

Preliminary specification of (and reasoning behind) Zing: A transport protocol for file transfers over circuits

Tim Moors
2000 Sep. 12

1 Introduction

Zing is a transport protocol designed for transmitting files on a unidirectional circuit from source to destination. The protocol is designed to exploit specific characteristics of circuits (e.g. known rate) and those of files (e.g. parts can be delivered in arbitrary order). Circuit-switched networks need to be accompanied by packet switched circuits, at least to carry signalling to establish and release circuits. We propose using such a packet-switched network for transport protocol signalling and control information, and to carry retransmissions from source to destination so that the circuit holding time can be known before use so as to facilitate scheduling of circuits [1]. We propose carrying this information over TCP over the packet-switched network in order to avoid reinventing a reliable transport protocol for packet-switched networks, and so as to simplify the core of the Zing protocol.

Figure 1 summarizes the concepts underlying Zing, leading from assumptions to features.

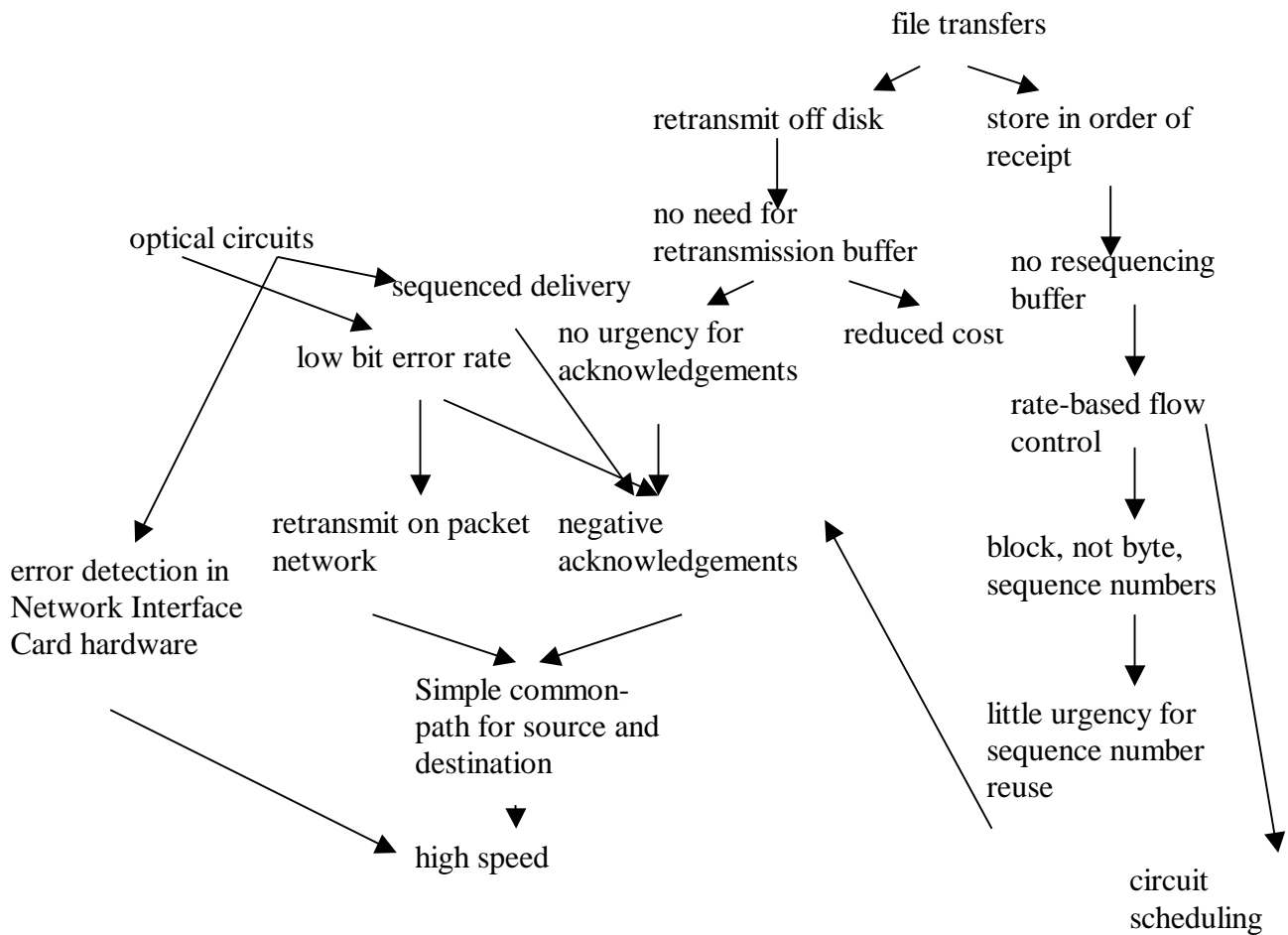


Figure 1: Summary of Zing concepts.

also: “file length -> known length -> circuit scheduling”, although that is more related to the network layer than the transport.

This document will first describe the assumptions underlying Zing, followed by a summary of Zing's Protocol Data Units (PDUs). It will then describe the services that Zing offers, and the mechanisms by which Zing provides these services.

2 Assumptions

2.1 Transport for “proper files”

Zing is designed for the transport of large files, such as backups, web mirrors, etc. The sources of such files are characterized by being able to determine the file length prior to transmission, and by being able to repetitively retrieve the file from where it is stored. Zing will exploit the latter attribute, and the network layer will use the known file length to facilitate scheduling. The destinations of such files are characterized by being insensitive, in terms of *operational correctness*, to the order in which segments of the file are received. Zing will also exploit this attribute. Note, however, that *system performance* may depend on the order in which segments are received. For example, if the storage device does not offer true random access (e.g. tape or a disk with significant seek overheads) then segments will store faster in-order than out-of-order.

We assume that files do not expand as they are transferred between source and destination. For example, some filesystems use different conventions for marking the end of a line of text (e.g. linefeed on some, carriage-return and linefeed on others), and transferring a file from one type of filesystem to another might require conversion which would interfere with ability for a destination to receive segments out of order (since the position for later segments depends on the amount of expansion in earlier segments).

