A Timer-Based Connection Management Protocol with Synchronized Clocks and its Verification

Ernst W. Biersack and David C. Feldmeier*

July 7, 1992

Abstract

Connection management ensures that each message of a connection is accepted at-most-once and that a connection is not closed before all messages and acknowledgements are received. We introduce a connection management protocol that is based on timers, the use of approximately synchronized clocks and unique connection identifiers. Synchronized clocks support accurate end-to-end lifetime enforcement and, in combination with unique connection identifiers, simplify duplicate detection after recovery from a failure. Our protocol has the following desirable characteristics: (1) no extra messages for connection setup or release, (2) no connection setup latency, (3) fast connection state release, and (4) immediate resumption of communication after an endsystem failure.

1 Introduction

A connection is an association between two users that exchange messages. A connection management protocol manages the state information that is needed to ensure that any message received is accepted no more than once. The connection management protocol keeps the state information until all messages are received (graceful close)[1]. Connection management is a necessary building block in any protocol that provides reliable transfer of data. It also can be used to implement a remote procedure call protocol (RPC) with at-most-once semantics.

*E. W. Biersack: Institut EURECOM, 2229 route des Cretes, Sophia Antipolis, 06560 Valbonne, FRANCE, e-mail:erbi@eurecom.fr D. C. Feldmeier: Bellcore, Morristown, NJ 07962-1910, USA, e-mail: dcf@bellcore.com

[1]We consider the allocation of resources such as bandwidth or buffer space to be separate from connection management.

A connection management protocol keeps information about the state of a connection in a *connection record* [WATS 81b]. For each connection there exists a connection record at the sender and receiver that contains, among other things, information about the identifiers used, the lifetime of the identifiers, and the messages received. The connection management system allocates a connection record at setup time, updates its contents during the lifetime of a connection, and releases the connection record when a connection terminates. The various connection management protocols differ in the way they setup and release a connection record.

The mechanisms available for implementing a connection management protocol are: (1) handshake [DOD 83], (2) timers [FLET 78, WATS 81a, WATS 83], and (3) (bounded) unique identifiers [CHER 88]. In a handshake-based connection setup, the sender and receiver must exchange setup messages before the actual data are transmitted to ensure that the setup message is not a duplicate. In a timer-based protocol, the receiver keeps information on the messages "recently" received and uses time to determine whether any message is a duplicate. Unique identifiers are used to distinguish connections. A unique connection identifier (CID) is assigned to every connection and all messages of a particular connection carry the same CID. The assignment of CIDs can be done by each endsystem independently: The CID is partitioned in a part uniquely identifies the endsystem and another part that uniquely identifies each connection within that endsystem. Neither handshake nor unique identifiers alone are sufficient for implementing a connection management system [WATS 89]. Both mechanisms must be complemented with timers. Handshake-based schemes need timers for graceful close and a scheme based on unique identifiers needs timers to determine when to reuse identifiers. Timers alone however, are sufficient to implement connection management.

Handshake-based and timer-based protocols trade off the exchange of additional messages and setup latency (handshake-based) for slightly later connection record release (timer-based). RPC and transactions usually transmit only a small amount of data that may even fit into a single message, in which case the overhead of a handshake-based scheme is high. An explicit connection setup using a handshake protocol introduces a latency of at least one round-trip time. In high bandwidth-delay product networks, the time to complete the handshake can be larger than the time to transmit the data. This disadvantage will become more pronounced as the transmission speeds will further increase while the round-trip times, which are largely determined by the signal propagation delay, will remain the same [KLEI 92]. With transmission speeds increasing and the round-trip times staying constant, many file transfers soon may have less than a round-trip time worth of data to send[2]. Timer-based connection management schemes do not introduce a setup latency of one round-trip time. However, they release the connection records later than handshake-based schemes and need, on the average, more memory to keep the

[2]If the setup latency is 100 msec and the bandwidth 150 Mbits/sec, 15 Mbits of data can be transmitted during the setup time.

connection records. With the dramatic increase in integration densities for memories, the higher memory requirement will not be a problem.

This paper describes a connection management protocol, referred to as **C**onnection **M**anagement with **S**ynchronized **C**locks, or *CMSC* for short, that uses timers and unique CIDs. *CMSC* requires that the clocks in the endsystems are approximately synchronized. *CMSC* operates correctly as long as there exists a known ϵ that bounds the clock skew. Synchronizing clocks is not hard. Mills [MILL 91] describes a synchronization protocol, called NTP, that synchronizes clocks in a large heterogeneous internetwork. NTP guarantees with high probability that the worst case clock skew is in the order of tens of milliseconds. NTP is robust in the face of failures and has a moderate communications overhead. A more accurate clock synchronization can be achieved using the time signal received from the global positioning system (GPS) [WELL 90].

As compared to connection management using three-way handshake, *CMSC* allows user data to be transmitted with the first message exchanged. It requires no separate control messages for connection setup or tear-down. When no connection exists, as few as two messages are sufficient to reliably exchange data, as compared to at least six messages for a three-way handshake[3]. Synchronized clocks and unique CIDs make it possible for the receiver and sender to resume communication immediately after a crash without waiting for old messages to have their lifetime expire (see Section 6). The immediate resumption of communication is important when the time to reboot an endsystem is smaller than the message lifetime.

The principles of timer-based connection management were explored first by Watson and implemented in the Delta-t transport protocol [FLET 78, WATS 83]. *CMSC* adopts these principles and extends them: *CMSC* introduces ϵ-synchronized clocks for end-to-end lifetime enforcement and uses unique CIDs. The use of synchronized clocks was suggested independently by Cheriton [CHER 89] and by Liskov, Shrira and Wroclwaski [LISK 91]. Cheriton proposes synchronized clocks for end-to-end lifetime enforcement at the transport protocol level. He sees as benefits that (1) the correct operation of the transport protocol is independent of the operation of intermediate nodes, (2) intermediate nodes are freed from the burden of lifetime enforcement, and (3) the time-stamping information carried in the messages can be used to accurately estimate the transit delay. Liskov et al. use synchronized clocks for an at-most-once message delivery protocol called SCMP. SCMP detects only duplicates of the first message sent, which is one sub-problem connection management must solve. Since SCMP cannot cope with reordering of messages, it can be applied only to the first message received and not to a sequence of messages. SCMP does not address the issues of graceful close, state release at the sender, and safe reuse of identifiers, all of which are dealt with by *CMSC*.

