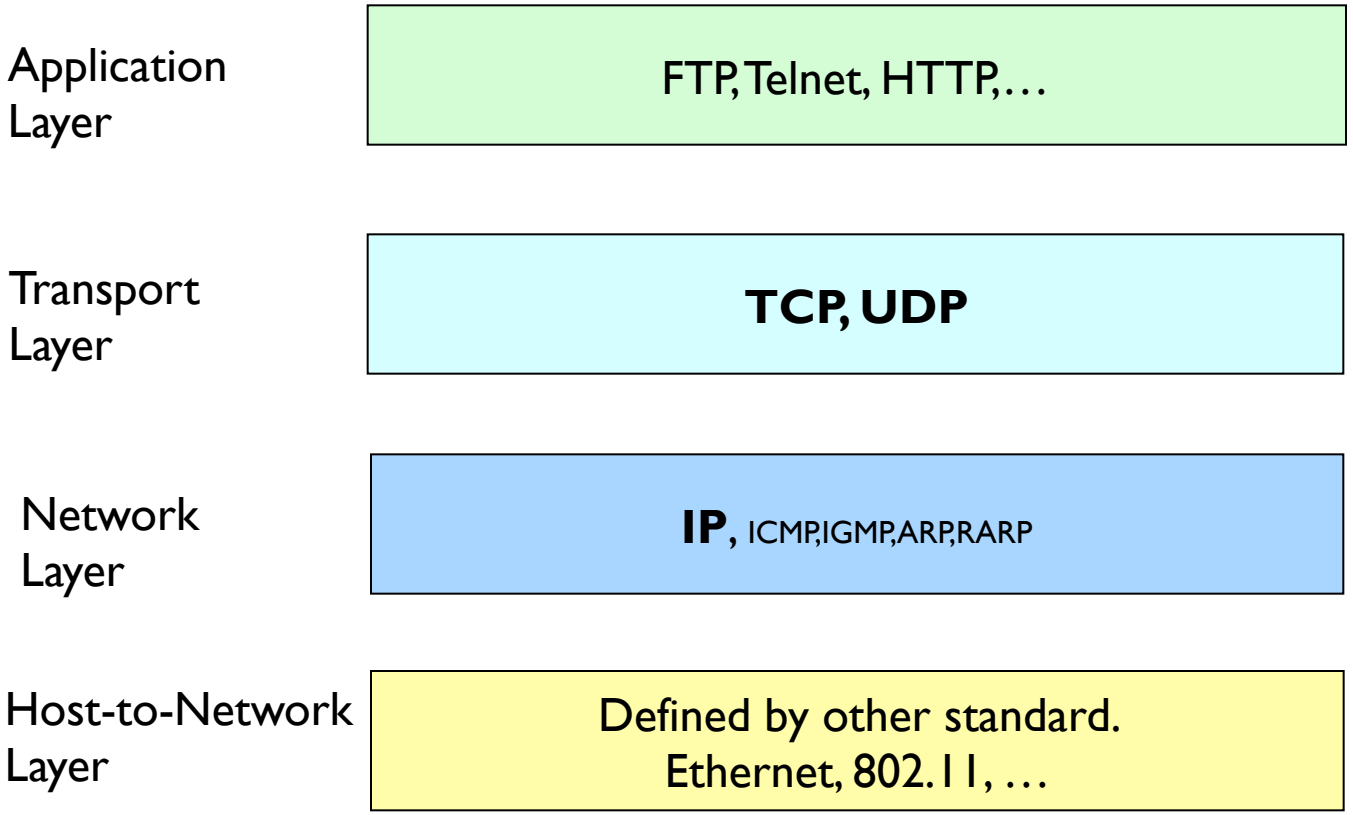
IP: Internet Protocol
RFC 791

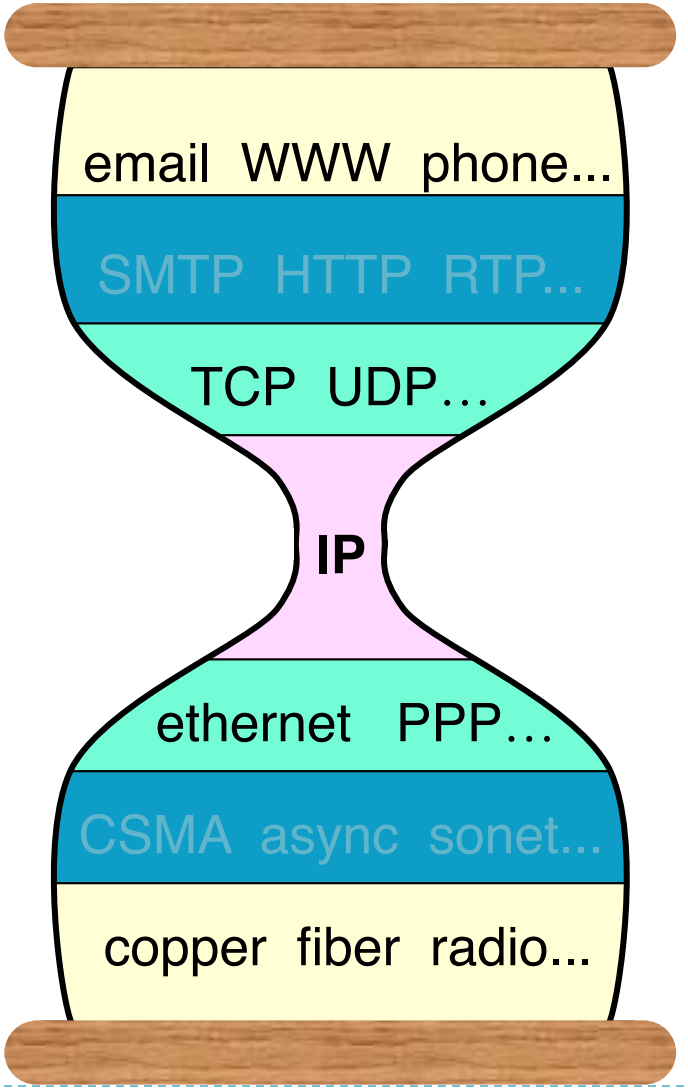
Prof. Lin Weiguo

Oct. 2013

TCP/IP Protocol Stack

