

FACULTY OF INFORMATICS AND CESNET  
MASARYK UNIVERSITY, BOTANICKÁ 68A, 602 00, BRNO

# **Active Router Communication Layer**

CESNET TECHNICAL REPORT NUMBER 11/2004

**Tomáš Rebok**

e-mail: [jeronimo@ics.muni.cz](mailto:jeronimo@ics.muni.cz)

Brno, Czech Republic, 2004

## **Abstract**

Future computer networks must be more flexible and faster than today. Active network paradigm is the way how to add flexibility to the networks of today. Passive transport medium present in current networks is transformed by application of active network principles into distributed programmable computing system with a lot of new features but lots of problems, too. Several architecture models have been proposed to solve problem of active networks and its elements. A new architecture based on model of active nodes has been presented by Eva Hladká and Zdeněk Salvét. This architecture is designed to be sufficiently general to accommodate any underlying packet-based networking technology. This architecture also proposes a prototype implementation of an active router using PC-class computers with NetBSD operating system. The main task of this router, which is the same as for any other usual router, is to transfer data from sending node to receiving node. Such communication requires special protocols that can provide the transfers with required quality of service – such protocols are called transport protocols.

The main goal of this work is to design and implement a new transport protocol called Active Router Transport Protocol that can be used as a communication layer of the active router being developed at Masaryk University in Brno. This router has special requirements discussed in this report and this is the reason why current transport protocols are not suitable for the router. Although proposed protocol is primarily designed for active routers, it is general enough to be usable outside of active networks environment.

## Contents

<b>1</b>	<b>Active Router Transport Protocol (ARTP)</b>	<b>1</b>
1.1	<i>Terminology</i>	1
1.2	<i>Requirements on Proposed Protocol</i>	2
1.3	<i>ARTP Design Philosophy</i>	3
1.4	<i>Functional Specification</i>	4
1.4.1	Header Format	4
1.4.2	Timestamps	10
1.4.3	Session Establishment	12
1.4.4	Session Termination	13
1.4.5	Data Communication	13
1.4.6	Interfaces	15
<b>2</b>	<b>Implementation and Tests</b>	<b>17</b>
2.1	<i>Prototype Implementation</i>	17
2.2	<i>How to Use Prototype ARTP Library</i>	17
2.3	<i>Experimental Validation</i>	18
2.3.1	Throughput Measurement	18
2.3.2	Reliability Test	19
2.3.3	The Stability Test	21
<b>3</b>	<b>Conclusions</b>	<b>23</b>

## Chapter 1

# Active Router Transport Protocol (ARTP)

### 1.1 Terminology

Before we discuss the ARTP protocol, we need to introduce terminology that is used throughout this document:

**client** – the side that initializes a connection. This is the sender in the case of one-way communication.

**server** – side that accepts a connection. In the case of one-way communication it is the receiver. Each server can be a client for another server at the same time (it is not the end point of communication but the middle one).

**session** – connection made between client and server used for data transmission. Each session can be in one of the two states:

- **non-established:** used for transmitting requests for new sessions.
- **established:** used for the actual data transmission. Both endpoints have to have all necessary structures prepared and all information required for providing Quality of Service must be kept.

**datagram** – a data element that is passed between the ARTP and an application. Its size is not limited by the ARTP.

**packet** – a data element that is passed through the network. Its structure is specified by the ARTP and its size is limited depending on network architecture used. There are the following types of packets used in the ARTP:

- **data packets:** they transmit application data or parts of them (fragments).

## 1. ACTIVE ROUTER TRANSPORT PROTOCOL (ARTP)

---

- **control packets:** they are used for transmitting control information between both communication points. They control session establishment and management.
- **acknowledge packets:** they transmit sequence numbers of received packets to be acknowledged (see below).

**fragment** – the part of datagram that is transmitted in one packet. If required by maximum transmission unit of the underlying network, a large datagram may be split into fragments on the sender side and reassembled again into original datagram on the receiver side.

**timestamp** – the time stored in packet header representing packet sending time.

**buffer** – container used for storing data in both communication points. There are several types of buffers:

- **sending buffer:** used for storing data to be sent.
- **receiving buffer:** used for storing received data.
- **acknowledge buffer:** used for storing sequence numbers of incoming packets to be acknowledged.

### 1.2 Requirements on Proposed Protocol

The common task of all transport protocols is to transmit data between sender and receiver. The quality requirements on the transfer can differ vastly depending on the application.

Proposed protocol is designed to be used in the active router being developed at Masaryk University in Brno. Its properties are given by router requirements – they can be divided into the following groups:

#### 1. Basic data transfer

Proposed communication protocol must be able to transfer datagrams in both directions between its users. Transferred data can be separated into two types – control data (used for session management) and the actual application data. Application data may consist of two parts, too – encrypted data and data signature.

#### 2. Reliable communication

The protocol must ensure reliable data transport (the whole data sent by sender must be received by receiver). It must recover from data

## 1. ACTIVE ROUTER TRANSPORT PROTOCOL (ARTP)

---

loss, duplication, or delay imposed by the underlying communication medium, e. g. Internet networks. Received datagrams are not passed to the receiver application in the sent order (protocol does not need to ensure delivery in the right order).

