End-to-End Protocols

Guevara Noubir

Textbook: Computer Networks: A Systems Approach,
L. Peterson, B. Davie, Morgan Kaufmann

Chapter 5.

Lecture Outline

- Connection
- Establishment/Termination
- Sliding Window Revisited
- Flow Control
- Adaptive Timeout

- Overview of Remote Procedure Call

End-to-End Protocols

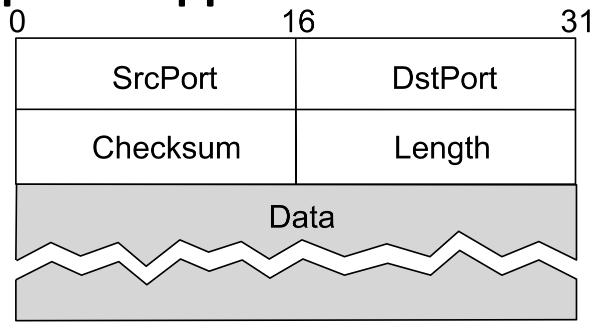
- Goal: turn host-to-host packet delivery into process-to-process communication channel
- Underlying best-effort network
  - drop messages
  - re-orders messages
  - delivers duplicate copies of a given message
  - limits messages to some finite size
  - delivers messages after an arbitrarily long delay
- Common end-to-end services
  - guarantee message delivery
  - deliver messages in the same order they are sent
  - deliver at most one copy of each message
  - support arbitrarily large messages
  - support synchronization
  - allow the receiver to flow control the sender
  - support multiple application processes on each host

Types of End-to-End Protocols

- Simple asynchronous demultiplexing service (e.g., UDP)
- Reliable byte-stream service (e.g., TCP)
- Request reply service (e.g., RPC)

Simple Demultiplexor (UDP)

- Unreliable and unordered datagram service
- Adds multiplexing
- No flow control
- Endpoints identified by ports
  - servers have *well-known* ports (e.g., DNS: port 53, talk: 517)
  - On Unix see **`/etc/services`** and **port mapper**
- Header format



0 16 31

SrcPort DstPort

Checksum Length

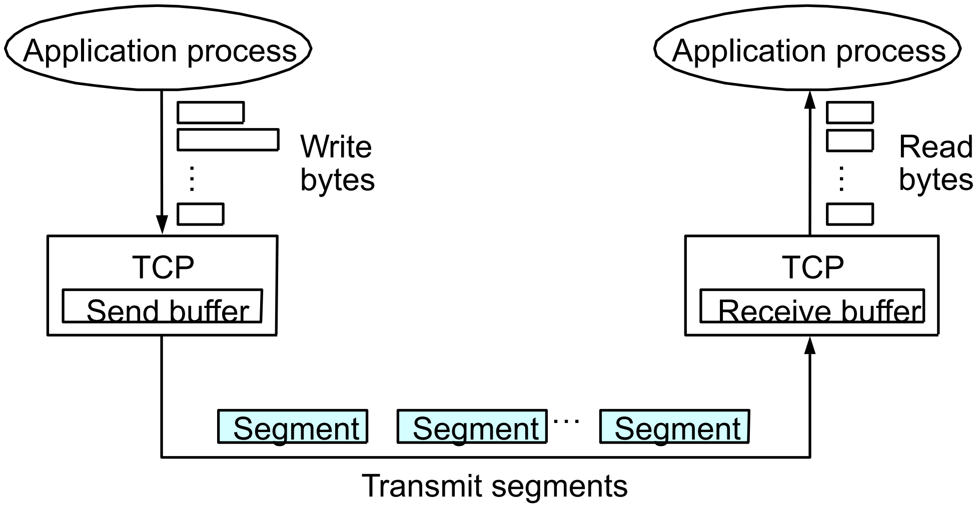
Data

- Optional checksum
  - pseudo header + UDP header + data
  - Pseudo header = protocol number, source IP addr, dest IP addr, UDP length

TCP Overview

- Reliable
- Connection-oriented
- Byte-stream
  - app writes bytes
  - TCP sends *segments*
  - app reads bytes

- Full duplex
- Flow control: keep sender from overrunning receiver
- Congestion control: keep sender from overrunning network



Data Link Versus Transport

- Potentially connects many different hosts
  - need explicit connection establishment and termination
- Potentially different RTT
  - need adaptive timeout mechanism
- Potentially long delay in network
  - need to be prepared for arrival of very old packets
- Potentially different capacity at destination
  - need to accommodate different node capacity
- Potentially different network capacity
  - need to be prepared for network congestion

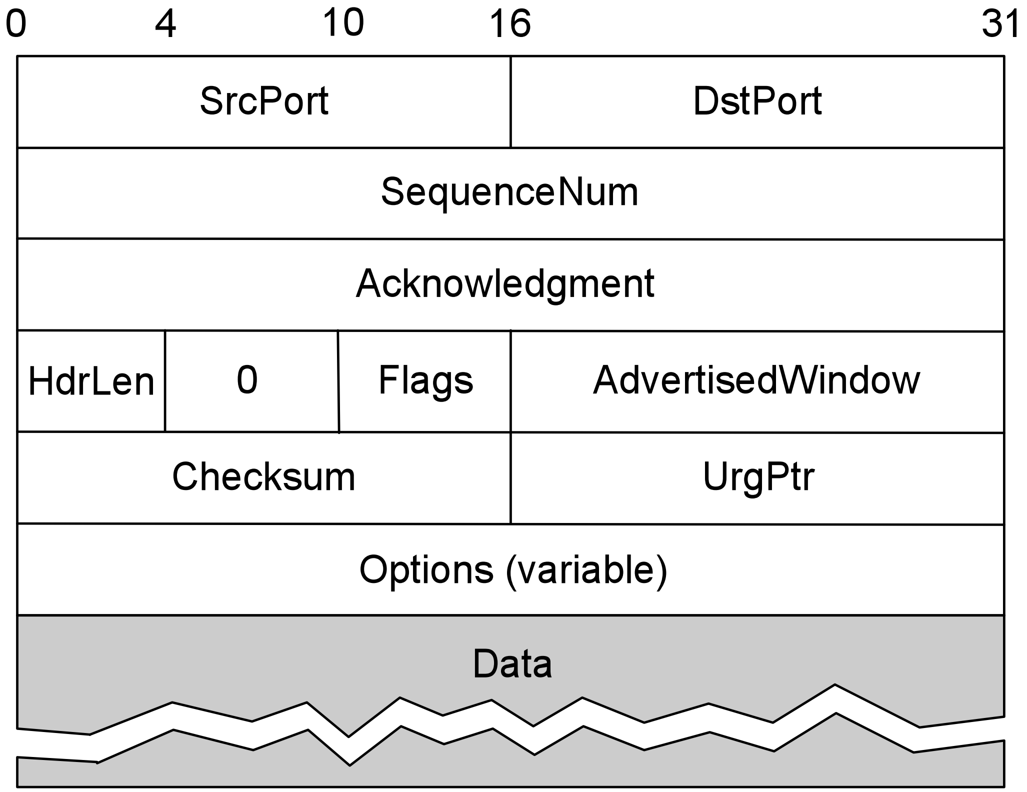
End-to-End Argument

- A function should not be provided in the lower levels of the system unless it can be completely and correctly implemented

