Axon: Application-Oriented Lightweight Transport Protocol Design

Authors: James P.G. Sterbenz and Gurudatta M. Parulkar

This paper describes the application-oriented lightweight transport protocol for object transfer (ALTP-OT) in the Axon host communication architecture for distributed applications. The Axon Project is investigating an integrated design of host architecture, operating systems, and communication protocols to allow the utilization of the high band-width provided by the next generation of communication networks. ALTP-OT provides the end-to-end transport of segment and message objects for interprocess communication across a very high speed internetwork, supporting demanding applications such as scientific visualization and imaging. ALTP-OT uses rate-based flow control specifically oriented to the transfer of objects directly between application memory spaces.

This document is intended to present the design of ALTP-OT, rather than serve as a complete specification and implementation report. It should be treated as a request for comments, and will be periodically updated to reflect comments from the research community and progress on Axon design and prototype implementation. Last revision April 5, 1990.

... Read complete abstract on page 2.
Axon: Application-Oriented Lightweight Transport Protocol Design

**Complete Abstract:**

This paper describes the application-oriented lightweight transport protocol for object transfer (ALTP-OT) in the Axon host communication architecture for distributed applications. The Axon Project is investigating an integrated design of host architecture, operating systems, and communication protocols to allow the utilization of the high band-width provided by the next generation of communication networks. ALTP-OT provides the end-to-end transport of segment and message objects for interprocess communication across a very high speed internetwork, supporting demanding applications such as scientific visualization and imaging. ALTP-OT uses rate-based flow control specifically oriented to the transfer of objects directly between application memory spaces.

This document is intended to present the design of ALTP-OT, rather than serve as a complete specification and implementation report. It should be treated as a request for comments, and will be periodically updated to reflect comments from the research community and progress on Axon design and prototype implementation. Last revision April 5, 1990.

This technical report is available at Washington University Open Scholarship: [http://openscholarship.wustl.edu/cse_research/727](http://openscholarship.wustl.edu/cse_research/727)
AXON: APPLICATION-ORIENTED LIGHTWEIGHT TRANSPORT PROTOCOL DESIGN

James P. G. Sterbenz
Gurudatta M. Parulkar

WUCS-89-14

September 1989

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
Saint Louis, MO  63130-4899

ABSTRACT

This paper describes the design of application-oriented lightweight transport protocol for object transfer (ALTP-OT) in the Axon host communication architecture for distributed applications. The Axon project is investigating an integrated design of host architecture, operating systems, and communication protocols to allow the utilization of the high bandwidth provided by the next generation of communication networks. ALTP-OT provides the end-to-end transport of segment and message objects for interprocess communication across a very high speed internetwork, supporting demanding applications such as scientific visualization and imaging. ALTP-OT uses rate-based flow control on a connection oriented internetwork substrate, and simplified error control specifically oriented to the transfer of objects directly between application memory spaces.

This document is intended to present the design of ALTP-OT, rather than serve as a complete specification and implementation report. It should be treated as a request for comments, and will be periodically updated to reflect comments from the research community and progress on Axon design and prototype implementation. Last revision April 5, 1990.

James Sterbenz is on leave of absense from IBM Corporation at Washington University in St. Louis.
AXON: APPLICATION-ORIENTED LIGHTWEIGHT TRANSPORT PROTOCOL DESIGN

James P. G. Sterbenz
jps@wucs1.wustl.edu
+1 314 725 4203

Gurudatta M. Parulkar
guru@flora.wustl.edu
+1 314 889 4621

1. Introduction

We have proposed a new communication architecture for distributed applications called Axon [StPa89a, StPa90b] whose principal motivation is to provide high performance IPC (interprocess communication) in the future generation of internetwork, which we refer to as the very high speed internetwork (VHSI) [Pa90]. The significant features of Axon are: [1] an integrated design of host and network interface architecture, operating systems, and communication protocols; [2] a network virtual storage facility which includes support for virtual shared memory across networks [StPa89b, StPa90a, StPa90c]; [3] a high performance, lightweight object transport facility which can be used by both message passing and shared memory mechanisms; [4] a pipelined network interface which can provide a high bandwidth low latency path directly between the VHSI and host memory [St90].

This paper describes the design of the Axon application-oriented lightweight transport protocol for object transfer (ALTP-OT), and is organised as follows: Section 2 provides a brief overview of the Axon host communications architecture for distributed systems. Section 3 describes the ALTP-OT design, including flow and error control strategies, and transport operations. Section 4 describes design and policy alternatives for flow and error control. Section 5 describes other relevant work, and Section 6 is the conclusion. Appendix A contains a detailed description of the transport operations.

2. Overview of the Axon Architecture

This section provides an introduction to the Axon architecture. First, IPC primitives are discussed within the framework of the VHSI environment. Then, a brief description is presented for significant Axon architectural components: system level IPC support, the transport protocol (ALTP-OT), host and network interface architecture, and the communications processor (CMP). Finally, an example of the interaction of these components is given by describing the transport of a segment across the VHSI.
2.1. IPC in the Axon architecture

A logical view of the Axon protocol hierarchy is presented in Figure 1. It is important to note that this is a logical view of functionality only, and does not imply that strict layering (in the ISO-OSI sense) is being adhered to.

IPC is supported with shared variable read/write \((r, w)\) and message passing send/receive \((s, r)\) primitives. Axon supports a more general form of RPC, in which the code and data segments can be located on arbitrary and independent hosts, with execution specified for an arbitrary host. This is referred to as generalised remote procedure call (GRPC). Conventional remote procedure call (RPC) [BiNe84] is thus a restricted form of GRPC. Additionally, the special demands of high performance visualisation and imaging applications motivate an additional IPC paradigm. Axon provides mechanisms to transfer segment streams at high bandwidth with low setup overhead. The performance advantage is that a single transport level operation performs the request for all of the segments, and each segment can be transmitted when ready without the latency of a request. GRPC and segment streaming are described in more detail in [StPa89b].

The shared memory mechanism for IPC across the VHSI is implemented by NVS (network virtual storage). This can be utilised by an application either by referencing segments that are non-local, through the facilities provided by GRPC, or by the use of segment streaming. Support for message passing IPC is provided by a network message passing interface (NMP), which invokes the appropriate message transfer ALTP-OT calls.

The transport mechanism is provided by an application-oriented lightweight transport protocol (ALTP) tailored for object transfer, called ALTP-OT. ALTP-OT resides as a set of software modules in the host system, and as hardware in the CMP (communications processor). The underlying internet/network layer of function is provided by a multipoint congrum-oriented high performance internet protocol\(^1\) (MCHIP) [Pa90, MaPa89], and network access protocols (NAP).

2.2. System level IPC support and NVS

The system level support for the various application level IPC paradigms is provided by NVS and NMP. NVS is the system shared memory interface for shared variables, GRPC, and segment streaming. NMP is the system level message passing interface. NMP performs a relatively straight-forward transformation of program \((send, receive)\) primitives to corresponding ALTP-OT message-object transfer calls \((send-message, receive-message)\). The remainder of this section will provide an introduction to NVS.

\(^1\)The definition of a congrum will be deferred until section §3.3, but for the present it may be considered a soft connection
NVS extends the typical virtual storage mechanisms to include systems throughout the VHSI. A segmented programming model is used, with underlying paging to facilitate storage management, in a manner similar to the Multics operating system [Be72, Or72]. Details on segmented paged virtual storage are provided in [StPa89b]; this section only briefly discusses NVS extensions.

NVS extensions allow segments to be addressed when resident on a non-local host. This is accomplished by either including a host id. field in the virtual address (network virtual address), or the host id. in the segment descriptor table entry (local virtual address). When a segment fault occurs for a nonlocal segment (indicated in the segment descriptor), the dynamic address translation facility invokes ALTP-OT to get a copy of the segment from the appropriate system. When the segment is returned, the appropriate page and segment descriptor presence bits are set so that program execution can resume with the normal fault recovery mechanisms.

The local storage management data structures are extended to allow the addressing of segments on other hosts. This is accomplished by adding a host id. field to the known segment table, which holds the symbolic segment bindings. This is an index into the per process known host table, which holds the symbolic host name to address/path bindings. This binding is resolved by searching the host address table for each host, which gets its binding by invoking an internet name server using the host name database. There are also tables to assist in n-way IPC using multipoint connections. Depending on the method used for network-to-host object mapping, a packet presence bit vector may reside in page descriptor table entries.

