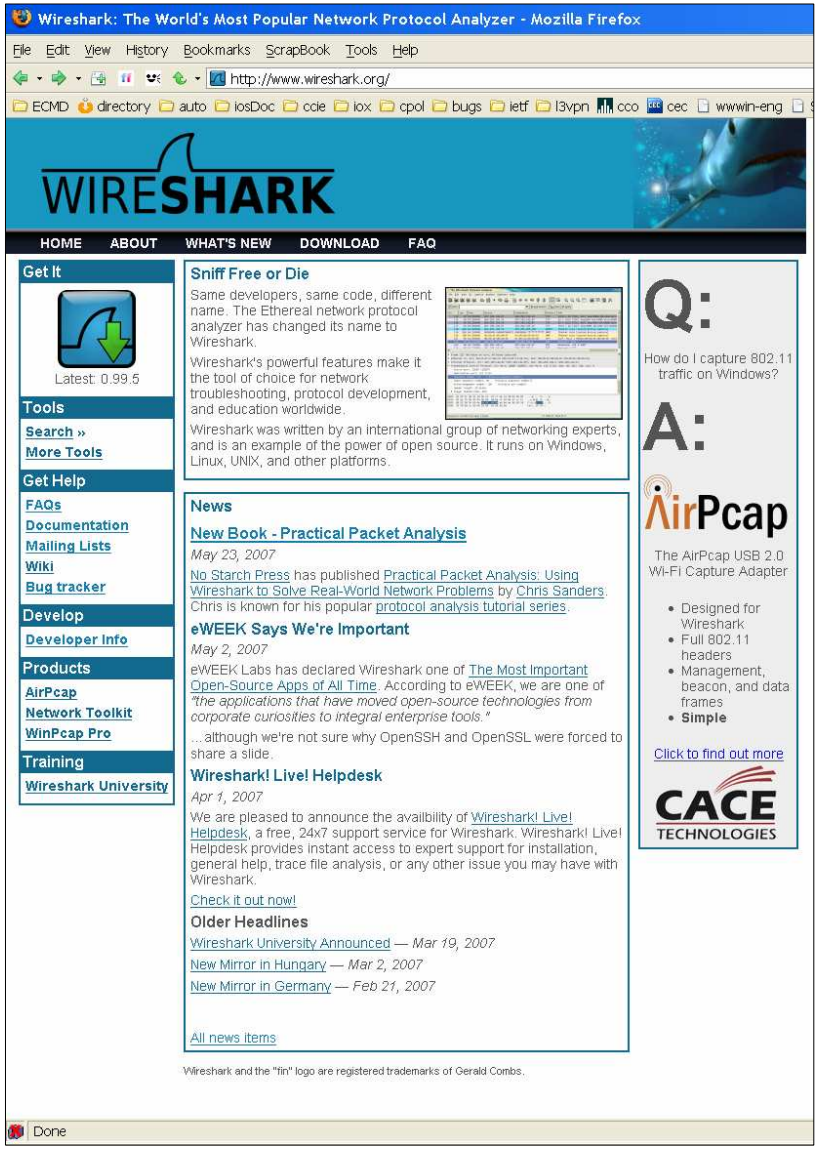
  http://anonsvn.wireshark.org/wireshark/trunk/AUTHORS
- libpcap folks
- Winpcap folks
- CACE Technologies

How is Wireshark Used today?

- Troubleshooting
- Performance issues
- Security Analysis
- Protocol Learning Tool
- Protocol Development

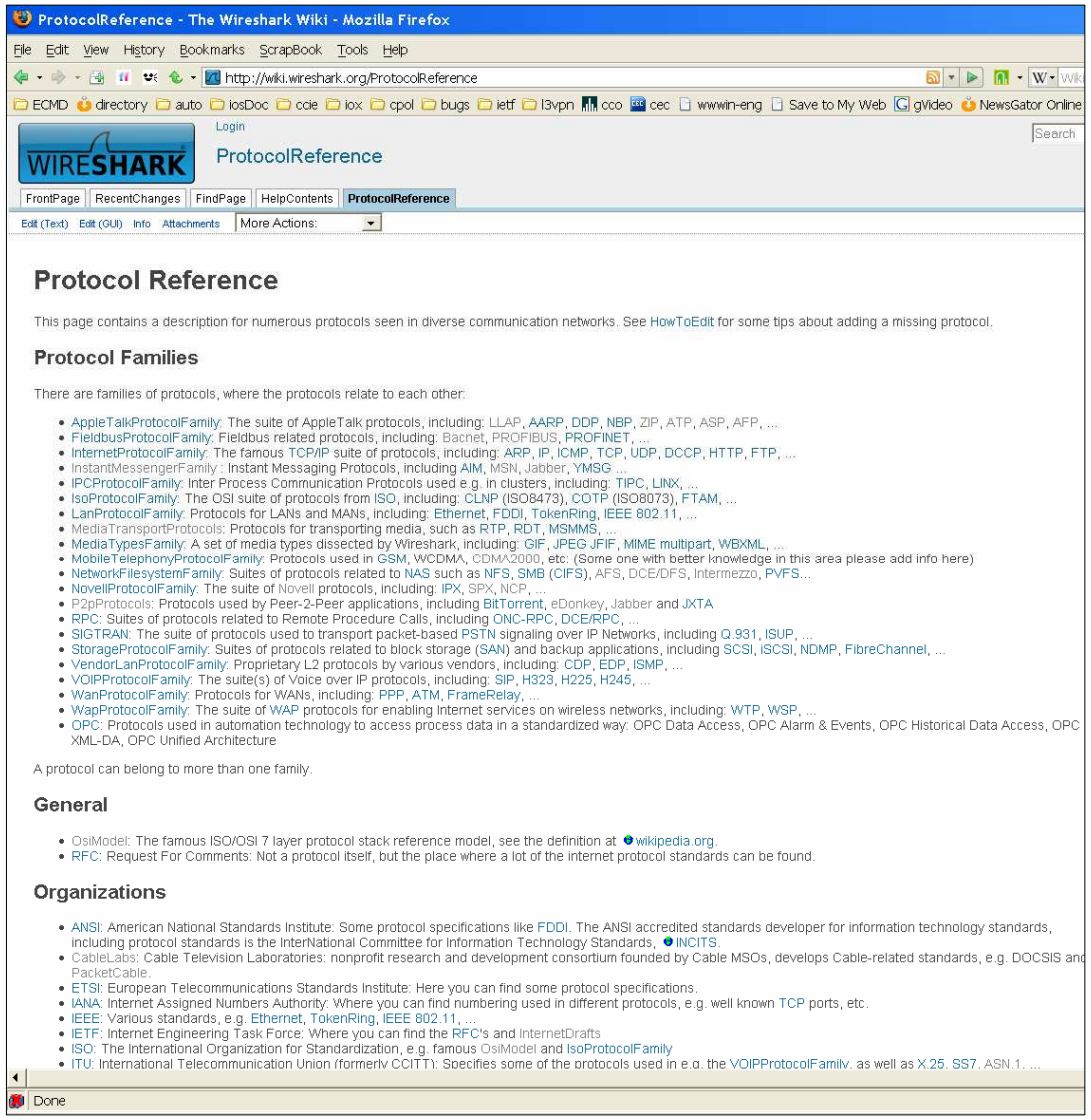
Wireshark Website

- http://www.wireshark.org
- Formerly ethereal.com
- Source tarball
- SVN repository
- Multi-platform compiled sources
- Documentation



Wireshark Wiki

- http://wiki.wireshark.org/
- Protocol reference
- Discussion on various network protocols and their function operation
- Growing sample pcap library



