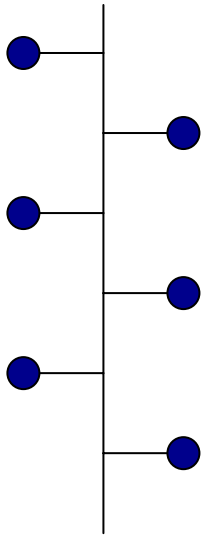
Unit 24
Application Support



Acknowledgements – slides coming from:

- Based to partially on slides from the earlier issues of the EECS 122 notably the slides form Prof. Stoica/Prof. Abhay Parekh for DNS.
- Some slides (and animations!) borrowed from Peter Danzig (Stanford).

For today:

- Reminder: ISO OSI layer vs. Internet architecture
  or: why do we need application support?
- More about application oriented QoS...
- A data representation: how do I know what the bits mean?
- Client- server architecture: work has to be done - do I care who does it?
- DNS: naming in the internet.

Summary: what did you learn about E-t-E data movement?

- How to reach a specifically designated host
- How to "save the network" – congestion control
  - Be aware: inside TCP
  - Has to be implemented separately in non- TCP applications!

  ➔ Does not have to be "EXACTLY" the same, but follow the spirit…
- This is the NETWORK oriented point of view.
- Now: why do we built the network? For usage…
- Internet architecture ➔➔ applications on top of Transport
- ISO OSI Architecture: recognize explicitly that there is a lot of functionalities needed by most of the applications… (session layer, presentation layer).
- Call it a layer, include it in application: you need some of it!

  ➔ This will be discussed today…

Different Requirements

How stringent the quality-of-service requirements are.

Application	Reliability	Delay	Jitter	Bandwidth
E-mail	High	Low	Low	Low
File transfer	High	Low	Low	Medium
Web access	High	Medium	Low	Medium
Remote login	High	Medium	Medium	Low
Audio on demand	Low	Low	High	Medium
Video on demand	Low	Low	High	High
Telephony	Low	High	High	Low
Videoconferencing	Low	High	High	High

We have discussed earlier what we can do in the networks.
Let us see what we can do within the applications.

QoS – again…

- Remember:
  - Throughput (amount of data per time unit)
  - Delay (mean value, jitter)
  - Error rates: Loss, duplication, corruption
- Different applications have different expectations as for QoS.

  Good News: we can trade one for the other!
  - Trading error rate for delay: all the ARQ approaches
  - Trading error rate for throughput: FEC
  - Trading jitter for delay: play out buffer (see more in the next slides)
- What can we do about delay? About lack of throuphput
  - Nothing DIRECTLY.
  - We can influence some of them… indirectly… will point this out while discussing some issues in application support/application structuring

Interactive Multimedia: VOIP (simplified)

Introduce Internet Phone by way of an example

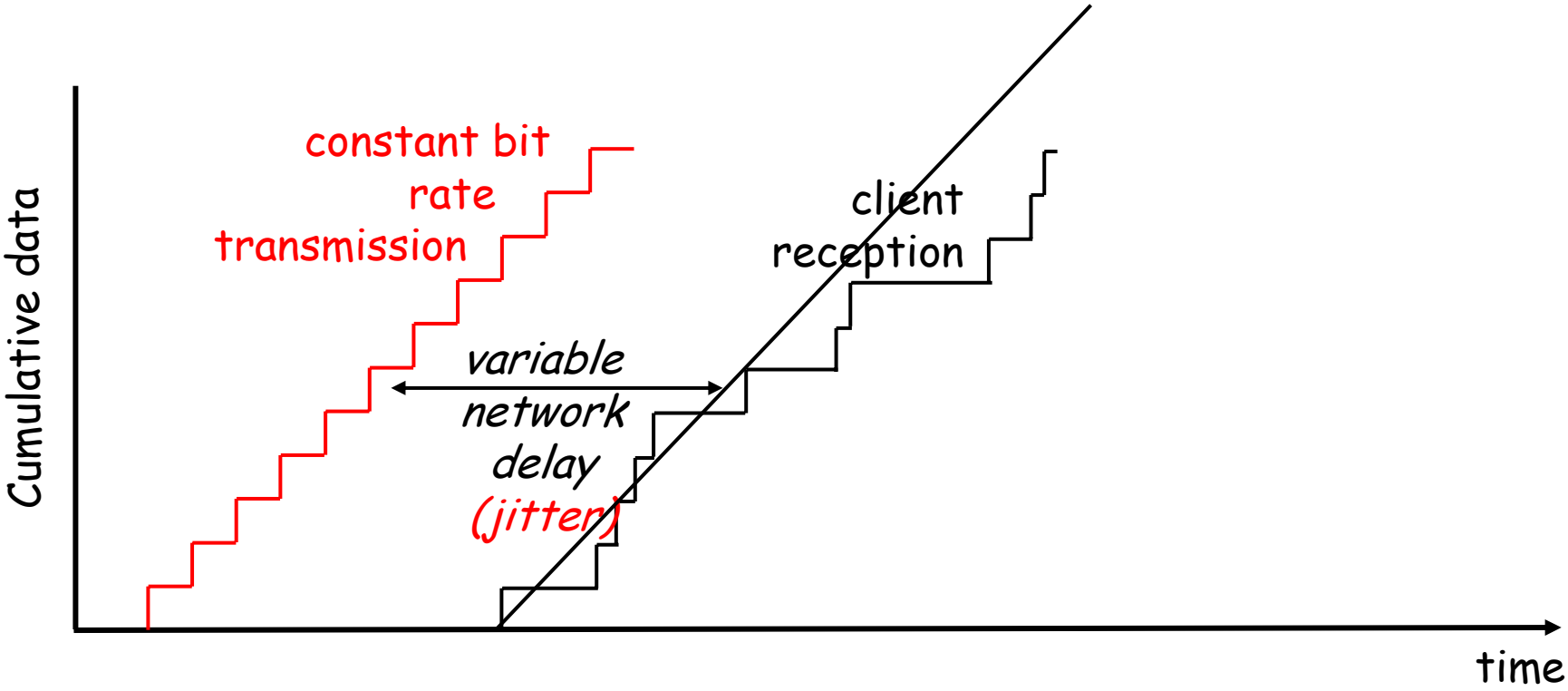
- speaker's audio: alternating talk spurts, silent periods.
  - 64 kbps during talk spurt
- pkts generated only during talk spurts
  - 20 msec chunks at 8 Kbytes/sec: 160 bytes data
- application-layer header added to each chunk.
- Chunk+header encapsulated into RTP inside of a UDP segment.
- application sends UDP segment into socket every 20 msec during talkspurt.