[3]When using a three-way handshake, data could be transmitted with the setup message. However, the receiver cannot deliver the data to the application until it is sure that the setup message received is not a duplicate. The receiver cannot decide this until the three-way handshake is completed [WATS 81a].

This paper is organized as follows. Section 2 describes the model and the principles of timer-based connection management. Section 3 and 4 describe the operation of *CMSC*. Section 5 proves the correctness of *CMSC*. Section 6 extends *CMSC* to deal with endsystem failures and Section 7 summarizes the features of *CMSC*.

2 Timer-Based Connection Management

2.1 Model and Assumptions

We assume that communication takes place between users that reside in different endsystems, which are connected through a network. Users communicate by exchanging *messages*. Messages can experience corruption, arbitrary transmission delays, loss, misordering, and duplication. The transport protocol resides in the endsystem and provides a reliable message transfer to the users. It uniquely identifies each message by its *connection identifier* and *sequence number*. The connection identifier determines which connection the message is part of and the sequence number denotes the position of a message within the messages exchanged across a connection. Since the identifier spaces of the connection identifier and sequence number are finite, the transport protocol must reuse these identifiers. To make this reuse safe, each message carries an *expiration time* that limits the lifetime of the sequence number. There are data and acknowledgment messages. A *data* message carries user data. An *acknowledgment* message carries the sequence number of the data message up to which all data messages were received correctly.

An endsystem can fail, in which case it will stop operating. We do not consider Byzantine failures. Endsystems may be equipped with some stable storage to maintain state despite failures. The endsystems also have ϵ-synchronized clocks. Let I denote the set of endsystems. Every endsystem $i \in I$ has a monotonic clock $\mathcal{C}_i$ that is kept ϵ-synchronized by executing a clock synchronization protocol [MILL 91]. Let $\mathcal{T}_{\mathcal{C}_i}$ denote the current time that is obtained by reading the local clock $\mathcal{C}_i$. A set $\{\mathcal{C}_i \mid i \in I\}$ of clocks is *ϵ-synchronized*, iff at any time $\forall j, k \in I : |\mathcal{T}_{\mathcal{C}_j} - \mathcal{T}_{\mathcal{C}_k}| \leq \epsilon$. The *clock skew* ϵ is typically no more than a few tens of milliseconds. For *CMSC* it is sufficient to have the clocks *internally* synchronized, i.e., the clocks are synchronized with respect to each other but not necessarily with respect to any other system.

We assume that the number of available identifiers and the amount of storage for keeping the state information are finite. Also, the field for the expiration time is sufficiently large to prevent a wrap-around during the lifetime of the network.

2.2 Bounds on the Lifetime of Identifiers

When the identifier space and storage are finite, connection management must allow for a safe reuse of identifiers and determine when it is safe to release a connection record. Usually, it is not feasible to keep all connection records in stable storage.

In this case connection management must cope with partial or complete loss of the connection records due to failures of the endsystems.

The identifiers we are concerned with are sequence numbers and unique connection identifiers (CIDs). To make the reuse of identifiers safe, the lifetime of the identifiers is limited. An identifier must not be reused while its lifetime has not yet expired. If LT is the lifetime of a sequence number and R the rate at which sequence numbers are used, the size $|SN|$ of the sequence number space must be $|SN| > LT \cdot R$ to make sequence number reuse safe. A CID must not be released while there exists a connection record associated with this CID at either end or there are messages carrying this CID whose lifetime has not yet expired. Lifetime enforcement discards messages with sequence numbers whose lifetime has expired. Lifetime enforcement is usually performed by a separate protocol and on a hop-by-hop basis [SLOA 79, SLOA 83]. *CMSC* takes advantage of the fact the clocks are ϵ-synchronized which allows end-to-end lifetime enforcement at the receiver. The sender puts the expiration time of the sequence number in the header of a message and the receiver discards a message if its expiration time is less than the current time of the receiver's clock.

2.3 Phases

A connection management protocol distinguishes an open, transfer, and close phase. In timer-based connection management protocols, the open and close phases do not require the exchange of separate messages. The open and transfer phases are combined and the sender does not have to wait for the open phase to complete but can start transmitting data immediately. The close occurs implicitly.

- Open: To set up a connection the sender sets a bit (FIRST = *on*) in the header of the first message and the sender and receiver establish a connection record at both ends. The issues we will address are: Lifetime of identifiers and detection of duplicate setup messages.

- Transfer: During the transfer phase, the sender transmits data messages and the receiver sends acknowledgment messages that indicate which messages have been received. The issues we will address are: How to detect duplicate messages, how to detect that a connection is dead — because either a path is down or the receiver has crashed — and when to reuse a sequence number.

- Close: After the sender stops transmitting data, the sender and receiver will implicitly close the connection by releasing their connection records and releasing the CIDs used. The issues we will address are: How to close a connection such that all data and acknowledgment messages transmitted can be received and when to release the CIDs for reuse.

3 Informal Description of *CMSC*

We describe the connection management for a simplex connection where the sender transmits data messages and the receiver transmits acknowledgment messages. Each simplex connection has a data channel from the sender to the receiver and an acknowledgment channel from the receiver to the sender. Note that all timers are set to the absolute time values at which they will expire relative to the local clock of their endsystem.

CMSC uses the following fields in the message header that uniquely identify each message, determine the lifetime of a message, and indicate if a message is the first one.

Texpire: Time at which the lifetime of the sequence number in this message will expire.

Cid: Connection identifier of this message.

Type: Message is either a data or an acknowledgment message.

Sn: Sequence number of this message.

Lt: The lifetime of the identifiers for this connection.

Wrap: Ratio of $|SN|$ over the maximum transmission rate R.

First: A single bit set to *on* or *off*. The bit indicates whether this message is the first message of a connection. If it is set to *on* it indicates a request for a new connection.

