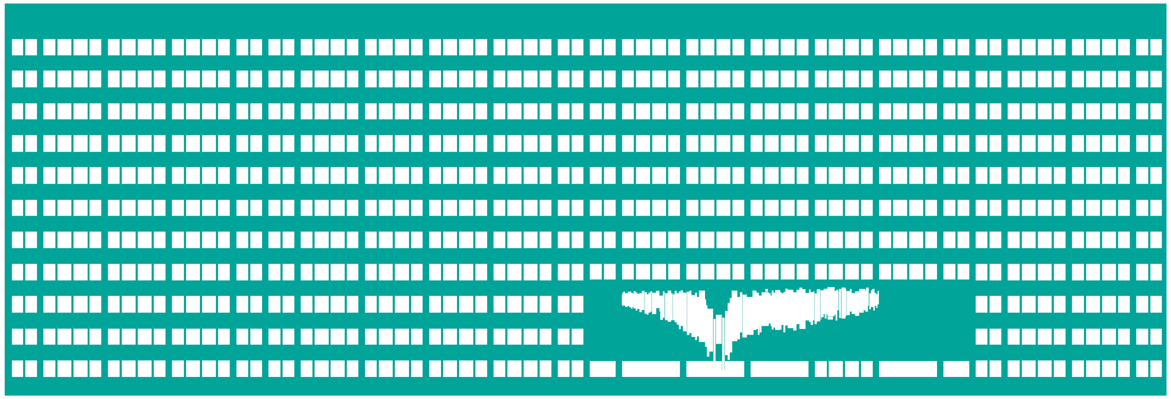
VSB TECHNICAL UNIVERSITY OF OSTRAVA | FACULTY OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE | DEPARTMENT OF COMPUTER SCIENCE

Protocols of TCP/IP Family analysis, NAT



Computer networks Seminar 7

ARP
Address Resolution Protocol

- IP to MAC address mapping
- If we need to find out the MAC address there is **ARP request** generated (broadcast). It contains the host IP address we need the MAC address of. The hosts with this IP address will answer with their MAC address (**ARP replay**).
- The source host of ARP request will save the result into ARP cache.
    - (station local cache keeps IP-MAC mapping)
- Following pair is also added to the **request**: < source IP, source MAC >, every computer watches all ARP broadcasts and updates its ARP cache

Using the command arp

- To see MAC-IP mapping table (Linux, Win)
  - Parameters:

-a	all records in arp cache
-s <*IP*> <*MAC*>	to insert static record manually
-d <*IP*>	to delete the record from arp cache