NVS in Axon also involves extensions and additions to storage management policies. The replacement policy is affected as a result of pages from remote segments in the locality set, which requires redefinition of the working set to account for non-local segments. An entirely new policy, the remote placement policy, is used to determine where remote segments are placed while being used by the local system. These include real store (RS), auxiliary store (AS), a combination (RAS), or frame buffer (FB) placement, with a number of sub-policy options (swappable, nailed, etc.). The NVS mechanisms, policies, and data structures are described in detail in [StPa89b, StPa90a, StPa90c].

2.3. Transport protocol

At the transport level, applications using the VHSI are supported by a set of simple ALTPs (application-oriented lightweight transport protocols) for various classes of applications. These transport protocols can have their critical path function implemented in VHSI hardware. The critical path consists of the data path and routine control functions allowing data to flow once a transport operation has begun. By optimising the critical path function, and by processing multiple packets in a single transport level operation, the per packet processing can be performed in real time, at the full VHSI data rate. For the protocol to be efficiently implemented in hardware, the protocol, hardware design, and host operating system should be well integrated. ALTPs can be optimised to provide the kind of performance guarantees and functionality the specific applications need.

The ALTP type that will be described in this paper is designed to support IPC by the transfer of objects, with primary consideration in supporting NVS segments, and is referred to as ALTP-OT. ALTP-OT uses rate based flow control, where the rate specification consists only of parameters important to IPC, and efficient error control streamlined to include only what is necessary for object transfer. ALTP-OT is discussed in detail in Section 3.

2.4. Host and network interface architecture

High performance computer systems typically consist of one or more central processors (CPU), which communicate with memory banks and I/O processors through an interconnection network. Communication is handled by front-end communications processors or network interfaces, which use the I/O
interface to the host system. In the VHSI environment, it is necessary to provide high bandwidth low latency data paths directly to memory, motivating new host architectures. Two host architecture configurations are defined for the Axon architecture.

**Interconnect interface architecture (IIA).** The first host configuration gives the CMP (communications processor) a relationship to the system similar to that of I/O processors, thus interfacing the CMP directly into the processor-memory interconnection network. This is referred to as **interconnect interface architecture (IIA)**. In addition, an interconnection between CMPs and I/O processors should be provided to allow direct, high-speed transfers between the network and I/O controllers or devices (which provides the path to auxiliary or secondary storage). Axon only imposes the requirement that the interconnection be rich enough to allow the added CMP connections, and high enough performance to sustain the additional VHSI communication traffic.

**Memory interface architecture (MIA).** The second host configuration interfaces the CMP to a special multi-ported communications memory module (CMM), which is referred to as **memory interface architecture (MIA)**. In this case, the CMM has a conventional random access port which appears like any other memory bank to the processor-memory interconnect. The other ports are high-speed serial access interfaces to the CMP. The design of the CMM is similar in concept to VRAM (video-RAM), and requires that the physical address space of the system be partitioned between conventional and communications memory.

**Host–network interface architecture.** A block diagram of an MIA host interface is presented in Figure 2.

![Figure 2: Axon Host–Network Interface (MIA)](image)

The ALT–OT critical path, consisting of the data path and per packet processing, is implemented in the CMP (communications processor). The CMP consists of datapath (CMP_d) and control (CMP_c) portions. The CMP datapath interfaces to the VHSI optical links and the serial ports of the CMM (communications memory module), and performs such functions as encryption/decryption and format conversion (encode/decode). The CMP control functions are those directly related to the datapath
such as header build/decode, checksum generate/compare, rate specification timing, as well as the per packet congram multiplexing and control.

The CMM is a multiported memory, with serial ports connected to the CMP transmitting and receiving data paths, and the random access port available to the host CPU for program execution.

A high performance microprocessor, the CMP assist processor (CAP), performs functions that are not part of the critical path, but require high performance that would be inadequately provided by the host CPU and would adversely impact the performance of other host processes. Examples include the packet arrival to page presence mapping, and packet retransmission timer management functions.

The host CPU is responsible for link/segment/page fault handling (NVS), and per congram functions. More information on Axon host architecture and the network interface is presented in [St00].

2.5. Communications processor (CMP)

To perform critical path functions at full VHSI data rate with no packet buffering, the CMP is organised as a pipeline, dynamically reconfigurable based on the ALTP type and options for a particular congram. The CMP block diagram (for an MIA host) is presented in Figure 3.

![Figure 3: CMP Block Diagram](image)

The transmit data pipe and receive data pipe are the main data paths of the CMP, and perform data encryption/decryption and format transformation. Data is clocked out the transmit data pipe from the CMM by the rate control logic, which is responsible for adhering to the rate specification for each congram.
Congram multiplexing is handled by the *mux control* logic, with the *congrame state registers* containing the state information for each congrame, allowing a fast hardware context switch of the CMP based on the connection/congrame id. of each packet. The arrival of each packet is tracked by the *packet presence logic* which is responsible for determining when entire pages and segments have arrived, so that the appropriate page descriptor table and segment descriptor table presence bits can be set and the host program dispatched. The error *control* logic is responsible for recording missing/corrupted packets, and generating the appropriate retransmission requests.

Associated with the transmit data pipe, the *header build* logic constructs the appropriate header information from a template in the CMM, and inserts the proper packet identifiers (eq) and index (ijk). The *checksum generate* logic generates the checksum as the packet passes through the pipeline, and inserts it into the packet trailer. Associated with the receive data pipe, the *header decode* logic decodes the header to determine the connection and request ids (eq) for CMP configuration. It also determines the packet address in CMM from the packet index (ijk) and the base address of the page from the corresponding congrame state register. The *checksum check* logic sums the packet as it passes through the pipe, and compares against the checksum field in the packet trailer. If the packet has been corrupted, it is discarded by clearing the appropriate packet presence state. Greater detail on the CMP design is presented in [St90].

2.6. Summary of segment transfer

ALTP-OT operation in the Axon architecture will be introduced by the description of a segment transfer in response to a NVS remote segment fault. Explicit references to Figure 4 in this discussion are enclosed in brackets: [ ]. Note that certain assumptions and policy choices have been made for clarity in this discussion.

An executing process has associated with it a virtual address space, which is a subset of the segments available to the user which owns the process. When a process refers to a remote segment, either explicitly by name, or via a GRPC, the appropriate segments must be transported from the desired system. The segment is located, either by an explicit reference to the segment and host name, or by resolution of the host name associated with the segment capability stored in the user context directory (UDIR). The first time a segment is referred to symbolically, a *link fault* resolves the name and location, and adds the segment binding to the KST (known segment table), and host name binding to the KHT (known host table) for the process. This allows further *symbolic* references to avoid the overhead of searching the user context for segment attributes. In addition, an entry is added to the process SDT (segment descriptor table), which contains the process specific attributes of the segment. An entry is added to the system AST (active segment table), which contains the attributes of the segment common to all processes sharing the segment, if the segment is not already in use by another process. The mechanism for sharing is to have the SDT entries of multiple processes pointing to a single AST descriptor, which refers to a single instantiation of the segment. When a remote segment transfer is necessary, the transport mechanism is accomplished by ALTP-OT.

The critical path function of ALTP-OT is implemented in the CMP hardware [ALTP-critical], and includes the data path and routine control functions (error and flow control). The non-critical part resides in the systems software on the host (or CMP assist processor [St90]), and is tightly integrated with the host architecture and operating system [ALTP-host]. In particular, the host portion of ALTP-OT must have direct access to operating system services such as the scheduler [OS-sched] through lightweight system calls, and be able to manipulate virtual storage management data structures [VS-tables].

The remote transfer is initiated by an ALTP-OT operation such as get-segment, which retrieves a segment from a remote host for local use. This requires a connection between the two hosts, thus
Figure 4: Interaction of Axon Components

ALTP-OT issues an open call to MCHIP which establishes the connection if not already present from a previous call. In addition, the CMP data pipeline is configured appropriately for the connection. ALTP-OT then sends the get-segment control packet out the VHSI link interface and through the internet, using the established connection.

At the remote end, the CMP receives and decodes the control packet at the internet link interface, and passes it to the host operating system. The normal mechanisms for locating the segment and authenticating the request are used. When the segment is found, locks are set (if necessary), and a copy of the segment is returned to the requesting host in a super-packet along the same connection. The data packets consist of fragments from each page of the segment, with an integral number of packets per page. Note that if multiple segments are defined within a segment access group, all of them are returned in a single super-packet. Thus the unit of structure is a superpacket \( s \) consisting of a segment group \( g \) of segments \( [s] \) of pages \( [p] \) of packets \( [\pi] \).