The field *Texpire* contains for a data message the expiration time of its sequence number and for an acknowledgment message it contains the expiration time of the sequence number in the cumulative acknowledgment. Whenever a data message is generated, its expiration time is set to $Texpire \leftarrow \mathcal{T}_{\mathcal{C}_S} + LT$, where $\mathcal{T}_{\mathcal{C}_S}$ is the current time of the sender's clock and LT the lifetime of the sequence number. A message that is retransmitted retains its original expiration time. A message with an expiration time smaller than the current time has its *lifetime expired* and is considered *invalid*. The sender does not retransmit a message whose lifetime has expired and the receiver will discard such a message when received.

Each connection in *CMSC* is uniquely identified by its CID. The benefits of using a CID are that a single sender-receiver pair can have several connections simultaneously, with each connection distinguished by its connection identifier. Since CIDs also can be made unique across endsystem failures (see Section 6), it is unnecessary for the sender to delay its communication after a crash until all messages prior to the crash have exceeded their lifetime.

The *CMSC* protocol uses timers to ensure liveness (a message transmitted often enough eventually will be accepted) and safety (a message will not be accepted more than once). As we will see, the *initial values* of the timers, which are the values these timers are initialized with, depend on the lifetime LT of the sequence numbers. *CMSC* explicitly bounds the lifetime of sequence numbers. The lifetime LT is connection-specific and should be large enough to allow for multiple retransmissions and for the transmission of the acknowledgment as well. Typically, LT will be much larger than ϵ. By making LT the lifetime of a sequence number, LT will also bound the time that a sender will spend retransmitting a given message, the lifetime of the acknowledgment message that acknowledges a given message, and the maximum interval between two successive transmissions.

In *CMSC* the receiver performs lifetime enforcement and rejects any message that has exceeded its lifetime. In contrast, Delta-t [WATS 81b] relies on lifetime enforcement by the network. Delta-t must divide the total lifetime LT into the time during which retransmissions are allowed, the maximum packet lifetime of a message in the network, and the time that elapses at the receiver between receiving and acknowledging a message. In Delta-t, a transmission will fail and the message is dropped when the estimate for the maximum packet lifetime is smaller than the actual transit time. In *CMSC* this cannot happen as long as the total lifetime LT is larger than the transmission time of the message.

In the following we will discuss each of the five timers used in *CMSC*, their purpose, and how they are set and reset.

3.1 Liveness

The receiver and the sender must assure that they can receive any valid message. At the sender there is a timer T_{s1} that does not expire while there exists a sequence number whose lifetime has not yet expired. Every time a new data message X with expiration time $X.Texpire$ $(= \mathcal{T}_{\mathcal{C}_S} + LT)$ is transmitted, the sender sets $T_{s1} \leftarrow X.Texpire$. The connection record of the sender is released when T_{s1} expires. Because the expiration time of an acknowledgment message is the expiration time of the sequence number in the cumulative acknowledgment, the sender will wait long enough to receive any valid acknowledgment.

The receiver needs a timer T_{r1} similar to T_{s1} to ensure that it will not close a connection while the transmission of new data messages is ongoing. The value for T_{r1} is derived from the expiration times of the messages received. The receiver knows the expiration times of all data messages received and computes $\max_{Q \in messages\ received} Q.Texpire$. The receiver also knows that the maximum difference between the expiration times of two consecutive[4] data messages is LT (for a proof see Lemma 1). Therefore, the expiration time of the message that is consecutive to message M with $M.Texpire = \max_{Q \in messages\ received} Q.Texpire$ cannot exceed $M.Texpire + LT + \epsilon$ as measured

[4]Two messages are *consecutive* if their sequence numbers are adjacent.

on the receiver's clock, which can differ from the sender's clock by up to ϵ. Every time a non-duplicate data message X is accepted, the receiver sets $T_{r1} \leftarrow \max_{Q \in messages\ received} Q.Texpire + LT + \epsilon$ and updates the connection record to remember that message X has been received. When T_{r1} expires, the connection record of the receiver is released. Timer T_{r1} also ensures that all duplicates of data messages can be detected, since the connection record is not released before the lifetimes of all received messages have expired.

3.2 Safety

In order to assure at-most-once semantics, the sender and receiver must avoid ambiguities as to what connection a message is part of and where its position is within the data of this connection.

Since the sequence number field of a message has a finite size, the sequence numbers are bounded and wrap around. To avoid sequence number ambiguities, the sender knows must control the maximum the rate R at which sequence numbers are used. If $|SN|$ is the total number of sequence numbers, the lifetime LT must be chosen such that $LT < \frac{|SN|}{R}$ to make sure a sequence number can be safely reused. Let X denote the message with sequence number x and $\widehat{X}$ the message with sequence number $(x+|SN|) \bmod |SN|$. The sender has a timer T_{s2} to ensure the safe reuse of a sequence number and to suspend the transmission in the case that the path between sender and receiver is disrupted or the receiver is dead. T_{s2} prevents the sender from generating message $\widehat{X}$ before message X has been acknowledged. T_{s2} is set every time an acknowledgment ACK is received: $T_{s2} \leftarrow ACK.Texpire - LT + \frac{|SN|}{R}$. When T_{s2} expires the transmission of new data messages is suspended[5]. If a new acknowledgment arrives before T_{s2} expires, T_{s2} will be reset and the sender can resume transmitting new messages.