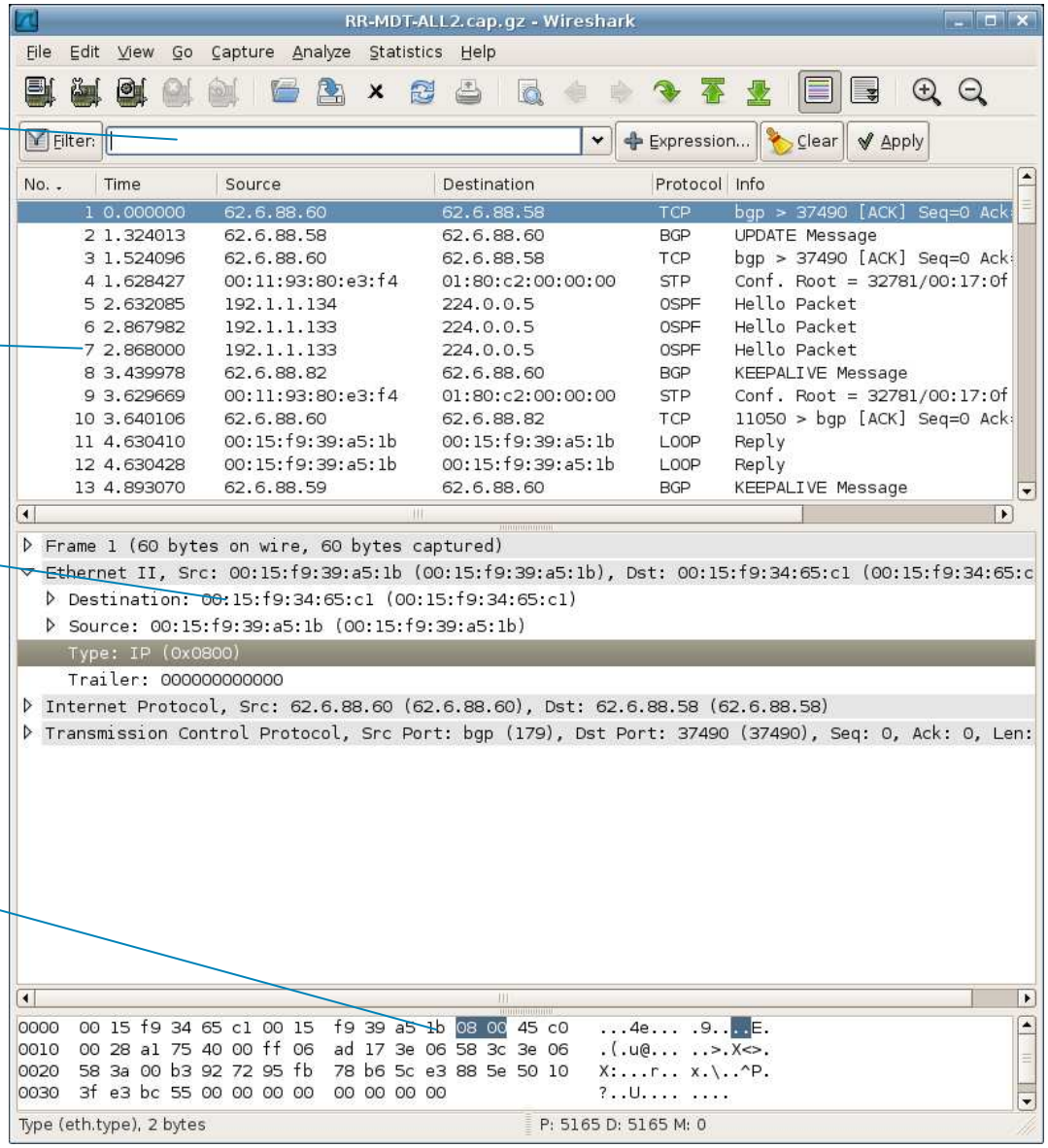
Basic Components

Captured Frame Filter Spec

Frames that match Filter Spec

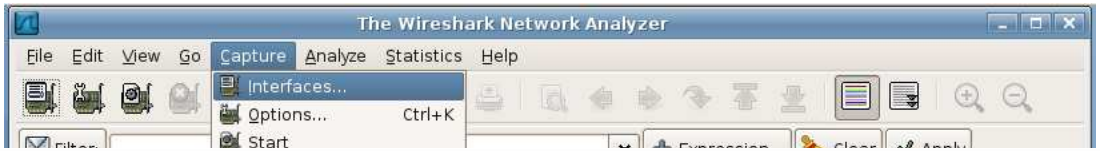
Protocol Dissection of selected frame

Hex view of frame highlighted, selection from protocol dissection is a also highlighted



Acquiring Packets (capturing)

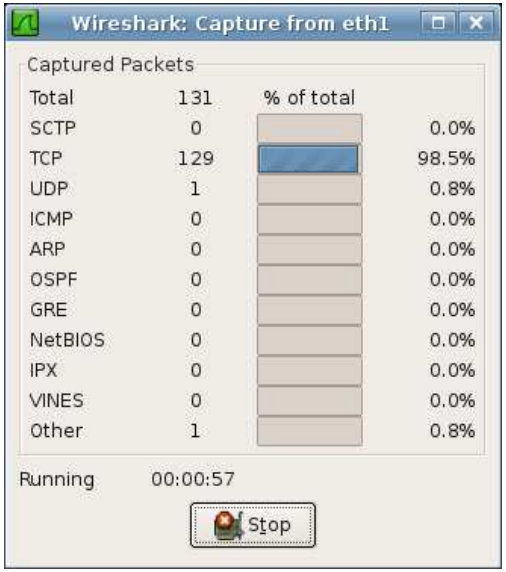
Select
Capture->Interfaces…



Pick which Interface to Capture

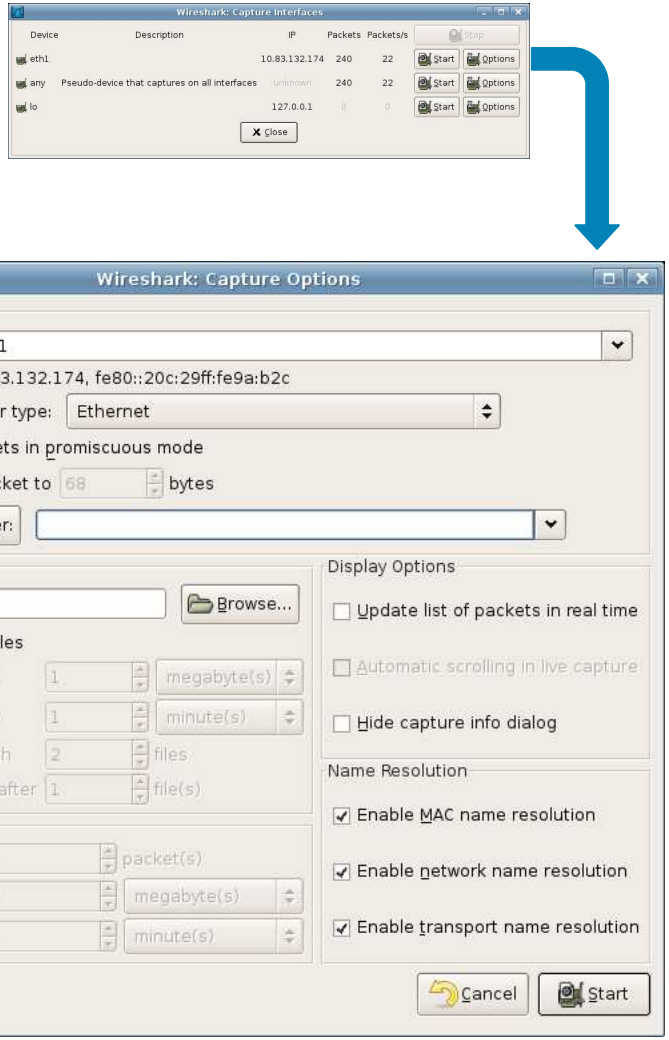
Real-time stats are shown with
basic breakdown of captured packets

Clock on Stop to
Stop and Analyze in detail



Capture Options

- Allow user to select ‘how’ the capture is done
- Capture Filters
- Where to store capture file
- Real-time Capture
- When to stop Capturing



Security- Capturing Packets

- Capturing generally super-user capability
  - BSD does not require SU to capture in promiscious
- Have been number of security related issues
  - Large number of dissectors from variety of people
  - Large infrastructure code (GTK, etc)
- For pure capture, ‘tshark’ in capture-only mode or ‘tcpdump’ might be better option
  - Analysis in Wireshark

Other Capture Options

- SPAN'ing or RSPAN'ing switch traffic to protocol analyzer port
- Vendors have IP packet copy feature, eg

  ERSPAN, copy of capture traffic sent via GRE
  IOS IP Traffic Export (sampling, local copy supported)
  JUNOS port-mirroring
  Flexible NetFlow, export payload
  Lawful Intercept