- Exception: optimization
  - Example: CRC at layer 2 + checksum at layer 4

- The end-to-end argument has to be revisited for wireless networks

Segment Format



0 4 10 16 31

SrcPort DstPort

SequenceNum

Acknowledgment

HdrLen 0 Flags AdvertisedWindow

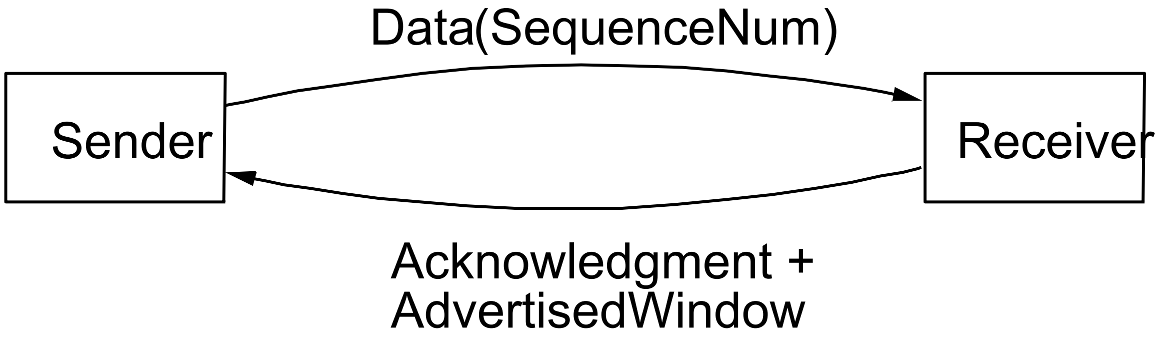
Checksum UrgPtr

Options (variable)

Data

Segment Format (cont)

- Each connection identified with 4-tuple:
  - **`(SrcPort, SrcIPAddr, DsrPort, DstIPAddr)`**
- Sliding window + flow control
  - **`acknowledgment, SequenceNum, AdvertisedWindow`**

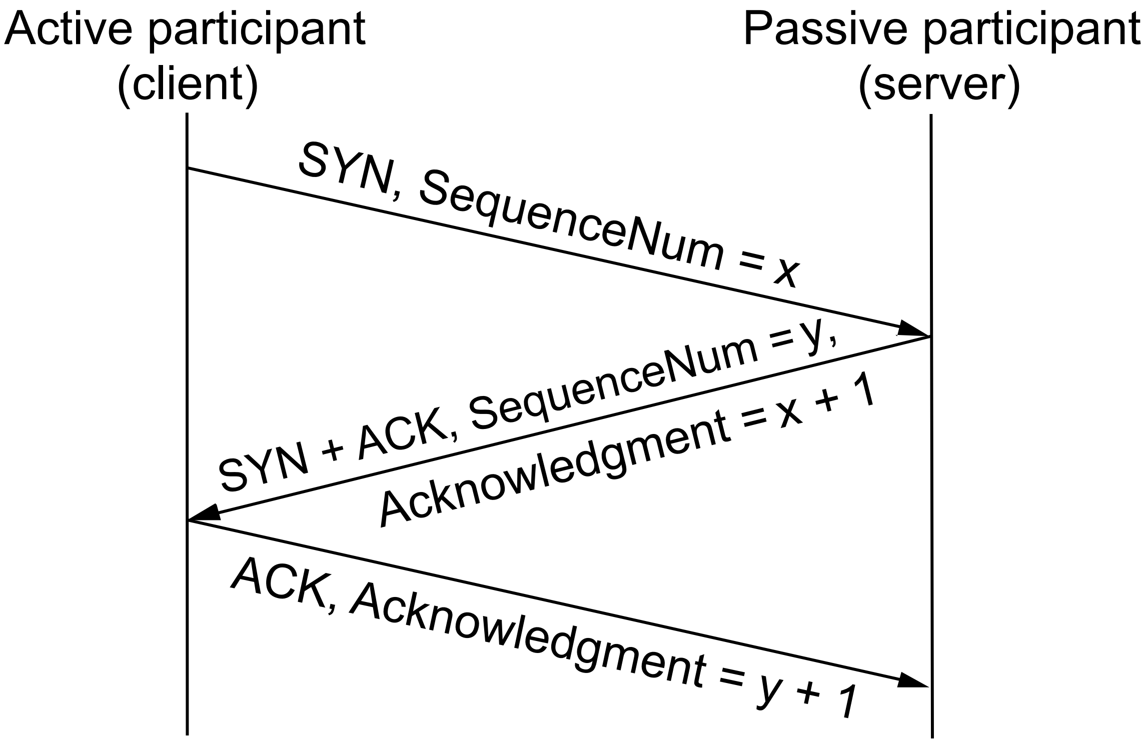


- Flags
  - **`SYN, FIN, RESET, PUSH, URG, ACK`**
- Checksum
  - pseudo header + TCP header + data