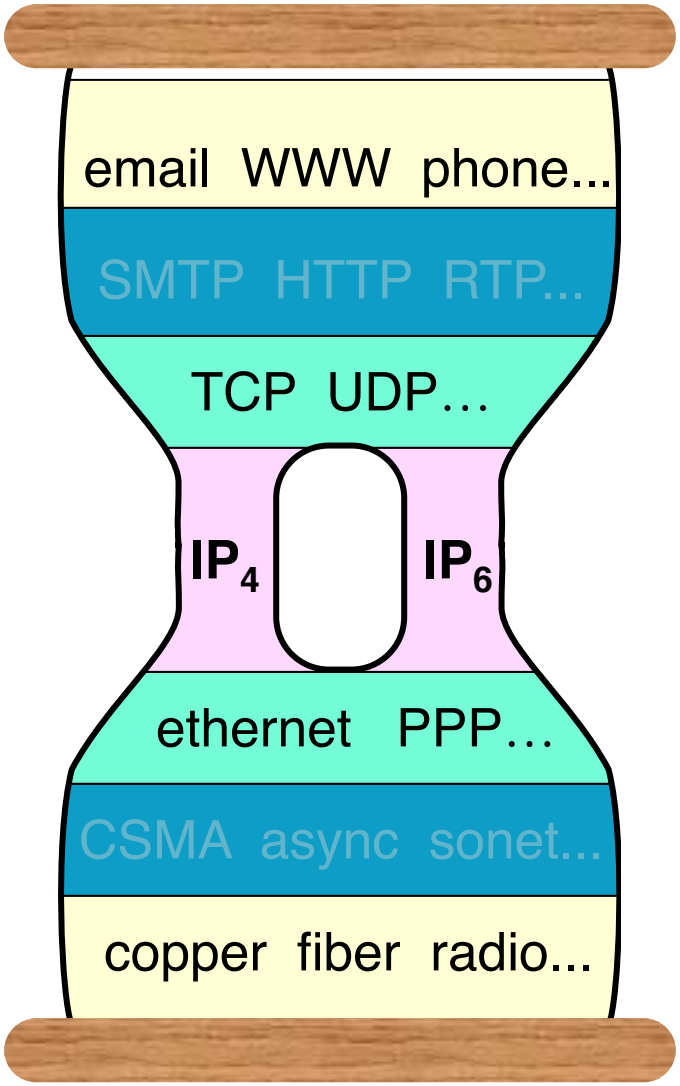


Hourglass of the Internet Architecture



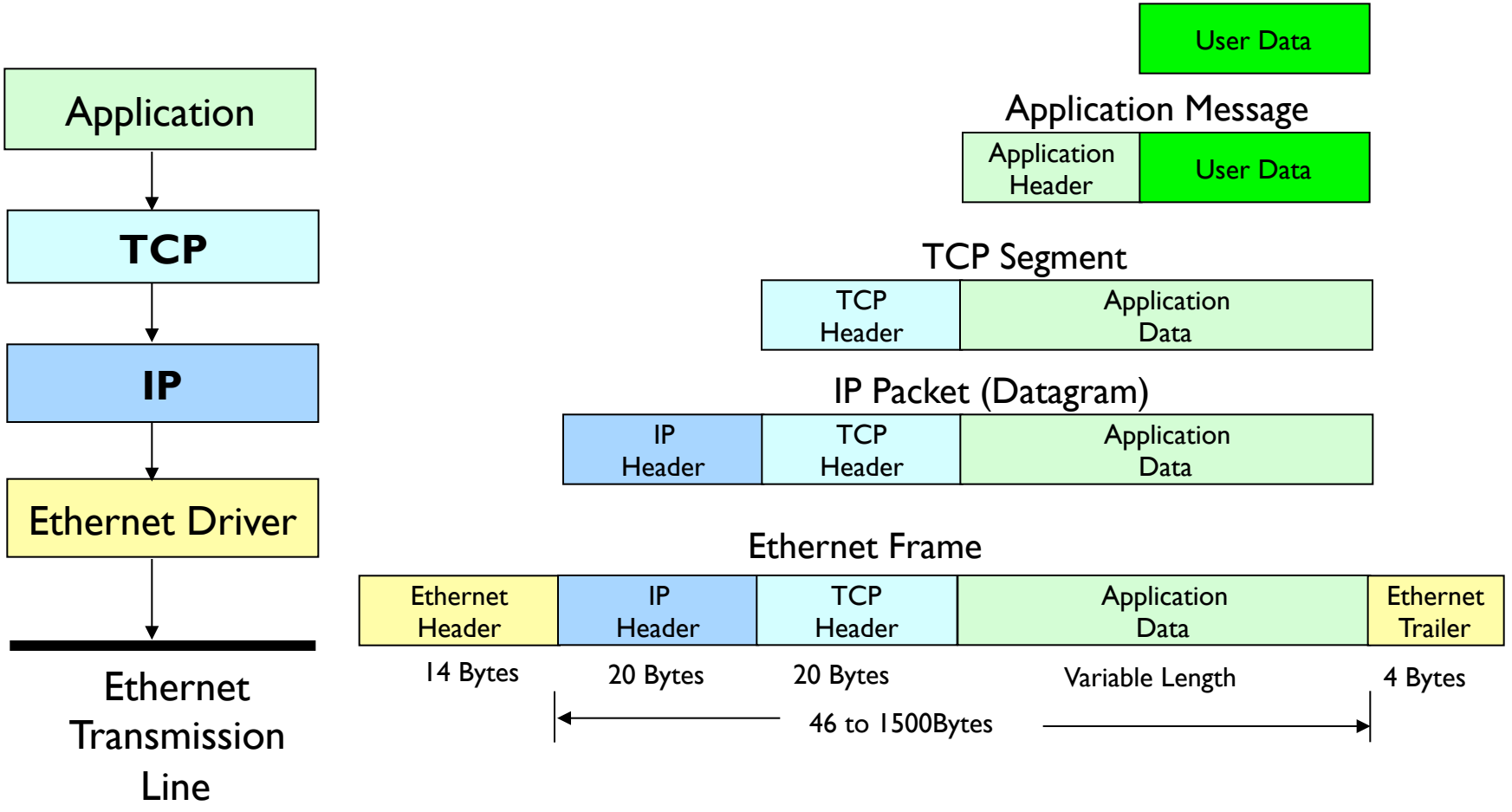
linwei@cuc.edu.cn

Mid-Life Crisis



linwei@cuc.edu.cn

