《计算机网络》实验报告 2

可靠数据传输协议-GBN 协议的设计与实现

童超宇

院 （系）：计算机科学与技术学院　　专　　业：计算机科学与技术

学　　号 ：1152130106　　指导教师：刘亚维

2018 年 5 月

目录

1 实验目的

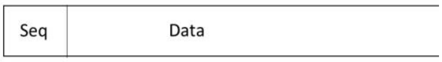
理解滑动窗口协议的基本原理;掌握 GBN 的工作原理;掌握基于 UDP 设计并实现一个 GBN 协议的过程与技术

2 实验内容

1. 基于 UDP 设计一个简单的 GBN 协议，实现单向可靠数据传输（服务器到客户的数据传输）
2. 模拟引入数据包的丢失，验证所设计协议的有效性
3. 改进所设计的 GBN 协议，支持双向数据传输（选作内容，加分项目）
4. 将所设计的 GBN 协议改进为 SR 协议。（选作内容，加分项目）

3 GBN 协议设计

3.1 数据分组格式



Seq	Data

数据分组帧第一个字节表示序列号，后面为数据，以字节为单位，字节数可变，并规定数据至少一个字节

这样数据帧至少 2 字节，通过长度可以和 ack 分组区别，以便实现双向传输

3.2 确认分组格式

ACK 分组帧为 1 字节，表示序列号从 0 到 255

3.3 起始序列号

规定 ack 为 x 表示 x 前（不包括 x）的分组已确认

发送端窗口 base 从 0 开始，接受端期望序列号从 0 开始

4 SR 协议设计

数据与确认分组与 GBN 相同

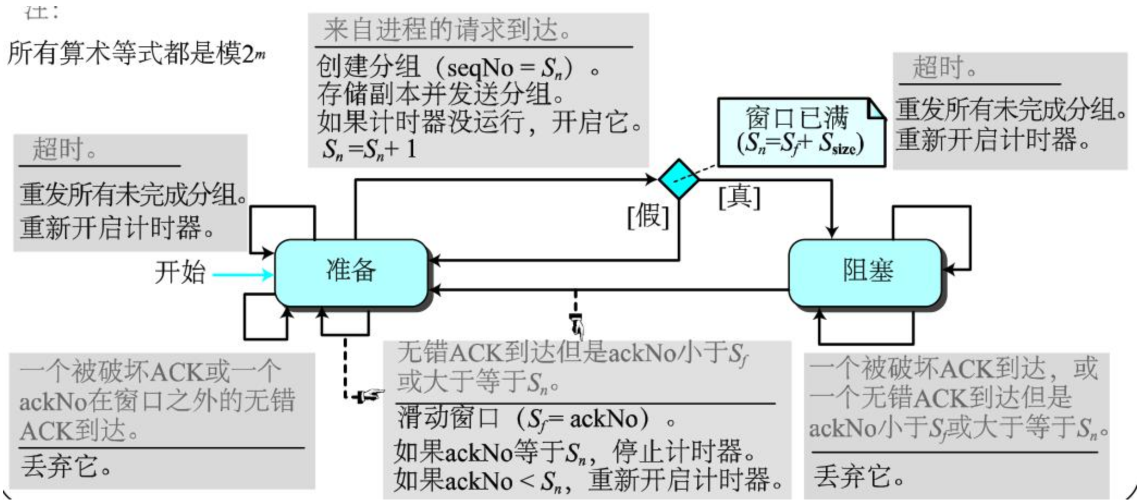
规定 ack 为 x 表示 x 分组确认

发送端窗口 base 从 1 开始，接受端期望序列号从 1 开始（初值为 0）

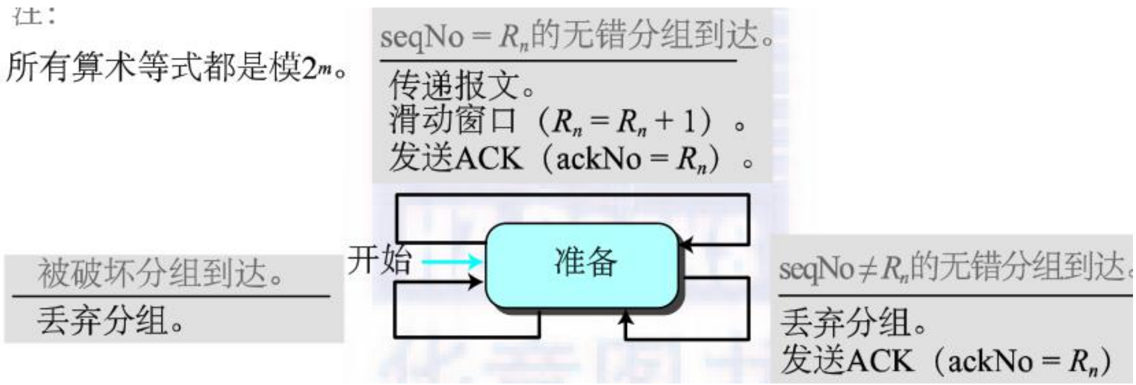
5 程序设计

5.1 协议两端程序流程图

5.1.1 发送端



5.1.2 接收端



5.2 协议典型交互过程

1. 发送方向接收方发送一段数据
2. 发送方发送数据分组，设置定时器，接收方回传 ack
3. 发送方超时重传，收到 ack 移动窗口，窗口移动放入新数据
4. 发送方窗口再次为空时过程结束

