Technology Transfer Partners

An Introduction to IP
(with companions UDP and TCP)

Joe Doupnik
jrd@netlab1.net
jdoupnik@microfocus.com
Mindworksuk and Micro Focus
Prof (ret.) Univ of Oxford

UNIVERSITY OF OXFORD

MICRO FOCUS

How the IP matter started, then grew and grew

- Back in olden days we communicated one to one, using a private comms pathway between two fixed points. Two tin cans and a string.
- Time passes and we could use phone lines to reach many remote points. Each connection was isolated from others. Still point to point but we did have phone books (a DNS predecessor) to help people choose numbers.

- Later local broadcast networks (LANs) arose where many stations shared a comms channel. Thus a need to identify stations so that a receiver could pick out traffic intended for itself from all the competition.
- Technology advanced, the Internet arrived connecting lots and lots of points through “router” boxes. Much traffic to sort and send to the right places far and wide. Stations needed globally unique identifiers, which evolved into IP numbers and DNS names.

Internet Protocol, identifying end points

- On a broadcast channel (LANs etc) numbering machines as 1, 2, 3 etc is fine for a small local network, but there was that Internet thing arriving. The IETF governing body decided that a global 32 bit/4 byte numeric address would *surely* be more than adequate. Oh yes, add a (DNS) name to number converter to placate humans.
- So far so good. But the problem is more complicated. Two stations don't own the comms channel so they can't just send bytes when they want to. Other stations add bytes, thus separating traffic becomes a problem.
- Challenges are knowing the beginning and end of long messages, whether the data arrived safely, and how to send to places far far away.
- A major consideration is knowing how much traffic the network and receivers can carry at any moment. Blast & pray tactics are not good.
- Sigh, complexity. This is another case of no good deed goes unpunished.

Still more considerations/fine print

- Traffic needs boundaries which separate one session's bytes from other competing bytes on the wire. That was solved by grouping outgoing bytes into bundles, named "**packets**", which appends the payload bytes to an **addressing header**.
- That was a significant step. Packets & addresses, thus sharing the comms media became realistic. But the transport media part also presented its own challenges. How to know when multiple stations try to transmit at the same time with resulting garbles, etc (see presentation Ethernet for Professionals). And about those packets, how large could they become without net domination ensuing?
- Realistic memory consumption by lan adapters and relay agents/routers indicated small packets, ~1.5KB or so, for most active networks.
- Also puzzling is knowing when a multi-packet data stream starts and stops. Packets could help because they have a stable structure and could add some header status bits such as start/end and even sequence numbers.

The IP story thus far

- The IP v4 packet uses addresses (From & To) four bytes long, 4 billion values each. Pity the poor routers. Once that busy work began then many other transport nuances were added. Feeding time at the zoo.
- There was an IP protocol version number to permit evolution over time, a header length value to permit adding sundry option fields, a checksum over the IP header to detect errors (alas, just IP header, not over the payload).
- Then appeared a time to live (hop count) field to prevent routers circulating packets in endless loops. Add a priority indicator for routers to handle different kinds of traffic (streaming etc).
- A helpful soul said a field ought to indicate the kind of data in the payload so the receiver could properly deal with it. That was clever foresight and permitted UDP and TCP headers beneath the IP header.

