Minimum-Latency Transport Protocols with Modulo-N Incarnation Numbers

A. Udaya Shankar
Institute for Advanced Computer Studies
and Department of Computer Science
University of Maryland
College Park, Maryland 20742
shankar@cs.umd.edu

David Lee
AT&T Bell Laboratories
Murray Hill, New Jersey 07974
lee@research.att.com

CS-TR-3046.1

UMIACS-TR-93-24.1

December 15, 1994

Abstract

To provide reliable connection management, a transport protocol uses 3-way handshakes in which user incarnations are identified by bounded incarnation numbers from some modulo-N space. Cacheing schemes have been proposed to reduce the 3-way handshake to a 2-way handshake, providing the minimum latency desired for transaction-oriented applications. In this paper, we define a class of cacheing protocols and determine the minimum N and optimal cache residency time as a function of real-time constraints (e.g. message lifetime, incarnation creation rate, inactivity duration, etc.). The protocols use the client-server architecture and handle failures and recoveries. Both clients and servers generate incarnation numbers from a local counter (e.g. clock). These protocols assume a maximum duration for each incarnation; without this assumption, there is a very small probability ($\approx \frac{1}{N^2}$) of misinterpretation of incarnation numbers. This restriction can be overcome with some additional cacheing.

Keywords: Connection management, cacheing, handshaking, real-time properties.

To appear in *IEEE/ACM Transactions on Networking*, 1995.

First author supported by NSF grant no. NCR-89-04590

1 Introduction

The transport layer in a computer network consists of clients and servers, collectively referred to as entities.[1] We assume that entities can send messages to each other over channels that can lose, reorder, and duplicate messages; this is the typical network service available to the transport layer. Entities and channels can fail and recover. An entity failure is fail-stop; the state of the entity is lost except for stable storage. A channel failure means that the probability of message delivery becomes negligible, i.e. even with retransmissions a message is not delivered within a specified time.

Clients can open and close connections to servers and exchange data over connections. A server can either accept or reject an incoming connection request. (Rejection can occur because the server is inactive (as in a TCP server that is "closed" rather than "listening") or does not have adequate resources to accept new requests.) Between any client-server pair there is at most one connection at any moment. This allows a client (server) to have multiple connections open at the same time, with different servers (clients).

This is a very general model, which subsumes, for example, the "well-known socket" architecture. Clients and servers typically represent user-level processes within hosts. But they may also represent hosts, with one client and server for each host.

Note that the same client-server pair can undergo many connections over time. We refer to each connection attempt of an entity, whether client or server, as a new *incarnation* of the entity. The notion of incarnations is essential for expressing desired correctness properties, e.g. a client incarnation becomes open to at most one server incarnation. (Incarnations and correctness properties are defined precisely below.)

Traditional transport protocols, including the well-known TCP[8] and TP4[6], identify successive incarnations by increasing (though not necessarily successive) *incarnation numbers* from some modulo-N space[2]. This means that the protocol must be designed to avoid **misinterpretable incarnation numbers**, i.e. an incarnation number of one incarnation being interpreted by an entity (client or server) as representing a different incarnation.

Another feature of traditional transport protocols is that an entity stores a remote incarnation's number only while it is connected to the remote incarnation. This necessitates a 3-way handshake for connection establishment [8, 10]. A client that wants to connect to a server sends a connection request with its incarnation number, say x. When the server receives this, it responds by sending a response containing x and a server incarnation number, say y. When the client receives the

[1] Actually, clients and servers are the users of the transport layer, and for each user there is an entity in the transport layer. But for notational brevity, we use "entity" (and "client" and "server") to refer to both the user and the associated entity.

[2] In TCP, incarnation numbers are referred to as initial sequence numbers.

response, it becomes open to y and responds by sending an ack containing x and y. The server becomes open when it receives the ack. The server could not become open when it received the connection request containing only x (because it may have been an old duplicate, whose acceptance would have violated the at-most-once correctness property mentioned above).

The delay incurred by the 3-way handshake is unacceptable for many transaction-oriented applications such as RPCs. Note that although transaction data can be sent with a connection request, the server cannot process the transaction until it confirms that this is a new request. This has motivated the development of *minimum-latency* transport protocols, where the server can determine the "newness" of a connection request as soon as it is received. This is equivalent to achieving connection establishment with a 2-way handshake.

To achieve this, the server has to retain information about clients even when it is not connected to them. In the above 3-way handshake between client incarnation x and server incarnation y, notice that if the server had remembered the incarnation number, say z, that the client had previously used when it connected to the server, then the server could have determined that the connection request with x was new (because $x > z$). In that case, it could have become open at once; i.e. a 2-way handshake would suffice.

A server cannot be expected to indefinitely remember the last incarnation number of every client to which it was connected, due to the enormous number of clients in a typical internetwork. However, a cacheing scheme is feasible, and several have been proposed (e.g. [5]), culminating recently in a proposed modification to TCP [2].

In this paper, we define a cache-based transport protocol and determine the relationship between various parameters (N, message lifetime, incarnation creation rate, inactivity duration, cache residency times, etc.) that ensures correct operation. The protocol uses an incarnation number generator that is periodically saved on (fail-proof) stable storage. We determine the optimal cache residency time for client incarnation numbers, which ensures minimum latency for every connection establishment. This optimal duration depends upon the message lifetime. Client incarnation numbers can be purged from the cache before this time; the *only* penalty is that connection establishment for such clients reduces to the traditional 3-way handshake for a period equal to the optimal duration.

Another approach to minimum-latency transport protocols is provided by so-called *timer-based* mechanisms (e.g. SCMP[7], Delta-t[4, 11], VMTP[3]). Here also, a server is required to maintain per-client information for a certain duration that depends on message lifetime. However if the entry is purged before this time, there is no backup 3-way handshake. Thus the cache-based approach may have a significant advantage in wide-area situations where only large loose bounds on message lifetime are available. (See Section 6.)

There is an intricate relationship between the modulo-N space of the incarnation numbers and the handshaking algorithms. Most references in the literature seem to assume that misinterpretable incarnation numbers (often referred to as "wrap-around") are avoided if

$$N \;\geq\; 2\mathtt{L}/\alpha$$

where α is the minimum time between incarnation creations at an entity, and $\mathtt{L}$ is the maximum message lifetime imposed by the channels.

In fact, we shall show that this condition is not adequate. Consider the messages in transit from, say, a client to a server. The condition does ensure that messages sent by different client incarnations have different client incarnation numbers. Thus the client can correctly interpret these client incarnation numbers, because it knows the highest incarnation number, say z, that it has sent. But it does not ensure that the server can correctly interpret these client incarnation numbers, because the server, unless it is already connected to the client, may have an old incarnation number of the client, say y. The server interprets received client incarnation numbers with respect to y; and the difference between y and z is not necessarily bound by the message lifetime. Furthermore, the server may send messages containing y to the client, and the above condition does not ensure that the client can correctly interpret these numbers.

With any connection management mechanism, unless a bound is placed on the lifetime of an incarnation (or equivalently, a connection), it is possible to misinterpret incarnation numbers and incorrectly open a connection. In the case of 3-way handshake protocols, the probability of this is negligible for any reasonable value of N because an incarnation becomes open only when a received incarnation number *equals* a local incarnation number. However in the case of 2-way handshake protocols, whether cache-based or timer-based, the probability can become significant because equality testing at the server is now replaced by comparison testing (i.e. is the cached number less than the received number?). Our cacheing protocol achieves the same low probability in both 2-way and 3-way handshakes (by not using cached entries that are "too old").

In the following subsections, we define desired correctness properties and summarize our contributions.

1.1 Correctness properties

We first make precise the notion of incarnations. Recall that the transport layer consists of a set of entities, partitioned into clients and servers. Every entity has a unique id.

An incarnation of a client is started whenever the client requests a connection to any server. An incarnation of a server is started whenever the server accepts a (potentially new) connection request from any client. Every incarnation is assigned an incarnation number when it starts; the incarnation is uniquely distinguished by its incarnation number and entity id.

Once an incarnation x of an entity a is started in an attempt to connect to an entity b, it has one of two possible futures (based on messages it receives):

(1) At some point x becomes open and acquires an incarnation number y of some incarnation of b (in short we say *x becomes open to incarnation y of b*); at some later point x becomes *closed*.

(2) The other possibility is that x becomes closed without ever becoming open.

Because an incarnation becomes open to at most one incarnation, this ensures "at-most-once" semantics; i.e. impossibility of two remote incarnations y and z that are both open to x.

A client incarnation closes without becoming open either because its connection request was rejected by the server or because of failure (in the server, the client, or the channels). A server incarnation closes without becoming open either because of failure or because it was started in response to a connection request that later turns out to be a duplicate request from some old (now closed) incarnation. Because of failures, it is also possible that an incarnation x of a becomes open to incarnation y of b but y becomes closed without becoming open.

A *connection* is an association between two open incarnations. Formally, a connection exists between incarnation x of entity a and incarnation y of entity b if y has become open to x and x has become open to y. The following are desired **correctness properties**:

- **Consistent connections**: If an incarnation x of entity a becomes open to an incarnation y of entity b, then incarnation y is either open to x or will become open to x unless there are failures.
- **Consistent data-transfer**: If an incarnation x of entity a becomes open to an incarnation y of entity b, then x accepts received data only if sent by y.
- **Progress**: If an incarnation x of a client requests a connection to a server, then a connection is established between x and an incarnation of the server within some specified time, provided the server does not reject x's request and neither client, server nor channels fail within that time.
- **Terminating handshakes**: An entity cannot stay indefinitely in a state (or set of states) where it is repeatedly sending messages (expecting a response that never arrives). (Such "infinite chatter" is worse than deadlock because in addition to not making progress, the protocol is consuming precious network resources.)

1.2 Our contribution

In this paper, we specify a class of cacheing protocols and obtain the relationship between various parameters that ensures correctness and minimum-latency performance. Channels can lose, reorder, and duplicate messages, and entities and channels can fail and recover (as described at the beginning of Section 1).

In our protocols, each server caches the latest incarnation numbers of client incarnations that have connected to the server. Connection establishment is achieved in a 2-way (3-way) handshake if an entry for the requesting client is (is not) found in the cache. Connection closing and connection request rejection is achieved by a 2-way handshake. In addition to data transfer with the connection establishment phase, there is also a data transfer phase which can use any of the typical data-transfer mechanisms (e.g. sliding window). Closing can be merged with connection establishment, resulting in a connection consisting entirely of a single 2-way or 3-way handshake.

Each entity (client and server) has a maximum "wait" duration. If a response is outstanding for longer than that duration, it assumes failure (of the remote entity or channels) and aborts the connection.[3] Each entity also has a minimum wait duration; a response must be outstanding for at least this duration before the entity can abort. When a failed client recovers, it can request a new connection after waiting a minimum recovery time. Thus it is possible for a server connected to client to receive a connection request from the client with a higher incarnation number (if the client failed, recovered, and issued a new request). In that case, the server closes its current incarnation and (optionally) can start a new connection with a new incarnation. The same can happen to the client: if the server fails, recovers, receives an old duplicate connection request and responds to it; when the client receives the response, it closes its current incarnation and (optionally) can start a new connection with a new incarnation.

Our protocols can accommodate any size of the server cache, including no cache at all. However, if an entry is cached, then it must be cached for a minimum time (unless the server crashes), otherwise correctness can be violated; it can be flushed out any time after that but before a maximum cache residency time. The cache can be lost in a crash at any time.

Ideally, a server's cache should be large enough to store the latest incarnation number of every client that has connected to the server for a period of the maximum message lifetime plus client wait duration. In this case, the 3-way handshake can be eliminated entirely.

In our class of protocols, denoted SC (for Server-Cache), each entity has an *incarnation number generator* which provides the incarnation numbers for local incarnations. The generator cycles through successive modulo-N values. We assume that successive values of the generator are sep-

[3] Thus, every opening and closing interval is bounded by this duration, as is any open period where a data ack is outstanding.

arated by at least α seconds, and that this suffices for successive incarnations to have different incarnation numbers (i.e. at least α seconds elapses between incarnation creations at the entity). Thus the generator can be a counter that is incremented by 1 for each new incarnation. It can also be a real-time clock with maximum rate of one tick every α seconds.