To avoid ambiguities in the use of connection identifiers, the sender must not release a connection identifier CID while the receiver has a connection record associated with this CID. The sender uses a timer T_{s3} to control the release of a data channel CID. Since T_{r1} determines the release of the connection record at the receiver, we require $T_{s3} \geq T_{r1}$ as measured on the sender's clock. The sender knows that $T_{r1} \leq X.Texpire + LT + \epsilon$, where X is the data message with the greatest expiration time that was transmitted by the sender. Every time a new (non-duplicate)

[5]If $\frac{|SN|}{R}$ is large compared to LT the sender will be allowed to transmit new messages for a long time after the last acknowledgment message was received. Such an "optimistic" strategy makes sense if the sender assumes that the path between the sender and receiver is only temporarily disrupted. However, if the disruption persists the sender will waste bandwidth by transmitting new messages until it will eventually abort. To limit this waste, it might be sensible to suspended sending earlier. If the sender sets $T_{s2} \leftarrow ACK.Texpire + \min(LT, \frac{|SN|}{R} - LT) + \epsilon$ it will be suspended, if no new acknowledgment arrives, no later than when the lifetime of the next message the be acknowledged expires. Such a change in the initial value of T_{s2} will also affect the initial value of T_{r2}.

message X is transmitted, the sender sets: $T_{s3} \leftarrow X.Texpire + LT + 2\epsilon$. When T_{s3} expires, the CID is released.

The receiver has a timer T_{r2} to control the release of the acknowledgment channel CID. T_{r2} makes sure that the receiver does not release a CID while the sender has a connection record associated with this CID. Since T_{s1} determines the release of the sender's connection record, we require that $T_{s1} \leq T_{r2}$ as measured on the receiver's clock. The worst case scenario for T_{r2} is one in which the sender keeps transmitting new data messages and resetting T_{s1} while these data messages never arrive at the receiver. The sender will eventually be suspended when T_{s2} expires. Let Y denote the message with the greatest expiration time that was received by the receiver. The receiver knows that $T_{s2} \leq Y.Texpire - LT + \frac{|SN|}{R} + \epsilon$ as measured on its clock and that the sender can generate new messages until T_{s2} expires. We have $T_{s1} \leq T_{s2} + LT \leq Y.Texpire + \frac{|SN|}{R} + \epsilon$ as measured on the receiver's clock. Therefore, to satisfy $T_{s1} \leq T_{r2}$ as measured on the receiver's clock, we can choose $T_{r2} \geq Y.Texpire + \frac{|SN|}{R} + \epsilon$. Whenever a non-duplicate data message is received, the receiver sets $T_{r2} \leftarrow Y.Texpire + \frac{|SN|}{R} + \epsilon$. When T_{r2} expires, the acknowledgment channel CID is released.

T_{s1} and T_{s3} at the sender and T_{r1} and T_{r2} at the receiver are always set simultaneously. Since $T_{s1} < T_{s3}$ and $T_{r1} < T_{r2}$ the sender can instead set T_{s3} when T_{s1} expires to $T_{s3} \leftarrow T_{s1} + LT$ and the receiver can set T_{r2} when T_{r1} expires to $T_{r2} \leftarrow T_{r1} - LT + \frac{|SN|}{R}$. This modification increases the efficiency by reducing the number of timers active any time without affecting the behavior of the protocol. Table 1 lists the timers used per connection.

Timers	**Where**	**Function**	**Set when**	**Set to**
T_{s1}	Sender	liveness	new message is transmitted	$\mathcal{T}_{\mathcal{C}_S} + LT$
T_{s2}	Sender	safe reuse of sequence number	ACK is received	$\max(T_{s2}, ACK.Texpire - LT + \frac{\lfloor SN \rfloor}{R})$
T_{s3}	Sender	safe reuse of CID	T_{s1} expires	$T_{s1} + LT + 2 \cdot \epsilon$
T_{r1}	Receiver	liveness and safety	new valid message DT is received	$\max(T_{r1}, DT.Texpire + LT + \epsilon)$
T_{r2}	Receiver	safe reuse of CID	T_{r1} expires	$T_{r1} - LT + \frac{\lfloor SN \rfloor}{R}$

Table 1: Timers per connection.

4 Formal Description of *CMSC*

The data and acknowledgment messages exchanged between the sender and the receiver have the following fields:

Texpire	/* expiration time */
Wrap	/* $\frac{\lvert SN \rvert}{R}$, i. e., the minimum time to cycle through the sequence number space */
Cid	/* connection identifier */
Type	/* message type: DATA or ACK */
Sn	/* sequence number */
Lt	/* lifetime */
First	/* indicates request for a new connection */
Data	/* user data */

The sequence number field Sn is only defined for data messages.
We use the following notation:
$\mathcal{T}_{\mathcal{C}_S}$ denotes the clock at the sender and $\mathcal{T}_{\mathcal{C}_R}$ denotes the clock at the receiver.
$\oplus$ denotes addition modulo $|SN|$.
$||$ denotes the concatenation operator.
undefined denotes a value different from any other value used.

4.1 Sender

The connection record CR_S at the sender contains the following components for each connection:

LT	/* lifetime */
R	/* maximum transmission rate */
WRAP	/* $\frac{\lvert SN \rvert}{R}$ */
CID	/* connection identifier of data channel */
T_{s1}	/* timer to ensure graceful close */
T_{s2}	/* timer to ensure safety */
T_{s3}	/* timer to ensure correct reuse of the data channel CID */
ACK_CID	/* CID of acknowledgment channel */
NSN	/* next message sequence number */

The sender uses the following auxiliary functions:

get_free_cid()
  Return a CID that is currently not in use
get_LT ()
  Return the lifetime LT of a particular connection based on peak transmission rate such that: $0 < LT < WRAP$

Connection setup:

```
$CID \leftarrow get\_free\_cid()$
$R \leftarrow max.\ transmission\ rate$
$WRAP \leftarrow \frac{|SN|}{R}$
```

```
  allocate a connection record $CR_S[CID]$
  $LT \leftarrow get\_LT()$ /* max. lifetime of sequence numbers */
  $NSN \leftarrow 0$ /* initialize sequence number */
```

Send data message *DT* of connection CID:

```
  if DT is transmitted for the first time /* generate message header */
    then $DT.Cid \leftarrow CID$
         $DT.Texpire \leftarrow \mathcal{T}_{\mathcal{C}_S} + LT$
         $DT.Wrap \leftarrow WRAP$
         $DT.Lt \leftarrow LT$
         $DT.Type \leftarrow DATA$
         if $DT$ is the first message of connection CID to be sent
           then $DT.First \leftarrow on$ /* request connection setup */
                $T_{s2} \leftarrow \mathcal{T}_{\mathcal{C}_S} + WRAP$
           else $DT.First \leftarrow off$
         endif
         $NSN \leftarrow NSN \oplus 1$
         $DT.Sn \leftarrow NSN$
         $T_{s1} \leftarrow \mathcal{T}_{\mathcal{C}_S} + LT$ /* (re-)set send timer */
  endif
  $DT.Data \leftarrow data\ part\ of\ message$
  send DT
```

