Université de Liège

Faculté des Sciences Appliquées

Systèmes et Automatique

Institut d'Electricité Montefiore, B28
Université de Liège au Sart Tilman
B-4000 Liège 1 (Belgique)

Esprit Project 5341

OSI95

The OSI95 Transport Service with Multimedia support

***

On the Provision of a Fast Connect Facility in a Connection-Mode Transport Service

G. Leduc and A. Danthine

[LeD 94] G. Leduc, A. Danthine, **On the Provision of a Fast Connect Facility in a Connection-Mode Transport Service**, *The OSI95 Transport Service with Nultimedia Support*, A. Danthine, University of liège, Belgium (ed.), Research Reports ESPRIT - Project 5341 - OSI95 - Volume 1, Springer-Verlag, 1994, pp. 225-238.

Date: July 1994
R94-06

On the Provision of a Fast Connect Facility in a Connection-Mode Transport Service

Guy Leduc[1] and André Danthine
[1] Research Associate of the Belgian National Fund for Scientific Research (F.N.R.S.)
Université de Liège, Institut d'Electricité Montefiore, B 28, B-4000 Liège 1, Belgium
Email: leduc@montefiore.ulg.ac.be

A "fast connect" facility in a connection-mode service allows the calling service user to send service data units without waiting for the round trip delay associated with the successful establishment of the connection. In this paper we investigate how such a fast connect facility can be added to a connection-mode transport service. Starting from the eXpress Transfer Protocol (XTP) which is at the origin of the concept, we analyse the High Speed Transport Service (HSTS) and study how it can be improved by the introduction of a listening interaction at the called side. After having criticized these two transport facilities, we finally propose three possible models of the fast connect facility. The first one is the 4-primitive connection establishment from the Broadband Transport Service (BTS) and the last ones are two 3-primitive connection establishments.

Keywords: Transport Service, Connection-mode, Fast Connect

1 Introduction

In today's high-speed networks, the 'throughput × round-trip delay' (denoted T × RTD) product may easily be of the order of several hundred Kbits or even several Mbits. Such values of the T × RTD product imply that an opportunity for sending a sizeable amount of data will be lost if data can only be sent when the negotiation of the Transport Connection (TC) is complete. The figures 1a and 1b depict the classical 4-primitive exchange of a connection establishment in case of success and refusal respectively. The ISO Transport Service ([ISO 8072]) is the standard example of such a TC establishment.

To allow the Transport Service (TS) users to send data without waiting for a RTD, the eXpress Transfer Protocol (XTP) [PEI XTP] introduced the "fast connect" idea. We would have liked to explain this idea on XTP directly. However, as there is unfortunately no official description of the service provided by XTP, this would have led us to a description of the protocol mechanisms, whereas we want to refrain from giving protocol details in this paper that deals essentially with service issues. The fast connect idea may nevertheless be explained on the associated High Speed Transport Service (HSTS) [ISO HSTS]. The HSTS definition originates from the effort to standardize XTP in ISO: the XTP group, which wanted to promote XTP, realized that a clear service definition was needed prior to any attempt to standardize the protocol.

HSTS is a service rather close to the ISO Connection-mode Transport Service ([ISO 8072]), and is based on the classical sequence of four connection establishment primitives. The fast connect idea appears as the ability for the calling TS-user to submit T-DATA.request primitives before receiving the T-CONNECT.confirm (fig. 2a). If the connection is refused (fig. 2b) then the data in transit will simply be discarded.