At the local end, storage has been allocated for the returning segment(s), based on the estimated segment size \( |s| \) and remote segment placement policy in use (either [real store frames] or [aux store slots]). The data packets contain the actual segment size \( |s| \), allowing adjustments to be made in the estimated storage allocation. The header of each data packet also indicates the packet and page (and segment) number \((i,j,k)\), as well as the connection and request ids. Since the connection has been established, the CMP pipeline configured, and storage allocated, packets are placed directly in storage according to the remote placement policy; no buffering of the data by the CMP takes place; the order of packet arrival is not significant (sequence by placement), and there is no involvement of MCHIP or the host software portion of ALTP-OT. The structure of data between the CMP and target memory is the page \([p']\). Note that the peer-to-peer connection between ALTPs is physical, without
the strict calling and data copying involved in the OSI or other layered models, and there is none of
the overhead associated with multiple packet encapsulation/decapsulation between layers.

When certain events occur, the CMP issues a signal to the host software portion of ALT-P-OT. For
example, each time all of the packets of a given page have been received, the presence bit in the
PDT (page descriptor table) must be set, and a lightweight system call must indicate to the low level
scheduler that the process can be dispatched, as in the standard page fault recovery mechanism.
When the entire segment has been received, the presence bit in the AST (active segment table) is
set, and the ALT-P-OT connection idles until the process ends, or an explicit leave-ipc is issued.

3. Transport Protocol

This section describes the design of ALT-P-OT (application-oriented lightweight transport protocol
for object transfer) used by IPC across the VHSI in Axon. The various ALT-P-OT flow and error
control mechanisms are described. Detailed description of the ALT-P-OT operations is provided in
Appendix A.

3.1. Taxonomy

A taxonomy of transport protocols based on the scope of their functionality is presented in Figure 5.

```
  general purpose

  functionally partitioned

  application oriented
    AOTP-OT  AOTP-V  AOTP-RL  AOTP-VD

  special purpose
    SPTP_1   SPTP_2   SPTP_n
```

Figure 5: Transport Protocol Taxonomy

Most current transport protocols (including TCP, X.25, and SNA) are general purpose, providing
complete flow and error control to all applications. This results in complexity of implementation
and operational overhead that is not necessary for all applications. It may be possible to func-
tionally partition a single transport protocol to provide only the functionality needed for various
classes of applications, while still allowing the use of a single protocol. By adjusting parameters
the appropriate functionality would be included. A similar strategy that does not require that a
single transport protocol serve all applications, is to have a small set of application-oriented trans-
port protocols (AOTPs). A possible set might consist of application oriented transport protocols for
object transfer (AOTP-OT), for remote login (AOTP-RL), for voice (AOTP-V), and for (entertainment)
video distribution (AOTP-VD). Finally, it is possible to have a large set of special purpose transport protocols (SPTPS), designed for very specific applications (such as for bulk data transfer, reservation transaction processing, etc).

Somewhat orthogonal to the scope of the transport protocol is the simplicity of design and efficiency of operation. A protocol that is simple, streamlined, and efficient is referred to as a lightweight protocol. Note that while it is possible to design even a general purpose protocol to be lightweight to some degree, it is much easier to do so with an application oriented protocol that can efficiently serve the corresponding application class with the appropriate error and flow control mechanisms. This is the approach taken in Axon with ALTPS (application-oriented lightweight transport protocols). A similar result could be possible with a functionally partitioned protocol, but it may be difficult in practice to isolate the functional requirements of different application classes, while maintaining optimal performance characteristics for each.

In the Axon transport level, IPC across the VHSI is supported by ALTP-OT, which has its critical path function implemented in VLSI hardware, and is optimised to provide performance guarantees and functionality for object transfer. ALTP-OT design will now be described.

3.2. Packet structure and format

Information is transferred throughout the internetwork in packets. A structured group of packets corresponding to a single ALTP-OT semantic action is a super-packet, consisting of an initial control packet (which may also contain a small amount of data), and optionally followed by data packets. Bits in the packet header indicate whether the packet is control (at MCHIP or ALTP level) or data. ALTP-OT control packets require processing by the ALTP-OT logic in the CMP (communications processor), as well as by the host system hardware and operating system. Data packets require considerably less processing, all of which can be done in real time by the CMP hardware. The format of a data packet is presented in Figure 6. Each data packet corresponds to a fragment \( \pi_k \) of a page \( p_j \) of a segment \( s_k \) of a segment-group \( g \), which are part of the superpacket \( \sigma \). In the case of a video-graphics segment a page corresponds to a scanline, a segment to a frame, and a segment group to a complete image. The connection-id. and request-id. fields of the packet header allows the CMP to associate data packets with connections set up by the corresponding control packet. Control packets have fields that are dependent on the type of operation.

<table>
<thead>
<tr>
<th>MCHIP</th>
<th>connid</th>
<th>ALTP</th>
<th>recid</th>
<th>segment (frame)</th>
<th>page (scanline)</th>
<th>pkt</th>
<th>data</th>
<th>chksum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>c</td>
<td></td>
<td>q</td>
<td>(</td>
<td>g</td>
<td>)</td>
<td>( k )</td>
<td>(</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Figure 6: ALTP-OT Data Packet Format

The benefits of this packet/super-packet hierarchy is that most of the usual per packet control processing is only performed per super-packet in Axon. A structuring of the data that is recognised by ALTP-OT allows the per packet processing to be simplified to the extent that VLSI implementation is reasonable and efficient. In addition, since ALTP-OT is tightly integrated with the host systems software, it has direct access to the appropriate operating system facilities (via lightweight system calls) and data structures, resulting in efficient coordination between ALTP-OT and conventional operating system operations.
3.3. Flow control

When ALTP-OT opens a connection, it specifies attributes of the connection in terms of parameters such as average and peak bandwidth, and a factor reflecting the burstiness of the transmission. These parameters can be translated into buffer requirements, based on a rate between the average and peak specifications [Ak87]. Since the connection set up is end-to-end, all the intermediate systems, including various packet switches and gateways, as well as the endpoint hosts that the connection goes through can make appropriate buffer and resource reservations [Pa90]. The rate specification will have to be negotiated between ALTP-OT and the internetwork/network layers to ensure that the requested rate does not exceed the capacity of internal network nodes (packet switches, gateways, and subnetworks). As a result, as long as both ends transmit subject to the rate specification, the probability of packet loss due to buffer overruns is very low. Note that due to the high bandwidth-x-latency product in the VHSI environment, dynamic adjustments to the rate specification should be infrequent, reflecting long term changes in application behavior.

It is assumed that the internet level has the functionality to support connections with specified bandwidth requirements, and furthermore, that the probability of packet loss, errors, and resequencing is low enough to design the critical path with the assumption that handling such events is the exceptional case. This is referred to as quasi-reliability.

The version of ALTP-OT and CMP described in this report is designed to be used on top of MCHIP (multipoint congram-oriented high performance internetwork protocol) [MaPa88]. MCHIP supports a connection abstraction referred to as a congram, which possesses the best characteristics of both (soft) connections and datagrams. A congram is similar to a soft connection with the added attributes of rapid setup (using pre-existing perpetual internet congrams), and survivability in the face of network failures (due to the ability to rapidly reconfigure). The quasi-reliability assumption of ALTP-OT corresponds to MCHIP plesio-reliability\(^4\), but is somewhat less strict. The MCHIP will guarantee a plesio-reliable connection through the use of rate-based connection-oriented fast packet switches (such as the BPN [Tu86]), and rate control mechanisms in gateways and subnet boundaries.

The only explicit flow control exercised by ALTP-OT is the control of the CMP data transmission rate to correspond to the rate specification. The benefit of the ALTP approach is manifest in that only attributes significant for object transfer need to be considered; the rate specification for a general purpose transport protocol could be considerably more complex. Additionally, due to the end-to-end quasi-reliability and resource reservation functionality of MCHIP, ALTPs need not be concerned with congestion control.

3.4. Error control

In the VHSI environment, error control is performed, as much as possible, on an end-to-end basis, and is decoupled from flow (rate) control. This is justified by the assumption of quasi-reliability. To allow for simple error control which can be efficiently implemented in VHSI hardware, the error control scheme is designed for the particular ALTP using a connection. The implication to the CMP design is that the error control modules are either designed for a particular ALTP, or when practical built from a generic set of control modules connected and configured appropriately.

\(^4\) plesio comes from the Greek πλησιος meaning almost
The **ALTP-OT** error control packet handling cases are:

- duplicate packets are discarded
- corrupted packets are discarded, and selective retransmission requests are made
- missing packets are detected by the expiration of a timer and selective retransmission requests are made (if the original then arrives, the retransmitted packet is treated as a duplicate and discarded)
- packet arrival sequence is irrelevant; packets do not need to be resequenced since they are placed directly into the proper location of the target store – this is referred to as *sequence by placement*.