Note that many operating systems (e.g. Unix) provide a uniform I/O interface, under which “proper files” as described above are accessed in the same manner as other sources of information, such as devices or pipes from other programs. These sources and destinations of information often cannot provide information in the same manner as proper files (e.g. information can only be retrieved once from a source) and so cannot be used with Zing.

2.2 Known terminal processing rates

We assume that the transport layer knows the rate at which an application can process a file before the transfer is requested. For example, a tool could measure the application performance when the transport layer is installed.

2.3 Isochronous circuits

We assume that the circuit that extends between source and destination is isochronous. For example, there are no rearrangeable switches on the path which could cause pauses or bursts in the transfer.

2.4 Transport only

The protocol only provides for the transfer of bits of information. It does not provide a complete file transfer service, e.g. the ability to list directories, provide access control, convert between requirements for different file systems (e.g. case-sensitive names, filename lengths), whether to copy symbolic links as links or as the files that they refer to, specifying the names of the files to be transferred, etc.

The following assumptions are limitations that simplify the initial protocol:

2.5 One transfer at a time

Only one circuit between any two addresses at any instant. Zing does not support multiplexing of multiple applications in a node that are concurrently engaged in file transfers.

3 Protocol data units

The Zing protocol uses four classes of protocol data units: blocks (that carry payload information), acknowledgements, signalling messages, and padding. All PDUs are a multiple of 32 bits in length so as to facilitate word-based implementations of integrity checks. In general, each PDU starts with a 32-bit word that describes its type. The only exception is for certain circuits that only carry payload blocks (e.g. PPP over SONET which provides rate adaptation and so padding PDUs are unnecessary), in which case the type preface is unnecessary and is eliminated. Table 1 summarizes the PDUs. Only the first three PDUs (payload, trailing payload padding information, and padding) are transmitted on the circuit; the other PDUs are transmitted over the packet-switched network. (Payload may also be transmitted over the packet-switched network under certain circumstances described in Section 4.11.)

Category	PDU type	Fields of PDU after type (optional fields in parenthesis)	Described in Section
Transmissions	Payload	Sequence number 0-B words of payload (ICV)	most, particularly 7
	Trailing payload padding indication	length of padding (ICV)	7
	Padding	length 0-2 ³² -1 words of 0s (ICV)	6.2
	Retransmission	length [remainder as for Payload]	4
Control	negative_acknowledgement	Sequence number	4.5
	positive_acknowledgement	(Sequence number of most-recently received block)	4.5
	file_received		4.6
	abort_transfer		4.5, 8.1.1
Signalling	request_from_source	file length	8.1.1
	request_from_destination	identifier	8.1.1
	response	followed by one 32-bit word, an integer multiple of three 32-bit words, followed by a 32-bit word with all bits set to 0	8.1.2
	inadequate_candidates	none	8.1.3
	commit	followed by four 32-bit words	8.1.3

Table 1 : Summary of Zing PDUs.

The following sections introduce these protocol data units in the contexts of the services to which they contribute. The services covered are, in order: reliable transfer (Section 4), flow control (Section 5), network adaptation (Section 6), and framing (Section 7). This is followed in Section 8 by a discussion of the signalling mechanism which supports these services.

4 Reliable transfer

This section will introduce the concept of reliable transfer, then elect the mechanism of retransmission-based error control to implement the reliable transfer, and then describe the steps that are required in order to implement that mechanism in the order in which they would usually be invoked (i.e. protecting information, detecting errors, and then retransmitting information).

A key role for a transport layer is to ensure that the transfer of information from source to destination is “reliable”. There are six aspects to reliability:

- integrity: ensuring that what arrives has the same value as what was transmitted,
- completeness: ensuring that everything that was transmitted arrives,
- delivery: informing the source that all information has been delivered,
- relevance: ensuring that only what was transmitted by the source arrives (no extraneous information should be inserted),
- sequence: ensuring that information arrives in sequence, and
- uniqueness: ensuring that information is not duplicated.

Circuit-switched networks are more “reliable” than packet-switched networks in that by design they will not resequence or duplicate information, and they are far less likely to deliver extraneous information to the destination (unlike in packet-switched networks in which a packet from one session may incur a large delay so that it is received as part of a later session). Consequently, the transport protocol can focus on recovering from errors relating to integrity, completeness and delivery. (The transport protocol should not, itself, introduce other errors, e.g. resequencing.)

Optical circuits tend to exhibit low bit error rates, e.g. as low as 10^{-9} . Zing will also exploit this characteristic.

File transfers are characterized by being tolerant of segments being delivered out of order (provided their position within the file is known).

The error control scheme is designed to exploit these characteristics by emphasizing negative acknowledgements and retransmission on a parallel packet-switched network. The transport protocol delegates responsibility for the reliable transfer of negative acknowledgements and retransmissions to TCP, which provides a connection over which this information is sent.

Zing will use retransmission-based error control. This choice is based on the quantitative analysis of Section 4.1 which shows that retransmission-based error control is adequate (has negligible transmission overhead for the conditions of interest) and from the following quote indicating qualitatively that retransmission-based error control is preferable (when information must be transferred with high integrity over low error-rate links):

“FEC error-control systems ...have some drawbacks. When a received word is detected in error, it must be decoded, and the decoded word must be delivered to the user regardless of whether it is correct or incorrect. Since the probability of a decoding error is much greater than the probability of an undetected error, it is harder to achieve high system reliability with FEC schemes. In order to attain high reliability, a long, powerful error-correcting code must be used and a large collection of error patterns must be corrected. This makes decoding hard to implement and expensive. For these reasons, ARQ schemes are often preferred over FEC schemes for error control in data communications systems” [2, p. 6]

4.1 Size of retransmission blocks

With retransmission-based error control, an Integrity Check Value (ICV) must be added to each transmission unit in order to allow the destination to verify its integrity. These ICVs and the volume of information that is retransmitted constitute the transmission overheads in retransmission-based error control. To analyse transmission efficiency, we will use the following symbols:

- B length of transmission unit (block), containing payload and overhead, measured in bits. We assume that transmission units can be treated as all having the same size, e.g. the number of transmission units is so large that any abbreviation of the final transmission unit can be neglected.
- H overHead per transmission unit (e.g. for ICVs), measured in bits. For purposes of analysis, we assume that the overhead is essentially independent of the block length. This is often true in practice where the fields are constrained to word lengths, but the ICV and sequence number fields could be influenced by the block size.
- E probability that a bit will be received in error. Strictly, we are only interested in the probability that a *block* contains one or more errors, however, this tends to depend on the length of the block. A reasonable approximation for the probability that a block contains an error is EB . This comes from assuming that bit errors occur randomly (not in bursts), in which case the probability of a block having an error is $1-(1-E)^B$, which is approximately EB when $E \ll 1$, as is expected for the clean optical environment.