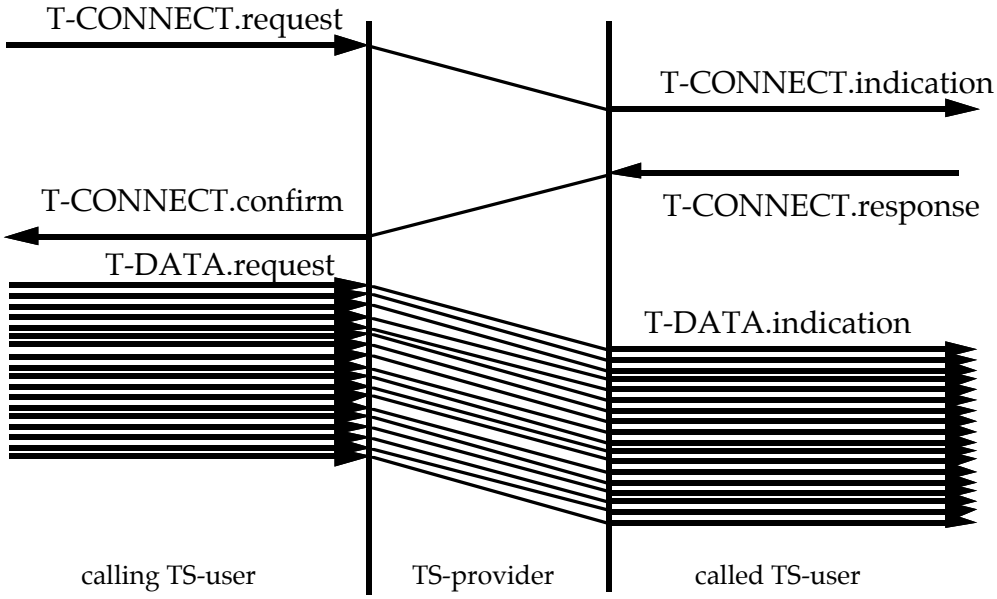

Fig.1a: Connection establishment in ISO 8072 - Successful case

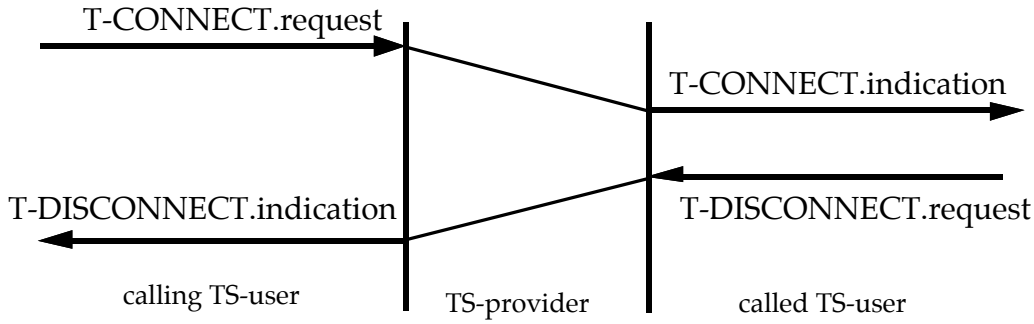

Fig.1b: Connection establishment in ISO 8072 - Unsuccessful case

This 4-primitive establishment was also considered in our group as an interesting first service description of the fast connect facility [Dan 92, DBL 92]. And from these figures 2a and 2b, we started to draw two main conclusions. First, the negotiation, which normally takes place during the 4-primitive exchange of the TC establishment, cannot be achieved any more in the fast connect scenario. This is simply because the data transfer may begin before the end of the TC establishment. Stated otherwise, the

negotiation is restricted to a "take-it-or-leave-it" negotiation. This means that, during the establishment, neither the TS-provider nor the called TS-user can change the QoS (Quality of Service) parameters specified in the T-CONNECT.request: they have to take it "as is" or leave it. This is a price to pay for the fast connect service. Moreover, this important change in the semantics requires the provision of new specific TC set-up primitives for the fast connect facility, or equivalently the addition of an explicit fast connect option in the T-CONNECT.request primitive. We also drew a second conclusion about the fast connect. In the refusal case, a very large amount of data in transit will be thrown away, which leads to a wasting of the provider resources. This is certainly not expected to occur too often. Therefore we made the second hypothesis that such "fast connections" are supposed to be accepted in most cases. This is why we came to consider that a "fast connect" service facility is only meaningful when the two TS-users are in a same and thus well-known environment.

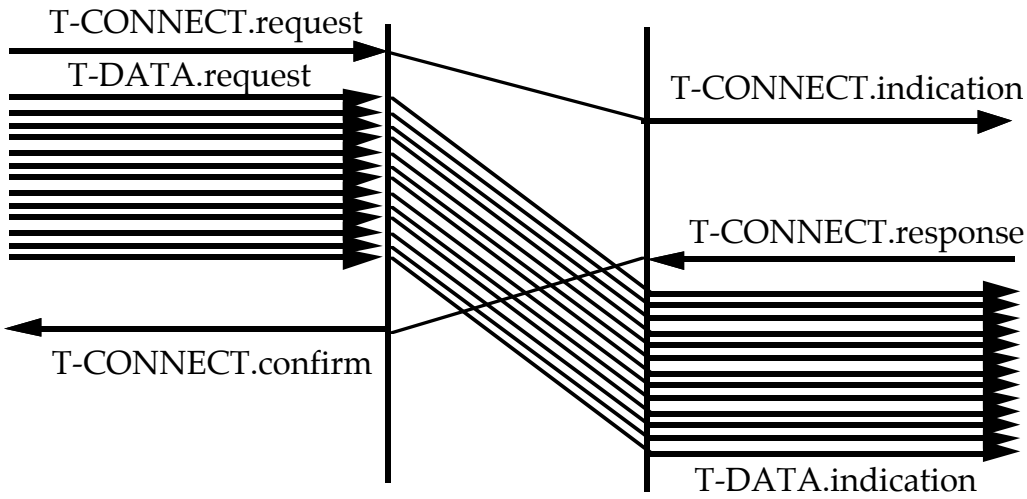

Fig.2a: Fast connect in HSTS - Successful case

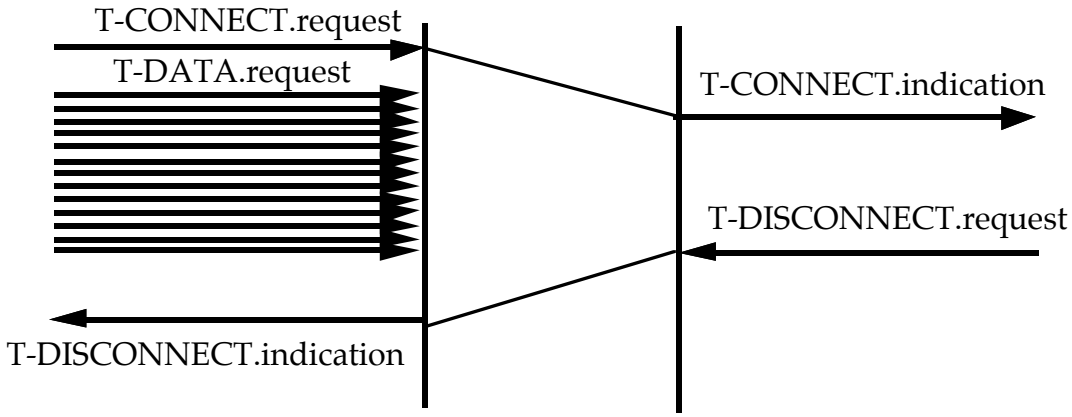

Fig.2b: Fast connect in HSTS - Unsuccessful case

In summary we may argue that the fast connect facility and the classical connection establishment facility with full negotiation are complementary. To avoid confusion between these two service facilities in the sequel, we will use distinct TS-primitives to refer to them. For a fast connect, the four establishment primitives will be denoted T-FastCONNECT.request, T-FastCONNECT.indication, T-FastCONNECT.response and T-FastCONNECT.confirm.