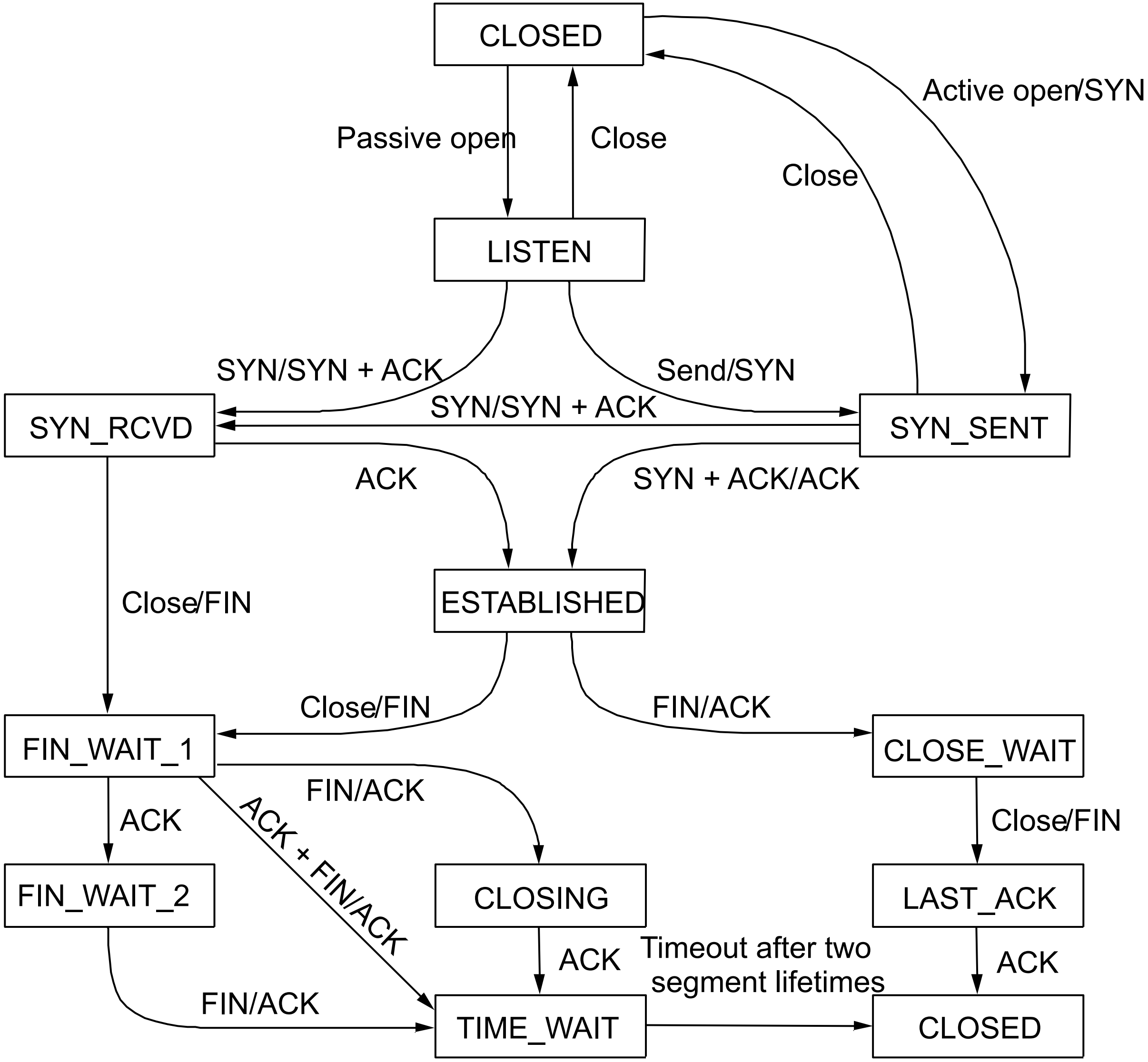
Segments Transmission

- Transmission of segments can be triggered by:
  - When the data to be sent reaches: Maximum Segment Size (MSS). MSS is usually equal to the longest segment that won't result in local IP fragmentation
  - Request from the application: Push operation (e.g., ssh, telnet)
  - Periodic timer

Connection Establishment and Termination



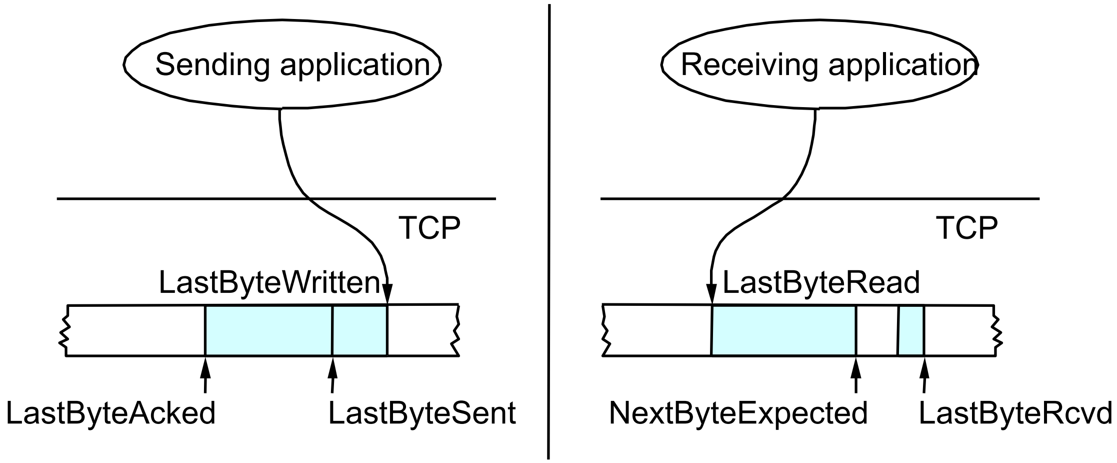
State Transition Diagram



Sliding Window in TCP

- Purpose:
  - Guarantees a reliable delivery of data (ARQ)
  - Ensures that data is delivered in order (SeqNum)
  - Enforces flow-control between sender and receiver (AdvertisedWindow field)

Sliding Window Revisited



- Sending side
  - **LastByteAcked <= LastByteSent**
  - **LastByteSent <= LastByteWritten**
  - buffer bytes between **LastByteAcked** and **LastByteWritten**

- Receiving side
  - **LastByteRead < NextByteExpected**
  - **NextByteExpected <= LastByteRcvd +1**
  - buffer bytes between **LastByteRead** and **LastByteRcvd**

Flow Control

- Send buffer size: **`MaxSendBuffer`**
- Receive buffer size: **`MaxRcvBuffer`**

- Receiving side
  - **`LastByteRcvd`** - **`LastByteRead`** < = **`MaxRcvBuffer`**
  - **`AdvertisedWindow`** = **`MaxRcvBuffer`** - (**`LastByteRcvd`** - **`LastByteRead`**)
- Sending side
  - **`LastByteSent`** - **`LastByteAcked`** < = **`AdvertisedWindow`**
  - **`EffectiveWindow`** = **`AdvertisedWindow`** - (**`LastByteSent`** - **`LastByteAcked`**)
  - **`LastByteWritten`** - **`LastByteAcked`** < = **`MaxSendBuffer`**
  - block sender if (**`LastByteWritten`** - **`LastByteAcked`**) + *y* > **`MaxSenderBuffer`**

- Always/only send ACK in response to arriving data segment
- Persist when **`AdvertisedWindow = 0`**

Protection Against Wrap Around

- 32-bit **`SequenceNum`**

Bandwidth	Time Until Wrap Around
T1 (1.5 Mbps)	6.4 hours
Ethernet (10 Mbps)	57 minutes
T3 (45 Mbps)	13 minutes
FDDI (100 Mbps)	6 minutes
STS-3 (155 Mbps)	4 minutes
STS-12 (622 Mbps)	55 seconds
STS-24 (1.2 Gbps)	28 seconds

Keeping the Pipe Full

- 16-bit **AdvertisedWindow**
  - **(assuming an RTT ~100ms)**

Bandwidth	Delay x Bandwidth Product
T1 (1.5 Mbps)	18KB
Ethernet (10 Mbps)	122KB
T3 (45 Mbps)	549KB
FDDI (100 Mbps)	1.2MB
STS-3 (155 Mbps)	1.8MB
STS-12 (622 Mbps)	7.4MB
STS-24 (1.2 Gbps)	14.8MB