The average overhead for transmitting a block equals the overhead when transmitting that block without errors, and the overhead when transmitting it with errors. Again, assuming that the probability of an error is small, we can approximate this by considering only single retransmissions (e.g. the probability that a retransmission gets errored and needs to be retransmitted itself is negligible). In this case, the average block transmission overhead is:

$$(1-EB)H + EB(H+B)$$

To ensure that this overhead is negligible, say less than 1%, we must choose a block size so that:

$$\frac{H + EB^2}{B} \leq 1/100$$

This leads to a quadratic equation whose solutions are:

$$B = \frac{1/100 \pm \sqrt{1/10000 - 4EH}}{2E}$$

If we take $E=10^{-9}$ (as is typical with fiber) and $H=64$ (e.g. a 32 bit CRC plus 32 bit sequence numbers), the overhead is a minimum for a transmission unit of 30KB, and is negligible over the range [800B, 1.2MB]. Since this range covers feasible block sizes (e.g. Lucent's Optistar card can handle frames as long as 8KB), we conclude that the transmission overhead from retransmission-based error control is negligible.

The transport protocol should not limit the length of files that it can transmit. If it is to handle files larger than 1.2MB with negligible transmission overhead, then it will need to segment those files into blocks. The blocks that carry the contents of a file will be of fixed size, with the possible exception of the last block which will carry remainders of the file that do not fill a fixed-size block.

4.2 Sequence numbers help detect errors

The receiver will verify the integrity of incoming blocks by matching the ICV contained in the incoming block to the ICV calculated at the receiver over the block's contents. Ideally, any mismatches should be reported to the transport layer immediately after they have been detected. However, in practice, it is common for the NIC hardware to calculate *an* ICV, and to silently discard errored blocks without

directly¹ informing the transport layer. For example, this is true of the Optistar NICs, which calculate a CRC-32. Thus, it is up to the transport layer to detect when a block has been discarded so that it can then initiate retransmissions.

With isochronous circuits, the transport layer would expect to receive blocks periodically. It could detect that the NIC has discarded a block by the NIC not delivering a block in the expected time. However, this may not be feasible in practice. For example, the Optistar OC12 card allows 8KB frames and communicates at 622Mb/s, i.e. blocks could recur every 0.1ms. A transport protocol implemented in software would need to process blocks as they arrive, i.e. be interrupted every 0.1ms, and would have to be able to set a timer to expire between 0.1 and 0.2 ms after receipt of each packet. This accuracy may not be feasible in software.

Another way to detect when the NIC discards blocks is to assign each block a sequence number. The transport layer will then notice a jump in the sequence numbers when the NIC discards a block. A sequence number of b bits would allow the receiver to deal with groups of 2^b blocks at a time, e.g. increasing the period between interrupts to $2^b/10$ ms.

Thus, the transport layer will associate with each block a sequence number. Zing will use a sequence number of 0 for the first block of the file, and will increment the sequence number for subsequent blocks of the file, wrapping if necessary from $2^{32}-1$ to 0. (Subsequent versions of the protocol might include initial sequence number negotiation so as to foster security and help filter out blocks that have been falsely inserted into the stream.)

4.3 For circuits, link ICV is adequate (without transport ICV)

The fact that the NIC calculates an ICV doesn't mean that the transport layer needs to use that ICV; the transport layer could use its own ICV. In packet-switched networks, it is common for the transport layer to add its own ICV, since the NIC ICV only protects the information as it propagates across links, and it is possible that the corruption may occur in a router where the information is not protected by the NIC ICV, so the transport layer provides its own ICV to provide end-to-end error detection capability. However, with circuit-switched networks, the NIC is accessing an end-to-end path (circuit). Thus, the only errors that cannot be detected by the NIC's ICV are those that occur in the terminals themselves. Such errors can probably be neglected, just as a transport layer check does not protect against errors introduced by the transport or higher layers. Furthermore, each ICV has a performance cost since the process calculating the ICV must "touch" every bit being communicated. For circuit-switched networks, the NIC's ICV is adequate and adding a transport layer ICV may add an unnecessary performance cost, so the transport layer will rely on the NIC's ICV.

(The disadvantages of using a NIC ICV are that the transport layer cannot create pseudo-headers that contain information that is incorporated into the ICV calculations, but not transmitted. For example, the more significant bits of the sequence number could be treated this way. Also, using a NIC ICV limits the control that the transport layer has in selecting the ICV to use; e.g. the NIC may only support CRC-32 while the transport desires a stronger check, such as a MD5.)

4.4 Specifying blocks to be retransmitted, and labeling retransmitted blocks

Once a receiver has determined that information has been lost, it needs to direct the source to retransmit that information, and the source needs to label the information that it retransmits so that the receiver can place it in the correct position with respect to correctly-received blocks. Clearly the sequence numbers assigned to transmitted blocks could be used for these purposes. However, the receiver must be able to

¹ The NICs usually maintain counters of the number of discarded blocks for statistical purposes, and the upper layer could *indirectly* determine that a block was lost by monitoring this statistic.

distinguish a retransmitted block from original blocks. In particular, if the sequence number is used for identifying retransmitted blocks, then the retransmission must arrive at the receiver before the source can reuse that sequence number to transmit an original block. Or looking at it from the source's end, the sequence numbers must not wrap during the interval between when a block is transmitted, and when a negative acknowledgement arrives at the source.