- Embedded capture

  c6500 monitor type capture
  WAAS tethereal capture

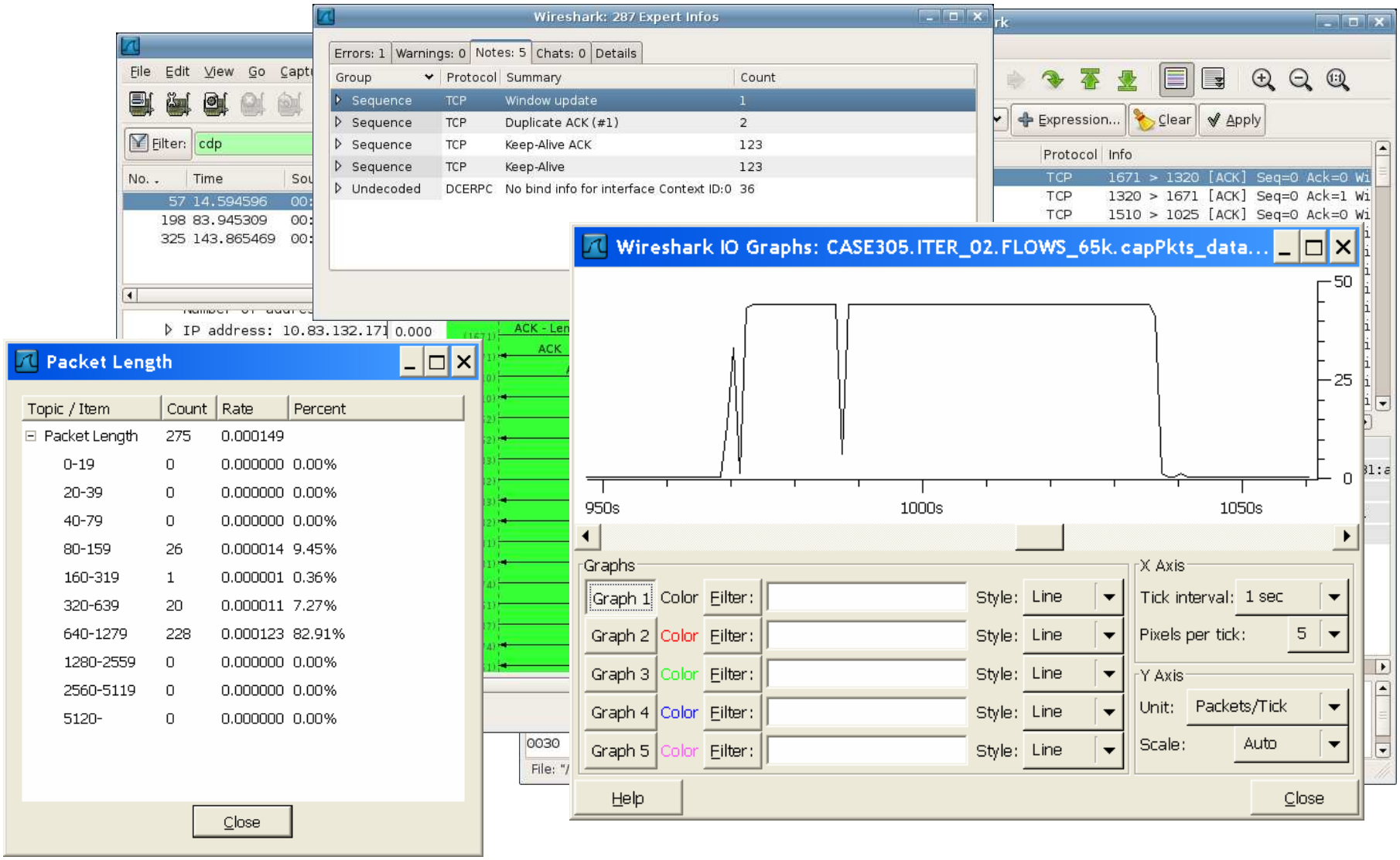
- Specialized hardware

  NAM modules in 6500, ISRs

Current Limitations with PCAP

- Single linktype per file
  - How to represent serial / etherenet / etc in the same file?
  - IP RAW means loss of L2 information
  - Linux pcap on ‘special’ interface can represent multiple link-types
- Per-Packet Information (PPI) Header
  - driven by CACE Tech
  - New shim (represented as linktype) in PCAP
  - Extensible:
    - Can encode multiple linktypes
    - Annotations, etc…
  - Wireshark 0.99.6

Analyzing Captured Frames



Wireshark
Basic Decode Flow

Reading File

- Can open one file at a time
- Each packet has a frame_data data structure
  - Points to next packet, previous packet
  - Information about the packet
    - time of capture
    - size of packet
    - location of where the actual data (what was seen on the wire) for this frame starts

Protocol Dissection

- When packet is selected in packet list top level dissector is called (eg ethernet)
- This dissector will call other dissectors as needed (eg, IP, MPLS, IPX etc), which will call other dissectors (IP,TCP, ICMP etc)
  - Keeps on going until a dissector doesn't call anybody
- Dissectors register at startup, create parent-child relationship with other protocols/dissectors

Protocol Dissection

- Protocol Tree pane is created by ‘typed’ functions (IPv4 address, bitfield, TLVs, strings, display (dec, hex) etc)

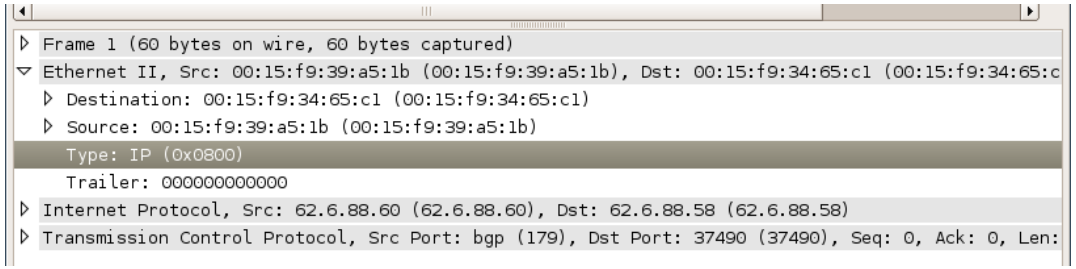
  Makes searching/filtering possible at same time

name abbrev type display

```
hf_eth_type, { "Type", "eth.type", FT_UINT16, BASE_HEX,
VALS(etype_vals), 0x0,   “EtherType”, HFILL }
```

strings bitmask blurb

```
const value_string etype_vals[] = {
        {ETHERTYPE_IP, "IP" },
        {ETHERTYPE_IPv6, "IPv6" },
```



Adding Items to Tree

- proto_XXX_DO_YYY()

```
proto_tree_add_item(tree, id, tvb, start,
length, little_endian);

proto_tree_add_string(tree, id, tvb, start,
length, value_ptr);

Many others…
```

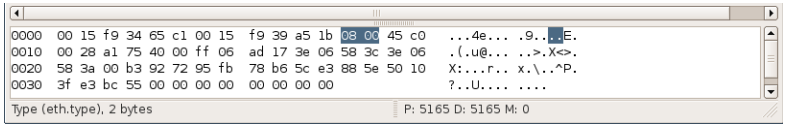
tree: current protocol tree to attach to

id: header format used

tvb: buffer to use

start: start offset for field

length: length of field



How to Update Wireshark

Why add to Wireshark

- Dean Wilson:

  "The switch talking to the master radio did have a lot of blinking lights, but watching blinking lights to measure traffic is about as accurate as using your tongue as a battery tester."

- Generally, protocol changes are just new fields on top of a existing framework.
- If you know where to make the changes, it is an extremely simple job.
- Beats looking at flashing lights

Getting a Build Environment

- Can use Windows environment but difficult to setup
- Alternatively, cygwin under windows
- Unix/Linux (incl Mac OS X) environment is easier.
  - Generally works out of the box
- Getting the source code
  - via SVN
  - via tarball

Getting a Build Environment (2)

- SVN is easier as you can
  - Dynamically update
  - Compare your changes against committed code
- Doing the checkout:

```
> svn co http://anonsvn.wireshark.org/wireshark/trunk/ wireshark
A    wireshark/cmake
A    wireshark/cmake/modules
A    wireshark/cmake/modules/FindGLIB2.cmake
A    wireshark/cmake/modules/FindYACC.cmake
…
```

'wireshark' directory created.
Checkout is from the latest code

Wireshark Directory

The checkout will create an entire directory structure.

Decoding code that we will be looking at is in:

wireshark/epan/dissectors

```
wireshark
|-- aclocal-fallback
|-- asn1
|-- autom4te.cache
|-- cmake
|-- codecs
|-- debian
|-- diameter
|-- doc
|-- docbook
|-- dtds
|-- epan
|-- gtk
|-- help
|-- idl
|-- image
|-- packaging
|-- plugins
|-- radius
|-- test
|-- tools
`-- wiretap
```

```
wireshark/epan
|-- dfilter
|-- dissectors
|    |-- dcerpc
|    |    |-- budb
|    |    |-- butc
|    |    `-- drsuapi
|    `-- pidl
|         `-- nspi
|-- ftypes
`-- wslua
```

Or use Aamer's VMware Build Box

- Xubuntu based (Debian Linux derivative)
- Build tools already installed
- Wireshark code is already checked out
- Available at:
  <URL TODO> volunteers for hosting 3gig vmware image?
  - Bittorrent?

