TTC 標準
TTC STANDARD

JF-IR004.10
赤外線通信インタフェース
簡易トランスポートプロトコル

〔'Tiny TP': A flow-Control Mechanism for use with IrLMP〕

第2版

1997年11月26日制定

社団法人
情報通信技術委員会

THE TELECOMMUNICATION TECHNOLOGY COMMITTEE

Telecommunication Technology Committee

<参考>

１．英文記述の適用レベル

適用レベル：Ｅ３

本標準の本文および付録の文章および図に英文記述を含んでいる。

２．国際勧告等との関連

本標準は、赤外線通信標準化団体 IrDA(Infrared Data Association)において１９９６年１０月に採択された標準 `Tiny TP' A Flow-Control Mechanism for use with IrLMP Ver 1.1 に基づいて定めたものである。

３．上記国際勧告等に対する追加項目等

３．１ オプション選択項目

なし

３．２ ナショナルマター決定項目

なし

３．３ 先行している項目

なし

３．４ 追加した項目

なし

３．５ 削除した項目

なし

３．６ 国際勧告に対する修正内容

なし

３．７ その他

(1) 国際勧告と本標準の図および表は次のとおり対応している。

国際勧告	本標準
Figure 2.1	図２－１ ／ＪＦ－ＩＲ００４．１０（E）

Figure 2.2	図２－２ ／ＪＦ－ＩＲ００４．１０（Ｅ）
Figure 2.3	図２－３ ／ＪＦ－ＩＲ００４．１０（Ｅ）
Figure 2.4	図２－４ ／ＪＦ－ＩＲ００４．１０（Ｅ）
Figure 4.1	付図４－１ ／ＪＦ－ＩＲ００４．１０（Ｅ）
Table 2-1	表２－１ ／ＪＦ－ＩＲ００４．１０（Ｅ）

４．改版の履歴

版数	制定日	改版内容
第１版	１９９６年１１月２７日	制定
第２版	１９９７年１１月２６日	IrDA Tiny TP 標準改版(1.0→1.1)の反映

５．工業所有権

本標準に関わる「工業所有権の実施の権利に係る確認書」の提出状況は、ＴＴＣホームページでご覧になれます。

６．その他

(1)参照勧告、標準等

IrDA 標準:

IrLAP(Serial Infrared Link Access Protocol)

IrLMP(Serial Infrared Link Management Protocol)

Tiny TP (A Flow-Control Mechanism for use with IrLMP)

IrCOMM (Serial and Parallel Port Emulation over IR(Wire Replacement))

目　　次

１．序文 ６
１．１　参考文献 ６
２．手続き要素 ７
２．１　簡易トランスポートプロトコルサービスアクセス点アドレス ７
２．２　簡易トランスポートプロトコルサービスプリミティブ ７
２．２．１　TTP_Connect ７
２．２．２　TTP_Disconnect ８
２．２．３　TTP_Data ９
２．２．４　TTP_UData ９
２．２．５　TTP_LocalFlow １０
２．３　簡易トランスポートプロトコルプロトコルデータ単位 １０
２．３．１　データ TTP-PDUs １０
２．３．２　コネクト TTP-PDUs １１
２．４　処理の詳細 １３
２．４．１　変数 １５
２．４．２　クレジット操作 １５
２．４．３　分割再構成(SAR) １５
２．４．４　状態遷移表 １７
３．リンク層管理プロトコル情報アクセスサービスオブジェクトと属性 ２１
３．１　外部から参照される属性 ２１
４．付録 A　実装留意事項 ２２
４．１　簡易トランスポートプロトコル バッファリング ２２
４．１．１　受信バッファプール ２２
４．１．２　分割再構成再構成バッファ ２２
４．１．３　バッファプールと分割再構成再構成バッファの結合 ２３
４．２　簡易トランスポートプロトコルコネクションの終了 ２３
４．３　バイトストリームと順序付きパケットサービス ２３

Infrared Data Association

‘Tiny TP’: A Flow-Control Mechanism for use with IrLMP

Document Status: Version 1.0

1.1a to 1.1	On 17th October 1996 the IrDA Board voted to accept the Technical Committee's recommendation that Draft Version 1.1a of the Tiny TP specification be adopted as Version 1.1 of that specification.
1.0. to 1.1a	The use of an explicit zero-valued MaxSduSize parameter during TTP connection establishment has been deleted to enable LITE implementations of Tiny TP that do not implement SAR to merely test for the presence or absence of the MaxSduSize parameter rather the having to inspect its value for zero as well. As a consequence the overloading of the MaxSduSize parameter to tag byte-stream or sequenced packet semantics during connection establishment, suggested in Appendix A Section 4.3 has been removed. Clarification has been added to section 2.3.2.1 that clearly states that the MaxSduSize parameter is strictly applied to the size of SDUs exchanged between peer TTP clients.

INFRARED DATA ASSOCIATION (IrDA) - NOTICE TO THE TRADE -

All IrDA publications are provided "AS IS" and without warranty of any kind. IrDA (and each of its members, wholly and collectively, hereinafter "IrDA") EXPRESSLY DISCLAIM ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE AND WARRANTY OF NON-INFRINGEMENT OF INTELLECTUAL PROPERTY RIGHTS. IrDA DOES NOT WARRANT THAT ITS PUBLICATIONS WILL MEET YOUR REQUIREMENTS OR THAT ANY USE OF A PUBLICATION WILL BE UN-INTERRUPTED OR ERROR FREE, OR THAT DEFECTS WILL BE CORRECTED. FURTHERMORE, IrDA DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING USE OR THE RESULTS OR THE USE OF IrDA PUBLICATIONS IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN PUBLICATION OR ADVICE OF A REPRESENTATIVE (OR MEMBER) OF IrDA SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY.

LIMITED MEDIA WARRANTY:

IrDA warrants ONLY the media upon which any publication is recorded to be free from defects in materials and workmanship under normal use for a period of ninety (90) days from the date of distribution as evidenced by the distribution records of IrDA. IrDA's entire liability and recipient's exclusive remedy will be replacement of the media not meeting this limited warranty and which is returned to IrDA. IrDA shall have no responsibility to replace media damaged by accident, abuse or misapplication. ANY IMPLIED WARRANTIES ON THE MEDIA, INCLUDING THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, ARE LIMITED IN DURATION TO NINETY (90) DAYS FROM THE DATE OF DELIVERY. THIS WARRANTY GIVES YOU SPECIFIC LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM PLACE TO PLACE.