Analysis of sequence number field size: Assuming that the worst-case round-trip propagation delay is 800ms (since 400ms is considered to be the maximum one-way delay tolerable for voice, and we assume that the networks in use can support voice service between points of interest²), the transmission rate should not exceed $2^b/800\text{ms}$ blocks/sec. With 4KB blocks, a b -bit sequence number could accommodate a transmission rate of $2^b/800\text{ms} * 4\text{KB}$ bits/sec. That is, with an OC12 transmission rate, the sequence number needs to have at least 14 bits. Rounding the sequence number size up to the word size of 32 bits ensures that the fields in the blocks will have word alignment (facilitating speedy implementation) and will provide sufficient range for the sequence numbers.

Note the multiple applications for which sequence numbers are used:

- Detecting loss
- Specifying what is to be retransmitted
- Identifying what is retransmitted so that it can be positioned properly.

4.5 Negative acknowledgements to request retransmissions

A common form of retransmission-based error control is “ARQ” in which the destination automatically (without prompting by the source) requests any necessary retransmissions. (An alternative to ARQ is polling, whereby the source polls the destination for its state, and then provides the necessary retransmissions.) ARQ can take the form of positive acknowledgements (which indicate explicitly what has been received, and imply what needs retransmitting), or negative acknowledgements (which have the opposite form). We suggest using negative acknowledgements because of the low error rate on optical circuits, which would make polling or positive acknowledgements “waste” network bandwidth on state exchanges that indicate that no retransmissions are necessary.

The transport layer would send a negative acknowledgement in response to receiving a block whose sequence number is not one larger than that of its predecessor (taking into account sequence number wrapping). However, it is still possible that a block is lost and no successor arrives which would stimulate the transmission of a negative acknowledgement. For example, a string of blocks at the end of the file may be lost. To accommodate this case, the destination should employ a timer. Whenever the destination receives a block, it should reset the timer to a duration `TIMER_SHORT`. If the timer expires, then the destination should send a positive acknowledgement to the source, indicating the sequence number of the last block successfully received. If it has not yet received any blocks from the source, then it should not send a sequence number. The destination should also reset the timer to a duration `TIMER_LONG`. If the destination receives a block before the timer expires again, then it should reset the timer to `TIMER_SHORT`, and continue as per normal. If it does not receive a block before `TIMER_LONG`, then it should abort the transfer by sending an `abort_transfer` PDU to the source, and closing the TCP connection. When the source receives a positive acknowledgement for a block that is not the last block of the file, then it should retransmit all blocks that follow the positively acknowledged block in the file. The source would effectively have `TIMER_LONG` after receiving a positive acknowledgement to ensure that another block arrives at the destination before the transfer is aborted.

² The actual delay for a circuit-switched network may be much less, since circuit-switched networks tend to have little internal buffering, so most of the delays are propagation delays.

Dimensioning the timers: `TIMER_SHORT` must be set to a period greater than the expected interarrival time between blocks. Blocks that are still propagating to the destination while the positive acknowledgement propagates to the source will be retransmitted, even if they would have arrived at the destination intact. Thus, `TIMER_SHORT` should be set to a period larger than the round-trip propagation delay between source and destination, in order to minimize the performance degradation of such retransmissions. Lower values of `TIMER_SHORT` may result in the destination sending positive acknowledgement messages when the source already intends to send more information (e.g. if several blocks in the midst of the file were lost due to bit errors), while larger values of `TIMER_SHORT` reduce the responsiveness of the protocol to node failures. `TIMER_LONG` should be set to at least the round-trip propagation delay between source and destination. Again, longer values of `TIMER_LONG` reduce the responsiveness of the protocol to node failures, whereas shorter values may result in the transfer being unnecessarily aborted. The recommended values for the timers are 800ms for `TIMER_SHORT` and 5 minutes for `TIMER_LONG`.

Note that most connection-based protocols include a timeout mechanism, such as that described above, to prevent indefinitely retaining state information for connections that have failed. In other words, the timer mechanism described above does not constitute an additional overhead on the destination.

4.6 Delivery indications require positive acknowledgements

The retransmissions and acknowledgements discussed in previous sections are used by the destination to ensure that it receives the complete file. However, the source may also be interested in determining whether the transfer was successful. It cannot determine this by the absence of negative acknowledgements, since a lack of negative acknowledgements can also be caused by the destination failing. To provide the source with a delivery indication, it is necessary for the destination to send the source a positive acknowledgement at the end of the transfer. It does this by sending a `file_received` PDU immediately after determining that it has received all blocks of the file.

The destination needs to be able to determine when it has received all blocks of a file. One way to achieve this would be for the source to indicate the file length to the destination during circuit setup, and for the destination to count the volume of arriving information, and to presume that the transfer is complete when it has received a volume of information that matches the file length. The disadvantages of this approach are that it requires the file length to be available at the start of the transfer (which is required for the network layer scheduling, but may limit the generality of the transport protocol), and that this length will require elaborate encoding so that it can increase without bound. An alternative approach exploits the fact that the last block of a file will generally³ be shorter than the other blocks, and so the destination knows that it has received the last block of the file by the fact that the block is shorter than the others. This framing mechanism is described in more detail in Section 7. Note that the timeout mechanism described in Section 4.5 will ensure that even if the last block is lost, then the transfer will eventually complete by either the source retransmitting the last block or by the destination aborting the transfer.

[Positive acknowledgements could also be used to allow sequence number wrapping, although the analysis in Section 4.4 indicates that this will not be necessary.]

[Positive acknowledgements could also be sent periodically (with a period to be specified by the sender during the initial signalling phase) to indicate its liveliness and report on the progress of the transfer.]

³ The exception is when the file is an integer multiple of the block size in length, in which case the last block may have the same length as preceding blocks. The source could then transmit a null block to indicate the end of the file.

4.7 Packet switched network carries acknowledgements and retransmissions

Although the goal of the transport protocol is to unidirectionally deliver information across a circuit from source to destination, previous sections have shown the need to have control information (e.g. acknowledgements) propagating in the opposite direction. It is desirable to avoid establishing a circuit for this control information, since its rate is very low, and many circuits are only available in multiples of a high base rate (e.g. 51.84 Mb/s for SONET).

Furthermore, it is desirable that the holding time for the unidirectional circuit be known before the transfer, yet the number of retransmissions that will be required will not be known until after the transmission has completed. Consequently, it is desirable to also send retransmissions on a channel separate from the unidirectional circuit.