IP protocol header format

Over time the IP v4 header evolved into 14 fields, 20+ bytes, as shown below

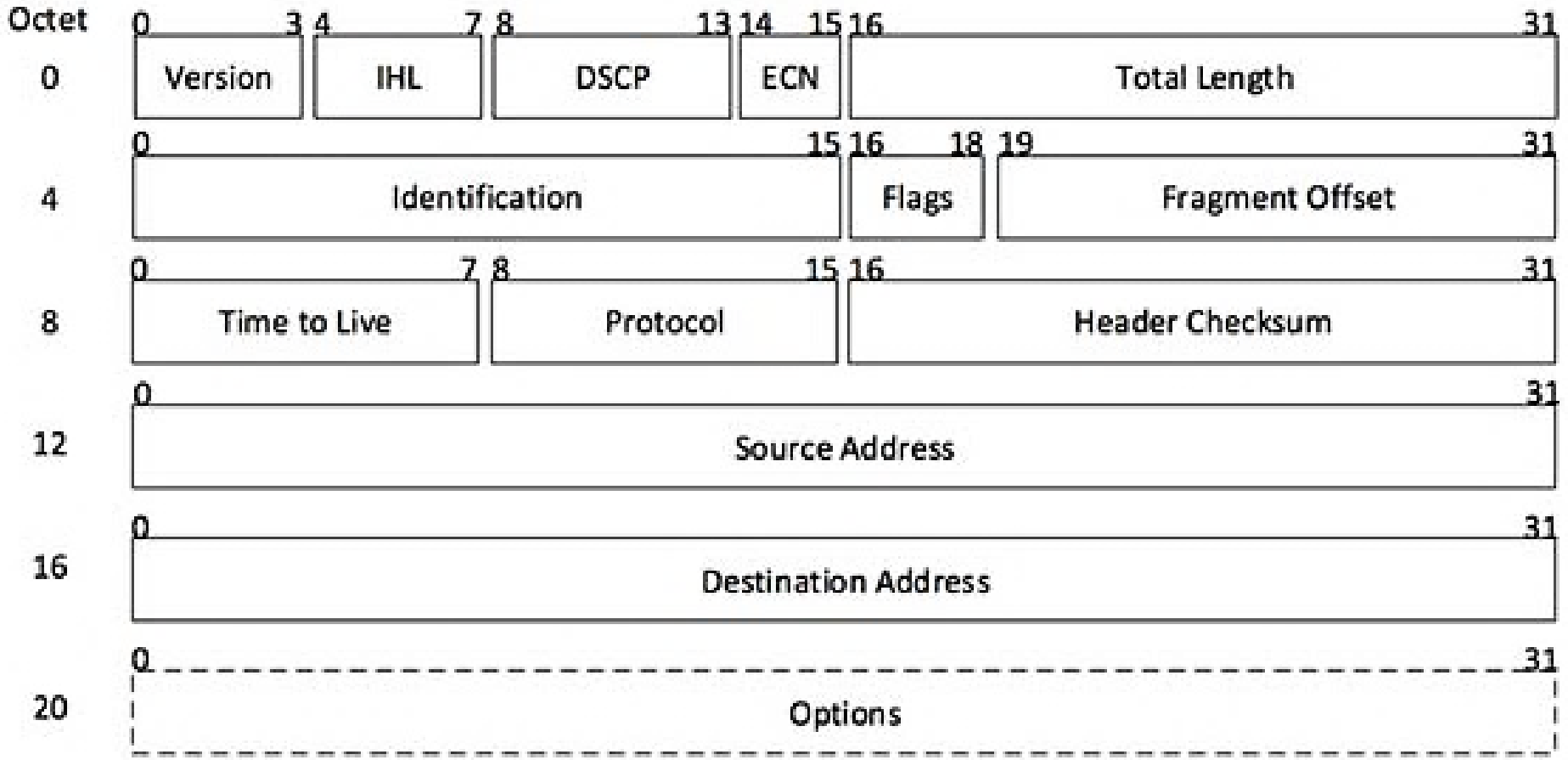


Diagram and its text are from www.tutorialspoint.com

Description of the IP fields

- **Version** – Version no. of Internet Protocol used (e.g. IPv4).
- **IHL** – Internet Header Length; Length of entire IP header.
- **DSCP** – Differentiated Services Code Point; this is Type of Service.
- **ECN** – Explicit Congestion Notification; It carries information about the congestion seen in the route.
- **Total Length** – Length of entire IP Packet (including IP header and IP Payload).
- **Identification** – If IP packet is fragmented during the transmission, all the fragments contain same identification number. to identify original IP packet they belong to.
- **Flags** – As required by the network resources, if IP Packet is too large to handle, these 'flags' tells if they can be fragmented or not. In this 3-bit flag, the MSB is always set to '0'.
- **Fragment Offset** – This offset tells the exact position of the fragment in the original IP Packet.
- **Time to Live** – To avoid looping in the network, every packet is sent with some TTL value set, which tells the network how many routers (hops) this packet can cross. At each hop, its value is decremented by one and when the value reaches zero, the packet is discarded.

Rest of the IP header fields

- **Protocol** – Tells the Network layer at the destination host, to which Protocol this packet belongs to, i.e. the next level Protocol. For example protocol number of ICMP is 1, TCP is 6 and UDP is 17.
- **Header Checksum** – This field is used to keep checksum value of entire header which is then used to check if the packet is received error-free.
- **Source Address** – 32-bit address of the Sender (or source) of the packet.
- **Destination Address** – 32-bit address of the Receiver (or destination) of the packet.
- **Options** – This is optional field, which is used if the value of IHL is greater than 5. These options may contain values for options such as Security, Record Route, Time Stamp, etc.

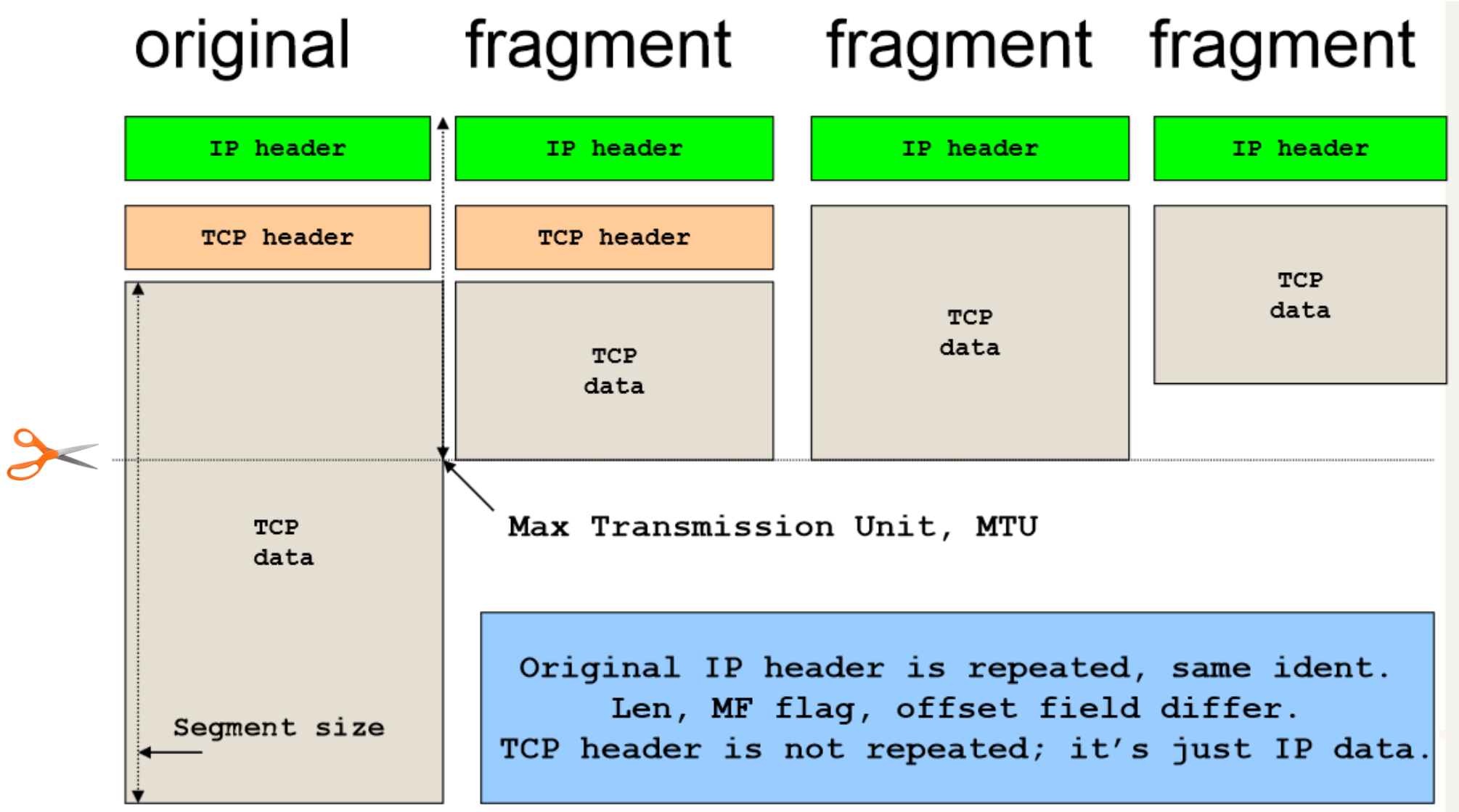
Fortunately for us, at this point designers were enticed to tinker with other protocols, leaving IP to work in peace.

Fragmentation and Humpty Dumpty

The party was nearly ruined when worries appeared about some media not being able to carry large packets. Enter the fragmentation saga. The mechanics are actually simple. An illustration follows.

- Snip a packet to the longest known acceptable value. Set the **MF** (**M**ore **F**rags follow) bit in the IP header and initially set the IP **Frag Offset** field to be 0.
- Next, copy this IP header to make a new packet, add as much remaining payload as possible, set Frag Offset for the start byte. Repeat for any remaining payload.
- In each IP header the MF bit is set except at the end, and the Frag Offset field says where this particular payload snippet fits back into the original.
- The IP Identification field is the same for each of the original's fragments.
- MF 0 and Frags Offset 0 means this packet is the whole thing, not fragmented.
- The receiver reassembles payload snippets with the MF, Frag Offset and Ident fields, employing sticky tape and jigsaw puzzle skills.

