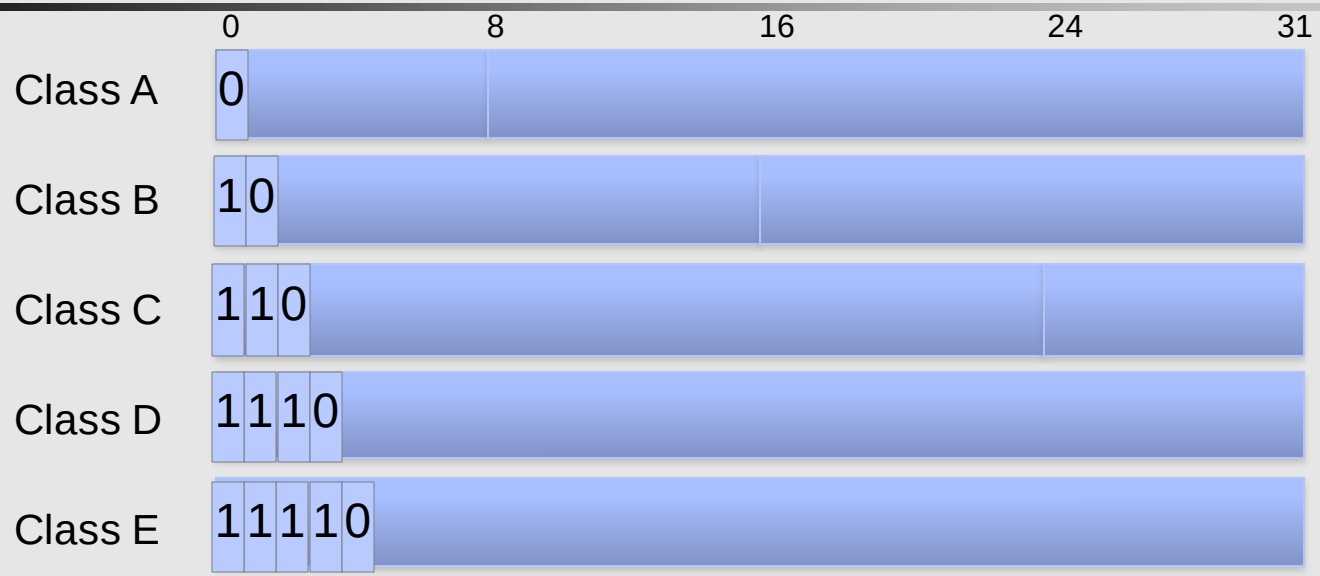
The Network Layer

Forwarding, Routing, Addressing

IP Address Classes



Class	First Byte	# Networks	Hosts per Network	Comments
A	< 128	2^7 (128)	2^{24} –2 (16M)	Mostly used
B	128…191	2^{14} (16384)	2^{16} – 2 (65534)	Mostly used
C	192…223	2^{21} (2M)	2^8 – 2 (254)	
D	224…239	2^{28} (268M)	-	Dynamic, multicast
E	240…255	2^{27} (134M)	-	reserved

Network Size Problems

- (net_id, host_id) pair, three classes, but
  - Class A (16M hosts) too large for most sites
  - Class C (254 hosts) too small
  - Class B (65,534 hosts) is OK
  - the efficiency problem
- Limited address space
  - Class thresholds, e.g., what if a class C net grows beyond 254 hosts?
- Organizations requested Class B addresses to avoid outgrowing the 8-bit host field of Class C:
  - only 65,534 Class B addresses → not enough
  - more than half of all Class B networks have fewer than 50 hosts
    - inefficient use

Towards Solutions

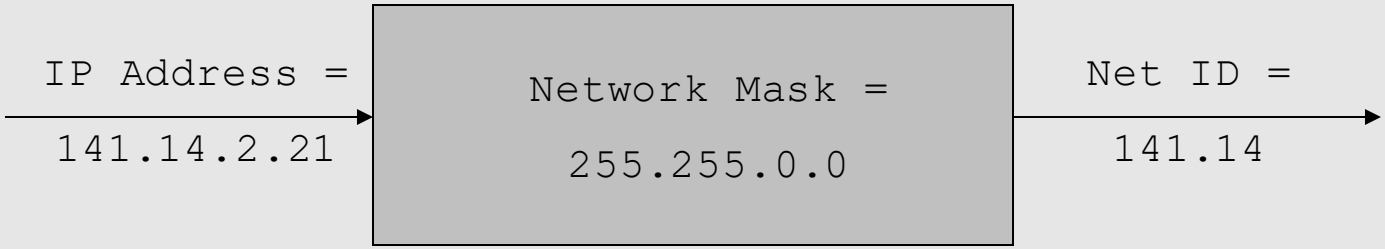
- Solutions must be backward compatible
- Problem 1: Large number of networks → routing table size
  - IP aggregation: same IP prefix must be shared by multiple physical networks
  - Sharing a large IP network to serve several physical networks
  - "***Subnetting***"
- Problem 2: Exhaustion of Class B addresses
  - temporary fix: use multiple class C addresses instead
  - Using several IP networks to serve one large physical networks
  - "***Supernetting***"
- Generalization
  - De-couple network size from IP itself
  - "CIDR"