We assume that periodically, once every Δ seconds, the value of the generator is saved in stable (fail-proof) storage. If an entity crashes, then it must wait a minimum "recovery time" that is at least Δ seconds before it can request a new connection (otherwise correctness can be violated). Upon recovery, the generator is set to the saved value plus Δ/α. This ensures that crashes do not falsify our above assumption about the generator. Different entities can have different Δ's.

We first show that the correctness conditions are satisfied if and only if the following timing constraints hold:

$$\textbf{(T1)} \qquad c_S > \mathtt{W}_C \quad \text{and} \quad r_S > \mathtt{W}_C \quad \text{and} \quad w_C > \mathtt{W}_S \quad \text{and} \quad r_C > \mathtt{W}_S$$
$$\text{and} \quad r_C > \Delta_C \quad \text{and} \quad r_S > \Delta_S$$

where

- $\mathtt{W}_C$ ($\mathtt{W}_S$) is the maximum wait duration for the client (server).
- w_C (w_S) is the minimum wait duration for the client (server); $w_C \leq \mathtt{W}_C$ and $w_S \leq \mathtt{W}_S$.[4]
- c_S is the minimum duration of an entry in the server cache (barring crashes).
- r_C (r_S) is the minimum recovery time for the client (server).
- Δ_S (Δ_C) is the time between saves to the server's (client's) stable storage.

T1 is the only constraint we place on the minimum cache residency time, the minimum recovery delays, and the minimum wait times. It is worth pointing out that **T1** does not depend upon message or incarnation lifetimes.

To illustrate the necessity of **T1**, suppose that the third condition does not hold, i.e. $w_C < \mathtt{W}_S$. Then the following is possible (see Figure 8): a client incarnation x becomes open to a server incarnation u which is not yet open (assume 3-way handshake); the client sends a primary message, receives no ack, aborts (after w_C of becoming open); it then immediately issues a new connection request with incarnation number y, which is accepted by u. At this point both x and y have become open to u, thereby violating the consistent connections property.

We next show that modulo-N incarnation numbers can be used if N satisfies the following

$$\textbf{(T2)} \qquad N \times \alpha \geq 2\mathtt{L} + \mathtt{W}_S + \max(2\mathtt{W}_C + \mathtt{C}_S,\ 2\mathtt{L} + 2\mathtt{W}_C + \mathtt{W}_S,\ 2\mathtt{L} + \mathtt{W}_S + \mathtt{I})$$

where

- α is (as mentioned above) the minimum time between successive values of the incarnation

[4] T1 does not constrain w_S, i.e. allows it to be 0.

number generator.

- $\mathtt{L}$ is the maximum message lifetime in a channel.
- $\mathtt{C_S}$ is the maximum duration of an entry in the server cache; we assume $\mathtt{C_S} > \mathtt{L} + \mathtt{W_C}$.[5]
- $\mathtt{I}$ is the maximum duration of an incarnation.

For many purposes, $\mathtt{I}$ is much greater than $\mathtt{W_C}$ and $\mathtt{W_S}$, and the above bound can be approximated by $N \times \alpha \geq 2\mathtt{L} + \max(\mathtt{C_S}, \mathtt{I})$. Note that the constraint on N, $N \times \alpha \geq 2\mathtt{L}$, is too small to guarantee a correct connection.

If a server stores the incarnation number of a client for at least $\mathtt{L} + \mathtt{W_C}$ seconds and there has been no crash for at least $\mathtt{L} + \mathtt{W_C}$ seconds, then all connections with that client can be opened through 2-way handshakes. That is, if there is no cache entry for an incoming connection request of that client, then it is guaranteed to be a new one (and the server can become open at once).[6] In particular, if we make the (admittedly unrealistic) assumption that the cache is never lost in crashes, we can obtain a much simpler special case of the SC protocol that uses only 2-way handshakes.

The only drawback with the SC protocols is N's dependence on $\mathtt{I}$, if one is concerned about exceedingly long-lived incarnations, say, of the order of days. If we assume that the probability of two successive connections having identical modulo-N client and server incarnation numbers is negligible (this probability is the same as for 3-way handshakes and is $\approx \frac{1}{N^2}$ under reasonable assumptions of incarnation lifetimes), then the following bound which does not depend on $\mathtt{I}$ suffices:

(T2') $\quad N \times \alpha \geq 2\mathtt{L} + 2\mathtt{W_C} + \mathtt{W_S} + \max(\mathtt{C_S}, 2\mathtt{L} + \mathtt{W_S})$

For those situations where this probability is not negligible, we can completely eliminate the $\mathtt{I}$ constraint by using an additional cache, referred to as a *Lin-generator cache*, at either the server or the client (or both). The entity with the generator cache, say a, stores its own incarnation number from its previous connection with the other entity, say b, for at least $2\mathtt{L}$ seconds. When a is next involved in a connection (or connection attempt) with b, if its generator cache contains an entry for b, it uses an incarnation number one higher than the entry; otherwise, it uses an arbitrary incarnation number. We require the admittedly unrealistic assumption that the generator cache is not lost in crashes, unlike the usual server cache. Given such a generator cache, **T2'** ensures correct interpretation of incarnation numbers, and hence correct operation.

We believe the above results are applicable to TP4 and TCP, because our SC protocols (without

[5] $\mathtt{C_S} < \mathtt{L} + \mathtt{W_C}$ results in suboptimal performance, as we see below.

[6] $\mathtt{W_C}$ seconds after the server caches the client incarnation number, say x, the client no longer sends any connection request with incarnation number x; after another $\mathtt{L}$ seconds, no such connection request would be in the channels.

cacheing) use basically the same handshakes as TP4 and TCP[7]. In our protocol as specified, only the client can close a connection; however it is straightforward to add messages for the server to request closing and to achieve graceful closing (where each side must issue a close). Our incarnation numbers correspond to TCP's initial sequence numbers; TCP usually generates them from a clock. The fact that TCP uses the same sequence number space for connection management messages and data bytes is irrelevant. Performance functions such as flow control, slow start, etc., are orthogonal to the correctness problem. There are some differences between our protocol and TCP (e.g. in the use of reject and disconnect messages), but we believe that they do not affect the handshakes.

1.3 Organization of paper

In Section 2, we specify a protocol SC1, which assumes unbounded incarnation numbers, and prove that it satisfies the correctness conditions. In Section 3, we first prove that the unbounded incarnation numbers in SC1 satisfy certain bounds with respect to their intended receivers. Based on these bounds, we modify SC1 to use modulo-N incarnation numbers, resulting in protocol SC2. In Section 4, we obtain a simplified protocol SC3, which eliminates 3-way handshakes entirely. In Section 5, we present the Lin-generator cache. Section 6 contains concluding remarks and a comparison with timer-based approaches. Details of proofs are in two Appendices.

2 Protocol SC1: Unbounded Incarnation Numbers

Every entity (client and server) has a local incarnation number generator that supplies incarnation numbers for local incarnations. The generator goes through successive values separated by at least α seconds. Successive incarnations obtain different (and not necessarily consecutive) incarnation numbers. Thus, the generator can be a counter or a real-time clock. We assume that periodically, once every Δ seconds, the value of the generator is saved in stable (fail-proof) storage.

Consider a client-server pair. The server caches an incarnation number x of a client in two ways: (1) if it becomes open to client incarnation x as a result of a 3-way handshake; and (2) if it receives a connection request from the client with incarnation number x when its cache has a value $y < x$ (whether or not the server accepts the request). In both cases, the cacheing of x signifies that the server has not connected previously to incarnation x or to any later incarnation of the client.

The server does not need to remember the value x after $\mathtt{L} + \mathtt{W_C}$ seconds since cacheing x. This is because any connection request received after that time comes from a later incarnation to which the server can open at once[8]. In fact, we require that it not use the value x after this time,

[7] Ignoring TCP's balanced connection establishment, a feature which apparently is not used in practice.

[8] $\mathtt{W_C}$ seconds after the server caches x, the client no longer sends any connection request with incarnation number

otherwise incorrect operation can result[9]; intuitively, x remaining too long in the cache is similar to x remaining too long in the channels. Thus after this time, the server just needs to remember that $\mathtt{L}+\mathtt{W_C}$ seconds have elapsed since the cache was updated. Note that it must remember this much; otherwise it would not be able to distinguish this situation from a post-crash period when the cache is lost and a 3-way handshake is needed.

There are various ways to manage the cache. To describe them, we define a (hypothetical) cache entry for every client. The entry equals

- the incarnation number of the client incarnation which last connected (or attempted to connect) to the server provided this occurred less than $\mathtt{C_S}(>\mathtt{L}+\mathtt{W_C})$ seconds ago; or
- the special value `old`, signifying that at least $\mathtt{L}+\mathtt{W_C}$ seconds have elapsed since the entry was assigned an incarnation number; or
- the special value `nil`, signifying that nothing is known about the last connection attempt by the client.

A 2-way handshake can be used in the first two cases, and a 3-way handshake is needed in the last case.

One way to manage the cache is to store only entries of the first and second type, i.e. incarnation numbers and `old` entries. Whenever the cache becomes too large, some client entry is removed, i.e. set to `nil`. This approach has maximum flexibility in that any cache replacement policy can be used and both incarnation numbers and `old` entries can be ejected.

Another way to manage the cache is to never discard incarnation number entries unless there is a crash. Therefore an entry can be `nil` only after a crash and that too only for $\mathtt{L}+\mathtt{W_C}$ seconds. Thus we can purge `old` and `nil` values from the cache provided we remember (in a separate variable) whether a crash recovery has taken place less than $\mathtt{L}+\mathtt{W_C}$ seconds ago. If it has, a 3-way handshake is used when there is no cache entry; in all other cases a 2-way handshake is used.

Convention: Throughout, we use a to range over client ids and b to range over server ids. For clarity, we assume that the incarnation number generator goes through consecutive integers 0, 1, $\cdots$. □

Each client a maintains the following state variables:

$LinGen_a$: $\{0, 1, \cdots\}$. Local incarnation number generator. Initially any value.

$Status_a(b)$: {`closed`, `opening`, `open`, `closing`}. Initially `closed`.

Status of client's relationship with server b. `closed` iff client has no incarnation involved with b. `opening` means client has an incarnation requesting a connection with b. `open` means client

x; after another L seconds, no such connection request would be in the channels.

[9] Another consequence would be that the probability of incorrectness associated with T2' would not be negligible; see end of Section 3.

has an incarnation open to b. `closing` means client has an incarnation closing a connection with b.

$Lin_a(b)$: $\{\texttt{nil}\} \cup \{0, 1, \cdots\}$. Initially `nil`.

Local incarnation number. `nil` if $Status_a(b) = \texttt{closed}$. Otherwise identifies client incarnation involved with server b.

$Din_a(b)$: $\{\texttt{nil}\} \cup \{0, 1, \cdots\}$. Initially `nil`.

Distant incarnation number. `nil` if $Status_a(b)$ equals `closed` or `opening`. Otherwise identifies the incarnation of server b with which the client incarnation is involved.

Each server b maintains the following state variables:

$LinGen_b$: $\{0, 1, \cdots\}$. Local incarnation number generator. Initially any value.

$Status_b(a)$: $\{\texttt{closed}, \texttt{opening}, \texttt{open}\}$. Initially `closed`.

Status of server's relationship with client a. `closed` iff server has no incarnation involved with a. `opening` means server has an incarnation accepting a connection request from a. `open` means server has an incarnation open to a.

$Lin_b(a)$: $\{\texttt{nil}\} \cup \{0, 1, \cdots\}$. Initially `nil`.

Local incarnation number. `nil` if $Status_b(a) = \texttt{closed}$. Otherwise identifies server incarnation involved with client a.

$Din_b(a)$: $\{\texttt{nil}\} \cup \{0, 1, \cdots\}$. Initially `nil`.

Distant incarnation number. `nil` if $Status_b(a) = \texttt{closed}$. Otherwise identifies the incarnation of client a with which the server incarnation is involved.

$Cache_b(a)$: $\{\texttt{nil}, \texttt{old}\} \cup \{0, 1, \cdots\}$. Initially `nil`.

Cache entry for client a. (The meaning of the values have been described above.) Note that if the server is open and $Cache_b(a) \neq \texttt{old}$, then $Cache_b(a) = Din_b(a) \neq \texttt{nil}$.