Test Build (autogen.sh)

- autogen.sh will make configure (next step) script for you (among other things)
- Configure script already provided in tarball
  - Ie autogen.sh is only needed with svn checkout

```
wireshark> ./autogen.sh
Checking for python.
processing .
aclocal -I ./aclocal-fallback
libtoolize --copy --force
autoheader
automake --add-missing --gnu
autoconf
processing wiretap
aclocal -I ../aclocal-fallback
autoheader
automake --add-missing --gnu
autoconf

Now type "./configure [options]" and "make" to compile Wireshark.
```

Test Build (configure)

```
wireshark> ./configure --without-ucd-snmp --without-net-snmp
checking build system type... i686-pc-linux-gnu
checking host system type... i686-pc-linux-gnu
checking target system type... i686-pc-linux-gnu
```

```
The Wireshark package has been configured with
the following options.
                    Build wireshark : yes
                       Build tshark : yes
                     Build capinfos : yes
                      Build editcap : yes
                      Build dumpcap : yes
                     Build mergecap : yes
                    Build text2pcap : yes
                      Build idl2wrs : yes
                      Build randpkt : yes
                       Build dftest : yes

                     Install setuid : no
                        Use plugins : yes
                   Build lua plugin : no
                   Build rtp_player : no
               Use GTK+ v2 library : yes
                       Use threads : no
           Build profile binaries : no
                 Use pcap library : yes
                 Use zlib library : yes
                 Use pcre library : no
                …
```

- In wireshark directory:
- Configure (generally default options will work)

Test Build (make)

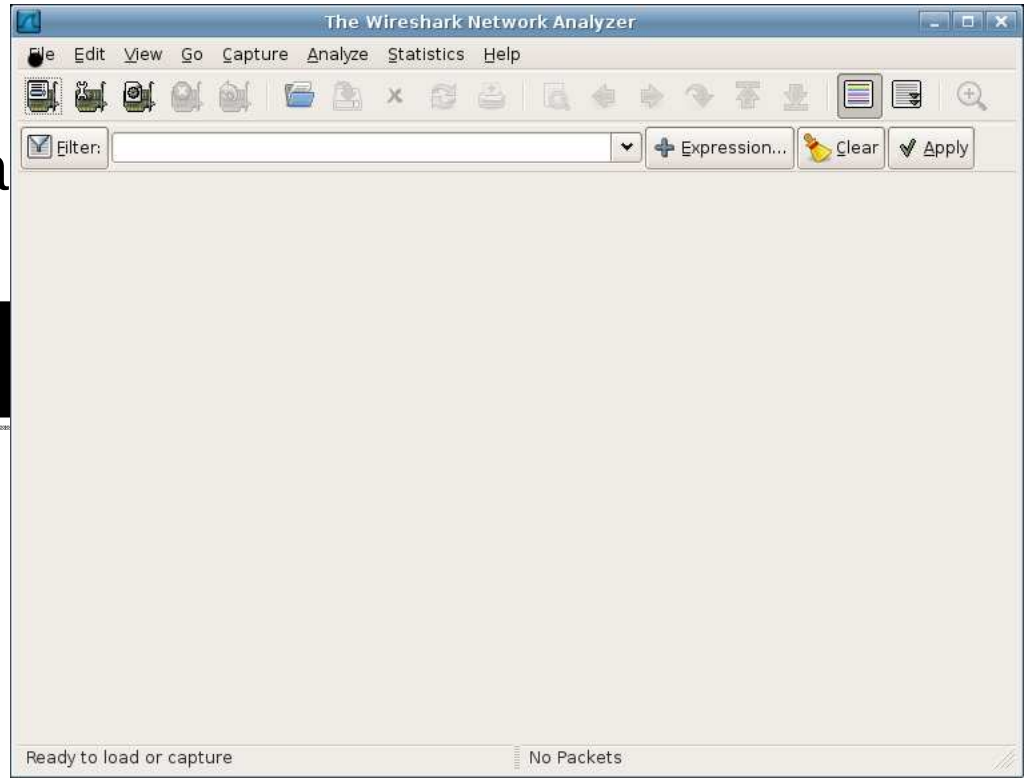
- If configure is successful:
- Run ‘make’
- This will compile wireshark

```
wireshark> make
cd . && /bin/sh /users/aakhter/src/wireshark-orig/missing --run autoconf
/bin/sh ./config.status --recheck
running /bin/sh ./configure  --without-ucd-snmp --without-net-snmp CC=gcc
…
```

Test Build (make)

- If configure is successful:
- Run ‘make’
- This will compile wiresha

```
wireshark> make
…
Wireshark> ./wireshark
```



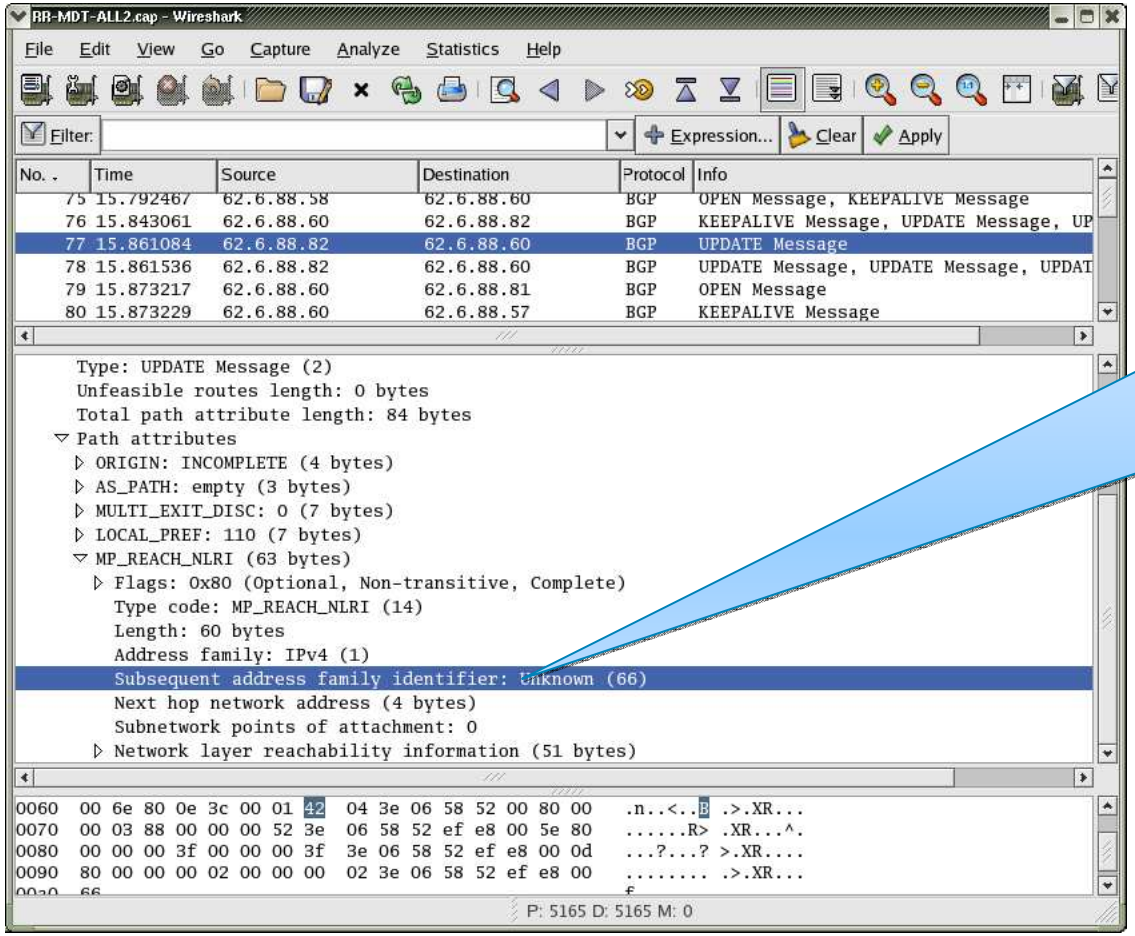
Words of Caution (Portability)

- Wireshark is supposed to run on multiple platforms and compilers. Be careful in what you use and how you use it.
- It's not C++, so no comments with //
- No 0-length Arrays
- Variable declaration should be outside function or beginning of function
- No inline
- No uchar, u_char, u_shart, use guint8 (8 bit unsigned) etc
- Many more in README.developer section 1.1.1