双向

1. 程序尝试绑定固定端口 p，若成功则监听该端口，并在第一次收到数据时开始向对方发一段数据
2. 若失败则随机绑一个端口，并开始向固定端口 p 发一段数据

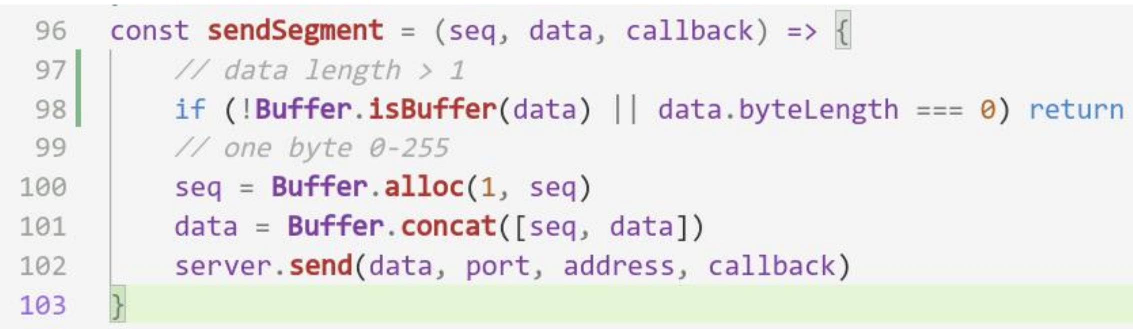
5.3 数据分组丢失验证模拟方法

预设丢包率，任一方收到数据或 ack 时生成随机数与之比较确定是否丢包

5.4 程序核心函数

5.4.1 发送分组

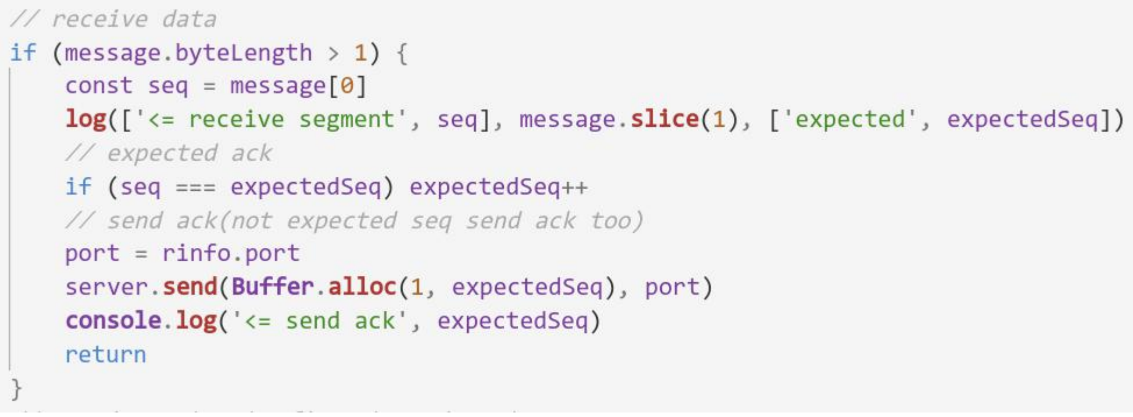
传入序列号，数据，序列号转为 1 字节存储，与数据拼成一个数据报



```
const sendSegment = (seq, data, callback) => {
    // data length > 1
    if (!Buffer.isBuffer(data) || data.byteLength === 0) return
    // one byte 0-255
    seq = Buffer.alloc(1, seq)
    data = Buffer.concat([seq, data])
    server.send(data, port, address, callback)
}
```

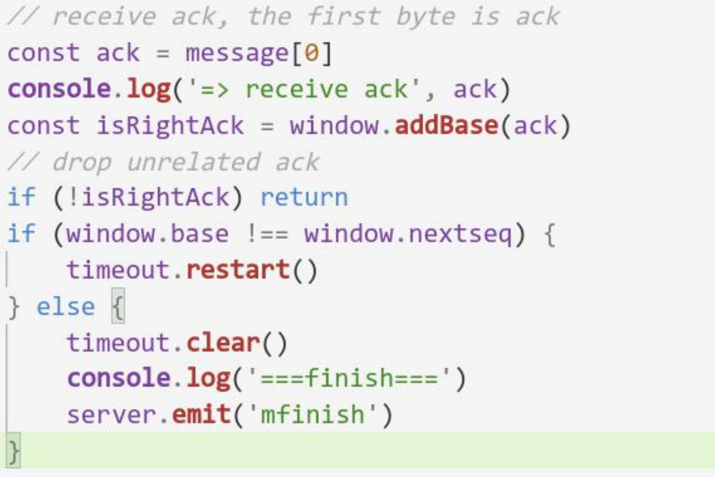
5.4.2 数据处理

收到数据若长度大于 1 字节，判断为数据分组，发送 ack



```
// receive data
if (message.byteLength > 1) {
    const seq = message[0]
    log(['<= receive segment', seq], message.slice(1), ['expected', expectedSeq])
    // expected ack
    if (seq === expectedSeq) expectedSeq++
    // send ack(not expected seq send ack too)
    port = rinfo.port
    server.send(Buffer.alloc(1, expectedSeq), port)
    console.log('<= send ack', expectedSeq)
    return
}
```

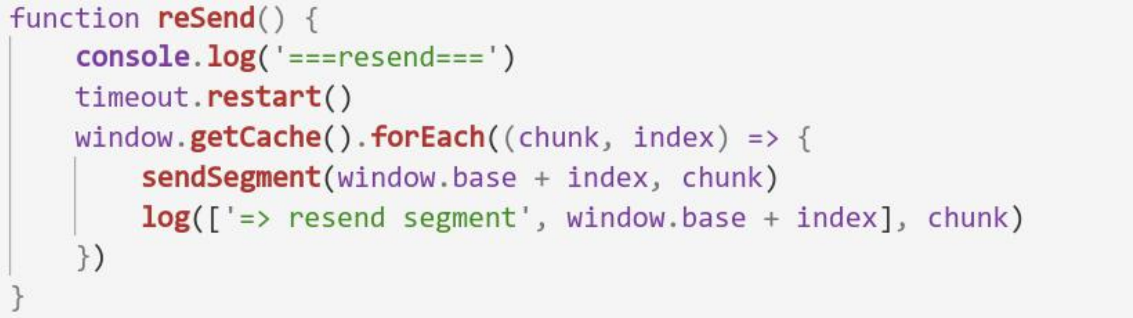
否则判断为 ack。若不是未确认的 ack 丢弃，否则重启计时器



```
// receive ack, the first byte is ack
const ack = message[0]
console.log('=> receive ack', ack)
const isRightAck = window.addBase(ack)
// drop unrelated ack
if (!isRightAck) return
if (window.base !== window.nextseq) {
    timeout.restart()
} else {
    timeout.clear()
    console.log('===finish===')
    server.emit('mfinish')
}
```

5.4.3 重传

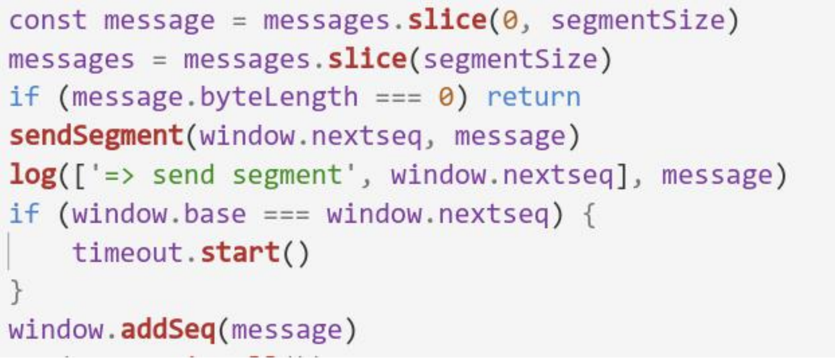
依次发送窗口中缓存的分组



```
function reSend() {
    console.log('===resend===')
    timeout.restart()
    window.getCache().forEach((chunk, index) => {
        sendSegment(window.base + index, chunk)
        log(['=> resend segment', window.base + index], chunk)
    })
}
```

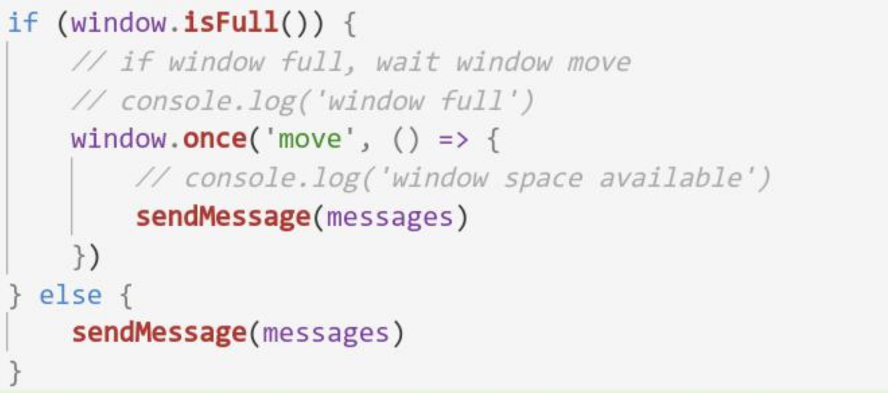
5.5 发送数据

用 sendMessage 发送一段数据，提取开头分组大小数据发送



```
const message = messages.slice(0, segmentSize)
messages = messages.slice(segmentSize)
if (message.byteLength === 0) return
sendSegment(window.nextseq, message)
log(['=> send segment', window.nextseq], message)
if (window.base === window.nextseq) {
    timeout.start()
}
window.addSeq(message)
```