2 An Improvement of the Fast Connect by Way of a Listening Concept

After having clearly defined under which conditions a fast connect facility was meaningful, we tried to improve the scenario of figure 2a to overcome the following problem.

On the called side, it is only after a complete interaction with the called TS-user (by way of the T-FastCONNECT.indication and then the T-FastCONNECT.response primitives) that the TS-provider is able to deliver the beginning of the stream of data to the called TS-user. The delay associated with this interaction between the TS-provider and the called TS-user means that the amount of data that the TS-provider will have to store before the delivery may be very important, which will interfere with the resources available in the TS-provider. Figures 3a and 3b are refinements of figures 2a and 2b. They better illustrate (i) that the data are stored in the called transport entity (fig. 3a), and (ii) that Network Service (NS) provider resources are wasted in the unsuccessful case (fig 3b).

Even if the second effect is unavoidable, we considered that the solution proposed by HSTS in figure 2a needed improvements to overcome the first undesired effect.

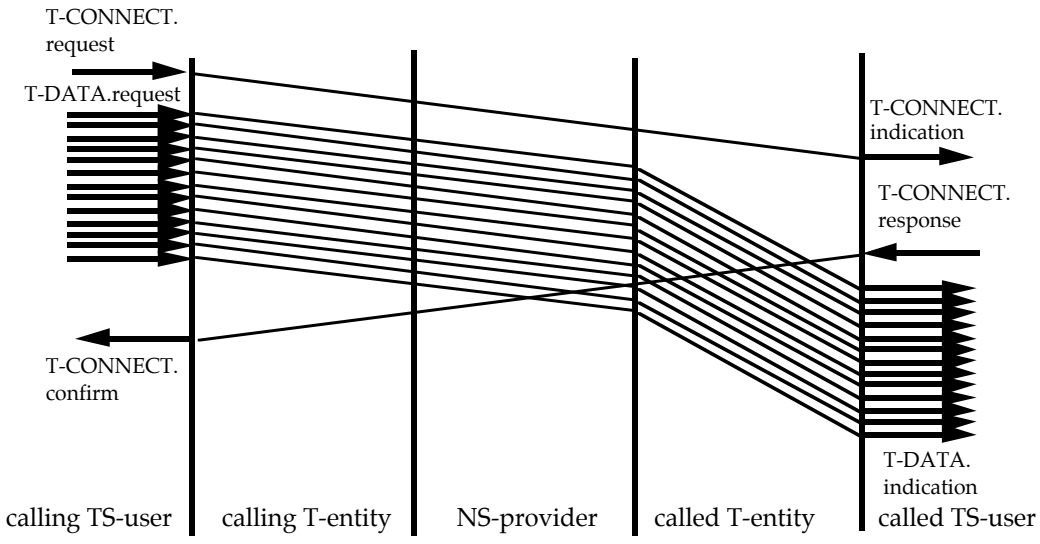

Fig. 3a: Refinement of the Fast connect in HSTS - Successful case

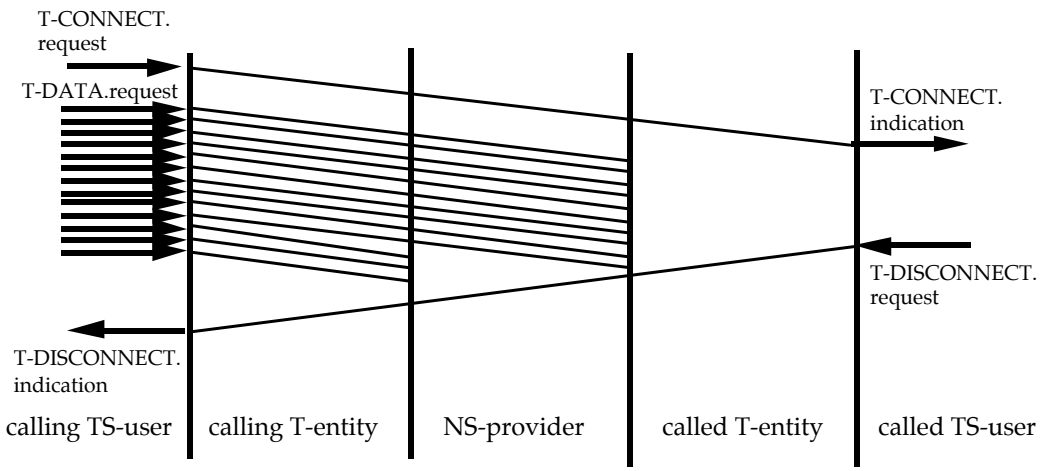

Fig. 3b: Refinement of the Fast connect in HSTS - Unsuccessful case

2.1 The Listening Interaction

Coming back to XTP, from which the fast connect idea originates, and looking at how this problem is solved in the implementations, we realize that in XTP implementations a user has the ability to pass some information to its local XTP entity to inform it on its acceptation conditions of incoming fast connections. Thanks to this interaction, denoted "listening interaction", the XTP entity may decide on the acceptation or refusal of an incoming fast connection. In the refusal case, the XTP entity discards the incoming data packets and sends back a disconnect packet. In the acceptation case, the XTP entity informs its local user, sends back an acceptation packet to the source entity, and forward immediately the incoming data packets to its user.