TCP Extensions

- Implemented as header options:
  - Why?
- Store timestamp in outgoing segments
- Extend sequence space with 32-bit timestamp (PAWS: Protection Against Wrapped Sequences)
- Shift (scale) advertised window
  - Count in 16 bytes
- Selective Acknowledgment (SACK)

Adaptive Retransmission (Original Algorithm)

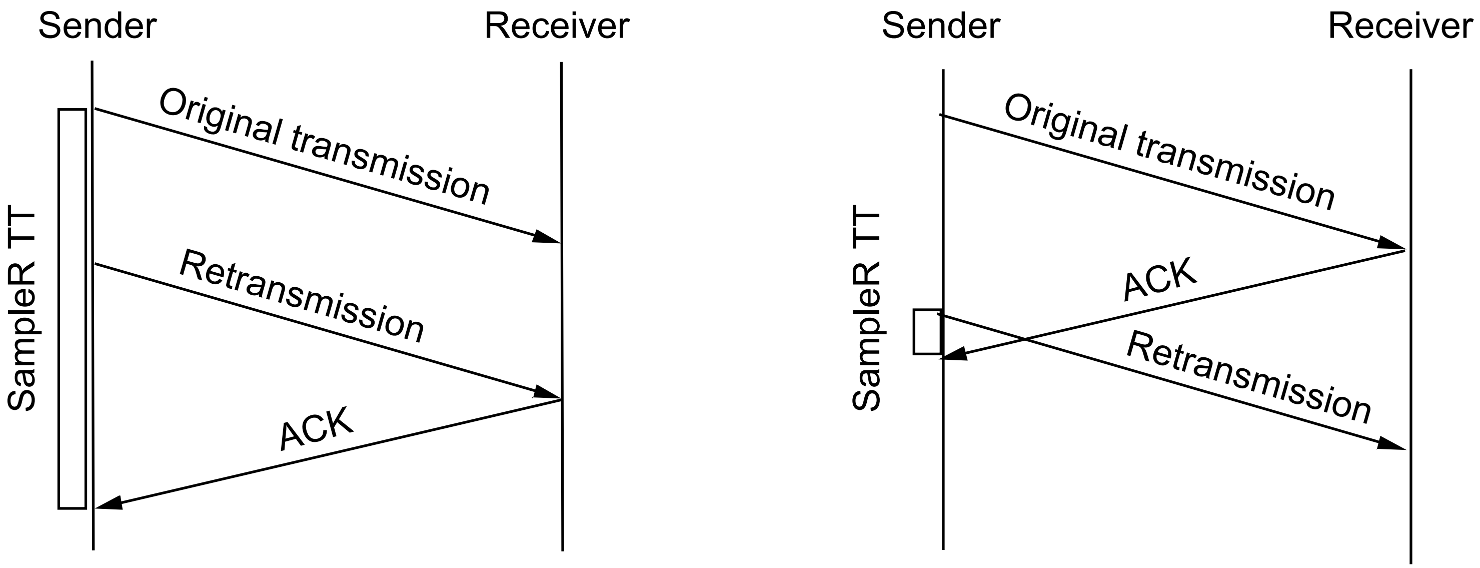
- Measure **`SampleRTT`** for each segment/ ACK pair
- Compute weighted average of RTT
  - **`EstRTT`** = α x **`EstRTT`** + β x **`SampleRTT`**
  - where $\alpha + \beta$ **`= 1`**
  - α between 0.8 and 0.9
  - β between 0.1 and 0.2
- Set timeout based on **`EstRTT`**
  - **`TimeOut`** = 2 x **`EstRTT`**

Problem with Original Algorithm

- ACK indicate receipt of data and not of packet

- If ACK corresponds to retransmitted packet than estimated RTT would be too large. The reverse is also possible.

Karn/Partridge Algorithm



- Do not sample RTT when retransmitting
- Double timeout after each retransmission

Jacobson/ Karels Algorithm

- New Calculations for average RTT
- **Diff** = **SampleRTT** - **EstRTT**
- **EstRTT = EstRTT + (δ x Diff)**
- **Dev = Dev + δ(|Diff| - Dev)**
  - where δ is a factor between 0 and 1
- Consider variance when setting timeout value
- **TimeOut** = μ x **EstRTT** + ϕ x **Dev**
  - where typically $\mu = 1$ and $\phi = 4$
- Notes
  - algorithm only as good as granularity of clock (500ms on old Unix)
  - accurate timeout mechanism important to congestion control (later)

Record Boundaries

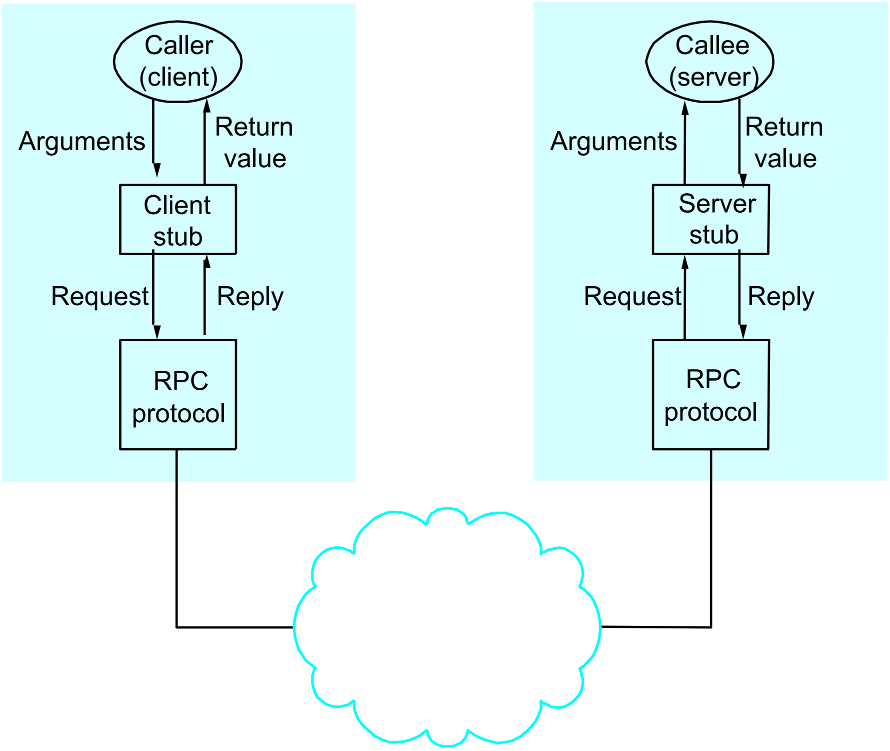
- TCP is a byte-stream protocol
- How to indicate some structure within the stream?
  - URG flag + UrgPtr (*out-of-band* data). Initially designed for urgent data.
  - PUSH mechanism:
    - Initially and still used by interactive applications
    - Can also be used to break the received stream into records
  - Application program