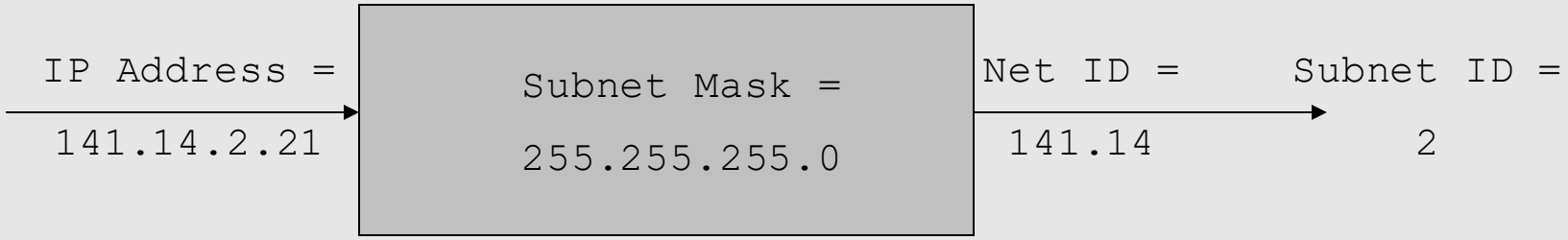
Subnet Masks

- *Masking*: the process of extracting the address of the physical network from an IP address
- A *mask* is a 32-bit number. Bits are set to…
  - 1, indicating the corresponding bit of the IP address is part of the network address (net ID or subnet ID)
  - 0, indicating the corresponding bit of the IP address is part of the host ID
- Based on first bits of the IP address, you can always figure out how long (how many bits) the network ID is
- With a subnet mask, you can *also* figure out what the subnet ID is

Subnet Masks

- Apply bitwise-AND operation on IP address and mask to find the net ID and subnet ID
- How can you tell if a destination IP address is…
  - On the same subnet as you?
  - On the same network as you?





Forwarding Table Selection Rules

- Compare destination IP address of an arriving packet against **ALL** rows within the router forwarding table because there may be multiple matches
- Select a single row that matches
- If multiple rows match, select **the longest match**
- If multiple rows tie on the longest match, select the row with the largest or smallest metric, depending on the specific metric (e.g., lowest delay)
- (If there is no match, select the default row)
  - Syntactic solution - always include 0.0.0.0 as a net / mask

Example

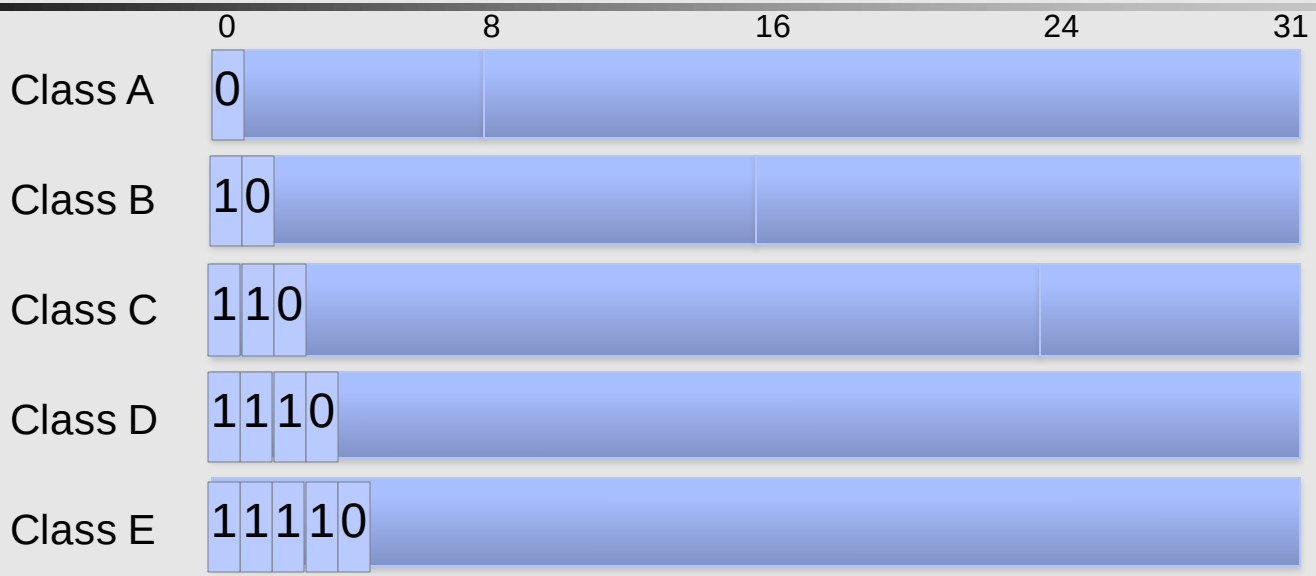
Line	Destination Address	Netmask	Metric (Cost)	Interface	Next-Hop Router
1	152.19.0.0	16	47	2	B
2	152.15.11.0	20	1	1	Local
3	152.1.0.0	16	12	2	B
4	152.40.0.0	16	33	2	B
5	152.229.0.0	16	34	1	D
6	152.40.6.0	24	47	3	E
7	152.19.17.0	24	55	4	H
8	152.229.0.0	16	20	3	E
9	152.40.8.0	24	23	1	D
10	152.15.12.0	20	9	2	Local
11	152.15.122.0	20	3	3	Local
12	0.0.0.0	0	5	3	H

- Default router?
- 152.1.1.211 – which router/rule?
- 152.40.8.44 – which router/rule?
- 152.15.13.99 – which router/rule?
- 125.1.2.3 – which router/rule?