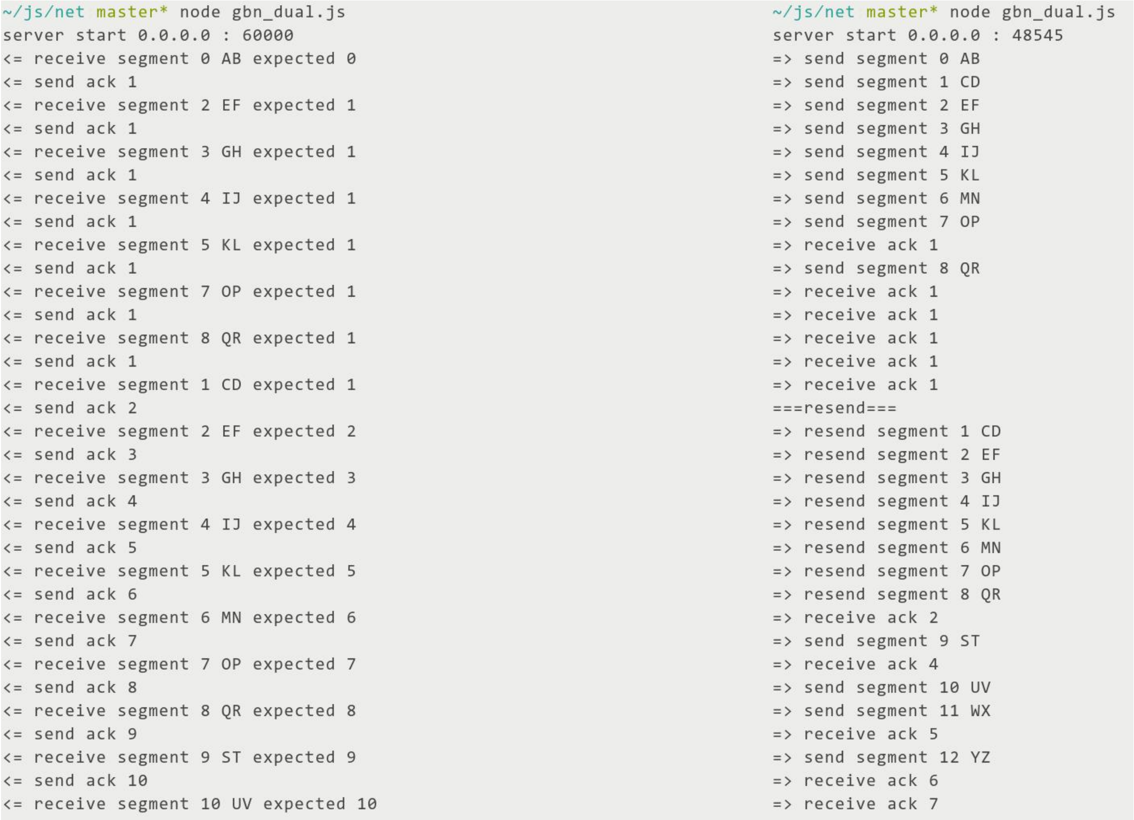
若窗口不满，发送余下数据，否则窗口移动发生时发送余下数据



```
if (window.isFull()) {
    // if window full, wait window move
    // console.log('window full')
    window.once('move', () => {
        // console.log('window space available')
        sendMessage(messages)
    })
} else {
    sendMessage(messages)
}
```

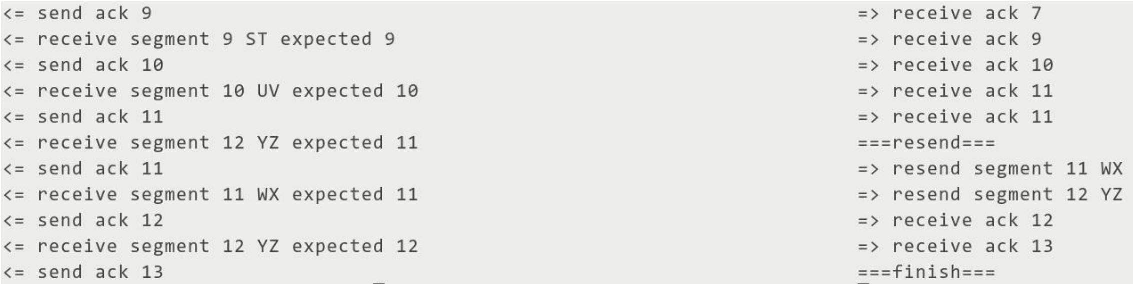
6 实验过程与结果

gbn 单向，右向左发送 A 到 Z，每个数据分组含 2 字节数据，丢包 0.2



```
~/js/net master* node gbn_dual.js
server start 0.0.0.0 : 60000
<= receive segment 0 AB expected 0
<= send ack 1
<= receive segment 2 EF expected 1
<= send ack 1
<= receive segment 3 GH expected 1
<= send ack 1
<= receive segment 4 IJ expected 1
<= send ack 1
<= receive segment 5 KL expected 1
<= send ack 1
<= receive segment 7 OP expected 1
<= send ack 1
<= receive segment 8 QR expected 1
<= send ack 1
<= receive segment 1 CD expected 1
<= send ack 2
<= receive segment 2 EF expected 2
<= send ack 3
<= receive segment 3 GH expected 3
<= send ack 4
<= receive segment 4 IJ expected 4
<= send ack 5
<= receive segment 5 KL expected 5
<= send ack 6
<= receive segment 6 MN expected 6
<= send ack 7
<= receive segment 7 OP expected 7
<= send ack 8
<= receive segment 8 QR expected 8
<= send ack 9
<= receive segment 9 ST expected 9
<= send ack 10
<= receive segment 10 UV expected 10
```