IP fragmentation illustration



From presentation TCP from the wire up on netlab1.net

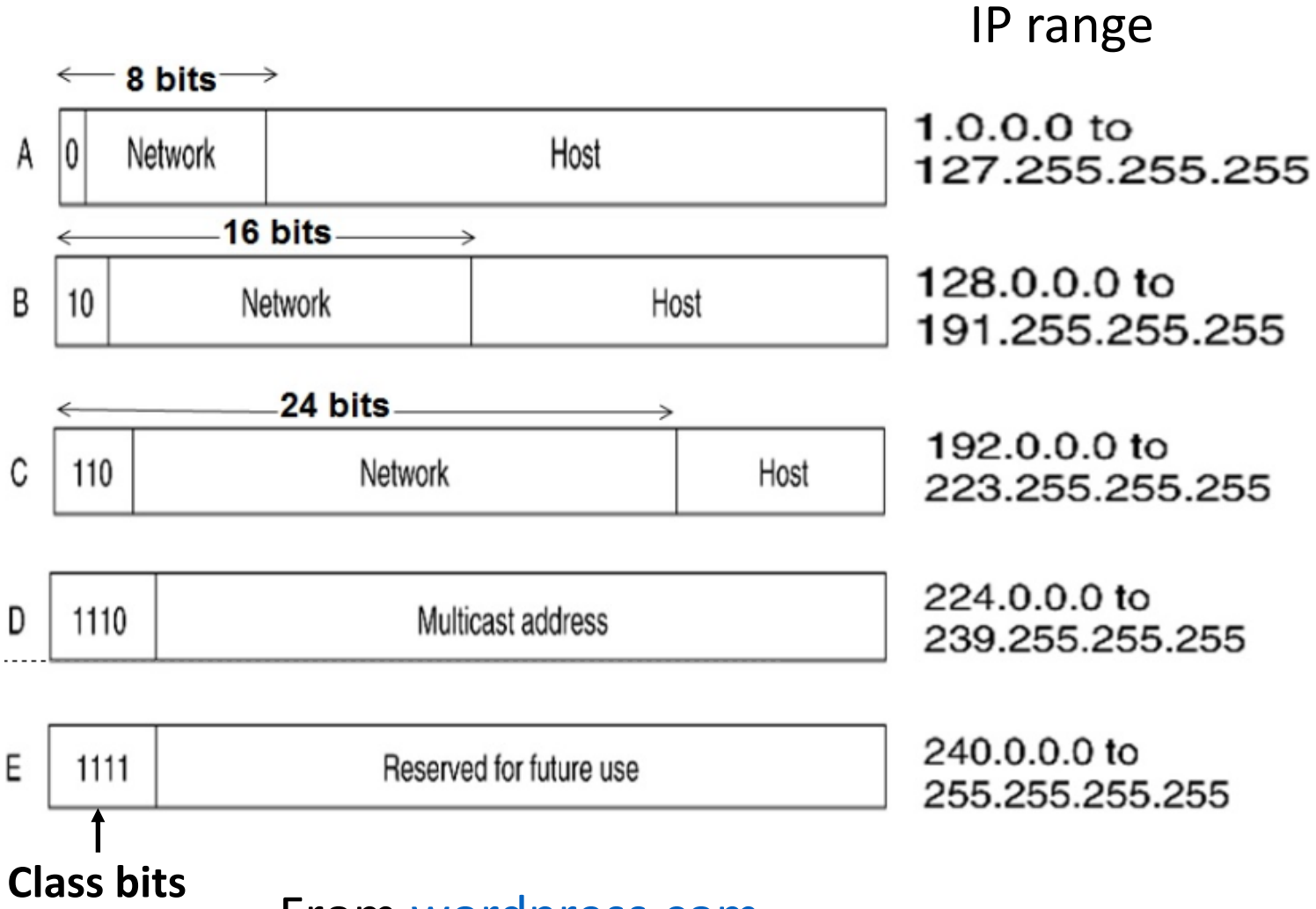
Fragmenting the fragments, for gosh sakes

- Interestingly, if fragmentation occurs along the network and if some fragments are still found to be too large for the next hop just repeat the snip/copy header/set MF/Frag Offset steps upon already fragmented items, thus forming more fragments which follow the same rules.
- This is rather clever logistics, needing only the MF bit, Frag Offset and Identification fields to denote the pieces and permit correct reassembly by the ultimate receiver.
- Clearly, fragmentation and reassembly require resources so we try to avoid fragmentation when possible by the sender limiting the payload length in each outgoing packet (Maximum Transmission Unit, **MTU**).
- Discovery of what the network allows is a problem for another discussion. Media (Ethernet etc) sets one limit, typically 1.5KB.

IP satnav work: routing problems

- A short story here. Way back in time we used 1, 2, 3 etc for local machines and life was simple. Then this IP stuff appeared with 32 bit addresses and the Internet was spread all over the globe. We needed a map and regional guides along the way. Interestingly, IP addressing did help with routing complexity.
- Routers knit local networks into one giant tree. If only we had direction signs to know when a router is needed rather than just transmit on the local wire directly to a local receiver as in olden times. Thus there is the situation of many networks, each containing many hosts, plus routers acting as tour guides.
- Internet authorities decided to assign each address to a “class” where leading bits of an IP identify different networks and trailing bits hosts on a net. This partitioning reduced the number of table entries in routers to just networks. A picture follows.
- Routers deal with just networks rather than a list of 4 billion individual addresses. Each router knows its child/dependent networks and the address of its parent closer-to-root router for other networks. This is parental or tree-like.

IP address intrinsic Classes



From wordpress.com

Classes steal top/leading net bits to denote the class.

Class A is for the big outfits.
Class B for many hefty sites.
Class C for small outfits.
Which leaves us at-home folks puzzled (which is resolved shortly).

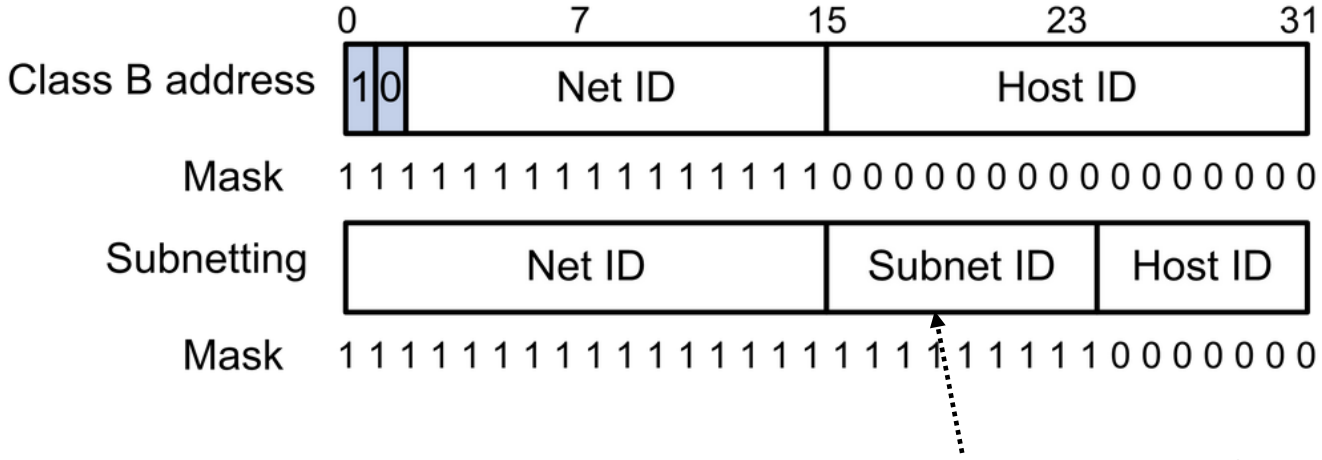
Classes intrinsically indicate which bits define networks.