Sample Exercise: Adding BGP MDT support

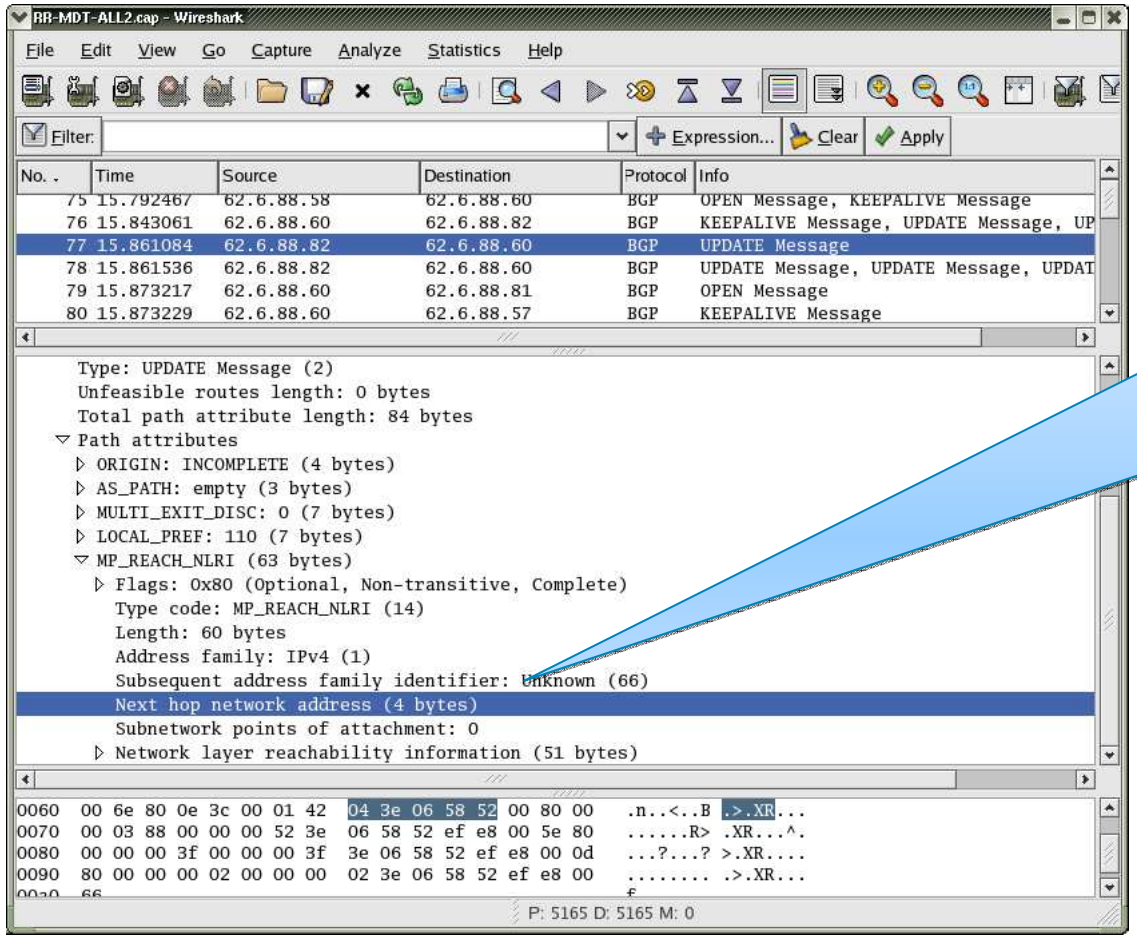
- MDT SAFI creates a new MP-BGP sub-address family for the advertisement of multicast-vpn route distringuisher, P source address, and multicast group address
- MDT SAFI is defined in:
  - draft-nalawade-idr-mdt-safi
- Currently Wireshark does not support MDT SAFI
- Walk through adding support
  - Identify work items
  - Add and test each item
- pcap file used available at:
  - http://www.employees.org/~aamer/RR-MDT-ALL2.cap

Identify Work Items



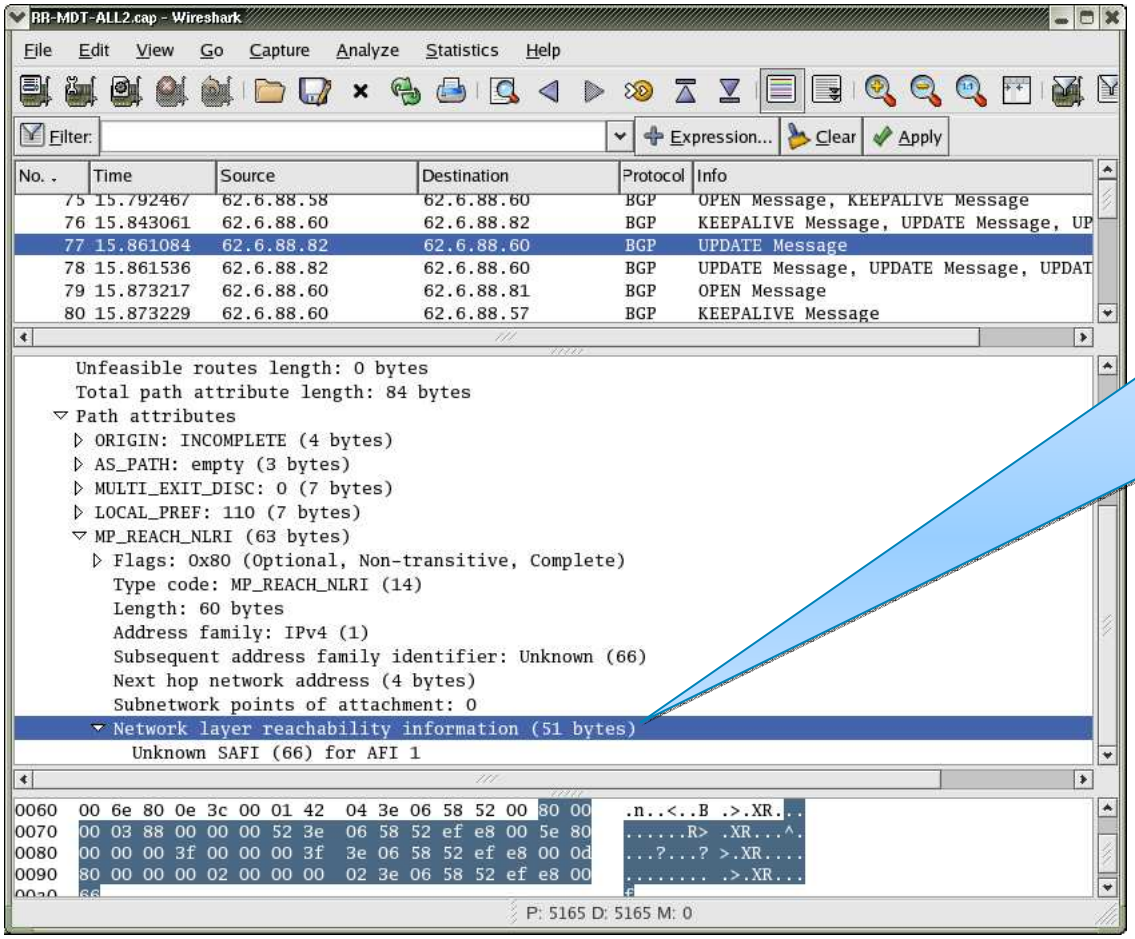
1. MDT SAFI is not identified

Identify Work Items



2. Next Hop is not expanded

Identify Work Items



3. NLRI is not understood

Things to Do

- In BGP MP_NLRI
  1. MVPN SAFI is unknown (66)
  2. Next Hop is not expanded to IPv4 address
  3. NLRI information is not understood
- All three items are related to BGP

  Basically 2 files:

  packet-bgp.c

  packet-bgp.h

  Majority of updates are 1-2 files

  File naming format is:

  packet-<protocol>.[c|h]

```
wireshark/epan
|-- dfilter
|-- dissectors
|   |-- dcerpc
|   |   |-- budb
|   |   |-- butc
|   |   `-- drsuapi
|   `-- pidl
|       `-- nspi
|-- ftypes
`-- wslua
```

MVPN SAFI is unknown

```
Length: 60 bytes
Address family: IPv4 (1)
Subsequent address family identifier: Unknown (66)
Next hop network address (4 bytes)
Subnetwork points of attachment: 0
```

- 66 = “Multicast Distribution Tree”
- Wireshark code is such that Number Value to Text Representation is usually held in constants in the beginning of the file.
- A quick check of packet-bgp.c shows SAFI names held in bgpattr_nlri_safi.