Internet Phone: Packet Loss and Delay

- network loss: IP datagram lost due to network congestion (router buffer overflow)
- delay loss: IP datagram arrives too late for playout at receiver
  - delays: processing, queueing in network; end-system (sender, receiver) delays
  - typical maximum tolerable delay: 400 ms
- loss tolerance: depending on voice encoding, losses concealed, packet loss rates between 1% and 10% can be tolerated.

Delay Jitter

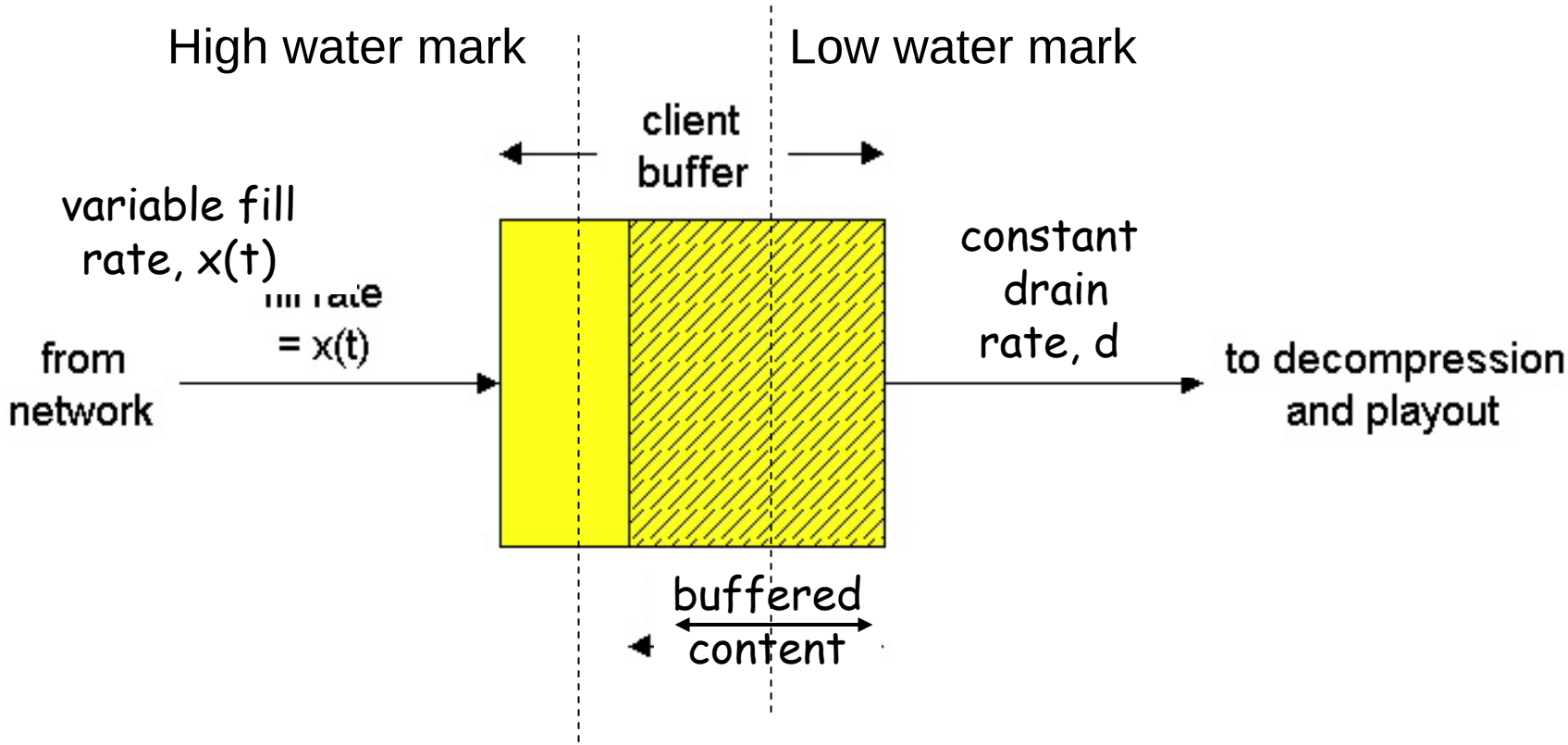
What happens if we start playing out upon arrival of the first bit ?



- Consider the end-to-end delays of two consecutive packets: difference can be more or less than 20 msec