COMPLIANCE and GENERAL:

Membership in IrDA or use of IrDA publications does NOT constitute IrDA compliance. It is the sole responsibility of each manufacturer, whether or not an IrDA member, to obtain product compliance in accordance with IrDA Specifications.

All rights, prohibitions of right, agreements and terms and conditions regarding use of IrDA publications and IrDA rules for compliance of products are governed by the laws and regulations of the United States. However, each manufacturer is solely responsible for compliance with the import/export laws of the countries in which they conduct business. The information contained in this document is provided as is and is subject to change without notice.

Contents

1. INTRODUCTION 6

1.1 References 6

2. ELEMENTS OF PROCEDURE 7

2.1 Tiny TP Service Access Point Addresses 7
2.2 Tiny TP Service Primitives 7
2.2.1 TTP_Connect 7
2.2.2 TTP_Disconnect 8
2.2.3 TTP_Data 9
2.2.4 TTP_UData 9
2.2.5 TTP_LocalFlow 10
2.3 Tiny TP Protocol Data Units 10
2.3.1 Data TTP-PDUs 10
2.3.2 Connect TTP-PDUs 11
2.4 Detailed Operation 13
2.4.1 Variables 15
2.4.2 Credit Operation 15
2.4.3 Segmentation and Reassembly 15
2.4.4 Event/Action Table 17

3. IRLMP IAS OBJECT AND ATTRIBUTES 21

3.1 Exported Attributes 21

4. APPENDIX A IMPLEMENTATION CONSIDERATIONS 22

4.1 Tiny TP Buffering. 22
4.1.1 Receive Buffer Pool 22
4.1.2 SAR Reassembly Buffer 22
4.1.3 Combined Buffer Pool and SAR Reassembly Buffer 23
4.2 Closing TTP Connections 23
4.3 Byte Stream v Sequenced Packet Service 23

1. Introduction

Whilst IrLAP provides a flow-control mechanism between peer IrLAP [1] entities, the introduction of multiplexed channels above IrLAP by IrLMP LM-MUX [2] introduces a problem. Reliance on IrLAP to provide flow-control for a multiplexed channel can result in dead-locks if the consumption of data from one multiplexed channel is dependent on data flowing in an adjacent multiplexed channel. Conversely, if inbound data on a multiplexed channel cannot be consumed and the underlying IrLAP connection cannot be flow-controlled off due to the possibility of deadlock, inbound data (freshly arrived or buffered) must be discarded in the event of buffer exhaustion. Sadly this reduces the reliable delivery service provided by IrLAP to a best effort delivery service provided by IrLMP LM-MUX (when multiple multiplexed channels are in operation).

There are at least two possible solutions for restoring a reliable delivery service above IrLMP LM-MUX.

1. Provide a per application stream flow-control mechanism above LM-MUX between peer application entities. This ensures that there is always sufficient buffer space available to accomodate in-bound application data. OR

2. Provide a per application stream retransmission mechanism above LM-MUX that recovers from the loss of data that arises if inbound buffering become exhausted.

The Tiny TP protocol detailed in this document provides just:

- independently flow controlled transport connections
- segmentation and reassembly

1.1 References

[1] "Serial Infrared Link Access Protocol", IrLAP, Version 1.0, Infrared Data Association, June 1994

[2] "Link Management Protocol", IrLMP, Version 1.0, Infrared Data Association, August 1994

2. Elements of Procedure

2.1 Tiny TP Service Access Point Addresses.

Since each Tiny TP Service Access Point (TTPSAP) is accessible through one and only one IrLMP LM-MUX LSAP the syntax of a TTPSAP address is identical to that of an IrLMP LSAP address. Namely:

<TTSAP Address> = <LSAP Address> = <DeviceAddress><LSAP-SEL>

Thus a TTSAP is identified by the address of the LSAP through which it is accessed.

Similarly, a TTP connection (or TTP connection endpoint) is identified by the pair of TTPSAP addresses at each end of the connection.

2.2 Tiny TP Service Primitives

2.2.1 TTP_Connect

```
TTP_Connect.request(    Called TTPSAP,
                        Requested QoS,
                        Calling MaxSduSize
                        Calling UserData )

TTP_Connect.indication( Calling TTPSAP,
                        Resultant QoS,
                        Calling MaxSduSize
                        Calling UserData   )

TTP_Connect.response(   Calling TTPSAP,
                        Called MaxSduSize,
                        Called UserData )

TTP_Connect.confirm(    Called TTPSAP,
                        Resultant QoS,
                        Called MaxSduSize
                        Called UserData)
```

TTPSAP	A reference to a TTPSAP which is also an implicit reference to the corresponding IrLMP LM-MUX LSAP.
QoS	IrLMP/IrLAP Quality of Service parameters.
MaxSduSize	The maximum size of the `UserData` field, in bytes, that may delivered in a `TTP_Data.indication` primitive at the Calling and Called ends of the TTP connection respectively. The Calling and Called values for `MaxSduSize` are specified independently and may differ. A value of zero disables segmentation and reassembly (SAR). The **`UserData`** field submitted in `TTP_Data.request` must then fit within a single Data TTP-PDU whose maximum size is determined by IrLAP negotiation.

UserData	Calling `UserData` is passed from the entity initiating a TTP connection to the entity that should respond to an incomming TTP connection request. Likewise, Called `UserData` is passed back from the responding entity to the initiating entity. The total size of the Connect TTP-PDUs transferred during connection establishment is limited by the size negotiated for the underlying IrLAP connection. Currently the maximum size of the `Parameters` field of the Connect TTP-PDU is 6 bytes, therefore a connect `UserData` field of 52 bytes[1] or less can always be transferred during connection establishment. Typically this might be used as a signature field to help decide whether to accept the connection; parameter negotiation between TTP clients; or simply to piggyback a small amount of data.

This service is used to establish a TTP-connection between two IrLMP LSAPs. They are similar to the corresponding IrLMP LM-MUX LM-Connect primitives.