Supernet Addressing

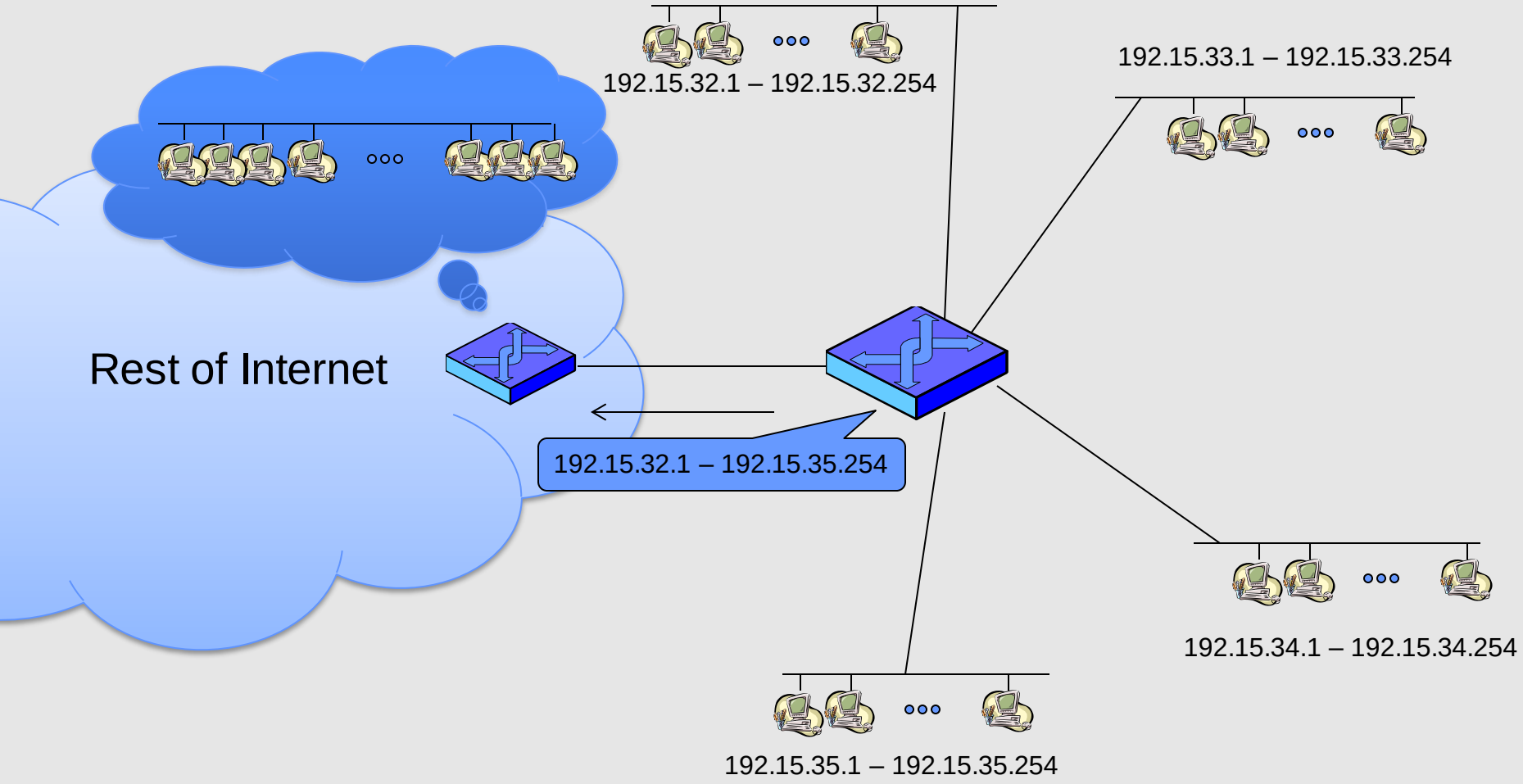
- Problem: Exhaustion of Class B addresses → use Class C addresses instead
- Organizations assigned blocks of 2^n contiguous Class C addresses:
  - if 2000 addresses (hosts) are needed, allocate a block of 2048 addresses (8 Class C networks)
- ISPs can use this scheme effectively
  - "supernetting"
  - If not: routing table of external routers may explode, since they need to create a routing entry for each Class C network.

IP Address Classes



Class	First Byte	# Networks	Hosts per Network	Comments
A	< 128	2^7 (128)	2^{24} –2 (16M)	Mostly used
B	128…191	2^{14} (16384)	2^{16} – 2 (65534)	Mostly used
C	192…223	2^{21} (2M)	2^8 – 2 (254)	
D	224…239	2^{28} (268M)	-	Dynamic, multicast
E	240…255	2^{27} (134M)	-	reserved