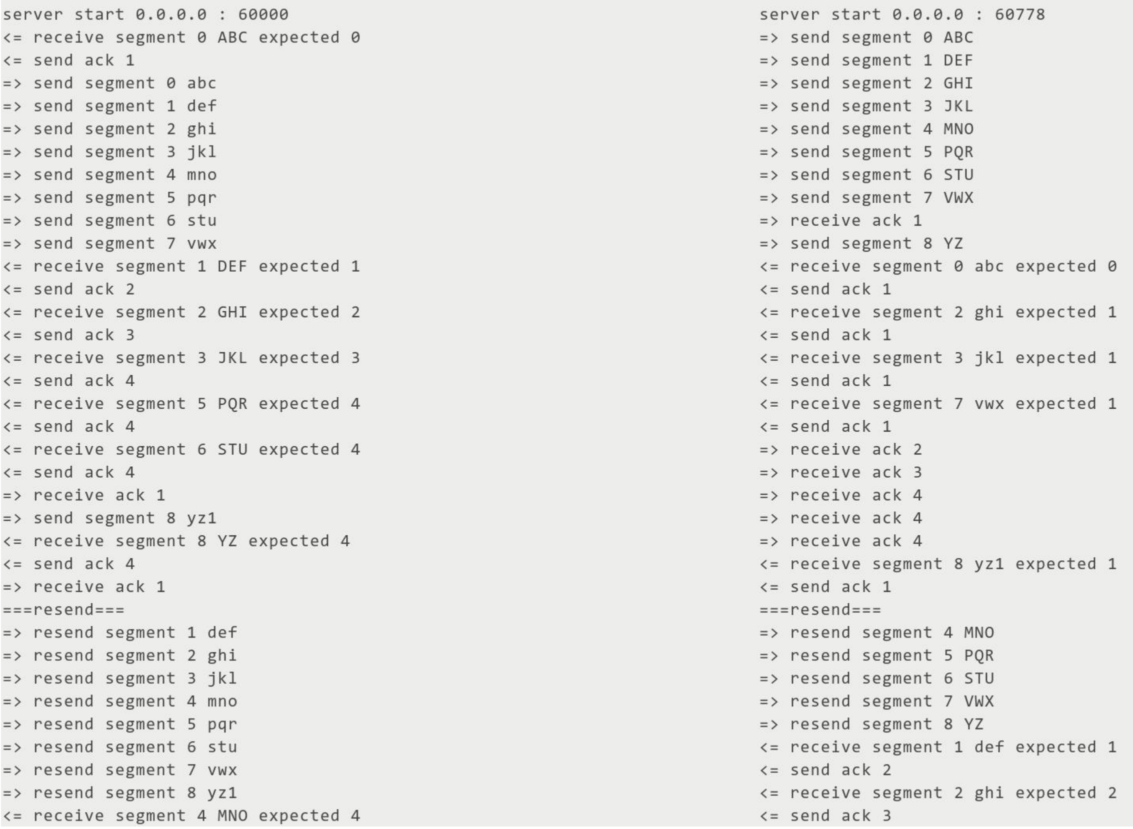
```
~/js/net master* node gbn_dual.js
server start 0.0.0.0 : 48545
=> send segment 0 AB
=> send segment 1 CD
=> send segment 2 EF
=> send segment 3 GH
=> send segment 4 IJ
=> send segment 5 KL
=> send segment 6 MN
=> send segment 7 OP
=> receive ack 1
=> send segment 8 QR
=> receive ack 1
=> receive ack 1
=> receive ack 1
=> receive ack 1
=> receive ack 1
===resend===
=> resend segment 1 CD
=> resend segment 2 EF
=> resend segment 3 GH
=> resend segment 4 IJ
=> resend segment 5 KL
=> resend segment 6 MN
=> resend segment 7 OP
=> resend segment 8 QR
=> receive ack 2
=> send segment 9 ST
=> receive ack 4
=> send segment 10 UV
=> send segment 11 WX
=> receive ack 5
=> send segment 12 YZ
=> receive ack 6
=> receive ack 7
```



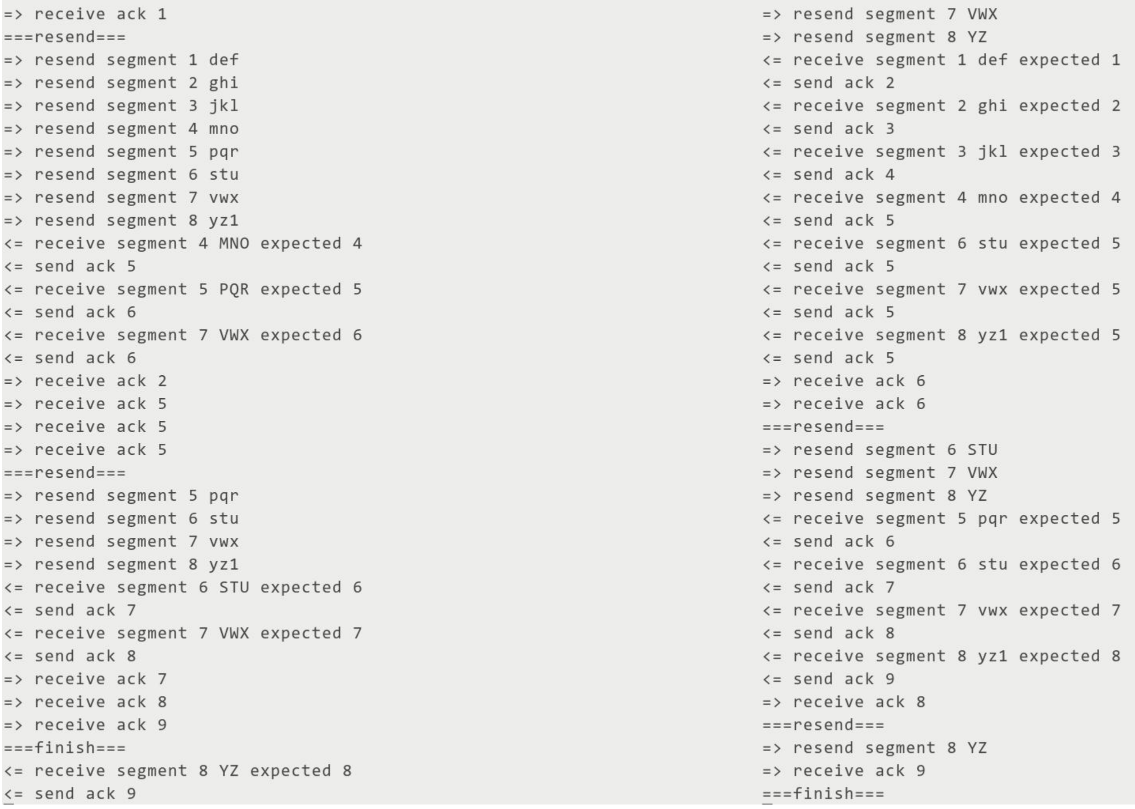
```
<= send ack 9
<= receive segment 9 ST expected 9
<= send ack 10
<= receive segment 10 UV expected 10
<= send ack 11
<= receive segment 12 YZ expected 11
<= send ack 11
<= receive segment 11 WX expected 11
<= send ack 12
<= receive segment 12 YZ expected 12
<= send ack 13
```

```
=> receive ack 7
=> receive ack 9
=> receive ack 10
=> receive ack 11
=> receive ack 11
===resend===
=> resend segment 11 WX
=> resend segment 12 YZ
=> receive ack 12
=> receive ack 13
===finish===
```