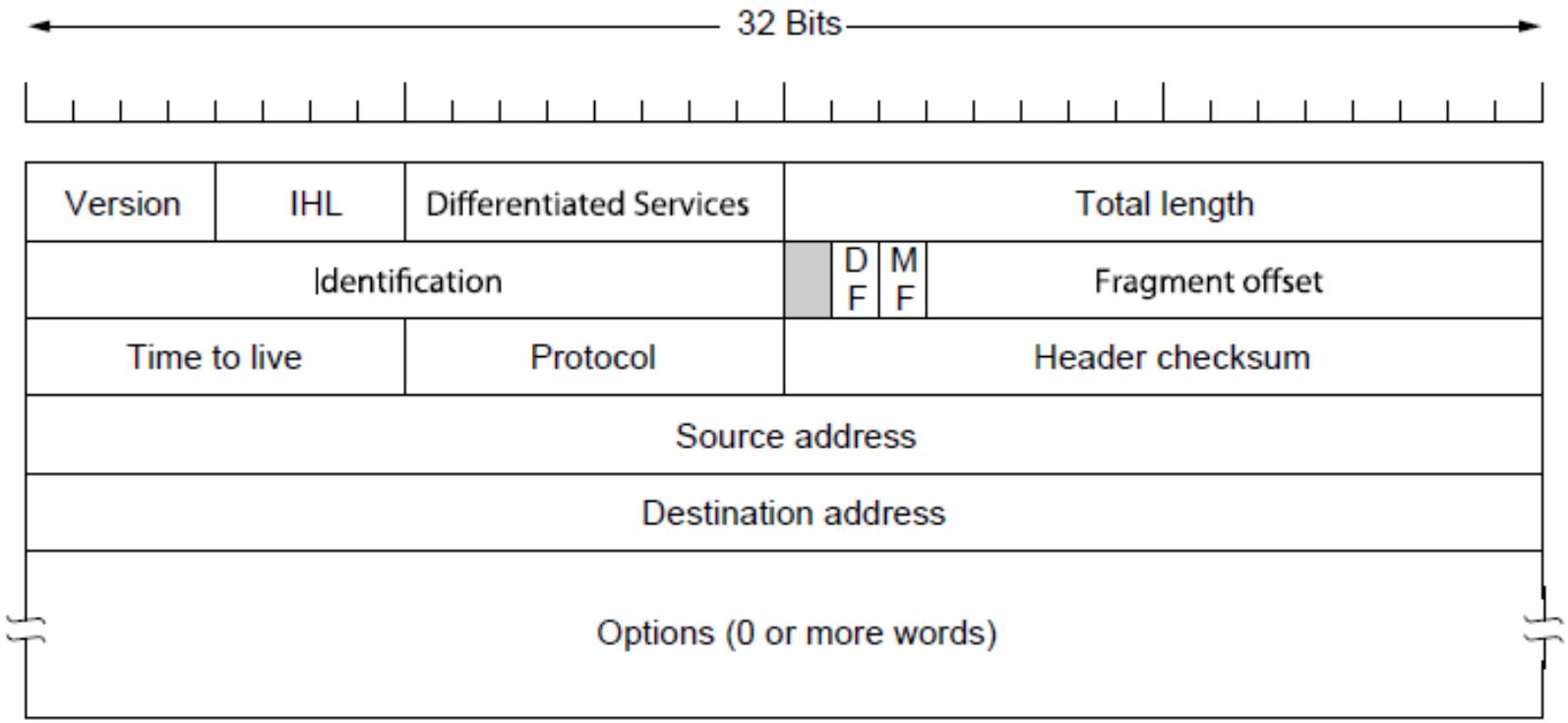
Protocol Data Unit Encapsulation



TCP/IP over Ethernet

linwei@cuc.edu.cn

The IP Datagram Format



The IPv4 (Internet Protocol) header.