### 3. Flow control

The sender has to control the amount of data sent to the network to avoid network and receiver congestion. In typical passive network medium that doesn't signalize congestion state in some explicit way, the congestion is usually signalized by packet loss. If the communication passes without problems the amount of data sent to the network may increase. When the receiver or network become congested, protocol must react as soon as possible by decreasing sending rate.

### 4. Multiplexing

The protocol should allow to create more than one session between two communication points (they are specified by their IP addresses and ports). Each session must be unambiguously defined.

## 1.3 ARTP Design Philosophy

The ARTP protocol is connection oriented transport protocol providing reliable communication channel without ensuring that the data will be received in the same order as they were sent. A client that wants to use ARTP protocol for data transport to the server must create a connection before starting this transport. The established connection has to be closed at the end of transmission.

Since session multiplexing is basic requirement for ARTP design, a session identification number is provided. Each session created between two points is unambiguously defined by its IP addresses and ports and its identification number. This allows more connections to be created between two applications. New session creation is initiated by client by sending a connection request to the server. Server considers this request and sends its reply to the client. When a positive reply is sent, both sides create necessary structures for session management and the session turns into established state.

Established session can be used for duplex data transport. There are two types of data that can be sent by the ARTP protocol – control data and main data. Control data are dedicated to both end-point application management. Each control information includes its identifier, type (request

or reply) and its value. Main data consists of two parts – encrypted data and its signature. Each main data datagram includes its identification number (data sequence number). This identifier is chosen by sender application itself. Datagram size is not limited by the ARTP.

Transmitted data are encapsulated into packets. Each packet consists of its header (it contains necessary information about the packet) and its data part (it contains transmitted data). Every packet may include ARTP control information, too. It can be used for communication between both communication points (e.g. controlling maximum packet size, session prioritization, etc.). ARTP control information is not passed to the applications. Every implementation of the ARTP may choose its protocol control information arbitrarily.

ARTP packet header must contain a timestamp (the information about its sending time) and expiration time describing how long the given packet is valid. This information is necessary for too delayed packets detection<sup>1</sup>.

The reliability of the transmission is based on individual acknowledgements of received packets and duplicity detection. Every packet contains its sequence number (stored in its header) that identifies the packet unambiguously. The receiver sends the list of received sequence numbers to their sender periodically (those received since last acknowledgement sending). If some packet is considered to be lost (the acknowledgement did not come in specified time) it is retransmitted.

The ARTP controls the amount of data sent to the network to avoid receiver or network congestion. For this purpose the congestion window mechanism has been chosen. The congestion window indicates the maximum amount of data that may be sent to the network without being acknowledged (shown in Fig. 1.1). The window size varies depending on the actual transport state – it increases when the transport proceeds without packet loss supposing that the capacity of the network is not saturated. The congestion is detected by packet loss – the sender decreases the window size and thus the amount of data sent to the network.

### 1.4 Functional Specification

#### 1.4.1 Header Format

Each ARTP packet starts with 20 B header followed by transmitted data.

---

1. It is essential to remember that the actual sequence number space is bounded, though the counter range is very large.

# 1. ACTIVE ROUTER TRANSPORT PROTOCOL (ARTP)

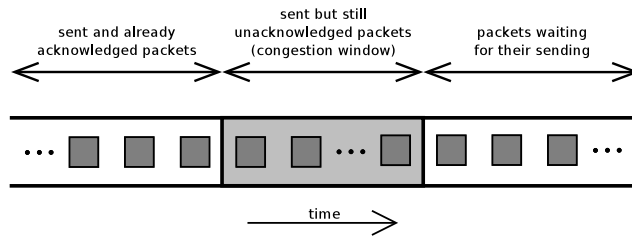


Figure 1.1: The congestion window mechanism.

Each information larger than one byte is stored in network byte order in an ARTP packet (in the context of microprocessors, this is known as big endian).

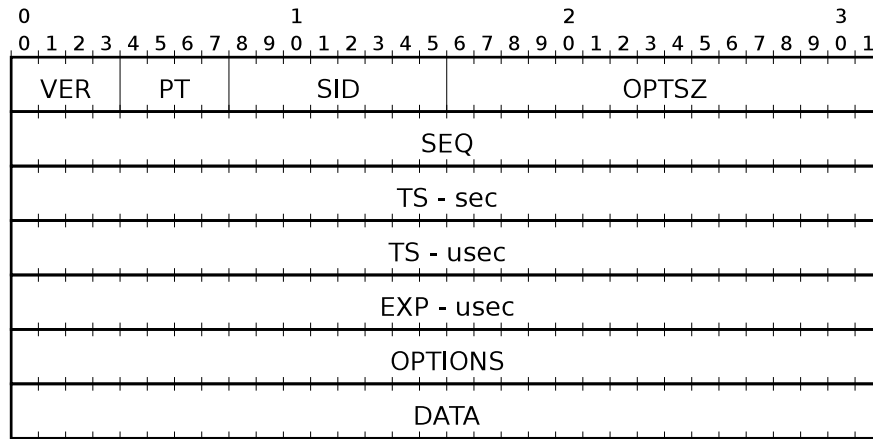


Figure 1.2: ARTP packet header.