Supernetting Example

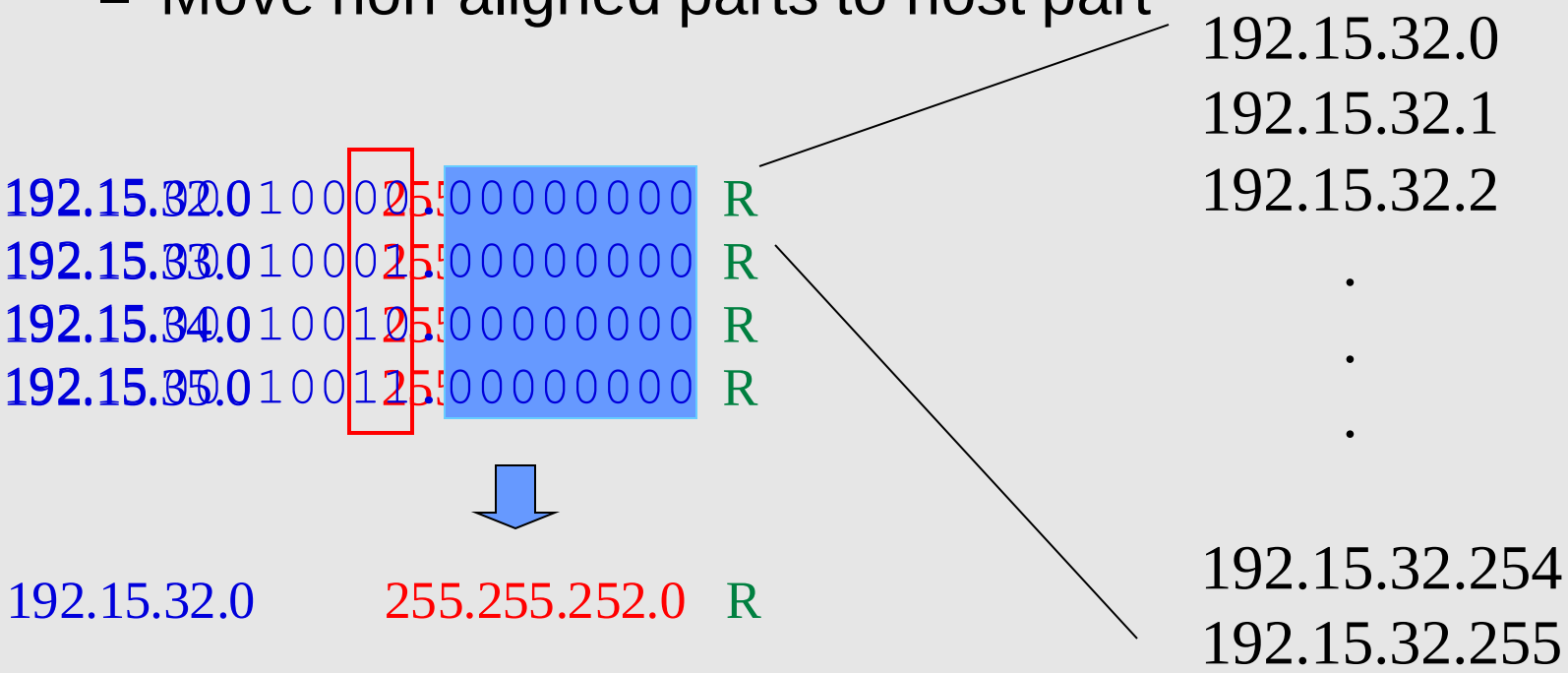


Supernetting Problem

- Avoid routing table size explosion
  - External routers must represent each IP network making up a supernet individually
- But - contiguous nets comes to the rescue
  - Allows route aggregation
- Aggregation: use the flexibility of masking to include rules for non-standard network sizes (not Class A, B or C) in the forwarding table
  - Another step in generalizing

Route Aggregation Example

- If networks are contiguous, one starts where the last one left off
  - Must be “aligned”
  - Move non-aligned parts to host part



IP Forwarding

IP Forwarding

When forwarding a packet based on its destination address, which of the following matching strategy is used to find the next-hop of the packet?

A	Shortest Match
B	Longest Match
C	First Match
D	Random Match

Route Aggregation

Given the following network addresses:
155.28.00101100.00000000/24
155.28.00101101.00000000/24
155.28.00101110.00000000/24
155.28.00101111.00000000/24
What is the aggregated routing table entry?

A	155.28.00101100.00000000/22
B	155.28.00101101.00000000/22
C	155.28.00101110.00000000/22
D	155.28.00101111.00000000/22

Classless InterDomain Routing (CIDR)

- Basic idea: allocate multiple IP addresses in a way that results in a smaller number of routing table entries
  - IP addresses must share the same high-order bits
- A block of contiguous addresses is collapsed, or *summarized*, into a single logical network
  - Thus facilitates route aggregation
- Such networks are also units of routing

Classless InterDomain Routing (cont'd)

- Not restricted to Class C addresses
  - old Class A, B, C networks no longer used with CIDR
  - "classless" - different individual networks have different sizes
- Routing table entry of the form (network_address, count), such that
  - network_address: the smallest address in the block
  - count: total number of addresses in the block
- In reality, "count" not used
  - size of each block is a power of two
  - 32-bit mask indirectly specifies the size of the block
- Really, combination of
  - Subnetting
  - Supernetting with arbitrary total size

Allocation of Class C Address Space

Lowest	Highest	Allocation
194.0.0.0	195.255.255.255	Europe
198.0.0.0	199.255.255.255	North America
200.0.0.0	201.255.255.255	Central and South America
202.0.0.0	203.255.255.255	Asia and the Pacific
204.0.0.0	223.255.255.255	Reserved

- All sites in Europe have a common prefix
- CIDR summarizes 32M addresses into one entry
  - only a single entry needed in most U.S. routers
- Once the packet gets to Europe, more detailed routing tables are needed

CIDR Example

- Three sites in Europe ask for 2048, 1024, and 4096 addresses, respectively

Site	Lowest	Highest	Mask	"Count"
1	194.24.0.0	194.24.7.255	255.255.248.0	2048 = 2^{11}
2	194.24.8.0	194.24.11.255	255.255.252.0	1024 = 2^{10}
3	194.24.16.0	194.24.31.255	255.255.240.0	4096 = 2^{12}

CIDR Example (cont'd)

- Routers all over Europe are now updated with three entries:

Network Address	Mask	Next Hop
11000010 00011000 00000000 00000000	11111111 11111111 11111000 00000000	R1
11000010 00011000 00001000 00000000	11111111 11111111 11111100 00000000	R2
11000010 00011000 00010000 00000000	11111111 11111111 11110000 00000000	R3

CIDR Example (cont'd)

Network Address	Mask	Next Hop
11000010 00011000 00000000 00000000	11111111 11111111 11111000 00000000	R1
11000010 00011000 00001000 00000000	11111111 11111111 11111100 00000000	R2
11000010 00011000 00010000 00000000	11111111 11111111 11110000 00000000	R3

- Packet comes in addressed to 194.24.17.4:
  11000010 00011000 00010001 00000100
- Boolean ANDed with Site 1 mask =
  11000010 00011000 00010000 00000000
  - does not match Site 1 base address
- Boolean ANDed with Site 2 mask =
  11000010 00011000 00010000 00000000
  - does not match Site 2 base address
- Boolean ANDed with Site 3 mask =
  11000010 00011000 00010000 00000000
  - matches Site 3 base address → sent to R3

CIDR Example – How to Construct

- Three sites in Europe ask for 2048, 1024, and 4096 addresses, respectively

Site	Lowest	Highest	Mask	"Count"
1	194.24.0.0	194.24.7.255	255.255.248.0	$2048 = 2^{11}$
2	194.24.8.0	194.24.11.255	255.255.252.0	$1024 = 2^{10}$
3	194.24.16.0	194.24.31.255	255.255.240.0	$4096 = 2^{12}$

How 194.24.8.0 in Site 2comes from?