Circuit-switched networks cannot exist in isolation, since it is necessary to propagate a signalling message between the source and destination prior to a circuit transfer in order to establish that circuit. Consequently, each circuit-switched network is associated with a packet-switched network for controlling the circuit. We propose using this packet-switched network to send control information for the transport protocol, as well as retransmissions. The reliability of optical circuits should limit the volume of control information and retransmissions so that it does not unduly load the signalling channel.

Most transport protocols need not only deal with unreliable delivery of the initial transmission of payload information from source to destination, but also with unreliable delivery of the control information or retransmissions. Zing can avoid this by using TCP to carry the information that it sends over the signalling network. This is justified because TCP is already designed to provide reliable transfer over packet-switched networks such as the signalling network. Furthermore, the low volume of information that Zing expects to send over this network provides a good match with the relatively slow software that is needed to implement a sophisticated, yet complicated, protocol such as TCP. Zing is optimised for the common case to use simple mechanisms that can be implemented at high speed, and to use slower but more sophisticated software for rare cases (such as errors on optical circuits).

The blocks that are retransmitted over TCP will have the same format as transmissions on the circuit, except they will need to be prefaced by two 32-bit fields. The first will indicate that this PDU is a retransmission, and the second 32 bit field will indicate the number of 32 bit words in the block (since TCP provides no framing).

4.8 Retransmission buffers

In conventional transport layers that use retransmission for error control, the source transport layer retains information until it has been acknowledged. Unfortunately, the size of this retransmission buffer increases with the bandwidth-delay product of the communication channel. For high-speed links, this buffer must be both large and fast, which leads to significant expense, all for retransmissions that should be rare. Instead, for “proper file” transfers, the source can retransmit from “disk”, rather than from retransmission buffers.

4.9 Resequencing buffers

Just as sources tend to have retransmission buffers in conventional transport layers, destinations tend to have resequencing buffers in which the destination stores segments while waiting to correctly receive preceding segments. Again, for the proper file transfers that Zing targets, resequencing buffers are not necessary, since file segments can be written to a “proper file” in arbitrary order. This has important consequences for flow control, as discussed in Section 5.

4.10 Relative order of transmissions and retransmissions

While the previous sections have described how the source retransmits information in response to receiving a negative acknowledgement, they have not defined exactly when the source retransmits that information. The answer is that the source can retransmit information any time after receiving a negative acknowledgement, provided it is confident that the retransmission will arrive at the destination before any timeout that would cause the destination to abort the transfer. In practice, the source would retransmit information as soon as it can, without interfering with the original retransmission process. Note that retransmitting information requires fetching information from the disk in an order different from the order in which it is stored in the file, and so may reduce the rate at which the source can transmit information. This should be a non-issue for files stored on hard disks, since they are often fragmented across the disk in any case, but may be significant for files stored on read-only compact disks, or on tape. Yet this must be taken into account when considering, for flow control purposes, the maximum rate at which the source can transmit (see Section 5).

4.11 Transmitting short payload over the packet-switched network

To transfer a file, it is often necessary to transfer information other than just the content of the file. For example, the name and attributes of the file may need to be conveyed, and a client may want to list files that are available on a server in order to select the ones of interest. Some applications, such as FTP, may open a separate transport layer connection to carry this other information. Other applications may require that this information be transmitted over the same transport connection as is used for transmitting the content of the file. Zing will transmit such short units of information over the packet-switched network. It will not provide any guarantee of the relative sequence of short and long units of information.

To ensure that high-capacity circuits are fully utilized when open, it is desirable to avoid prevent layers using the circuit to establish connections, since connection establishment usually involves short units of information and round-trip propagation delays. Instead, Zing allows these short units of information to be exchanged over the packet-switched network, so that the circuit can be fully utilized when it is open.

5 Flow control

Flow control ensures that the source transmits no faster than the destination can receive information. This means ensuring that the source does not transmit bursts that are too large to fit in the destination's buffers, and that the source does not transmit at a mean rate that exceeds the destination's capacity to process information (since this would also cause buffer overflow). Because the destination can write information to disk as it is received without requiring a resequencing buffer (Section 4.9), and because the destination will be able to specify during setup the maximum permissible block size (Section 2), Zing does not need explicit volume-based flow control. Resequencing buffers can also lead to speedup: when a mis-sequenced block finally arrives it can release a large amount of information that had been received but had a later sequence number than the mis-sequenced block, and so had to be held until the arrival of the mis-sequenced block. Because Zing avoids resequencing buffers, it will not require flow control to deal with this speedup.

Zing will use rate-based flow control: The source will transmit blocks at a fixed rate, determined during circuit setup (as described in Sections 8.1.2 and 8.1.3). This rate will take into account the available rate of communication circuits, and the rate at which the source and destination can process blocks. The source will transmit at a rate equal to the minimum of these constraints.

6 Network adaptation

In addition to providing reliable delivery between endpoints, transport layers often need to also match the ends to the network. In particular, the rate at which the transport layer transmits should be matched

to the capabilities of the network. This involves congestion control to prevent the transport layer from transmitting too fast for the network, and rate adaptation to prevent it from transmitting too slow.

6.1 Congestion control

Transport protocols designed for packet-switched networks emphasize congestion control, whereby the transport layer is slowed down to match the capacity of the network. However, Zing does not require this feature for transfers over the circuit, since during circuit setup it reserve along the end-to-end path sufficient resources to support a transfer at a specific rate. Thus, there is no need for techniques such as the Slow Start of TCP which probe for network capacity, but also force link utilization to be lower than its potential.

6.2 Rate adaptation

Zing provides a padding PDU type to allow a source to raise the rate at which it transmits to match the capacity of a link. For example, a pure SONET interface may offer circuits in rates that are integer multiple of the base circuit rate, e.g. 51.84 Mb/s. However, a transport layer may be capable of processing payload information at 80 Mb/s. Rather than selecting a circuit with a rate lower than the transport bottleneck (which could result in no circuit being selected if the transport cannot support 51.84 Mb/s), the transport layer can insert padding PDUs between its payload PDUs to expand its transmission rate to the circuit rate. For example, to match an 80 Mb/s transport to a 103.68 Mb/s OC2 link, the transport layer would insert a padding PDU of length $(103.68-80)(B+64)/80$ bits between each payload block. Receivers can discard padding PDUs, and will effectively receive payload PDUs with the same interarrival intervals as would occur if the source was transmitting at the lower transport rate. Note that in order to use padding on the circuit, payload PDUs must be labelled with their type (c.f. Section 3).