VER (*Version*): 4 bits

Version of the ARTP protocol (currently 1).

PT (*Packet type*): 4 bits

Type of data stored in the ARTP packet. Available types are:

- data packet
- control packet
- acknowledge packet

## 1. ACTIVE ROUTER TRANSPORT PROTOCOL (ARTP)

---

SID (*Session identifier*): 8 bits

Session identification number.

OPTSZ (*Option field size*): 8 bits

The total size of OPTIONS array (in bytes).

SEQ (*Sequence number*): 32 bits

Packet sequence number.

TS – sec (*Timestamp – seconds*): 32 bits

Packet sent time – the amount of seconds since 1.1.1970.

TS – usec (*Timestamp – microseconds*): 32 bits

Packet sent time – the amount of microseconds.

EXP – usec (*Expiration time – microseconds*): 32 bits

Packet validity time from its sending (in microseconds).

OPTIONS: variable size

Protocol control information sent in each packet (see below).

DATA: variable size

Data sent in packet (see below).

OPTIONS array consists of 0 or more structures as shown in the following picture.

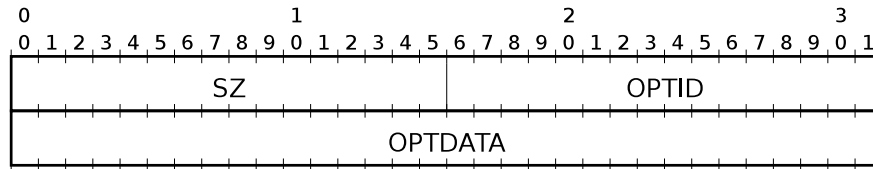


Figure 1.3: OPTIONS array structure.

SZ (*Option total size*): 16 bits

The total size of the given control information.

OPTID (*Option identifier*): 16 bits

The identification mark of the given control information.

OPTDATA (*Option data*): variable size

The data of the given control information.

## 1. ACTIVE ROUTER TRANSPORT PROTOCOL (ARTP)

---

The DATA array is used for storing transmitted data and its structure depends on packet type.

### **Application data**

The DATA array contains exactly one structure shown in Fig. 1.4.

# 1. ACTIVE ROUTER TRANSPORT PROTOCOL (ARTP)

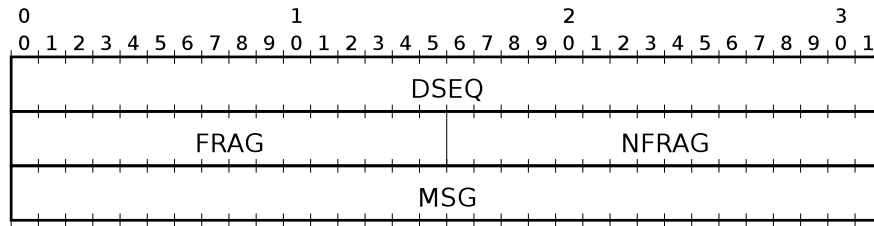


Figure 1.4: DATA array structure.

The MSG array contains exactly one structure shown on the picture 1.5. This information is fragmented when the datagram size is too big.

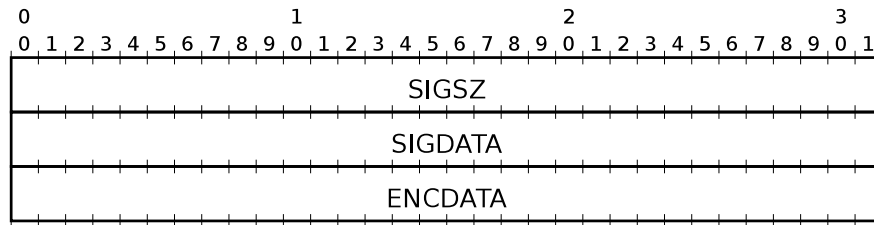


Figure 1.5: MSG array structure.

- DSEQ** (*Data sequence number*): 32 bits  
Sequence number indicating the right order of given datagram.
- FRAG** (*Fragment identifier*): 16 bits  
The number of given data fragment.
- NFRAG** (*Fragment count*): 16 bits  
The total count of data fragments of given datagram.
- SIGSZ** (*Signature size*): 32 bits  
The total size of SIGDATA array (in bytes).
- SIGDATA** (*Data signature*): variable size  
Data signature.
- ENCDATA** (*Encrypted data*): variable size  
Encrypted data.

## 1. ACTIVE ROUTER TRANSPORT PROTOCOL (ARTP)

---

### Session control information

The DATA array contains at least one structure shown on the picture 1.6.

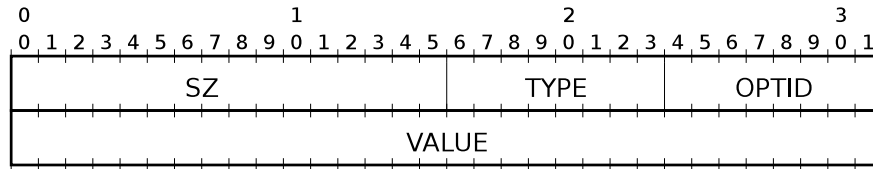


Figure 1.6: CTRL array structure.

