Multiplayer Game Programming

Lecture 3: TCP and Sockets

ITP 484

Problems with UDP

- Offers no guarantees over the underlying network layer, which means it too is Unreliable
  - Segments might arrive out of order
  - Segments aren't guaranteed to arrive at all

TCP: Transmission Control Protocol

- "Reliable"
  - Segments are received in order
  - Segments are "guaranteed" to arrive

```
struct TCPHeader
{
    uint16_t      mSourcePort;
    uint16_t      mDestinationPort;
    uint32_t      mSequenceNumber;
    uint32_t      mAcknowledgementNumber;
    unsigned int  mDataOffset : 4; //in 32bit words
    unsigned int  mUnused : 4;
    uint8_t       mFlags;
    uint16_t      mReceiveWindow;
    uint16_t      mChecksum;
    uint16_t      mUrgentPointer;
};
```

How does reliability work

- Receiver acknowledges segments
- Sender resends unacknowledged segment
- Sequence number increases by size of segment
- Acknowledgement number is sequence number of last received segment + the size of the segment's data

Examples

- Normal
- Lost segment in middle of stream
- Lost segment at end of stream
- Delayed ACK
  - Up to 500 ms or every other packet
- Lost ACK

Requirements

- Buffer for segments
  - So they can be resent (by sender)
  - So they can be processed in order (by receiver)
- Stateful Connection
  - Sequence number
  - Acknowledgement number
  - Buffers
  - Timers

How much to send per segment?

- Maximum Segment Size (MSS)
  - Don't Fragment (DF) Flag on Network Layer, causes segment to get dropped if too big

```
struct TCPHeader
{
    uint16_t      mSourcePort;
    uint16_t      mDestinationPort;
    uint32_t      mSequenceNumber;
    uint32_t      mAcknowledgementNumber;
    unsigned int  mDataOffset : 4;
    unsigned int  mUnused : 4;
    uint8_t       mFlags;
    uint16_t      mReceiveWindow;
    uint16_t      mChecksum;
    uint16_t      mUrgentPointer;
};
```

Flags

- CWR: Congestion Window Reduced
- ECE: Explicit Congestion Notification (ECN)-capable
- URG: Urgent pointer field is significant
- ACK: Acknowledgment field is significant
- PSH: Push function
- RST: Reset the connection
- SYN: Synchronize the sequence number
- FIN: Finished!

Handshake and Shutdown

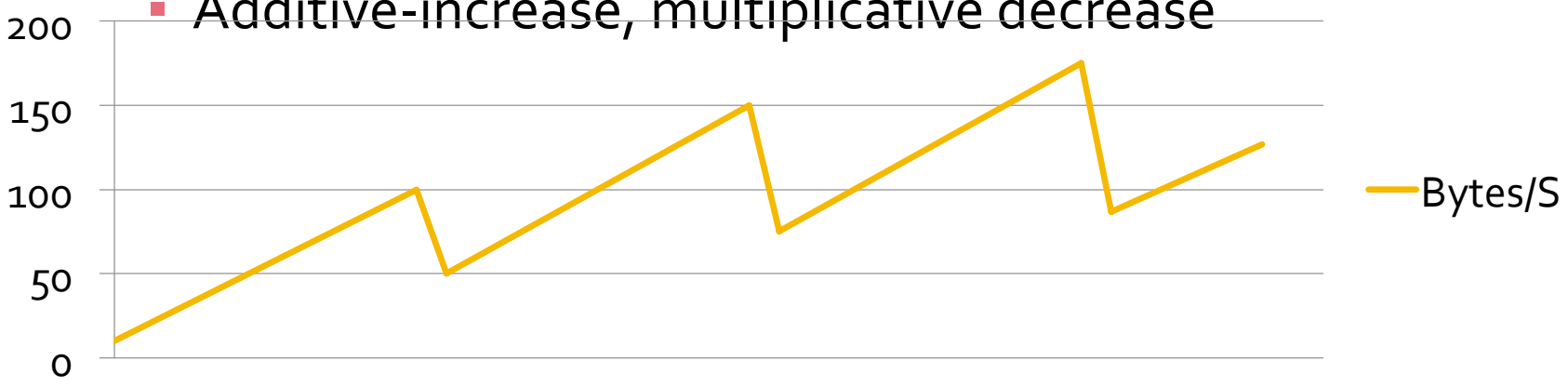
- Three way to establish
  - SYN->SYN,ACK->ACK
  - “Phantom Byte” sequence number increase
- Four way to shutdown
  - FIN->ACK…..FIN->ACK
  - ….. Receiver can finish sending or other activities

Encouraging Reliability

- Flow Control
  - Prevents receiver's data buffer getting full
  - Receive Window: maximum unack'd data
  - Max bandwidth?
    - Receive Window / Round Trip Time

Encouraging Reliability

- Congestion Control
  - Prevents network getting full
  - Congestion Window: maximum unack'd data
  - Up when no packets lost, Down when packets lost
    - Additive-increase, multiplicative decrease