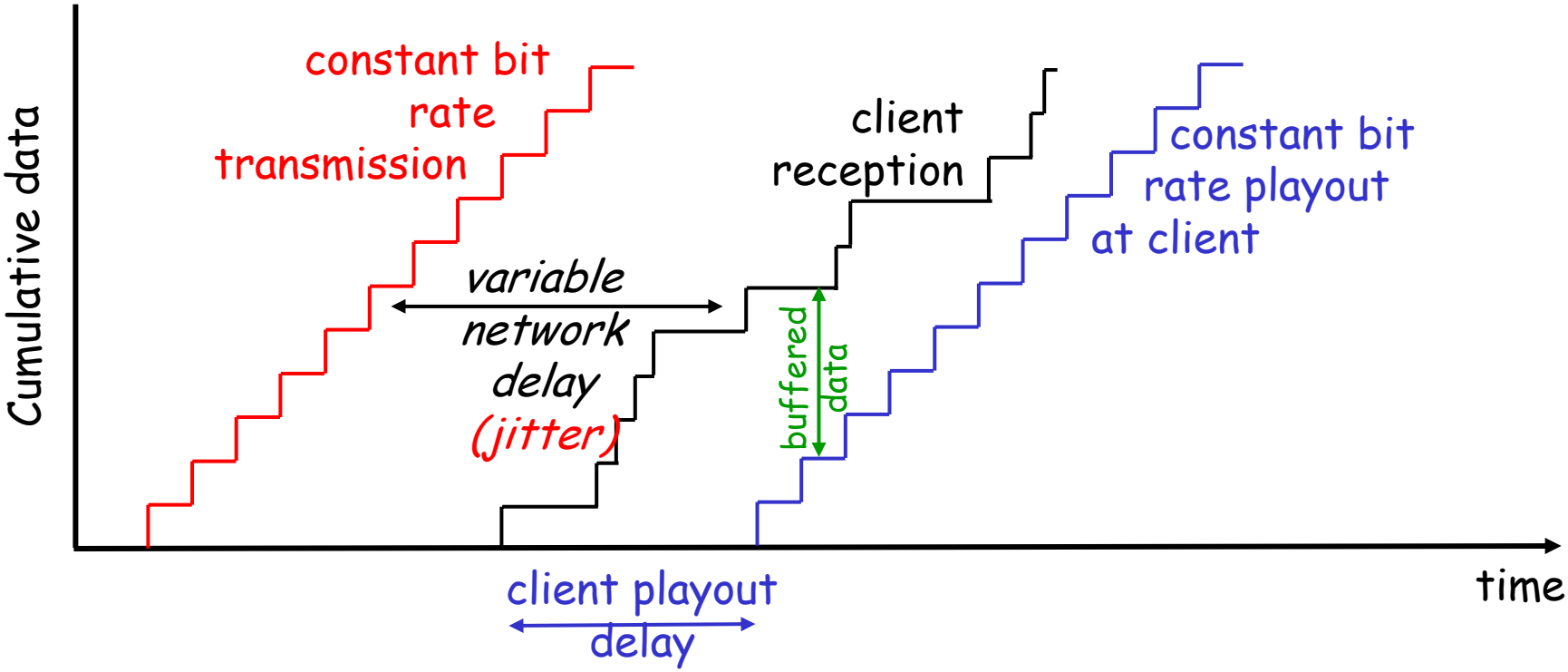
Remedy: Client Buffering

Store some data before starting playing out....



- Client-side buffering, playout delay compensate for network-added delay, delay jitter

Delay Jitter



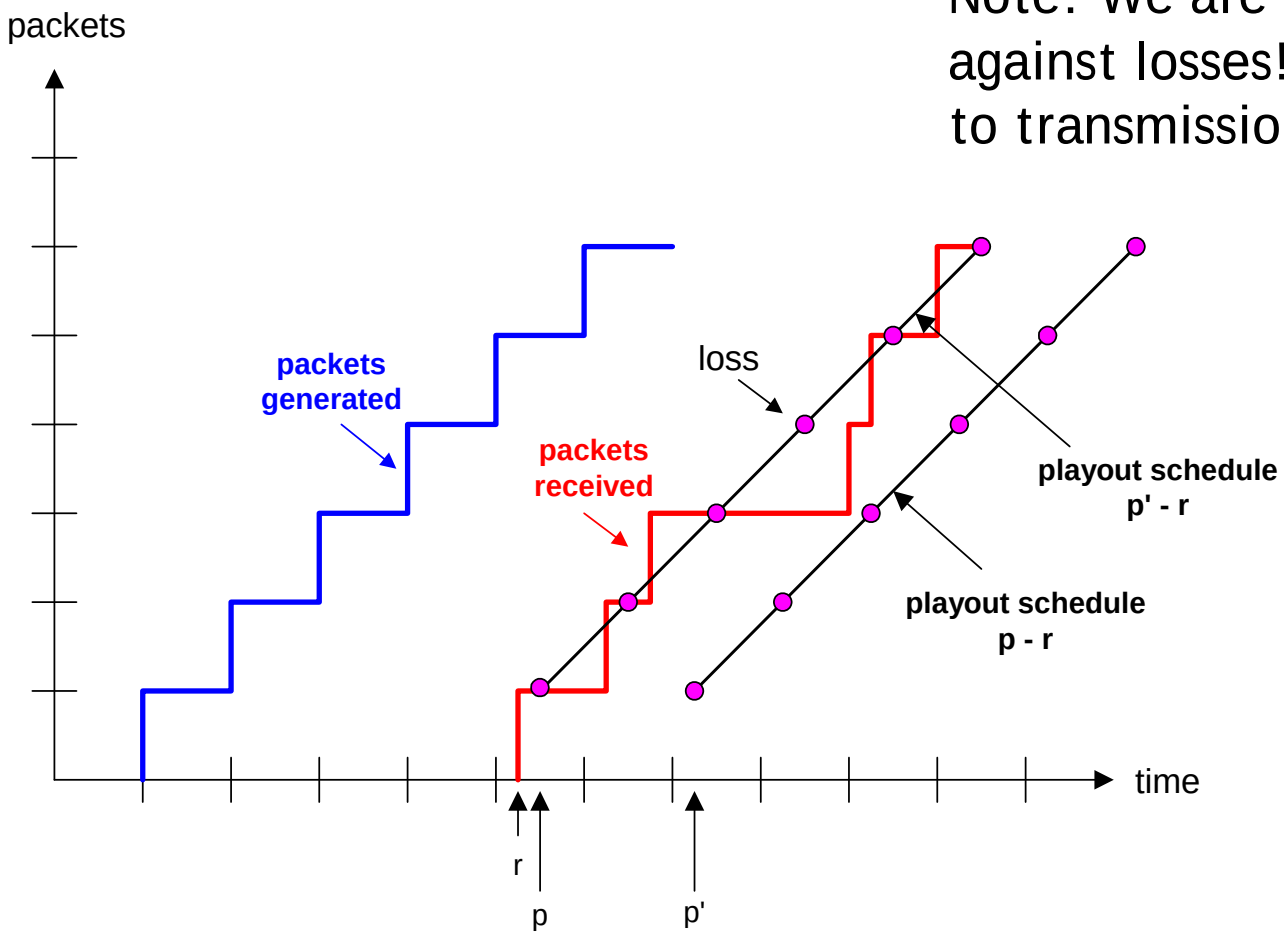
Internet Phone: Fixed Playout Delay

- Receiver attempts to playout each chunk exactly q msecs after chunk was generated.
  - chunk has time stamp t: play out chunk at t+q .
  - chunk arrives after t+q: data arrives too late for playout, data “lost”
- Tradeoff for q:
  - large q: less packet loss
  - small q: better interactive experience

Fixed Playout Delay

- Sender generates packets every 20 msec during talk spurt.
- First packet received at time r
- First playout schedule: begins at p
- Second playout schedule: begins at p'

Note: We are trading delay against losses!!! (in addition to transmission losses….)