The IP Datagram Format

1. Version number (4-bits):
   - The current protocol version is 4.
   - Including a version number allows a future version of IP be used along side the current version, facilitating migration to new protocols.
2. IHL: Header length (4-bits):
   - Length of the datagram header (excluding data) in 32-bit words.
   - The minimum length is 5 words = 20 bytes, but can be up to 15 words if options are used.
   - In practice, the length field is used to locate the start of the data portion of the datagram.

The IP Datagram Format

- Differentiated services(8-bits):
  - Originally, it was called the Type of Service field.
  - It was and still is intended to distinguish between different classes of service.
    - Digitized voice, file transfer …
  - A hint to the routing algorithms as to what type of service we desire. But in practice, routers ignore the TOS field in IPv4.
  - IETF has changed the field slightly to accommodate differentiated services.
    - Top 6 bits are used to mark the packet with its service class.
    - The bottom 2 bits are used to carry explicit congestion notification information.

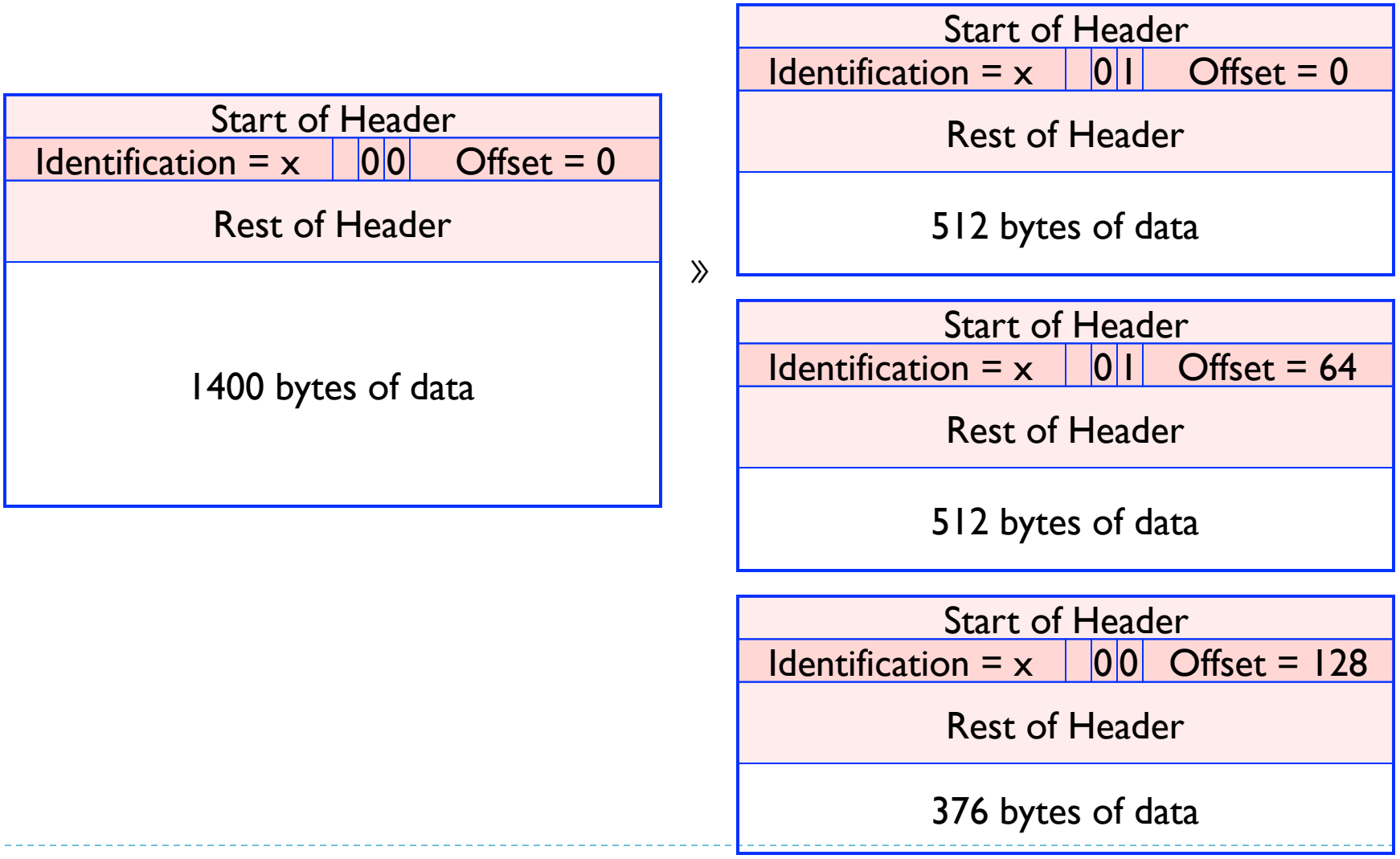
The IP Datagram Format

4. Total length (16-bits): Max=65535 bytes
   - Total length of the IP datagram (in bytes), including data and header.
   - Data length = Total length – Header size.
5. Identification
   - allow the destination host to determine which datagram a newly arrived fragment belongs to. All the fragments of a datagram contain the same Identification value.

