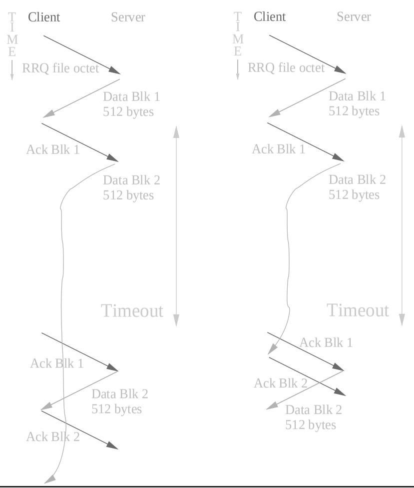
TFTP - duplications

- Packets delivered twice
- Several cases
  - RRQ / WRQ
  - Data
  - ACK
  - Error

TFTP Duplicate DATA

- Data packets carry block number
  - duplicate detection easy
- duplicates expected (loss recovery)



TFTP Duplicate ACK

- ACK is never duplicate
  - Always treated as ACK
  - Always transmit next block
    - (if any)
- Required for lost data packet recovery
  - May cause duplicate data packet
  - That is handled

TFTP Duplicate requests

- RRQ (or WRQ) is duplicated
- Each RRQ (WRQ) initiates a new connection
  - Server replies to each
    - from a different port number
- Client sees 2 replies
  - Knows it sent only one request
  - picks one reply - ignores the other

TFTP - out of order

- Packets delivered in wrong order
  - rare - only one outstanding packet
  - block number in data and ACK
- good enough

TFTP simultaneous transfers

- Multiple requests from one client
- Multiple requests to one server
- Port numbers differ
  - client picks a port number unique in its system
  - server sends from a port number unique in its system
- Together with IP address
  - those port numbers allow many requests

TFTP Large file transfer

- Block number in each packet
  - Block number is 16 bits
- Max block number is 65535
  - Each block 512 bytes
- Biggest file 65535 * 0.5KB
  - just under 32 MB.
- Some implementations
  - Simply continue
    - block numbers wrap
    - no file size limit

TFTP Throughput

- Work out maximum possible throughput of TFTP
  - Assume 1 Gbps (1000Mbps) ethernet & routers
  - Assume 1 ms RTT through network
  - Assume no delays in hosts
- 1ms RTT
- Lock Step protocol
  - 1000 round trips per second
- 1000 round trips, 512 bytes each time
  - 512 K Bytes/second

Increasing throughput

- Lock step protocol cannot work
  - we often cannot reduce the RTT
- So, need to allow multiple packets per RTT
  - send many packets before ACK of first
- Can send so much
  - that network cannot take more
    - (has its own problems)
  - need flow control
- TCP works this way

TFTP - History

- RFC1123
  - Hosts Requirements (October 1989)
    - Sorcerer's Apprentice Bug Fix
    - Requires adaptive timeouts
    - Deprecates "mail" transfer mode
- RFC1350
  - TFTP (Revision 2) (July 1992)
    - No significant changes
    - Includes RFC1123 mods into spec
- RFC1782
  - TFTP Option Extension (March 1995)
    - Allows options in RQ packets
    - Options follow mode, option name, value
    - OACK (new packet type) to accept
      - contains accepted options & values

TFTP History (continued)

- RFC1783 - TFTP Blocksize option
    - Allows altering the 512 to another number
    - Client sends its request in xRQ
    - Server picks size (<=) and sends OACK
- RFC1784 - TFTP Timeout Interval & Transfer Size
    - Allows altering timeout
    - Allows knowing/specifying file size
- RFC2347/2348/2349
  - Updates for RFC1782/3/4 (May 1998)

TFTP - Summary

- Protocol Issues
  - Text (English) Description
  - Network Byte Order
  - NetASCII
  - Errors using number & text
  - (newer) Option enhancement
- Protocol Evaluation
  - Imperfect
  - Adequate for its purpose

UDP

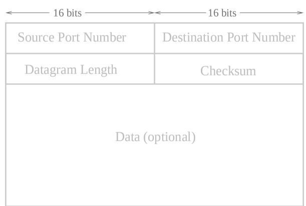
- User Datagram Protocol
  - RFC 768
    - August 1980

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism.

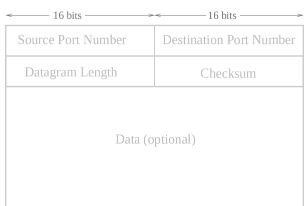
The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed.

User Datagram Protocol (UDP)



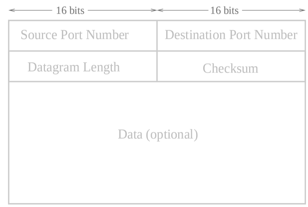
Source Port is an optional field, when meaningful, it indicates the port of the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.

User Datagram Protocol (UDP)



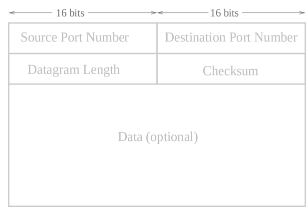
Destination Port has a meaning within the context of a particular internet destination address.

User Datagram Protocol (UDP)



Length is the length in octets of this user datagram including this header and the data. (This means the minimum value of the length is eight.)

User Datagram Protocol (UDP)



Checksum is the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded with zero octets at the end (if necessary) to make a multiple of two octets.

Pseudo Header

- Pseudo == False (Fake, Pretend)

- Values taken from IP header
- Similar version defined for IPv6 (128 bit addresses)

Checksum Algorithm

- Inverse of 1's complement sum
  - of 16 bit words in data checksummed
- If result is 0, use -0 instead (0xFFFF)

```
◦ 1AE3 + 34C2  ==>  4FA5
◦ 4FA5 + C207  ==>  11AD
  2's complement, unlimited size:
      4FA5+C207==111AC

◦ Overflow is ignored,
    ▹ but every time overflow occurs,
    ▹ we have been past both -0 and 0
          ie: have added 2
                        result is add of 1
          so need to add an extra 1
```

UDP Checksums

- UDP Checksum is optional
  - Value of 0 indicates no checksum present
    - if checksum value had been 0
      - it would be represented as -0 (or FFFF)
  - Intended for applications where
    - data integrity is not important
  - But most link layers checksum packets,
      - and drop those with errors
    - Omitting checksum doesn't help much
  - And checksum also includes port numbers
    - packet could be delivered to wrong application,
    - or reply sent to wrong place
- Hence UDP checksum
  - now recommended to always be used.
  - mandatory in IPv6

UDP Evaluation

- Does the protocol work?
  - Lost packets?
  - Corrupted packets?
  - Out of order packets?
  - Duplicated packets?
  - Multiple simultaneous transfers?
- It promises very little.
  - How can it not be correct?