Note that due to the orientation of **ALTP-OT** to object transfer, the handling of duplicate and out-of-sequence packets is considerably simpler and more efficient than would be the case for a general purpose transport protocol. In particular, the latency and buffer space associated with packet resequencing are not required with **ALTP-OT**.

A significant aspect of the error control of **ALTP-OT** remains in the manner that retransmission requests are made for missing and corrupted packets. Alternative strategies are described in Section 4.2.

### 3.5. **ALTP-OT** operations

Some **ALTP-OT** operations are initiated by system calls or operating system requests, such as for object transfer. Others are initiated by **ALTP-OT** internally, such as for packet retransmission in response to errors. A detailed description of the **ALTP-OT** operations is given in Appendix A; a brief summary follows:

<table>
<thead>
<tr>
<th>Connection management</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>join-ipc</td>
<td>join or establish multiway IPC connection</td>
</tr>
<tr>
<td>respecify-rate</td>
<td>alter rate specification for existing connection</td>
</tr>
<tr>
<td>leave-ipc</td>
<td>leave or terminate multiway IPC connection</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object receive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>get-segment</td>
<td>obtain copy of named segment from specified host</td>
</tr>
<tr>
<td>acquire-segment</td>
<td>obtain access to named segment for get-page</td>
</tr>
<tr>
<td>get-page</td>
<td>obtain copy of page from acquired segment</td>
</tr>
<tr>
<td>get-copy</td>
<td>obtain a permanent copy of segment from specified host</td>
</tr>
<tr>
<td>get-stream</td>
<td>receive segment stream from specified host</td>
</tr>
<tr>
<td>receive-message</td>
<td>receive IPC message</td>
</tr>
<tr>
<td>retransmit-packets</td>
<td>request selective packet retransmission</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object transmit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>release-segment</td>
<td>release or return local segment copy (after get/acquire-segment)</td>
</tr>
<tr>
<td>release-page</td>
<td>release or return local copy of page (after get-page)</td>
</tr>
<tr>
<td>remote-execute</td>
<td>initiate execution of process on specified host</td>
</tr>
<tr>
<td>send-copy</td>
<td>send a permanent copy of segment to specified host</td>
</tr>
<tr>
<td>send-stream</td>
<td>transmit segment stream</td>
</tr>
<tr>
<td>send-message</td>
<td>send IPC message</td>
</tr>
<tr>
<td>invalidate-segment</td>
<td>invalidate segment copy on another host</td>
</tr>
</tbody>
</table>
4. Transport Protocol Policies

This section describes policy choices available in the ALTP-OT flow and error control mechanisms. Due to the orientation of ALTP-OT to object transfer applications and the super-packet structure, flow and error control decisions can be made to optimise performance based on application characteristics.

4.1. Flow (rate) control

The model for flow control is based on bandwidth rate specification, providing low packet loss rates due to the quasi-reliable VHSI substrate which is able to make resource reservations.

The rate specification that an application requests for the establishment of a connection is a vector such as \( r = [\text{peak-rate}, \text{average-rate}, \text{burst-factor}] \). For ALTP-OT the average-rate applies to the overall super-packet transmission (segment or segment group), with individual pages transmitted at the peak-rate and the burst-factor constraining the inter-page transmission gap.

Applications request bandwidth with two rate specification vectors. The desired rate \( r_{\text{max}} \) is the maximum rate that the application wants to utilise. The minimum acceptable rate \( r_{\text{min}} \) specifies the minimum bandwidth that the application will accept to use the requested connection. MCHIP will attempt to set up the connection using \( r_{\text{max}} \). If this is not possible due to VHSI link capacity and loading, the maximum available rate \( r_{\text{net}} \) will be will be used for the connection, as long as \( r_{\text{net}} > r_{\text{min}} \). If a connection cannot be established with at least \( r_{\text{min}} \) bandwidth, the connection will be denied by MCHIP.

Even though the VHSI can support a connection at \( r_{\text{net}} \), considering the loading of the host system, the actual usable rate to a given application may be smaller: \( r_{\text{host}} \). Since the cost of the connection [8] is a function of the reserved bandwidth, the application may not desire to reserve bandwidth in excess of \( r_{\text{host}} \). By monitoring the host system loading, ALTP-OT can estimate \( r_{\text{host}} \). When an application requests a connection, it may optionally request that \( r_{\text{host}} \) be used to bound the requested bandwidth. In this case ALTP-OT will use \( \max(r_{\text{host}}, r_{\text{max}}) \) as the desired rate in the open-con request to MCHIP.

4.2. Error control

In ALTP-OT duplicate packets are discarded, the order of packet arrival is insignificant, and missing packets are selectively retransmitted. Two conditions can indicate that a packet retransmission request should be made: a received packet which has been corrupted or the timeout of an expected packet.

Several issues in the error control mechanism can be considered: location of retransmission requests, granularity of retransmission and timer values, retransmission fetch policy, and preemption by the retransmission.

Location. Requests for retransmission can originate from either the receiving end of the connection (RECV), or from the sending end (SEND). Since the receiving end is best able to estimate when packets should arrive [C87a], and since under some fetch policies retransmission may not be requested, the obvious choice is RECV. This will be the location policy used by ALTP-OT.
Granularity. The granularity of retransmission determines how many missing packet events are accumulated before a request for retransmission is made. In a general purpose transport protocol, retransmission is typically based on selective or cumulative acknowledgement. Due to the knowledge of the super-packet structure of segment (access groups) by ALTP-OT, a richer set of options can be explored, that are based on the granularity of the data structure transmitted.

Identify each packet as $\pi_i p_j s_k \in g \subset \sigma$, indicating the $i$th packet of the $j$th page of the $k$th segment in segment access group $g$ corresponding to superpacket $\sigma$. Assuming that the network latency is (conservatively) $\delta_n$, and the time to locate the segment and initiate packet transmission at the remote end is $\delta_r$, the first (control) packet $\pi_0 p_0 s_0$ should appear at the local end after $\tau_0 = \tau_{s_0 p_0 s_0} = 2\delta_n + \delta_r [\text{sec}]$ for a get-segment or related operation. The estimation of $\delta_n$ has been extensively researched, particularly in the Internet community [Kapa87, Zh86]. For a connection oriented substrate, the bound can probably be somewhat tighter than would be required using the current IP [RFC791], since the rate specification can assist in the estimate of round trip delay (including the object transmission time), and provides for less variability in latency.

Four obvious possibilities for retransmission granularity exist. Let the packet size be $|\pi|$ bits, and the transmission rate $r$ bits/sec. The packet interarrival time is thus $|\pi|/r$. The $i$th packet is then expected to have arrived $\tau_{pi} = \tau_0 + |\pi|/r$ seconds after the get-segment request.

Using this, and letting the size of segment $s_k = n_{sk}$ pages, the page size $n_p$ packets, and the number of segments in the access group $n_a$, the timeout bound to request retransmission of a packet $\pi_i p_j s_k$ can be easily expressed. The granularities, along with corresponding time-out bounds are:

**PKT** — Packet granularity: request retransmission whenever a packet received is corrupted, or after a timeout value:

$$\tau_{pi p_j s_k} = \tau_0 + \frac{|\pi|}{r} \left[ i + (j - 1)n_p + \sum_{n=1}^{k-1} kn_p n_{sk} \right]$$

**PGE** — Page granularity: request retransmission of all packets corrupted or missing from a page, after all packets in the page have been received, or the last packet in the page has timed out:

$$\tau_{p j s_k} = \tau_0 + \frac{|\pi|}{r} \left[ j n_p + \sum_{n=1}^{k-1} kn_p n_{sk} \right]$$

**SEG** — Segment granularity: request retransmission of all packets corrupted or missing from a segment, after all pages have been fully received, or the last packet in the segment has timed out:

$$\tau_{s_k} = \tau_0 + \frac{|\pi|}{r} \sum_{n=1}^{k} kn_p n_{sk}$$

**GRP** — Segment access group granularity: request retransmission of all packets corrupted or missing from the entire superpacket, corresponding to an access group of segments, after all segments have been fully received, or the last packet in the segment group has timed out. The form of the timeout value is the same as under SEG granularity, where the segment number is the highest in the access group, $n_s$:

$$\tau_{s_{ng}} = \tau_0 + \frac{|\pi|}{r} \sum_{n=1}^{n_s} kn_p n_{sk}$$
Fetch policy. The retransmission strategies can also be classified by whether packets are always requested for retransmission, or only if a page is referenced that contains them. These are referred to as fetch policies due to the analogy with operating system page fetch policies. In both cases timers will be necessary:

AR – If all packets corrupted or missing are retransmitted, this corresponds to anticipatory retransmission, thus anticipating the future reference of all missing packets. In this case the timers discussed above indicate when a packet retransmission request should be made.