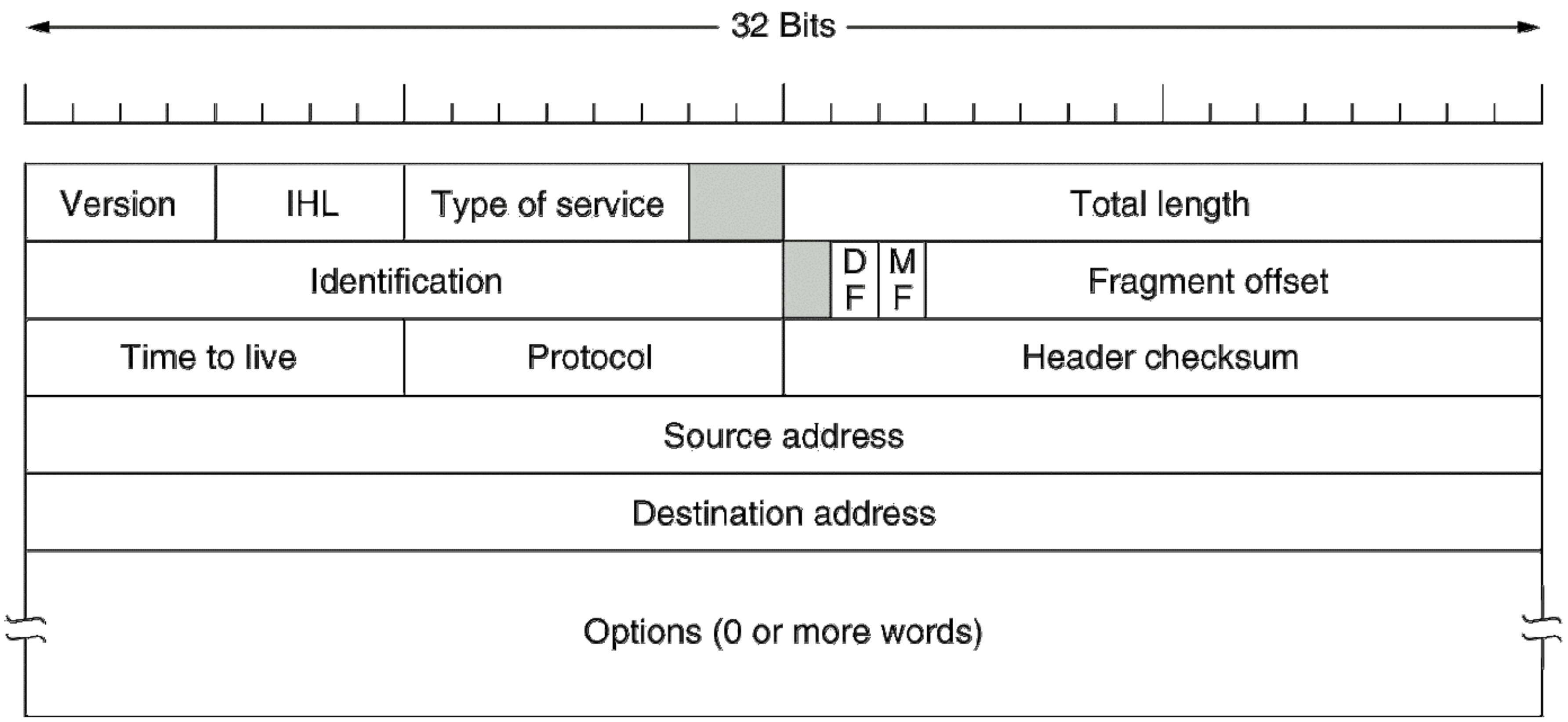
  - Parameters in Linux:

-v	detailed output
-n	numerical form outputs (without DNS)

- Example (Windows):

```
Rozhraní: 158.196.64.66 --- 0x10004
  internetová adresa    fyzická adresa         typ
  158.196.64.1          00-0a-f3-6e-bc-0a    dynamická
  158.196.64.137        00-0c-f1-3c-54-87    dynamická
```

IP header


32 Bits
Version
IHL
Type of service
Total length
Identification
DF
MF
Fragment offset
Time to live
Protocol
Header checksum
Source address
Destination address
Options (0 or more words)