***ACK* is received that acknowledges the data messages received on connection CID:**

```
  if $ACK.Texpire \geq \mathcal{T}_{\mathcal{C}_S}$ and $ACK.First = on$ and $ACK\_CID = undefined$
    then /* setup message for Ack channel */
         $ACK\_CID \leftarrow ACK.Cid$
  endif
  if $ACK.Texpire \geq \mathcal{T}_{\mathcal{C}_S}$ and $ACK\_CID = ACK.Cid$
    then pass ACK on to error control module
         $T_{s2} \leftarrow \max(T_{s2}, ACK.Texpire - LT + WRAP)$ /* reset whenever ACK arrives */
  endif
  else discard ACK
```

T_{s1} for connection identifier CID expires:

```
  $T_{s3} \leftarrow T_{s1} + LT + 2 \cdot \epsilon$
  $T_{s1} \leftarrow \infty$
  $T_{s2} \leftarrow \infty$
  release connection record $CR_S[CID]$
  send disconnect notification to application
```

T_{s2} for connection identifier CID expires:

```
  stop transmitting new data messages
```

```
$T_{s3}$ for connection identifier CID expires:
  $T_{s3} \leftarrow \infty$
  make CID available for reuse
```

4.2 Receiver

The receiver uses a data structure *msg_entry*:

```
struct msg_entry {  /* contains information about a received message */
      m_sn          /* sequence number of message */
      m_expire      /* time when the lifetime of sequence number expires */
}
```

The connection record CR_R of the receiver contains the following components for each connection:

```
  $T_{r1}$          /* timer to ensure duplicate detection and graceful close */
  $T_{r2}$          /* timer to ensure correct reuse of the CID of the ACK channel */
  CID          /* CID of data channel */
  ACK_CID /* CID of acknowledgment channel */
  LT           /* lifetime of the sequence numbers of the data channel */
  WRAP        /* $\frac{|SN|}{R}$ as chosen by sender */
  MSGS_RECEIVED /* list of elements of type msg_entry */
  CUM_SN /* sequence number up to which all messages have been received */
```

The receiver uses the following auxiliary functions:

```
  update_msgs(cid, sn, expire)
      Add a new entry new_msg of type msg_entry to the list
      $CR_R[cid].MSGS\_RECEIVED$ and set
          new_msg.m_sn ← sn
          new_msg.m_expire ← expire
  cum_ack(cid) /* return cumulative acknowledgment */
      Return the sequence number of the message up to which all messages on
      CID cid have been received
  expire_cum_ack(cid, sn) /* return lifetime of cumulative acknowledgment */
      Return $(M.m\_expire \mid M \ in \ CR_R[cid].MSGS\_RECEIVED$ and $M.m\_sn = sn)$
  is_duplicate(cid, sn)
      Return true if there exists an entry M in $CR_R[cid].MSGS\_RECEIVED$ with
      $(M.m\_sn = sn)$ and $(M.m\_expire > \mathcal{T}_{\mathcal{C}_R})$.
      Return false otherwise.
```

```
Message $DT$ is received:
  if $DT.Texpire < \mathcal{T}_{\mathcal{C}_R}$ /* check lifetime */
```

```
    then discard DT
            goto END_received
  endif
  if $DT.First = off$ and $CR_R[DT.Cid]$ does not exist
    then discard DT
            goto END_received
  endif
  if $DT.First = on$ and $CR_R[DT.Cid]$ does not exist
    then /* create connection record and establish ACK channel */
            allocate a connection record $CR_R[DT.Cid]$
            $CID \leftarrow DT.Cid$
            $ACK\_CID \leftarrow get\_free\_cid()$
            $LT \leftarrow DT.Lt$
            $WRAP \leftarrow DT.Wrap$
            $T_{r1} \leftarrow DT.Texpire + LT + \epsilon$
  endif
  if is_duplicate(DT.Cid, DT.Sn)
    then discard DT
            goto END_received
  endif
  $T_{r1} \leftarrow \max(T_{r1},\ DT.Texpire + LT + \epsilon)$
  update_msgs(DT.Cid, DT.Sn, DT.Lt)
  store DT.Data
  END_received
```

Periodically for each existing connection record $CR_R[CID]$:

```
  store in $CUM\_SN$ the value of the sequence number up to which
        all data messages were received
  remove all entries M from $CR_R[CID].MSGS\_RECEIVED$ with
        $M.m\_sn < CUM\_SN$ or $M.m\_expire < \mathcal{T}_{C_R}$
```

Receiver generates an acknowledgment ACK for connection CID:

```
  $ACK.Data \leftarrow cum\_ack(CID) \parallel CID$
  $ACK.Cid \leftarrow ACK\_CID$
  $ACK.Type \leftarrow ACK$
  $ACK.Texpire \leftarrow expire\_cum\_ack(CID, cum\_ack(CID))$
  if ACK is first acknowledgment to be sent
    then $ACK.First \leftarrow on$
    else $ACK.First \leftarrow off$
  endif
  send ACK
```

```
$T_{r1}$ for connection identifier CID expires:
  $T_{r2} \leftarrow T_{r1} - LT + WRAP$
  $T_{r1} \leftarrow \infty$
  release connection record $CR_R[CID]$

$T_{r2}$ for connection identifier CID expires:
  $T_{r2} \leftarrow \infty$
  make CID available for reuse
```

5 Protocol Correctness

The correctness of *CMSC* can be established by proving that it meets the safety and liveness properties:

- Its safety is shown by proving that at any time there are never two messages with (i) the same *(cid,sn)*, (ii) different data, and (iii) un-expired lifetimes, and that no duplicate messages will be accepted.