We next describe the messages exchanged between clients and servers. Each message is of the form ($\texttt{M}$, sid, rid, sin, rin), where `M` is the type of the message, sid is the sender's id, rid is the intended receiver's id, sin is the sender's incarnation number, and rin is the intended receiver's incarnation number. In some messages, sin or rin may be absent. For notational brevity, we have omitted the optional data fields in messages, and messages related to the data transfer phase.[10] (Concerning the analysis of misinterpretable incarnation numbers, data transfer messages are equivalent to the DR and DRACK messages defined below.)

[10] Such messages contain all the fields mentioned above and additional fields such as sliding window sequence numbers and size. It is trivial to add the data transfer function to the connection management protocol defined here [9].

Each message is either a *primary* message or a *secondary* message. A primary message is sent repeatedly[11] until a response is received or the maximum wait duration has elapsed. A secondary message is sent only in response to the reception of a primary message. Note that the response to a primary message may be another primary message (as in a 3-way handshake).

We next list the messages sent by clients:

(`CR`, sid, rid, sin). Connection request. Sent when opening. Primary message.

(`CRRACK`, sid, rid, sin, rin). Acknowledgement to connection request reply. Secondary message.

(`DR`, sid, rid, sin, rin). Disconnect request. Sent when closing. Primary message.

(`REJ`, sid, rid, rin). Reject response to a connection request reply (`CRR`) which is received when closed. The sin of the received `CRR` is used as the value of rin. Secondary message.

The messages sent by servers are as follows:

(`CRR`, sid, rid, sin, rin). Reply to connection request in 3-way handshake. Sent when opening. Primary message.

(`CRACK`, sid, rid, sin, rin). Acknowledgement to connection request in 2-way handshake. Sent if cache has entry for sid. Secondary message.

(`DRACK`, sid, rid, sin, rin). Response to disconnect request. Secondary message.

(`REJ`, sid, rid, rin). Reject response to a `CR` received when closed. The sin of the received message is used as the value of rin. Secondary message.

Figures 3 and 4 illustrate connection establishment by 3-way and 2-way handshakes. Figures 5 and 6 illustrate connection rejection.

The events of client a are shown in Figure 1, and the events of server b are shown in Figure 2. There are two types of events. A "nonreceive" event has an enabling condition (ec) and an action (ac); the action can be executed whenever the event is enabled. A receive event for a message has only an action; it is executed whenever the message is received. We assume that *LinGen* is incremented by one between successive reads (by an event not shown). We use abbreviations like $sin > Cache_b(a) \notin \{\texttt{nil}, \texttt{old}\}$ to denote $Cache_b(a) \notin \{\texttt{nil}, \texttt{old}\} \wedge sin > Cache_b(a)$.

Failure model: An entity (client or server) can fail and recover at any time. All state information except for stable storage is lost in a failure, and no events are executed while failed. Upon recovery, the entity reinitializes the *Status*, *Lin* and *Din* for every remote entity, and sets *LinGen* to the value in stable storage plus Δ/α. The channels can fail and recover at any time. When the channels are not failed, a primary message is delivered and its response received within the primary message sender's maximum wait time (the channels can still lose, reorder, and duplicate messages).

[11] according to some retransmission policy guided by flow control needs.

□

Theorem 1. Protocol SC1 satisfies the correctness properties of consistent connections, consistent data-transfer, progress, and terminating handshakes, assuming the following:

(T1') $\quad c_{\mathtt{S}} > \mathtt{W}_{\mathtt{C}}$ and $r_{\mathtt{S}} > \mathtt{W}_{\mathtt{C}}$ and $w_{\mathtt{C}} > \mathtt{W}_{\mathtt{S}}$ and $r_{\mathtt{C}} > \mathtt{W}_{\mathtt{S}}$

□

The proof of Theorem 1 is in Appendix A. Figures 7 and 8 show how the consistent-connections property can be falsified if **T1'** does not hold. Note that **T1'** is **T1** without the conditions on $\Delta_{\mathtt{S}}$ and $\Delta_{\mathtt{C}}$; these conditions are used in the next section.

3 Protocol SC2: Modulo-N Incarnation Numbers

We first determine conditions under which we can replace the unbounded incarnation numbers by modulo-N values, for some N.

To illustrate the approach, let's consider the sin numbers from client a received at server b. In protocol SC1, if the server receives an sin value, it tests sin against $Din_b(a)$; there are two possible outcomes: $sin = Din_b(a)$ and $sin > Din_b(a)$ (depending on the message received and the state of the server). If we replace sin by sin mod N, we must also replace each test by an equivalent test for the same purpose.

We can do this if sin is within fixed bounds of $Din_b(a)$, i.e. if $sin \in [Din_b(a) - K_1, \cdots, Din_b(a) + K_2]$ for some K_1 and K_2. In this case, we can replace sin by sin mod N provided $N > K_1 + K_2$ so that there is no ambiguity. Then the test $sin = Din_b(a)$ becomes the test $sin = Din_b(a) \bmod N$. The test $sin > Din_b(a)$ becomes $sin \in [(Din_b(a) - K_1) \bmod N, \cdots, (Din_b(a) + K_2) \bmod N]$.

The same treatment is needed for the rin numbers, which are compared against the server's $Lin_b(a)$. And the same has to be done at the client side. In short, we need to determine conditions under which we can bound the sin and rin numbers received by an entity relative to the entity's Din and Lin variables.

We have the following result. Recall that

- $\mathtt{L}$ is the maximum message lifetime in a channel.
- $\mathtt{W}_{\mathtt{C}}$ ($\mathtt{W}_{\mathtt{S}}$) is the maximum wait duration for the client (server).
- $\mathtt{C}_{\mathtt{S}}$ is the maximum duration of an entry in a server cache.
- α is the minimum time between successive incarnation generations.
- $\mathtt{I}$ is the maximum lifetime of an incarnation.

Lemma 1. Protocol SC1 satisfies the following properties assuming

(T1) $\quad c_{\mathtt{S}} > \mathtt{W}_{\mathtt{C}}$, $r_{\mathtt{S}} > \mathtt{W}_{\mathtt{C}}$, $w_{\mathtt{C}} > \mathtt{W}_{\mathtt{S}}$, $r_{\mathtt{C}} > \mathtt{W}_{\mathtt{S}}$, $r_{\mathtt{C}} > \Delta_{\mathtt{C}}$, $r_{\mathtt{S}} > \Delta_{\mathtt{S}}$.

(a) Every sin in a $\mathtt{CR}$ received at server b when open or closed with $Cache_b(a) \neq \{\mathtt{nil}, \mathtt{old}\}$

satisfies

$sin \in [Cache_b(a) - \frac{\mathtt{L}+\mathtt{W_C}}{\alpha}, \cdots, \ Cache_b(a) + \frac{\mathtt{L}+\mathtt{W_C}+\mathtt{C_S}+\mathtt{W_S}}{\alpha}]$

(b) Every sin in a CR received at server b when opening satisfies

$sin \in [Din_b(a) - \frac{\mathtt{L}+\mathtt{W_C}}{\alpha}, \cdots, \ Din_b(a) + \frac{\mathtt{L}+\mathtt{W_C}+\mathtt{W_S}}{\alpha}]$

(c) Every sin in a CRRACK received at server b when opening satisfies

$sin \in [Din_b(a) - \frac{3\mathtt{L}+\mathtt{W_C}+\mathtt{W_S}}{\alpha}, \cdots, \ Din_b(a) + \frac{\mathtt{L}+\mathtt{W_C}+\mathtt{W_S}}{\alpha}]$

(d) Every sin in a DR received at server b when open satisfies

$sin \in [Din_b(a) - \frac{\mathtt{L}+\mathtt{I}}{\alpha}, \cdots, \ Din_b(a)]$

(e) Every sin in a CRR received at client a when open satisfies

$sin \in [Din_a(b) - \frac{\mathtt{L}+\mathtt{W_S}}{\alpha}, \cdots, \ Din_a(b) + \frac{2\mathtt{L}+\mathtt{W_C}+\mathtt{W_S}}{\alpha}]$

(f) Every sin in a DRACK received at client a when open satisfies

$sin \in [Din_a(b) - \frac{2\mathtt{L}+\mathtt{I}+\mathtt{W_S}}{\alpha}, \cdots, \ Din_a(b) + \frac{2\mathtt{L}+\mathtt{W_S}}{\alpha}]$

(g) Every rin in a REJ or CRRACK received at server b satisfies

$rin \in [Lin_b(a) - \frac{2\mathtt{L}+\mathtt{W_S}}{\alpha}, \cdots, \ Lin_b(a)]$

(h) Every rin in a DR received at server b when server is open satisfies

$rin \in [Lin_b(a) - \frac{\mathtt{L}+\mathtt{W_S}+\mathtt{I}}{\alpha}, \cdots, \ Lin_b(a)]$

(i) Every rin in a REJ or CRACK received at client a satisfies

$rin \in [Lin_a(b) - \frac{2\mathtt{L}+\mathtt{W_C}}{\alpha}, \cdots, \ Lin_a(b)]$

(j) Every rin in a CRR received at client a satisfies

$rin \in [Lin_a(b) - \frac{2\mathtt{L}+\mathtt{W_S}+\mathtt{W_C}}{\alpha}, \cdots, \ Lin_a(b)]$

(k) Every rin in a DRACK received at client a satisfies

$rin \in [Lin_a(b) - \frac{2\mathtt{L}+\mathtt{I}}{\alpha}, \cdots, \ Lin_a(b)]$

□

The proof of Lemma is in Appendix B. The above bounds are tight. That is, for each bound, if the bound is made any tighter, there is a behavior, easily obtained by exploiting the worst-case real-time constraints, that will violate it (Appendix B illustrates this for the upper bound in (a)).

We obtain protocol SC2 by replacing the unbounded incarnation numbers in protocol SC1 by modulo-N values. Lemma 1 tells us the minimum value of N to ensure no misinterpretation of incarnation numbers. Specifically, for an sin (or rin) in $[A - K_1, \ A + K_2]$, we must have $N > K_1 + K_2$ to avoid ambiguity. From the constraints (a)-(k) of Lemma 1, we obtain the following lower bounds on $N \times \alpha$:

(a) $2\mathtt{L} + 2\mathtt{W_C} + \mathtt{C_S} + \mathtt{W_S}$

(b) $2\mathtt{L} + 2\mathtt{W_C} + \mathtt{W_S}$ (subsumed by (a))

(c) $\mathtt{4L} + \mathtt{2W_C} + \mathtt{2W_S}$

(d) $\mathtt{L} + \mathtt{I}$ (subsumed by (f))

(e) $\mathtt{3L} + \mathtt{W_C} + \mathtt{2W_S}$ (subsumed by (c))

(f) $\mathtt{4L} + \mathtt{I} + \mathtt{2W_S}$

(g) $\mathtt{2L} + \mathtt{W_S}$ (subsumed by (c))

(h) $\mathtt{L} + \mathtt{W_S} + \mathtt{I}$ (subsumed by (f))

(i) $\mathtt{2L} + \mathtt{W_C}$ (subsumed by (c))

(j) $\mathtt{2L} + \mathtt{W_S} + \mathtt{W_C}$ (subsumed by (c))

(k) $\mathtt{2L} + \mathtt{I}$ (subsumed by (f))

Combining the above bounds, we obtain:

$$\textbf{(T2)} \qquad N \times \alpha \geq \mathtt{2L} + \mathtt{W_S} + \max(\mathtt{2W_C} + \mathtt{C_S},\ \mathtt{2L} + \mathtt{2W_C} + \mathtt{W_S},\ \mathtt{2L} + \mathtt{W_S} + \mathtt{I})$$

□

We obtain protocol SC2 by modifying SC1 as follows:

- Redefine the domains of variables $LinGen_a$, $Lin_a(b)$, $Din_a(b)$, $LinGen_b$, $Lin_b(a)$, $Din_b(a)$, $Cache_b(a) \neq \{\mathtt{nil}, \mathtt{old}\}$, and message fields sin and rin to be $\{0, \ldots, N-1\}$
- Every test of *equality* involving these variables and fields (e.g. $sin = Din_b(a)$) is unchanged (but now each side is a modulo-N number).
- Replace the test $sin > Cache_b(a)$ in the server when closed or open by
  $$1 \leq sin \ominus Cache_b(a) \leq \frac{\mathtt{L+W_C+C_S+W_S}}{\alpha}$$
  (obtained from part (a) of Lemma 1) where $\ominus$ is the modulo-N subtraction.
- Replace the test $sin > Din_b(a)$ in the server when opening by
  $$1 \leq sin \ominus Din_b(a) \leq \frac{\mathtt{L+W_C+W_S}}{\alpha}$$
  (obtained from part (b) of Lemma 1).
- Replace the test $sin > Din_a(b)$ in the client when open by
  $$1 \leq sin \ominus Din_a(b) \leq \frac{\mathtt{2L+W_C+W_S}}{\alpha}$$
  (obtained from part (e) of Lemma 1).