Our first idea in OSI95 [Dan 92, DBL 92] has been to derive from these implementation considerations an abstract service facility in terms of an ISO-like sequence of TS-primitives. A first proposal is depicted in figure 4a. A T-LISTEN.request is used to model the listening interaction of XTP. Its parameters are composed of QoS and addressing requirements. We may consider it as a pending request to any incoming demand matching its QoS and addressing requirements. When the TS-provider receives a T-FastCONNECT.request, it can thus check whether the requested QoS and addressing parameters are matching those of the T-LISTEN.request. If the matching is successful, the TS-provider delivers to the service user a T-FastCONNECT.indication which may be immediately followed by the stream of TSDUs through T-DATA-indications. If the matching is not possible (fig. 4b), the connection will not be established, the TSDUs sent following the T-FastCONNECT.request will be lost and the provider will issue a T-DISCONNECT.indication to the calling TS-user.

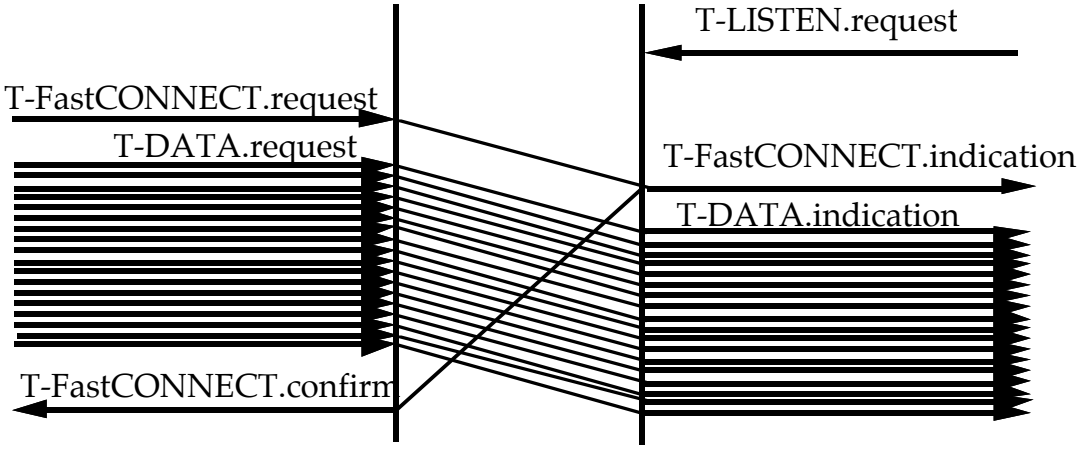

Fig. 4a: Fast connect with listening interaction - Successful case

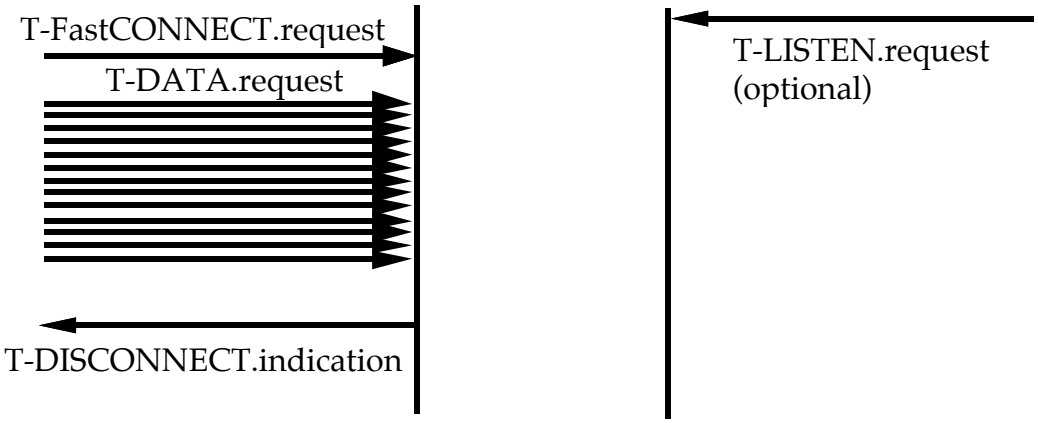

Fig. 4b: Fast connect with listening interaction - Unsuccessful case

2.2 Shortcomings of the Solution Based on the Listening Interaction

At first glance the listening interaction seems to avoid at the called side the delay associated with the decision on the acceptation or refusal of the incoming connection. This is wrong. This delay is still present but it is spent in the TS-provider rather than in the TS-user. Indeed, the delay between the T-FastCONNECT.request and the T-FastCONNECT.indication in fig. 4a is comparable to the delay between the T-FastCONNECT.request and the T-FastCONNECT.response in fig. 2a. In the first case, the TS-provider has to evaluate a certain predicate which is a comparison between the requested QoS from the T-FastCONNECT.request and the required QoS from the T-LISTEN.request. In the second case, the TS-user has to evaluate a similar predicate between the QoS from the T-FastCONNECT.indication and its own requirements. There is no reason to consider that the TS-provider will decide faster than the TS-user.

Therefore, as the delay associated with a decision at the called side is unavoidable, we simply consider this issue as an implementation matter, which has only to do with the design of a 'performant' local service interface. At an abstract level, this is a non-problem that we will not consider any more in the sequel.

Furthermore, the T-LISTEN.request primitive is a very special primitive. First, in contrast to all other TS-primitives, it is not associated with a given connection. It occurs on a TSAP but no TCEP is created until an acceptable T-FastCONNECT.indication is issued on this TSAP. This induces some unclear situations. The following questions give an idea of the additional problems that cannot be ignored.