7 Framing

A transport layer must convert between the units of information that it exchanges with the application and the units of information used for transmission. As discussed in the context of error control, this will require the transport layer to segment files that are larger than 1.2MB into several blocks. Segmentation is also required for purposes of flow control, as described in Section 5. The transport layer must also be able to mark where a file ends. This section examines these framing topics.

7.1.1 Block format

Zing sends blocks of payload on the circuit, and possibly over a packet-switched connection. The format for these blocks is shown below. Note that the outer layers of encapsulation are optional, depending on the context in which the block is used. For example, blocks that are transmitted over PPP over a circuit need not include length and integrity check fields since PPP provides that functionality; whereas blocks that are transmitted over a TCP connection do need a length field, but don't need an integrity check value. The rationale behind the format is to locate fields that are usually required closer to the payload so that the core of the format is constant.

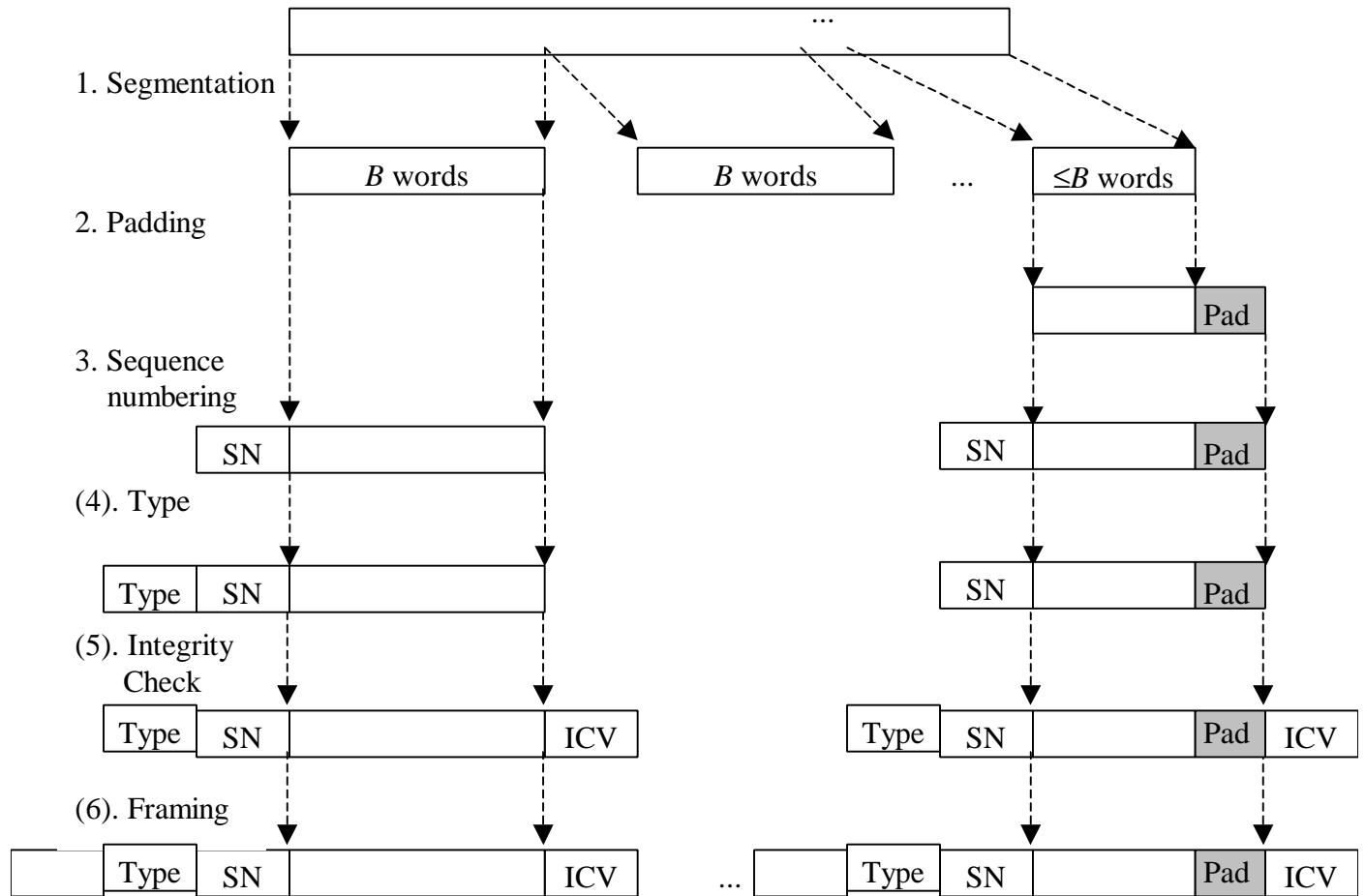


Figure 2: Block format.

Before transferring the file, the transport layer will pad the file so that it is a multiple of 32 bits in length.

To transfer a file, the transport layer will send a series of blocks of fixed-length (B bytes) followed by two blocks, which will be shorter than the fixed-length blocks.

Determination of block size, B : The source of the transfer will determine what block length to use by accounting for the line error rate, block constraints as determined by signalling, as well as the degree of transport layer block overheads. The line error rate will result in a range of preferable block sizes, as analysed in Section 4.1. Block constraints will be indicated during circuit setup, as described in Section 8.1. Transport layer block overheads consist of a 32-bit block sequence number, plus any optional fields such as ICVs and lengths. A realistic block size might be 4KB (to allow the block and sequence number to fit in the 8 KB packets supported by Lucent Optistar NICs). In this case, the sequence number space would span $2^{32} * 2^{12} B = 2^{44}B$. To ensure that the sequence numbers do not wrap before a retransmission arrives at the destination, the bandwidth-delay product should not exceed $2^{44}B$. For example, if the maximum time to receive a retransmission is 5 minutes (the destination will abort the transfer after $TIMER_LONG=5$ minutes of inactivity), then the link could operate at speeds of up to roughly 256 Gb/s ($2^{44}B / 300 \text{ sec} < 2^{44} / 2^9 = 2^{35} = 256 \text{ Gb/s}$).

