

Serialization and Bit Operations

CS 113: Introduction to C

Instructor: Saikat Guha

Cornell University

Spring 2007, Lecture 9

Serialization

- ▶ Sending data between programs
 - ▶ Disk
 - ▶ Network
 - ▶ Pipes
- ▶ Between programs on multiple hosts
 - ▶ Different endianness
 - ▶ Different architectures

Binary vs. Text

Binary...

- ▶ Compact
- ▶ Easy to encode/decode
- ▶ Faster

e.g. IP, TCP, AIM, ...

Text...

- ▶ Easily debugged
- ▶ (Can be) self-documenting
- ▶ Arch/Endian independent

e.g. HTTP, SMTP, MSN

Handling Endianness

Decimal: 3735928559

Binary: 11011110101011011011111011101111

Hex: 0xdeadbeef

Big Endian: 0xde 0xad 0xbe 0xef

Little Endian: 0xef 0xbe 0xad 0xde

Always in big-endian form when loaded into the CPU

Bit-Operations

AND-Mask (clear bits)

a & b

1101111010101101**10111110**11101111

&

0000000000000000**11111111**00000000

=

0000000000000000**10111110**00000000

0xdead**beef**

&

0x0000**FF00**

=

0x0000**be00**

Bit-Operations

OR-Mask (sets bits)

a		b	
1101111010101101		10111110	11101111
0000000000000000		01010101	00000000
=			
1101111010101101		11111111	11101111

0xdead	beef
0x0000	5500
=	
0xdead	FFef

Bit-Operations

Left-Shift

$a \ll b$

11011110101011011011111011101111

\ll

8

=

10101101101111101110111100000000

0xdeadbeef

\ll

8

=

0xadbeef00

Bit-Operations

Right-Shift

a >> b

11011110101011011011111011101111

>>

8

=

00000000110111101010110110111110

0xdeadbeef

>>

8

=

0x00deadbe¹

¹for unsigned ints only. For signed ints, the instead of zero-padding, the top-most bit is repeated