Remote Procedure Calls (RPC)

- RPC:
  - Generic mechanism for structuring distributed systems
- Components:
  - Protocol: manages the messages sent between the client and the server processes and handles network issues
  - Programming language and compiler support:
    - Arguments translation from one machine architecture to another…

RPC Components

- Protocol Stack
  - BLAST: fragments and reassembles large messages
  - CHAN: synchronizes request and reply messages
  - SELECT: dispatches request to the correct process
- Stubs



Transport for Real-Time Applications

- Applications with time constraints on packets delivery
  - E.g., VoIP, interactive video, multimedia
- Constraints include
  - Deadlines,
  - Multiple streams synchronization (e.g., audio/video)
  - Interoperability, e.g., codecs negotiation
  - Packet loss detection
  - Frame boundaries
- Example RTP + RTCP
  - Run on top of UDP

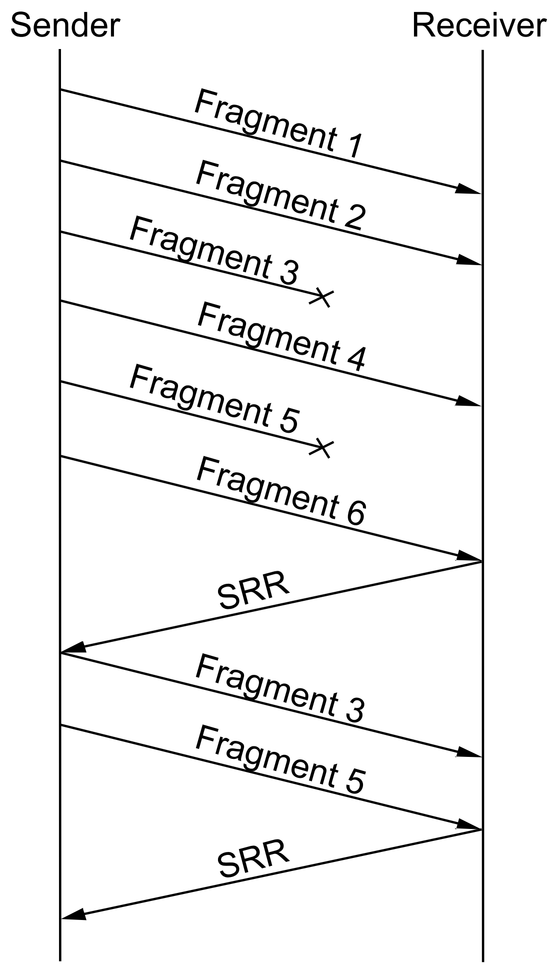
RTP & RTCP

- Realtime Transport Protocol
  - Multimedia date exchange
- Realtime Transport Control Protocol
  - Periodic control information e.g., statistics on packets loss, inter-arrival jitter, sender identity
- Run on consecutive UDP ports
- Flexible and low overhead to support a wide variety of applications
  - through *profiles*, and *formats*,
  - compact header format (ver., padding, extension, # contributing sources, payload type (e.g., codecs), sequence number, timestamp, synchronization source (SSRC), contributing source (CSRC)

Bulk Transfer (BLAST)

- Unlike AAL and IP, tries to recover from lost fragments

- Strategy
  - selective retransmission
  - partial acknowledgements



BLAST Details

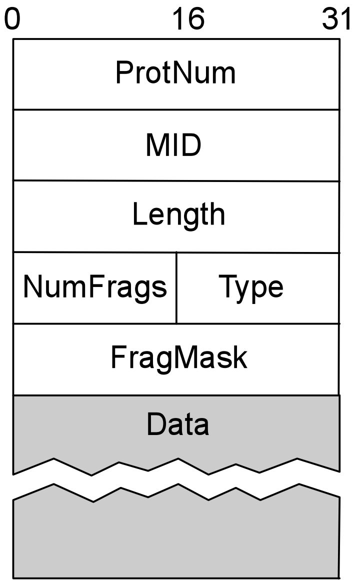
- Sender:
  - after sending all fragments, set timer DONE
  - if receive SRR, send missing fragments and reset DONE
  - if timer DONE expires, free fragments

BLAST Details (cont)

- Receiver:
  - when first fragments arrives, set timer LAST_FRAG
  - when all fragments present, reassemble and pass up
  - four exceptional conditions:
    - if last fragment arrives but message not complete
      - send SRR and set timer RETRY
    - if timer LAST_FRAG expires
      - send SRR and set timer RETRY
    - if timer RETRY expires for first or second time
      - send SRR and set timer RETRY
    - if timer RETRY expires a third time
      - give up and free partial message

BLAST Header Format

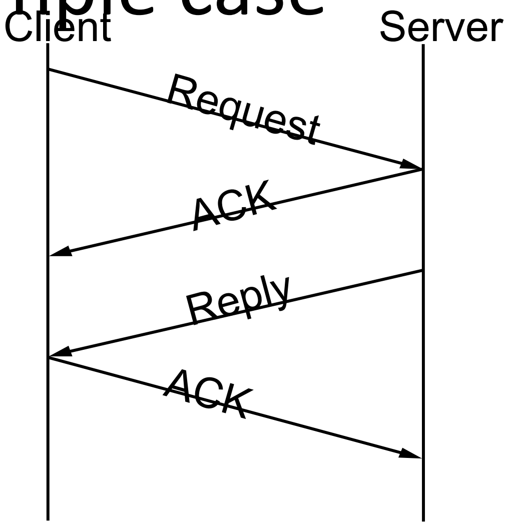
- MID must protect against wrap around
- TYPE = DATA or SRR
- NumFrags indicates number of fragments
- FragMask distinguishes among fragments
  - if Type=DATA, identifies this fragment
  - if Type=SRR, identifies missing fragments



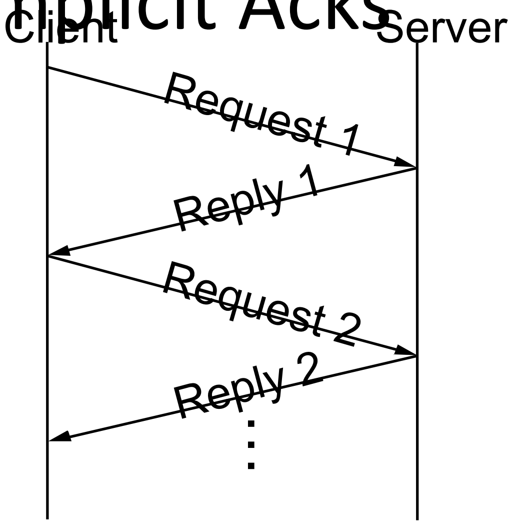
Request/Reply (CHAN)

