과제 3 추가 설명

16bit ones’ complement checksum

Checksum algorithm

Checksum algorithm (송신측)
{
패킷 안의 모든 요소(data, acknum, seqnum, checksum) 16bit 으로 맞춤
합 = 16bit 이 된 모든 요소를 bit-wise 로 합 계산
계산 결과 = 합에 wrap around carry 계산
checksum = 계산 결과의 1's complement
}

Checksum algorithm (수신측)
{
수신한 패킷 안의 모든 요소(data, acknum, seqnum, checksum) 16bit 으로 맞춤
합 = 16bit 이 된 모든 요소를 bit-wise 로 합 계산
계산 결과 = 합에 wrap around carry 계산
checksum = 계산 결과의 1's complement
checksum이 0 일 경우 손상되지 않은 패킷
}

16bit Bit wise operation

Bit wise 연산자를 사용하여 연산

integer type , char type 모두 16bit 으로 맞춘 후 bit-wise operation 진행

ex) bit-wise operation

UINT16 Sum = A ^ B

UINT16 Carry = A & B

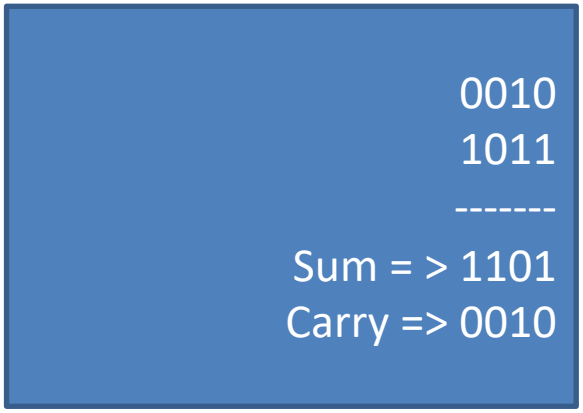
Char 의 경우 아래의 예시처럼 16bit 으로 맞춘 후 진행 할 수 있음

ex) char data[4] = AAAA

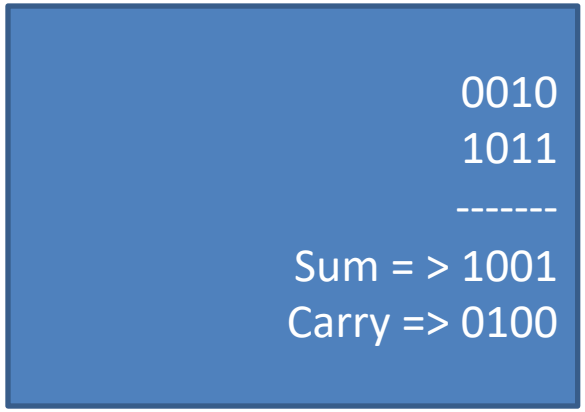
UINT16 sum = (left shift data[0] by 8 bit ^ data[1]) ^ (left shift data[2] by 8 bit^ data[3])

UINT16 carry = (left shift data[0] by 8 bit ^ data[1]) & (left shift data[2] by 8 bit^ data[3])

예시 결과



원래 결과



Bit-wise operation 을 사용하는 one's complement addition

```
one's complement addition:
{
        UINT16 Sum = 0
        Loop(until  패킷의 모든 요소에 대한 addition 완료 할 때까지)
        {
                x = Sum
                y= 16bit 으로 변환된 패킷의 요소
                UINT16 Sum = x bit-wise XOR y
                UINT16 carry = x bit-wise AND y
                loop(carry != 0)
                {
                        condition(carry bit-wise AND left-most-bit)
                        {
                                wrap around carry state
                        }
                        left shift carry by 1 bit
                        x = Sum
                        y = carry
                        Sum = x bit-wise XOR y
                        carry = x bit-wise AND y
                }
                condition (wrap around carry state){
                        unsigned variable 1
                        Sum = unsigned variable bit-wise addition Sum
                }
        }
        Checksum =  bitflip sum

}
```

Logic에 따라, Carry의 맨 상위 bit 에 carry bit이 발생하면 곧 wrap around carry 다 발생한 것이므로 wrap around addition을 진행해줘야 하며, carry 안의 최상위 bit이 1인지를 검사하여 이를 알 수 있음

이 수도코드는 두 개의 요소에 대한 계산을 수행
구현 목표 :

- 패킷 요소의 모든 값의 합
- wrap around bit 처리
- 모든 요소에 대한 addition 완료 후 bitflip 을 통한 one's complement