CIDR Example

How 194.24.8.0 in Site 2 comes from?

- 1) Start from the “count”

  you need 2^10 host IDs, so it requires 10 zeros in host ID area,
  i.e., 194.24.00000100.00000000 (194.24.4.0)

2) check the overlaps

194.24.4.0 is overlapped with the IP range of site 1.

So use 194.24.8.0 instead (as highest IP of site 1 is 192.24.7.255)

CIDR Example – How to Construct

- Three sites in Europe ask for 2048, 1024, and 4096 addresses, respectively

Site	Lowest	Highest	Mask	"Count"
1	194.24.0.0	194.24.7.255	255.255.248.0	$2048 = 2^{11}$
2	194.24.8.0	194.24.11.255	255.255.252.0	$1024 = 2^{10}$
3	194.24.16.0	194.24.31.255	255.255.240.0	$4096 = 2^{12}$

How 194.24.16.0 in Site 2comes from?

CIDR Example

How 194.24.16.0 in Site 3 comes from?

- 1) Start from the “count”

  you need 2^12 host IDs, so it requires 12 zeros in host ID area,
  i.e., 194.24.00010000.00000000 (194.24.16.0)

2) check the overlaps

194.24.16.0 is not overlapped with the IP range of site 2.
So 194.24.16.0 works.

CIDR Example

- Three sites in Europe ask for 2048, 1024, and 4096 addresses, respectively

Site	Lowest	Highest	Mask	"Count"
1	194.24.0.0	194.24.7.255	255.255.248.0	$2048 = 2^{11}$
2	194.24.8.0	194.24.11.255	255.255.252.0	$1024 = 2^{10}$
3	194.24.16.0	194.24.31.255	255.255.240.0	$4096 = 2^{12}$

Can these address blocks be aggregated?

CIDR Example

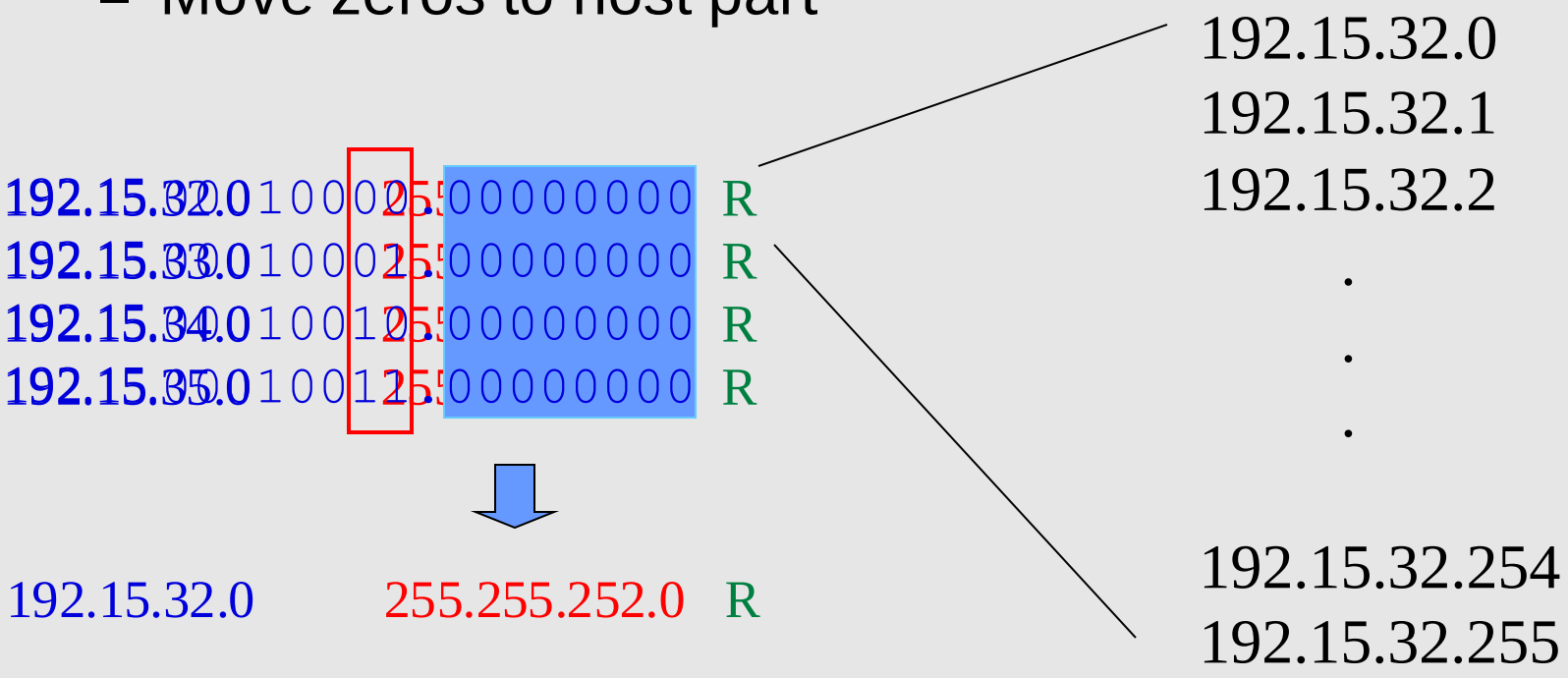
- Three sites in Europe ask for 2048, 1024, and 4096 addresses, respectively

Site	Lowest	Highest	Mask	"Count"
1	194.24.0.0	194.24.7.255	255.255.248.0	$2048 = 2^{11}$
2	194.24.8.0	194.24.11.255	255.255.252.0	$1024 = 2^{10}$
3	194.24.16.0	194.24.31.255	255.255.240.0	$4096 = 2^{12}$