Adaptive Playout Delay, I

- Goal: minimize playout delay, keeping late loss rate low
- Approach: adaptive playout delay adjustment:
  - Estimate network delay, adjust playout delay at beginning of each talk spurt.
  - Silent periods compressed and elongated.
  - Chunks still played out every 20 msec during talk spurt.

t_i = timestamp of the ith packet

r_i = the time packet i is received by receiver

p_i = the time packet i is played at receiver

$r_i - t_i$ = network delay for ith packet

d_i = estimate of average network delay after receiving ith packet

Dynamic estimate of average delay at receiver:

$$d_i = (1-u)d_{i-1} + u(r_i - t_i)$$

where u is a fixed constant (e.g., u = .01).

Adaptive playout delay II

Also useful to estimate the average deviation of the delay, v_i:

$$v_i = (1-u)v_{i-1} + u\,|\,r_i - t_i - d_i\,|$$

The estimates d_i and v_i are calculated for every received packet, although they are only used at the beginning of a talk spurt.

For first packet in talk spurt, playout time is:

$$p_i = t_i + d_i + Kv_i$$

where K is a positive constant.

Remaining packets in talkspurt are played out periodically

Adaptive Playout, III

Q: How does receiver determine whether packet is first in a talkspurt?

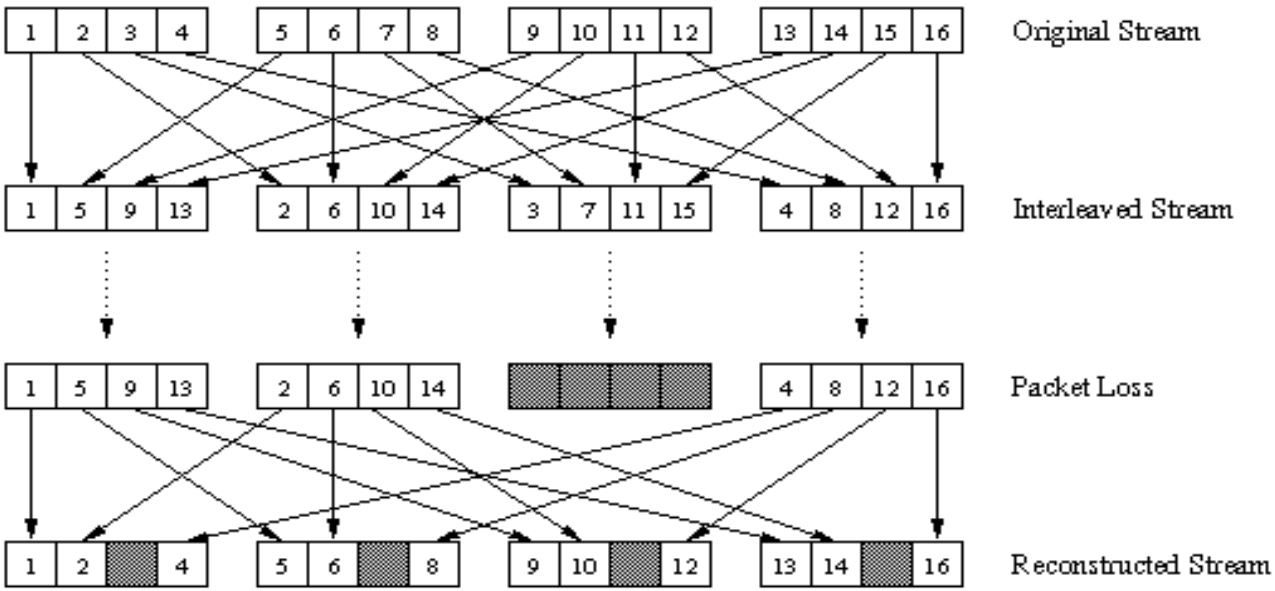
- If no loss, receiver looks at successive timestamps.
  - difference of successive stamps > 20 msec -->talk spurt begins.
- With loss possible, receiver must look at both time stamps and sequence numbers.
  - difference of successive stamps > 20 msec and sequence numbers without gaps --> talk spurt begins.

Recovery from packet loss – the simplest case..

forward error correction (FEC): simple scheme

- for every group of n chunks create a redundant chunk by exclusive OR-ing the n original chunks
- send out n+1 chunks, increasing the bandwidth by factor 1/n.
- can reconstruct the original n chunks if there is at most one lost chunk from the n+1 chunks
- Playout delay needs to be fixed to the time to receive all n+1 packets
- Tradeoff:
  - increase n, less bandwidth waste
  - increase n, longer playout delay
  - increase n, higher probability that 2 or more chunks will be lost

Recovery from packet loss - interleaving



Interleaving

- chunks are broken up into smaller units
- for example, 4-5 msec units per chunk
- Packet contains small units from different chunks
- if packet is lost, still have most of every chunk
- has no redundancy overhead
- but adds to playout delay

But packet loss is not equal to packet loss

- Time aspect

  - G.711 → PCM 64kbps source

  - G.723.1 → 5.3 and 6.4kbps speech codecs

  - audio flow (in time) can be fairly well simulated by a two state Markov chain with geometric distributions (talk-spurts)

  - states TALK and SILENCE with means λ_{TALK} =1.35 ms and $\lambda_{SILENCE}$ =1.15 ms

- Importance of packets...

  - not identical!

  - position of the lost packet influences essentially the quality

  -„Voiced“ frames: important, decoder needs more time to synchronize after loss....