With all these constraints satisfied, similar to that of Theorem 1, we can prove the correctness properties of SC2:

Theorem 2. Protocol SC2 satisfies the correctness properties provided N satisfies **T1** and **T2**.

□

Almost always, $\mathtt{I}$ is much greater than $\mathtt{W_C}$ and $\mathtt{W_S}$. Then the bound **T2** approximates to

$$N \times \alpha \geq \mathtt{2L} + \max(\mathtt{C_S},\ \mathtt{I})$$

Typically, $\mathtt{I}$ is also much greater than $\mathtt{L}$ and C_S, and the above bound simplifies to

$$N \times \alpha \geq \mathtt{I}$$

For example, if we use 32-bit incarnation numbers ($N = 2^{32}$) and assume a maximum incarnation generation rate of 10^4 incarnations per second, then the above bound requires incarnation lifetimes to be less than 100 hours.

Observe from Lemma 1 that the $\mathtt{I}$ constraint enters the bound **T2** through the `DR` and `DRACK` messages. In each of these messages, when an entity receives the message it tests for $sin = Din \wedge rin = Lin$. Thus, misinterpretation can occur only if client and server incarnation numbers in the current connection are exactly the same as client and server incarnation numbers in the previous (long-lived) connection. The probability of this is very low. Specifically, the probability that the client's new incarnation number equals its old incarnation number is approximately $\frac{1}{N}$, assuming that the duration of an incarnation is uniformly random. The corresponding probability for the server is also approximately $\frac{1}{N}$ under similar assumptions. Thus the overall probability is at most $\frac{1}{N}$, and approximately $\frac{1}{N^2}$ if the start times of the client and server incarnations are independent.

If we are willing to live with this probability of misinterpretation, then we get the following lower bound on N (by ignoring the $\mathtt{I}$ constraint):

(T2') $\quad N \times \alpha \geq 2\mathtt{L} + 2\mathtt{W_C} + \mathtt{W_S} + \max(\mathtt{C_S},\ 2\mathtt{L} + \mathtt{W_S})$

□

4 Protocol SC3: 2-way Handshakes

Protocol SC3 is obtained from protocol SC2 by requiring the following additional constraints:

(T3) Each cache entry is stored for at least $\mathtt{L} + \mathtt{W_C}$ seconds, and
No cache entry is lost due to crash.

□

It is clear from section 2 that this means that $Cache_b(a)$ is never `nil`. Thus the server completely avoids 3-way handshakes and the `opening` state. Protocol SC2 can then be simplified as follows to obtain protocol SC3:

- Remove all clauses in server b whose guard contains $Status_b(a) =$ `opening`.
- Remove the *Send*`CRR` event in server b.
- Remove the *Receive*(`CRR`) event in client a.
- Keep only non-`old` entries in the cache.

5 Lin-Generator Cache for SC Protocols

Constraint **T2** depends on I because a long-lived connection can be immediately followed by another connection between the same client-server pair. Specifically, suppose the first connection started at time t_1 with client incarnation number x and server incarnation number y. Suppose the client closes the connection (just) before $t_1 + N\alpha$, and reopens it at $t_1 + N\alpha$. It is possible (but very unlikely as argued in the previous section) that the new (modulo-N) incarnation numbers chosen by the client and server are again x and y, respectively. In this case, duplicates of the DR and data-transfer messages of the first connection can be received and misinterpreted as belonging to the second connection.

To avoid this, we require that successive connection attempts by the client to the server (or by the server to the client) be identified with "close-by" incarnation numbers. One obvious way to do this is for one of the entities (or both) to cache its incarnation number from its previous connection with the other entity, for at least 2L seconds. We refer to this as a *Lin-generator cache*.

When an entity with a Lin-generator cache becomes involved in a connection attempt with a remote entity, it obtains its local incarnation number as follows: if its Lin-generator cache contains an entry for the remote entity, it uses an incarnation number one higher than the entry; otherwise, it uses an arbitrary incarnation number.

To see why this works, consider client a and server b. Suppose client a maintains a Lin-generator cache. If server b receives a DR (or data-transfer) message when it is open, the sin in the message satisfies $Din_b(a) - \frac{\mathtt{L}}{\alpha} \leq sin \leq Din_b(a)$. If the client a receives a DRACK (or data-transfer) message when it is open or closing, the rin in the message satisfies $Lin_a(b) - \frac{2\mathtt{L}}{\alpha} \leq rin \leq Lin_a(b)$. Thus $N \times \alpha \geq 2\mathtt{L}$ ensures no misinterpretation, and this bound is implied by **T2**.

The same argument holds if server b maintains a Lin-generator cache (and the client does not), except that now DR messages are distinguished by their rin fields, and DRACK messages are distinguished by their sin fields.

6 Conclusions

We have presented a transport protocol that uses server cacheing to achieve 2-way handshake connection establishment, providing the minimum latency needed for transaction-oriented users such as RPCs. When no cache entry is available, the protocol degenerates to 3-way handshakes. By having a sufficiently large cache, 3-way handshakes can be eliminated entirely, i.e. if cache entries are retained for $\mathtt{L} + \mathtt{W_C}$ seconds. Our protocols tolerate crashes, assuming a stable storage that is periodically updated with the current value of the incarnation number generator. If the server cache is lost, then 3-way handshakes have to be used until $\mathtt{L} + \mathtt{W_C}$ seconds elapses. Upon recovery an entity is subjected to only a brief delay, i.e. the maximum wait duration of the other entity; this delay does not depend on message or incarnation lifetimes.

Note that there is a distinction between message lifetime ($\mathtt{L}$) and message delay. The lifetime is the maximum duration for which a message may survive, whereas the delay is the average time for a message to go from sender to receiver. The delay is less than the lifetime, sometimes by several orders especially in internetworks. Certainly $\mathtt{W_C}$ should be larger than twice the delay but it can be much less than the lifetime.

Our protocol uses modulo-N numbers to identify incarnations. We have obtained the minimum value of N that guarantees correct operation. To our knowledge, no such bound has been previously presented in the general setting we have considered.

Unlike other cacheing protocols proposed [2], our protocol does not use a cache entry if it is older than the maximum message lifetime plus maximum client wait duration. This is key to ensuring that even if N does not satisfy the lower bound with respect to maximum incarnation lifetime, the probability of misinterpretation is very low.

Timer-based techniques provide another approach for achieving minimum-latency connection-establishment [1, 11, 4, 7, 3]. Here also, a server is required to maintain information on each client it has served for a certain duration. The duration is roughly comparable to the optimal duration in our cache-based mechanism (the major component in both is the message lifetime).

In most timer-based protocols, if a client's entry is removed before the specified duration (e.g. due to a crash or memory limitation), then the server can incorrectly accept old connection requests of that client. SCMP [7] is an exception: by assuming synchronized clocks, it maintains correctness but it may reject new connections for a period of time depending on clock skews and other parameters. In any case, timer-based approaches do not have a back-up 3-way handshake.

Thus one advantage of the cache-based mechanism is that it provides a 3-way handshake as back-up whenever the 2-way is not possible. This allows the server to implement different cache-replacement policies that adapt to varying client and internetwork characteristics. This should be particularly convenient for wide-area internetworks where only large loose bounds on message lifetime may be available. Other than this qualitatively important difference, it appears difficult to compare the cache-based and timer-based approaches without a quantitative performance evaluation.

We point out that the incarnation lifetime, which appears in our lower bound for N, does not usually show up in the analyses of timer-based mechanisms. This is because these analyses assume that clock values are unbounded. This is not really a reasonable assumption if one is concerned about correctness. Every protocol has clock values that come from a modulo-N space because messages have a fixed number of bits for the clock value. One can argue that N is very large and can be assumed unbounded. If so, that assumption would be equally valid for cache-based protocols and would enormously simplify the analysis almost to the point of triviality. But more to the point, such an assumption may be invalid for very high speed networks (i.e. very small α).

Acknowledgements

We are grateful for the excellent criticism given by the anonymous referees and the editor, Michael Schwartz. In particular, the detailed and insightful comments of referees 1 and 3 gave rise to many clarifications and motivated the present treatment of stable storage.

References

[1] E. W. Biersack and D. C. Feldmeier. A Timer-Based Connection Management Protocol with Synchronized Clocks and its Verification. *Computer Networks and ISDN Systems*, July 1993.

[2] R. Braden. Extending TCP for Transactions – Concepts. Request for Comment RFC-1379, Network Information Center, November 1992.

[3] D. R. Cheriton. VMTP: A Transport Protocol for the Next Generation of Communication Systems. In *Proceedings ACM SIGCOMM '86*, pages 406–415, Stowe, Vermont, August 1986.

[4] J. G. Fletcher and R. W. Watson. Mechanisms for a Reliable Timer Based Protocol. *Computer Networks*, 2(4/5):271–290, September 1978.

[5] L. Garlick, R. Rom, and J. Postel. Issues in Reliable Host-to-Host Protocols. In *Second Berkeley Workshop on Distributed Data Management and Computer Networks*, May 1977.

[6] International Standards Organization. Information Processing Systems – Open Systems Interconnection – Transport Protocol Specification Addendum 2: Class four operation over connectionless network service. International Standard ISO 8073/DAD 2, ISO, April 1989.

[7] B. Liskov, L. Shrira, and J. Wroclawski. Efficient At-Most-Once Messages Based on Synchronized Clocks. *ACM Transactions on Computer Systems*, 9(2):125–142, May 1991.

[8] J. Postel. Transmission Control Protocol: DARPA Internet Program Protocol Specification. Request for Comment RFC-793, STD-007, Network Information Center, September 1981.

[9] A.U. Shankar. Modular Design Principles for Protocols with an Application to the Transport Layer. *Proceedings of the IEEE*, December 1991.

[10] C.A. Sunshine and Y.K. Dalal. Connection Management in Transport Protocols. *Computer Networks*, 2(6), December 1978.

[11] R. W. Watson. The Delta-t Transport Protocol: Features and Experience. In *IEEE 14th Conferance on Local Computer Networks*, pages 399–407, Minneapolis, MN, October 1989.

A Proof of Theorem 1

Theorem 1. Protocol SC1 satisfies the correctness properties of consistent connections, consistent data-transfer, progress, and terminating handshakes, assuming the following:

(T1') $\quad c_{\mathtt{S}} > \mathtt{W_C}$ and $r_{\mathtt{S}} > \mathtt{W_C}$ and $w_{\mathtt{C}} > \mathtt{W_S}$ and $r_{\mathtt{C}} > \mathtt{W_S}$

□

Consistent connections

An incarnation becomes open at most once. If an incarnation x becomes open to an incarnation y, then x does not become open again to another incarnation. Thus, it suffices to prove that there is no incarnation $z(\neq y)$ that was previously open to x.

Incarnation x at a becomes open to incarnation y at b

Case 1: Suppose incarnation x of a started at time t_1 and becomes open to incarnation y at time t_2 upon receiving a $(\mathtt{CRR}, b, a, y, x)$.

When y sent the CRR, say at time u_1, it was opening to x and its cache entry for a was `nil`.

Assume another incarnation z was previously open to x. Suppose it became open at $u_2(t_1 \leq u_2 \leq u_1 \leq t_2)$. At u_1, z was closed and no entry for a remained in the cache.[12]

If b did not crash between u_2 and u_1, we have $u_1 > u_2 + c_{\mathtt{S}}$ (because x stays in cache for at least $c_{\mathtt{S}}$ seconds).

If b did crash between u_2 and u_1, we have $u_1 > u_2 + r_{\mathtt{S}}$ (because b does not respond to connection requests for at least $r_{\mathtt{S}}$ seconds after recovery).