- Its liveness is shown by proving that once a connection is setup the receiver will accept any valid message whose lifetime has not yet expired. The receiver will acknowledge the received data and never terminate while there exists a messages whose lifetime has not yet expired. Also, the transmitter of the data messages, i.e. the receiver of the acknowledgment messages, will never terminate while there exist acknowledgments whose lifetime has not expired.

Assumption 1 $\mathcal{T}_{\mathcal{C}_S}$ *and* $\mathcal{T}_{\mathcal{C}_R}$ *are monotonically increasing.*

Assumption 2 $|\mathcal{T}_{\mathcal{C}_S} - \mathcal{T}_{\mathcal{C}_R}| \leq \epsilon$

The following parameters are used in the code.

Parameter 1 *The number of available sequence numbers is* $|SN|$

Parameter 2 *The maximum rate at which sequence numbers are allocated is* R.

Parameter 3 *The maximum lifetime* LT *of a connection is* $LT < \frac{|SN|}{R}$.

The following lemma states a property about the sequence numbers that will be used in our proofs.

Lemma 1 *The maximum time interval as measured on the sender's clock between the generation of the sequence numbers* $X.Sn$ *and* $Y.Sn$ *of any two consecutive messages* X *and* Y *on a single connection is* LT.

Proof: Let t_X denote the time at which message X is generated. Messages (except for the very first message) can be sent on a connection only if $T_{s1} \geq \mathcal{T}_{\mathcal{C}_S}$. Thus the generation time t_Y of message Y must satisfy

$$t_Y \leq T_{s1}$$

Since T_{s1} was reset the last time at t_X,

$$t_Y \leq t_X + LT = T_{s1}$$

Therefore,

$$t_Y - t_X \leq LT$$

□

5.1 Safety

We first prove the safety for the data channel and then for the acknowledgment channel.

5.1.1 Safety of the Data Channel

First we shall discuss a system that has no failures. We want to show that the system never accepts a message incorrectly. We must be sure that the connection state is not left over from the past and that no old messages are accepted as part of a new connection. If connection state exists, then we have the sequence numbers of the messages that have been accepted. As long as there is no ambiguity about which data a sequence number refers to, then we never accept duplicates. We can avoid sequence number ambiguity if the receiver detects the loss of sequence number x before sequence number $x + n \cdot |SN|$ arrives, where $n = \{1, 2, 3, \ldots\}$.

We must show that duplicate messages are detected and that messages of one connection are not mistaken for messages of another connection.

Theorem 1 *A setup message (First = on) is accepted at-most once.*

Proof: After a setup message X is received the first time:

$$X.Texpire < X.Texpire + LT + \epsilon = T_{r1}$$

Duplicates of X will be detected and rejected since the connection record will be held until the lifetime of X expires.

Theorem 2 *There will never be two messages arriving at the receiver with the same CID and un-expired lifetimes that belong to different connections.*

Proof: To prove the theorem we must show
(i) The sender does not release a data channel CID until the receiver has dropped its connection state, i.e., $T_{s3} - \mathcal{T}_{\mathcal{C}_S} > 0$ *if* $T_{r1} - \mathcal{T}_{\mathcal{C}_R} > 0$*. Thus the receiver will never accept a "new" message as part of an "old" connection.*
Let X be the last message transmitted and Y the received message with the greatest expiration time. Starting with the definition of T_{s3},

$$T_{s3} = T_{s1} + LT + 2 \cdot \epsilon$$

Using the definition of T_{s1},

$$T_{s3} = X.Texpire + LT + 2 \cdot \epsilon$$

Using the definitions of T_{r1} and Y,

$$T_{r1} - \mathcal{T}_{\mathcal{C}_R} = Y.Texpire + LT + \epsilon - \mathcal{T}_{\mathcal{C}_R}$$

We know that Y.Texpire $\leq$ X.Texpire, due to Assumption 1 and the fact that every received message must have been transmitted, thus,

$$T_{r1} - \mathcal{T}_{\mathcal{C}_R} \leq X.Texpire + LT + \epsilon - \mathcal{T}_{\mathcal{C}_R}$$

Substituting from above, we get,

$$T_{r1} - \mathcal{T}_{\mathcal{C}_R} \leq T_{s3} - \epsilon - \mathcal{T}_{\mathcal{C}_R}$$

Using Assumption 2 above,

$$-\epsilon \leq \mathcal{T}_{\mathcal{C}_R} - \mathcal{T}_{\mathcal{C}_S}$$

Therefore,

$$T_{r1} - \mathcal{T}_{\mathcal{C}_R} \leq T_{s3} - \mathcal{T}_{\mathcal{C}_S}$$

(ii) The sender does not release a data channel CID until all data messages have exceeded their lifetime. Thus no "old" message X with connection identifier CID from a previous connection is accepted as part of a "new" connection with connection identifier CID.
By definition of T_{s1},

$$X.Texpire \leq T_{s1}$$

Using definition of T_{s3},

$$X.Texpire \leq T_{s1} < T_{s3}$$

Therefore, the lifetime of any message X will have expired before the connection identifier CID is released by the sender. □

Theorem 3 *The receiver will never accept a message with sequence number* x *as a message with sequence number* $x \oplus n \cdot |SN|$*, where* $n = \{1, 2, 3, \ldots\}$*.*

Proof: The only way for confusion to occur is if sequence number x is never received and the next sequence number received is $x + n \cdot |SN|$ (from Parameter 1). Let X be the message with sequence number x and let $\widehat{X}$ denote any message with sequence number $x + n \cdot |SN|$.
By definition of T_{s2}, X, and Parameters 2 and 3, we know that

$$T_{s2} < X.Texpire - LT + \frac{|SN|}{R}$$

By definition of X and $\widehat{X}$ and Parameters 2 and 3 we know that

$$X.Texpire + \frac{|SN|}{R} \leq \widehat{X}.Texpire$$

Substituting T_{s2},

$$T_{s2} < \widehat{X}.Texpire - LT$$

Since $\widehat{X}.Texpire - LT$ is the time $\widehat{X}$ is generated, the sender is suspended before $\widehat{X}$ can be generated. □

5.1.2 Safety of the Acknowledgment Channel

The proof of the safety of the acknowledgment channel is analogous to the proof for the data channel.