- Does the TS-user have to re-issue a new T-LISTEN.request after a successful T-FastCONNECT.indication ?
- Is it possible that several T-LISTEN.requests be pending at the same TSAP ? If so, what is the semantics of the "union" of T-LISTEN.requests ?
- How does the TS-user retract a pending T-LISTEN.request ? In particular, how does it identify them if several such requests are pending at the same TSAP ?
- How shall we define the called TS-user requirements to be included in the T-LISTEN.request ? The fact that the T-LISTEN.request is not associated with a given connection adds some non trivial intertwinings between the addressing requirements and the QoS requirements.

All the previous questions indicate that the T-LISTEN.request primitive should not be considered as a TS-primitive, but instead as a local management primitive. Indeed, its sole purpose is *not* end-to-end but purely local. It just tries to improve the design of the local TS-interface. Being local in scope, and of a management nature, one may even wonder if it is correct to consider the T-LISTEN.request as an OSI primitive. To further substantiate this point, it suffices to remark that, whatever the answers to the above questions are, the service does not change conceptually. However, the presence of the T-LISTEN.request requires for completeness that these questions be answered in the service specification. This is unfortunate because abstraction is lost when particular implementations are described explicitly.

In conclusion, the fast connect facility based on the T-LISTEN.request does not solve anything as regards the potential bottleneck at the called side, and leads to an important loss of abstraction.

Furthermore, we realized that neither HSTS nor the variant based on the listening interaction have made consistent hypotheses to address the fast connect facility correctly. Indeed, opposite hypotheses on the behaviour of the TS-provider and on that of the called TS-user have been made: the TS-provider was supposed to be able to accept "immediately" the T-DATA.requests that follow a T-FastCONNECT.request, whereas the called TS-user was *not* supposed to be able to accept "immediately" the T-DATA.indications after a T-FastCONNECT.indication. These two cases are however fully comparable. The TS-provider, like the called TS-user, has to create a context before it can accept incoming TSDUs. It has also to check the QoS parameters of the T-FastCONNECT.request to avoid accepting TSDUs when it knows that the requested QoS cannot be achieved. And so on.

So if we harmonize our hypotheses and abstract away from local and/or implementation-oriented considerations, we come to three possible well-defined

scenarios to express the fast connect service facility: a relaxed 4-primitive establishment and two variants of a 3-primitive establishment. We will present them and compare them in the remaining part of this paper.

3 Fast Connect as a Relaxed 4-primitive Establishment

In this scenario, depicted on figures 5a and 5b, the classical 4-primitive establishment applies, but T-DATA.requests may occur between the T-FastCONNECT.request and the T-FastCONNECT.confirm on the calling side, and *similarly* T-DATA.indications may occur between the T-FastCONNECT.indication and the T-FastCONNECT.response on the called side. This fast connect scenario has been first proposed in the Broadband Transport Service (BTS) of the RACE / CIO project [HeD 92].

If the called TS-user does not accept the connection, it *may* not accept the T-DATA.indications and *shall* use the T-DISCONNECT.request to release the connection. This case is illustrated on figure 5b.

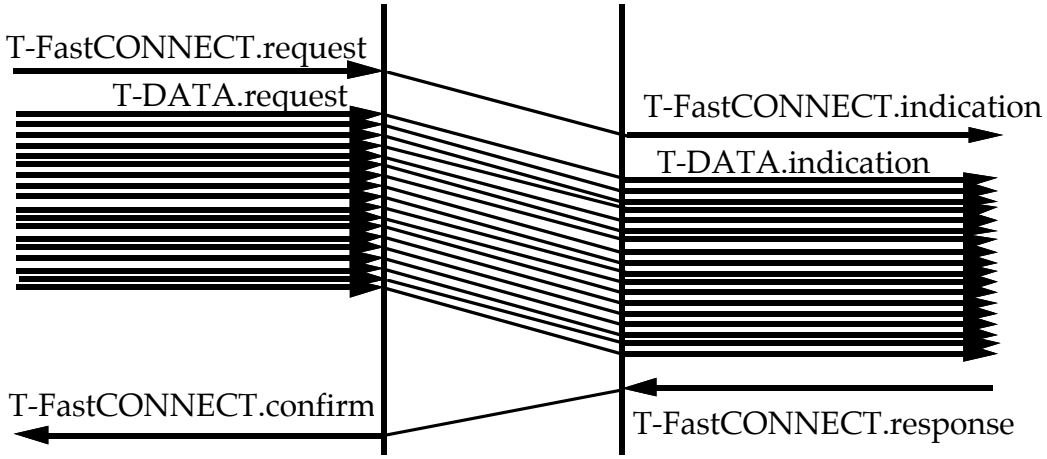

Fig.5a: Fast Connect in BTS - Successful case

As regards the QoS associated with the TS-primitives, BTS does not distinguish between the classical establishment and the fast connect. In both cases, the T-FastCONNECT.response and the T-FastCONNECT.confirm have QoS parameters and nothing precludes the called TS-user to respond with different QoS. By contrast we restrict the negotiation to a take-it-or-leave-it in which the called TS-user cannot change the QoS vector present in the T-FastCONNECT.indication. Therefore, this vector becomes useless in the T-FastCONNECT.response and T-FastCONNECT.confirm primitives and will simply be removed.