In either case, because of the timing constraint **T1** (and $t_1 \leq u_2 \leq u_1 \leq t_2$), we have $t_2 \geq t_1 + \mathtt{W_C}$. Thus x would be closed before t_2. Contradiction.

Case 2: Suppose incarnation x of a started at time t_1 and becomes open to incarnation y at time t_2 upon receiving a $(\mathtt{CRACK}, b, a, y, x)$.

When b sent the CRACK, it had a cache entry, say u, for a such that $u < x$. Thus no incarnation z could have been open to x (otherwise the cache entry would have equalled x).

Incarnation x at b becomes open to incarnation y at a

Case 1: Suppose incarnation x of b started at time t_1, when b received a $(\mathtt{CR}, a, b, y)$ and no cache entry existed for a. Suppose x became open to y at $t_2(\geq t_1)$, when it received $(\mathtt{CRRACK}, a, b, y, x)$. The only message that x has sent so far is $(\mathtt{CRR}, b, a, x, y)$. Thus no other incarnation z of a could have become open to y.

[12] Thus z became open by 2-way handshake and the CRACK did not reach a; otherwise a would be open by u_1.

Case 2: Suppose incarnation x of b started at time t_1, when b received a $(\texttt{CR}, a, b, u)$ with $u < y$ and no cache entry existed for a. Let b have received a $(\texttt{CR}, a, b, y)$ at time $t_2(\geq t_1)$. Let x have become open to y at $t_3(\geq t_2)$, when it received $(\texttt{CRRACK}, a, b, y, x)$.

Assume some other incarnation z of a became open to x at some time $u_1(\geq t_1)$. By t_2, z had terminated and y started.

If z did not crash, then we have $t_2 > u_1 + w_{\texttt{C}}$ because z would have sent at least one primary message, namely a close a request. (Note that the wait time for the server starts at t_1.)

If z did crash, then we have $t_2 > u_1 + r_{\texttt{C}}$.

In either case, because of the timing constraint **T1** (and $u_1 \geq t_1$), we have $t_2 \geq t_1 + \texttt{W}_{\texttt{S}}$. Thus x would be closed before t_2. Contradiction.

Case 3: Assume incarnation x became open to y at t_1, when b received a $(\texttt{CR}, a, b, y)$ and there was a cache entry $u(< x)$ for a. Then no other incarnation z of a could have become open to x.

Consistent data-transfer

Assume incarnation x of a is open to incarnation y of b. x accepts data only if identified by $rin = x$. Such data is sent only by an incarnation of b that is open to x. From the consistent-connections condition, there is at most one incarnation like that, namely y. Therefore, x accepts data sent by y and no other incarnation.

Progress

Assume x of a requests a connection to b at time t_1. Assume that after t_1, there are no failures and b does not reject requests from a. Then a repeatedly transmits $(\texttt{CR}, a, b, x)$, and from some time t_2, this message is received repeatedly at b. There are three possibilities.

Case 1: b has a cache entry for a. Then b starts an incarnation y and sets it open to x. It responds with $(\texttt{CRACK}, a, b, y, x)$ to this and every $(\texttt{CR}, a, b, x)$ it receives. Eventually, one of the CRACK's reaches a at some time t_3; we have $t_3 \leq t_1 + \texttt{W}_{\texttt{C}}$, because of no failures. Thus x becomes open to y at t_3. The connection is established.

Case 2: b has no cache entry for a, and starts an incarnation y at t_2 that is opening. It repeatedly sends $(\texttt{CRR}, b, a, y, x)$. Eventually one of these reaches a at time t_3, where $t_3 \leq t_1 + \texttt{W}_{\texttt{C}}$ (because of no failures). At t_3, x becomes open to y. And a responds with $(\texttt{CRRACK}, a, b, x, y)$ to this and every $(\texttt{CRR}, b, a, y, x)$ it receives. Eventually, one of the CRRACK's reaches b at some time t_4, where $t_4 \leq t_2 + \texttt{W}_{\texttt{C}}$, because of no failures. Thus y becomes open to x at t_4. The connection is established.

Case 3: b has no cache entry for a, and at t_2 already has an opening incarnation y that was started at some earlier time (due to an old duplicate CR message). It sets y opening to x, and at this point the scenario progresses as in case 2.

Terminating handshakes

Each phase of a handshake is assured of termination because of the maximum wait bounds, $\mathtt{W_C}$ and $\mathtt{W_S}$. There is no infinite chain of handshakes (the 3-way handshake is the highest degree handshake). Thus, every handshake terminates.

B Proof of Lemma 1

Lemma 1. Protocol SC1 satisfies the following properties assuming

(T1) $c_\mathtt{S} > \mathtt{W_C}$, $r_\mathtt{S} > \mathtt{W_C}$, $w_\mathtt{C} > \mathtt{W_S}$, $r_\mathtt{C} > \mathtt{W_S}$, $r_\mathtt{C} > \Delta_\mathtt{C}$, $r_\mathtt{S} > \Delta_\mathtt{S}$

(a) Every *sin* in a `CR` received at server b when open or closed with $Cache_b(a) \neq \{\mathtt{nil}, \mathtt{old}\}$ satisfies
$sin \in [Cache_b(a) - \frac{\mathtt{L+W_C}}{\alpha}, \cdots,\ Cache_b(a) + \frac{\mathtt{L+W_C+C_S+W_S}}{\alpha}]$

(b) Every *sin* in a `CR` received at server b when opening satisfies
$sin \in [Din_b(a) - \frac{\mathtt{L+W_C}}{\alpha}, \cdots,\ Din_b(a) + \frac{\mathtt{L+W_C+W_S}}{\alpha}]$

(c) Every *sin* in a `CRRACK` received at server b when opening satisfies
$sin \in [Din_b(a) - \frac{3\mathtt{L+W_C+W_S}}{\alpha}, \cdots,\ Din_b(a) + \frac{\mathtt{L+W_C+W_S}}{\alpha}]$

(d) Every *sin* in a `DR` received at server b when open satisfies
$sin \in [Din_b(a) - \frac{\mathtt{L+I}}{\alpha}, \cdots,\ Din_b(a)]$

(e) Every *sin* in a `CRR` received at client a when open satisfies
$sin \in [Din_a(b) - \frac{\mathtt{L+W_S}}{\alpha}, \cdots,\ Din_a(b) + \frac{2\mathtt{L+W_C+W_S}}{\alpha}]$

(f) Every *sin* in a `DRACK` received at client a when open satisfies
$sin \in [Din_a(b) - \frac{2\mathtt{L+I+W_S}}{\alpha}, \cdots,\ Din_a(b) + \frac{2\mathtt{L+W_S}}{\alpha}]$

(g) Every *rin* in a `REJ` or `CRRACK` received at server b satisfies
$rin \in [Lin_b(a) - \frac{2\mathtt{L+W_S}}{\alpha}, \cdots,\ Lin_b(a)]$

(h) Every *rin* in a `DR` received at server b when server is open satisfies
$rin \in [Lin_b(a) - \frac{\mathtt{L+W_S+I}}{\alpha}, \cdots,\ Lin_b(a)]$

(i) Every *rin* in a `REJ` or `CRACK` received at client a satisfies
$rin \in [Lin_a(b) - \frac{2\mathtt{L+W_C}}{\alpha}, \cdots,\ Lin_a(b)]$

(j) Every *rin* in a `CRR` received at client a satisfies
$rin \in [Lin_a(b) - \frac{2\mathtt{L+W_S+W_C}}{\alpha}, \cdots,\ Lin_a(b)]$

(k) Every *rin* in a `DRACK` received at client a satisfies
$rin \in [Lin_a(b) - \frac{2\mathtt{L+I}}{\alpha}, \cdots,\ Lin_a(b)]$

□

In the proof below, we make use of the following property: Over any period of time T, $LinGen_a$ (and $LinGen_b$) is nondecreasing and increases in value by at most T/α. This property is holds even with crashes because of the last two conditions in T1 (involving $\Delta_{\mathtt{S}}$ and $\Delta_{\mathtt{C}}$).

Bounds on *sin* in CR received at server

Let the server receive $(\mathtt{CR}, a, b, sin)$ at time t_1. We first obtain bounds on sin in terms of $LinGen_a$. Let l_1 be the value of $LinGen_a$ at t_1. Clearly $sin \leq l_1$, which gives us an upper bound on sin.

To obtain a lower bound on sin, we note that the client sent the sin after time $t_1 - \mathtt{L}$ (because of the L constraint). Furthermore, the client incarnation sin was started at most $\mathtt{W_C}$ seconds earlier (otherwise it would have aborted before $t_1 - \mathtt{L}$). Thus the smallest possible sin is the value of $LinGen_a$ at $t_1 - \mathtt{L} - \mathtt{W_C}$. Hence, we have $sin \geq l_1 - \frac{\mathtt{L}+\mathtt{W_C}}{\alpha}$ (using the α constraint).

Combining the two bounds, we have the following:

$$(1) \quad sin \in [l_1 - \frac{\mathtt{L} + \mathtt{W_C}}{\alpha}, \cdots, \ l_1]$$

The server tests sin against $Din_b(a)$ when opening, and against $Cache_b(a)$ when $Cache_b(a) \notin \{\mathtt{nil}, \mathtt{old}\}$ and server is open or closed. We examine each case separately.

Assume the server is open at t_1, when $Cache_b(a) = d_1$. Let it have become open at $t_2 (\leq t_1)$. Because $Cache_b(a) \neq \mathtt{old}$, we have $t_2 \geq t_1 - \mathtt{C_S}$. The server received the CR to which it opened at $t_3 (\geq t_2 - \mathtt{W_S})$. Let $LinGen_a = l_3$ at t_3. We have $l_3 \geq l_1 - \frac{\mathtt{C_S}+\mathtt{W_S}}{\alpha}$. We also have $d_1 \geq l_3 - \frac{\mathtt{L}+\mathtt{W_C}}{\alpha}$ (applying condition (1) at t_3). Therefore we have $d_1 \geq l_1 - \frac{\mathtt{L}+\mathtt{W_C}+\mathtt{C_S}+\mathtt{W_S}}{\alpha}$, which gives us an upper bound on l_1. For a lower bound, we have the obvious $d_1 \leq l_2 \leq l_1$. Combining the two bounds, we have

$$(2) \quad sin \in [d_1 - \frac{\mathtt{L} + \mathtt{W_C}}{\alpha}, \cdots, \ d_1 + \frac{\mathtt{L} + \mathtt{W_C} + \mathtt{C_S} + \mathtt{W_S}}{\alpha}]$$

Assume the server is closed at time t_1, when $Cache_b(a) = d_1$. It last become open at $t_2 (\geq t_1 - \mathtt{C_S})$. The same treatment as in the case of open applies, resulting in the same bounds (2).

Part (a) of Lemma 1 follows from condition (2).

[**Note:** These bounds are tight because equality is possible with each of the inequalities above. For the upper bound in (2), the following scenario is possible. The $(\mathtt{CR}, a, b, sin)$ received at time t_1 has $sin = l_1$, i.e. it was sent just before time t_1. The server is open at t_1 with $Cache_b(a) = d_1$. It became open at time $t_2 = t_1 - \mathtt{C_S}$, and the CR to which it opened was received at $t_3 = t_2 - \mathtt{W_S}$. Assume $LinGen_a$ ticks at max rate. At t_3, let $LinGen_a = l_3 = l_1 - \frac{\mathtt{C_S}+\mathtt{W_S}}{\alpha}$. The CR received at t_3 was sent at time $t_3 - \mathtt{L} + \mathtt{W_C}$. Thus $d_1 = l_3 - \frac{\mathtt{L}+\mathtt{W_C}}{\alpha}$. Combining these, we have $sin = d_1 + \frac{\mathtt{L}+\mathtt{W_C}+\mathtt{C_S}+\mathtt{W_S}}{\alpha}$.]