Class was a good idea at the time, but…

- The “but” is Class divisions result in large numbers of hosts seeming to be on the same broadcast network. That would produce a horrid traffic congestion problem on our networks. But it does ease top level Internet clerical work and routing became somewhat easier. What to do?
- Enter knight in shining armour to rescue our IP damsel in distress, and allow a given IP address to be re-divided by routers into many smaller local networks.
- The rescue method is locally allow some upper part of the “hosts” field to be reassigned as “network” and thus add local nets and reduce the number of hosts on each net to a reasonable number, with far less traffic on each wire. Routers implement this subdivision. ISPs use this method for customers, and customers also use it for even finer local divisions.
- Thus a human managed local datum is associated with IP numbers, a **network mask,** to indicate on this network which IP bits are networks and which are hosts. Introduce term **CIDR**, Classless Internet Domain Routing. Democracy?

A ~~face~~ network mask example

Figure 2 Subnetting a Class B network



From www.h3c.com

Kidnapped, rebranded as local network bits

CIDR notation comes into play here. We might say an IP assignment is 67.22.37.42/25 to mean 25 leading 1's in the network mask for this machine. The mask overrides normal Class representation.

The mask has 1's for network bits and 0's for host bits.

A *subnet* steals Class *host* bits to denote *networks*, as we do at local sites when splitting traffic over many local wires joined by routers.

A *supernet* aggregates adjacent networks to steal Class *net* bits to denote local *hosts*, say with several contiguous Class C nets.

Routing decisions using the network mask

- A station sending a packet needs to decide whether the recipient is on *this network* or not.
- If *yes* then a destination media address (Ethernet etc) is needed to send directly on the wire.
- If *no* then the media address of an attached router is needed to forward the packet to another network. [Routers have higher level routers to help.]
- That *yes/no* decision is made with a little binary math, and the media address is found by two ARP packets. Caution: a math Illustration follows.

- IPs 10.0.0.0/8, 172.16.0/12, 192.168.0.0/16 are defined as non-routable.
- A host field of all 0's means "this network", not a host; all 1's is broadcast.

Network mask math in action, XOR + AND

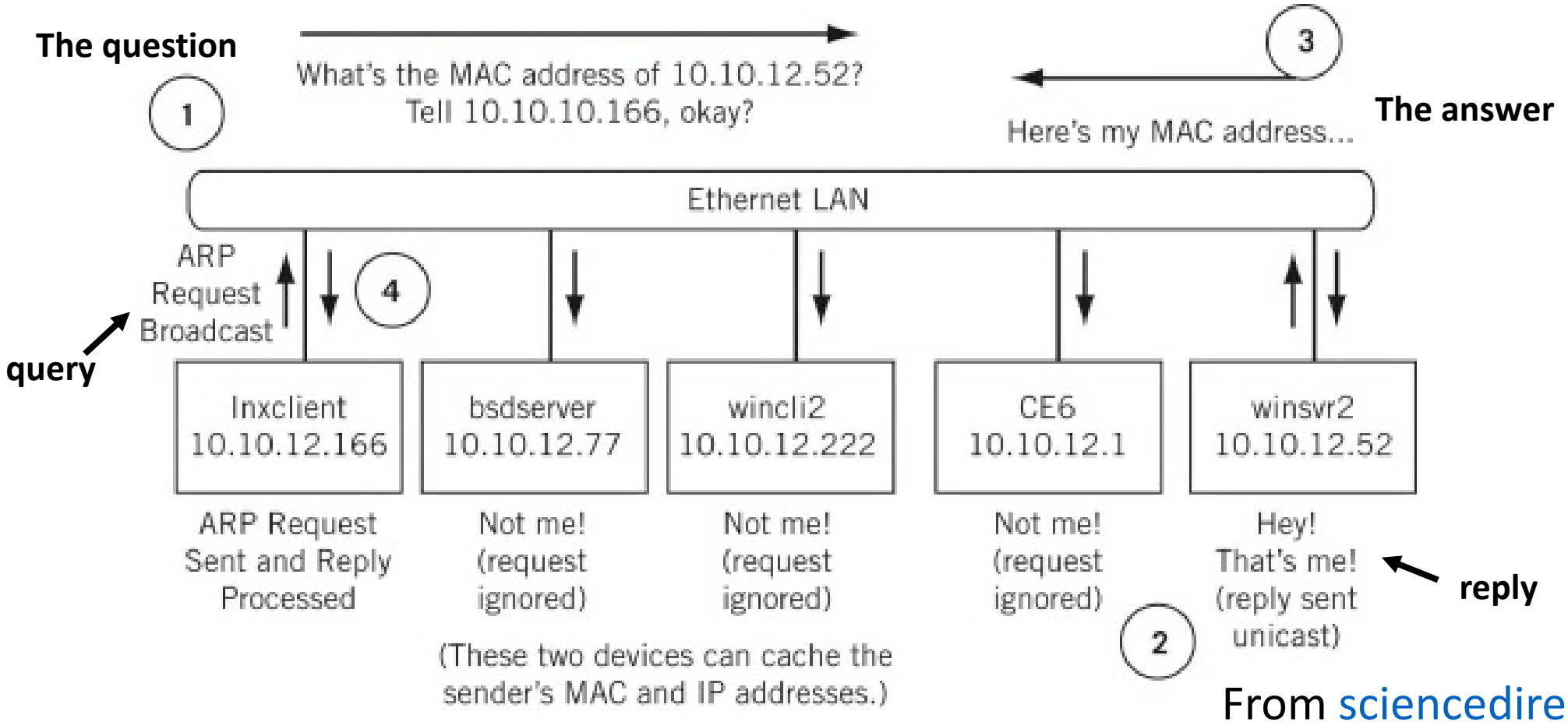
Network same/different calc

```
10000000 00111011 00100111 00000010  their_IP 128.59.39.2
10000001 01111001 00000001 00101011  my_IP 129.123.1.43
                                      XOR column at a time
00000001 01000010 00100110 00101001  -> differences
                                      (1 = different, 0 = same)
   AND column at a time with netmask to show only network diffs
11111111 11111111 11111111 00000000  netmask 255.255.255.0
         (networks)          (hosts)
 (mask is transparent)     (is opaque)
00000001 010000010 00100110 00000000 -> masked differences

Non-zero final result means the IP networks are not the
same and thus we must use a gateway/router to talk.
   If (((their_IP ^ my_IP) & netmask)) != 0)
        use_gateway();          /* long distance */
   else
        go_direct();            /* on same wire */
```

From presentation TCP from the wire up

ARPing on the network: who owns this IP?