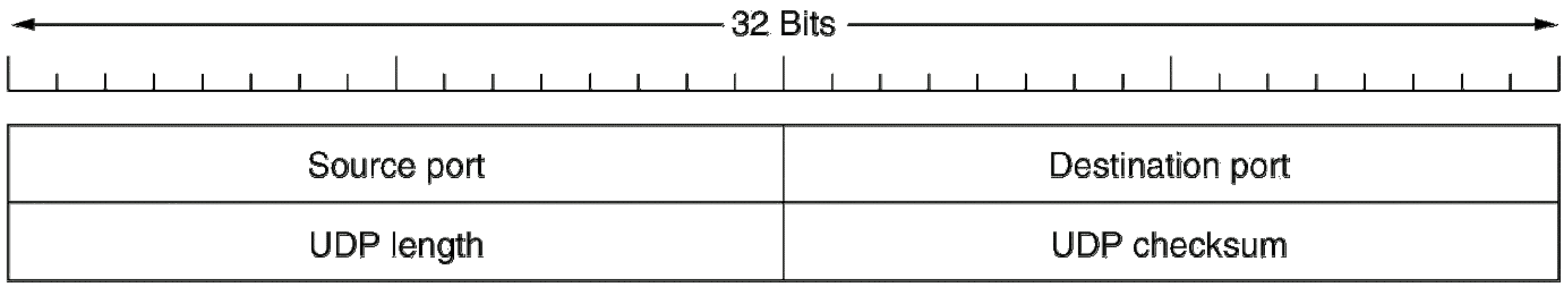
ICMP messages

- „Classic“ messages
  - **Echo request** , **echo reply**
  - **Destination unreachable**
    - (network, host, port, protocol unreachable, forbidden but neccessary fragmentation)
    - + administratively prohibited
  - **Time exceeded** (TTL=0 or time for re-fragmentation expired)
  - **Redirect**
  - **Parameter problem**
- Newer (but not always supported) messages
  - **Source quench** – request of target station to source to decrease the speed of generating messages (buffers overrun)
  - **Address mask request**, **Address mask reply** – finding interface subnet mask
  - **Router solicitation**, **Router advertisement**