Can be aggregated only if 1) all IP addresses from different sites are continuous, and 2) aligned (occupying an entire power of 2 block of addresses). If yes, all blocks can be aggregated as one entry: e.g., **199.24.0.0/K**. (2^{32} = (all counts)*2^{K})

Route Aggregation Example

- If networks are contiguous, one starts where the last one left off
  - Must be “aligned”
  - Move zeros to host part

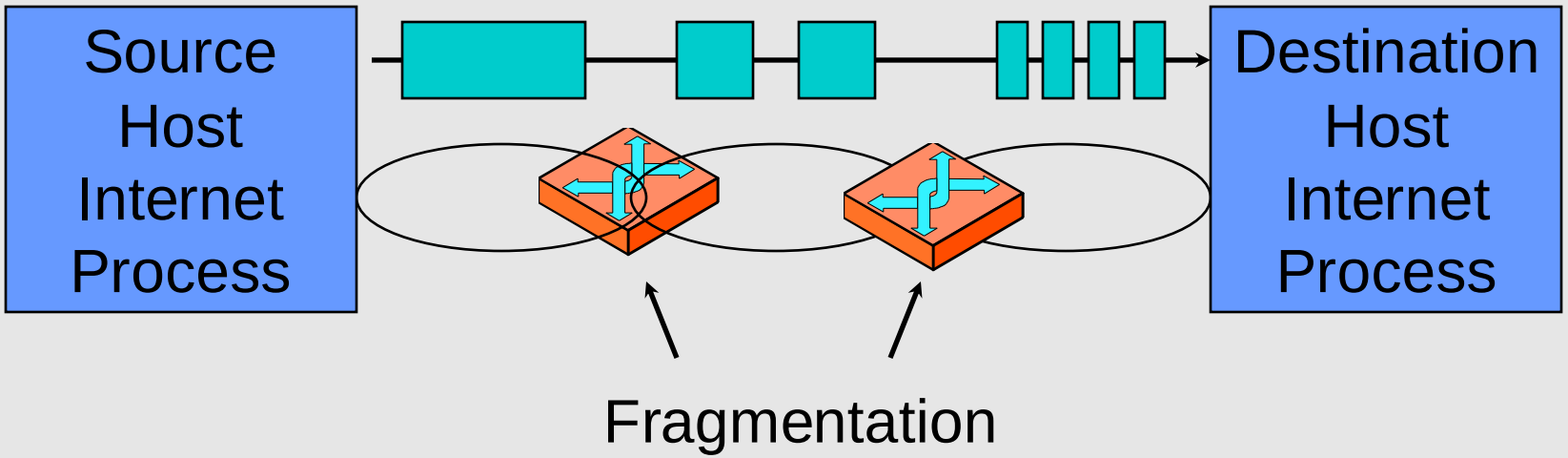


Summary: CIDR Routing

- Router entries not tried sequentially; accelerating data structures used
- Longest prefix match
  - if two entries match, the one whose mask has the most 1 bits wins
- Studies claim that if:
  - CIDR were applied to all IP addresses, and
  - existing IP addresses were reallocated according to continental boundaries/ISPs
  - Then routing table entries could be reduced significantly
  - http://www.cidr-report.org/
- "Short-term" solution
  - long-term solution: IPv6 (challenges - IoT)