- Guarantees message delivery
- Synchronizes client with server
- Supports *at-most-once* semantics

Simple case



Implicit Acks



CHAN Details

- Lost message (request, reply, or ACK)
  - set RETRANSMIT timer
  - use message id (MID) field to distinguish
- Slow (long running) server
  - client periodically sends "are you alive" probe, or
  - server periodically sends "I'm alive" notice
- Want to support multiple outstanding calls
  - use channel id (CID) field to distinguish
- Machines crash and reboot
  - use boot id (BID) field to distinguish

Synchronous vs Asynchronous Protocols

- Asynchronous interface

```
xPush(Sessn s, Msg *msg)
xPop(Sessn s, Msg *msg, void *hdr)
xDemux(Protl hlp, Sessn s, Msg *msg)
```

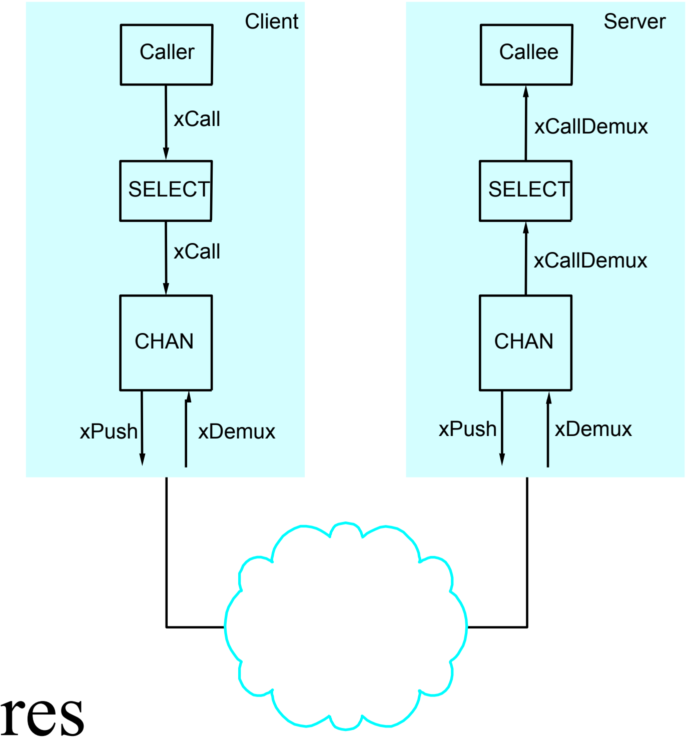
- Synchronous interface

```
xCall(Sessn s, Msg *req, Msg *rep)
xCallPop(Sessn s, Msg *req, Msg *rep, void *hdr)
xCallDemux(Protl hlp, Sessn s, Msg *req, Msg *rep)
```

- CHAN is a hybrid protocol
  - synchronous from above: **`xCall`**
  - asynchronous from below: **`xPop/xDemux`**

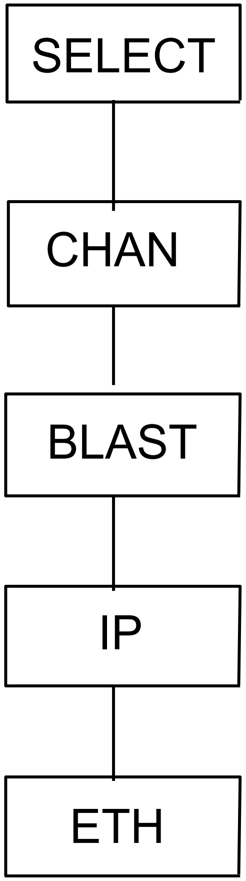
Dispatcher (SELECT)

- Dispatch to appropriate procedure
- Synchronous counterpart to UDP



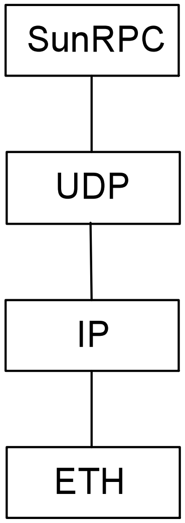
- Address Space for Procedures
  - flat: unique id for each possible procedure
  - hierarchical: program + procedure number

Simple RPC Stack



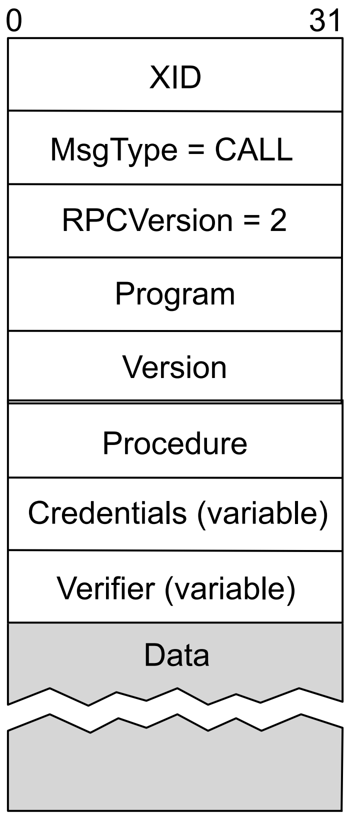
SunRPC

- IP implements BLAST-equivalent
  - except no selective retransmit
- SunRPC implements CHAN-equivalent
  - except not at-most-once
- UDP + SunRPC implement SELECT-equivalent
  - UDP dispatches to program (ports bound to programs)
  - SunRPC dispatches to procedure within program



SunRPC Header Format

- XID (transaction id) is similar to CHAN's MID
- Server does not remember last XID it serviced
- Problem if client retransmits request while reply is in transit



0 31

XID

MsgType = CALL

RPCVersion = 2

Program

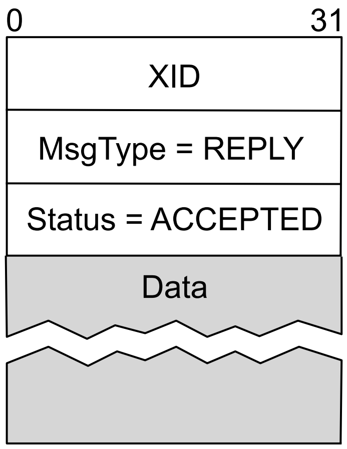
Version

Procedure

Credentials (variable)

Verifier (variable)