gbn 双向，右向左发送 A 到 Z，左向右发送 a 到 z，每个数据分组含 3 字节数据，丢包 0.2

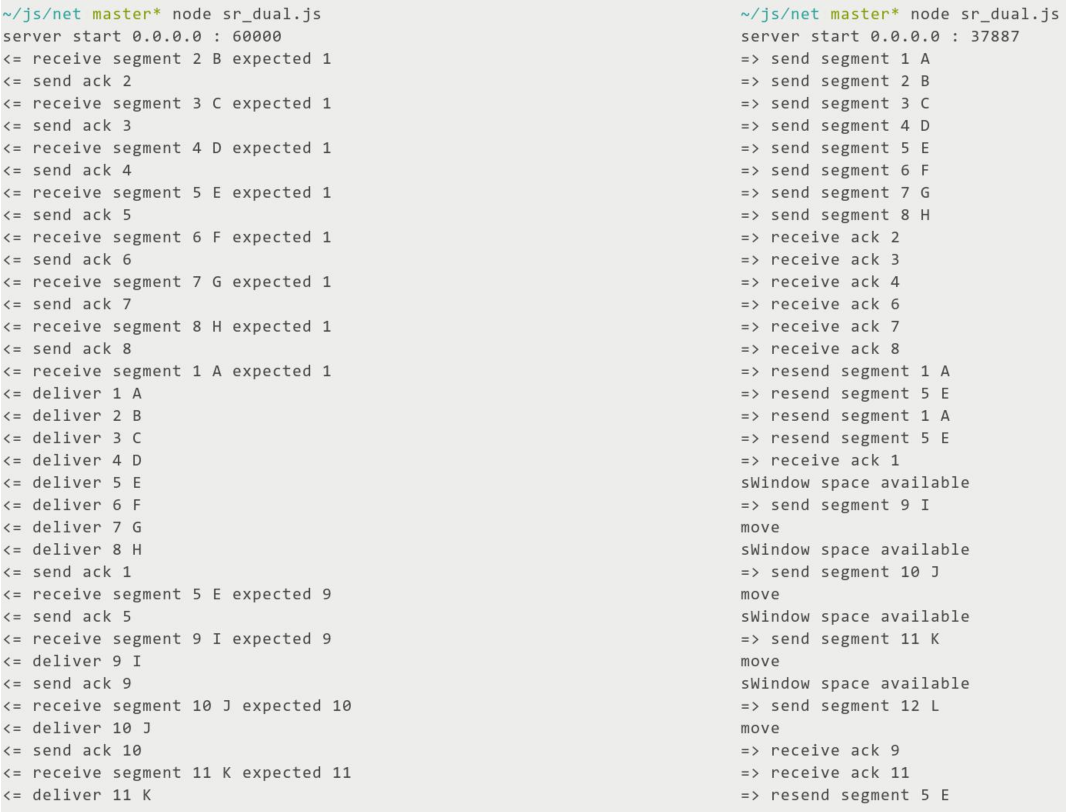


```
server start 0.0.0.0 : 60000            server start 0.0.0.0 : 60778
<= receive segment 0 ABC expected 0     => send segment 0 ABC
<= send ack 1                           => send segment 1 DEF
=> send segment 0 abc                   => send segment 2 GHI
=> send segment 1 def                   => send segment 3 JKL
=> send segment 2 ghi                   => send segment 4 MNO
=> send segment 3 jkl                   => send segment 5 PQR
=> send segment 4 mno                   => send segment 6 STU
=> send segment 5 pqr                   => send segment 7 VWX
=> send segment 6 stu                   => receive ack 1
=> send segment 7 vwx                   => send segment 8 YZ
<= receive segment 1 DEF expected 1     <= receive segment 0 abc expected 0
<= send ack 2                           <= send ack 1
<= receive segment 2 GHI expected 2     <= receive segment 2 ghi expected 1
<= send ack 3                           <= send ack 1
<= receive segment 3 JKL expected 3     <= receive segment 3 jkl expected 1
<= send ack 4                           <= send ack 1
<= receive segment 5 PQR expected 4     <= receive segment 7 vwx expected 1
<= send ack 4                           <= send ack 1
<= receive segment 6 STU expected 4     => receive ack 2
<= send ack 4                           => receive ack 3
=> receive ack 1                        => receive ack 4
=> send segment 8 yz1                   => receive ack 4
<= receive segment 8 YZ expected 4      => receive ack 4
<= send ack 4                           <= receive segment 8 yz1 expected 1
=> receive ack 1                        <= send ack 1
===resend===                            ===resend===
=> resend segment 1 def                 => resend segment 4 MNO
=> resend segment 2 ghi                 => resend segment 5 PQR
=> resend segment 3 jkl                 => resend segment 6 STU
=> resend segment 4 mno                 => resend segment 7 VWX
=> resend segment 5 pqr                 => resend segment 8 YZ
=> resend segment 6 stu                 <= receive segment 1 def expected 1
=> resend segment 7 vwx                 <= send ack 2
=> resend segment 8 yz1                 <= receive segment 2 ghi expected 2
<= receive segment 4 MNO expected 4     <= send ack 3
```



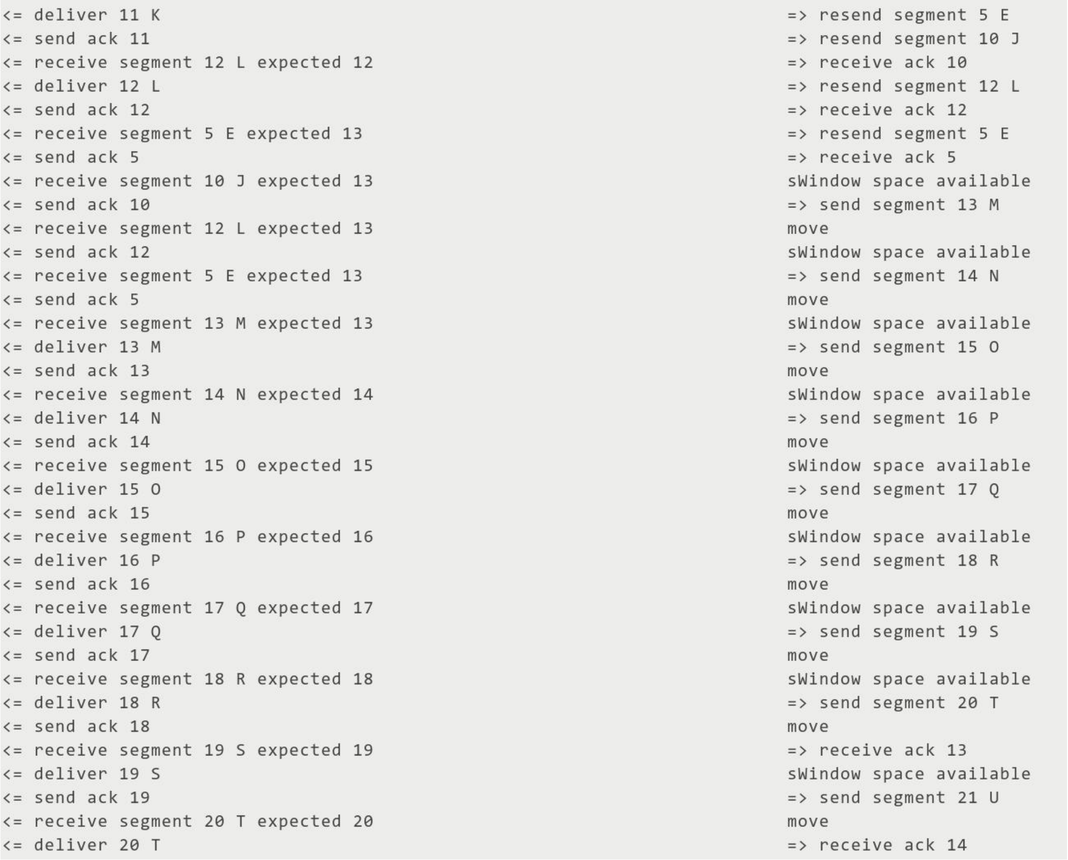
```
=> receive ack 1                        => resend segment 7 VWX
===resend===                            => resend segment 8 YZ
=> resend segment 1 def                 <= receive segment 1 def expected 1
=> resend segment 2 ghi                 <= send ack 2
=> resend segment 3 jkl                 <= receive segment 2 ghi expected 2
=> resend segment 4 mno                 <= send ack 3
=> resend segment 5 pqr                 <= receive segment 3 jkl expected 3
=> resend segment 6 stu                 <= send ack 4
=> resend segment 7 vwx                 <= receive segment 4 mno expected 4
=> resend segment 8 yz1                 <= send ack 5
<= receive segment 4 MNO expected 4     <= receive segment 6 stu expected 5
<= send ack 5                           <= send ack 5
<= receive segment 5 PQR expected 5     <= receive segment 7 vwx expected 5
<= send ack 6                           <= send ack 5
<= receive segment 7 VWX expected 6     <= receive segment 8 yz1 expected 5
<= send ack 6                           <= send ack 5
=> receive ack 2                        => receive ack 6
=> receive ack 5                        => receive ack 6
=> receive ack 5                        ===resend===
=> receive ack 5                        => resend segment 6 STU
===resend===                            => resend segment 7 VWX
=> resend segment 5 pqr                 => resend segment 8 YZ
=> resend segment 6 stu                 <= receive segment 5 pqr expected 5
=> resend segment 7 vwx                 <= send ack 6
=> resend segment 8 yz1                 <= receive segment 6 stu expected 6
<= receive segment 6 STU expected 6     <= send ack 7
<= send ack 7                           <= receive segment 7 vwx expected 7
<= receive segment 7 VWX expected 7     <= send ack 8
<= send ack 8                           <= receive segment 8 yz1 expected 8
=> receive ack 7                        <= send ack 9
=> receive ack 8                        => receive ack 8
=> receive ack 9                        ===resend===
===finish===                            => resend segment 8 YZ
<= receive segment 8 YZ expected 8      => receive ack 9
<= send ack 9                           ===finish===
```