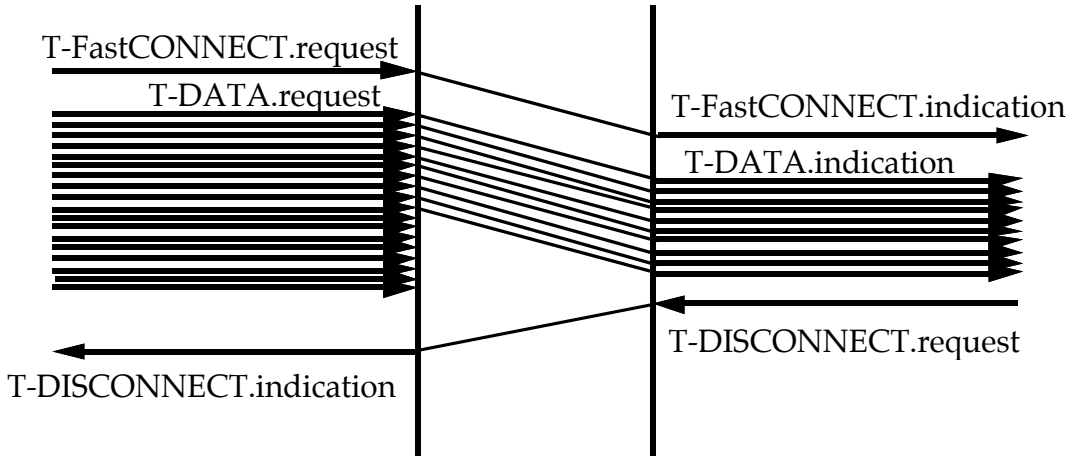

Fig.5b: Fast Connect in BTS - Unsuccessful case

4 Fast Connect as a 3-primitive Establishment

Noticing that the T-FastCONNECT.response has almost no purpose, we may even get rid of this primitive, and thereby propose a 3-primitive establishment as depicted in figures 6a and 6b, if we take the following two-step argument into account:

- The T-FastCONNECT.response in the relaxed 4-primitive establishment is reduced to a very simple interaction because no QoS parameter vector is needed. The same reasoning applies to the T-FastCONNECT.confirm primitive.
- If we want to optimize the successful establishments with respect to the unsuccessful ones, we may consider that a T-FastCONNECT.response is implicit in the T-FastCONNECT.indication, and thereby completely remove the need for this response primitive.

The scenario depicted in figure 6a shows the successful establishment. The refusal scenario is depicted in figure 6b. When the called TS-user decides to refuse the connection, it may not accept the T-DATA.indications and shall use the T-DISCONNECT.request to release the connection.

Note that in this case, the calling TS-user may first receive the T-FastCONNECT.confirm before the T-DISCONNECT.indication. Actually, the refusal scenario per se disappears, but the more general release scenario remains and can be used for this purpose. This is aligned with the idea that such “fast connections” are not supposed to be refused. Acceptation is implicit or taken by default.

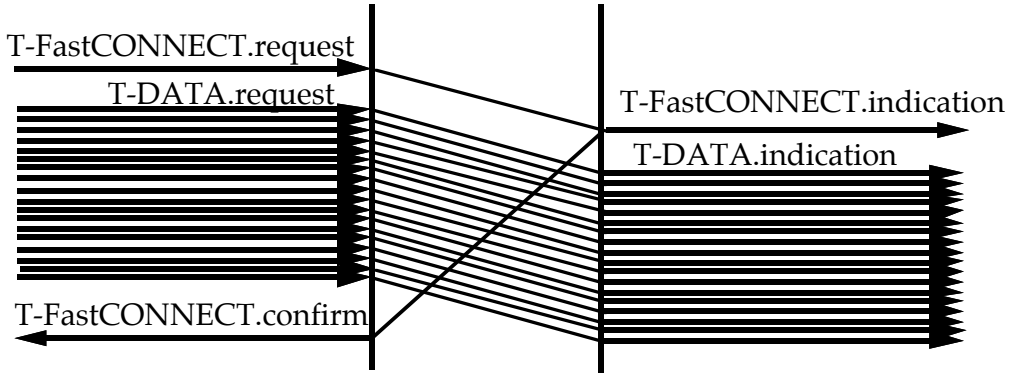

Fig.6a: Fast connect as a 3-primitive establishment - Successful case

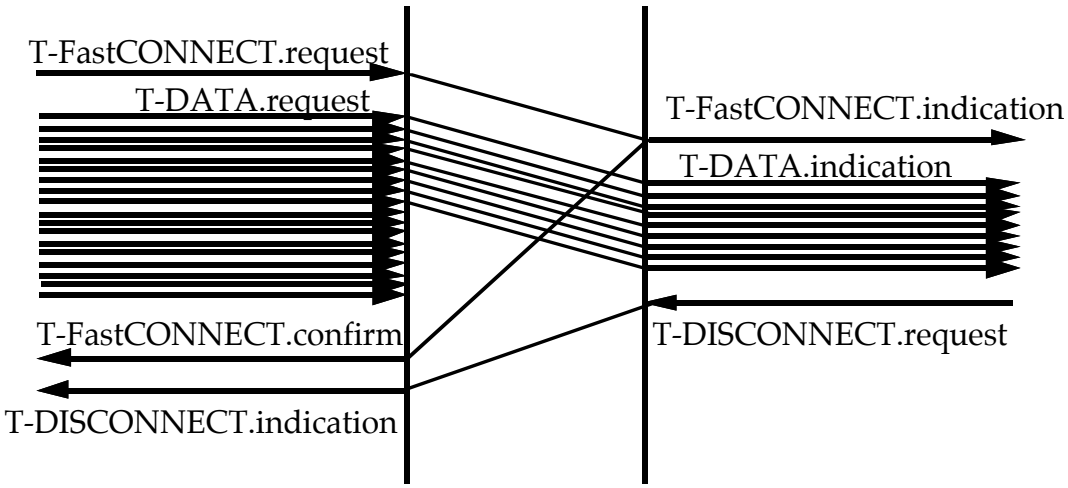

Fig.6b: Fast connect as a 3-primitive establishment - Unsuccessful case

5 A Variant of the 3-primitive Establishment

In the previous 3-primitive establishment, the implicit a priori acceptation of the connection by the called TS-user leads to a less elegant scenario when the called TS-user wants to refuse the connection (which is supposed to be an exception). Indeed the TS-user first accepts the connection as always and then has to disconnect it. This scenario may be improved if the TS-user has a means to inform the TS-provider of its acceptation or refusal *when the T-FastCONNECT.indication occurs*. This will be explained in detail after a reminder of the concept of a *service primitive*.