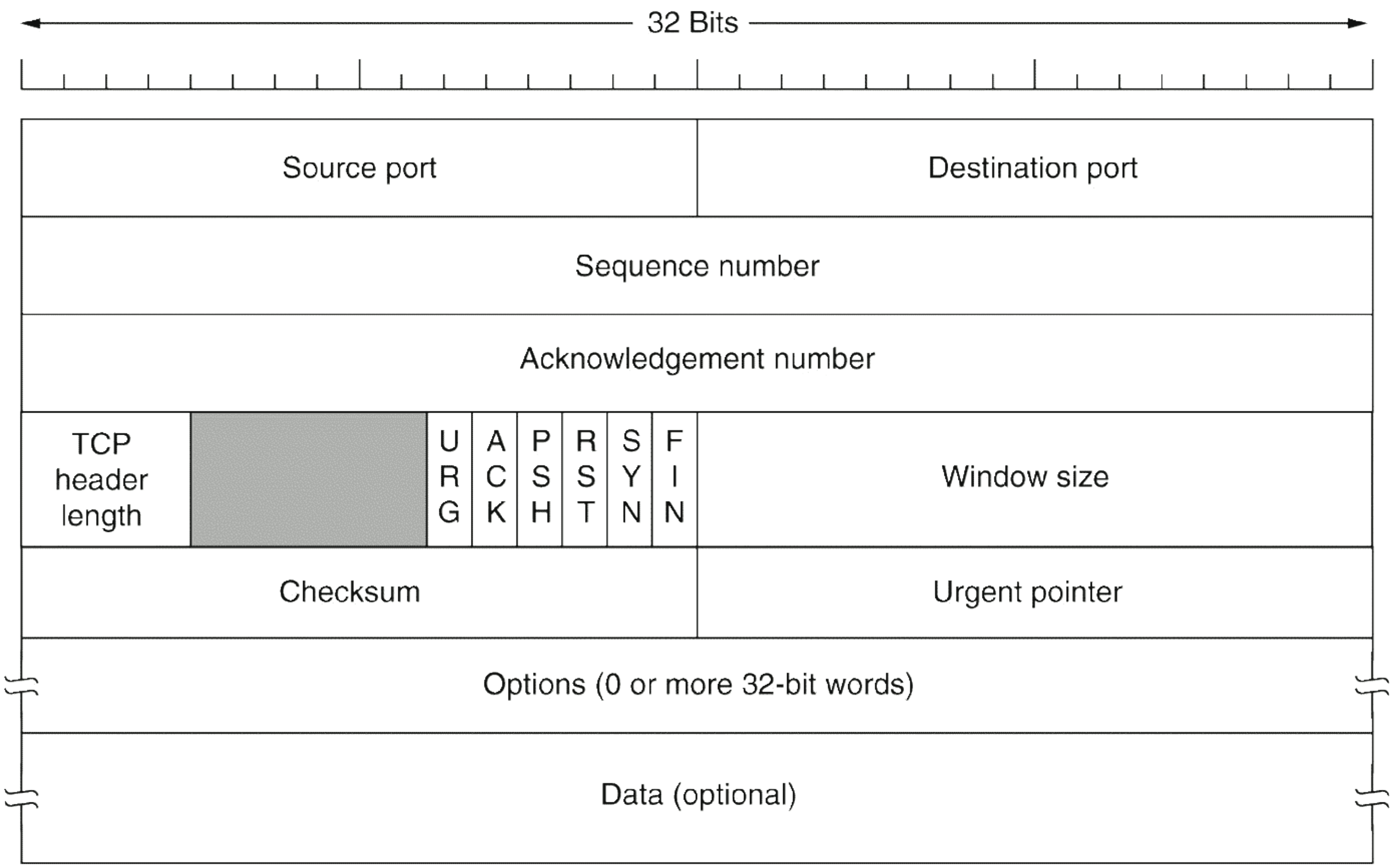
Ports

- Together with IP address identify particular process (service) on device in Internet
- 16bit (0-65535), separately for TCP and UDP
  - 0-1023: well-known
  - >1024 (4096) – registered ports, usually assigning of free ports by operating system
- Always target and source port