**SZ** (*Control option size*): 16 bits

The total size of given session control information.

**TYPE** (*Control option type*): 8 bits

The type of given control information – request or reply.

**OPTID** (*Control option identifier*): 8 bits

The identification mark of given session control information.

**VALUE** (*Control option value*): variable size

The data of given session control information.

### Acknowledge packets

The DATA array contains at least one structure shown on the picture 1.7.

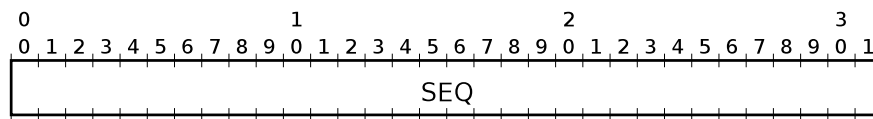


Figure 1.7: ACK array structure.

**SEQ** (*Sequence number*): 32 bits

The acknowledge packet number.

### 1.4.2 Timestamps

As mentioned above, there is the timestamp stored in each packet header. Timestamp is the information about packet sending time and together with expiration time it serves for too delayed packets detection.

In the general case there is no synchronization between clocks on sender and receiver. That is the reason why we can not base delayed packets detection simply by comparing packet sending time (expressed in sender time) with packet receiving time (expressed in receiver time) increased by packet expiration time.

#### ARTP timestamps

Due to possible time difference between sender and receiver time the ARTP sender must maintain the supposed difference between its time and the other side time (let us name it  $ts\_delta$ ). When the duplex communication occurs this time is maintained by both sides.

The ARTP mechanism of inserting timestamps into packet headers is as follows:

- if the time difference between both communication sides is not known ( $ts\_delta$ ) then the actual sender time is inserted into every packet. The expiration time is set to 0 simultaneously.
- if the time difference is known then the expiration time is inserted depending on application requests. The timestamp is inserted depending on packet type:
  - the supposed receiver time (sender actual time that is adjusted by  $ts\_delta$ ) is inserted into data and control packets.
  - the actual sender time is stored into acknowledge packets.

The mechanism of packet validity detection is:

- if the packet expiration time is set to 0 then this packet is always said to be valid.
- otherwise next steps depend on packet type:
  - when control or data packet is received the sum of time stored in packet header<sup>2</sup> and the packet expiration time is compared

---

2. This timestamp is stored in supposed receiver time so it does not have to be adjusted by  $ts\_delta$ .

## 1. ACTIVE ROUTER TRANSPORT PROTOCOL (ARTP)

---

with the actual receiver time. The packet validity is determined according to the comparison.

- acknowledge packet contains its sending time (it is expressed in sender time) stored in its header. This time has to be adjusted by actual  $t_{s\_delta}$ <sup>3</sup> – this time is now expressed in receiver time. The sum of this counted time with the packet expiration time is compared with the actual receiver time. The packet validity is determined according to this comparison.

The time difference between sender and receiver time ( $t_{s\_delta}$ ) is estimated using acknowledge packets that inform about non-retransmitted packets only. The calculation formula is:

$$t_{s\_delta} = CT - \left( ST + \frac{RTT}{2} \right)$$

where

- CT ... the receiving time of given acknowledge packet,
- ST ... the sending time stored in packet header of given acknowledge packet,
- RTT ... the time passed between sending packet and receiving its acknowledgement (Round Trip Time)<sup>4</sup>.

The algorithm says that there are all sent packets set as non-expirable at the beginning of the communication. When some acknowledgement is received, the time difference between sender time and receiver time is computed. This time difference is used for calculating estimated receiver time that is inserted into data and control packet header. Estimated time difference is periodically recalculated based on acknowledgement packets (their sending time is expressed in sender time).

Given algorithm can be used for both simplex and duplex communication.

---

3. This is not true for the first packet received – it is always said to be valid because it is used for time difference computing

4. Instead of RTT we can use SRTT time (Smooth Round Trip Time) that is not so susceptible to sudden network deviations. It can be periodically calculated using this formula:  $SRTT = (\alpha \cdot SRTT) + ((1 - \alpha) \cdot RTT)$ , where  $\alpha = 0.875$ .

**1.4.3 Session Establishment**

Session establishment is done in two stages in the ARTP protocol. In the first stage, a new session must be set up between the communication partners. Depending on the result of the first stage the second one takes place (session creation). After the successful completion of both stages the session is said to be established.

**1. Session arrangement**

The client application must request for available session identifier of new session. When some identifier is available the client application sends control packet with new session request to the server. The server application decides whether the request should be accepted or not and send the result to the client using control packet, too. In the case of positive decision both sides must create arranged session before any data sending.

The typical communication between some client and some server is shown in Fig. 1.8.