2.2.2 TTP_Disconnect

`TTP_Disconnect.request(UserData)`

`TTP_Disconnect.indication(Reason, UserData)`

Reason	Disconnect Reason: passed through from IrLMP. Values other than `UserRequested` indicate that the connection was terminated by the TTP service provider rather than by the peer TTP client entity.
UserData	Present only if the reason field specifies `UserRequested` Up to 60 bytes of data accompanying the disconnect. Typically this might be used to carry application specific diagnostic/reason information concerning the disconnect.

This service is used to: reject incoming TTP connections; terminate a TTP connection; and to indicate both the normal and abnormal termination of a TTP connection.

The `TTP_Disconnect` service primitives are mapped to the corresponding IrLMP[8] `LM_Disconnect` service primitives. However, a `TTP_Disconnect.indication`, generated in response to the invocation (by IrLMP) of an `LM_Disconnect.indication`, is schedule to occur after all preceding data received on the TTP connection has been passed to the local TTP client (with the exception of any partially reassembled TTP SDUs). Likewise, a `LM_Disconnect.request`, generated in response to the invocation of `TTP_Disconnect.request` by the local TTP client, is schedule after all preceding data for the TTP connection received from the local TTP client has been transferred to IrLMP.

---

[1] The minimum IrLAP maximum data size is 64 bytes. The Connect LM-PDU has a 4 byte header. If a maximum length MaxSduSize parameter is present then the Connect TTP-PDU currently has a maximum header length of 8 bytes. 64-4-8=52

2.2.3 TTP_Data

`TTP_Data.request(UserData)`

`TTP_Data.indication(UserData,Status)`

UserData	This is the TTP Client Service Data Unit (SDU) that is exchanged between TTP Clients. If SAR is active then the size of the `UserData` field submitted in the `.request` primitive must not exceed the `MaxSduSize` indicated in the TTP_Connect.indication or TTP_Connect.confirm primitive that established the connection. Likewise the size of `UserData` field delivered using the `.indication` primitive may not exceed the `MaxSduSize` indicated in the `TTP_Connect.request` or `TTP_Connect.response` primitive that established the connection. If SAR is inactive, then the size of the `UserData` field is constrained to fit within a single Data TTP-PDU.
Status	Delivery `Status` may take the following values: `OK`: If the reassembly process is inactive or has successfully reassembled the inbound SDU. `Truncated`: If the reassembly process has truncated the inbound SDU because its exceeds the agreed size.

TTP SDUs submitted by the invocation of `TTP_Data.request` are delivered to the corresponding peer by the invocation of `TTP_Data.indication`. The `TTP_LocalFlow` service is used to suspend and resume the generation of `TTP_Data.indication` service primitives.

2.2.4 TTP_UData

`TTP_UData.request(UserData)`

`TTP_UData.indication(UserData)`

Send data unreliably and without flow control. There is no guarantee that data sent in this manner will be delivered. These service primitives are mapped directly to the corresponding LM-UData service primitives for the underlying LSAP-connection. The size of the UserData field is constrained to fit a single `Data LM-PDU` used to convey the `UserData` (SDU).

2.2.5 TTP_LocalFlow

`TTP_LocalFlow.request(Flow=on|off)`

Flow — `Flow=on` enables the flow of received TTP SDUs to pass from the receiving TTP entity to the local TTP client via the local invocation of `TTP_Data.indication` primitives.

`Flow=off` halts the flow of TTP SDUs to the local TTP client. Inbound data is held backlogged within the receiving TTP entity which may apply backpressure to halt the data flow at the sending peer TTP entity.

This service is used to control the flow of received TTP SDUs between the receiving TTP entity and its local client.

2.3 Tiny TP Protocol Data Units

2.3.1 Data TTP-PDUs

Data TTP-PDUs are carried in the `UserData` field of IrLMP LM-MUX Data LM-PDUs. Hence, Data TTP-PDUs are passed as the `UserData` parameter in IrLMP LM_Data service primitives

These are the only IrLMP service primitives used to carry Data TTP-PDUs.

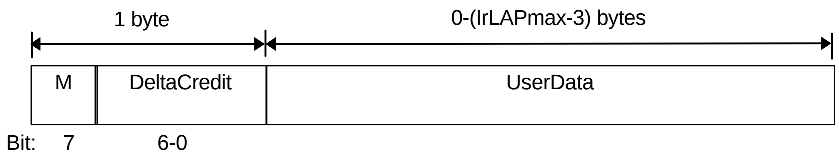

Figure 2.1 Data TTP-PDU

M — The More bit.

Only significant if SAR has been specified by the use of non-zero `MaxSduSize` during connection establishment.

When set indicates that the `UserData` field does not contain the last segment of a segmented TTP-SDU.

When clear indicates that the `UserData` field contains the final segment of a segmented TTP-SDU.

For Dataless Data TTP-PDUs (see below) the `M` bit **MUST** be sent as 0 and is ignored on reception.

DeltaCredit — Specifies the number (0-127) of additional Data-Carrying Data TTP-PDUs that may be sent in the reverse direction.

A Data-Carrying Data TTP-PDU carries 1 or more octets of `UserData`.

A Dataless Data TTP-PDU has a zero length `UserData` field.

It is **always** permissible to send a dataless Data TTP-PDU in order to advance credit to the peer TTP Entity.

UserData	If SAR is in operation then this field carries a segment of a segmented TTP-SDU. It is recommended that all segments of a segmented TTP-SDU except the last should fill outbound Data TTP-PDUs[2]. When SAR is not in operation TTP requires that TTP-SDUs fit within a single Data TTP-PDU.

2.3.2 Connect TTP-PDUs

Connect TTP-PDUs are exchanged during connection establishment and are carried in the `UserData` field of Connect LM-PDUs and Connect Confirm LM-PDUs. Hence Connect TTP-PDUs are passed as `UserData` in the IrLMP LM_Connect service primitives.

There are two forms of Connect TTP-PDU, once that carries a `Parameters` field and one that does not.