Assume the server is opening at time t_1, with $Din_b(a) = d_1$. Then at some time t_2 (where $t_1 \geq t_2 \geq t_1 - \mathtt{W_S}$), it received a CR message with $sin = d_1$. Let $LinGen_a = l_2$ at t_2. By the same argument as in the open case, we have $d_1 \geq l_2 - \frac{\mathtt{L}+\mathtt{W_C}}{\alpha}$ and $l_2 \geq l_1 - \frac{\mathtt{W_S}}{\alpha}$, which give an upper bound on l_1; and $d_1 \leq l_2 \leq l_1$, which gives a lower bound on l_1. Combining with (1), we have

$$(3) \quad sin \in [d_1 - \frac{\mathtt{L} + \mathtt{W_C}}{\alpha}, \cdots,\ d_1 + \frac{\mathtt{L} + \mathtt{W_C} + \mathtt{W_S}}{\alpha}]$$

Part (b) of Lemma 1 follows from condition (3).

Bounds on *sin* in CRRACK received at server

Suppose the server receives $(\mathtt{CRRACK}, a, b, sin, rin)$ at time t_1, when $Din_b(a) = d_1$ and server is opening (otherwise sin is not tested). The CRRACK was sent at some time $t_2(\geq t_1 - \mathtt{L})$, upon reception of $(\mathtt{CRR}, b, a, x, y)$ at time t_2, where $x = rin$ and $y = sin$. The CRR was sent at some time $t_3(\geq t_2 - \mathtt{L})$, by a server incarnation that started at some time $t_4(\geq t_3 - \mathtt{W_S})$, upon reception of $(\mathtt{CR}, a, b, y)$. The CR was sent at some time $t_5(\geq t_4 - \mathtt{L})$, and the client incarnation y was started at some time $t_6(\geq t_5 - \mathtt{W_C})$, at which point $LinGen_a = y(= sin)$. Thus, we have

$$(1) \quad sin \in [l_1 - \frac{3\mathtt{L} + \mathtt{W_C} + \mathtt{W_S}}{\alpha}, \cdots,\ l_1]$$

Now the server became opening at some time $u_1(\geq t_1 - \mathtt{W_S})$, by receiving $(\mathtt{CR}, a, b, d_1)$. Let $LinGen_a = x_1$ at u_1. We have $d_1 \in [x_1 - \frac{\mathtt{L}+\mathtt{W_C}}{\alpha}, \cdots,\ x_1]$ (from condition (1) of the CR case). We also have $x_1 \geq l_1 - \frac{\mathtt{W_S}}{\alpha}$. Combining these with the lower bound on d_1, we have $d_1 \geq l_1 - \frac{\mathtt{L}+\mathtt{W_C}+\mathtt{W_S}}{\alpha}$ We also have $d_1 \leq x_1 \leq l_1$. Combining the last two bounds with (1), we have

$$(2) \quad sin \in [d_1 - \frac{3\mathtt{L} + \mathtt{W_C} + \mathtt{W_S}}{\alpha}, \cdots,\ d_1 + \frac{\mathtt{L} + \mathtt{W_C} + \mathtt{W_S}}{\alpha}]$$

which gives us part (c) of Lemma 1.

Bounds on *sin* in DR received at server

Suppose the server receives $(\mathtt{DR}, a, b, sin, rin)$ at time t_1, when $Din_b(a) = d_1$ (otherwise sin is not tested). Let $LinGen_a = l_1$ at t_1.

The DR was sent at some time $t_2(\geq t_1 - \mathtt{L})$ by the client. The client incarnation sin was started at some time $t_3(\geq t_2 - \mathtt{I})$. Therefore $sin \geq l_1 - \frac{\mathtt{L}+\mathtt{I}}{\alpha}$. Because $d_1 \leq l_1$, we have $sin \geq d_1 - \frac{\mathtt{L}+\mathtt{I}}{\alpha}$.

For an upper bound on sin, we note that DR is sent only by an incarnation that was open (to b). Therefore $sin \leq d_1$.

Thus, we have

$$sin \in [d_1 - \frac{\mathtt{L} + \mathtt{I}}{\alpha}, \cdots,\ d_1]$$

which is part (d) of Lemma 1.

Bounds on sin in CRR received at client

Suppose the client receives $(\mathtt{CRR}, b, a, sin)$ at time t_1, when it is open (otherwise sin is not tested). Let $LinGen_b = l_1$ and $Din_a(b) = d_1$ at t_1.

The CRR was sent at some time $t_2(\geq t_1 - \mathtt{L})$, by a server incarnation that was started at some time $t_3(\geq t_2 - \mathtt{W_S})$, upon reception of a CR. Thus:

$$(1) \quad sin \in [l_1 - \frac{\mathtt{L} + \mathtt{W_S}}{\alpha}, \cdots, \ l_1]$$

The CR was sent at some time $t_4(\geq t_3 - \mathtt{L})$. Thus $t_4 \geq t_1 - 2\mathtt{L} - \mathtt{W_S}$.

Suppose the current client incarnation became open at time u_1. Because the client does not send CR when open, we have $u_1 \geq t_4$. Thus the current incarnation was started at $u_2(\geq u_1 - \mathtt{W_C})$. Therefore $u_2 \geq t_4 - \mathtt{W_C} \geq t_1 - 2\mathtt{L} - \mathtt{W_S} - \mathtt{W_C}$.

Let $LinGen_b = x_2$ at u_2. Therefore $x_2 \geq l_1 - \frac{2\mathtt{L}+\mathtt{W_C}+\mathtt{W_S}}{\alpha}$. Because the current client incarnation started after u_2, its $Din_a(b)$ value, d_1, equals the value of $LinGen_b$ at some point after u_2. Therefore $d_1 \geq x_2$. Combining this with the above, we have $d_1 \geq l_1 - \frac{2\mathtt{L}+\mathtt{W_C}+\mathtt{W_S}}{\alpha}$. And we have $d_1 \leq l_1$. Combining these with (1), we have

$$sin \in [d_1 - \frac{\mathtt{L} + \mathtt{W_S}}{\alpha}, \cdots, \ d_1 + \frac{2\mathtt{L} + \mathtt{W_C} + \mathtt{W_S}}{\alpha}]$$

which is part (e) of Lemma 1.

Bounds on sin in DRACK message received at client

Suppose the client receives $(\mathtt{DRACK}, b, a, sin, rin)$ at time t_1, when $Din_a(b) = d_1$ (otherwise sin is not tested). Let $LinGen_b = l_1$ at t_1.

The DRACK was sent at some time $t_2(\geq t_1 - \mathtt{L})$ by the server, when open or closed. If the server was open, the argument and resulting bounds are the same as for DR, i.e. part(d).

Suppose the DRACK was sent when server was closed. It responded to a DR sent by the client at some time $t_3(\geq t_2 - \mathtt{L})$, and the sin in the DRACK equalled the rin in the DR.

The client incarnation that sent the DR was started at some time $t_4(\geq t_3 - \mathtt{I})$, and it opened to a server incarnation started at $t_5(\geq t_4 - \mathtt{W_S})$. Therefore $rin \geq l_1 - \frac{2\mathtt{L}+\mathtt{I}+\mathtt{W_S}}{\alpha}$. Because $d_1 \leq l_1$, we have $sin \geq d_1 - \frac{2\mathtt{L}+\mathtt{I}+\mathtt{W_S}}{\alpha}$.

For an upper bound on sin, we note that the current client incarnation became open after $t_1 - 2\mathtt{L}$ (otherwise the server would not have received the DR at t_2). Thus it opened to a server incarnation that started after $t_1 - 2\mathtt{L} - \mathtt{W_S}$, i.e. $d_1 \geq l_1 - \frac{2\mathtt{L}+\mathtt{W_S}}{\alpha}$.

Combining the two bounds, we have the following (which subsumes the open case):

$$sin \in [d_1 - \frac{2\mathtt{L} + \mathtt{I} + \mathtt{W_S}}{\alpha}, \cdots, \ d_1 + \frac{2\mathtt{L} + \mathtt{W_S}}{\alpha}]$$

which is part (f) of Lemma 1.

Bounds on rin in REJ or CRRACK received at server

Let the server receive a rin value in a REJ message or a CRRACK message at time t_1, when $Lin_b(a) = l_1$. The REJ or CRRACK message was sent at some time $t_2(\geq t_1 - \mathtt{L})$ by client in response to reception of a message $(\mathtt{CRR}, b, a, s, r)$ with $s = rin$. The $(\mathtt{CRR}, b, a, s, r)$ message was sent by the server at some time $t_3(\geq t_2 - \mathtt{L})$, and the server incarnation s was started at some time $t_4(\geq t_3 - \mathtt{W_S})$. We have $t_1 \leq t_4 + 2\mathtt{L} + \mathtt{W_S}$, and therefore $l_1 \leq rin + \frac{2\mathtt{L}+\mathtt{W_S}}{\alpha}$. Combining this with the obvious bound $rin = s \leq l_1$, we have

$$(1) \quad rin \in [l_1 - \frac{2\mathtt{L} + \mathtt{W_S}}{\alpha}, \cdots, l_1]$$

which is part (g) of Lemma 1.

Bounds on rin in DR received at server

Let the server receive a $(\mathtt{DR}, a, b, rin)$ at time t_1, when the server is open and $Lin_b(a) = l_1$. The DR was sent at some time $t_2(\geq t_1 - \mathtt{L})$ by the client, when $Din_a(b) = rin$ and the client was open or closing. This client incarnation started at some time $t_3(\geq t_2 - \mathtt{I})$, and became open some time later to a server incarnation that was started at some time $t_4(\geq t_3 - \mathtt{W_S})$. At t_4, $LinGen_b = rin$. Therefore $rin \geq l_1 - \frac{\mathtt{L}+\mathtt{W_S}+\mathtt{I}}{\alpha}$, and we have

$$rin \in [Lin_b(a) - \frac{\mathtt{L} + \mathtt{W_S} + \mathtt{I}}{\alpha}, \cdots, Lin_b(a)]$$

which gives part (h) of Lemma 1.

Bounds on rin in REJ or CRACK received at client

The argument and results are the same as for server case (REJ or CRRACK), with CR replaced by CRR, and $\mathtt{W_S}$ by $\mathtt{W_C}$. This gives part (i) of Lemma 1.

Bounds on rin in CRR received at client

Suppose the client receives $(\mathtt{CRR}, b, a, sin, rin)$ at time t_1, when $Lin_a(b) = l_1$ and the client is opening or open. The server sent the CRR at some time $t_2(\geq t_1 - \mathtt{L})$, after receiving a $(\mathtt{CR}, a, b, x)$ with $x = rin$ at some time $t_3(\geq t_2 - \mathtt{W_S})$. The CR was sent by the client at some time $t_4(\geq t_3 - \mathtt{L})$. Therefore the client incarnation x was started at some time $t_5(\geq t_4 - \mathtt{W_C})$. Therefore, we have $x(= rin) \geq l_1 - \frac{2\mathtt{L}+\mathtt{W_S}+\mathtt{W_C}}{\alpha}$. And we have the obvious $rin \leq l_1$. Combining these, we have

$$rin \in [l_1 - \frac{2\mathtt{L} + \mathtt{W_S} + \mathtt{W_C}}{\alpha}, \cdots, l_1]$$

which is part (j) of Lemma 1.

Bounds on rin in $\mathtt{DRACK}$ received at client

Let the client receive a $(\mathtt{DRACK}, b, a, sin, rin)$ at time t_1, when $Lin_a(b) = l_1$ and the client is closing. The $\mathtt{DRACK}$ was sent at some time $t_2 (\geq t_1 - \mathtt{L})$ by the server, upon receiving $(\mathtt{DR}, a, b, x, y)$ where $x = rin$ and $y = sin$. The $\mathtt{DR}$ was sent by client at some time $t_3 (\geq t_2 - \mathtt{L})$, by incarnation x that started at at some time $t_4 (\geq t_3 - \mathtt{I})$. Therefore $x \geq l_1 - \frac{2\mathtt{L}+\mathtt{I}}{\alpha}$. And we have $x \leq l_1$. Combining, we have

$$rin \in [l_1 - \frac{2\mathtt{L}+\mathtt{I}}{\alpha}, \cdots, \ l_1]$$

which is part (k) of Lemma 1.