From [sciencedirect.com]

LAN adapters join the personal name game, with media addresses, doing address recognition within the adapter's hardware, not software.
Linux command arp -a shows a station's list of cached replies

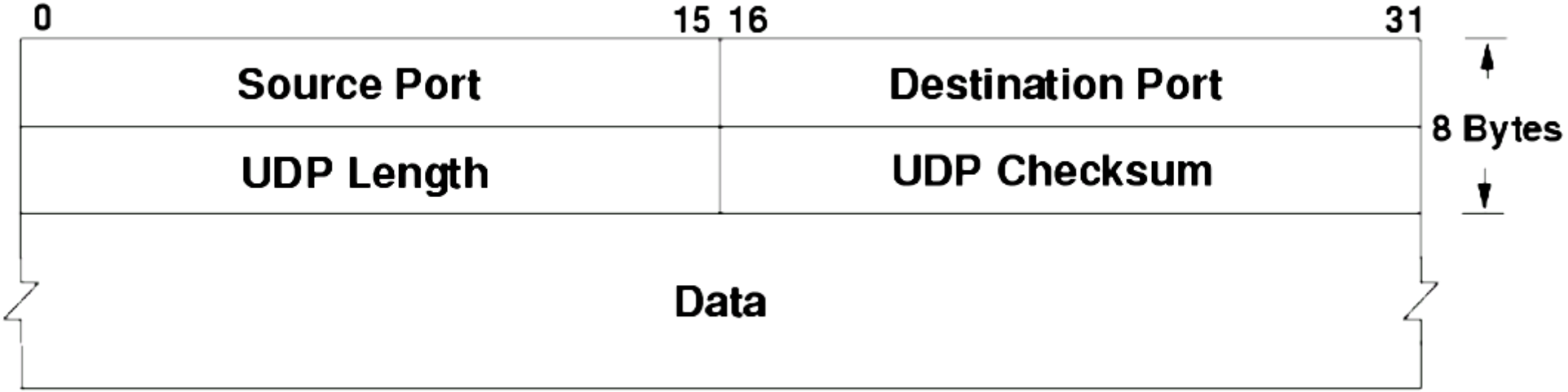
Isn’t the IP header enough to network?

Close, but not a yes. The situation is complicated.

- IP deals with just routing traffic from machine A to B.
- IP and UDP are “send and forget” processes, ignoring consequences. Hope, but no feedback, no delivery verification, no traffic knowledge.

- UDP and TCP Ports help source and destination machines use the correct application/session in a box. Building # (IP), then room # (Port). Each session keeps a 5-tuple: src IP & Port, Protocol, dest IP & Port.
- TCP uses ACK feedback and data sequence numbering for robustness and verification of transfers. It learns and adjusts to network carrying capacity. TCP employs effective measurement and control mechanisms.

UDP header components



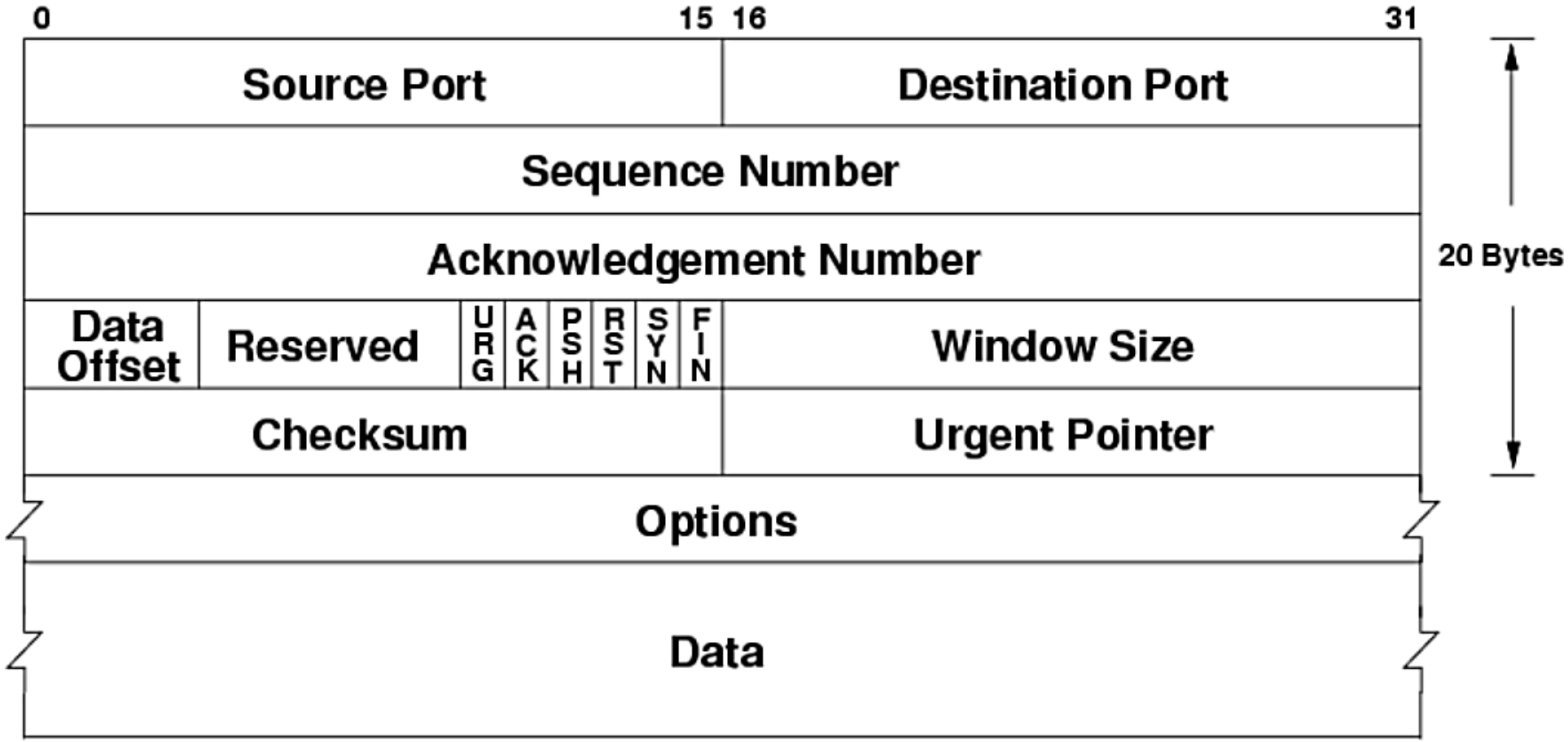
From wikimedia

- This is simple. Two 16 bit port numbers (to identify sending and listening processes in each machine), a length field, and thankfully a checksum over both the UDP header and the payload data.
- Length is only 16 bits/64KB.
- Why a source port? To help identify individual parallel sessions. (5-tuple)
- UDP or TCP headers immediately follow the IP header, and then real data.