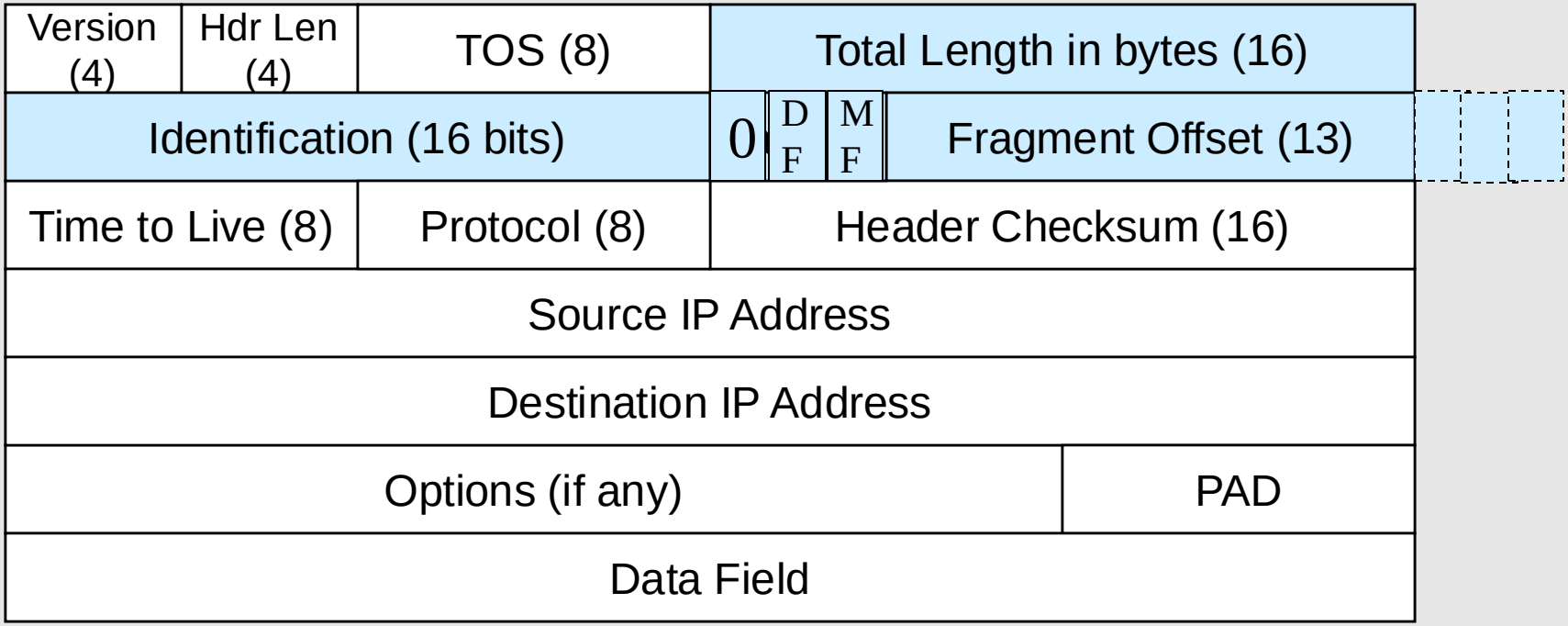
IP Fragmentation

- Various physical networks may have different MTU
  - If incoming IP packet is longer, fragment
- Fragments are still IP packets
  - Only slicing data (header is replicated) (assign same IP packets)
- Original packet may be fragmented multiple times along its route – defragmentation only by destination IP



Defragmentation

- Each fragment has same datagram ID as original
- Individual datagram lengths
- Offset of this fragment's payload in original
  - Use start location of the first byte (e.g., 0, 2, 4, 16, …)



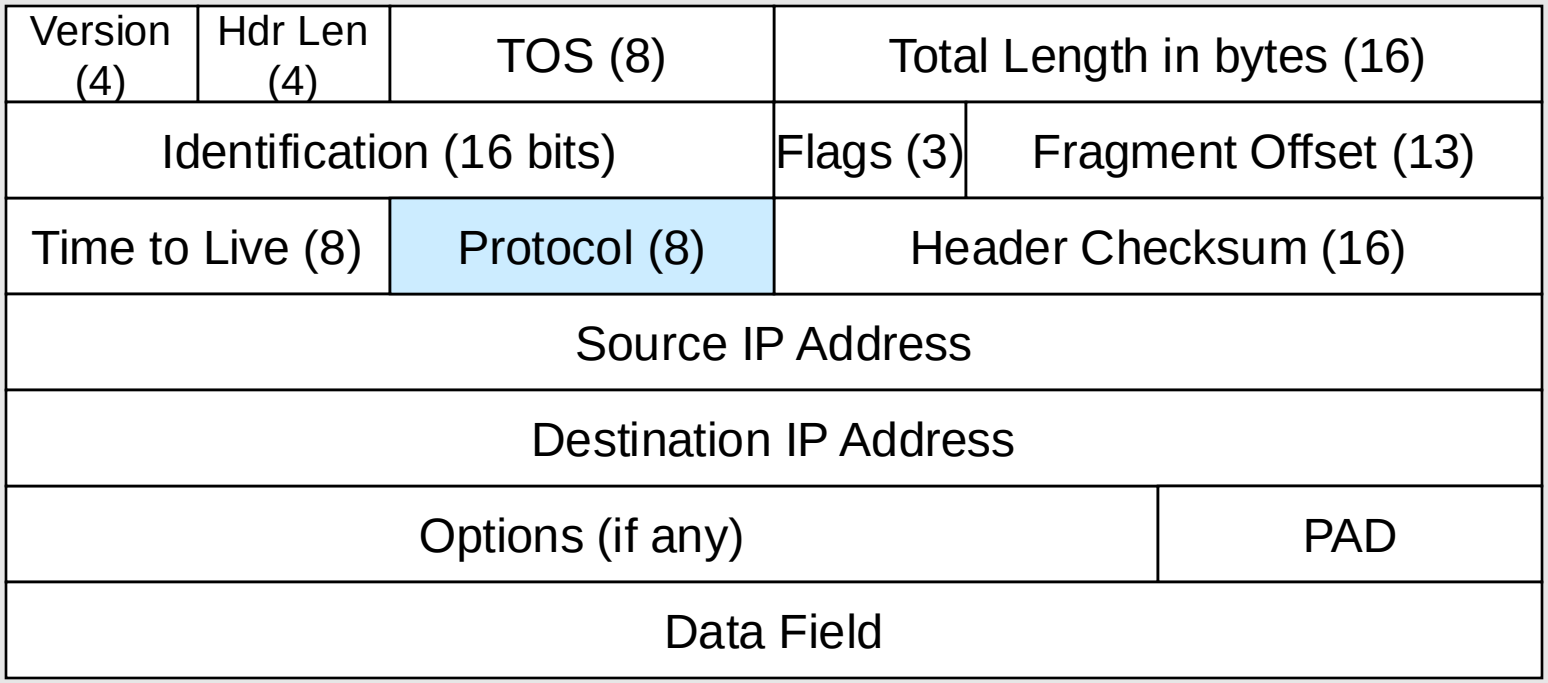
Time To Live

- Intended to guard against datagrams surviving indefinitely even under routing inconsistencies
- Each router expected to hold for less than a second, decrement by 1, discard if TTL=0

Version (4)	Hdr Len (4)	TOS (8)	Total Length in bytes (16)	
Identification (16 bits)			Flags (3)	Fragment Offset (13)
Time to Live (8)	Protocol (8)		Header Checksum (16)	
Source IP Address				
Destination IP Address				
Options (if any)				PAD
Data Field				