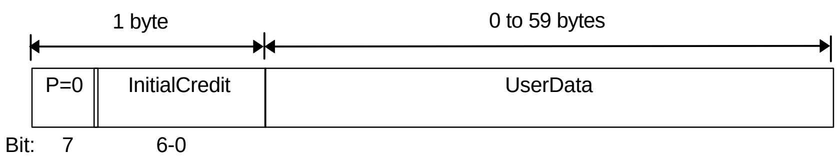

Figure 2.2 Parameterless Connect TTP-PDU

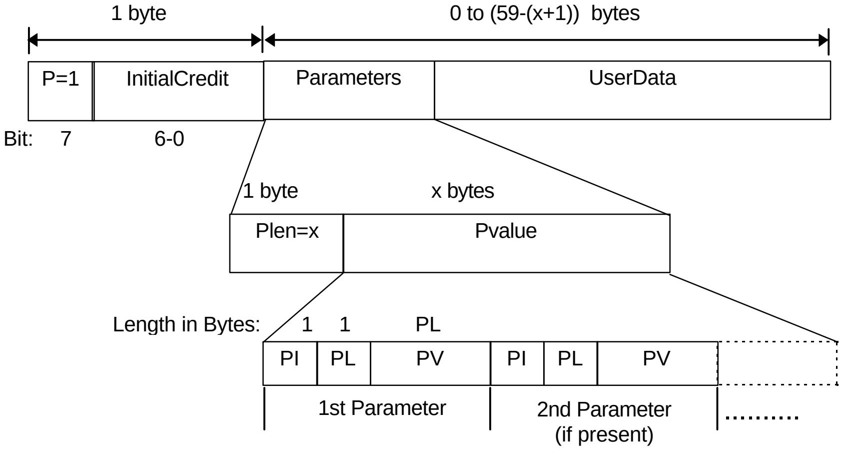

Figure 2.3 Parameter Carrying Connect TTP-PDU

[2] The maximum size of an outbound PDU may be constrained smaller than that imposed by IrLAP negotiation due to local buffer management considerations

P	When set specifies that a variable length `Parameters` field follows the `InitialCredit` field (Figure 2.3). When clear specifies that the `Parameters` field is absent (Figure 2.2) and that parameters should assume their default values.
InitialCredit	Specifies the initial number (0-127) of Data-Carrying Data TTP-PDUs that may be sent in the reverse direction. .
Parameters	A variable length field composed of two subfield, a single byte `Plen` that indicates the size in bytes (0-255) of the second sub-field `Pvalue`. `Pvalue` contains a list of tuples of the form `PI`, `PL` and `PV`. `PI` and `PL` are each a single byte in size and identify the parameter being carried and specify the length of the value carried in its `PV` field respectively. The `PV` field carries the parameter value and its interpretation depends on the value of `PI`. This tuple mechanism is identical to that used for IrLAP parameter negotiation and for the IrCOMM control channel. If there are `N` parameters then the value of `Plen` is: $Plen = \sum_{x=1}^{N} (2 + PL_X)$
UserData	The remainder of the Connect LM-PDU used to carry the Connect TTP-PDU carries `UserData` that is exchanged between TTP clients during connection establishment. Implicitly this field carries a single unsegmented TTP-SDU, ie. it DOES NOT carry the first segment of an SDU that is continued in subsequent Data TTP-PDUs. Currently the maximum size of the `Parameters` field is 7 bytes therefore a `UserData` field of up to 52 bytes may always be transferred. Future revisions of this standard may reduce this value.

2.3.2.1 Connect TTP-PDU Parameters

Tiny TP curently defines only one parameter that may be carried in the `Parameters` field of a Connect TTP-PDU

Parameter Name: MaxSduSize

PI Value: 0x01

PL Range: 0x00-0x04

Value Semantics: The maximum size of the `UserData` field, in bytes, that may delivered in a `TTP_Data.indication` primitive at the end of the connection **sending** the parameter.

Non-zeroed values of `MaxSduSize` indicate the maximum TTP-SDU sizes that receiving TTP clients are prepared to accept. The value of this parameter should be strictly applied even if it is smaller than that indicated by the appropriate IrLAP maximum data size.

The PV field is interpreted as an unsigned integer that is transferred most significant byte first (big endian). Values lie in the range 1 through (2^32)-1. Leading zero bytes may be truncated.

The default value for this parameter is 0 and arises only when either the `Parameters` field is absent or the `MaxSduSize` parameter is absent from the `Parameters` field.

0 — **MaxSduSize should never be sent with an explicit value of zero**[3]. Zero is the default value which arises when the parameter is absent Connect TTP-PDUs.

The value of zero for `MaxSduSize` disables the operation of Segmentation and Reassembly. All outbound Data TTP-PDUs should be sent with the `M` bit cleared. The `M` bit is ignored on all inbound Data TTP-PDUs. The size of TTP-SDU passed as the `UserData` parameter in TTP_Data service primitives will be constrained to fit within a single Data LM-PDU.

1 to $\left(2^{32}-2\right)$ — Values between 1 and $\left(2^{32}-2\right)$ inclusive specify the maximum size in bytes of TTP-SDU that may be delivered to the end of the connection sending the Connect TTP-PDU that carries the `MaxSduSize` parameter.

$\left(2^{32}-1\right)$ — Specifies an unbounded `MaxSduSize`. In general, this indicates that TTP entity sending this parameter AND the corresponding TTP client are capable of receiving arbitrarily large SDUs. This is like to require that the implementation of the TTP entity supports the delivery of partially reassembled TTP-SDUs and that the TTP client is capable of processing partially delivered TTP-SDUs so that buffers may be recycled.

2.4 Detailed Operation

The operation of the TTP involves the exchange of Data TTP-PDUs described in Section 2.3.1. Effectively this adds a single octet of header to the IrLMP LM-MUX Data LM-PDUs. This additional octet is used to convey increments (credits) to the number of Data TTP-PDUs that may be exchanged in each direction using the underlying LM_Data service.

[3] This is to ensure that LITE implementations of TTP that do not support SAR NEVER have to inspect the value of the MaxSduSize parameter, they merely have to test for its presence.