UDP header

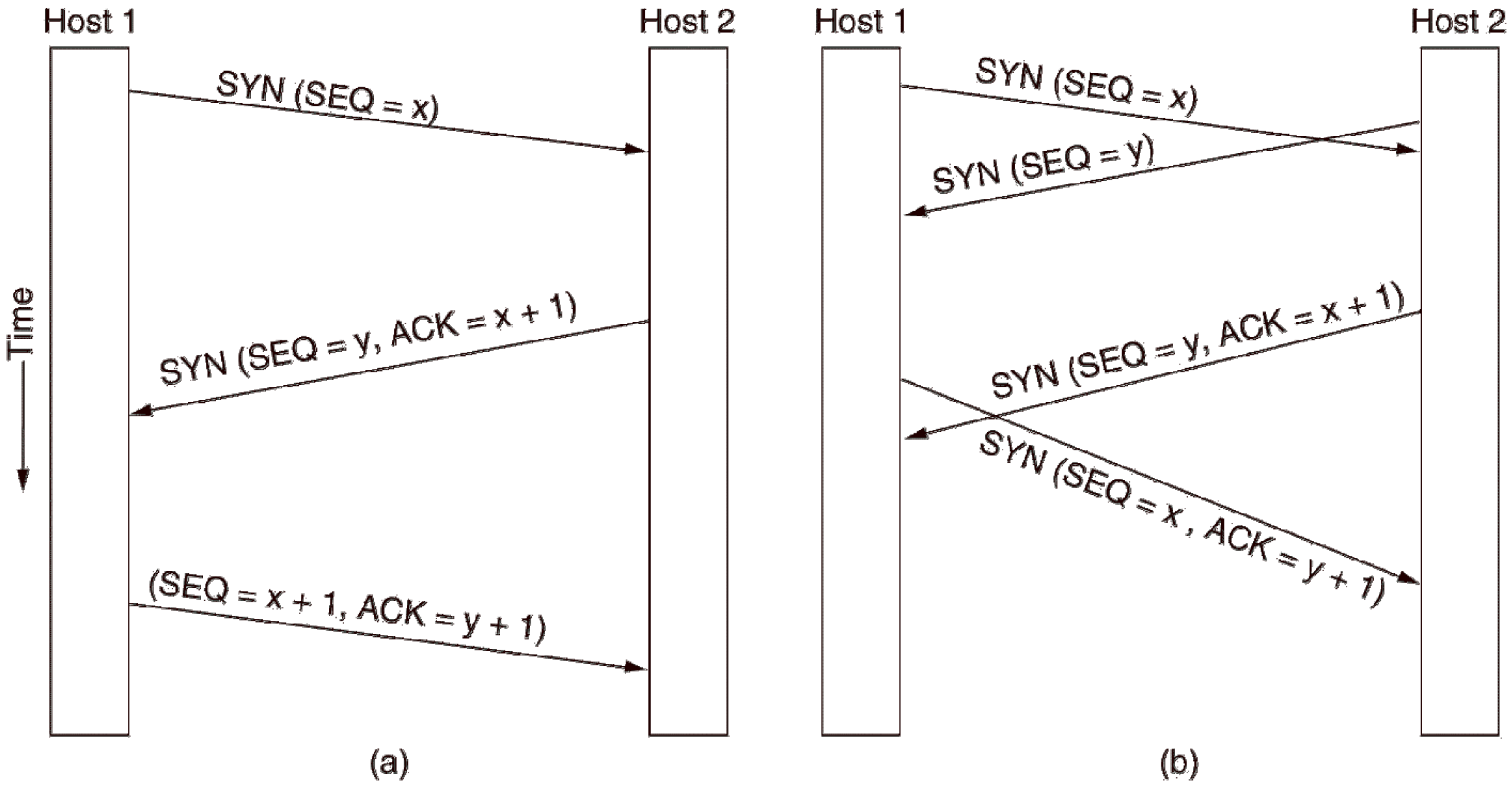

32 Bits
Source port
Destination port
UDP length
UDP checksum

TCP header

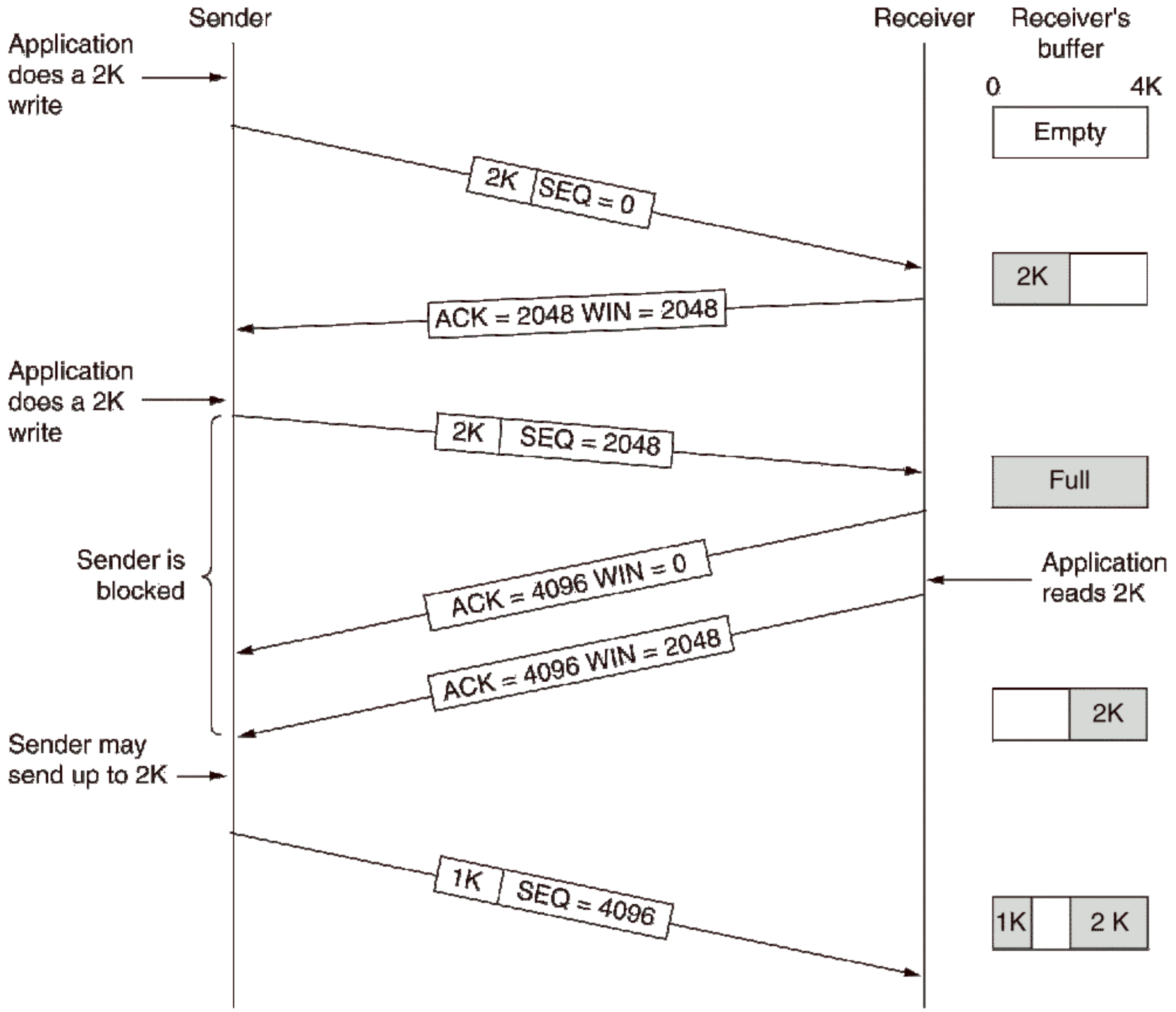


32 Bits
Source port
Destination port
Sequence number
Acknowledgement number
TCP header length
URG
ACK
PSH
RST
SYN
FIN
Window size
Checksum
Urgent pointer
Options (0 or more 32-bit words)
Data (optional)

Establishing TCP connection


Host 1
Host 2
Time
SYN (SEQ = x)
SYN (SEQ = y, ACK = x + 1)
(SEQ = x + 1, ACK = y + 1)
(a)
Host 1
Host 2
SYN (SEQ = x)
SYN (SEQ = y)
SYN (SEQ = y, ACK = x + 1)
SYN (SEQ = x, ACK = y + 1)
(b)