A few TCP basics, dense reading material

- TCP presumes a transfer may be long, many packets in a row to carry the data. Both ends learn if the data arrived intact because TCP provides end to end feedback for reliability & robustness, versus IP & UDP send-and-forget. TCP also marks the start and end of a long data stream. Each side reports its receiver buffer current free space, to help avoid overloads.
- TCP requires the receiver acknowledge arrival of every data byte by returning an ACK packet having the sequence number of the last data byte received in-sequence, +1.
- If that ACK does not arrive “soon” then that’s a loss and a packet will be resent.
- TCP will repeat lost packets. Nearly all loss is from overloading buffers in relay boxes along the pathway, known as *congestion loss*. Clever heuristics deal with this.
- Thus arriving ACKs basically pace new transmissions, and TCP adapts to the network. An estimated network capacity limit, **cwin** bytes in flight, is computed continuously.
- Timing and success/failure information allows TCP to vary its sending rate to reduce overloading the network or the receiver, and TCP verifies delivery.

TCP header components



From wikimedia

Goodness, there are the two port numbers, a checksum, and a lot more.
Those extra fields are for feedback and heuristics control information.

TCP operational flag bits

- SYN (synchronize): Packets that are used to initiate a connection.
- ACK (acknowledgment): Packets that are used to confirm that the data packets have been received, also used to confirm the initiation request and tear down requests
- RST (reset): Signify the connection is down or maybe the service is not accepting the requests
- FIN (finish): Indicate that the connection is being torn down. Both the sender and receiver send the FIN packets to gracefully terminate the connection
- PSH (push): Indicate that the incoming data should be passed on directly to the application instead of getting buffered
- URG (urgent): Indicate that the data that the packet is carrying should be processed immediately by the TCP stack

From howtouseLinux.com

We spot the start/SYN and end/FIN flags, ACK plus other admin types .

Room for another? Permission to come aboard

- A TCP transmission is limited by a) free space in receiver's buffer, b) data amount in sender's buffer, and c) the estimated network capacity **cwin** as the number of bytes allowed in flight (those sent but not yet ACK'd). A *slow start threshold* marker is used. The sender computes cwin from ACK measurements. Cwin is also spelled CWND.
- Set cwin to 1 and the threshold to unlimited. Start sending. An arriving ACK permits sending a replacement packet plus an additional packet, a cwin doubling tactic. Repeat cwin doubling up to the threshold. This "slow start" phase is really not slow.
- After a loss or timeout reset cwin to 1 and set the *slow start threshold* to half the last cwin. Halving is an exponential decrease, a rate necessary to avoid instability. See Control Systems theory and having poles in the right hand plane.
- When cwin exceeds the *slow_start_threshold* grow cwin more slowly. An ACK adds one unit to cwin after cwin ACKs, thus grow cwin by 1/cwin per ACK. Slow growth is to gently learn current network capacity. Cwin credits accumulate even if not used.
- A dynamic timer ensures that losses are replaced promptly.

An algorithmic sketch for programmers

Slow Start State: fill the pipe quickly, get the ACK clock ticking.
Set slow start threshold (safe amount of in-flight) to huge
Set congestion window cwin to 1 packet, allow sending
cwin is the network's learned capacity window, a limit on in-flight amount

Arrival of an ACK for data changes cwin:
If slow start threshold is not yet been reached, grow cwin quickly

```
    cwin <- (cwin + 1)          "slow start", fill the pipe, get ACKs flowing
else                            use congestion avoidance tactic
    cwin <-  (cwin + 1/cwin)    gently probe the net for new capacity
```

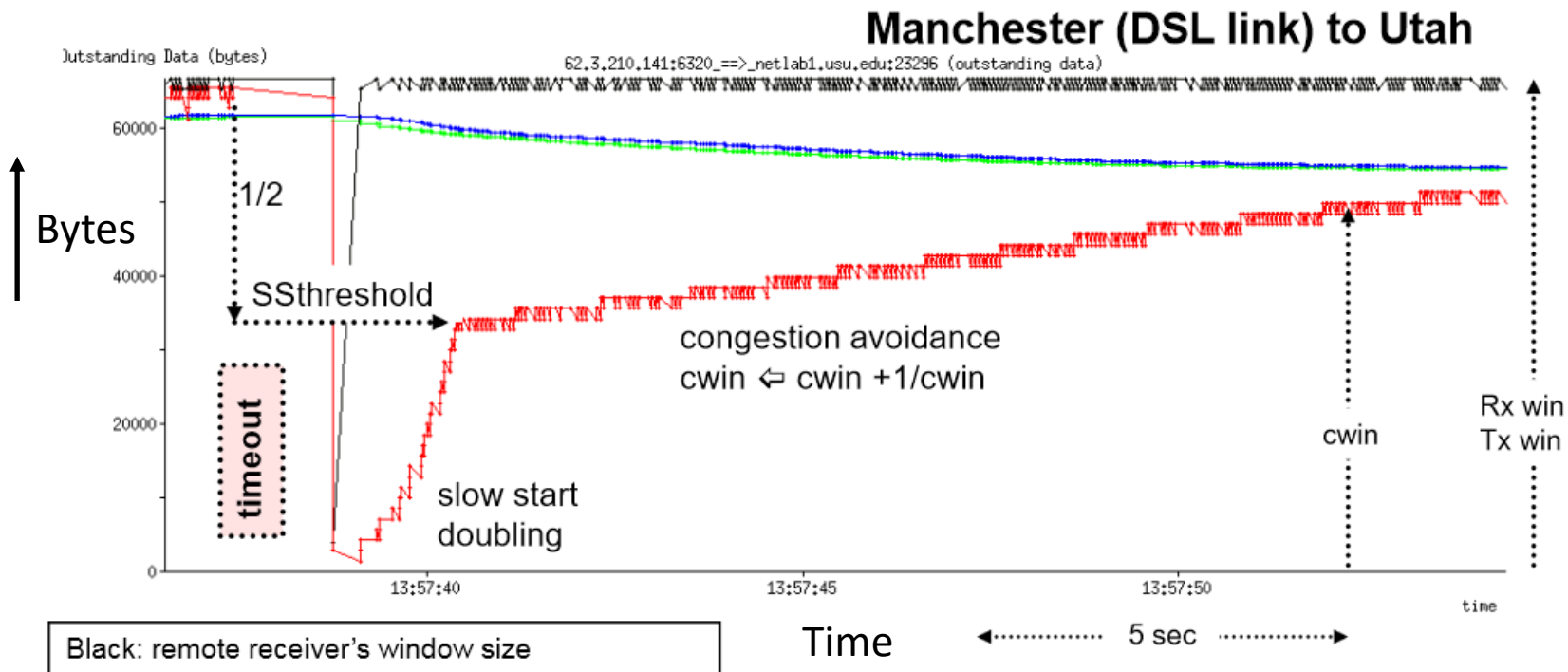
If timeout (packet loss) then

```
slow start threshold <- cwin/2   set fallback safe amount of in-flight data
cwin = 1                         restart slowly, re-learn the net
resend all old data
```

From presentation TCP, a network process

TCP recovering from loss, relearning net capacity

Detail of a loss recovery interval, illustrating slow start and 1/cwin steps
Example starts with packet loss at 64KB full network capacity



Duration at a step is set by waiting for ACKs, "net window" cwin is full

cwin is TCP's congestion window limit for sending. It limits data sent but not yet ACK'd. It grows from successes.