DR – If the only packets retransmitted are those corrupted or missing which are part of a page actually referenced, the policy is demand retransmission. This assumes that a number of packets in the segment will not necessarily ever be referenced, i.e. the program address reference trace \( \rho \) will hit only a subset of the pages in the segment. In this case, the timers indicate how long to wait before a referenced packet is assumed to be missing, and thus retransmitted.

Preemption. Since error control is in-band, packets retransmitted use the same connection and allocated bandwidth \( r \) as the primary data stream. The relative priority of original data and retransmitted packets can vary. The extreme cases are to allow all of the original request to flow before any of the retransmission requests to be serviced, or to immediately retransmit while blocking the primary data stream:

NP – Packet retransmission requests at the remote end are queued after the primary data stream, resulting in a non-preemptive strategy.

PE – Packet retransmission requests at the remote end preempt the primary data stream, and are transmitted as soon as possible. This minimises the the delay for reception of retransmitted packets.

Packet retransmission strategies. The total number of strategies is the cross-product of these orthogonal sub-policies: location, granularity, fetch, and preemption. Since ALTP-OT is designed assuming RECV location, the remaining three sub-policies determine the overall strategy, e.g. PGE-DRPE indicates retransmit a page's error packets (PGE) only when the page is referenced (DR), and preempt the primary data stream (PE).

The determination of the correct policy is a tradeoff between latency (delay in waiting for a referenced packet that must be retransmitted) vs. efficiency (utilisation of the connection by minimising the number of retransmission requests). The policy that minimises latency is PCT-ARPE; the policy that maximises connection efficiency is GRP-XRNP (the program address reference trace \( \rho \) will dictate whether AR or DR is better for a particular case).

In addition, some combination policies are useful e.g. PGE-DRPE/SEG-ARNP. This policy uses a page granularity, requesting preemptive retransmission of any page referenced (i.e. page fault). Otherwise, the primary data stream is allowed to complete before all other error packets are retransmitted. This provides a compromise between latency and efficiency by accumulating requests for the entire segment except in response to a page fault.

Based on the application characteristics and host and VHSI load, ALTP-OT can default to the appropriate strategy using parameters from a particular request. Intelligent IPC applications may wish to determine the policy used explicitly.
5. Related Work

Currently, most transport protocols in wide use [MePe82] (such as TCP [Co88, St88, RFC793], and the transport levels of X.25 [St87], SNA [Cy78, MaCh87, IBM85a, IBM85b], BNA [Un87a, Bu81], DCA [Un88a, Sp81], etc.) are designed to be general purpose, and are therefore not optimised to different classes of applications requiring high performance. These protocols are sufficiently complex that their implementation must reside largely in the software of the host system and front end communications processor. They are also not specified to allow the separation of critical functions to be optimally implemented in hardware. The classical layering mechanisms (exemplified in the extreme by the ISO-OSI model), result in significant penalties if layer boundaries are strictly adhered to. The layered model also frequently results in a number of functions (such as error and flow control) replicated at multiple layers, when much functionality could be moved to the ends of the connection [LeLo83, SaRe84, PoWa81].

TCP has been the most popular and successful transport protocol in the current DOD Internet, and there have been constant efforts to make it better [Ja88, KaPa87, Zh86]. Although these efforts have significantly improved TCP performance, the fundamental changes dictated by the VHSI environment indicate that assumptions about the communications substrate made by TCP are becoming increasingly outdated. This justifies the consideration of completely new transport protocols.

Several other protocols have been proposed for use in higher performance versions of the DOD Internet. These include VMP [Ch89a, Ch89a] and NETBLT [Cl87a, Cl87b]. VMP is designed as a general purpose transport protocol, with emphasis on RPC (remote procedure call) and page level file access. Significant aspects of VMP design applicable to ALTP-OT include the packet grouping and selective retransmission based on bit vectors. Additionally, a network interface has been specifically implemented for VMP, the VMP NAB (network adapter board) [Sa89, KaCh88]. Although the NAB is a high performance network interface, full error control (including resequencing) and packet buffering takes place.

NETBLT is a protocol designed for transport of large blocks of data with high throughput. The most significant aspect of NETBLT design applicable to ALTP-OT is the decoupling of error and flow control. In addition it is based on a simple rate-based flow control mechanism, with selective retransmission determined by timers at the receiving end of a transmission. Both of these protocols group packets to increase efficiency of transport.

Another approach to the performance problem is to implement existing transport protocol mechanisms in hardware. This is manifest in the work on the express transport protocol (XTP) and the protocol engine (PE) [Ch86b, ChEl88, Ch88b, ChGr88]. The XTP approach is to take existing general purpose protocols mechanisms, streamline the packet format for pipeline processing, and implement each step in the pipeline using a customised VLSI processor.

In summary, NETBLT, VMP, and XTP have contributed a number of interesting ideas to the design of transport protocols, and they do improve upon TCP within the current DOD Internet for the applications they were originally designed for. However, we believe that these protocols are not appropriate solutions for the VHSI environment, because the underlying assumptions and trade-offs that these protocols are based on are very different. Specifically, these assumptions include the quasi-reliability provided by an underlying connection-oriented internet substrate (MCHIP and VHSI), and data rates that are several orders of magnitude greater than these proposed protocols assume. More detail concerning the incompatibility of these protocols extended to the VHSI environment, and the justification of ALTPs has been discussed in [BlSt88].
6. Conclusions

There has been significant progress in the areas of communication and computer architecture over the past few years. We will soon have communication networks and internetworks that can support data rates of more than hundreds of Mbps, and have computers that can process numerous demanding applications such as video distribution, computer imaging, distributed scientific computation and visualization, distributed file and procedure access, and multimedia conferencing. These applications in a network environment can be characterised as needing to transmit large bursts of data with sub-second latency. The major bottleneck in supporting these applications remains at the host–network interface. In particular, IPC performance across a network has not kept pace with the demands of applications and the data rates of underlying networks.

We have proposed a new host communication architecture for the distributed systems called Axon, which can support IPC with high throughput and low latency across the VHSI. In this paper we have presented the overall design of Axon, and have described the design of ALTP-OT in detail. This includes the motivations for application oriented lightweight protocols, and the rate and error control mechanisms used by ALTP-OT. The ALTP-OT operations have been described, and will be refined as the Axon project continues, and a prototype is implemented.
A. ALTP-OT Operations

This section gives a high level description of the ALTP-OT protocol in terms of its operations. Some of these are initiated by system calls or operating system requests, such as for object transfer. Others are initiated by ALTP-OT internally, such as for packet retransmission in response to errors. A detailed description of get-segment is presented in §A.1.

Each of these operations will be specified in terms of the invocation parameters (operations in brackets are only initiated by other ALTP-OT operations), the object or request that is sent to the remote host, the response (if any) that is expected from the remote host, any parameters returned locally, possible error conditions, a flow diagram, and finally a summary description. If a particular operation involves no request or response from the remote host, these categories will be omitted. In the flow diagrams, thick arrows indicate data transfer (as opposed to only control packets), parenthesis ( ) indicate conditional operations, and brackets [ ] relate to error or abnormal conditions. Other ALTP-OT operations that are typically related to a particular flow diagram are also shown. For example, in the get-segment flow diagram, (acquire-segment) may conditionally be performed beforehand, [retransmit-packets] in response to packet errors during, and (release-segment) conditionally afterward.

Join-ipc

invocation: join-ipc (h, c, r, o)

h host address/path
c connection id. (for joining existing connection)
r rate specification vector (for initiating connection)
o options
   allow congarm reroute
   routing policy constraints

returns:
connection/congarm id. from MCHIP if establishment

error conditions:
1 invalid connection id.
2 insufficient resources for rate spec
3 host unreachable

description:
Join-ipc is used to establish or join a multipoint connection to be utilised for n-way IPC. If a connection does not yet exist, an open-con.mtp MCHIP call will be issued, which causes MCHIP to establish a congarm, and the MCHIP congarm id. is returned as the connection id. If a connection already exists, but does not include the local host, an add-endpoint.mtp MCHIP call will be issued. When joining an existing connection, the connection id. may be specified.