The IP Datagram Format

6. Unused (1 bit)
7. DF (Don't Fragment , 1bit)
   - It is an order to the routers not to fragment the datagram because the destination is incapable of putting the pieces back together again.
8. MF (More Fragments, 1bit)
   - All fragments except the last one have this bit set. It is needed to know when all fragments of a datagram have arrived.
9. Fragment offset (13 bits)
   - The offset field shows order of the fragments.
   - All fragments except the last one in a datagram must be a multiple of 8 bytes, the elementary fragment unit. Since 13 bits are provided, there is a maximum of 8192 fragments per datagram, giving a maximum datagram length of 65,536 bytes, one more than the Total length field.

Example of Fragmentation



linwei@cuc.edu.cn

The IP Datagram Format

10. TTL (Time to Live, 8 bits)
    - A counter that is decremented by each gateway.
    - Should this hopcount reach 0, discard the datagram.
    - Originally, the time-to-live field was intended to reflect real time (up to 255 sec).
    - In practice, it is now a hopcount.
    - The time-to-live field squashes looping packets.

Note: TCP/IP and NBT configuration parameters for Windows XP
→http://support.microsoft.com/kb/314053/en-us

The IP Datagram Format

11. Protocol (8-bits):
    - What type of data the IP datagram carries (TCP, UDP, etc.)
    - Needed by the receiving IP to know the higher level service that will next handle the data.
    - originally defined by RFC 1700, now maintained by the http://www.iana.org/assignments/protocol-numbers/
        - ICMP:00000001
        - IGMP:00000010
        - TCP: 00000110
        - UDP: 00010001

The IP Datagram Format

12. Header Checksum (16-bits):A checksum of the IP header ONLY.
    - The IP checksum is computed as follows (RFC 1071):
        - Treat the data as a stream of 16-bit words (appending a 0 byte if needed).
        - Compute the 1's complement sum of the 16-bit words. Take the 1's complement of the computed sum.
    - We can place the checksum in a fixed location in the header, set it to zero, compute the checksum, and store its value in the checksum field.
    - On receipt of a datagram, the computed checksum calculated over the received packet should be zero.
    - Check summing only the header reduces the processing time at each gateway, but forces transport layer protocols to perform error detection (if desired).
    - The header must be recalculated at every router since the time_to_live field is decremented.

The IP Datagram Format

13. Source address (32-bits):
    - Original sender's address. This is an IP address, not a MAC address.
14. Destination address (32-bits):
    - Datagram's ultimate destination.

The IP embedded datagram contains the source of the original sender (not the forwarding gateway) and the destination address of the ultimate destination.

The IP Datagram Format

15. IP Options

- IP datagrams allow the inclusion of optional, varying length fields that need not appear in every datagram. We may sometimes want to send special information, but we don't want to dedicate a field in the packet header for this purpose.
- Options start with a **1-byte option code**, followed by zero or more bytes of option data.

The option code byte contains three parts:

copy flag (1 bit): If 1, replicate option in each fragment of a fragmented datagram. That is, this option should appear in every fragment as well. If 0, option need only appear in first fragment.

option class (2 bits): Purpose of option:

0 = network control

1 = reserved

2 = debugging and measurement

3 = reserved

The IP Datagram Format

option number (5 bits): A code indicating the option's type.

Option	Description
Security	Specifies how secret the datagram is
Strict source routing	Gives the complete path to be followed
Loose source routing	Gives a list of routers not to be missed
Record route	Makes each router append its IP address
Timestamp	Makes each router append its address and timestamp

Some of the IP options.

www.iana.org/assignments/ip-parameters

Microsoft Windows IP Helper API

- Internet Protocol Helper (IP Helper) assists network administration of the local computer by enabling applications to retrieve information about the network configuration of the local computer, and to modify that configuration. IP Helper also provides notification mechanisms to ensure that an application is notified when certain aspects of the local computer network configuration change.
- IP Helper provides capabilities in the following areas:
  - Retrieving Information About Network Configuration
  - Managing Network Adapters
  - Managing Interfaces
  - Managing IP Addresses
  - Using the Address Resolution Protocol
  - Retrieving Information on the Internet Protocol and the Internet Control Message Protocol
  - Managing Routing
  - Receiving Notification of Network Events
  - Retrieving Information About the Transmission Control Protocol and the User Datagram Protocol

About Microsoft Windows SDK

- The Windows SDK provides tools, compilers, headers, libraries, code samples, and a new help system that developers can use to create applications that run on Microsoft Windows.
- Windows SDK samples
  - Installed with Windows SDK version 7.1 (released May 19, 2010)
  - \Program Files\Microsoft SDKs\Windows\v7.1\samples

```
\Begin
\Com
\DataAccess
\Multimedia
\NetDS
\Security
\SysMgmt
\TabletPC
\Web
\WinBase
\WinUI
```