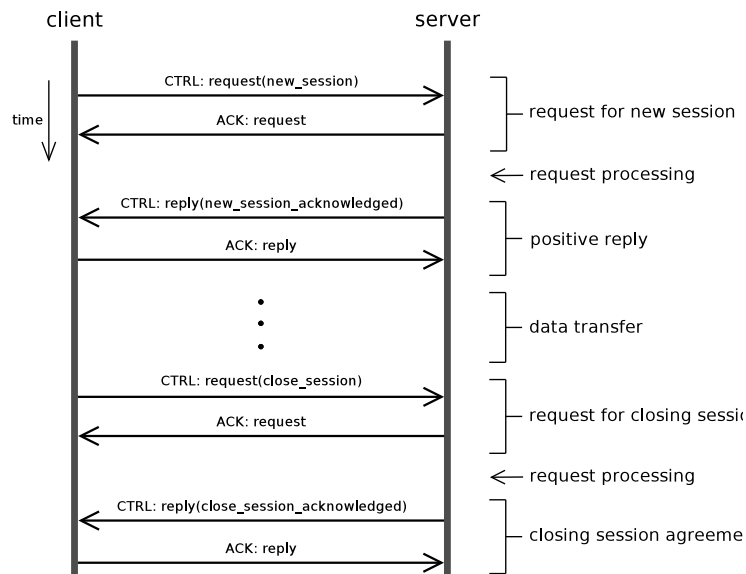


Figure 1.8: Session arrangement between client and server.

**2. Session creation**

The session must be created to ensure the reliable communication be-

tween both communication end points – send and receive buffers are created, necessary structures are initialized and so on.

### 1.4.4 Session Termination

The session termination is done in two stages similar to the session establishment. In the first stage, the termination request must be sent to the communication partner<sup>5</sup>. This stage is not defined by the ARTP itself. The second stage takes place after completion of the first stage – all allocated information is released and session is terminated.

### 1.4.5 Data Communication

#### Data exchange

As soon as the session is established the data transfer may begin. The data exchange between the ARTP and the application is done using data blocks called datagrams. The datagrams may have arbitrary size so they may not pass through the network at once. The sender must fragment them into smaller parts called packets, a packet header is added and it is sent over the network to their receiver. When the receiver receives all fragments of a datagram it reassembles them into original datagram and passes the datagram to the receiver application. The order of passing assembled datagrams is not given – the ARTP guarantees the correct datagrams assembling only.

When a valid packet is received<sup>6</sup>, the receiver must send an acknowledgement of received sequence number to the sender. These sequence numbers may be merged into bigger blocks. The acknowledgement sending interval must be set carefully because too long sending interval may cause too many retransmissions (see the discussion below).

Since the receiver side must detect duplicities of incoming packets, it is necessary that the receiving side must keep and manage sequence numbers history. When packet duplicity is detected, the duplicate packet is acknowledged again but it is not stored into the buffer.

---

5. This stage could be skipped and the session could be terminated immediately and the communicating partner would recognize session termination by other way – see Section 1.4.5.

6. Invalid or expired packets are discarded immediately.

### Retransmissions

To ensure reliable data communication the ARTP provides retransmission mechanism. The sent packet is kept in the send buffer until its acknowledgement is received and then it is thrown away. When this acknowledgement does not come in the specified time (retransmission timeout) the packet is sent again and the whole procedure repeats until the acknowledgement is received or the timeout for its sending is out. When this timeout expires the session is said to be dead.

Note: The retransmission timeout must be recalculated dynamically. One can use the RTT (Round Trip Time) for this calculating. To ensure robustness to sudden network fluctuation it is better to use the SRTT (Smooth Round Trip Time). See footnotes in the chapter 1.4.2 (ARTP timestamps) or documentation of ARTP prototype implementation.

### Congestion control

The congestion control is provided using congestion window in the ARTP. Every session knows the maximum amount of data that may be sent to the receiver without waiting for an acknowledgement and the amount of data actually sent. Another packet may only be sent iff the sum of actual window size and size of the packet is equal or less than the maximum window size.

The congestion window size is increased when an acknowledgement is received. The congestion is detected when packet loss occurs, i. e. when a retransmission occurs, the window size is decreased. When the session is not used for the defined time the congestion window size is set to the initial value in order to avoid sudden receiver congestion.

Congestion window management has great impact on the ARTP transfer performance. Slow window size increase in the case of error-less transfer or steep window size decrease in the case of error occurrence may significantly degrade the ARTP performance. A suggested congestion window management strategy could look as follows:

- the initial window size is set to the three times Maximum Segment Size ( $3 \times \text{MSS}$ ).
- when an acknowledgement is received, the window size is increased by

$$\frac{\text{MSS}^2}{\text{CWND}_{\text{old}}},$$

where  $\text{CWND}_{\text{old}}$  is the previous window size.

- when packet loss is detected, the window size is set to

$$\max(0.5 \cdot \text{CWND}_{\text{old}}, \text{CWND}_{\text{init}})$$

where  $\text{CWND}_{\text{old}}$  is the previous window size and  $\text{CWND}_{\text{init}}$  is its initial size.