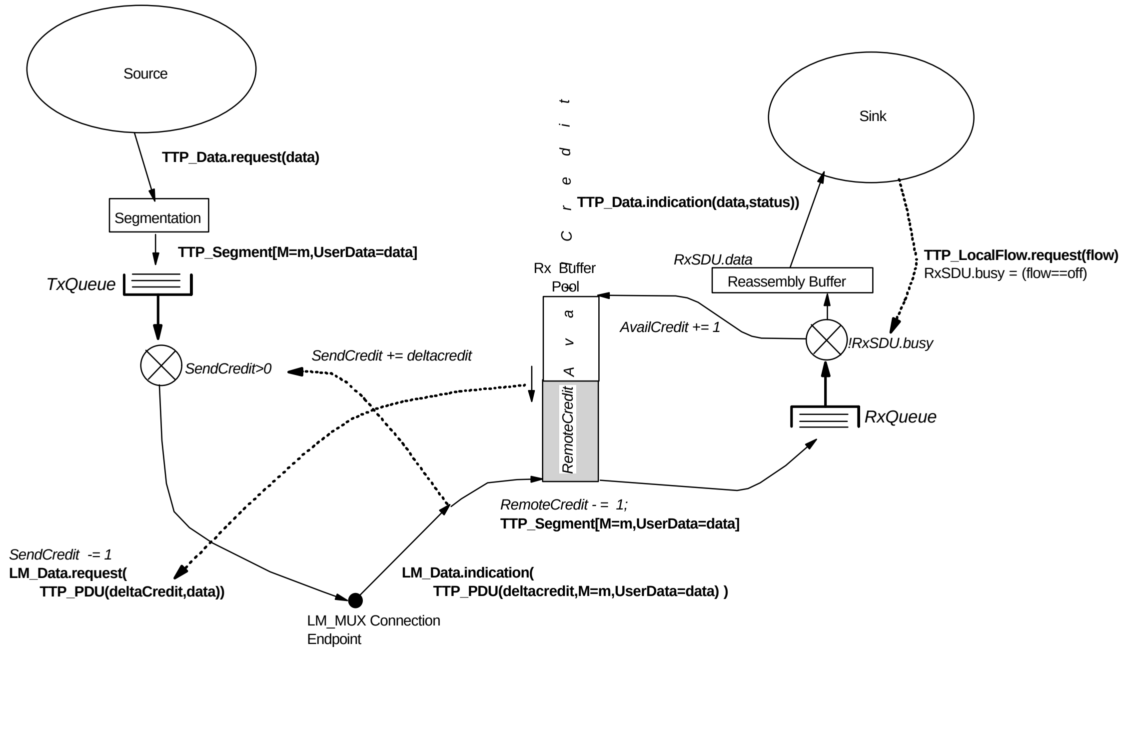

Figure 2.4 Tiny TP SAR and Credit Flow

Figure 2.4 shows the manipulation of both inbound and oubound credit at one end of a TTP connection that has reached its data phase.

For the purposes of describing the operation of Tiny TP, Figure 2.4 and Table 2-1 (below) describe a buffer management scheme that assumes a fixed number of receive buffers is available to the connection and that available credit is advanced the peer TTP entity in an aggressive way. **However, other buffer management policies are legal**.

Possible variations include:

- dynamic variation of the number of receive buffers in use by a TTP connection.
  NB. Once credit has been advance to a peer (ie. transfered from `AvailCredit` to `RemoteCredit`) it cannot be reclaimed.
- a lazy policy for advancing credit. In the context of the description given, this means that credit is held longer at on `AvailCredit` rather than being advanced at the eariliest opportunity. This leaves buffers available to be reclaimed (from `AvailCredit`) and redeployed to other needy TTP connections or to relieve resource problems elsewhere in a system.

Note that Connect TTP-PDUs exchanged during connection establishment are **not** regarded as requiring or consuming credit. During connection establishment the use of segmentation and reassembly is indicated as are any constraints on the maximum Service Data Unit (SDU) that can be conveyed using the TTP_Data service.

The TTP entity manipulates variables associated with each TTP-Connection that implicitly encode the state of the flow control mechanism for that connection. The mechanism is described in the Event/Action pairs of Table 2-1 (below).

2.4.1 Variables

AvailCredit	Credit available to advance to the peer TTP entity.
RemoteCredit	Credit held by peer TTP Entity:
SendCredit	Credit held by local TTP Entity
Connected	A flag that monitors the state of the underlying LM-MUX connection.
TxQueue	FIFO queue used to hold TTP_Segments and TTP_Disconnect requests. Holds segmented SDUs and prevents a disconnect overtaking queued data.
RxQueue	FIFO queue used to hold inbound TTP_Segments and TTP_Disconnect requests. TTP_Segments are transferred from **RxQueue** to the reassembly buffer, **RxSdu.data**. Credit is recycled as this transfer takes place.
MaxSegSize	The maximum size of segment conveyed in an outbound Data TTP-PDU.
TxMaxSduSize	Received from peer TTP entity during connection establishment. Used to guard the size of TTP-SDUs submitted using the TTP_Data.request service primitive.
RxMaxSduSize	Transmitted to peer TTP entity during connection establishment. Used to police the size of inbound TTP-SDUs.
RxSdu.size	The current size of a partially received SDU undergoing reassembly.
RxSdu.data	The current partially received SDU undergoing reassembly.
RxSdu.busy	A flag that controls the consumption of the receive queue. Set/cleared by invocation of TTP_LocalFlow.request.

2.4.2 Credit Operation

If **SendCredit** is non-zero then the local entity may reliably send data. While **SendCredit** is zero TTP_Data.request primitives are left queued in sequence on **TxQueue** awaiting credit from the peer entity.

If **RemoteCredit** is non-zero then the peer entity is able to reliably send data.

If **AvailCredit** is non-zero then the local entity has credit available that it has not yet advanced to its peer. Credit is advanced in blocks of up to 127 with the normal flow of data.

If **RemoteCredit** falls below some configured **LowThreshold** and **AvailCredit** is or becomes non-zero whilst **TxQueue** is empty or **SendCredit** is zero, then **RemoteCredit** is advanced by the transmission of a dataless FlowData TTP-PDU.

2.4.3 Segmentation and Reassembly

When Segmentation and Reassembly (SAR) is disabled in a given direction then all SDUs exchanged must fit within a single Data TTP-PDU. In this case the M bit is ignored on reception.

When SAR is enabled and a received SDU exceeds the maximum SDU size indicated during connection establishment the resulting SDU is truncated and the truncated portion delivered (logically) when the final segment arrives with a status code that indicates that an error has occurred.

In the formal description that follows it should be noted that it is assumed that data is delivered to the TTP client only on reception of the last segment of an SDU subjected to SAR.

This may not be the case in some practical interfaces which allow the delivery of partially reassembled SDUs in the interests of economising on buffer space. This latter style of interface is capable of supporting unbounded SDU sizes. See Section 4.1 for more discussion of Tiny TP receive buffering strategies.