Events of client a concerning server b

$ConnectRequest_a(b)$
ec: $Status_a(b) = \mathtt{closed}$
ac: $Status_a(b) := \mathtt{opening}$; $Lin_a(b) := LinGen_a$

$DisconnectRequest_a(b)$
ec: $Status_a(b) = \mathtt{open}$
ac: $Status_a(b) := \mathtt{closing}$

$Abort_a(b)$
ec: $Status_a(b) \neq \mathtt{closed} \wedge$ ⟨response outstanding for more than $\mathtt{W_C}$ seconds⟩
ac: $Status_a(b) := \mathtt{closed}$; $Lin_a(b) := \mathtt{nil}$; $Din_a(b) := \mathtt{nil}$

$Send\mathtt{CR}_a(b)$
ec: $Status_a(b) = \mathtt{opening}$
ac: $Send(\mathtt{CR}, a, b, Lin_a(b))$

$Send\mathtt{DR}_a(b)$
ec: $Status_a(b) = \mathtt{closing}$
ac: $Send(\mathtt{DR}, a, b, Lin_a(b), Din_a(b))$

$Receive(\mathtt{CRR}, b, a, sin, rin)$
ac: $Status_a(b) = \mathtt{opening} \wedge rin = Lin_a(b) \longrightarrow$ {3-way handshake}
$Status_a(b) := \mathtt{open}$; $Din_a(b) := sin$;
$Send(\mathtt{CRRACK}, a, b, Lin_a(b), Din_a(b))$
$\Box$ $Status_a(b) = \mathtt{open} \wedge rin = Lin_a(b) \wedge sin = Din_a(b) \longrightarrow$ {duplicate $\mathtt{CRR}$}
$Send(\mathtt{CRRACK}, a, b, Lin_a(b), Din_a(b))$
$\Box$ $Status_a(b) = \mathtt{open} \wedge rin = Lin_a(b) \wedge sin > Din_a(b) \longrightarrow$ {server crashed, recovered,
$Send(\mathtt{REJ}, a, b, sin)$; $Status_a(b) := \mathtt{closed}$; responding to old $\mathtt{CR}$}
$Din_a(b) := \mathtt{nil}$; $Lin_a(b) := \mathtt{nil}$
$\Box$ $Status_a(b) \in \{\mathtt{closed}, \mathtt{closing}\} \longrightarrow Send(\mathtt{REJ}, a, b, sin)$

$Receive(\mathtt{CRACK}, b, a, sin, rin)$
ac: $Status_a(b) = \mathtt{opening} \wedge rin = Lin_a(b) \longrightarrow$ {2-way handshake}
$Status_a(b) := \mathtt{open}$; $Din_a(b) := sin$;
$\Box$ $Status_a(b) \in \{\mathtt{open}, \mathtt{closing}, \mathtt{closed}\} \longrightarrow$ ⟨no action⟩

$Receive(\mathtt{REJ}, b, a, rin)$
ac: $Status_a(b) \in \{\mathtt{opening}, \mathtt{closing}\} \wedge rin = Lin_a(b) \longrightarrow$
$Status_a(b) := \mathtt{closed}$; $Din_a(b) := \mathtt{nil}$; $Lin_a(b) := \mathtt{nil}$
$\Box$ $Status_a(b) \in \{\mathtt{open}, \mathtt{closed}\} \longrightarrow$ ⟨no action⟩

$Receive(\mathtt{DRACK}, b, a, sin, rin)$
ac: $Status_a(b) = \mathtt{closing} \wedge rin = Lin_a(b) \wedge sin = Din_a(b) \longrightarrow$
$Status_a(b) := \mathtt{closed}$; $Din_a(b) := \mathtt{nil}$; $Lin_a(b) := \mathtt{nil}$
$\Box$ $Status_a(b) \in \{\mathtt{opening}, \mathtt{open}, \mathtt{closed}\} \longrightarrow$ ⟨no action⟩

Figure 1: Events of Client a in Protocol SC1.

Events of server b concerning client a

$MakeOldCache_b(a)$ {executed between $\mathtt{L} + \mathtt{W_C}$ and $\mathtt{C_S}$ seconds of becoming enabled}

ec: $Cache_b(a) \notin \{\mathtt{nil}, \mathtt{old}\}$

ac: $Din_b(a) := \mathtt{old}$

$Abort_b(a)$

ec: $Status_b(a) \neq \mathtt{closed} \wedge \langle$response outstanding for more than $\mathtt{W_S}$ seconds$\rangle$

ac: $Status_b(a) := \mathtt{closed}$; $Lin_b(a) := \mathtt{nil}$; $Din_b(a) := \mathtt{nil}$

$Send\mathtt{CRR}_b(a)$

ec: $Status_a(b) = \mathtt{opening}$

ac: $Send(\mathtt{CRR}, b, a, Lin_b(a), Din_b(a))$

$Receive(\mathtt{CR}, a, b, sin)$

ac: $Status_b(a) = \mathtt{closed} \wedge \langle$rejecting connnections$\rangle \longrightarrow$
$Send(\mathtt{REJ}, b, a, sin)$;
if $sin > Cache_b(a) \neq \mathtt{nil}$ then $Cache_b(a) := sin$

$\Box$ $Status_b(a) = \mathtt{closed} \wedge \langle$ accept conn$\rangle \wedge Cache_b(a) = \mathtt{nil} \longrightarrow$ {no cache entry
$Status_b(a) := \mathtt{opening}$; $Lin_b(a) := LinGen_b$; $Din_b(a) := sin$ 3-way hndshk}

$\Box$ $Status_b(a) = \mathtt{closed} \wedge \langle$ accept conn$\rangle$
$\wedge\, (Cache_b(a) = \mathtt{old} \vee sin > Cache_b(a) \neq \mathtt{nil}) \longrightarrow$ {cache entry, 2-way hndshk}
$Status_b(a) := \mathtt{open}$; $Lin_b(a) := LinGen_b$; $Din_b(a) := sin$;
$Cache_b(a) := sin$; $Send(\mathtt{CRACK}, b, a, Lin_b(a), Din_b(a))$

$\Box$ $Status_b(a) = \mathtt{opening} \wedge sin > Din_b(a) \longrightarrow$ {previous $Din_b(a)$ value
$Din_b(a) := sin$ was from some old $\mathtt{CR}$}

$\Box$ $Status_b(a) = \mathtt{open}$
$\wedge\, (Cache_b(a) = \mathtt{old} \vee sin > Cache_b(a) \notin \{\mathtt{nil}, \mathtt{old}\}) \longrightarrow$ {client crashed, reconnecting}
if $\langle$willing to reopen$\rangle$ then
$Lin_b(a) := LinGen_b$; $Din_b(a) := sin$; $Cache_b(a) := sin$;
$Send(\mathtt{CRACK}, b, a, Lin_b(a), Din_b(a))$
else $Status_b(a) := \mathtt{closed}$; $Lin_b(a) := \mathtt{nil}$; $Din_b(a) := sin$; $Cache_b(a) := sin$

$\Box$ $Status_b(a) = \mathtt{open} \wedge sin = Cache_b(a) \notin \{\mathtt{nil}, \mathtt{old}\} \longrightarrow$ {duplicate $\mathtt{CR}$}
$Send(\mathtt{CRACK}, b, a, Lin_b(a), Din_b(a))$

$Receive(\mathtt{CRRACK}, a, b, sin, , rin,)$

ac: $Status_b(a) = \mathtt{opening} \wedge sin = Din_b(a) \wedge rin = Lin_b(a) \longrightarrow$
$Status_b(a) := \mathtt{open}$; $Cache_b(a) := Din_b(a)$

$\Box$ $Status_b(a) \in \{\mathtt{open}, \mathtt{closed}\} \longrightarrow \langle$no action$\rangle$

$Receive(\mathtt{DR}, a, b, sin, rin)$

ac: $Status_b(a) = \mathtt{open} \wedge sin = Din_b(a) \wedge rin = Lin_b(a) \longrightarrow$
$Send(\mathtt{DRACK}, b, a, Lin_b(a), Din_b(a))$;
$Status_b(a) := \mathtt{closed}$; $Lin_b(a) := \mathtt{nil}$; $Din_b(a) := \mathtt{nil}$

$\Box$ $Status_b(a) = \mathtt{closed} \longrightarrow Send(\mathtt{DRACK}, b, a, rin, sin)$;

$\Box$ $Status_b(a) = \mathtt{opening} \longrightarrow \langle$no action$\rangle$

$Receive(\mathtt{REJ}, a, b, rin)$

ac: $Status_b(a) = \mathtt{opening} \wedge rin = Lin_b(a) \longrightarrow Status_b(a) := \mathtt{closed}$; $Lin_b(a) := \mathtt{nil}$;

$\Box$ $Status_b(a) \in \{\mathtt{open}, \mathtt{closed}\} \longrightarrow \langle$no action$\rangle$

Figure 2: Events of Server b in Protocol SC1.

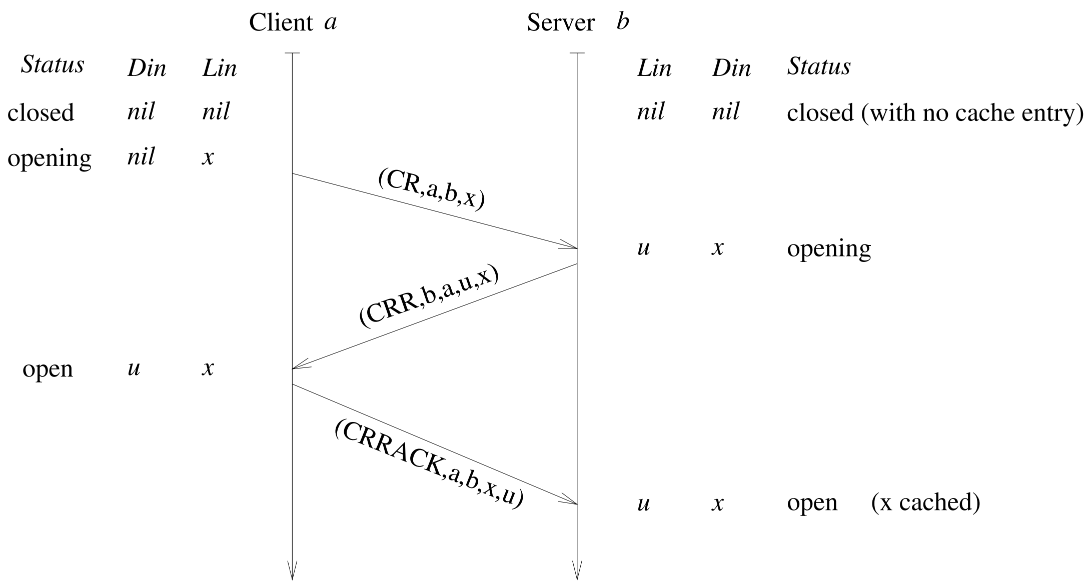

Figure 3: 3-way connection establishment.

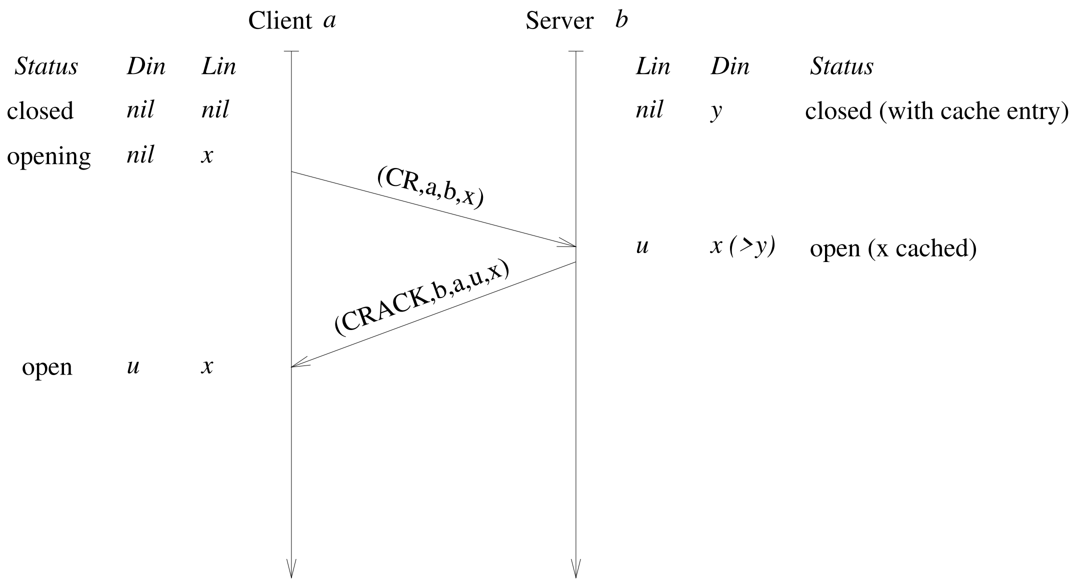

Figure 4: 2-way connection establishment.