The type field is only required when multiple types of blocks pass over a link. Hence, it is required for blocks that are retransmissions over the packet-switched network, and when padding blocks co-exist with payload blocks on the circuit (see Section 6.2).

Step 6 of the block format shows framing encapsulating the integrity check. This is the common technique when the framing is tied to decoding, e.g. when using byte stuffing, or when using a length field that has its own separate integrity check. However, it does not consider the alternative of encapsulating the framing information within the ICV. There will also be situations in which the integrity check will not be needed, while framing will be needed. An example of this occurs for the retransmissions over TCP.

The last block of the transfer will be of the “padding length indication” type and will indicate the number of bits of padding that were added to the file.

The second last block of the transfer will be empty if the length of the padded file is an integer multiple of B . Otherwise, the second last block will contain the remainder of the file that could not fit in the series of fixed-length blocks. That is, if the file is F bytes long, then the second last block will carry

$$4 \left\lceil \frac{F - \left\lfloor \frac{F}{B} \right\rfloor B}{4} \right\rceil \text{ bytes.}$$

Any other blocks of the transfer (other than the last and second last block) will be of the fixed block length B .

A receiver can detect the end of the file by noticing a block whose length does not match the maximum block size.

The reason for the transport protocol indicating the amount of padding rather than the file length is that the amount of padding will be bounded to at most 4 bytes, whereas the file length need have no bound.

8 Signalling in support of file transfers

This section describes how Zing establishes and releases connections that are used for file transfers.

8.1 Setup

Definitions: Zing is designed to transport information from a source to a destination. The bulk of the transfer is from the source to the destination, although Zing may also carry upper-layer information from the destination to the source (e.g. FTP application information). When the application passes information to the transport layer to be transmitted, it specifies whether that information should be transmitted over the circuit or packets. Either the source or the destination may initiate a transfer. The node that initiates the transfer is known as the “initiator”, and the other node is known as the “target”.

To summarize the setup process, assuming that all things go well:

1. The initiator sends a request to the target.
2. The target responds with candidate times for the transfer. Switches refine these candidate times as the response propagates back to the initiator.
3. The initiator selects a particular candidate time, and sends a signal committing it and the other nodes commit to the transfer at that time.

The detailed description follows:

Note that this description of signalling assumes, in the main, that the signalling messages are delivered “reliably” (i.e. they are not lost, unduly delayed, or mis-sequenced en route to their destination, and their value is not changed en route).

8.1.1 Requesting a transfer

The initiator must be able to determine the addresses of the target on both the packet-switched and the circuit-switched networks, e.g. by performing a DNS lookup. We will call these two addresses the “packet address” and the “circuit address”, respectively.

To initiate a transfer, the initiator does two things: It establishes a TCP connection to the target’s packet address, and it attempts to determine the reachability of the target’s circuit address. These two things can be started simultaneously. The TCP connection is established in the standard manner. To determine circuit reachability, the initiator sends a “setup request packet” (to be defined shortly) to the target’s circuit address with loose source routing indicating that the “setup request packet” should only be forwarded to nodes that can also be accessed by a circuit. The initiator waits for one of two responses: an ICMP host unreachable message, or a “setup response packet”. If it receives an ICMP host unreachable message, then the transfer will occur with TCP as with a conventional file transfer. If the initiator receives a “setup response packet” then the transfer can occur using the circuit. If the initiator does not receive any response within 10 minutes, then it will abort the file transfer by sending an abort_transfer message, and will close the TCP connection.

The format of the setup request packet is as follows:

```
1 x 32 bit word: type
1 x 32 bit word: if(type==request_from_source)
                  file_length
                  else if(type==request_from_destination)
                  identifier
```

The “type” field can assume one of two values: “request_from_source” or “request_from_destination”. If the type of the setup request message is “request_from_source” then the second 32-bit word will indicate as an unsigned integer indicating the number of 32-bit words in the file to be transferred. (Later this will be encoded in a form that allows arbitrarily long files to be transferred.) If the type of the setup request message is “request_from_destination” then the second 32-bit word will provide an identifier for the file transfer.

When a source initiates a transfer, it will use a setup request packet that indicates the length of the file to be transferred. When a destination initiates a transfer, it will use a setup request packet that indicates an identifier for the file transfer. The destination is free to choose any identifier for the file transfer, provided the identifier does not match that of an existing file transfer, and provided that the identifier has not been used in the previous 10 minutes.

When a target transport layer receives a setup request packet that contains a file transfer identifier, then it will wait to receive an indication from the target application that has the same identifier. This indication will include the file length, and other information to be described shortly. If the target transport layer does not receive such an indication within 5 minutes, then it will discard the setup request packet.

8.1.2 Disseminating candidate transfer intervals

Once the target transport layer has determined the file length (either from a setup request from the source or from the application), it will issue a setup response to the initiator. The setup response has the following form:

```
1 x 32 bit field: type = response
1 x 32 bit unsigned integer: F (file length, in bytes)
1 x 32 bit unsigned integer: block_required_maxlen (in 32 bit words)
```

1 x 32 bit unsigned integer: block_required_minlen (in 32 bit words)
1 x 32 bit unsigned integer: block_preferred_maxlen (in 32 bit words)
1 x 32 bit unsigned integer: block_preferred_minlen (in 32 bit words)
followed by one or more of the following triplets describing candidate transfer intervals:
1 x 32 bit unsigned integer: available_rate (in bits per millisecond)
1 x 32 bit unsigned integer: start_time (in seconds from 1970, ala Unix)
1 x 32 bit unsigned integer: end_time (in seconds from 1970, ala Unix)
followed by one instance of
1 x 32 bit string of 0s.

The target will complete this setup response using the file length as previously determined. While the file length is needed for network-level scheduling, the transport layer can avoid using it by appropriate choice of framing method, as discussed in Section 7.