5.1 The Service Primitive

It is important to keep in mind that a service primitive is an *interaction* between two entities (the service user and the service provider). This interaction occurs at the SAP to which both entities are "attached". This abstraction has several consequences:

- When a service primitive occurs, both entities are aware of it simultaneously.
- A service primitive cannot occur if one entity does not agree on its occurrence. This idea is used in ISO standards to describe interface flow control. For example, when a TS-provider is momentarily blocking the T-DATA.requests, or when a TS-user is blocking the T-DATA.indications.
- When a service primitive occurs, a negotiation of values may take place instead of the more classical value passing. For example, a connection endpoint identifier (cep_id) is negotiated by both entities on the occurrences of a T-CONNECT.request as well as of a T-CONNECT.indication. Of course, the way this negotiation is made is implementation-dependent and is therefore beyond the scope of the standard.

5.2 The New 3-primitive Establishment

Along the same principles, pieces of information may be passed in both directions when a primitive occurs. For example, suppose that a certain parameter of the T-FastCONNECT.indication primitive (denoted 'verdict' for example) is not determined by the TS-provider but instead by the TS-user. By way of this parameter the TS-user informs the TS-provider of its acceptance or refusal of the QoS values proposed unilaterally by the TS-provider. The figures 7a and 7b respectively depict the successful establishment and the refusal of the fast connection.

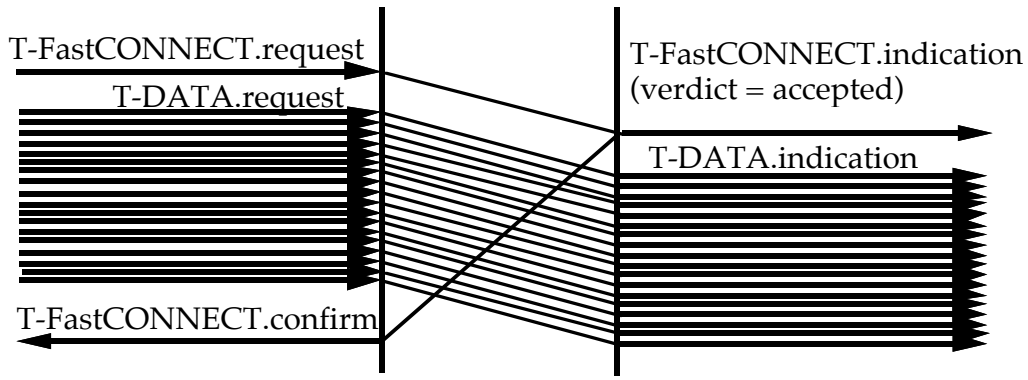

Fig.7a: Fast connect as a new 3-primitive establishment - Successful case

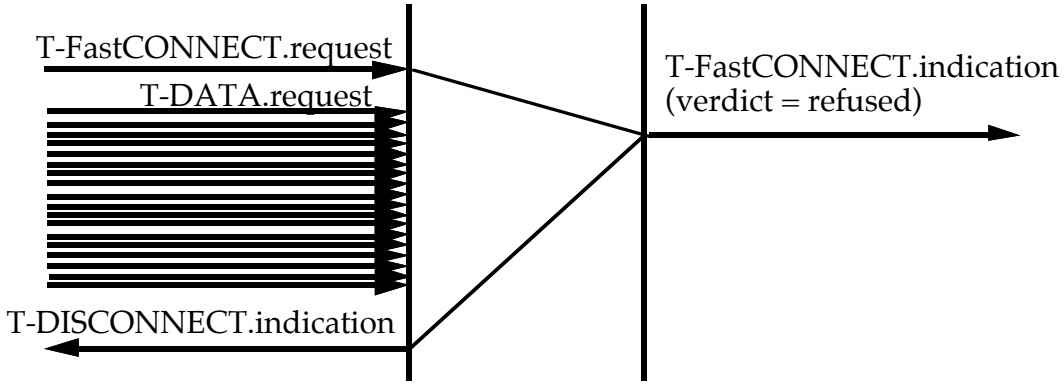

Fig.7b: Fast connect as a new 3-primitive establishment - Unsuccessful case

The verdict parameter determined by the TS-user plays the role of the T-FastCONNECT.response primitive. In other words, two primitives (T-FastCONNECT.indication and T-FastCONNECT.response) are merged into a single *atomic* primitive. This fast response included in the T-FastCONNECT.indication is only interesting if the TS-user has at its disposal (that is in its local context) all the information required to return the verdict immediately. Let us consider three cases:

1) The called TS-user is an application entity which sits directly on top of the TS-provider. In this case the TS-user can obviously determine the verdict by evaluating some predicate based on the other T-FastCONNECT.indication parameters and its local context.
2) The called TS-user is an intermediate entity (e.g. a session entity if a complete OSI stack is present). There are two subcases:
   - This entity has been informed of the acceptance conditions by the above layers. We are back in case 1.
   - This entity has not been informed of the acceptance conditions. It shall always accept the connection and thus fix the verdict to 'accepted'. The decision will be taken in higher layers, but the data flow is not blocked in the transport entity.
3) There may also exist TS-users that are never capable of handling fast connections (e.g. the existing ISO session entity). In this case the TS-user shall always return the verdict "refused".