Disadvantages of TCP

- Must establish a connection
- Higher latency than UDP
- No way to send packets unreliably

Sockets

- Berkeley Socket API
- Originally from BSD UNIX
- Ported in some way to almost every language and operating system

Operating System Differences

- POSIX / Berkeley Sockets
  - <sys/socket.h>
  - Socket is a file descriptor (int)
- Windows
  - <Winsock2.h>
  - Socket is a SOCKET

Creating a socket

```
SOCKET WSAAPI socket(
    _In_  int af,
    _In_  int type,
    _In_  int protocol );
```

```
af = [AF_INET|AF_INET6|...]
type =[SOCK_STREAM|SOCK_DGRAM|SOCK_RAW|...]
protocol =[ IPPROTO_TCP|IPPROTO_UDP|0|...]
```

Binding a socket to an address

```
int bind(
      _In_   SOCKET s,
      _In_   const struct sockaddr *name,
      _In_   int namelen );
```

- Actual type of sockaddr can change based on protocol family, etc.

SOCKADDR

```
struct sockaddr
{
      ushort sa_family;
      char sa_data[14];
};

struct sockaddr_in
{
      short sin_family;
      u_short sin_port;
      struct in_addr sin_addr;
      char sin_zero[8];
};
```

IN_ADDR IPv4 address

```
typedef struct in_addr
{
	union
	{
		struct
		{
			u_char s_b1,s_b2,s_b3,s_b4;
		} S_un_b;
		struct
		{
			u_short s_w1,s_w2;
		} S_un_w;
		u_long S_addr;
	} S_un;
} IN_ADDR;
```

INADDR_ANY will allow you to bind to all local interfaces on the host

Creating an address from a string

```
int getaddrinfo(
      const char FAR* nodename,
      const char FAR* servname,
      const struct addrinfo FAR* hints,
      struct addrinfo FAR* FAR* res );
```

- nodename: dns name or ip address
- Servname: port number or service name
- hints: can specify address family, etc
- res: results. Must be deallocated with `freeaddrinfo`

Byte Order

- Network Byte Order is Big Endian
- Your platform is probably Little Endian
- Use conversion functions when manually filling in or reading address structures

```
u_short htons( u_short hostshort );
u_long htonl( u_long hostlong);
u_short ntohs( u_short netshort);
u_long ntohl(u_long netlong);
```

Datagram Transmission

```
int sendto(
    _In_   SOCKET s,
    _In_   const char *buf,
    _In_   int len,
    _In_   int flags,
    _In_   const struct sockaddr *to,
    _In_   int tolen );
```

- Returns number of bytes sent or a negative error number

Datagram Reception

```
int recvfrom(
      _In_          SOCKET s,
      _Out_         char *buf,
      _In_          int len,
      _In_          int flags,
      _Out_         struct sockaddr *from,
      _Inout_opt_   int *fromlen );
```

- Returns number of bytes sent or a negative error number

Stream Connections

```
int listen(
    _In_  SOCKET s,
    _In_  int backlog );
```

- backlog: number of connections that can be pending acceptance
- Returns 0 for no error

Stream Connections

```
SOCKET accept(
    _In_      SOCKET s,
    _Out_     struct sockaddr
*addr,
    _Inout_   int *addrlen );
```

- Returns a new socket, and its address info

Stream Connections

```
int connect(
   _In_  SOCKET s,
   _In_  const struct sockaddr *name,
   _In_  int namelen );
```

Blocking

- Sending, receiving, connecting and accepting all block by default if there's no space, data, pending connection available or just not enough time to complete

```
int ioctlsocket(
        _In_      SOCKET s,
        _In_      long cmd,
        _Inout_   u_long *argp );
```

Pass FIONBIO as cmd and and argp that points to the number 1 to enable nonblocking mode
In nonblocking mode, many sockets will return what looks like and error, but when you call `WSAGetLastError` you'll see the error is just WSAEWOULDBLOCK. This means there's no result yet because the operation hasn't completed yet

Blocking with Select

```
int select(
    _In_      int nfds,
    _Inout_   fd_set *readfds,
    _Inout_   fd_set *writefds,
    _Inout_   fd_set *exceptfds,
    _In_      const struct timeval *timeout );
```

Returns number of ready sockets or negative error.

Sets are modified by function

Select Socket Set

`FD_ZERO(fd_set* set )`

Initializes set

`FD_SET(SOCKET s, fd_set* set )`

Add socket to set

`FD_ISSET(SOCKET s, fd_set* set )`

Tests if a socket is set

`FD_CLR(SOCKET s, fd_set* set )`

Removes socket from set

Initialization and Shutdown

```
int WSAStartup(
   _In_   WORD wVersionRequested,
   _Out_  LPWSADATA lpWSAData );
```

Use MAKEWORD(2, 2) for version

```
int WSACleanup(void);
```