Join-ipc may be explicitly issued by the application as soon as it is known that remote host access will take place, to avoid the setup delay of connection establishment at the time the remote segment is actually referenced. It also may take place as a result of an ALTP-OT object transfer operation for which a connection does not yet exist.
Respecify-rate

**invocation:** respecify-rate (c, r, o)
- **c** connection id.
- **r** new rate specification vector
- **o** options
  - allow congram reroute
  - routing policy constraints
  - host rate only

**error conditions:**
1. invalid connection id.
2. insufficient resources along existing route
3. host unreachable on new route at rate spec
4. host rate greater than congram/connection rate

**description:**
Respecify-rate is used to alter the rate specification for an existing connection. This may be done by an application that is aware of changing requirements for bandwidth or resource availability, or by the host operating system to reflect host system load and resource management. In general, this is done infrequently based on long term requirements. The host rate option will readjust the rate specification *only* at the end-point hosts, and may be used to meet shorter term application requirements without the overhead of resource management and delay of congram reconfiguration by MCHIP and the VHSI.
Leave-ipc

**invocation:** leave-ipc (c)

- c    connection id.

**error conditions:**

- 1    invalid connection id.

**description:**

Leave-ipc is used by a process to disconnect from a multipoint connection established by a join-ipc. This results in a `remove-endpoint.mtp` or `close-con.mtp` call to MCHIP (depending on whether or not this is the last process in the IPC set).
Get-segment

invocation: get-segment \((s), h, c, s\)
\(s\) symbolic segment name
\(h\) host address or path
\(c\) connection id. (optional)
\(s\) segment characteristics
  - search specification \((user, connid, well-known)\)
  - estimated size \(|s|\) in pages
  - local page (or scanline) size \(|p|\) in bits
  - granularity \((seg, grp)\)
  - access intent \((rwxs, override)\)
  - authorisation \((encrypted userid, password)\)
  - lock info \((priority, defer)\)
  - PDT address

sends:
  control packet containing the segment name \((s)\) and access and location information

response:
  super-packet of data packets containing segment \(\sigma = s\) or group \(\sigma = \emptyset \cup s\)

error conditions:
1. \(h\) not found or unreachable
2. segment \((s)\) not found
3. segment \((s)\) access not authorised
4. wait for lock on \((s)\)

flow diagram:

```
(acquire-segment)

send segment request packet ———> receive request

  [not found]  ———>
  [unauthorised access] ———>
  [wait for lock]

receive data packets ———> return data packets
```

[packet error/timeout]
[retransmit-packets]

[request timeout]
[get-segment]

(release-segment)
description:
Get-segment is used to retrieve a segment (or segment access group) from a remote host. It is typically called by NVS when a segment fault results in requiring a segment not on the local system. It may also be called directly by an application to pre-fetch the segment into the process virtual address space, to avoid the end-to-end latency when the segment is actually referenced. Finally, if a get-segment is issued for a segment already acquired, the authorisation and location of the segment (essentially a link fault on the remote host) are not performed again, avoiding the overhead of link fault for a remote segment fault.

A get-segment control packet is sent to the remote host, using a connection. If a connection id. c is specified, it is used for the transport. If it is not specified, a join-ipc operation will be done first. Note that MCHIP supports the transport across a permanent internet congram (PICON), to avoid the delay while the user congram (UCON) is being established [Pa90]. The packet contains the name of the segment needed (s), along with various options indicating how the segment will be found, authorisation information, and access intent and lock information. The local page size (p) is used to indicate how the remote CMP should assign page numbers (or scanlines) to the segment as it is being transmitted. The estimated segment size (s) is used to allocate local space for the segment according to the remote placement policy [StPa89b].

Typically, a data super-packet is returned, which contains the requested segment (or segment access group if the granularity is specified as grp). Note that if the segment type is video/image, then the access group corresponds to the complete image, the segment is a frame, and a page is a scanline. The actual segment size (s) is used to adjust the allocation of space based on the initial estimate (s). If the segment was not found on the remote host, an error control packet is returned, indicating segment-not-found. If the authorisation of the request fails, an authorisation-failed control packet is returned. If the segment is locked at a higher priority than the request, a wait-for-lock control packet is returned.
Acquire-segment

Invocation: acquire-segment (\(s\), \(h\), \(c\), \(s\))

\(s\) symbolic segment name
\(h\) host address or path
\(c\) connection id. (optional)
\(s\) segment characteristics
  search specification (user, connid, well-known)
  estimated size \(|s|\) in pages
  local page size \([p]\) in bits
  granularity (seg, grp)
  access intent (rwxs, override)
  authorisation (encrypted userid, password)
  lock info (priority, defer)
  PDT address

Sends:
control packet containing the segment name (\(s\)) and access and location information

Response:
acknowledgement control-packet indicating segment or segment access group acquirement

Error conditions:
1. \(h\) not found or unreachable
2. segment (\(s\)) not found
3. segment (\(s\)) access not authorised
4. wait for lock on (\(s\))

Flow diagram:

send acquire request packet  \(\rightarrow\) receive request

\(\leftarrow\) [not found]
\(\leftarrow\) [unauthorised access]
\(\leftarrow\) [wait for lock]

receive authorisation  \(\leftarrow\) segment acquired

[request timeout]
[acquire-segment]

\((\text{get-page})\)
:
\((\text{get-page})\)

(description:
Acquire-segment is used to gain access to a segment (or segment access group) in a manner similar to get-segment, but the segment itself is not returned. This is typically used when the ASR (auxiliary storage remote) remote placement policy is used, in which case pages will be fetched across the network using get-page operations.)
Release-segment

invocation: release-segment ((s), o)
  (s)  symbolic segment name
  o   options
    flush segment

sends:
  control packet containing the segment name (s), optionally followed by superpacket containing packets of modified pages being returned.

response:
  acknowledgement control-packet indicating segment released

error conditions:
  1  segment (s) not in process address space

flow diagram:

(return modified pages)  (replace pages)
send release control packet  release segment
receive acknowledgment  acknowledge release

[acknowledgment timeout]
[release-segment]

description:
Release-segment is used to release or return a segment to the originating host. If the segment is not writable, has not been modified, or the flush option has been specified, a control packet will be returned to the remote host releasing the segment from use. Otherwise, a control packet will be sent indicating that there is data to be returned, followed by a super-packet consisting of all pages in the segment that have been modified.

This can either be done by the application when it is known that segment access is no longer necessary, by NVS management in response to storage shortages (real or auxiliary depending on the remote placement policy), or by the operating system as part of process termination.
Get-page

_invocation:_ get-page (\(s\), \(p\))

\(s\) symbolic segment name
\(p\) page number

_sends:_
control packet containing the segment name \(s\) and page number \(p\)

_response:_
super-packet of data packets containing the requested page

_error conditions:_
1 segment \(s\) not acquired
2 page \(p\) out of segment bounds
3 page \(p\) access not authorised

_flow diagram:_

\[
\begin{align*}
\text{(acquire-segment)} \\
\text{send page request packet} \quad \rightarrow & \quad \text{receive page request} \\
\quad \leftarrow & \quad \text{[segment not acquired]} \\
\text{receive page packets} \quad \leftarrow & \quad \text{return page packets} \\
\text{[packet error/timeout]} \quad \leftarrow & \quad \text{[retransmit-packets]} \\
\text{[request timeout]} \quad \leftarrow & \quad \text{[get-page]} \\
\text{(release-page)} & \quad \leftarrow \\
\text{(release-segment)} & \quad \leftarrow
\end{align*}
\]

_description:_
Get-page is similar to get-segment, except that transfer is at the page level (ASR remote placement policy). Since the segment is paged from the remote host, it must have been acquired (appropriate authorisation and locks) using the acquire-segment operation.

A get-page may also be issued after a get-segment for the segment containing the corresponding page. In this case its transfer preempts the remainder of the segment transfer. This may be done when a page is referenced early that is near the end of a very large segment to reduce process blocking. When a second copy of the page arrives as part of the normal segment transfer, its packets will be treated as duplicates and discarded.