The block fields indicate constraints on the size of blocks that the target and intermediate systems can accept. There are two types of fields: required constraints (which must be satisfied in order for the transfer to proceed) and preferred constraints (which are desirable for performance reasons, but need not be satisfied during the transfer). These constraints reflect such things as the sizes of buffers in nodes and the type of framing used at lower layers. Examples of required constraints include Optistar OC12 NICs limiting frames to 8KB in length, and PPP limiting frames to 64KB in length. The target may seek to align the size of frames used with the size of blocks used for disk storage, so as to improve performance, leading to a preferred constraint. The target must set the required constraint fields to non-zero values, although it can set the preferred constraint fields to zero if it has no constraints. Block constraint are measured in 32 bit words.

The target will also describe when it can handle a transfer of the proposed length, indicating the rate at which it can process information during the transfer, and the start and end times for when that rate applies transfer. The value of 0 for a start_time or end_time indicates “whenever”, e.g. the transfer can start or end at any time for that rate. The product of the available_rate and interval between start_time and end_time must match or exceed the F (i.e. $\text{available_rate} * (\text{end_time} - \text{start_time}) \geq F$ if the start_time and end_time are non-zero). The target is encouraged to indicate the broadest possible interval over which the specified rate is available, since other nodes may have to use a lower rate, and so extend the holding time. (A simple extension would be to allow a file to be transferred over discontinuous intervals.)

Note that the reason for separating the dissemination of candidate transfer times from the initial transfer request is that the initiator may be the destination, and so may be unable to supply the file length which is needed for evaluating candidate times. The setup delay could be reduced by an end-to-end delay if source initiators started the candidate determination process with their request. Clearly this increase in performance increases complexity, and so has been omitted from the initial version of the protocol.

Note also that the following description of this protocol assumes that all nodes are synchronized and that a second is a sufficiently fine scheduling granularity. Both assumptions could be relaxed in future work.

The setup response will be sent back to the initiator along the path of the circuit. Each switch along the circuit refines the block constraints and list of candidate transfer intervals so that they also reflect its requirements. For example, if a switch is not available during a candidate interval listed in an incoming setup response, then it would delete the triplet for that interval from the setup response.

Revision of block constraints: If a switch cannot meet the required block constraints, then it should set the preferred constraint fields to match the incoming required constraint fields, and set the require

constraint fields to 0 to indicate failure to meet the constraints somewhere along the path. If a switch has more limiting block constraints that are reflected by an incoming message (larger minima or smaller maxima) then it should revise the block constraints accordingly. Note that a preferred constraint value of 0 indicates no constraint, rather than a constraint of 0 length.

Revision of candidate intervals: If a switch was available only for part of the time of a candidate interval, or at a reduced rate, than it would modify the corresponding candidate interval. Note that it must be possible to transmit the file length at the available rate of any candidate intervals. Any other intervals should be deleted from the list of candidates.

8.1.3 Committing to one candidate transfer interval

By the time that the setup response reaches the initiator, it will indicate any candidate intervals that are mutually agreeable to the target and all switches. It will apply its own availability criterion to the remaining candidates. If no candidates remain, then the initiator will send a message to the target indicating that the transfer is to be aborted due to inadequate candidates (type = inadequate_candidates), and will close the TCP connection. If one or more candidates remain, then the initiator will select the candidate that would complete the transfer earliest (i.e. the candidate for which $\text{start_time} + \text{available_rate} * F$ is lowest). (The initiator may have to account for delays that must be incurred before the transfer can start that would postpone the start time.)

When the initiator has selected a candidate, it will remember the rate and start time for that candidate, and send a “commit” message through the network to ensure that the target and switches commit to the candidate that it has selected. The format of a commit message is as follows:

1 x 32 bit field: type = commit
1 x 32 bit unsigned integer: F (file length, in bytes)
1 x 32 bit unsigned integer: rate (in bits per millisecond)
1 x 32 bit unsigned integer: start_time (in seconds from 1970, ala Unix)
1 x 32 bit unsigned integer: end_time (in seconds from 1970, ala Unix)
1 x 32 bit boolean: payload_will_include_type

Unlike in response messages, the end_time must be set so that $\text{end_time} = \text{start_time} + F * \text{rate}$. Furthermore, the start_time must be specific (not “whenever”).

Note that available rates may be quantized, e.g. a SONET network may only offer links in multiples of the base OC1 rate (51.84 Mb/s). If a network element receives a commit message that indicates a rate intermediate between two rates that it supports, then it should round the committed rate up to the next highest quantized rate. The initiator would only be able to select a candidate of the intermediate rate if the network element in question suggested a candidate with the “rounded-up” rate. This situation might occur when different network elements have different rate granularities. For example, a transport layer may be capable of handling a rate of 80Mb/s which is intermediate between SONET link rates of 51.84Mb/s and 103.68Mb/s. *We assume that any network element that uses a rate higher than the rate specified in the commit message provides any necessary rate adaptation.* For example, the network element could use the padding type of PDU described in Section 6.2.

Once the source determines the committed times (after transmitting the commit message if the source was the initiator, or after receiving the commit message if the source was the target), then it can commence the transfer over the circuit at the committed start time. (Note that we assume that the TCP connection between source and destination has been established by the start_time. If this does not happen, then when the TCP connection is established, both the source and destination will send abort messages to each other, and the file transfer will be aborted.)

8.2 Connection release

!!!this section is out of date, and needs to be revised to summarise the discussion in previous sections about conclusion of the transfer!!!

The transfer is deemed to be complete when the destination closes the TCP connection. Immediately before closing the TCP connection, the destination will issue its final acknowledgement list. It will start this process if it can match the number of bits in successfully received blocks to the length of the file that the source indicated at the beginning of the transfer. It will also start this process if it times out waiting for information. !!!specify interval!!! In this case, it will also signal to the application that the transfer has been aborted. Thus, the source can determine whether the transfer was successful by the form of the terminal acknowledgement list.

9 Security issues

nacks and DOS attacks

10 References

- [1] R. Grobler and others: "Scheduling of calls with known holding times", *Proc. submitted to Infocom 2001*, 2000
- [2] S. Lin and others: "Automatic-repeat-request error-control schemes", *IEEE Comm. Mag.*, 22(12):5-17, Dec. 1984