Data



0 31

XID

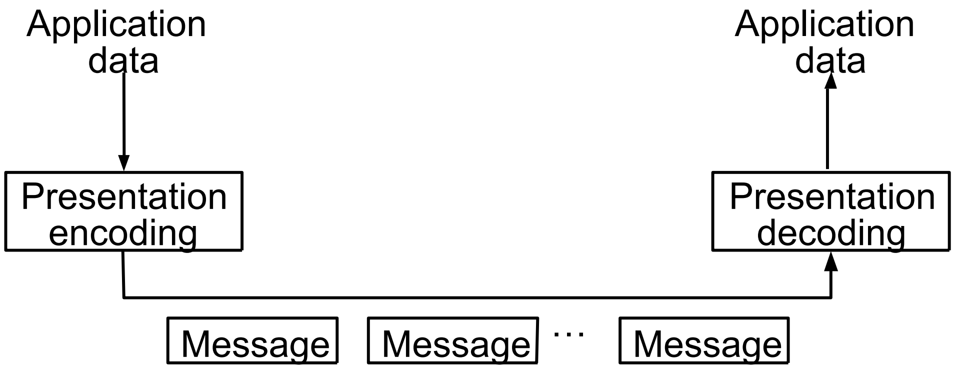
MsgType = REPLY

Status = ACCEPTED

Data

Presentation Formatting

- Marshalling (encoding) application data into messages
- Unmarshalling (decoding) messages into application data

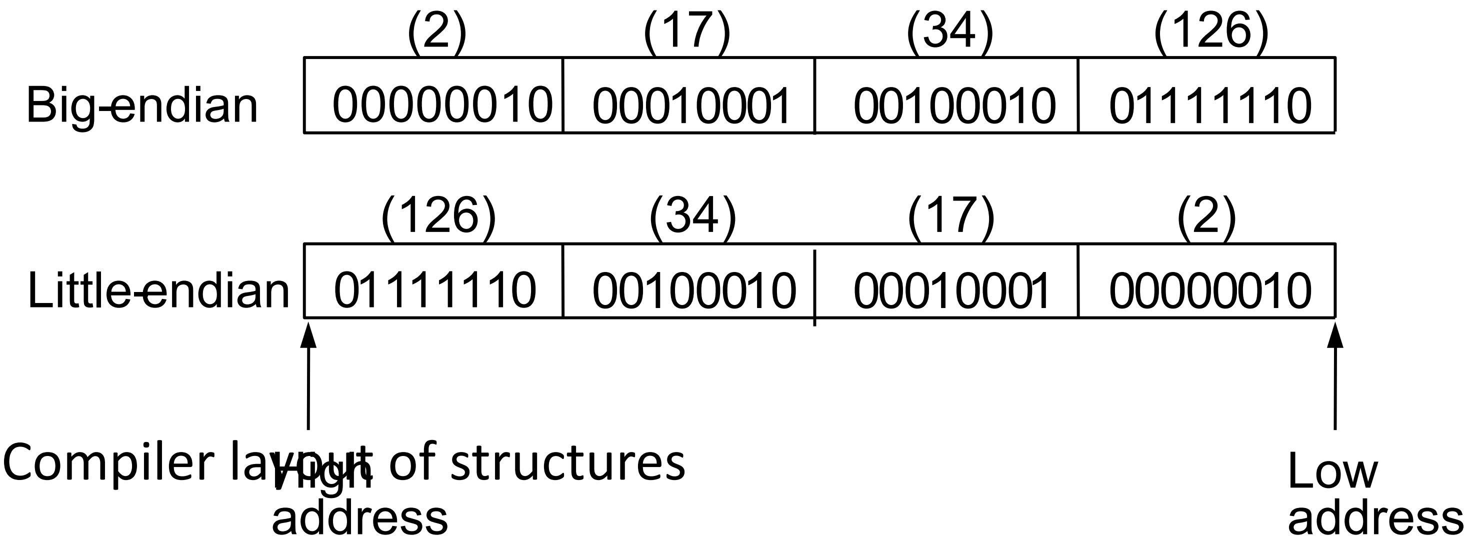


- Data types we consider
  - integers
  - floats
  - strings
  - arrays
  - structs

- Types of data we do not consider
  - images
  - video
  - multimedia documents

Difficulties

- Representation of base types
  - floating point: IEEE 754 versus non-standard
  - integer: big-endian versus little-endian (e.g., 34,677,374)

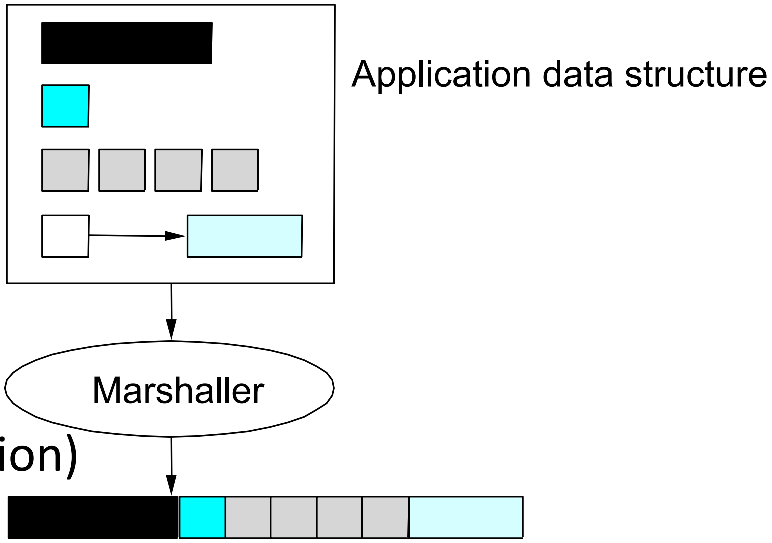


	(2)	(17)	(34)	(126)
Big-endian	00000010	00010001	00100010	01111110
	(126)	(34)	(17)	(2)
Little-endian	01111110	00100010	00010001	00000010

- Compiler layout of structures

Taxonomy

- Data types
  - base types (e.g., ints, floats); must convert
  - flat types (e.g., structures, arrays); must pack
  - complex types (e.g., pointers); must linearize