TCP connection – data flow control



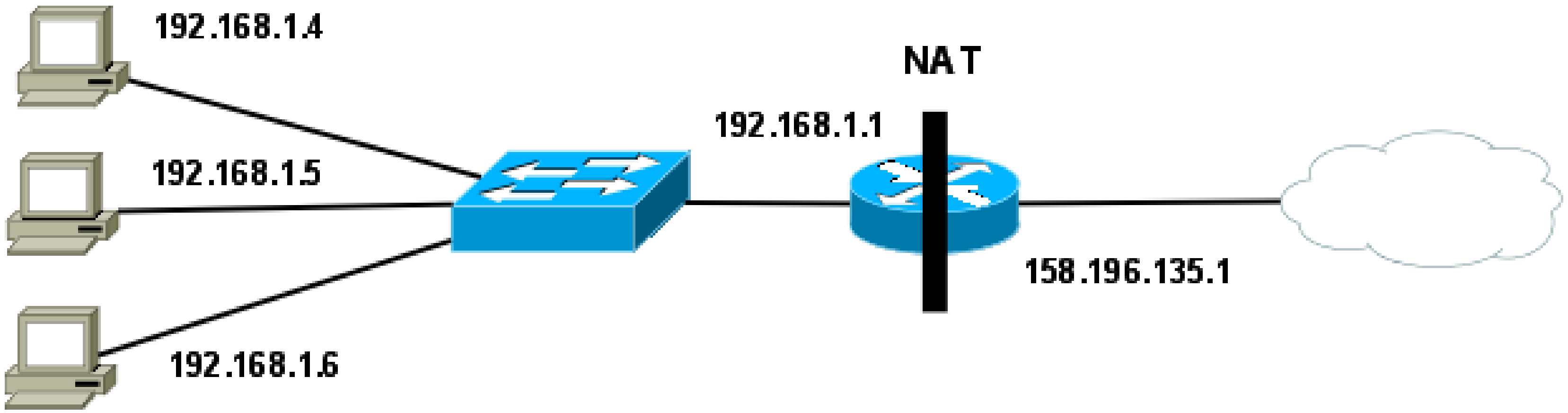
Using the command netstat

- List of active connections (Linux, Windows)
  - Parameters:
    - **`-a`** to see all connections and listening servers
    - **`-r`** to see routing table
    - **`-v`** detailed outputs
    - **`-n`** list of connection in numerical form (without DNS)
  - Parameters in Windows:
    - **`-p`** *`<protocol>`* just specified protocol (tcp, udp, …)
    - **-b** name of program which is using the socket
  - Parameters in Linux:
    - **`-u | -t | -w`** just given protocol (tcp, udp, raw, …)
    - **`-p`** PID and name of program using the socket

NAT

- Network address translation (translator)
  - Dynamic, static – IP→IP
  - Static translation
    - Translation table configured statically
  - Dynamic translation
    - Translation table is being created during operation
    - Addresses are borrowed from address pool
  - Typical example of translation
    - From inside private address to outside public address

Example of translation table using ports



Source IP	Source port	Source IP	S. port
192.168.1.4	2345	158.196.135.2	2345
192.168.1.5	4589	158.196.135.2	4589
192.168.1.4	5678	158.196.135.2	5678
192.168.1.6	5678	158.196.135.2	5679

NAT in IOS

- Specifying inside and outside interface
  - Inside: **`(config-if)# ip nat inside`**
  - Outside: **`(config-if)# ip nat outside`**
- Defining the addresses WHICH will be translated (typically private addresses)
- Defining the addresses TO WHICH it will be translated (typically public addresses)
- Putting it all together

Static NAT

- Address translation:
  - **`(config)#ip nat inside source static`** *<local_IP> <global_IP>*

- Address translation (using specified L4 port):
  - **`(config)#ip nat inside source static`** **`{tcp|udp}`** *<local_IP> <local_port> <global_IP> <global_port>*

Dynamic NAT – Defining the addresses

- Defining address pool
  (it means TO WHAT I am translating):
  - **`(config)# ip nat pool`** <*NAME*> <*start_IP*> <*stop_IP*> **`netmask`** <*mask*>
    - Ex.: `ip nat pool MyNATPool 20.0.0.1 20.0.0.100 netmask 255.255.255.0`
- Specifing addresses to be translated – using ACL (it means WHAT is to be translated):
  - **`(config)#access-list`** <*ACL number 1-99*> **`permit`** <*IP*> <*wildcard*>
    - Ex.: `access-list 1 permit 10.0.0.0 0.0.0.255`

Dynamic NAT

- Translation to addresses from pool:
  - **`(config)# ip nat inside source list`** <*ACL number*> **`pool`** <*NAME*> [**`overload`**]
    - Ex.: ip nat inside source list 1 pool MyNATPool overload
- Translation to the address of outside interface:
  - **`(config)# ip nat inside source list`** <*ACL number*> **`interface`** <*interface name*> [**`overload`**]
    - Ex.: ip nat inside source list 1 interface fa0/1 overload

NAT – seeing translation table

- To see translation table
  - **`#sh ip nat translations`**
- To clear translation table:
  - **`#clear ip nat translations *`**
- Timeout of records in the table:
  - **`(config)# ip nat translations timeout`** <seconds>
  - **`(config)# ip nat translations icmp-timeout`** *<seconds>*
- NAD debugging
  - **`#debug ip nat`**

NAT – assignment

- Interconnect 3 routers in a line (chain)
- Connect PC to each router
- Router in the middle simulates the network with the public addresses (all its interfaces use public addresses)
- PCs connected to the side routers are in private network and side routers realize the NAT