emacs@localhost

File Edit Options Buffers Tools C Help

```
/* Subsequent address family identifier, RFC2858 */
static const value_string bgpattr_nlri_safi[] = {
    { 0, "Reserved" },
    { SAFNUM_UNICAST, "Unicast" },
    { SAFNUM_MULCAST, "Multicast" },
    { SAFNUM_UNIMULC, "Unicast+Multicast" },
    { SAFNUM_MPLS_LABEL, "Labeled Unicast"},
    { SAFNUM_TUNNEL, "Tunnel"},
    { SAFNUM_VPLS, "VPLS"},
    { SAFNUM_LAB_VPNUNICAST, "Labeled VPN Unicast" },        /* draft-rosen-rfc
2547bis-03 */
    { SAFNUM_LAB_VPNMULCAST, "Labeled VPN Multicast" },
    { SAFNUM_LAB_VPNUNIMULC, "Labeled VPN Unicast+Multicast" },
    { 0, NULL },
};
```

MVPN SAFI is unknown

```
Length: 60 bytes
Address family: IPv4 (1)
Subsequent address family identifier: Unknown (66)
Next hop network address (4 bytes)
Subnetwork points of attachment: 0
```

- A grep for SAFNUM_UNICAST shows that it is defined in packet-bgp.h

Add in packet-bgp.h:

```
#define SAFNUM_MPLS_LABEL 4      /* rfc3107 */
#define SAFNUM_TUNNEL    64  /*draft-nalawade-kapoor-tunnel-safi-02.txt */
#define SAFNUM_VPLS      65
#define SAFNUM_MDT       66  /*draft-nalawade-idr-mdt-safi*/
#define SAFNUM_LAB_VPNUNICAST  128     /* Draft-rosen-rfc2547bis-03 */
#define SAFNUM_LAB_VPNMULCAST  129
```

MVPN SAFI is unknown

```
Length: 60 bytes
Address family: IPv4 (1)
Subsequent address family identifier: Unknown (66)
Next hop network address (4 bytes)
Subnetwork points of attachment: 0
```

Add in packet-bgp.c for bgpattr_nlri_safi:

```
/* Subsequent address family identifier, RFC2858 */
static const value_string bgpattr_nlri_safi[] = {
    { 0, "Reserved" },
    { SAFNUM_UNICAST, "Unicast" },
    { SAFNUM_MULCAST, "Multicast" },
    { SAFNUM_UNIMULC, "Unicast+Multicast" },
    { SAFNUM_MPLS_LABEL, "Labeled Unicast"},
    { SAFNUM_TUNNEL, "Tunnel"},
    { SAFNUM_VPLS, "VPLS"},
    { SAFNUM_MDT, "Multicast Distribution Tree"},
    { SAFNUM_LAB_VPNUNICAST, "Labeled VPN Unicast" },
    { SAFNUM_LAB_VPNMULCAST, "Labeled VPN Multicast" },
    { SAFNUM_LAB_VPNUNIMULC, "Labeled VPN Unicast+Multicast" },
    { 0, NULL },
};
```

MVPN SAFI is unknown

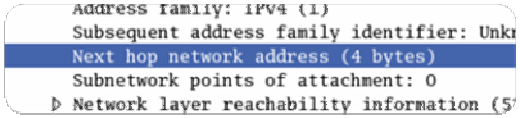
```
Length: 60 bytes
Address family: IPv4 (1)
Subsequent address family identifier: Unknown (66)
Next hop network address (4 bytes)
Subnetwork points of attachment: 0
```

- Save changes and running ‘make’ (configure not required)
- MDT SAFI is recognized

```
Unfeasible routes length: 0 bytes
Total path attribute length: 84 bytes
▽ Path attributes
  ▷ ORIGIN: INCOMPLETE (4 bytes)
  ▷ AS_PATH: empty (3 bytes)
  ▷ MULTI_EXIT_DISC: 0 (7 bytes)
  ▷ LOCAL_PREF: 110 (7 bytes)
  ▽ MP_REACH_NLRI (63 bytes)
    ▷ Flags: 0x80 (Optional, Non-transitive, Complete)
      Type code: MP_REACH_NLRI (14)
      Length: 60 bytes
      Address family: IPv4 (1)
      Subsequent address family identifier: Multicast Distribution Tree (66)
      Next hop network address (4 bytes)
      Subnetwork points of attachment: 0
    ▷ Network layer reachability information (51 bytes)

00 6e 80 0e 3c 00 01 42  04 3e 06 58 52 00 80 00   .n..<..B .>.XR...
00 03 88 00 00 00 52 3e  06 58 52 ef e8 00 5e 80   ......R> .XR...^.
00 00 00 3f 00 00 00 3f  3e 06 58 52 ef e8 00 0d   ...?...? >.XR....
```

2. Next Hop is Not Expanded

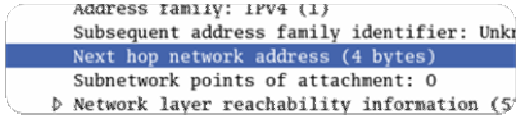


- The next hop address isn't shown (just the length)
- It's an IPv4 address here.
- Slightly more complicated than the last example. A quick search for "Next hop network address" in packet-bgp.c leads us to this code:

```
ti = proto_tree_add_text(subtree2, tvb, o + i + aoff + 3,
                 nexthop_len + 1,
                 "Next hop network address (%d %s)",
                 nexthop_len, plurality(nexthop_len,
                 "byte", "bytes"));
subtree3 = proto_item_add_subtree(ti, ett_bgp_mp_nhna);
```

- Note that we've found where the '4 bytes' is being added
- Note that a new subtree (subtree3) is being attached

Next Hop is Not Expanded

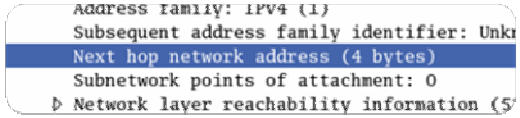



```
advance = mp_addr_to_str(af, saf, tvb, o + i + aoff + 4 + j,
   junk_gbuf, MAX_STR_LEN) ;
…
proto_tree_add_text(subtree3, tvb,o + i + aoff + 4 + j,
   advance, "Next hop: %s (%u)", junk_gbuf, advance);
```

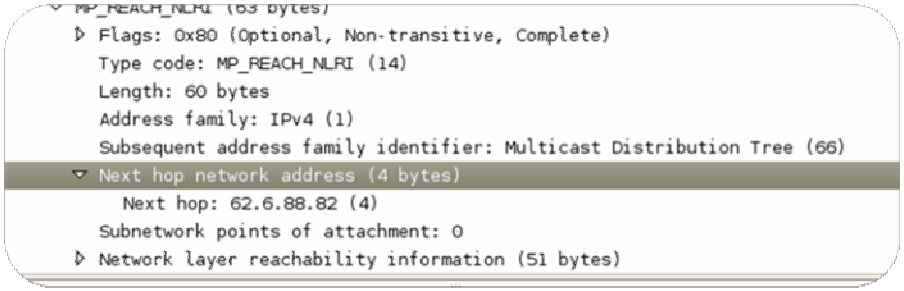
- A few lines down the text is added to `subtree3` “Next hop…” That is what we’re missing!
- The text for this is coming from ‘`advance`’ which is created by the function ‘`mp_addr_to_str`’
- `mp_addr_to_str` has this bit of code:

```
switch (safi) {
                        case SAFNUM_UNICAST:
                        case SAFNUM_MULCAST:
                        case SAFNUM_UNIMULC:
                        case SAFNUM_MPLS_LABEL:
                        case SAFNUM_TUNNEL:
                                  length = 4 ;
```

Next Hop is Not Expanded



- Our MDT SAFI is also an IPv4 address, just like SAFNUM_UNICAST, and SAFNUM_MPLS_LABEL.
- If we add SAFNUM_MDT to the list and recompile



- Note how subtree3 is a child of the ‘Next hop network address’
- The IPv4 next-hop address is now decoded!

3. NLRI not Understood

```
Subsequent address family identifier: Unknown (66)
Next hop network address (4 bytes)
Subnetwork points of attachment: 0
▼ Network layer reachability information (51 bytes)
  Unknown SAFI (66) for AFI 1
```

- The NLRI information is not understood.
- draft-nalawade-idr-mdt-safi section 5 describes the format:

Route Distinguisher (8 Bytes)
MDT Source Address (4 Bytes)
MDT Group Address (4 Bytes)

Route-Distinguisher: is the RD of the VRF to which this MDT attribute belongs.

MDT Source Address: is the source address of MDT.

MDT Group Address: is the Group-address of the MDT-Group that a VRF is associated to.