2.4.4 Event/Action Table

Event	Action

TTP_Connect.request(
CalledTTPSAP=sap-id,
RequestedQos=qos,
MaxSduSize=mSduSize
UserData=data)
	Connected=False; AvailCredit=0; RxMaxSduSize=mSduSize;
	RxSdu.size=0; RxSdu.busy=False;
	
	n = DEFAULT_INITIAL_CREDIT /* Local Policy */
	
	RemoteCredit=0; SendCredit=0;
	if(n>127) { AvailCredit=n-127; n=127 }
	RemoteCredit=n;
	
	if(mSduSize == 0)
	ttpPdu =
	ConnectTTP-PDU(P=0,InitialCredit=n,UserData=data);
	else
	ttpPdu =
	ConnectTTP-PDU (P=1, InitialCredit=n,
	Parameters={(PiMaxSduSize,mSduSize)}
	UserData=data);
	
	LM_Connect.request (
	CalledLsap=sap-id, RequestedQos=qos,ClientData=ttpPdu)

TTP_Connect.response(
CallingLSAP=sap-id
MaxSduSize=mSduSize
UserData=data)
	AvailCredit=0;RemoteCredit=0; RxMaxSduSize = mSduSize;
	RxSdu.size = 0; RxSdu.busy = False
	
	n = DEFAULT_INITIAL_CREDIT /* Local Policy */
	
	if(>127) { AvailCredit=n-127; n=127 }
	RemoteCredit=n
	
	if(mSduSize == 0)
	ttpPdu =
	ConnectTTP-PDU(P=0,InitialCredit=n,UserData=data);
	else
	ttpPdu =
	ConnectTTP-PDU (P=1, InitialCredit=n,
	Parameters={(PiMaxSduSize,mSduSize)}
	UserData=data);
	
	LM_Connect.response(
	CallingLsap=sap-id, RequestedQos=qos,ClientData=ttpPdu)
	Connected=True;

TTP_Disconnect.request(UserData=data)	AppendTail(TxQueue, [TTP_Disconnect, UserData=data])
TTP_Data.request(UserData=data) ∧ (sizeof(UserData)==0 ∨ !Connected)	Error;
TTP_Data.request(UserData=data) ∧ TxMaxSduSize == 0 ∧ sizeof(UserData)>(MaxSegSize) ∧ Connected	//SAR Disabled Error

Event	Action
TTP_Data.request(UserData=data) $\wedge$ TxMaxSduSize $\neq$ 0 $\wedge$ sizeof(UserData)>TxMaxSduSize $\wedge$ Connected	//SAR Enabled Error
TTP_Data.request(UserData=data) $\wedge$ TxMaxSduSize == 0 $\wedge$ sizeof(UserData)>0 $\wedge$ sizeof(UserData) <= MaxSegSize $\wedge$ Connected	// SAR Disabled queue as a last segment. AppendTail(TxQueue [TTP_Segment,M=0,.UserData=data])

TTP_Data.request(UserData=data) $\wedge$
TxMaxSduSize $\neq$ 0 $\wedge$ sizeof(UserData)>0 $\wedge$
(TxMaxSduSize == UnBounded $\vee$
sizeof(UserData) < TxMaxSduSize) $\wedge$
Connected
	// SAR Enabled
	
	NumSegs = INT ((sizeof(data)+MaxSegSize-1) / MaxSegSize)
	
	// Queue all but the last segment
	for(i=1;i<NumSegs;i++) {
	AppendTail(TxQueue,
	[TTP_Segment, M=1, GetSegment(i,data)]
	}
	
	// Queue the last segment of the SDU
	AppendTail(TxQueue,
	[TTP_Segment, M=0, GetSegment(NumSegs,data)])

TTP_UData.request(UserData=data) $\wedge$!Connected	Error
TTP_UData.request(UserData=data) $\wedge$ Connected	LM_UData.request(ClientData=data)

TTP_LocalFlow.request(Flow=flow)	if (flow == on)
	RxSDU.busy = false;
	else
	RxSDU.busy = true;

LM_Connect.indication(CallingLsap=sap-id, ResultantQos=qos, ClientData=ConnectTTP-PDU [P=0,InitialCredit=n, UserData=data])	SendCredit = n; TxMaxSduSize = 0 MaxSegSize = MaxTxIrLapDataSize-3 TTP-Connect.indication(CallingTTPSAP=sap-id, ResultantQos=qos, MaxSduSize=TxMaxSduSize, UserData=data);

LM_Connect.indication(
CallingLsap=sap-id, ResultantQos=qos,
ClientData=ConnectTTP-PDU
[P=1,InitialCredit=n,
Parameters=plist,
UserData=data])
	SendCredit=n; TxMaxSduSize = 0
	
	MaxSegSize = MaxTxIrLapDataSize-3
	
	for (each (pi,pv) in plist)
	if (pi==PiMaxSduSize)
	TxMaxSduSize = pv;
	
	TTP-Connect.indication(CallingTTPSAP=sap-id,
	ResultantQos=qos,
	MaxSduSize=TxMaxSduSize,
	UserData=data);

LM_Connect.confirm(	SendCredit=n; TxMaxSduSize = 0;

Event	Action
CalledLsap=sap-id, ResultantQos=qos, ClientData=ConnectTTP-PDU [P=0, InitialCredit=n, UserData=data])	MaxSegSize = MaxTxIrLapDataSize-3 TTP_Connect.confirm(.CalledTTPSAP=sap-id, ResultantQos=qos, MaxSduSize=TxMaxSduSize UserData=data); Connected=True;

LM_Connect.confirm(
CalledLsap=sap-id,
ResultantQos=qos,
ClientData=ConnectTTP-PDU
[P=1, InitialCredit=n,
Parameters=plist,
UserData=data])
	SendCredit=n; TxMaxSduSize = 0;
	
	MaxSegSize = MaxTxIrLapDataSize-3
	
	for (each (pi,pv) in plist)
	if (pi==PiMaxSduSize)
	TxMaxSduSize = pv;
	
	TTP_Connect.confirm(.CalledTTPSAP=sap-id,
	ResultantQos=qos,
	MaxSduSize=TxMaxSduSize
	UserData=data);
	Connected=True;