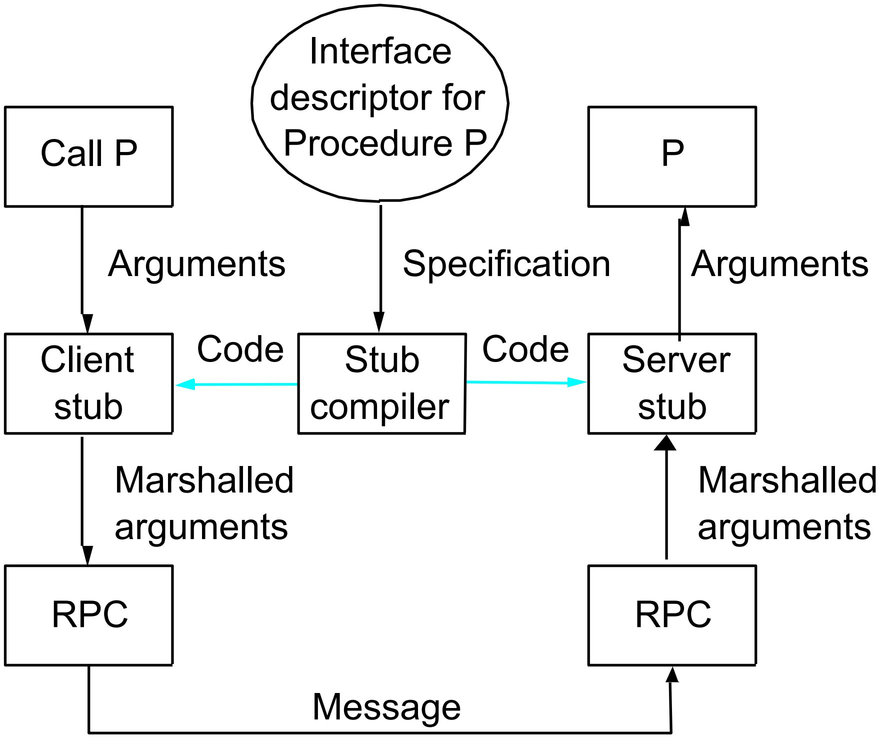
- Conversion Strategy
  - canonical intermediate form
  - receiver-makes-right (an *N* x *N* solution)

Taxonomy (cont)

- Tagged versus untagged data

type = INT	len = 4	value = 417892

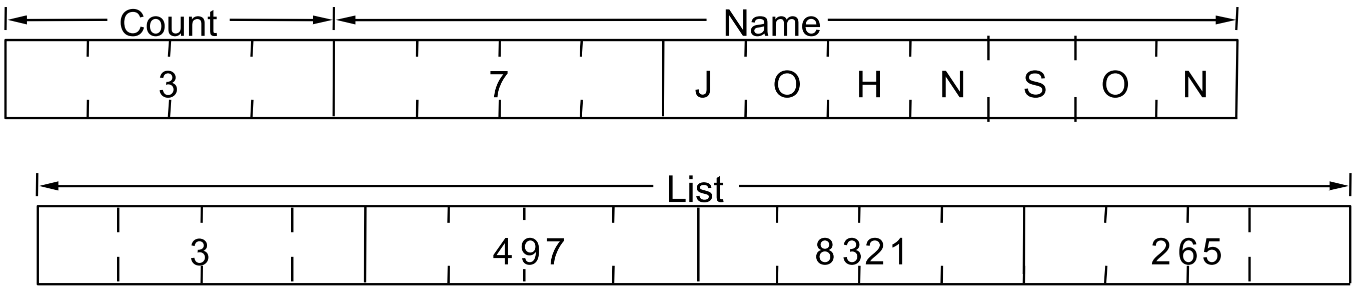
- Stubs
  - compiled
  - interpreted



eXternal Data Representation (XDR)

- Defined by Sun for use with SunRPC
- C type system (without function pointers)
- Canonical intermediate form
- Untagged (except array length)
- Compiled stubs

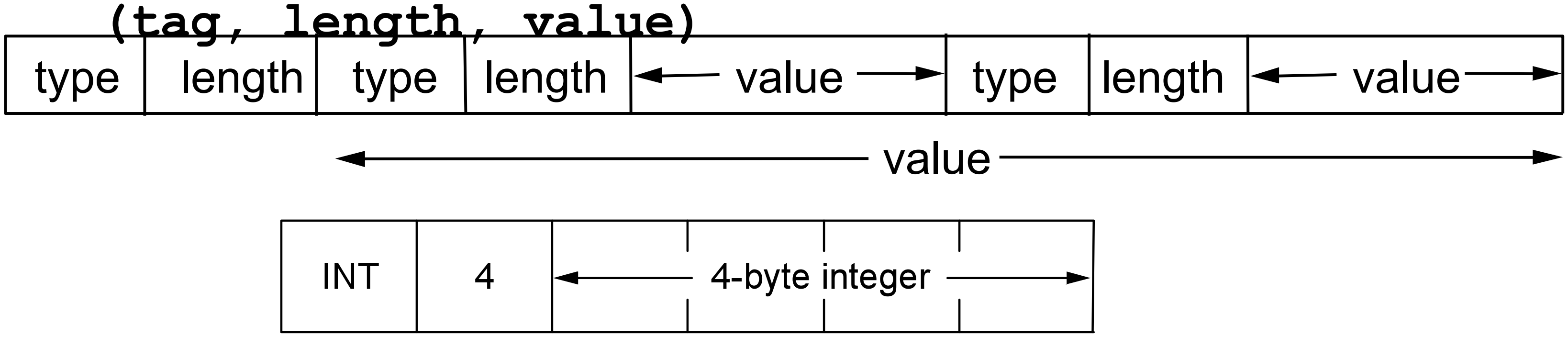
```
#define MAXNAME 256;
#define MAXLIST 100;

struct item {
   int     count;
   char    name[MAXNAME];
   int     list[MAXLIST];
};

bool_t
xdr_item(XDR *xdrs, struct item *ptr)
{
   return(xdr_int(xdrs, &ptr->count) &&
       xdr_string(xdrs, &ptr->name, MAXNAME) &&
       xdr_array(xdrs, &ptr->list, &ptr->count,
                 MAXLIST, sizeof(int), xdr_int));
}
```



Abstract Syntax Notation One (ASN-1)

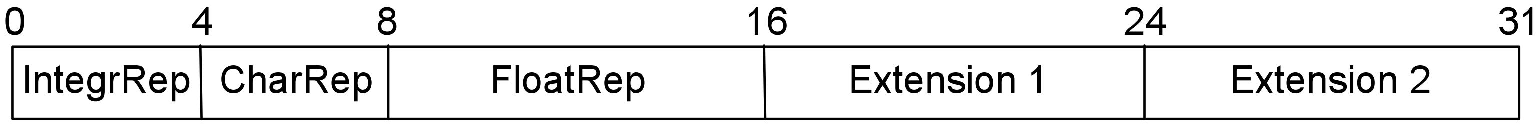
- An ISO standard
- Essentially the C type system
- Canonical intermediate form
- Tagged
- Compiled or interpreted stubs
- BER: Basic Encoding Rules

`(tag, length, value)`



Network Data Representation (NDR)

- Defined by DCE
- Essentially the C type system
- Receiver-makes-right (architecture tag)
- Individual data items untagged
- Compiled stubs from IDL
- 4-byte architecture tag
  - IntegerRep
    - 0 = big-endian
    - 1 = little-endian
  - CharRep
    - 0 = ASCII
    - 1 = EBCDIC
  - FloatRep
    - 0 = IEEE 754
    - 1 = VAX
    - 2 = Cray
    - 3 = IBM



0 4 8 16 24 31

IntegrRep | CharRep | FloatRep | Extension 1 | Extension 2