NLRI not Understood

```
Subsequent address family identifier: Unknown (66)
Next hop network address (4 bytes)
Subnetwork points of attachment: 0
Network layer reachability information (51 bytes)
  Unknown SAFI (66) for AFI 1
```

- The text for NLRI in the field is:

```
0070  00 03 88 00 00 00 52 3e 06 58 52 ef e8 00 5e 80
0080  00 00 00 3f 00 00 00 3f 3e 06 58 52 ef e8 00 0d
0090  80 00 00 00 02 00 00 00 02 3e 06 58 52 ef e8 00
00a0  66
```

- This maps to:

  RD: 904:82
  Source: 62.6.88.82
  Multicast Destination: 239.232.0.94

  RD: 63:63
  Source: 62.6.88.82
  Multicast Destination: 239.232.0.13

  ….

Adding MDT NLRI support

```
Subsequent address family identifier: Unknown (66)
Next hop network address (4 bytes)
Subnetwork points of attachment: 0
▼ Network layer reachability information (51 bytes)
  Unknown SAFI (66) for AFI 1
```

- `decode_prefix_MP` has the code for decoding the NLRI. This section shows how the decoding is done for IPv4 unicast, multicast, labeled prefixes

```
/*
 * Decode a multiprotocol prefix
 */
static int
decode_prefix_MP(proto_tree *tree, int hf_addr4, int hf_addr6,
    guint16 afi, guint8 safi, tvbuff_t *tvb, gint offset, const char *tag)
{
.......
    switch (afi) {

    case AFNUM_INET:
        switch (safi) {

        case SAFNUM_UNICAST:
        case SAFNUM_MULCAST:
        case SAFNUM_UNIMULC:
            total_length = decode_prefix4(tree, hf_addr4, tvb, offset, 0, tag);
            if (total_length < 0)
                return -1;
            break;

        case SAFNUM_MPLS_LABEL:
            plen =  tvb_get_guint8(tvb, offset);
            labnum = decode_MPLS_stack(tvb, offset + 1, lab_stk, sizeof(lab_stk)
);

            offset += (1 + labnum * 3);
            if (plen <= (labnum * 3*8)) {
                proto_tree_add_text(tree, tvb, start_offset, 1,
-:**  packet-bgp.c      (C Abbrev)--L651--21%-----------------------------------
```

Adding MDT NLRI support

```
Subsequent address family identifier: Unknown (66)
Next hop network address (4 bytes)
Subnetwork points of attachment: 0
Network layer reachability information (51 bytes)
  Unknown SAFI (66) for AFI 1
```

```
case SAFNUM_MDT:
        /* MDT format is [RD 8B][unicast ipv4 4B][MDT 4B] = 16B*/
          plen =  tvb_get_guint8(tvb, offset);
          offset += 1;
          if (plen != (16*8)) {
              proto_tree_add_text(tree, tvb, start_offset, 1,
                          "%s MDT NLRI length %u is invalid",
                      tag, plen);
              return -1;
          }
          rd_to_str(tvb, offset, &rd_str);
          ip4addr.addr = tvb_get_ipv4(tvb, offset + 8);   /* IPv4 nexthop */
          ip4addr2.addr = tvb_get_ipv4(tvb, offset + 12);   /* IPv4 mcast MDT */
          ti = proto_tree_add_text(tree, tvb, offset, 16,
                          "RD=%s, source=%s, default MDT=%s",
                          rd_str,ip_to_str((guint8 *)&ip4addr), ip_to_str((guint8 *)&ip4addr2));
          prefix_tree = proto_item_add_subtree(ti, ett_bgp_prefix);
          proto_tree_add_text(prefix_tree, tvb, start_offset, 1, "%s Prefix length: %u",
                      tag, plen);
          proto_tree_add_text(prefix_tree, tvb,offset,8,"%s Route Distinguisher: %s",
                              tag, rd_str);
          proto_tree_add_item(prefix_tree, hf_bgp_mdt_source, tvb,
                              offset + 8, 4, FALSE);
          proto_tree_add_item(prefix_tree, hf_bgp_mdt_default, tvb,
                              offset + 12, 4, FALSE);
          total_length = 17;
          break;
```

Let’s break it up

```
Subsequent address family identifier: Unknown (66)
Next hop network address (4 bytes)
Subnetwork points of attachment: 0
Network layer reachability information (51 bytes)
  Unknown SAFI (66) for AFI 1
```

- We catch the SAFI 66 (MDT), and grab the first byte from the buffer.
- The first byte in the BGP NLRI is the bit length of the prefix.
- As we always have 128 bits (16B), that is a constant
- Anything else is a malformed NLRI

  So we add text to the tree stating as such

```
case SAFNUM_MDT:
        /* MDT format is [RD 8B][unicast ipv4 4B][MDT 4B] = 16B*/
          plen =  tvb_get_guint8(tvb, offset);
          offset += 1;
          if (plen != (16*8)) {
              proto_tree_add_text(tree, tvb, start_offset, 1,
                          "%s MDT NLRI length %u is invalid",
                      tag, plen);
              return -1;
          }
```

Let's break it up (2)

```
Subsequent address family identifier: Unknown (66)
Next hop network address (4 bytes)
Subnetwork points of attachment: 0
▼ Network layer reachability information (51 bytes)
  Unknown SAFI (66) for AFI 1
```

- We previously incremented `offset` by 1, moving past the NLRI length
- Offset is setting at the RD, which is 8B

  There are multiple formats in RD (AS, IPv4 address etc)

  `rd_to_str` translates 8B to string via `rd_str`
- `tvb_get_ipv4` grabs the ipv4 source (8B past `offset`) and mcast destination (4B past ipv4 source)

```
 case SAFNUM_MDT:
 offset += 1;
…
            rd_to_str(tvb, offset, &rd_str);
            ip4addr.addr = tvb_get_ipv4(tvb, offset + 8);   /* IPv4 nexthop */
            ip4addr2.addr = tvb_get_ipv4(tvb, offset + 12);   /* IPv4 mcast MDT */
```

Let’s break it up (3)

```
Subsequent address family identifier: Unknown (66)
Next hop network address (4 bytes)
Subnetwork points of attachment: 0
▽ Network layer reachability information (51 bytes)
  Unknown SAFI (66) for AFI 1
```

- We add a free text line showing RD, source and default MDT
- The section of data runs from `offset` to 16 bytes (1 NLRI slice)
- This just a blob of text, it is not searchable
- `total_length` 17 denotes how much stuff was ‘eaten’

```
…
            ti = proto_tree_add_text(tree, tvb, offset, 16,
                                     "RD=%s, source=%s, default MDT=%s",
                                     rd_str,ip_to_str((guint8 *)&ip4addr), ip_to_str((guint8 *)&ip4addr2));
            total_length = 17;
            break;
```

```
Subsequent address family identifier: Multicast Distribution Tree (66)
▽ Next hop network address (4 bytes)
    Next hop: 62.6.88.82 (4)
  Subnetwork points of attachment: 0
▽ Network layer reachability information (51 bytes)
    RD=904:82, NextHop=62.6.88.82, default MDT=239.232.0.94
    RD=63:63, NextHop=62.6.88.82, default MDT=239.232.0.13
    RD=2:2, NextHop=62.6.88.82, default MDT=239.232.0.102
```

Searchable Fields

- Current method has a blob of text, not searchable, individual fields not broken down
- We need to create a sub-tree to attach the individual fields to
- ett_bgp_prefix has already been designated as the protocol tree to use
- We use `proto_item_add_subtree` to create a child tree of `ti` called `prefix_tree`

```
ti = proto_tree_add_text(tree, tvb, offset, 16,
                  "RD=%s, source=%s, default MDT=%s",
                  rd_str,ip_to_str((guint8 *)&ip4addr),
                   ip_to_str((guint8 *)&ip4addr2));
prefix_tree = proto_item_add_subtree(ti, ett_bgp_prefix);
```

Searchable Fields (2)

Route Distinguisher (8 Bytes)
MDT Source Address (4 Bytes)
MDT Group Address (4 Bytes)

- Add the NLRI prefix length (should be 128b) as free text from `plen`

  Note that `start_offset` was used and length of 1

- RD text is put in as free text using the rd_str variable we had populated earlier

  Note that RD starts at `offset` and has length of 8

```
…
            proto_tree_add_text(prefix_tree, tvb, start_offset, 1, "%s Prefix length: %u",
                        tag, plen);
            proto_tree_add_text(prefix_tree, tvb,offset,8,"%s Route Distinguisher: %s",
                                tag, rd_str);
```

Searchable Fields (3)