LM_Disconnect.indication(Reason=r, ClientData=data)	Connected=False; Flush(TxQueue); /* Queue inbound disconnect to allow buffer data to drain */ AppendTail(RxQueue, [TTP_Disconnect, Reason=r, ClientData=data]);
LM_Data.indication(ClientData=FlowData TTP-PDU [M=m, DeltaCredit=n, UserData=data]) ∧ sizeof(UserData)==0	/* Dataless FlowData TTP-PDU */ SendCredit = SendCredit+n;
LM_Data.indication(ClientData=FlowData TTP-PDU [M=m, DeltaCredit=n, UserData=data]) ∧ sizeof(UserData) > 0	/* Deal with the inbound Credit */ SendCredit = SendCredit+n; RemoteCredit = RemoteCredit-1; /* Put Received Segment on Rx Queue */ AppendTail(RxQueue,[TTP_Segment, M=m, Userdata=data])
LM_UData.indication(ClientData=data)	TTP_UData.indication(UserData=data)
Head(TxQueue)== [TTP_Disconnect, UserData=data]	Connected=False; Flush(TxQueue); Flush(RxQueue); LM_Disconnect.request(Reason=UserRequested, ClientData=data);

Head(TxQueue) ==
[TTP_Segment, M=m, UserData=data] ∧
SendCredit>0
	n=AvailCredit; AvailCredit=0;
	If(n>127) { AvailCredit=n-127; n=127 }
	RemoteCredit=RemoteCredit+n;
	SendCredit=SendCredit-1;
	LM_Data.request(ClientData=
	Data TTP-PDU
	[M=m, DeltaCredit=n,UserData=data])

Event	Action
	DeQueueHead(TxQueue)

(Empty(TxQueue) ∨ SendCredit==0) ∧
RemoteCredit<=LowThreshold ∧
AvailCredit>0 ∧
Connected
	/* Send a Dataless FlowData TTP-PDU */
	
	n=AvailCredit; AvailCredit=0;
	If(n>127) { AvailCredit=n-127; n=127 }
	RemoteCredit=RemoteCredit+n;
	
	LM_Data.request(ClientData=
	Data TTP-PDU
	[M=0, DeltaCredit=n,UserData=NULL])

Head(RxQueue) ==
[TTP_Segment, M=1, UserData=data] ∧
RxMaxSduSize ≠ 0 ∧
!RxSdu.busy
	/* Non-terminal SDU Segment */
	RxSdu.size=RxSdu.size+sizeof(data);
	
	if(RxSdu.size<=RxMaxSduSize ∨
	RxMaxSduSize==UnBounded) {
	RxSdu.data = Reassemble(RxSdu.data,data);
	}
	
	/* Recycle Segment Buffer */
	DeQueueHead(RxQueue)
	AvailCredit = AvailCredit+1;

(Head(RxQueue) ==
[TTP_Segment, M=0, UserData=data] ∧
!RxSdu.busy) ∨
 	
(Head(RxQueue) ==
[TTP_Segment, M=m,UserData=data] ∧
RxMaxSduSize ==0) ∧
!RxSdu.busy)
	/* Last SDU Segment or Inbound SAR disabled*/
	
	RxSdu.size=RxSdu.size+sizeof(data)
	
	if(RxSdu.size<=RxMaxSduSize ∨
	RxMaxSduSize==UnBounded ∨
	RxMaxSduSize==0)
	RxSdu.data = Reassemble(RxSdu.data,data);
	TTP_Data.indication(UserData=RxSdu.data, Status=OK);
	} else {
	TP_Data.indication(UserData=RxSdu.data,
	Status=Truncated);
	}
	
	/* Recycle Segment Buffer */
	DeQueueHead(RxQueue)
	AvailCredit = AvailCredit+1;

Head(RxQueue) == [TTP_Disconnect, Reason=r, ClientData=data] ∧ !RxSdu.busy	Flush(RxQueue); /Should be empty anyway */ TTP_Disconnect.indication(Reason=r,ClientData=data);

Table 2-1   Tiny TP Entity - Event Action Table

3. IrLMP IAS Object and Attributes

3.1 Exported Attributes

This section defines an attribute that is intended for use in the definition of object classes which represent service providers that make their service available via a TTP entity.

Use of this attributes is not mandatory, but its use is strongly encouraged in those circumstances where an attribute is required for the same purpose as this attribute is defined.

Attribute Name	Value Type	Description
IrDA:TinyTP:LsapSel	Integer	The value carried in this attribute identifies the IrLMP LSAP/TTPSAP of the TTP entity that provides access to the service represented by the containing object Legal values are restricted to the range 0x01-0x6f.

4. Appendix A Implementation Considerations

4.1 Tiny TP Buffering.

The description of Tiny TP given in Section 2.4 explicitly includes both segment buffering, the receive buffer pool and the RxQueue and a SAR Reassembly buffer (the variable RxSDU).

4.1.1 Receive Buffer Pool

The receive buffer pool and receive queue shown in Figure 2.4 may be implemented as a circular buffer as shown below.

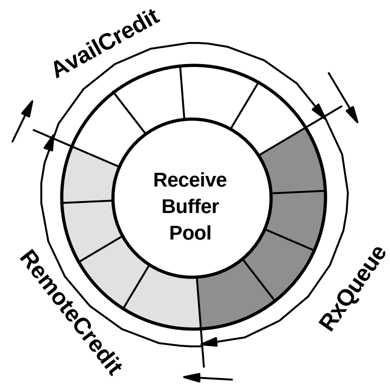

Figure 4.1 Receive Buffer Pool Organisation

If this is implemented as a circular list rather than an array, its size may be altered dynamically, provided buffers are added or removed in the current AvailCredit region.

Note that Tiny TP can function with a single buffer in this receive buffer pool. However, a single TTP connection cannot then take full advantage of the underlying IrLAP window, except for an IrLAP window size of 1. If resources allow the size of the TTP receive buffer pool for a TTP connection should be at least equivalent to the size of the current IrLAP receive window. In this way a single TTP connection can the entire IrLAP window. Additional credit up to twice the IrLAP window size enables the connection to make smooth progress without the need to rapidly advance fresh credit as inbound PDUs are consumed and buffer space recycled.

Thus a TTP receive buffer pool size of twice the IrLAP receive window size should allow a Tiny TP connection to progress smoothly. Nevertheless, even with a TTP receive buffer pool size of just 1 TTP will function, although progress is unlikely to be smooth.