Theorem 4 *A setup message (First = on) on the acknowledgment channel is accepted at-most once.*

Proof: We will show that no acknowledgment message is accepted more than once. For any acknowledgment message ACK,

$$ACK.Texpire \leq T_{s1}$$

Since the connection record is kept long enough, duplicate acknowledgment messages are detected. □

Theorem 5 *There is never an ambiguity at the receiver as to which connection an acknowledgment message with connection identifier CID is part of.*

Proof: The proof is similar to the proof of theorem 2. To prove the theorem we must show

(i) The receiver does not release the acknowledgment channel CID until the sender has dropped its connection state, i.e. $T_{r2} - \mathcal{T}_{\mathcal{C}_R} > 0$ if $T_{s1} - \mathcal{T}_{\mathcal{C}_S} > 0$. Thus the sender will never accept a "new" acknowledgment message as part of an "old" connection.

Let X be the last message transmitted, Y the received message with the greatest expiration time, and Z the acknowledged message with the greatest expiration time.

Thus
X.Texpire $\geq$ Y.Texpire $\geq$ Z.Texpire.
From the definition of T_{s1}, T_{s2}, and T_{r2},

$$T_{s1} = X.Texpire \leq T_{s2} + LT = Z.Texpire + \frac{|SN|}{R} \leq Y.Texpire + \frac{|SN|}{R} = T_{r2} - \epsilon$$

Subtracting $\mathcal{T}_{\mathcal{C}_S}$ from each side,

$$T_{s1} - \mathcal{T}_{\mathcal{C}_S} \leq T_{r2} - \epsilon - \mathcal{T}_{\mathcal{C}_S}$$

Using Assumption 2,

$$T_{s1} - \mathcal{T}_{\mathcal{C}_S} \leq T_{r2} - \mathcal{T}_{\mathcal{C}_R}$$

(ii) The receiver does not release an acknowledgment channel CID until all acknowledgment messages have exceeded their lifetime. Thus no "old" acknowledgment message ACK with connection identifier CID from a previous connection is accepted as part of a "new" connection with connection identifier CID.
By definition of T_{r1} and T_{r2}:

$$ACK.Texpire < T_{r1} < T_{r2}$$

Therefore, the lifetime of any acknowledgment message ACK will have expired before the connection identifier CID is released by the receiver. □

Theorem 6 *The receiver of the ACK message will never confuse data sequence number x with data sequence number $x \oplus n \cdot |SN|$.*

The proof is identical to the proof of Theorem 3.

5.2 Liveness

5.2.1 Liveness of the Data Channel

We now show that a non-duplicate message that is received before its lifetime has expired is accepted as long as no previous messages have expired before they were received[6].

Theorem 7 *At the receiver, a new (non-duplicate) message X always is accepted at time t as measured on the receiver's clock if $X.Texpire + \epsilon \geq t$ and all messages Q such that $Q.Texpire + \epsilon < t$ have been received.*

[6]Once a data message has not been received and has its expiration time exceeded, the transmission has failed and the receiver is no longer required to accept any messages.

Proof: $X.Texpire + \epsilon \geq t$ implies via Lemma 1 that there exists a message Y such that $Y.Texpire + \epsilon \geq t - LT$.
By definition of T_{r1},

$$T_{r1} = \max(\text{Q.Texpire}) + LT + \epsilon \geq Y.Texpire + LT + \epsilon \geq t - LT + LT = t$$

Therefore,

$$T_{r1} \geq t$$

Because T_{r1} has not expired at time t and X is not a duplicate it must be accepted. □

5.2.2 Liveness of the Acknowledgment Channel

We now show that the sender will accept any acknowledgment message that is received before its lifetime has expired.

Theorem 8 *The transmitter always accepts an acknowledgment message ACK that has been received before its lifetime expired.*

Proof: By definition of T_{s1},

$$ACK.Texpire \leq T_{s1}$$

Therefore, the sender will not release his connection record while there exits an acknowledgment message that has not exceeded its lifetime. □

6 Operation of *CMSC* in Case of Failures

So far, we have discussed the operation of *CMSC* in absence of failures. Failures may cause connection state to be lost. We assume that when the system fails it will simply halt. To ensure the correctness of *CMSC* in case of failures, we make some additions to its operation.

6.1 Sender Failure

If the sender loses state, it must not reuse a CID until it is assured that the timer T_{s3} would have expired. In the following we discuss various strategies to cope with sender failure. The strategies differ in the waiting time after a failure until the communication can be resumed and the amount of stable storage required.

The first approach keeps the maximum value MT_{s3} of all CID release timers T_{s3} in stable storage. MT_{s3} is recomputed every time a new message is sent. After a failure, a connection identifier can be reused when $\mathcal{T}_{C_S} > MT_{s3}$, i. e. no connection can be opened until all timers T_{s3} would have expired.

The second approach keeps in stable storage the maximum lifetime $\mathcal{L}_S$ that no existing connection exceeds. After a failure no connection may be established for a period of $2 \cdot (\mathcal{L}_S + \epsilon)$. This approach requires stable storage for a single value ($\mathcal{L}_S$). Compared to the first approach, the number of writes to stable storage is reduced at the expense of a longer waiting time after a failure. In case of greatly varying lifetimes, one may use a hybrid approach in which the system keeps track of individual connections with long lifetimes and stores a single lifetime $\mathcal{L}_S$ to cover all connections with short lifetimes.

The third approach keeps the lifetime of each connection in stable storage. After a failure, a connection identifier can be reused $2 \cdot (LT + \epsilon)$ time units later, which is long enough to ensure the timer T_{s3} for this connection would have expired. The advantage of this approach is that a new connection can be opened immediately after a failure, if not all connection identifiers were in use at the time of the failure, or after a timer T_{s3} for a connection has expired, otherwise. The disadvantage is that stable storage is needed for each connection identifier in use.