### 1.4.6 Interfaces

In this section, we describe the basic user interface of proposed protocol. Every implementation of the ARTP protocol may specify its functionality on its own and that is the reason why the function description here is limited to the very basic functions that have to be implemented in every ARTP implementation.

#### 1. Create session

*Format:*

```
CREATE(session identifier, receiver address,  
        receiver port[, session attributes])
```

The purpose of this function is to create session to allow data communication with specified receiver – it creates all necessary structures, initializes all attributes, etc.

When the parameter `session attributes` is set, this function may set required session behavior as well (e.g. maximum segment size, expiration time, etc.).

#### 2. Close session

*Format:*

```
CLOSE(session identifier, receiver address,  
        receiver port)
```

This function closes previously established session and releases all previously allocated structures.

#### 3. Send datagram

*Format:*

```
SEND(datagram, session identifier,  
      receiver address, receiver port)
```

## 1. ACTIVE ROUTER TRANSPORT PROTOCOL (ARTP)

---

This function is used for sending datagram through defined session. The user data may be sent through established session only. Control information (e.g. requests for new sessions, etc.) may also be sent through non-established session. First of all, the function finds out whether the session is alive and buffers are not full (when limited). If one of these events occurs the function fails.

Datagrams are inserted into packets (as fragments if necessary) which are provided with packet headers and stored into sending buffer. We assume asynchronous environment so that this function ends after saving the packet(s) successfully and does not wait for sending.

### 4. Receive datagram

*Format:*

```
RECEIVE(session identifier, receiver address,  
         receiver port, received datagram)
```

RECEIVE function can be used for receiving datagrams that belong to the indicated session. When the datagram consists of more than one fragment it is first assembled and then passed to the calling application. Calling this function is non-blocking – if the receive buffer is empty, the function ends with appropriate error code.

## Chapter 2

### Implementation and Tests

One part of this work is prototype implementation of proposed ARTP protocol. The source codes are released under BSD license and can be found on the web page <http://www.fi.muni.cz/~xrebok/artp/>. Documentation and demonstration programs are available there as well.

#### 2.1 Prototype Implementation

The prototype implementation is written as a library using C language because of speed requirements. The library was developed and tested on the Linux operating system (Debian with kernel 2.4.26 and Mandrake with kernel 2.4.23). The comments are written in doxygen style and the provided documentation is generated using doxygen tool.

The library implements the whole functionality of proposed ARTP protocol. Due to supposed ARTP development, there were several key requests required – simpleness, legibility, and structureness of the source code. This is the reason why the legibility and coding style was chosen over efficiency if it was necessary.

#### 2.2 How to Use Prototype ARTP Library

It is necessary to initialize all structures and threads to start the ARTP library correctly using `artp_init` function. One may pass the path to ARTP configuration file as a parameter to this function. After the initialization step, the ARTP is prepared to establish new sessions (using the function `artp_prepare_connection`), sending datagrams (`artp_send_dgram`), etc. Detailed information about implementation interface can be found in the documentation of implemented library. The module that wants to use these functions has to include `artp/artp.h` file. When linking the resulting executable, it is necessary to attach the protocol library `libartp`.

### 2.3 Experimental Validation

The functionality of the library was tested using three tests – throughput test, transport reliability and code stability. The network architecture used for all tests is shown in Fig. 2.1 and configuration of PCs used for experiments was as follows:

#### 1. 1 Gbit/s network

DELL PowerEdge 1600 SC, 2 × Intel Xeon CPU 2.8 GHz, 1024 MB memory, Intel PRO/1000 32 bit/66 MHz network interface card, Linux Debian Woody operating system with 2.4.26 kernel.

#### 2. 100 Mbit/s network

Intel Pentium 4 2.0 GHz, 512 MB memory, Intel PRO/100 VE 32 bit 33 MHz network interface card, operating system Linux Debian Woody 2.4.26.



Figure 2.1: Network architecture used for ARTP tests.

All tests were performed on 1 Gbit/s network except for the speed test, that was done on 100 Mbit/s network, too. We have developed specialized testing tool for speed testing and the other tests were implemented using demonstration program mentioned above. The test data was generated pseudo-randomly from `/dev/urandom` device.

The ARTP configuration used for tests was uniform – all buffers were limited to 10 MB (unlimited buffers were used for fragmentation test only) and the maximum segment size was set to 2 KB.

#### 2.3.1 Throughput Measurement

The throughput measurement was focused on determining the maximum transfer rate and its comparison with some existing transport protocol. For

the experiment, we have chosen the most common transport protocol – TCP. The transfer rate was computed by counting transferred data only, i. e. without headers of any protocol (TCP, IP, Ethernet). The maximum bandwidth available on the network was measured using UDP transport protocol for reference.

Both the ARTP and the TCP protocols were measured using three different sizes of transferred datagrams – 2048 B, 5000 B and 10000 B. The results for the TCP protocol are very similar so the results are shown for single datagram size only in graph for the sake of legibility.

Achieved results on the gigabit network are shown in Fig. 2.2, the results on the 100 Mbps network in Fig. 2.3. The following table summarizes average rates achieved for measured protocols and their standard deviations.