sr 单向，右向左发送 A 到 Z，左向右发送 a 到 z，每个数据分组含 3 字节数据，丢包 0.2



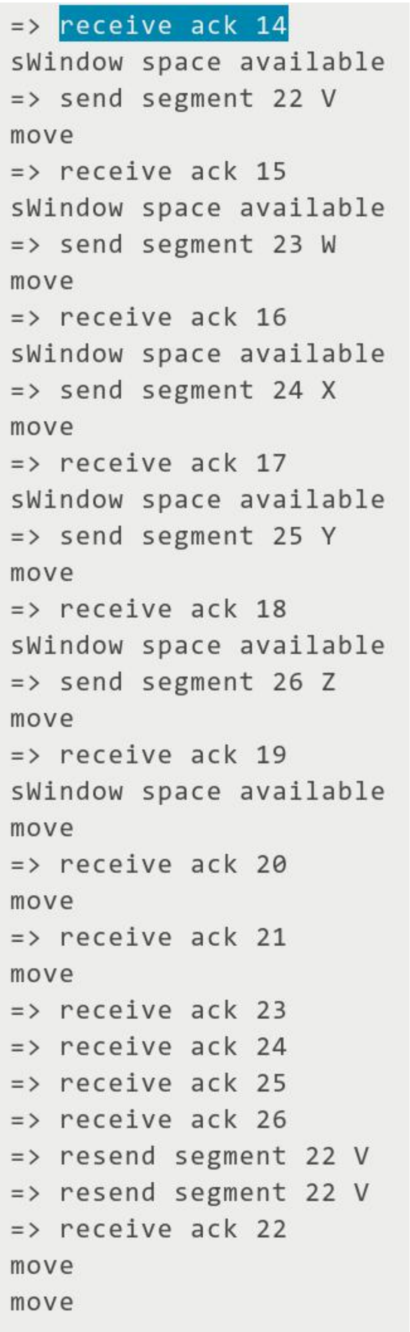
```
~/js/net master* node sr_dual.js
server start 0.0.0.0 : 60000
<= receive segment 2 B expected 1
<= send ack 2
<= receive segment 3 C expected 1
<= send ack 3
<= receive segment 4 D expected 1
<= send ack 4
<= receive segment 5 E expected 1
<= send ack 5
<= receive segment 6 F expected 1
<= send ack 6
<= receive segment 7 G expected 1
<= send ack 7
<= receive segment 8 H expected 1
<= send ack 8
<= receive segment 1 A expected 1
<= deliver 1 A
<= deliver 2 B
<= deliver 3 C
<= deliver 4 D
<= deliver 5 E
<= deliver 6 F
<= deliver 7 G
<= deliver 8 H
<= send ack 1
<= receive segment 5 E expected 9
<= send ack 5
<= receive segment 9 I expected 9
<= deliver 9 I
<= send ack 9
<= receive segment 10 J expected 10
<= deliver 10 J
<= send ack 10
<= receive segment 11 K expected 11
<= deliver 11 K
```

```
~/js/net master* node sr_dual.js
server start 0.0.0.0 : 37887
=> send segment 1 A
=> send segment 2 B
=> send segment 3 C
=> send segment 4 D
=> send segment 5 E
=> send segment 6 F
=> send segment 7 G
=> send segment 8 H
=> receive ack 2
=> receive ack 3
=> receive ack 4
=> receive ack 6
=> receive ack 7
=> receive ack 8
=> resend segment 1 A
=> resend segment 5 E
=> resend segment 1 A
=> resend segment 5 E
=> receive ack 1
sWindow space available
=> send segment 9 I
move
sWindow space available
=> send segment 10 J
move
sWindow space available
=> send segment 11 K
move
sWindow space available
=> send segment 12 L
move
=> receive ack 9
=> receive ack 11
=> resend segment 5 E
```

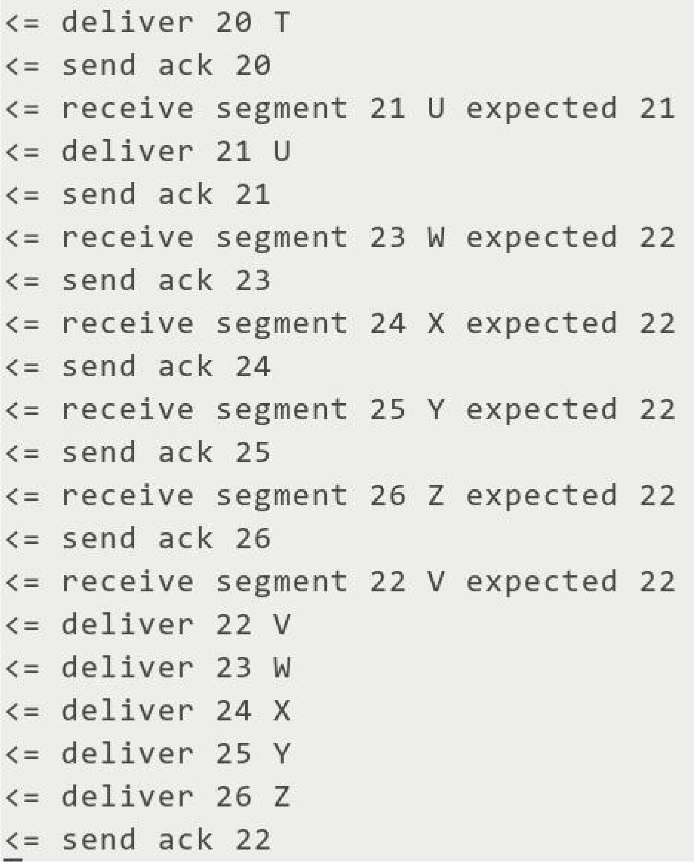


```
<= deliver 11 K
<= send ack 11
<= receive segment 12 L expected 12
<= deliver 12 L
<= send ack 12
<= receive segment 5 E expected 13
<= send ack 5
<= receive segment 10 J expected 13
<= send ack 10
<= receive segment 12 L expected 13
<= send ack 12
<= receive segment 5 E expected 13
<= send ack 5
<= receive segment 13 M expected 13
<= deliver 13 M
<= send ack 13
<= receive segment 14 N expected 14
<= deliver 14 N
<= send ack 14
<= receive segment 15 O expected 15
<= deliver 15 O
<= send ack 15
<= receive segment 16 P expected 16
<= deliver 16 P
<= send ack 16
<= receive segment 17 Q expected 17
<= deliver 17 Q
<= send ack 17
<= receive segment 18 R expected 18
<= deliver 18 R
<= send ack 18
<= receive segment 19 S expected 19
<= deliver 19 S
<= send ack 19
<= receive segment 20 T expected 20
<= deliver 20 T
```