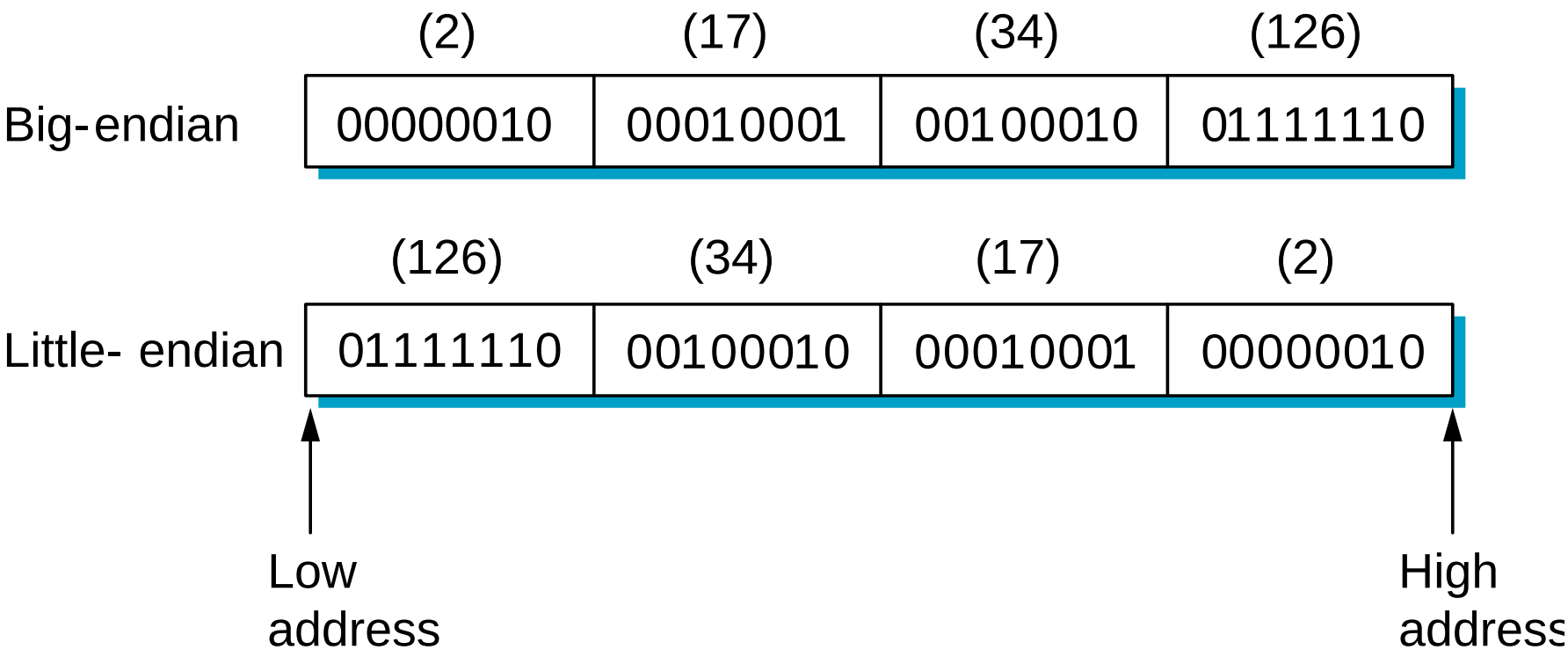
  -Loss of a „voiced“ frame at the border „unvoiced“-“voiced“ even worse…

Streams of Bits/bytes can be transmitted: so what?

How do we know what is the INFORMATION inside?

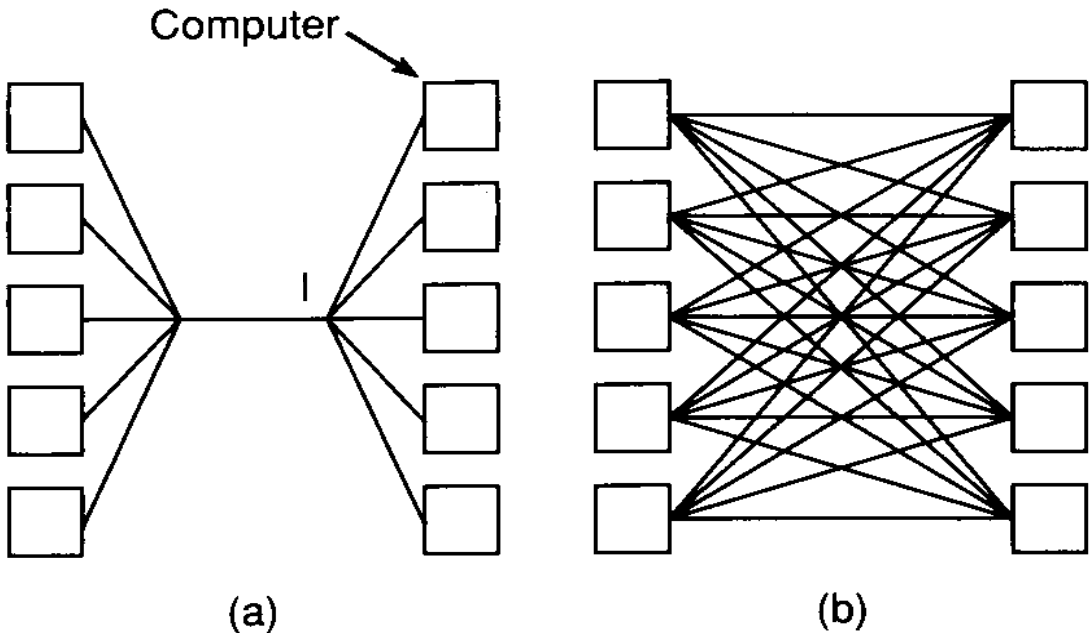
Simple example

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



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)

Data Conversion

- Two different types of rules are needed:
  - Abstract syntax: a station must define what datatypes are to be transmitted
  - Transfer syntax: it must be defined how these datatypes are transmitted, i.e. which representation has to be used.

Tagged versus untagged data

type = INT	len = 4		value =	417892	

Abstract Syntax Notation.1 - ASN.1

- Each transmitted data value belongs to an associated data type.
- For the lower layers of the OSI-RM, only a fixed set of data types is needed (frame formats), for applications with their complex data types ASN.1 provides rules for the definition and usage of data types.
- ASN.1 distinguishes between a data type (as the set of all possible values of this type) and values of this type (e.g. '1' is a value of data type Integer).
- Basic ideas of ASN.1:
  - Every data type has a globally unique name (type identifier)
  - Every data type is stored in a library with its name and a description of its structure (written in ASN.1)
  - A value is transmitted with its type identifier and some additional information (e.g. length of a string).