Advanced IP Helper Samples

- *C:\Program Files\Microsoft SDKs\Windows\v7.1\Samples\NetDs\IPHelp*
- EnableRouter
  - use the **EnableRouter** and **UnenableRouter** IP Helper functions to enable and disable IPv4 forwarding
- Iparp
  - use the IP Helper functions to display and manipulate entries in the IPv4 ARP table.
- Ipchange
  - use IP Helper functions to programmatically change an IP address for a specific network adapter. This program also demonstrates how to retrieve existing network adapter IP configuration information.
- IPConfig
  - retrieve IPv4 configuration information similar to the IPCONFIG.EXE utility. It demonstrates how to use the **GetNetworkParams** and **GetAdaptersInfo** functions.
- IPRenew
  - programmatically release and renew IPv4 addresses obtained through DHCP. This program also demonstrates how to retrieve existing network adapter configuration information.
- IPRoute
  - use the IP Helper functions to manipulate the IPv4 routing table.
- Ipstat
  - show IPv4 connections for a protocol. By default, statistics are shown for IP, ICMP, TCP and UDP.
- Netinfo
  - use the new IP Helper APIs introduced on Windows Vista and later to display/change address and interface information for IPv4 and IPv6.

Creating a Basic IP Helper Application

1. Create a new empty project.
2. Add an empty C++ source file to the project.
3. Ensure that the build environment refers to the Include, Lib, and Src directories of the Platform Software Development Kit (SDK).
4. Ensure that the build environment links to the IP Helper Library file *Iphlpapi.lib* and the Winsock Library file *WS2_32.lib*.
   - **Note** Some basic Winsock functions are used to return IP address values and other information.
5. Begin programming the IP Helper application. Use the IP Helper API by including the IP Helper header file.

#include <iphlpapi.h>

Some IP Helper functions

GetIpNetTable	GetIpAddrTable	GetIpForwardTable
GetIpStatistics	GetBestInterface	GetBestRoute
NotifyRouteChange	NotifyAddrChange	SendARP
GetUdpTable	GetIcmpStatistics	GetInterfaceInfo
GetNetworkParams	**GetAdaptersAddresses**	**GetAdapterInfo**
GetRTTAndHopCount	GetTcpStatistics	GetTcpTable
GetUdpStatistics		

- The **GetIpNetTable** function retrieves the IPv4 to physical address mapping table.
- The **SendARP** function sends an Address Resolution Protocol (ARP) request to obtain the physical address that corresponds to the specified destination IPv4 address.

GetAdaptersInfo Function (IPv4)

```
DWORD GetAdaptersInfo(
  __out    PIP_ADAPTER_INFO pAdapterInfo,
  __inout  PULONG pOutBufLen
);
```

```
typedef struct _IP_ADAPTER_INFO {
  struct _IP_ADAPTER_INFO *Next;
  DWORD           ComboIndex;
  char            AdapterName[MAX_ADAPTER_NAME_LENGTH + 4];
  char            Description[MAX_ADAPTER_DESCRIPTION_LENGTH + 4];
  UINT            AddressLength;                          //The length of the hardware address
  BYTE            Address[MAX_ADAPTER_ADDRESS_LENGTH];    //The hardware address
  DWORD           Index;
  UINT            Type;
  UINT            DhcpEnabled;
  PIP_ADDR_STRING CurrentIpAddress;     //Reserved
  IP_ADDR_STRING  IpAddressList;        //The list of IPv4 addresses
  IP_ADDR_STRING  GatewayList;          //The IPv4 address list of the gateway
  IP_ADDR_STRING  DhcpServer;
  BOOL            HaveWins;
  IP_ADDR_STRING  PrimaryWinsServer;
  IP_ADDR_STRING  SecondaryWinsServer;
  time_t          LeaseObtained;
  time_t          LeaseExpires;
} IP_ADAPTER_INFO, *PIP_ADAPTER_INFO;
```

```
typedef struct _IP_ADDR_STRING {
  struct _IP_ADDR_STRING  *Next;
  IP_ADDRESS_STRING       IpAddress;
  IP_MASK_STRING          IpMask;
  DWORD                   Context;
} IP_ADDR_STRING, *PIP_ADDR_STRING;
```

On Windows XP and later: Use the GetAdaptersAddresses function instead of GetAdaptersInfo.

GetAdaptersAddresses Function(IPv4/v6)

- The GetAdaptersAddresses function retrieves information for IPv4 and IPv6 addresses and returns this information as a linked list of **IP_ADAPTER_ADDRESSES** structures

```
ULONG WINAPI GetAdaptersAddresses(
_In_    ULONG Family,
_In_    ULONG Flags,
_In_    PVOID Reserved,
_Inout_  PIP_ADAPTER_ADDRESSES AdapterAddresses,
_Inout_  PULONG SizePointer );
```

AdapterAddresses [in, out] is a pointer to a buffer that contains a linked list of **IP_ADAPTER_ADDRESSES** structures on successful return.