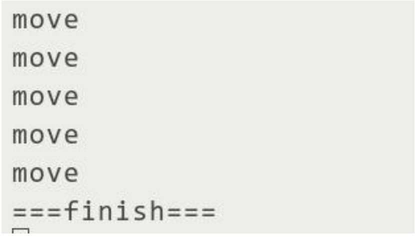
```
=> resend segment 5 E
=> resend segment 10 J
=> receive ack 10
=> resend segment 12 L
=> receive ack 12
=> resend segment 5 E
=> receive ack 5
sWindow space available
=> send segment 13 M
move
sWindow space available
=> send segment 14 N
move
sWindow space available
=> send segment 15 O
move
sWindow space available
=> send segment 16 P
move
sWindow space available
=> send segment 17 Q
move
sWindow space available
=> send segment 18 R
move
sWindow space available
=> send segment 19 S
move
sWindow space available
=> send segment 20 T
move
=> receive ack 13
sWindow space available
=> send segment 21 U
move
=> receive ack 14
```



```
=> receive ack 14
sWindow space available
=> send segment 22 V
move
=> receive ack 15
sWindow space available
=> send segment 23 W
move
=> receive ack 16
sWindow space available
=> send segment 24 X
move
=> receive ack 17
sWindow space available
=> send segment 25 Y
move
=> receive ack 18
sWindow space available
=> send segment 26 Z
move
=> receive ack 19
sWindow space available
move
=> receive ack 20
move
=> receive ack 21
move
=> receive ack 23
=> receive ack 24
=> receive ack 25
=> receive ack 26
=> resend segment 22 V
=> resend segment 22 V
=> receive ack 22
move
move
```



```
<= deliver 20 T
<= send ack 20
<= receive segment 21 U expected 21
<= deliver 21 U
<= send ack 21
<= receive segment 23 W expected 22
<= send ack 23
<= receive segment 24 X expected 22
<= send ack 24
<= receive segment 25 Y expected 22
<= send ack 25
<= receive segment 26 Z expected 22
<= send ack 26
<= receive segment 22 V expected 22
<= deliver 22 V
<= deliver 23 W
<= deliver 24 X
<= deliver 25 Y
<= deliver 26 Z
<= send ack 22
```



```
move
move
move
move
move
===finish===
```

sr 双向与前类似，不再演示

7 源代码（有详细注释）

7.1 gbn_dual.js

```
const EventEmitter = require('events')
const dgram = require('dgram')
const readline = require('readline')

let address = 'localhost'
let port = 60000
let messages = Buffer.from('abcdefghijklmnopqrstuvwxyz1')
let messages2 = Buffer.from('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
const dual = true
let dropRate = 0.2
let time = 500
let segmentSize = 3
const windowSize = 8
let expectedSeq = 0
class Window extends EventEmitter {
    constructor(num = 8) {
        super()
        this.num = num
        // interface index
        this.base = this.nextseq = 0
        // circular array index
        this.left = this.right = 0
        this.cache = [...Array(num)]
        this.cacheSize = 0
    }
    addIndex(x, y = 1) {
        return (x + y) % this.num
    }
    addBase(ack) {
        const before = this.base
        this.base = (ack === undefined ? this.base : ack)
        if (before < this.base) {
            this.left = this.addIndex(this.base, 0)
            if (this.right >= this.left) this.cacheSize = this.right - this.left
            else this.cacheSize = this.num - (this.left - this.right)
            this.emit('move')
            // console.log('window move', 'base:', this.base)
            return true
        }
        // wrong ack
    }
    addSeq(chunk) {
        // add to cache
        this.cache[this.right] = chunk
        this.right = this.addIndex(this.right)
        this.cacheSize++
        this.nextseq++
    }
    isFull() {
        return this.cacheSize === this.num
    }
    getCache() {
        if (this.isFull() || this.right < this.left) {
            return [...this.cache.slice(this.left), ...this.cache.slice(0, this.right)]
        } else {
            // this.right > this.left or empty
            return this.cache.slice(this.left, this.right)
        }
    }
}
class Timeout {
    constructor(fn, timeout) {
        this.fn = fn
        this.time = timeout
    }
    start() {
        if (!this.exist) {
            this.exist = true
            this.timeout = setTimeout(() => {
                this.exist = false
                this.fn()
            }, this.time)
        }
    }
```

```
        clear() {
            clearTimeout(this.timeout)
            this.exist = false
        }
        restart() {
            this.clear()
            this.start()
        }
    }
    const window = new Window()
    const timeout = new Timeout(reSend, time)
    const server = dgram.createSocket('udp4').on('listening', () => {
        const info = server.address()
        console.log('server start', info.address, ':', info.port)
    }).on('message', message)
    const log = (pre, message, next = [], limit = 10) => {
        if (message.byteLength <= limit)
            console.log(...pre, message.toString(), ...next)
        else
            console.log(...pre, 'byteLength', message.byteLength, ...next)
    }
    const sendSegment = (seq, data, callback) => {
        // data length > 1
        if (!Buffer.isBuffer(data) || data.byteLength === 0) return
        // one byte 0-255
        seq = Buffer.alloc(1, seq)
        data = Buffer.concat([seq, data])
        server.send(data, port, address, callback)
    }
    function message(message, rinfo) {
        // random drop ack and data
        if (Math.random() < (typeof dropRate === 'undefined' ? 0 : dropRate)) return
        // receive data
        if (message.byteLength > 1) {
            const seq = message[0]
            log(['<= receive segment', seq], message.slice(1), ['expected', expectedSeq])
            // expected ack
            if (seq === expectedSeq) expectedSeq++
            // send ack(not expected seq send ack too)
            port = rinfo.port
            server.send(Buffer.alloc(1, expectedSeq), port)
            console.log('<= send ack', expectedSeq)
            return
        }
        // receive ack, the first byte is ack
        const ack = message[0]
        console.log('=> receive ack', ack)
        const isRightAck = window.addBase(ack)
        // drop unrelated ack
        if (!isRightAck) return
        if (window.base !== window.nextseq) {
            timeout.restart()
        } else {
            timeout.clear()
            console.log('===finish===')
            server.emit('mfinish')
        }
    }
    function reSend() {
        console.log('===resend===')
        timeout.restart()
        window.getCache().forEach((chunk, index) => {
            sendSegment(window.base + index, chunk)
            log(['=> resend segment', window.base + index], chunk)
        })
    }
    function sendMessage(messages) {
        const message = messages.slice(0, segmentSize)
        messages = messages.slice(segmentSize)
        if (message.byteLength === 0) return
        sendSegment(window.nextseq, message)
        log(['=> send segment', window.nextseq], message)
        if (window.base === window.nextseq) {
            timeout.start()
        }
        window.addSeq(message)
        if (window.isFull()) {
            // if window full, wait window move
            // console.log('window full')
            window.once('move', () => {
                // console.log('window space available')
                sendMessage(messages)
            })
        } else {
            sendMessage(messages)
        }
    }
    const portAvailable = port => new Promise((resolve, reject) => {
        const tester = dgram.createSocket('udp4')
            .once('error', err => (err.code == 'EADDRINUSE' ? resolve(false) : reject(err)))
            .once('listening', () => tester.once('close', () => resolve(true)).close())
            .bind(port)
    })
    const bindPort = async () => {
        const result = await portAvailable(port)
        if (result) {
            server.bind(port)
            server.once('message', (_, rinfo) => {
                // get remote info
                port = rinfo.port
                address = rinfo.address
                // start send to remote
                if (typeof dual !== 'undefined' && dual) sendMessage(messages)
            })
        } else {
            // bind to another port
            server.bind(() => {
                sendMessage(messages2)
            })
        }
    }
    bindPort()
    const rl = readline.createInterface({
        input: process.stdin,
    })
```

```
server.once('mfinish', () => {
	rl.on('line', input => {
		// 3 byte for chinese charactor
		segmentSize = 3
		sendMessage(Buffer.from(input))
	})
})
```