4.1.2 SAR Reassembly Buffer

The description of Tiny TP given in this document also explicitly includes a per TTP connection SAR reassembly buffer. However, if the API exposed by an implementation of Tiny TP supports the partial delivery of SDUs then this buffer is entirely unnecessary. Buffering of inbound PDUs is all this is required ie. the receive buffer pool described in the previous section. Buffers from the

RxQueue section of the pool may be delivered to the TTP client, thus freeing them to collect more data segments.

Thus a implementation of Tiny TP with minimal buffering requirements would provide paritial SDU delivery and use a single TTP-SDU buffer per receive buffer pool.

Another observation that may assist the implementor is that inbound Data TTP-PDUs need only be buffered on RxQueue whilst the SAR reassembly buffer is unavailable. If the SAR buffer is available, the data carried a freshly delivered PDU may be transferred directly.

4.1.3 Combined Buffer Pool and SAR Reassembly Buffer

Another alternative for reducing the amount of buffer space require for TTP is to combine the buffer pool and the SAR buffer. Referring to Figure 4.1 above, the RxQueue segment performs the function of the SAR buffer. In this case received Data TTP-PDUs are packed into the receive buffer and the total size of the pool must equal or exceed that of client specified maximum receive SDU size. Credit is only advanced as the amount of space under the 'AvailCredit' portion successively exceeds integer multiples of the current maximum segment size. If the entire buffer become filled with an incompletely reassembled SDU truncated delivery should occur (freeing the buffer space) and resynchronisation is then accomplished at the next SDU boundary.

4.2 Closing TTP Connections

Tiny TP does not implement graceful disconnect. However, under normal circumstances, a `TTP_Disconnect.request` will be invoked from one end of a TTP connection. Data in the reverse direction may be lost if it has not all been delivered to a TTP client. However, data previously send by the TTP client that initiates the disconnect will be delivered before the corresponding `TTP_Disconnect.indication` is delivered to the peer TTP client.

If TTP disconnect can be initiated from either end then it is necessary for TTP clients to ensure that it is safe to close a TTP connection. Implementors of TTP clients should be aware that if no measures are taken in the application protocol to ensure that it is 'safe' to close a TTP connection (eg. an exchange of "I'm Done", "So am I" messages) prior to the invocation of TTP_Disconnect.request data may be lost in EITHER direction.

4.3 Byte Stream v Sequenced Packet Service

Strictly, Tiny TP offers a sequence packet service. Even if `MaxSduSize` are allowed to default during connection establishment then SDU boundaries are still maintained, however the SDU is constrained in size to fit within a single maximum sized Data TTP-PDU.

Thus with SAR disabled, TTP may be used to implement either: a sequenced packet service, where SDU boundaries are preserved between TTP peers but the maximum SDU size is constrained by the maximum sized Data TTP-PDU; or a byte-stream service where there is no guarantee that SDU boundaries are preserved between peer TTP clients.

Implementors of TTP clients should be aware of this distinction, particularly in cases where the relative alignment of SDUs and PDUs is important to the operation of the application protocol.

<付録>

IrDA赤外線プロトコル簡易トランスポートプロトコル(Tiny TP)の概要紹介

簡易トランスポートプロトコルは、IrDA 標準プロトコルスタックにおいて、OSI 参照モデルのトランスポート層に相当する機能を提供する。簡易トランスポートプロトコルはトランスポート層の機能のなかで利用頻度が高いフロー制御機能と分割再構成の機能のみを規定している。

簡易トランスポートプロトコルのサービスには、呼の確立を行なう TTP_Connect サービス、呼の解放や切断を行なう TTP_Disconnect サービス、高信頼なデータ転送を行なう TTP_Data サービス、低信頼データ転送を行なう TTP_UData サービスがある。

簡易トランスポートプロトコルは、リンク層管理プロトコル(IrLMP)サービスを用いて規定されている。簡易トランスポートプロトコルは実装任意（オプション）となっている。

簡易トランスポートプロトコルのフロー制御機能と分割再構成機能の概要について説明する。

・フロー制御機能

フロー制御は受信側の処理が追いつかないうちに送信側がデータを送信してしまい受信側で受信処理に支障が発生するのを防ぐため、送信側と受信側との間でデータ転送を調節する機能である。簡易トランスポートプロトコルのフロー制御はクレジットという値を用いて行なわれる。クレジットとは、それぞれの時点において相手局へ連続して送信することが許される最大パケット数である。まず、コネクション確立時に自局の受信バッファの量や割り当てに従い初期クレジットパラメタを互いに相手へ送りあう。これが相手局のクレジットの初期値となる。クレジットが１以上のときパケットを送信することができ、パケットを１つ送る度にクレジットを１つずつ減少させる。クレジットが０になった時点でパケットの送信が禁止される。いっぽう、相手局から送信されてくるパケットには、受信処理の結果増加した受信バッファの空きを表す差分クレジットパラメタが重畳され、このパラメタに従ってクレジットを更新し、パケットが送信可能となる。以上のような処理によって、リンク層管理プロトコルの各リンクを通過するパケットのフローをそれぞれ受信側の都合で決定することができる。 IrDA のプロトコルスタックにおいては、一般に１本のデータリンク層プロトコル(IrLAP)のリンクに複数のリンク管理層プロトコルのリンクが多重化されるため、あるリンク管理層プロトコルのリンクのトラフィック増大に従って他のリンクの通信がブロックされる恐れがあるが、簡易トランスポートプロトコルのフロー制御によって、ブロッキングを避けることができる。

・分割再構成機能

リンク層管理プロトコルではリンク層プロトコルの折衝により決まる最大パケット長よりも大きなパケットを送信することができない。簡易トランスポートプロトコルの分割再構成機能とは、大きなパケットを送信可能とするためのものであり、送信側で大きなパケットをリンク層管理プロトコルで送信可能な複数の小さなパケットに分割し受信側で元の大きなパケットに復元する機能である。簡易トランスポートプロトコルでは、利用者データ長の長さとして、０（分割再構成を使用せずデータリンク層の折衝の結果定まるパケット長を越えないデータのみ送信）、１～2^{32}-2（予め有限の最大利用者データ長を指定）、2^{32}-1（無限長を指定）、の選択が可能である。