IP_ADAPTER_ADDRESSES Structure

```
typedef struct _IP_ADAPTER_ADDRESSES {
  union {
    ULONGLONG Alignment;
    struct {
      ULONG Length;
      DWORD IfIndex;
    } ;
  } ;

  struct _IP_ADAPTER_ADDRESSES *Next;

  PCHAR                       AdapterName;
  PIP_ADAPTER_UNICAST_ADDRESS FirstUnicastAddress;
  PIP_ADAPTER_ANYCAST_ADDRESS FirstAnycastAddress;
  PIP_ADAPTER_MULTICAST_ADDRESS FirstMulticastAddress;
  PIP_ADAPTER_DNS_SERVER_ADDRESS FirstDnsServerAddress;
  PWCHAR          DnsSuffix;
  PWCHAR          Description;
  PWCHAR          FriendlyName;
  BYTE            PhysicalAddress[MAX_ADAPTER_ADDRESS_LENGTH];
  DWORD           PhysicalAddressLength;
  DWORD           Flags;
  DWORD           Mtu;
  DWORD           IfType;
  IF_OPER_STATUS                OperStatus;
  DWORD                         Ipv6IfIndex;
  DWORD                         ZoneIndices[16];
  PIP_ADAPTER_PREFIX            FirstPrefix;
  ULONG64                       TransmitLinkSpeed;
  ULONG64                       ReceiveLinkSpeed;
  PIP_ADAPTER_WINS_SERVER_ADDRESS_LH
FirstWinsServerAddress;
  PIP_ADAPTER_GATEWAY_ADDRESS_LH
FirstGatewayAddress;
  ULONG                         Ipv4Metric;
  ULONG                         Ipv6Metric;
  IF_LUID                       Luid;
  SOCKET_ADDRESS                Dhcpv4Server;
  NET_IF_COMPARTMENT_ID         CompartmentId;
  NET_IF_NETWORK_GUID           NetworkGuid;
  NET_IF_CONNECTION_TYPE        ConnectionType;
  TUNNEL_TYPE                   TunnelType;
  SOCKET_ADDRESS                Dhcpv6Server;
  BYTE
Dhcpv6ClientDuid[MAX_DHCPV6_DUID_LENGTH];
  ULONG                         Dhcpv6ClientDuidLength;
  ULONG                         Dhcpv6Iaid;
  PIP_ADAPTER_DNS_SUFFIX        FirstDnsSuffix;
} IP_ADAPTER_ADDRESSES, *PIP_ADAPTER_ADDRESSES;
```

IP_ADAPTER_UNICAST_ADDRESS

```
typedef struct _IP_ADAPTER_UNICAST_ADDRESS {
  union {
    struct {
      ULONG Length;
      DWORD Flags;
    };
  };
  struct _IP_ADAPTER_UNICAST_ADDRESS  *Next;
  SOCKET_ADDRESS          Address;          //The IP address for this unicast IP address entry
  IP_PREFIX_ORIGIN        PrefixOrigin;
  IP_SUFFIX_ORIGIN        SuffixOrigin;
  IP_DAD_STATE            DadState;
  ULONG                   ValidLifetime;
  ULONG                   PreferredLifetime;
  ULONG                   LeaseLifetime;
  UINT8     OnLinkPrefixLength;  //The length, in bits, of the prefix or network part of the IP address.
} IP_ADAPTER_UNICAST_ADDRESS, *PIP_ADAPTER_UNICAST_ADDRESS;
```

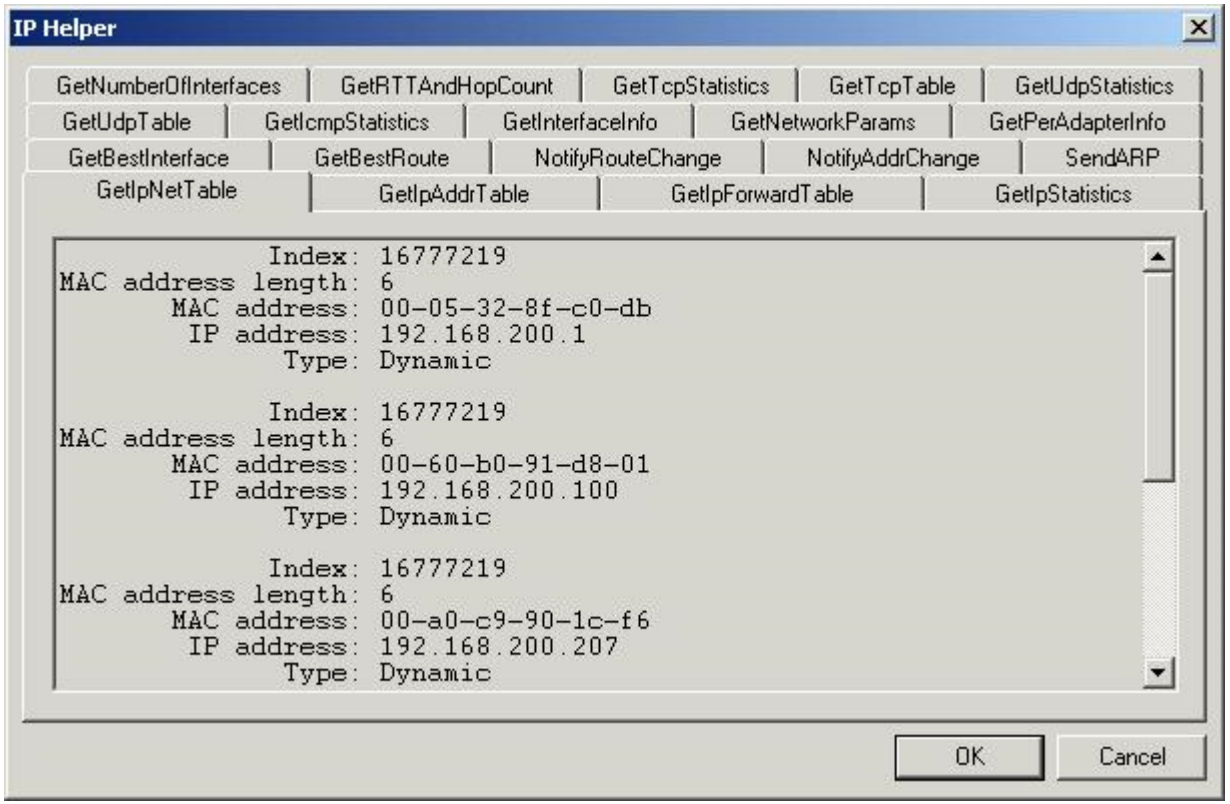
iphlpapi.h and winsock2.h

```
#include <winsock2.h>
#include <iphlpapi.h>
#include <stdio.h>
int main()
{
    return 0;
}
```