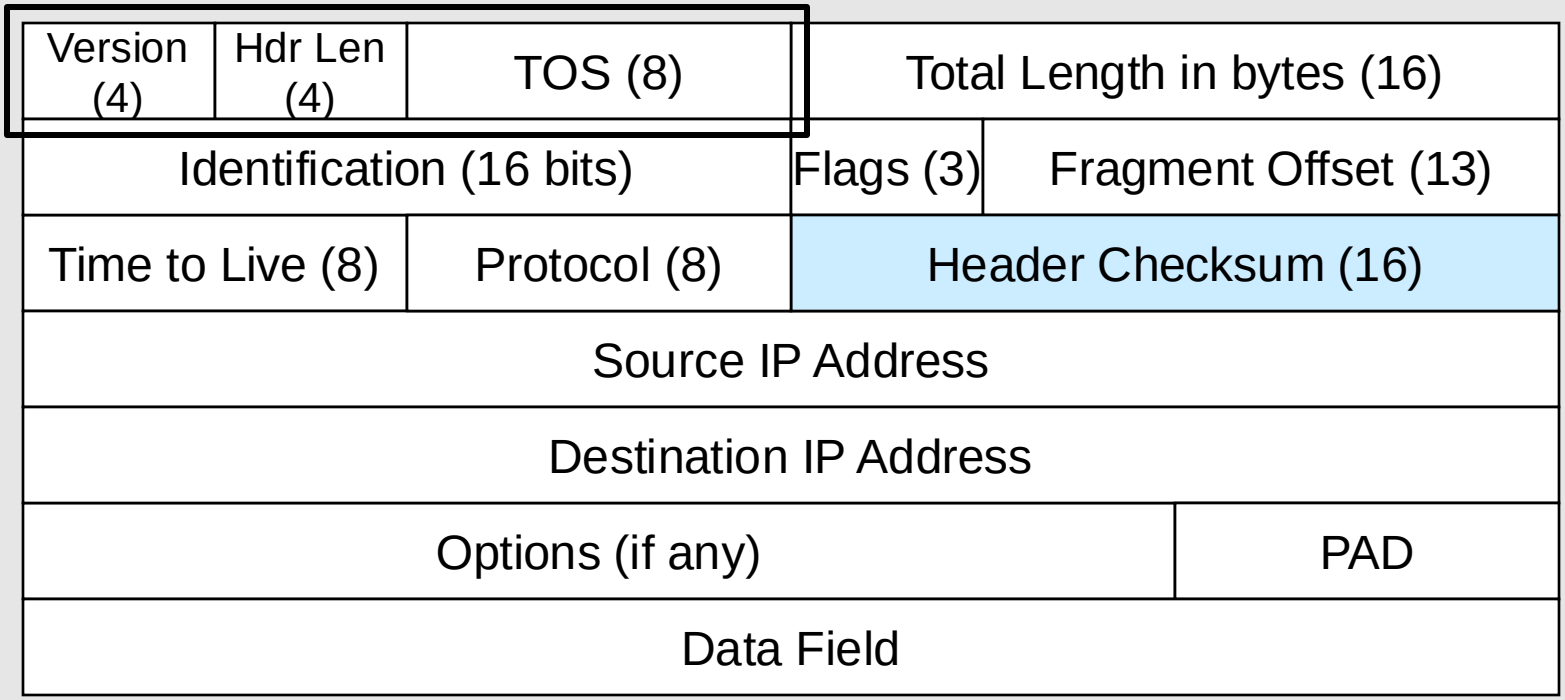
Protocol

- Indicates protocol used by the encapsulated payload
- ICMP = 1, TCP = 6, UDP = 17, etc.



Header Checksum

- Covers header only (not payload)
  - One's complement of :
    (one's complement sum of (all 16-bit words in header))
  - Checksum itself left out (equivalently, read as 0^{16})



ns-3

- IP addressing: Ipv4AddressHelper and Ipv6AddressHelper
  - provide methods for creating IP address objects and assigning them to network interfaces.

```
#include "ns3/ipv4-address.h"
#include "ns3/ipv4-interface.h"

// create a new network interface
Ptr<NetDevice> device1 = CreateObject<NetDevice> ();

// create an IPv4 address object
Ipv4Address ipAddress ("192.0.2.1");
Ipv4Mask subnetMask ("255.255.255.0");
Ipv4InterfaceAddress interfaceAddress (ipAddress, subnetMask);

// assign the IP address to the device
interfaceAddress = address.assign (device1);

// create an IPv4 interface for the device
Ptr<Ipv4Interface> ipv4Interface = CreateObject<Ipv4Interface> ();
ipv4Interface->AddAddress (interfaceAddress);

// configure routing for the interface
ipv4Interface->SetMetric (1);
ipv4Interface->SetMtu (1500);
```

Example

Line	Destination Address	Netmask	Metric (Cost)	Interface	Next-Hop Router
1	152.19.0.0	16	47	2	B
2	152.15.33.0	24	0	1	Local
3	152.1.0.0	16	12	2	B
4	152.40.0.0	16	33	2	B
5	152.229.0.0	16	34	1	D
6	152.40.6.0	24	47	3	E
7	152.19.17.0	24	55	4	H
8	152.229.0.0	16	20	3	E
9	152.40.8.0	24	23	1	D
10	152.15.12.0	24	9	2	Local
11	152.15.122.0	24	3	3	Local
12	0.0.0.0	0	5	3	H

- Default router?
- 152.1.1.211 – which router/rule?
- 152.40.8.44 – which router/rule?
- 152.15.13.99 – which router/rule?
- 125.1.2.3 – which router/rule?

ns-3

- IP addressing: Ipv4AddressHelper and Ipv6AddressHelper
  - provide methods for creating IP address objects and assigning them to network interfaces.

```
#include "ns3/ipv6-address-helper.h"

// create a helper object
Ipv6AddressHelper addressHelper;

// create an IPv6 address object
Ipv6Address address = addressHelper.GetGlobal ();

// assign the address to a network interface
interface->SetAddress (Ipv6InterfaceAddress (address, prefix));
```

ns-3

- New address base:

```
Ipv4StackHelper stack;
stack.install (P2PNodes);

Ipv4AddressHelper address;
address.Setbase (“10.0.0.1”, “255.255.255.0”);
Ipv4Interface interface1; interface2, interface3;
interface 1 = address.assign (device1);
address.newNetwork ();
interface 2 = address.assign (device2);
address.newNetwork ();
interface 3 = address.assign (device3);
```