Definition of Datatypes using ASN.1 (1)

- A data type definition is called „abstract syntax“; it uses a Pascal-like syntax.
- Lexical rules:
  - Lowercase letters and uppercase letters are different
  - A type identifier must start with an uppercase letter
  - Keywords are written in uppercase letters
- ASN.1 offers some predefined simple types:
  - BOOLEAN (Values: True, False)
  - INTEGER (natural numbers without upper bound)
  - ENUMERATED (association between identifier and Integer value)
  - REAL (floating point values without upper or lower bound)
  - BIT STRING (unbounded sequence of bits)
  - OCTET STRING (unbounded sequence of bytes/ octets)
  - NULL (special value denoting absence of a value)
  - OBJECT IDENTIFIER (denoting type names or other ASN.1-objects)

Definition of Datatypes using ASN.1 (2)

- Examples:
  - MonthsPerYear ::= INTEGER
    MonthsPerYear ::= INTEGER (1..12)
    Answer ::= ENUMERATED (correct(0), wrong(1),noAnswer(3))
- With the following type constructors new types can be built from existing ones:
  - SET: the order of transmission of the elements of a set is not specified. The number of elements is unbounded, their types can differ
  - SET OF: like SET, but all elements are of the same type.
  - SEQUENCE: the elements of a sequence are transmitted in the defined order. They can be of different types. The number of elements is unbounded.
  - SEQUENCE OF: like SEQUENCE, but all elements are of the same type
  - CHOICE: the type of a given value is chosen from a list of types (like a Pascal variant record)
  - ANY: unspecified type

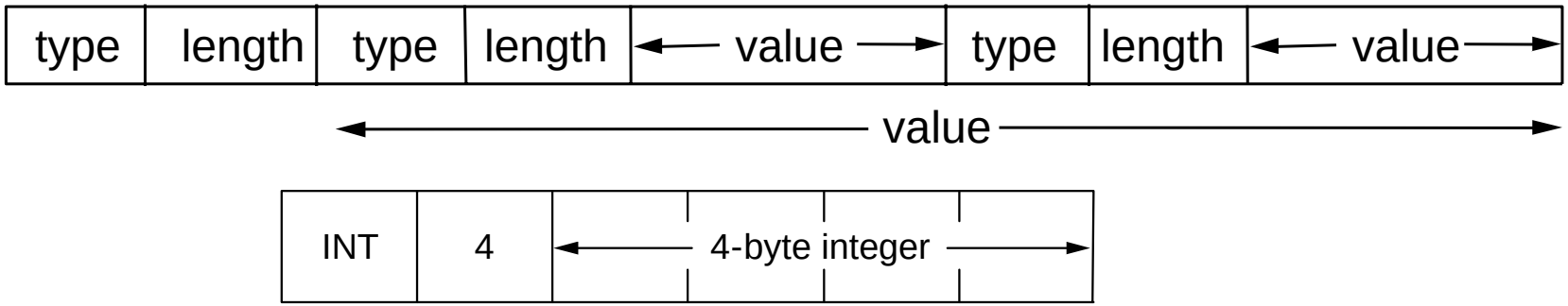
ASN.1 Transfer Syntax (1)

- Some coding rules (the „transfer syntax“) specifies how a value of a given type is transmitted. A value to be transmitted is coded in four parts:
  - identification (type field or tags)
  - length of data field in bytes
  - data field
  - termination flag, if length is unknown.
- The coding of data depends on their type:
  - integer numbers are transmitted in High-Endian Two’s complement representation, using the minimal number of bytes: numbers smaller 128 are encoded in one byte, numbers smaller than 32767 are encoded in two bytes, …
  - Booleans: 0 is false, every value not equal 0 is true.
  - for a sequence type first a type identification of the sequence itself is transmitted, followed by each member of the sequence.
  - Similar rules apply to the transfer of set types

Abstract Syntax Notation One (ASN-1) – Summary.

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

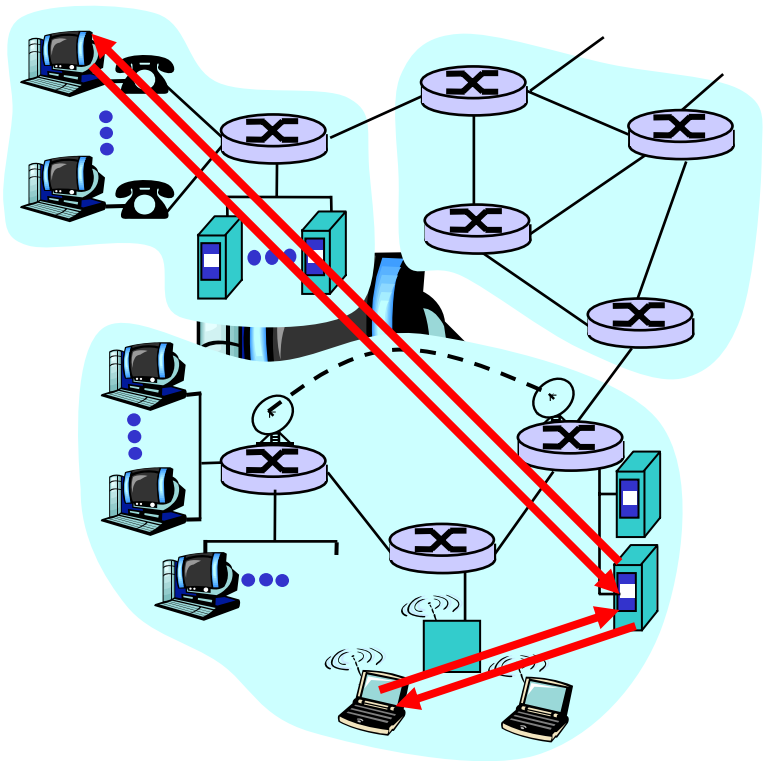
`(tag, length, value)`



The Client- Server approach

- The client / server approach:
  - A client wants to perform a specific action, e.g. to print a file. The client itself is not able or willing to do that (he has no printer), but he knows someone (another computer), who could do it (the printer is connected to that other computer). The other one is the server.
  - The client transmits a request message to the server (including the file to be printed), asking the server to perform the service
  - The server receives this messages and performs (probably) the appropriate action (i.e. prints the file).
  - The results are send back to the client via a reply message.