A get-page control packet is sent to the remote host, across a connection. The packet contains the page number and segment name of the page needed. A super-packet containing the requested page is returned.
Release-page

**invocation:** release-page ((s), p, o)

(s) symbolic segment name

p page number

○ options

flush page

**sends:**

control packet containing the segment name (s), along with any data packets being returned

**response:**

acknowledgement control-packet indicating page released

**error conditions:**

1 segment (s) not in address space
2 page p out of segment bounds
3 page p not in local working set

**flow diagram:**

```
(return modified page) ────────→ (replace page)

(receive acknowledgment) ←─────── (acknowledge release)

([[acknowledgment timeout]])

([[release-page]])
```

description:

Release-page is used to release or return a page to the originating host, when page level movement is taking place. If the segment is not writable, the page has not been modified, or the flush option has been specified, the page space can be reclaimed. Otherwise, a super-packet will be sent, consisting of the page to be returned to the remote host. This is done by NVS operating system resource management for working set adjustments.
Get-copy

**invocation:** get-copy ($s, h, c, s, s, u$)
- $s$: symbolic segment name
- $h$: host address or path
- $c$: connection id. (optional)
- $s$: segment characteristics
  - search specification (user, connid, well-known)
  - local page size $p$ in bits
  - estimated size $s$ in pages
  - granularity (seg, grip)
  - access intent (rwx,s, override)
  - authorisation (encrypted userid, password)
- $s$: local segment location
- $u$: user id (optional)

**sends:**
control packet containing the segment name ($s$) and access information and location options

**response:**
super-packet containing the requested segment

**error conditions:**
1. $h$ not found or unreachable
2. segment ($s$) not found
3. segment ($s$) access not authorised
4. wait for lock on ($s$)

**flow diagram:**

```
send segment request packet --send-- receive request
                             \- [not found]
                             \- [unauthorised access]
                             \- [wait for lock]
receive data packets --receive-- return data packets
```

- [packet error/timeout]
- [retransmit-packets]
- [request timeout]
- [get-copy]

**description:**
Get-copy is used to get a segment (or segment access group) similar to get-segment, but the copy becomes permanent (until explicitly destroyed) on the local host. The segment becomes part of the context of the specified user (defaulting to the caller), with an entry added to the corresponding udir. This may be used for a conventional file-transfer type service.
Receive-message

**invocation:** receive-message \((m, h)\)
- \(m\) symbolic message buffer name or
- message buffer address
- host address or path

**response:**
assumes that message super-packet has been received from \(h\)

**error conditions:**
1. message buffer \(m\) invalid
2. host mismatch

**flow diagram:**
```
receive message ← transmit message

(packet error/timeout) [retransmit-packets]

(acknowledge) → (receive acknowledge)

([[negative acknowledge]]) → ([[receive negative ack]])

([[receive message]]) → ([[send-message]])
```

**description:**
Receive-message is used to receive a message from a remote host, when message-passing is being used for the IPC paradigm. The program issuing the receive is assumed to be synchronised with the sender such that the message exists upon receipt.
Get-stream

invocation: get-stream \((g), h, s, o\)

\(g\) symbolic segment stream name
\(h\) host address or path
\(s\) segment characteristics
  search specification \((user, connid, well-known)\)
  local page (or scanline) size \(|p|\) in bits
  estimated size \(|s|\) in pages
  granularity \((seg, grp)\)
  access intent \((rwxs, override)\)
  authorisation \((encrypted user id, password)\)
  lock info \((priority, defer)\)
  PDT address
\(o\) options
  inhibit/require error packet retransmissions
  repeated \((r)\) or sequential \((s)\) transfer
  program synchronised \((p)\) or time synchronised \((r=n)\)

sends:
  control packet containing the segment name \(g\), streaming options, and access information and location options

response:
  segment \(s = g\) repeatedly, or each segment in access group \(s_i \subseteq g\) (for sequential transfer)

error conditions:
  1 \(h\) not found
  2 segment (group) \(s\) not found
  3 segment (group) \(s\) access not authorised
  4 wait for lock on \(g\) or \(s\)

flow diagram:

send stream request packet \(\rightarrow\) receive request

\(\leftarrow\) [not found]
\(\leftarrow\) [unauthorised access]
\(\leftarrow\) [wait for lock]

receive data packets \(\leftarrow\) return segment packets

receive data packets \(\leftarrow\) return segment packets

receive data packets \(\leftarrow\) return segment packets

\[\text{[packet error/timeout]}\]
\([\text{[retransmit-packets]}]\)

\[\text{[request timeout]}\]
\([\text{[get-stream]}]\)
description:

Get-stream is used to retrieve a segment stream from a remote host.

A get-stream control packet is sent to the remote host, using a connection on top of a MCHIP congram (UCon), which must have been established using a join-ipc call. The packet contains the name of the segment needed, along with various options indicating how the segment will be found, authorisation information (if required), and access intent and lock information.

A series of data super-packets are returned, which contain the requested segment(s). Error conditions are similar to the get-segment operation.
Remote-execute

**Invocation:** remote-execute ((s), h, e)
- s: symbolic (local) segment name
- h: host address or path
- e: execution characteristics

**Sends:**
- control packet indicating segment(s) size and execution characteristics, followed by data super-packet containing code segment to be executed (if local), and (optionally) data parameter segment(s), and execution characteristics

**Returns:**
- request acknowledgement control packet, and after execution super-packet containing termination status, and (optionally) return data parameter segment(s)

**Error Conditions:**
1. h not found
2. (local) segment (s) not found
3. (local) segment (s) access not authorised
4. wait for lock
5. unable to build remote execution context subject to e
6. remote data segment not found
7. remote data segment access not authorised

**Flow Diagram:**

- Send execute request packet ➔ receive request
  - (send code segment) ➔ (receive data packets)
  - [unable to build context]
  - [unauthorised access]
  - (send data segment(s)) ➔ (receive data packets)
  - receive acknowledgement ➔ acknowledge execute request
  - [request timeout]
  - [remote-execute]
  - (receive data packets) ➔ (return data segment(s))
  - [packet error/timeout]
  - [retransmit-packets]
  - [abnormal termination]
description:

Remote-execute is used to initiate execution of a code segment on a remote host. Typically, the code segment resides on the local host, but this is not necessarily the case. Some data segments containing parameters may also reside locally. If the code segment, or any required data segments reside on other remote hosts, these will be fetched by the remote host as part of the NVS segment fault mechanism.

A remote-execute control packet is sent to the remote host across a connection, indicating the total size of any segments that will be sent plus the estimated sizes of any other remote segments, and the characteristics of the execution. This is followed by a super-packet containing the code segment and any parameter data segments (if local).

Typically, a super-packet is returned, which contains a remote-execution-terminated control packet, indicating either successful termination, or abnormal termination with process state information. If the procedure returns parameters, packets from a data segment follow in the super-packet.
Send-copy

**invocation:** send-copy ((s), h, c, u)

- **(s)** symbolic segment name
- **h** host address or path
- **c** connection id. (optional)
- **u** user id

**sends:**
- control packet containing the segment (s) and access information and location options for remote placement

**response:**
- acknowledgement of successful copy

**error conditions:**
1. **h** not found
2. segment (s) not found
3. segment (s) access not authorised
4. wait for lock

**flow diagram:**

- send copy request packet ——> receive request
- send segment data packets ——> receive data packets
- [unauthorised access]
- [unable to allocate]
- [packet error/timeout]
- [retransmit-packets]
- receive acknowledgement ——> acknowledge copy
- [request timeout]
- [send-copy]

**description:**
Send-copy is used to permanently move a copy of a segment to a remote host. The segment becomes part of the context of the target user u, with a corresponding entry added to the udir. If the segment (s) resides on a remote host, the effect will be that of a remote-to-remote segment copy. This could be used for a conventional file-transfer type service.
Send-stream

invocation: send-stream ((g), h, o)
  (g) symbolic segment stream name
  h host address or path
  o options
    inhibit/require error packet retransmissions
    repeated (r) or sequential (s) transfer
    program synchronised (p) or time synchronised (t=n)

sends:
  segment s ≡ g repeatedly, or each segment in access group s_i ⊆ g

error conditions:
  1 h not found
  2 (local) segment (s) not found
  3 (local) segment (s) access not authorised
  4 wait for lock

flow diagram:

send stream request packet ———————> receive request

send segment packets ———————> receive data packets
send segment packets ———————> receive data packets

receive acknowledgement ————> acknowledge stream

[unable to allocate]
[unauthorised access]

[request timeout]
[send-stream]

send segment packets ———————> receive data packets
send segment packets ———————> receive data packets

[packet error/timeout]
[retransmit-packets]

description:
Send-stream is used to send a segment stream from a remote host.
A send-stream control packet is sent to the remote host, using a connection on top of
a MCHIP congram (UCon), which must have been established using a join-ipc call. The
packet contains the name of the segment and destination user and process information.
A series of data super-packets will then follow, which contain the segment(s) as they are
streamed.
Send-message

invocation: send-message ((m)|m, h, p, o)
  (s) symbolic message buffer name
  m message buffer address
  h host address or path
  p process id.
  o options
    reliable send (acknowledge)
    blocked send (wait for acknowledge)

sends:
  message super-packet

response:
  acknowledgement if appropriate

error conditions:
  1 h not found
  2 p invalid
  3 message m not found

flow diagram:

```
send message  -----→  receive message
               [packet error/timeout]
               [retransmit-packets]

(receive acknowledge)  ←  (acknowledge)

([receive negative ack]) ←  ([negative acknowledge])
```

description:
Send-message is used to send a message from a remote host, when message-passing is being used for the IPC paradigm. A send-message control packet is built which contains the destination host address, the connection id, and the message object.
Retransmit-packets

**invocation:** \([\text{retransmit-packets}] (\mu, r, c)\)
- \(\mu\): packet bit array
- \(r\): request id.
- \(c\): connection id.