This diagram starts with the blast, a timeout, "slow start" refilling, then the slow growth testing of the network. Tic marks are arriving ACKs.

From presentation TCP, a network process

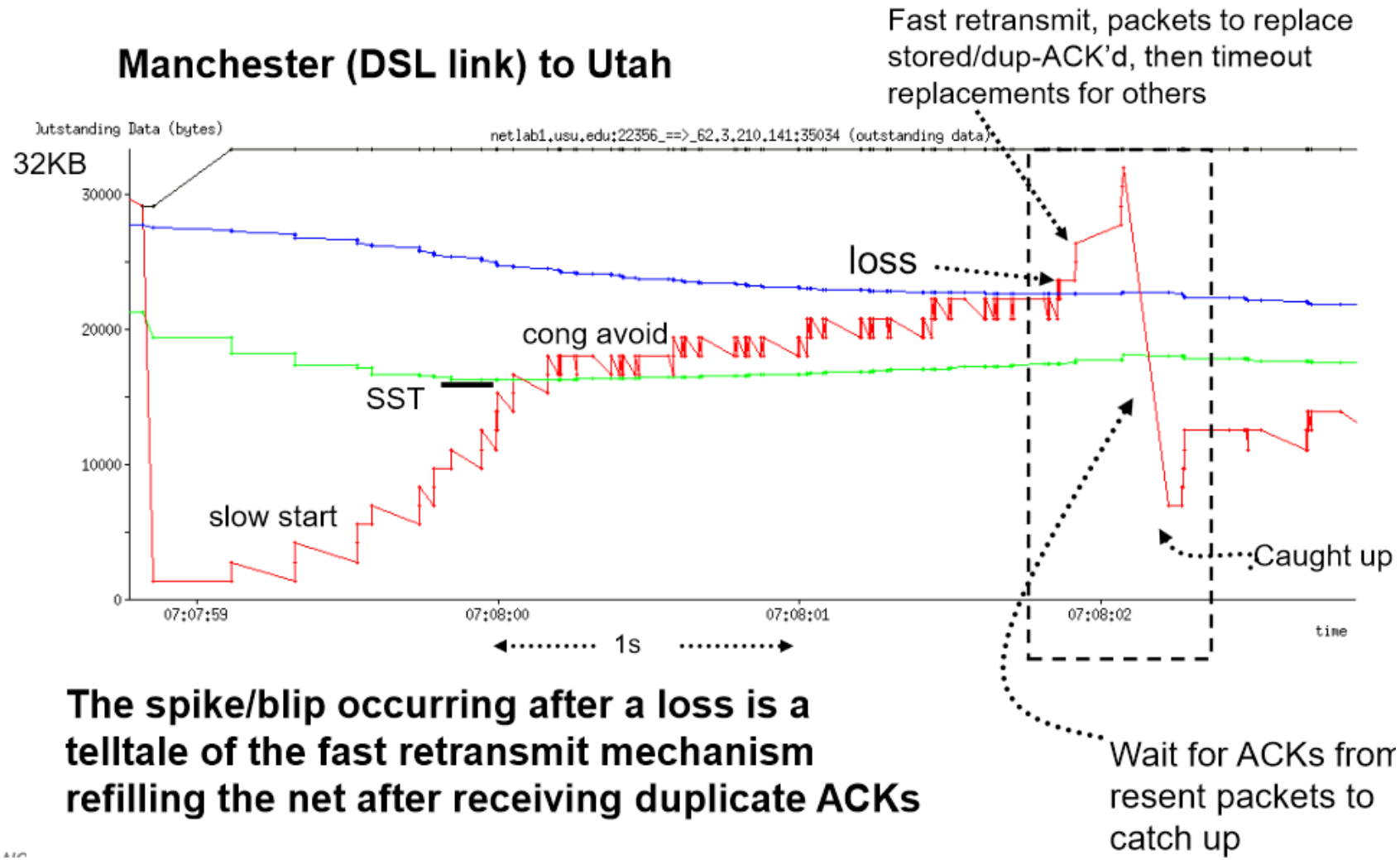
Clever work with TCP ACK data

ACK packets carry the sequence number of the last data byte received in proper sequence, plus 1. All arrivals must be ACK'd, sooner or later.

- If packets were lost midway in a stream then a clever step can be taken. ACK the last good+1, plus, in the ACK Options field indicate where up to three gaps exist. Then the sender would better know what to replace quickly. Meanwhile the receiver retains post-gap packets. This method is known as Selective ACKs, SACK. For details see RFC2018.
- Another method is when the ACK sequence number is stuck by a gap then post-gap arrivals trigger ACKs also stuck at that sequence number. The sender recognizes repeats and quickly tries to fill a gap without a slow start. This is the simpler Fast Retransmit method, RFC2582.
- An ACK can wait a little and cover multiple successful arrivals in one ACK (seq number stuff). This is a Delayed ACK. It reduces network chatter.

A detail of recovery from a simple loss

Fast Retransmit observation



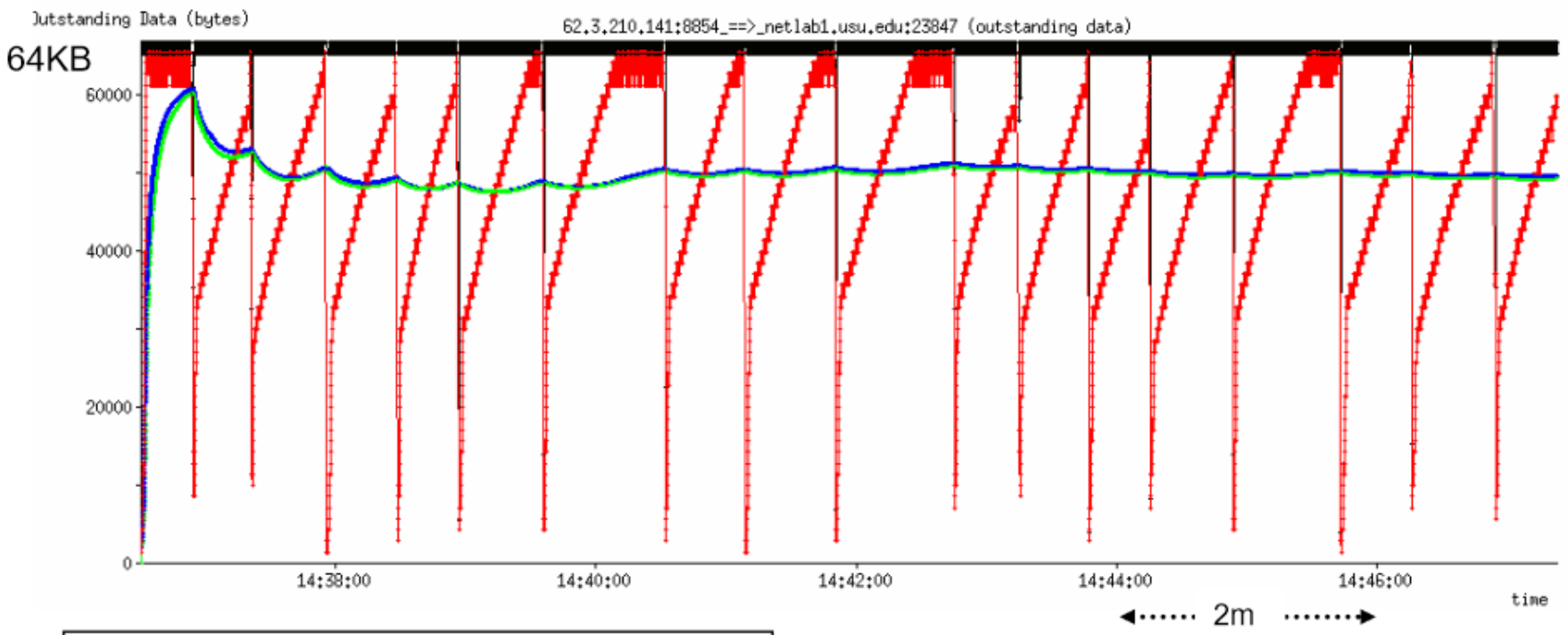
The spike/blip occurring after a loss is a telltale of the fast retransmit mechanism refilling the net after receiving duplicate ACKs