- Note: The Winsock2.h header file for Windows Sockets 2.0 is required by most applications using the IP Helper APIs. When the Winsock2.h header file is required, the #include line for this file should be placed before the #include line for the Iphlpapi.h header file.

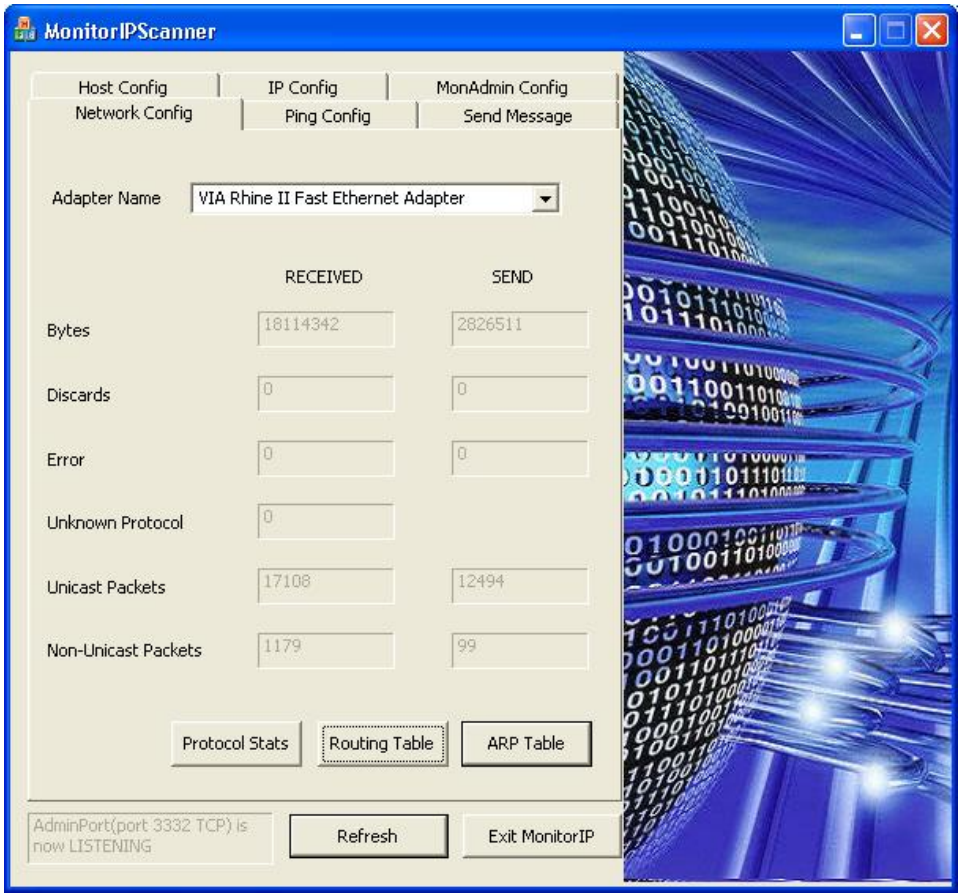
More on using IP Helper API's

- http://www.codeproject.com/KB/IP/IPHelper.aspx



MonitorIPScanner

- http://www.codeproject.com/KB/IP/Monitor_IP_Scanner.aspx



References

- RFC 791 - INTERNET PROTOCOL
- RFC 1071 - Computing the Internet checksum
- http://www.iana.org/assignments/protocol-numbers/
- http://www.iana.org/assignments/ip-parameters
- Chapter 8, IP : Internet Protocol, TCP/IP Illustrated, Volume 2: The Implementation, Gary R. Wright, W. Richard Stevens
- IPv6, http://technet.microsoft.com/en-us/network/bb530961.aspx
- Windows Sockets, http://technet.microsoft.com/en-us/library/cc940028.aspx
- IP Helper API, http://msdn.microsoft.com/en-us/library/aa366073%28v=VS.85%29.aspx
- http://www.codeproject.com/KB/IP/IPHelper.aspx
- http://www.codeproject.com/KB/IP/Monitor_IP_Scanner.aspx