7.2 sr_dual.js

```
const EventEmitter = require('events')
const dgram = require('dgram')
const readline = require('readline')
let address = 'localhost'
let port = 60000
let messages = Buffer.from('abcdefghijklmnopqrstuvwxyz1')
let messages2 = Buffer.from('ABCDEFGHIJKLMNOPQRSTUVWXYZ')
// const dual = true
let dropRate = 0.2
let time = 500
let segmentSize = 1
let windowSize = 8
class Timeout {
	constructor(fn, timeout) {
		this.fn = fn
		this.time = timeout
	}
	start() {
		if (!this.exist) {
			this.exist = true
			this.timeout = setTimeout(() => {
				this.exist = false
				this.fn()
			}, this.time)
		}
		return this
	}
	clear() {
		clearTimeout(this.timeout)
```

```
            this.exist = false
            return this
        }
        restart() {
            this.clear()
            this.start()
            return this

        }
    }
    class SWindow extends EventEmitter {
        constructor(num = 8) {
            super()
            this.num = num
            // interface index
            this.base = this.nextseq = 1
            // circular array index
            this.left = this.right = 1
            this.cache = [...Array(num)]
            this.timeout = [...Array(num)]
            this.cacheSize = 0
        }
        addIndex(x, y = 1) {
            return (x + y) % this.num
        }
        addBase(ack) {
            if (this.base <= ack && ack <= this.base + this.num) {
                // clear this timeout
                this.timeout[this.addIndex(ack, 0)].clear()
                this.cache[this.addIndex(ack, 0)] = undefined
                // this.timeout[this.addIndex(ack, 0)] = undefined
                // move
                while (this.cache[this.left] === undefined && this.cacheSize > 0) {
                    this.base++
                    this.left = this.addIndex(this.left)
                    this.cacheSize = (this.right >= this.left) ? this.cacheSize =
this.right - this.left
                        : this.num - (this.left - this.right)
                    this.emit('move')
                    console.log('move')
                }
                return true
            }

            // wrong ack
        }
        addSeq(chunk, ...args) {
            // add to cache
            this.cache[this.right] = chunk
            this.timeout[this.right] = (new Timeout(...args)).start()
            this.right = this.addIndex(this.right)
            this.cacheSize++
            this.nextseq++
        }
        isFull() {
            return this.cacheSize === this.num
        }
        // restart timeout
        restart(seq) {
            this.timeout[seq % this.num].restart()
        }
    }
    class RWindow extends EventEmitter {
        constructor(num = 8) {
            super()
            this.num = num
            // interface index
            this.base = 1
            // circular array index
            this.left = 1
            this.cache = [...Array(num)]
        }
        addIndex(x, y = 1) {
            return (x + y) % this.num
        }
        addBase(ack, chunk) {
            // not in [base-N,base+N-1]
            if (this.base - this.num > ack || ack >= this.base + this.num) return false
```

```
            if (this.base <= ack && this.cache[this.addIndex(ack, 0)] === undefined)
{
                // not duplicate
                this.cache[this.addIndex(ack, 0)] = chunk
                // deliver
                while (this.cache[this.left] !== undefined) {
                    this.emit('deliver', this.base, this.cache[this.left])
                    this.cache[this.left] = undefined
                    this.base++
                    this.left = this.addIndex(this.left)
                }
            }
            return true
        }
    }

    const log = (pre, message, next = [], limit = 10) => {
        if (message.byteLength <= limit)
            console.log(...pre, message.toString(), ...next)
        else
            console.log(...pre, 'byteLength', message.byteLength, ...next)
    }
    const sWindow = new SWindow(windowSize)
    const rWindow = new RWindow(windowSize)
    rWindow.on('deliver', (seq, chunk) => {
        log(['<= deliver', seq], chunk)
    })
    // const timeout = new Timeout(reSend, time)
    const server = dgram.createSocket('udp4').on('listening', () => {
        const info = server.address()
        console.log('server start', info.address, ':', info.port)
    }).on('message', message)
    const sendSegment = (seq, data, callback) => {
        // data length > 1
        if (!Buffer.isBuffer(data) || data.byteLength === 0) return
        // one byte 0-255
        seq = Buffer.alloc(1, seq)
        data = Buffer.concat([seq, data])
        server.send(data, port, address, callback)
    }
    function message(message, rinfo) {
        // random drop ack and data
        if (Math.random() < (typeof dropRate === 'undefined' ? 0 : dropRate)) return
        // receive data
        if (message.byteLength > 1) {
            const seq = message[0]
            log(['<= receive segment', seq], message.slice(1), ['expected', rWindow.base])
            // expected ack
            const isRightAck = rWindow.addBase(seq, message.slice(1))
            if (!isRightAck) return
            // send ack if ack in range
            port = rinfo.port
            address = rinfo.address
            server.send(Buffer.alloc(1, seq), port, address)
            console.log('<= send ack', seq)
            return
        }
        // receive ack, the first byte is ack
        const ack = message[0]
        console.log('=> receive ack', ack)
        const isRightAck = sWindow.addBase(ack)
        // drop unrelated ack
        if (!isRightAck) return
        if (sWindow.base === sWindow.nextseq) {
            console.log('===finish===')
            // if (typeof dual === 'undefined' || !dual) server.close()
            server.emit('mfinish')
        }
    }
    function reSend(seq, chunk) {
        sWindow.restart(seq)
        sendSegment(seq, chunk)
        log(['=> resend segment', seq], chunk)
    }
    function sendMessage(messages) {
        const message = messages.slice(0, segmentSize)
        messages = messages.slice(segmentSize)
        if (message.byteLength === 0) return
        sendSegment(sWindow.nextseq, message)
        log(['=> send segment', sWindow.nextseq], message)
        // sWindow.start(sWindow.nextseq, fn, time)
        sWindow.addSeq(message, ((seq, message) => () => reSend(seq, message))(sWindow.nextseq, message), time)
        if (sWindow.isFull()) {
            // if sWindow full, wait sWindow move
            sWindow.once('move', () => {
                console.log('sWindow space available')
                sendMessage(messages)
            })
        } else {
            sendMessage(messages)
        }
    }
    const portAvailable = port => new Promise((resolve, reject) => {
        const tester = dgram.createSocket('udp4')
            .once('error', err => (err.code == 'EADDRINUSE' ? resolve(false) : reject(err)))
            .once('listening', () => tester.once('close', () => resolve(true)).close())
            .bind(port)
    })
    const bindPort = async () => {
        const result = await portAvailable(port)
        if (result) {
            server.bind(port)
            server.once('message', (_, rinfo) => {
                // get remote info
                port = rinfo.port
                address = rinfo.address
                // start send to remote
                if (typeof dual !== 'undefined' && dual) sendMessage(messages)
            })
        } else {
            // bind to another port
            server.bind(() => {
                sendMessage(messages2)
            })
```

```
	}
}
bindPort()
const rl = readline.createInterface({
	input: process.stdin,
})
server.once('mfinish', () => {
	rl.on('line', input => {
		// 3 byte for chinese charactor
		segmentSize = 3
		sendMessage(Buffer.from(input))
	})
})
```