Cwin is inflated temporally to allow sending replacements.

From presentation [TCP, a network process]

Man-Utah, sending from slow side

Going outward (DSL "Upload") is much less congested



Black: remote receiver's window size

Red: bytes in flight (sent - ACK'd)

Blue: simple average bytes in flight

Green: long time weighted average bytes in flight

Ten minutes of steady transferring, 56KB/sec

Nearly all time is in congestion avoidance mode

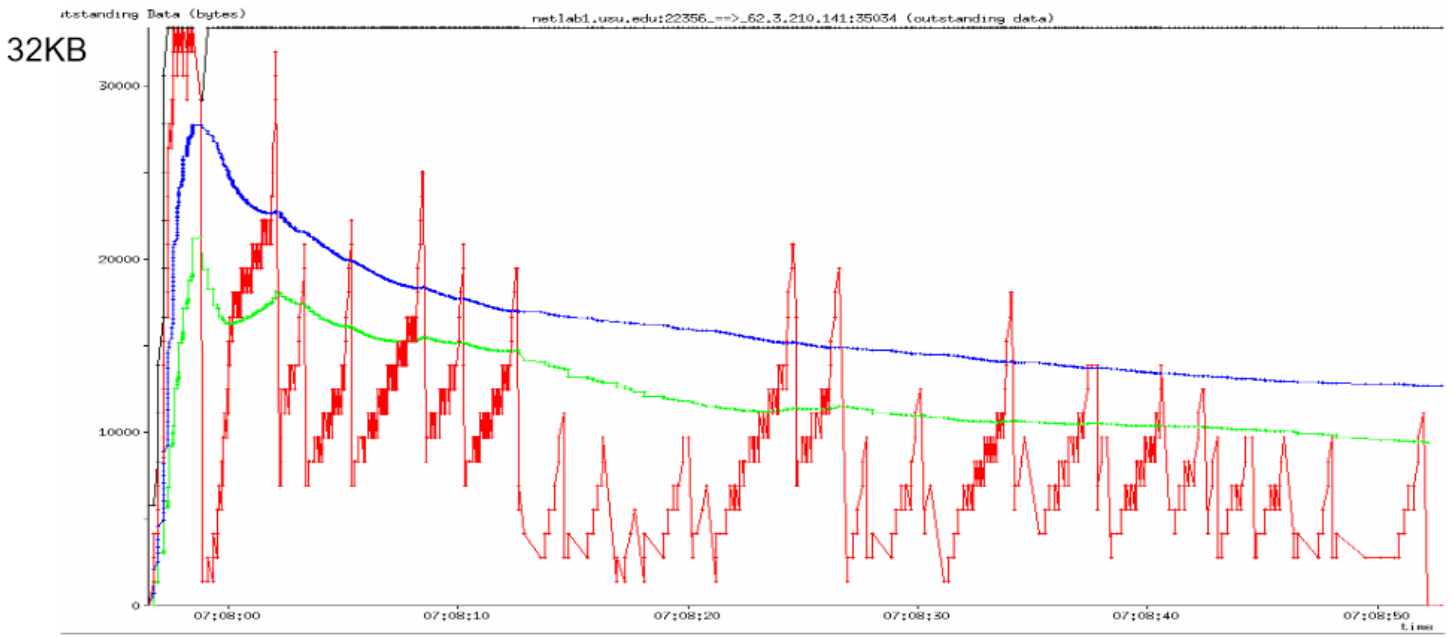
Note congestion effects even here

From presentation [TCP, a network process]

The reverse direction, heavily congested

Utah to Manchester DSL

Network congestion causes packet loss, restart & relearning



◂····· 20s ·····▸

Black: remote receiver's window size

Red: bytes in flight (sent - ACK'd)

Blue: simple average bytes in flight

Green: long time weighted average bytes in flight

About 30KBps, declining, rtt 200ms

"Slow start" really is not slow

Spikes are quickly refilling of intermediary losses via the fast retransmit algorithm.

Without TCP's feedback and adjustment rules the Internet would not survive.

IP and UDP do not have this capability.

From presentation TCP, a network process

Summary

- We understand that IP itself is simple and tries to route packets box to box over the Internet. But many problems exist, including congestion, duplication, losses and multiple apps/sessions on a given IP number. IP requires helpers.
- UDP solves the multi-apps/sessions part via using Port numbers.
- TCP also uses Ports and does reliable delivery, where both ends know what worked or not, plus cope with losses while being a good network neighbour. Knowing that a transfer worked, despite losses etc, is vital.
- TCP has many heuristics to deal with these problems, too much material for this presentation. Reading books on the subject is recommended.
- I have not discussed IPv6 which, in my opinion, has overly complicated matters. It's adoption has been very slow. HTTP/3 (HTTP over UDP) is allowing aggressiveness to compete with sound engineering (see presentation HTTP3).

Book references

"Computer Networks", by Andrew Tanenbaum
"TCP/IP Illustrated", by Richard Stevens
"Unix Network Programming", by Richard Stevens
"Routing in the Internet", by Christrian Huitema
"Gigabit Ethernet", by Rich Seifert
"The Switch Book", by Rich Seifert
"Interconnections, Bridges and Routers", by Radia Perlman
"The Linux Networking Architecture", by K. Wehrle and others

MindWorks Inc. Ltd
210 Burnley Road
Weir
Bacup
OL13 8QE UK

Telephone: +44 (0) 170 687 1900
Fax: +44 (0) 170 687 8203
Web: www.mindworksuk.com
Email: training@mindworksuk.com