Datagram	on 1 Gbps network [Mbps]			on 100 Mbps network [Mbps]		
	UDP	TCP	ARTP	UDP	TCP	ARTP
2048 B	898.46 ± 2.49	882.94 ± 30.15	215.24 ± 9.45	91.75 ± 0.00	90.00 ± 0.00	84.01 ± 1.47
5000 B	–	881.63 ± 27.67	359.82 ± 7.18	–	89.53 ± 0.00	87.50 ± 1.43
10000 B	–	881.27 ± 32.42	387.98 ± 9.72	–	89.17 ± 0.00	87.80 ± 1.48

Table 2.1: Summary of results for throughput measurement of ARTP and TCP transport protocols.

One can see the prototype implementation is competitive on the 100Mbit network but it falls behind on the gigabit network significantly. This may be caused by several factors, but the major is that the prototype implementation is primarily focused on legibility and not optimized for performance. On the other hand production implementation of TCP in Linux is known to be highly tuned even at the cost of “hacking” the code in obscure manner (see [13]).

The results on the gigabit network also show that the achievable throughput of the ARTP protocol depends on datagram size – the throughput increases with increasing datagram size. This behavior is similar to e. g. well known influence of MTU or MSS on transmission efficiency on Ethernet and TCP layers respectively.

### 2.3.2 Reliability Test

This test was designed to verify whether prototype implementation transfers data reliably and without any failures, duplicities or other defects. The test had two parts: the first one verified the reliability of data transport and the second one verified the correctness of fragmentation and reassembly process.

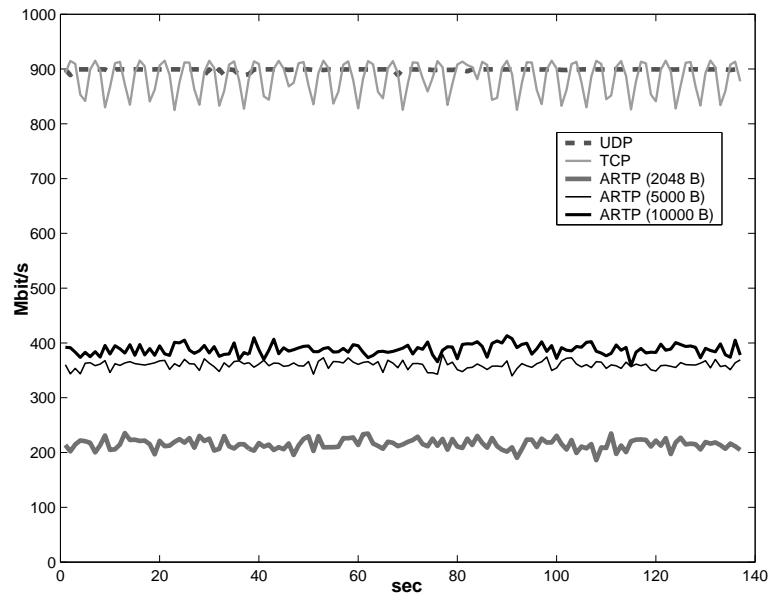


Figure 2.2: Throughput achieved using ARTP and TCP protocols on 1 Gbps network.

### 1. Reliability of data transport

A pseudo-random file was created for the purpose of this test. Its size was 10.15 GB and it was transferred in 10000 B datagrams to the receiver. The whole file was correctly transferred by the ARTP protocol. However, the validation may not be done by simply comparing the source and the resulting files as the ARTP protocol does not ensure right order of datagrams delivered.

We should mention that the time taken by this test differed a lot depending on writing received datagrams into a file or just summing their size only. When making sum of received datagrams size only the test took 3 minutes and 56 seconds (the average speed 352.33 Mbps). But when the received datagrams was written into the file, the elapsed time increased due to waiting for disc I/O to 20 minutes and 7 seconds (the average speed was 69 Mbps).

### 2. Fragmentation correctness

The fragmentation correctness was verified on file transfer – an pseudo-random file was created and its size was 102.57 MB. This file was read

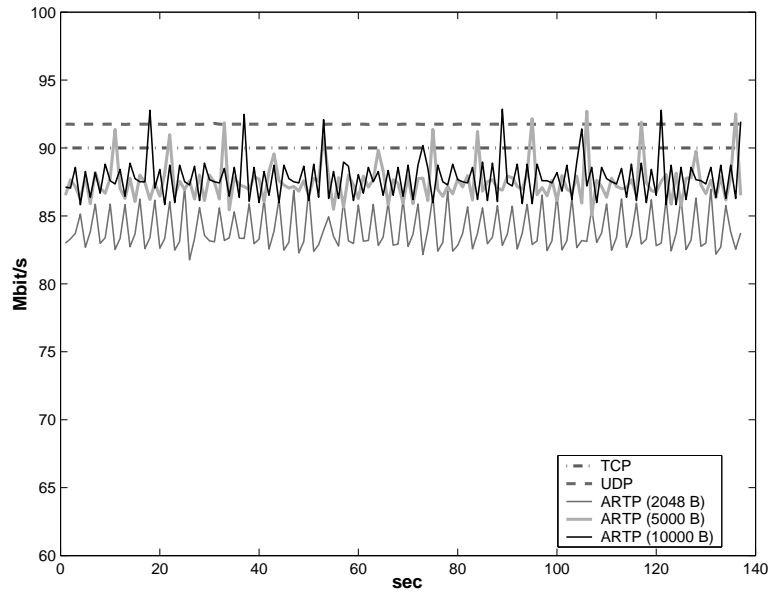


Figure 2.3: Throughput achieved using ARTP and TCP protocols on 100 Mbps network.

into one string and passed as a one datagram to the ARTP protocol<sup>1</sup>. The ARTP had to separate this datagram into fragments and transfer them to the receiver. The receiver had to reassemble all received fragments into original datagram and pass it to the receiver application.