5.3 An Example of LOTOS Description

For information we provide hereafter a simplified LOTOS description that formalizes the above ideas. It consists of two processes (a TS-provider and a called TS-user) that are synchronized on the gate `dest_tsap` to interact via a T-FastCONNECT.indication primitive. The specification illustrates that the TS-provider determines the calling address and the QoS parameters, whereas the called TS-user determines the acceptation verdict parameter. Depending on the verdict the resulting behaviour of the TS-provider is also described (either it issues a T-FastCONNECT.confirm or a T-DISCONNECT.indication at the `source_tsap`).

```
specification fast_connect_example [source_tsap,dest_tsap] :noexit
behaviour   TS_provider [source_tsap,dest_tsap]
            |[dest_tsap]|
            TS_user [dest_tsap] (...)
where
process TS_provider [source_tsap,dest_tsap] :noexit :=
      source_tsap?tsp1: T_FastCONNECT_request;
      dest_tsap?tsp2: T_FastCONNECT_indication
            [(QoS(tsp2) eq QoS(tsp1))
             and (calling_of(tsp2) eq calling_of(tsp1))
             and (dest_sap is_valid_value_for called_of(tsp1))];
      (([verdict(tsp2) = accepted] ->
            source_tsap!T_FastCONNECT_confirm;
            Data_transfer [source_tsap,dest_tsap])
       []
       ([verdict(tsp2) = refused] ->
            source_tsap!T_DISCONNECT_indication;
            stop))
endproc (* TS_provider *)
process TS_user [dest_tsap] (u: user_context) :noexit :=
      dest_tsap?tsp: T_FastCONNECT_indication
          [(Acceptance_predicate(calling_of(tsp),QoS(tsp),u)
                implies (verdict(tsp) eq accepted))];
      Receive_data [dest_tsap]
      []
      dest_tsap?tsp: T_FastCONNECT_indication
           [(not (Acceptance_predicate(calling_of(tsp),QoS(tsp),u))
                implies (verdict(tsp) eq refused))];
      stop
endproc (* TS_user *)
endspec
```

5.4 The Responding Address Issue

There remains a pending issue regarding the 3-primitive scenario. How can the TS-provider fill in the responding address field of the T-FastCONNECT.confirm primitive if no T-FastCONNECT.response primitive exists to provide this responding address to the TS-provider ?

According to us, the TSAP at which the T-FastCONNECT.indication primitive actually occurred is known by the TS-provider and the responding address *is precisely* the address of this TSAP. So, the TS-provider knows the responding address, and a T-FastCONNECT.response is not necessary to provide this address.

By contrast, the called address that is present in the T-FastCONNECT.indication primitive does *not* in general identify the TSAP at which the T-FastCONNECT.indication is occurring. Two obvious counter-examples are when the called address is a generic or a group address.

6 Conclusion

A fast connect facility is incompatible with a full negotiation. Instead it can only provide the limited "take-it-or-leave-it" negotiation. Moreover the refusal of a fast connection has worse consequences than the refusal of a classical connection due to the resources wasted in the TS-provider. For these reasons, we argued on the complementarity of the fast connect facility and the existing connection-mode TC establishment facility.

Several models of a fast connect facility have been presented and the last three ones have been retained. They all are defensible but, according to us, the last one is the best one. It combines soundness with simplicity and abstraction: 3 primitives for a take-it-or-leave-it negotiation is necessary and sufficient. It also preserves all the nice properties of a classical connection: the called TS-user can refuse the connection (not true in the first version of the 3-primitive establishment), and no T-DATA.indications occur at the called side when the connection is refused (not true in BTS and not applicable in the first 3-primitive establishment).

Acknowledgements

We would like to thank Yves Baguette and Olivier Bonaventure for many valuable discussions and for their in-depth reviewing of this paper.

References

[Dan 92] A. Danthine, **A New Transport Protocol for the Broadband Environment**, in: A. Casaca, ed., Broadband Communications, Elsevier Science Publishers (North-Holland), Amsterdam, 1992, 337-360.

[ISO HSTS] Source: USA, Proposed working draft for high-speed transport service definition, ISO/IEC JTC1/SC6/N7070, 1991-11-28.

[ISO 8072] ISO-TC97/SC6/WG4, Information Processing Systems - Open Systems Interconnection - Transport Service Definition, IS 8072, 1986-06-15.

[DBL 92] A. Danthine, Y. Baguette, G. Leduc, **Issues Surrounding the Specification of High-Speed Transport Service and Protocol**, Rept. no. OSI95/ULg/A/15/TR/P/V2, University of Liège, Institut Montefiore B28, B-4000 Liège, Belgique, Jan. 1992, also in: ISO/IEC JTC1/SC6/N7312, 11-05-1992.

[HeD 92] L. Henckel, S. Damaskos, **Multimedia Communication Platform: Specification of the Broadband Transport Service**, Rept. no. R2060/GMD/CIO/DS/P/001/b1, GMD-FOKUS, Hardenbergplatz 2, D-1000 Berlin 12, Germany, Dec. 1992.

[PEI XTP] Protocol Engines Inc., **XTP Protocol Definition - Revision 3.6**, PEI 92-10, 1900 State Street, Santa Barbara, CA 93101, Jan. 1992.