Client-Server Model



server:

- always-on host
- permanent IP address
- server farms for scaling

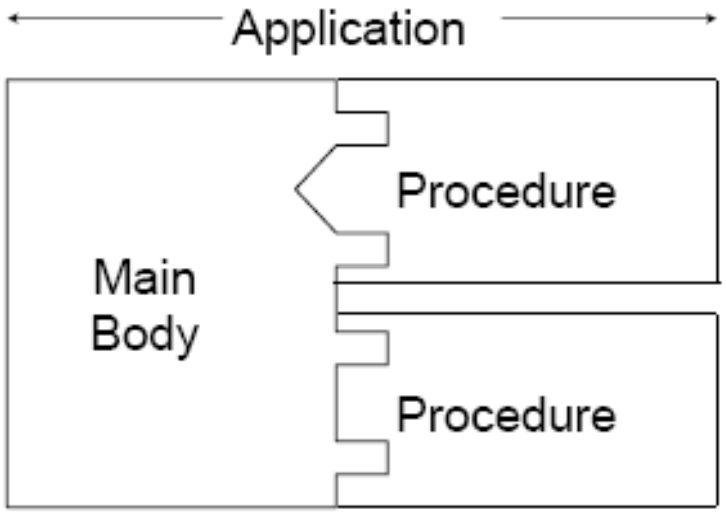
clients:

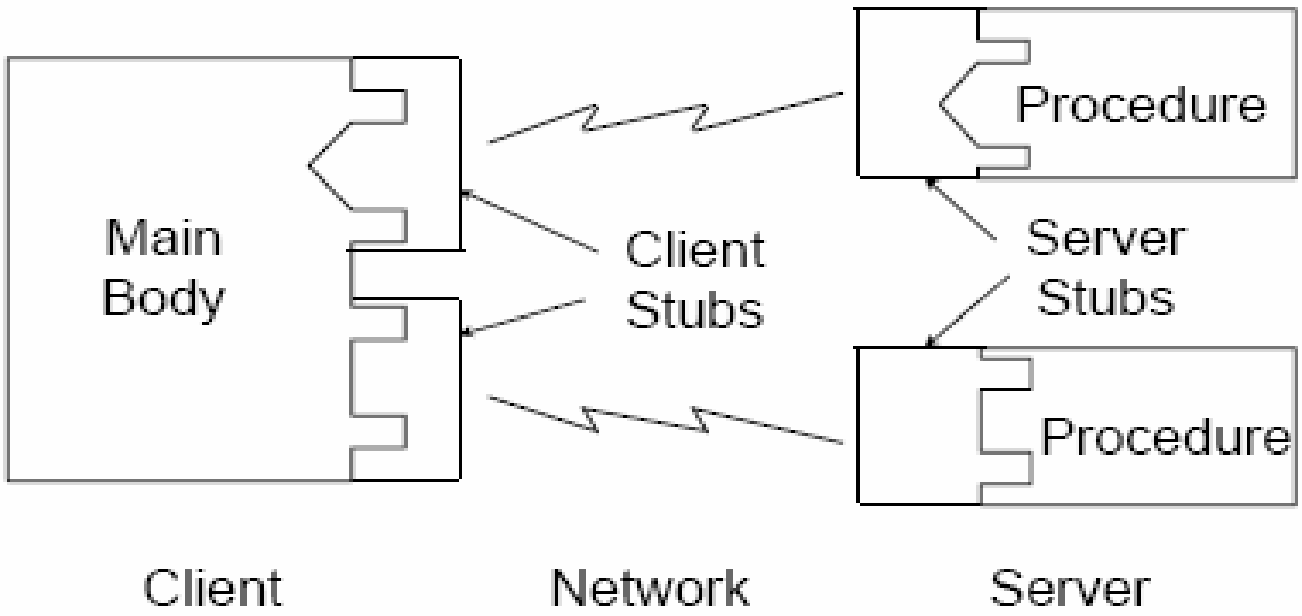
- communicate with server
- may be intermittently connected
- may have dynamic IP addresses
- do not communicate directly with each other

Remote Procedure Calls (RPC)

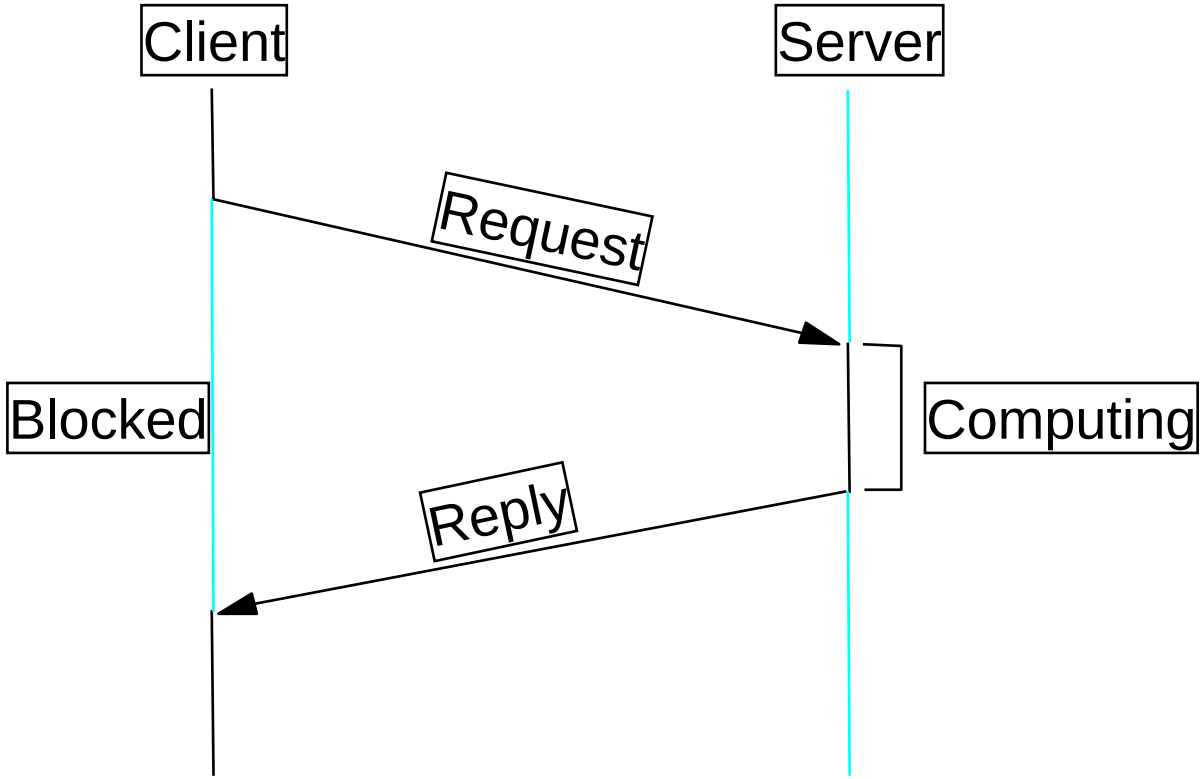
- Remote Procedural Calls are the preferred tool to implement the client-server model.
- In classical procedure calls the code of the procedure is located on the same computer (in the same address space) as the calling program, in an RPC the code is located on another computer.
- One major design goal of an RPC system is transparency: ideally the caller should not know if the callee is located locally or remotely. So in RPC we have to consider the following topics:
  - Parameter handling and marshalling
  - Semantics
  - Addressing
- An RPC system is attractive for the users because automatic support for the conversion from local to remote procedural call can be supported (see below).