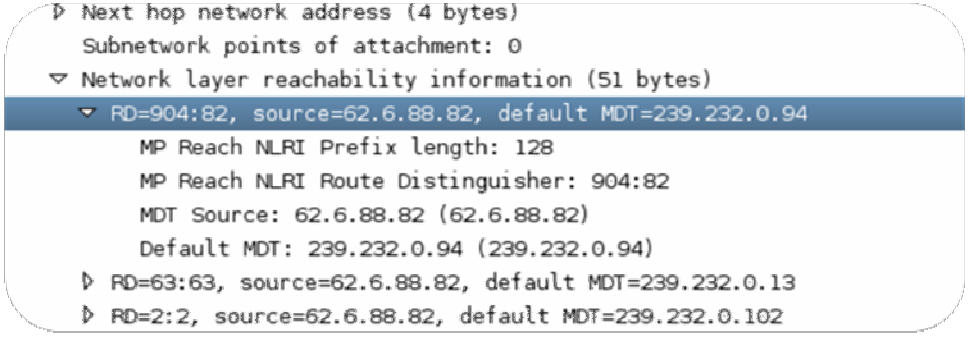
Route Distinguisher (8 Bytes)
MDT Source Address (4 Bytes)
MDT Group Address (4 Bytes)

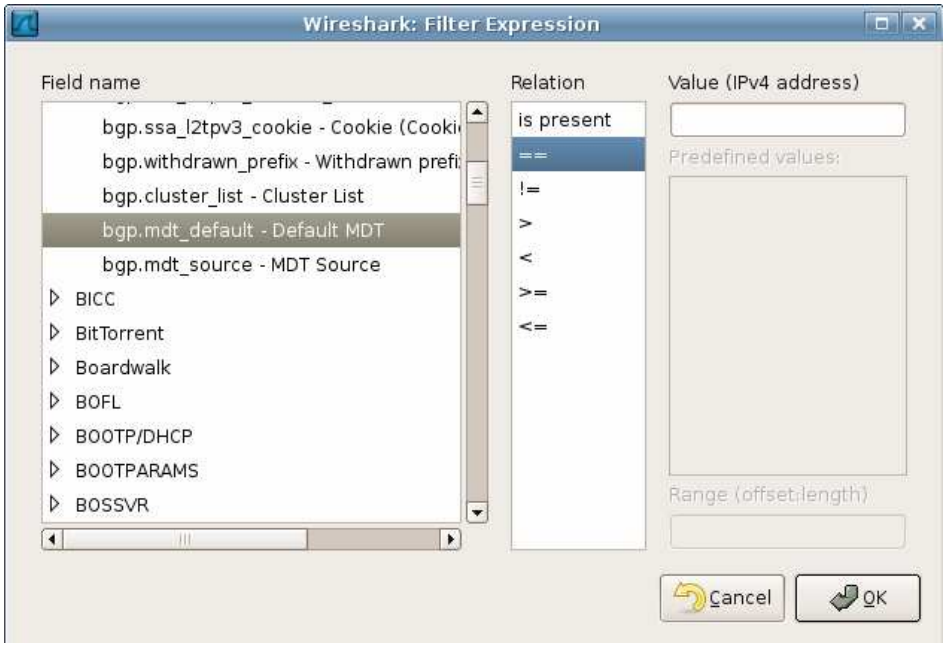
- For source IP and multicast MDT address used the hf_ field definition
  - Created `hf_bgp_mdt_default` and `hf_bgp_mdt_source`
  - Note that they are of type `FT_IPv4`
- Also note that they will be searchable via the `bgp.mdt_default` and `bgp.mdt_source` keywords

```
…
            proto_tree_add_item(prefix_tree, hf_bgp_mdt_source, tvb,
                                offset + 8, 4, FALSE);
            proto_tree_add_item(prefix_tree, hf_bgp_mdt_default, tvb,
                                offset + 12, 4, FALSE);
……
      { &hf_bgp_mdt_default,
        { "Default MDT", "bgp.mdt_default", FT_IPv4, BASE_NONE,
          NULL, 0x0, "", HFILL}},
      { &hf_bgp_mdt_source,
        { "MDT Source", "bgp.mdt_source", FT_IPv4, BASE_NONE,
          NULL, 0x0, "", HFILL}},
```

Searchable Fields and breakdown

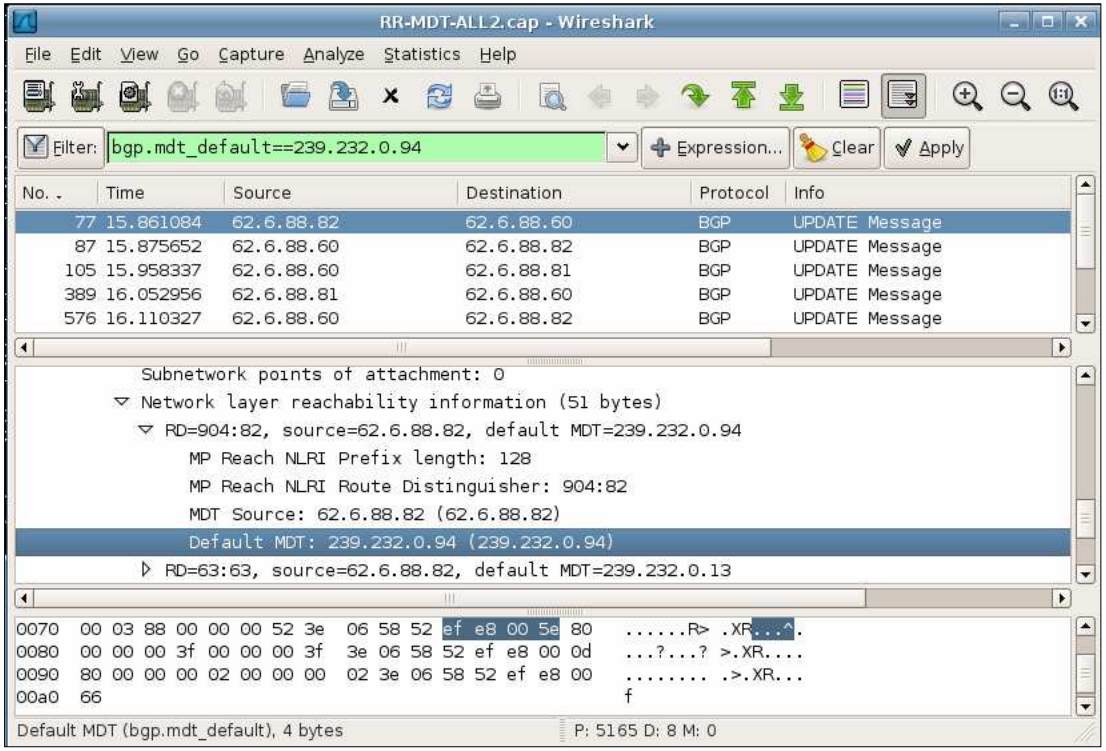
- Now there is a subtree under the blob text line
- Prefix len, RD, MDT source and MDT destination are broken out
- They are also individually highlighted in the hex display
- MDT source and MDT default are searchable





Support for Filters Added Automatically

- When using the hf_ fields not only is the rendering a lot easier
- But, support for field search is also added



Run Fuzz Tests

- Fuzz tests creates random packet variations based on capture files
- Tests your code against some variance (not just your perfect packets)
- By default, runs until breakage

```
wireshark$ ./tools/fuzz-test.sh ~/traces/RR-MDT-ALL2.cap
Running ./tshark with args: -nVxr (forever)

Pass 1:
    /home/vmware/traces/RR-MDT-ALL2.cap:  OK
Pass 2:
    /home/vmware/traces/RR-MDT-ALL2.cap:  OK
Pass 3:
    /home/vmware/traces/RR-MDT-ALL2.cap:  OK
Pass 4:
    /home/vmware/traces/RR-MDT-ALL2.cap:  OK
Pass 5:
    /home/vmware/traces/RR-MDT-ALL2.cap:  OK
Pass 6:
    /home/vmware/traces/RR-MDT-ALL2.cap:
```

Submitting The Patch

- You definitely want to submit your patch to the official Wireshark codebase. Benefits
  - Wider understanding of protocol
  - Maintenance of your code by other folks
  - Availability of your code in other platform (ie windows) builds
- Create a patch file by running svn diff in the wireshark directory. Email to wireshark-dev mailing list or attach it to bug at http://bugs.wireshark.org/
- Attach your capture file (remove private data, if needed) so it can be added to common fuzz library.

```
svn diff | gzip > my_patch.gz
```

Additional Help

- Wireshark Developers List

  http://www.wireshark.org/mailman/listinfo/wireshark-dev

- Archive:

  http://news.gmane.org/gmane.network.wireshark.devel

Summary

- Free Open Source Network Protocol Analyzer
- Multi-platform: Runs on Windows, Mac, Linux, Solaris, NetBSD, FreeBSD
- CLI as well as Graphical display
- 100’s of protocols supported

- Multiple methods to capture packets
- Easy to add additional protocols
- There are (nice) people to help you along the way