이 수도코드는 하나의 예시일뿐 구현 방식 자유

Wrap around carry state on

수도코드 에 따른 4bit 연산 예시

```
left-most-bit = 1000
                                1011
                                0110
                Bit-wise OP -------
                        Sum = 1101
                        Carry = 0010
                left-most-bit = 1000
        Carry bit-wise AND left-most-bit -------
        Carry bit-wise AND left-most-bit = 0000
                Left Shift carry by 1bit = 0100
Sum bit-wise OP left shift carry by 1 bit -------
                        Sum = 1001
                        Carry = 0100
                left-most-bit = 1000
        Carry bit-wise AND left-most-bit -------
        Carry bit-wise AND left-most-bit = 0000
                Left Shift carry by 1bit = 1000
Sum bit-wise OP left shift carry by 1 bit -------
                        Sum = 0001
                        Carry = 1000
                left-most-bit = 1000
        Carry bit-wise AND left-most-bit -------
        Carry bit-wise AND left-most-bit = 1000
                Left Shift carry by 1 bit = 0000
Sum bit-wise OP left shift carry by 1 bit -------
                        Sum = 0001
                Wrap around bit = 0001
        Sum bit-wise OP Wrap around bit -------
                        Sum = 0000
                        Carry = 0010
                left-most-bit = 1000
        Carry bit-wise AND left-most-bit -------
        Carry bit-wise AND left-most-bit =0000
                Left Shift carry by 1 bit = 0100
Sum bit-wise OP left shift carry by 1 bit ------
                        Sum = 0010
                        Carry = 0000
```

checksum

Checksum

패킷안의 모든 요소(data, seqnum, acknum, checksum)의 합 의 one's complement

one's complement를 이용한 corruption 판별

Checksum 은 bitwise 연산을 통해 corruption 을 판별

Wrap around carry 가 발생하여도 summation 값이 0 일 경우

패킷 손상되지 않음

필수 구현 사항

구현 방식 자유

가산점을 위한 필수 구현 사항

- Bit-wise summation 을 통한 acknum, seqnum , data, checksum 의 합
- checksum = 합의 One’s complement
- 패킷 손상 여부 판별

BiGBN 출력 예시

```
A : Send_packet without ACK (seq = 1)
A : Send_packet without ACK (seq = 2)
A : Send_packet without ACK (seq = 3)
A : Send_packet without ACK (seq = 4)
A : Send_packet without ACK (seq = 5)
B : Send_packet with ACK (ACK = 1, seq = 1)
B : Send_packet with ACK (ACK = 2, seq = 2)
B : Send_packet with ACK (ACK = 3, seq = 3)
B : Send_packet with ACK (ACK = 4, seq = 4)
B : Send_packet with ACK (ACK = 5, seq = 5)
A : Send_packet with ACK (ACK = 1, seq = 6)
A : Send_packet with ACK (ACK = 2, seq = 7)
A : Send_packet with ACK (ACK = 3, seq = 8)
A : Send_packet with ACK (ACK = 4, seq = 9)
A : Send_packet with ACK (ACK = 5, seq = 10)
B : Send_packet with ACK (ACK = 6, seq = 6 )
B : Send_packet with ACK (ACK = 7, seq = 7 )
B : Send_packet with ACK (ACK = 8, seq = 8 )
B : packet corrupted (seq = 9)
B : Send_packet with ACK (ACK = 8, seq = 9)
B : not expected packet (seq = 10)
B : Send_packet with ACK (ACK = 8, seq = 10)
A : timerinterrupt : Send_packet with ACK (ACK = 10, seq = 9)
A : timerinterrupt : Send_packet with ACK (ACK = 10, seq = 10)
```

출력 양식

Function name	Event type	Output format
A_input & B_input	Packet corrupted	Function name, “ : Packet corrupted. Drop.”
	Got NAK	Function name : got NAK(ack = #). Drop.
	Not the expected sequence number	Function name : not the expected seq. send NAK (ack = seq #)
	Buffer is full	Function name : Buffer is full. Drop the message.
	Packet received without error	Function name : recv packet (seq = #) : data
	ACK received	Function name : got ACK(ack = #)
	Stop timer	Function name : stop timer.
	Start timer	Function name : start timer.
A_output & B_output	Send packet	Function name : send packet(seq = #) : data
	Send Ack	Function name : send ACK (ack = #).
A_timerinterrupt & B_timerinterrupt	Resend packet	Function name : resend packet (seq = #): data