The fourth approach partitions the connection identifiers and introduces a subfield whose value changes after each failure. If the value of this subfield is kept in a variable *epoch*, each time the system crashes, *epoch* is incremented. Therefore, any new connection identifier that is generated after a failure will be different from any connection identifier a receiver may associate a connection record with and new connections can be established immediately. The value of *epoch* is kept in stable storage. The number of different values *epoch* must assume can be small. All we have to assure is that
(Max value of epoch) $*$ (Min time between two failures) $> 2 \cdot (\mathcal{L}_S + \epsilon)$, where $\mathcal{L}_S$ is defined as in the second approach.

6.2 Receiver failure

When the receiver recovers after a failure, it must avoid accepting duplicates. The four approaches that are available for dealing with failures of the sender could also be used to deal with receiver failures. However, there is a simpler solution that takes advantage of the fact the clocks are ϵ-synchronized and monotonically increasing and that all messages carry an expiration time. Whenever a message is accepted, the receiver updates the value of the highest expiration time HTE of any message accepted and keeps HTE in stable storage [LISK 91]. After a failure, the receiver resumes communication immediately. It recovers the value of HTE and rejects any incoming message X with $X.Texpire \leq HTE$.

6.3 Handshake to Speed up the Release of Connection Records

In an environment with many short-lived connections, a significant amount of storage might be tight up for connection records of connections that are no longer active. To reduce the holding time of a connection record , we modify the operation of *CMSC* as defined in Section 4 and introduce a handshake to speed-up the release of a connection record. The header of a message contains another field for the *LAST*-bit: The sender sets the *LAST*-bit when it has no more data to send and wants to close the connection. After the sender has received the acknowledgements for all messages transmitted, it releases the connection record immediately. When the receiver receives a message X with the *LAST*-bit set, it sets $T_{r1} \leftarrow X.Texpire$. This modification does not change any other property of *CMSC*.

The initial values of the timers T_{s3} and T_{r2}, which are defined with respect to T_{s1} and T_{r1} respectively must be modified. The requirements for T_{s3} and T_{r2} were: (i) $T_{s3} - \mathcal{T}_{\mathcal{C}_S} \geq T_{r1} - \mathcal{T}_{\mathcal{C}_R}$ and (ii) $T_{r2} - \mathcal{T}_{\mathcal{C}_R} \geq T_{s1} - \mathcal{T}_{\mathcal{C}_S}$. Let X denote the message with the *LAST*-bit set. To meet (i), set $T_{s3} \leftarrow X.Texpire + \epsilon$ when T_{s1} expires. Since $T_{s1} \leq T_{r1}$ we only must account for the clock skew. To meet (ii) we set $T_{r2} \leftarrow \epsilon$ when T_{r1} expires.

These modifications reduce the holding times of the connection records and the connection identifiers. At the sender the release of the connection record is sped up by the difference between X.Texpire and the time at which the acknowledgment for X arrives. The release of the connection identifier is sped up by *LT*. At the receiver the release of the connection identifier is sped up by *LT* and the release of the connection record by $\frac{|SN|}{R}$.

7 Conclusion

CMSC combines the various mechanisms available for connection management: It uses timers, unique connection identifiers, stable storage, handshake, and ϵ-synchronized clocks. The use of ϵ-synchronized clocks in *CMSC* is novel. Synchronized clocks support an accurate and easy to implement end-to-end lifetime enforcement. Unique connection identifiers allow multiple connections between the same sender-receiver pair. Sender and receiver can resume communication immediately after a crash by generating unique connection identifiers across crashes and by using synchronized clocks to filter out duplicate messages at the receiver. The release of connection records is sped up if the connection is closed by handshake. *CMSC* can be incorporated in any protocol that must provide at-most-once semantics.

Acknowledgements
We thank Brian Coan and the anonymous referees for their constructive comments. This work was carried out while the first author was with Bellcore.

References

[CHER 88] D. R. Cheriton, "VMTP: A Versatile Message Transaction Protocol", *Internet Request for Comments*, RFC 1045, February 1988.

[CHER 89] D. R. Cheriton, "SIRPENT: A High Performance Internetworking Approach", *Proc. ACM SIGCOMM 89*, pp. 158–169, Austin, TX, September 1989.

[DOD 83] Department of Defense, "Transmission Control Protocol", *MIL-STD-1778*, May 1983.

[FLET 78] J. G. Fletcher and R. W. Watson, "Mechanisms for a Reliable Timer-Based Protocol", *Computer Networks*, 2(4/5):271–290, September 1978.

[KLEI 92] L. Kleinrock, "The Latency/Bandwidth Tradeoff in Gigabit Networks", *IEEE Communications Magazine*, 30(4):36–41, April 1992.

[LISK 91] 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.

[MILL 91] D. L. Mills, "Internet Time Synchronization: The Network Time Protocol", *IEEE Transactions on Communications*, 39(10):1482–1493, October 1991.

[SLOA 79] L. Sloan, "Limiting the Lifetime of Packets in Computer Networks", *Computer Networks*, 3(6):435–445, 1979.

[SLOA 83] L. Sloan, "Mechanisms that Enforce Bounds on Packet Lifetimes", *ACM Transactions on Computer Systems*, 1(4):311–330, November 1983.

[WATS 81a] R. Watson, "Timer-Based Mechanisms in Reliable Transport Protocol Connection Management", *Computer Networks*, 5:47–56, 1981.

[WATS 81b] R. W. Watson, "IPC-Interface and End-to-End Protocols", B. W. Lampson, M. Paul and H. Siegert, Eds., *Distributed Systems, Architecture and Implementation*, volume 105 of *Lecture Notes in Computer Science*, chapter 7, pp. 140–174, Springer Verlag, New York, Berlin, Heidelberg, Tokyo, 1981.

[WATS 83] R. W. Watson, "Delta-t Protocol Specification", UCID-19293, Lawrence Livermore Laboratory, Livermore, CA, April 1983.

[WATS 89] R. W. Watson, "The Delta-t Transport Protocol: Features and Experience", *Proc. 14th Conf. on Local Computer Networks*, pp. 399–407, Minneapolis, MN, October 1989, IEEE.

[WELL 90] D. Wells, *Guide to GPS Positioning*, Canadian GPS Associates, 1990.

Contents