Exercise 1

```
int htonl(int x) {
    int b1, b2, b3, b4, y;

    b1 = (x _____) ____;
    b2 = (x _____) ____;
    b3 = (x _____) ____;
    b4 = (x _____) ____;

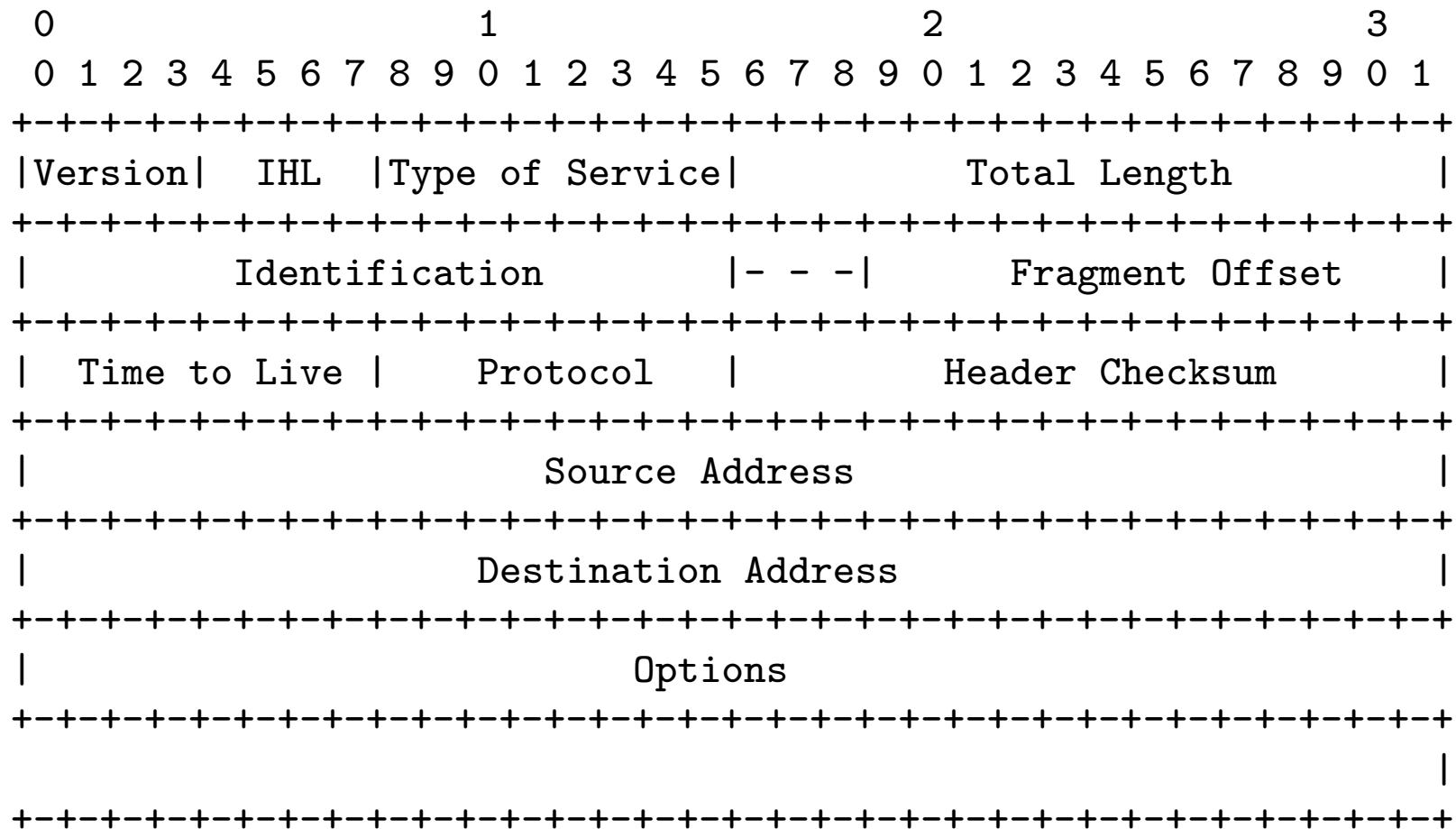
    y = (b1 _____) __ (b2 _____)
        __ (b3 _____) __ (b4 _____);

    return y;
}
```

Exercise 2

```
int htonl(int x) {  
    int y;  
    char *xs = &x, *ys = &y;  
  
    __ = __;  
    __ = __;  
    __ = __;  
    __ = __;  
  
    return y;  
}
```

Exercise 3



Use `uint8_t`, `uint16_t`, `uint32_t`

Exercise 3

```
#pragma pack(push)
struct ip {
#ifdef LITTLE_ENDIAN
    uint8_t  ihl:4, ver:4;
#else
    uint8_t  ver:4, ihl:4;
#endif
    uint8_t  tos;
    uint16_t len;
    uint16_t iid;
    uint16_t off;
    uint8_t  ttl;
    uint8_t  prt;
    uint16_t csm;
    uint32_t src;
    uint32_t dst;
    char opt[40];
};
#pragma pack(pop)
```

Serialization

```
void foo(void) {
    struct ip *ip1, *ip2;

    ip1 = (struct ip *)malloc(sizeof(struct ip));
    ip2 = (struct ip *)malloc(sizeof(struct ip));

    ip1->ver = 4;
    ip1->protocol = 6;
    ip1->len = htonl(40);

    memcpy(ip2, ip1, sizeof(struct ip));

    printf("%d %d", ip2->ver, ntohl(ip2->len));

    ...
}
```

Serialization

- ▶ Use structures for data-types
- ▶ Copy data in one-go
`memcpy(dst, src, numbytes)`
- ▶ Use standard (big) endianness for multi-byte variables
- ▶ **NEVER** serialize pointer values. Why?

Bit-Operations

Compliment (flips bits)

$\sim a$

$\sim 11011110101011011011111011101111$

=

00100001010100100100000100010000

$\sim 0xdeadbeef$

=

0x21524110

2's compliment representation for negative numbers:

$$-x = \sim x + 1$$