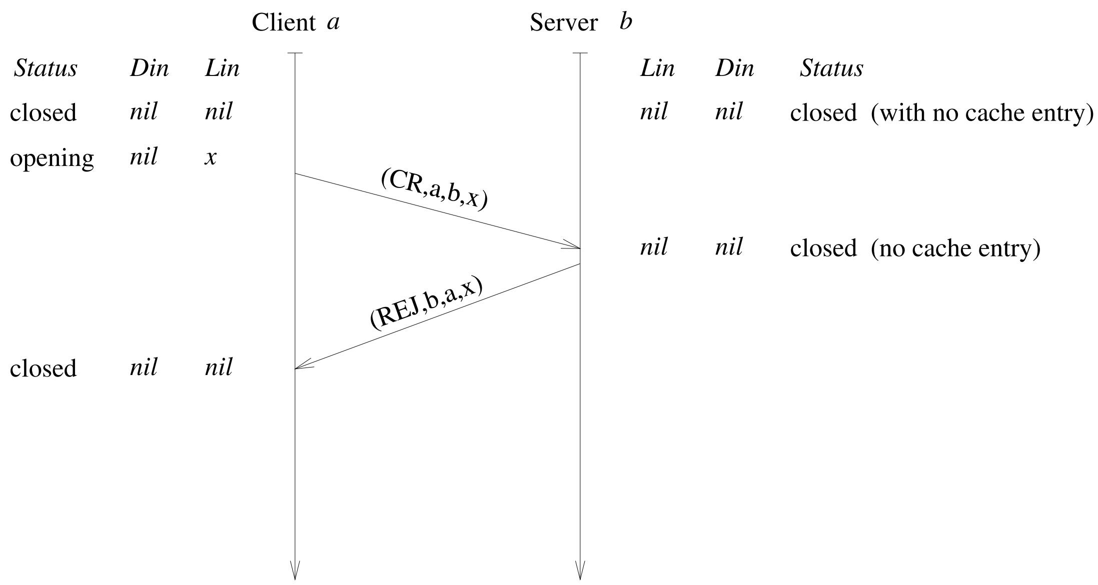

Figure 5: Connection rejection when no cache entry.

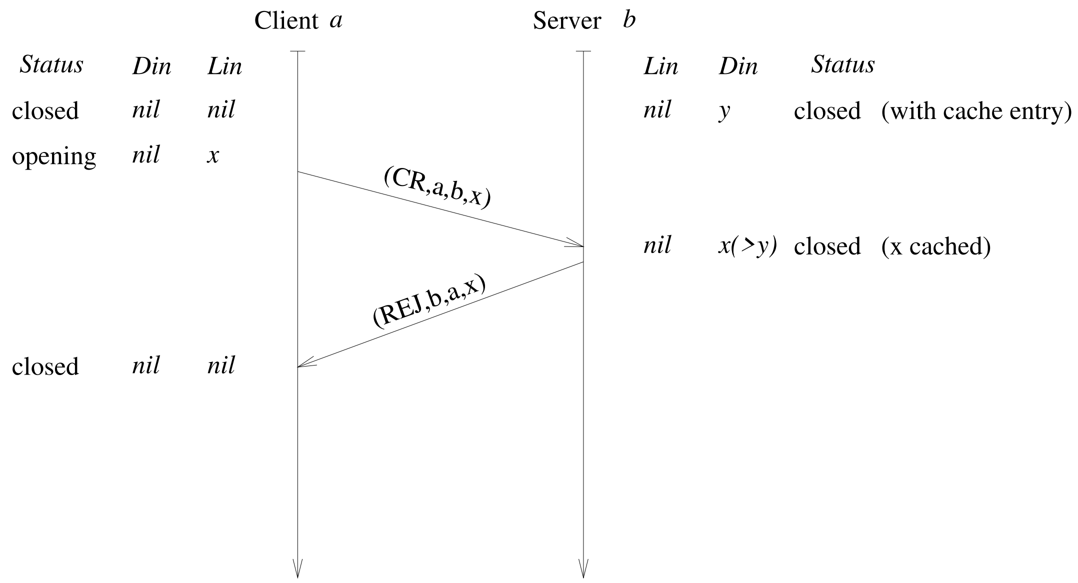

Figure 6: Connection rejection when cache entry present.

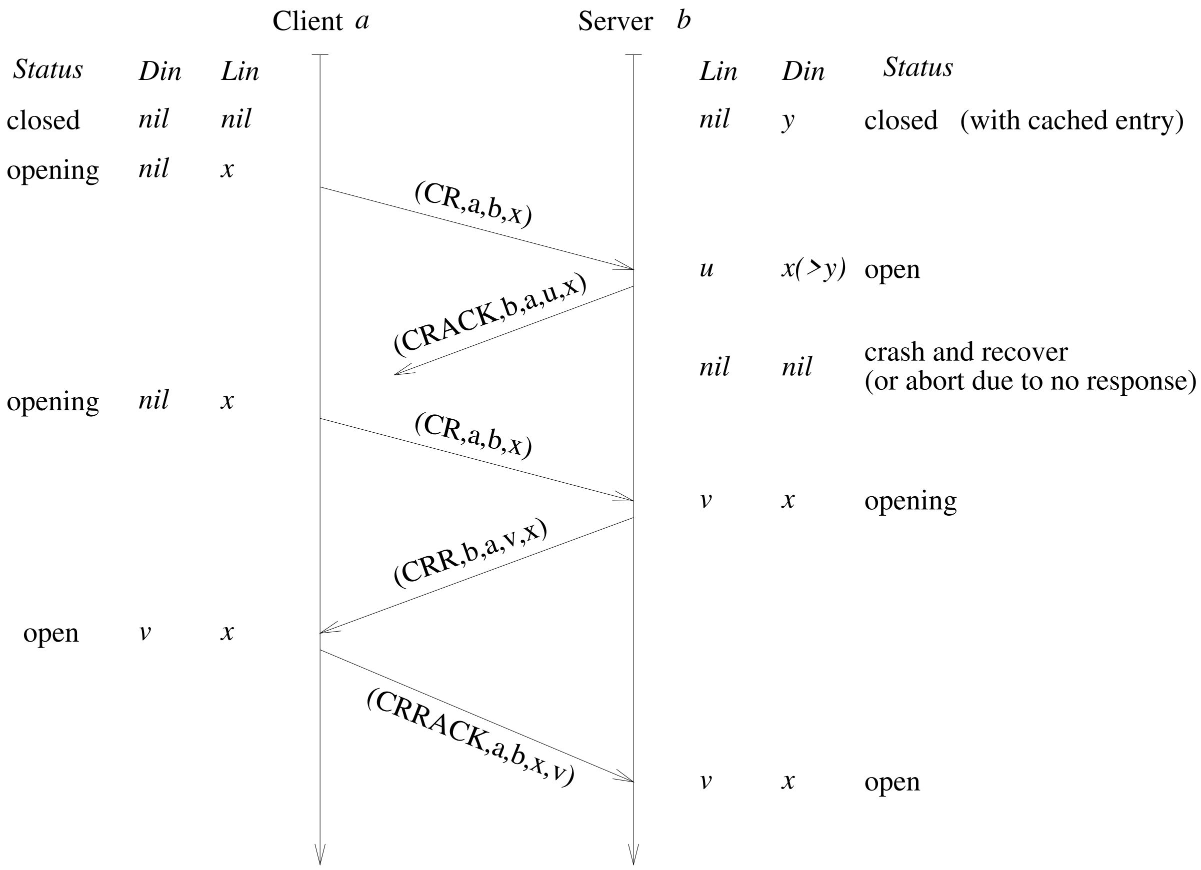

Figure 7: Scenario where incarnations u and v of server b become open to incarnation x of client a. Can be avoided if maximum wait of client is less than minimum recovery delay of server.

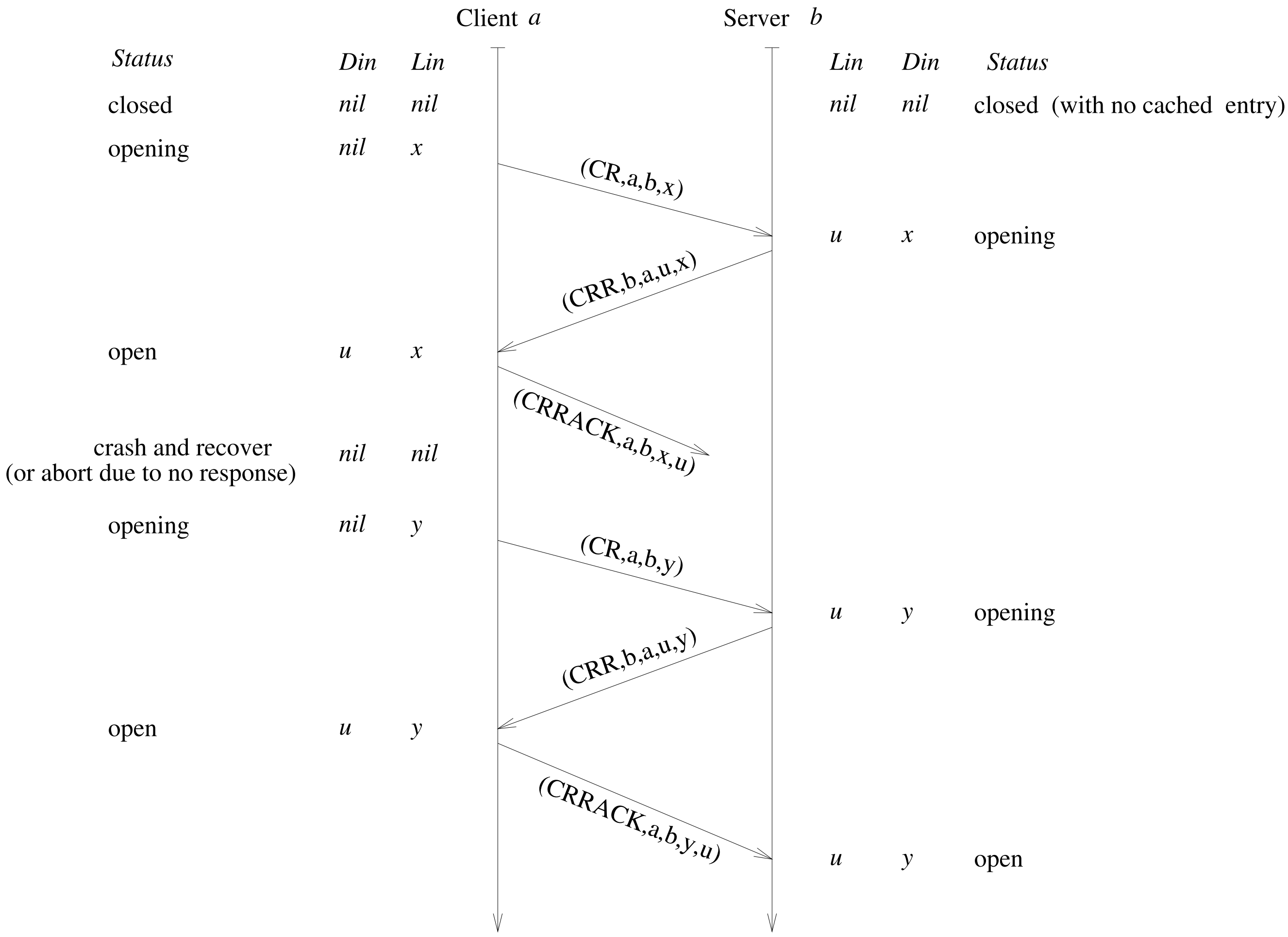

Figure 8: Scenario where incarnations x and y of client a become open to incarnation u of server b. If client crashed and recovered, then this can be avoided if maximum wait of server is less than minimum recovery delay of client. If client aborted due to no response, then this can be avoided if maximum wait of server is less than minimum wait delay of client.