**sends:**
- super-packet containing the bit vector \(\mu\) of missing packets to be retransmitted.

**response:**
- super-packet containing the requested packet set.

**flow diagram:**

```
packet error/timeout
retransmission request    receive request
receive packets           retransmit packets
[packet error/timeout]
[retransmit-packets]
```

**description:**
Retransmit-packets is used to request the retransmission of missing or corrupted packets from a previous ALTP-OT request.
A retransmit-packets control packet is sent to the remote host, across the connection used for the original get-segment request. This contains the packet bitmap array \(\mu\), which has a bit set for each packet missing. This provides for selective retransmission of packets, with the retransmission strategy dictating when the request is made, and thus the number of packets for each request.
A data super-packet is returned, which contains the packets corresponding to the \(\mu\) array sent.
Invalidate-segment

**invocation:** \([\text{invalidate-segment}] ((s), h, o)\)

- \(s\) symbolic segment name
- \(h\) host address or path
- \(o\) options
  - return modified copy

**sends:**
- invalidation control packet

**response:**
- status

**error conditions:**

**description:**

Invalidate-segment is used to invalidate copies of segments when a lock is preempted by a high-priority get/acquire-segment, or by the owner of the segment. This is called from the segment coherency protocol portion of ALTTP-OT.

An invalidate-segment control packet is broadcast across the multipoint connection which reaches all hosts that have a copy of a particular segment (access group).
A.1. Example protocol operational details (get-segment)

This section provides details of the get-segment operation. The operation at both the local and remote ends will be described.

![Diagram of ALTP-OT Get-segment Operation (Local Host)](image)

Figure 7: ALTP-OT Get-segment Operation (Local Host)

**Local end.** Figure 7 shows the state diagram for the local operation. At the local end, ALTP-OT is responsible for checking if a MCHIP congram exists, and if not, making an open-con.mtp request to MCHIP. While MCHIP is establishing the connection, ALTP-OT can perform the initial control set-up in parallel. Space is allocated based on the estimated segment size, $|S|$, and according to the remote placement policy [StPa89b]. The get-segment control packet is built by inserting the appropriate SDT (segment descriptor table) and AST (active segment table) fields into the template in CMM (communications memory module), and is sent to the remote host on the MCHIP congram.

If MCHIP cannot provide the requested connection, either due to resource constraints, or unreachability of the remote host, control is returned along with the reason for denial. If the connection was denied due to the unavailability of resources according to the rate specification, it may be desirable for MCHIP to indicate which of the parameters could not be met, and by how much.
Each packet returning on the connection is checked for a connection-id. and request-id. matching the outstanding get-segment request. If a return control packet does not arrive before a timeout $\tau_0$ occurs, the get-segment request is reissued.

If a segment-not-found control packet arrives, an error exit is taken. If a wait-for-lock control packet arrives, ALTP-OT continues to wait for data packets.

When a matching return-data control packet is received, the actual segment length $|s_i|$ is checked against the original estimate $\hat{|s_i|}$. If additional space needs to be allocated, this is done, and then the estimate is updated to reflect the actual length.

When each data packet $\pi_ip_js_k$ arrives, the segment, page, and packet numbers in the header are checked ($i^{th}$ packet of the $j^{th}$ page of the $k^{th}$ segment). The CMP packet presence logic maintains the state $\mu$ of received packets. The packet is checked to be within bounds of the allocation of space for the segment $|s_k|$, and checked against the packet presence state $\mu_{ijk}$ to see if it is a duplicate. If either of these conditions is met, it is discarded. Otherwise, it is placed in real or auxiliary storage, dependent on the remote placement policy. The appropriate presence state $\mu_{ijk}$ is updated to indicate the packet arrival. When all of the packets in a particular page are present, the CAP sets the corresponding PDTE (page descriptor table entry) presence bit, and signals the host to recover from a page fault.

When all of the packets in the segment (or access group) have arrived, the CAP sets the corresponding SDTE (segment descriptor table entry) presence bit(s). The ALTP-OT connection then idles, until the process ends or issues an explicit leave-ipc. The ALTP-OT then issues a close-con.mtp to MCHIP, and terminates for this process.

In the state diagram, the receiving-packet states are divided to show both ALTP-OT and process state. Due to the interaction between the operating system and ALTP-OT, the process state is affected by the protocol operation. In particular, if a page is referenced for which pages have not yet been received, the process blocks by incurring a page fault. When the entire page has been received and placed in storage, the presence bit in the appropriate PDTE (page descriptor table entry) is set, and the process is dispatched. The process state indication and operating system actions are depicted in the state diagram within brackets.

The actions in the receiving $\pi_ip_js_k$ state are:

```plaintext
if $j > |s_k|/|p|$ [check packet in bounds of allocated page]
    then retransmit-packet ($\pi_ip_js_k$) [discard and request retransmit (strategy based)]
else do
    if $\mu_{ijk} = 1$ [if duplicate]
        then skip [discard packet]
    else do
        $\mu_{ijk} \leftarrow 1$ [marked received]
        store ($\pi_ip_js_k$) [store packet]
    od
    fi
    if $\prod_j \mu_{ijk} = 1$ [if all packets in a page present]
        then do
            ($\text{SDTE}(s_k), \uparrow \text{PDT}(p_j), \text{pres} \leftarrow 1$ [set page presence bit]
            $\text{sysevent-dispatch} (\uparrow \text{APT})$ [mark process dispatchable]
        od
    fi
fi
```
Remote end. Figure 8 shows the state diagram for the remote operation. At the remote end, ALTP-OT is responsible for receiving and decoding the get-segment request, based on the search specification (user, connid, or well-known). The operating system is notified to process a link fault on the named segment. Authorisation is checked, both for access rights, as well as against an ACL (access control list) if the segment is not public. If the segment is not found or authorisation fails, the request is terminated and a control packet returned indicating the reason for termination. If the segment is active, locks are resolved. If the lock priority of the requesting host is lower, a control packet indicating wait-for-lock is returned. If the requesting host has a higher priority, the lock is released, and all other copies in a multipoint connection invalidated with an invalidate-segment multicast. The segment (access group) is then returned to the requesting host, subject to the rate specification $\tau$.

The actions for significant states are summarised below:

Sending $\pi_i p_j s_k$ state:

\[
\forall s_k \in \text{gt} \quad \forall p_j \in s_k \quad \forall \pi_i \in p_j
\]

[all segments in access group]
[all pages in segment]
[all packets in page]

$\sigma.\text{seg} \leftarrow k$ [set segment number in return data packet header]
$\sigma.\text{pge} \leftarrow j$ [set page number in return data packet header]
$\sigma.\text{pkt} \leftarrow i$ [set packet number in return data packet header]
$\sigma.\text{data} \leftarrow \pi_i p_j s_k$ [load data to return packet data buffer]
transmit($\sigma\pi$) [return data packet]
Locating $s_k$ state:

$\langle s_k \rangle \leftarrow \sigma \pi \text{.name} \quad \text{[extract segment name from header]}

s_k \leftarrow \beta(s_k) \quad \text{[system call to bind segment name]}

if $s_k = \text{null}$ \quad \text{[if not found]}

then do

$\sigma \pi_0 \text{.found} \leftarrow 0 \quad \text{[clear existence bit in header of return control packet]}

transmit(\sigma \pi_0) \quad \text{[return control packet]}

od

auth \leftarrow \text{check-auth}(s_k) \quad \text{[system call to check access authorisation]}

if auth = 0 \quad \text{[if unauthorised access attempt]}

then do

$\sigma \pi_0 \text{.unauth} \leftarrow 1 \quad \text{[set unauthorised access bit of return control packet]}

transmit(\sigma \pi_0) \quad \text{[return control packet]}

od

held \leftarrow \text{check-lock}(s_k) \quad \text{[system call to check if segment locked]}

if held = 1 \quad \text{[if locked]}

then do

$\sigma \pi_0 \text{.lockwait} \leftarrow 1 \quad \text{[set wait bit of return control packet]}

transmit(\sigma \pi_0) \quad \text{[return control packet]}

od

do until lock-free skip od \quad \text{[hardware spin/suspend on lock]}
References


Sterbenz and Parulkar

Washington University Department of Computer Science, technical report wucs-88-7, St. Louis, 1988.