Local vs. Remote Procedural Call





RPC Timeline



Basic RPC Operation (1)

- The caller uses a specified procedure interface (like an ANSI-C function prototype), where all necessary procedure parameters are declared with their datatypes.
- In case of a local procedure, the callee performs the requested operation, in the RPC case the callee is a stub procedure that takes the given parameters, forms a message out of them (marshalling) and sends it to the appropriate server.
- In the server another stub procedure (also often called a skeleton procedure) receives the message, extracts the parameter values (unmarshalling) and calls the (local) procedure.
- The server stub takes the results of this call and sends them back to the client.
- The client stub returns the result extracted from the received message to its caller.

RPC Parameter Passing

- Procedures in common programming languages have different types of parameters and calling conventions, which have to be treated in a RPC:
  - Simple call-by-value parameters are passed “as is“ (e.g. simple integer values)
  - Call-by-reference parameters are pointers; since different address spaces are used by sender and receiver, the denoted value (e.g. a buffer) has to be completely transmitted (so its length and its type must be known in advance). If the server changes some buffer values, the buffer must be retransmitted.
  - Complex data types using pointers (e.g. graphs, trees or lists) cannot (or only with difficulty) be transmitted.
- Beyond these explicit parameters, implicit parameters must be passed: because a server can provide many procedures, the name of the requested procedure must also be transmitted.
- Since different types of computers can be used, the stub procedures must use a common encoding convention for different parameter types. The parameter values can be transmitted without type information (unlike the ASN.1-principles), because their types are predefined in the specification of the procedure.

Finding an RPC server (Addressing)

- A client can use fixed, hard coded addresses for finding the appropriate server station. This approach is simple but not flexible.

- A dynamic binding approach can be used:
  - A server stub transmits at its initialization a message containing its name (procedure name), its version number, its address and a unique (within the server station) identification to a special station, the „bindery“ station, which maintains a database of all available services.
  - A client stub, if operating the first time, queries the bindery station for an appropriate server providing the requested service (i.e. service name, version number). If no server exists, the client stub fails. Otherwise, the bindery returns the address and the unique identification to the client stub.

RPC Semantics (1)

- In normal operation the RPC should behave exactly as the corresponding local procedure call (LPC). In the local case it is assumed that a procedure call returns correctly (unless the system fails). This assumption is not valid for RPC systems. Several problems can arise:
  - Addressing (the client could not find an appropriate service)
  - The client or the server can fail
  - Message loss
- If the client could not find an appropriate server, a kind of exception handling is needed, thus violating the transparency requirement.
- The server can fail before executing the requested action or while executing it or immediately before returning an answer; another possibility is the loss of either an answer of a request message. The client is not able to distinguish these cases.

RPC Semantics (2)

- The client stub has three possibilities for further behavior if the result is missing:
  - He can retransmit his request until he receives a correct message (the server can be restarted or another server was found). In case that the server crashed after execution of the command, it could be executed twice. This is called „at least once semantics“.
  - He can stop after transmitting one message and report an error to the client. This is called „at most once semantics“
  - He can do anything else (e.g. make exactly 37 attempts), thus failing to give any guarantees to the client.

- An action is called idempotent if multiple executions do not change the result or the state of the system (e.g. reading from a file does not change its state - a second read operation yields the same result - but this is not true for writing to a file). There are also semi-idempotent actions.

- If all RPC actions are idempotent, the RPC semantics does not matter, since every request can be repeated without harm.

- If there are non-idempotent actions, „exactly once“ semantics is required

RPC Semantics: Orphans

- If the client fails after having issued an RPC call, the server performs an action associated to nobody, a so called „orphan“.
- The client can recover quickly and request the same service again. There is a possibility that it receives the answer for the orphan instead of its own.
- There are different approaches to handle this problem:
  - „Extermination“: the client can maintain a logfile, logging all RPCs. With the help of this file orphans can be detected and deleted. This approach has high performance costs and cannot handle orphans generated by his RPC („grand orphans“)
  - „Reincarnation“: Time is divided into „epochs“. After every startup a client starts a new epoch and sends an appropriate message to all servers. All actions belonging to an old epoch are canceled.
  - „Gentle Reincarnation“: if a server receives a message announcing a new epoch, he tries to find the client and ask him what to do. If he can‘t find him, all actions belonging to an old epoch are canceled. Note: canceling operations can bring problems, e.g. in case of file locks.
  - „Expiration“: a RPC has to be performed within time T. If the clients waits time T after restart, no orphans can occur.

RPC-Example: Sun RPC

- The Sun RPC system is used within Sun NFS. Sun RPC uses either TCP or UDP. It provides „at least once semantics“
- Sun provides an „Interface Definition Language“ XDR, used for declaring a server interface, i.e. all used datatypes (thus datatypes can be defined within XDR) and all exported procedures, including their parameter lists. Also a version number is provided with an interface definition.
- Procedures are grouped into programs. A version number is associated with a program.
- A procedure is assigned a unique (within the surrounding program) number. Each procedure accepts only one parameter, so structure types must be used for passing more than one parameter. Each procedure returns only one value.
- An interface compiler rpcgen generates client and server stub procedures, marshalling procedures for both stubs, a header file for use by application programs.
- No dedicated bindery station is used; instead, a so called port mapper assigns a TCP/UDP-port (socket) to every registered service. A client must know in advance which station is the appropriate server. The portmapper returns the port number to the client in case of a request. If TCP is used, some parameters (e.g. window size) can be set in the binding process.