This file was transferred by the ARTP correctly. The fragmentation correctness was verified using MD5 hash algorithm – its result was the same on both sides. Thus we can conclude that the ARTP fragments datagrams correctly.

### 2.3.3 The Stability Test

This test had to verify whether the prototype implementation works correctly in long-term process (especially the correctness of memory management).

The ARTP transferred 9.71 TB of pseudo-random data and this test took exactly 75 hours (the average speed was 301.68 Mbit/s). There were no

1. Unlimited buffers had to be used due to big file transferred.

## 2. IMPLEMENTATION AND TESTS

---

problems found regarding memory – each allocated block was released correctly. That is why we may say that this test was also successful.

## Chapter 3

### Conclusions

The main goal of this work was to design and implement new transport protocol that can be used as a communication layer of active router developed at Masaryk University in Brno. Proposed protocol had to implement reliable communication between the active routers without necessarily preserving right order of received data. This main goal has been successfully accomplished as the proposed protocol implements all features described in this report.

The prototype implementation of proposed ARTP protocol has been developed in order to verify real-world behavior of the protocol. It covers all functionality of the ARTP protocol and its behavior was tested and measured on 100 Mbps and 1 Gbps networks. This implementation was comparable to existing implementations of TCP and UDP transport protocol on 100 Mbps network whereas on 1 Gbps network, it achieved roughly 1/3 of performance of TCP and UDP protocols. The reasons for this behavior are discussed in this report and the next work on the ARTP protocol and especially its implementation should thus focus on efficiency in high-speed networks. Possible ways are these ones:

- making memory management and structures used in created implementation more efficient – especially send and receive buffers; this step could be reached easily due to modular architecture of created prototype implementation,
- proposing better algorithms for congestion window management and evaluating its behavior,
- putting the ARTP into kernel space of Linux/Unix operating systems.

There are also other directions that could be explored in further development of the protocol:

- due to putting the ARTP into operating system kernel, the implementation should support more applications simultaneously,

- tracking development of the active router to keep up with new required features and properties as they arise from the active router.

Proposed protocol can be practically used as a communication layer of active router being developed. Although it is not capable of saturating a high-speed networks, it is still sufficient for lots of applications. The first stage of router development requires full functionality of all its layers and the communication layer provided by the ARTP protocol complies with this requirement.

## Bibliography

- [1] M. Allman, V. Paxson, and W. Stevens. TCP Congestion Control. RFC 2581, April 1999.
- [2] T. Bova and T. Krivoruchka. Reliable UDP protocol. Internet-Draft, February 1999.
- [3] Stefan Covaci (Ed.). *Active Networks*. Springer, GmbH & Co.KG, Tiergartenstrasse 17, Heidelberg, 1999.
- [4] Atanu Ghosh, Michael Fry, and Glen MacLarty. An Infrastructure for Application Level Active Networking. 2000.
- [5] Eva Hladká and Zdeněk Salvet. An Active Network Architecture: Distributed Computer or Transport Medium. In *First International Conference on Networking*, volume 2094 of *Lecture Notes in Computer Science*, pages 612–619. Springer, July 2001.
- [6] Information Sciences Institute, University of Southern California. Internet Protocol. RFC 791, September 1981.
- [7] Information Sciences Institute, University of Southern California. Transmission Control Protocol. RFC 793, September 1981.
- [8] Van Jacobson and Michael J. Karels. Congestion Avoidance and Control. In *ACM SIGCOMM '88*, pages 314–329, Stanford, CA, August 1988.
- [9] D. Murphy. Building an Active Node on the Internet. Technical Report MIT/LCS/TR-723, 1997.
- [10] J. Postel. User Datagram Protocol. RFC 768, August 1980.
- [11] K. Psounis. Active Networks, Applications, Safety, Security, and Applications. *IEEE Communication Surveys Magazine*, 1999.

- [12] W. Richard Stevens, Bill Ferner, and Andrew M. Rudoff. *Unix Network Programming – The Sockets Networking API*. AW, 75 Arlington Street, Suite 300, Boston, 2004.
- [13] S. Ubik and P.Cimbal. Debugging End-to-End Performance in Commodity Operating Systems. *PFLDnet2003, CERN, Geneva, 2003*.
- [14] D. Wetherall, J. Guttag, and D. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *IEEE Conference on Open Architectures and Network Programming*, pages 117–129, April 1998.