Contemporary Control Systems, Inc.
AI-SRVR Sockets API

Revision 1.2
25 June 2014

Table of Contents

OVERVIEW

The AI-SRVR has been designed to provide connectivity between Ethernet and ARCNET networks. It features one Ethernet port, one ARCNET port, and runs on an embedded system utilizing the Linux operating system.

It provides the following services:

- TCP or UDP communications with the Ethernet port.
- Supports multiple Ethernet clients.
- ARCNET side can operate in "promiscuous mode" to receive all ARCNET packets.
- Transmits ARCNET packets to the ARCNET network.
- Receives ARCNET packets from the ARCNET network.
- Can automatically forward received ARCNET packets to Ethernet clients.
- Provides a "Source ID" filter that allows you to specify the ARCNET nodes from which you would like to receive packets.
- Stores received ARCNET packets in "mailboxes fifos" that can be individually read.
- Provides register-level control of the COM20022 ARCNET chip on the AI-SRVR (for advanced users only).

A block diagram of a typical AI-SRVR installation is shown below:

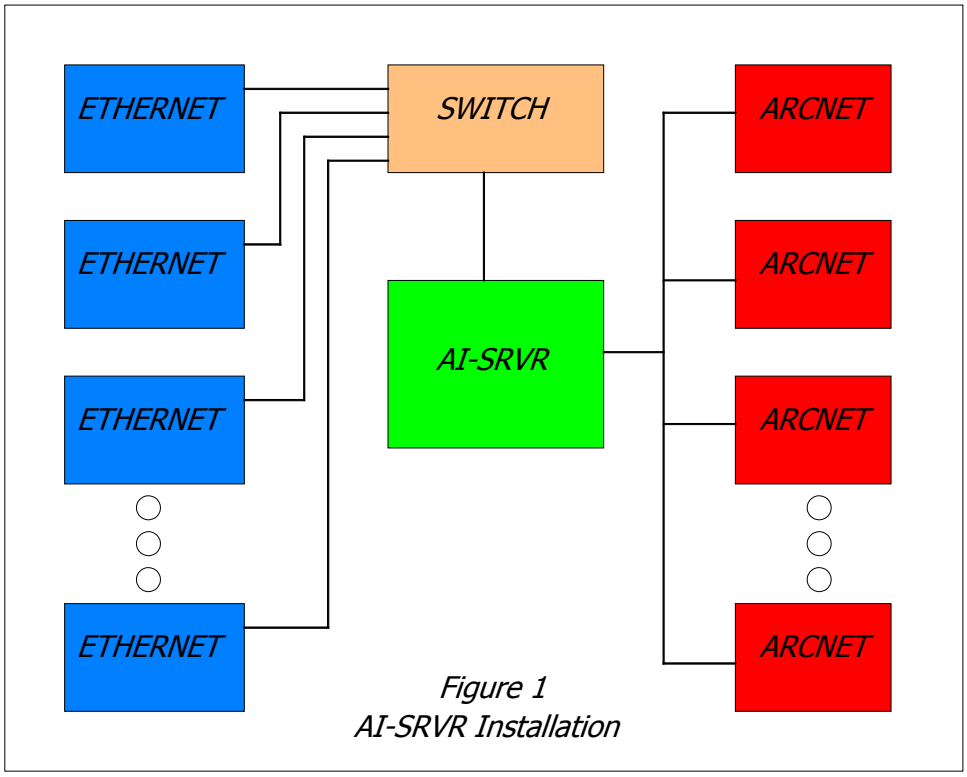

Figure 1
AI-SRVR Installation

The AI-SRVR provides access to the ARCNET network in three ways: 1) All received ARCNET packets are stored in individual "mailbox fifos" for later retrieval, 2) Received ARCNET packets can be automatically sent to the client, and 3) ARCNET packets can be sent from the client through the AI-SRVR to any node on the ARCNET network, including broadcast messages.

All received ARCNET packets are stored in the mailbox fifo's; 5 messages deep by default. When the fifo overflows, old messages are lost. In promiscuous mode, all packets on the ARCNET network are received and stored. Packets in the mailboxes can be retrieved using the "Data Request" command described below.

ARCNET packets can automatically be forwarded to clients when received using the "Data Advise" command. This command allows you to specify a bitmap representing all 256 nodes on the ARCNET network. This bitmap specifies the source ID's of the nodes you would like to receive packets from. After sending the "Data Advise" command, all ARCNET packets sent by the nodes specified in the bitmap will be automatically forwarded to the client.

The "Write Request" command is used to send an ARCNET packet to a node on the ARCNET network.

Other commands are provided that allow control and reporting of many aspects of AI-SRVR operation, as described below.

The sockets interface to the AI-SRVR is based on a command/response format that can be implemented using either TCP or UDP packets. There is a structure for each command and response; these structures are described in the header file "protdef.h". You send a command to the AI-SRVR by transmitting one of the command structures. The AI-SRVR will respond with one of the response structures. ***Data within the structures is binary and little-endian (a short integer is two bytes, lsb then msb) unless otherwise noted***.

Commands

Each command structure begins with one of the following structures:

```
//
// Minimal operation header
//
struct OpHdr
{
    uint16 op_len;      // Length of this operation (incl. header)
    uint16 op_type;     // Type specification for this operation
    uint8  op_data[];   // Parameter / data
};

//
// Secure operation header
//
struct OpSecureHdr
{
    uint16 op_len;      // Length of this operation (incl. header)
    uint16 op_type;     // Type specification for this operation
    uint8  op_security_code[4];  // security code
    uint8  op_data[];   // Parameter / data
};
```

Some commands require the use of the secure header, others don't. The first two fields are identical in each case; the size of the structure and the command number. The values for 'op_type' are defined in the Operations section of the header file. Data fields after the first two (or three) structure elements are different for each command.

NOTE: The 'op_security_code[]' member is always "0000" within 'OpSecureHdr'.

> ***Note the use of non-dimensioned arrays ('op_data[]', etc) within "protdef.h". These arrays of indeterminate size will generate warnings when compiled within a Microsoft environment, but will operate correctly.***

Responses

The AI-SRVR will respond with either an error packet or a data packet specific to the command issued. The error packet format is shown below:

```
//
//  ER - Error Response
//
struct op_er
{
      uint16 op_len;    // Length of this operation (incl. header)
      uint16 op_type;   // Operation type: 'ER' (0x4552)
      uint16 er_opref;  // Type of failed operation
      uint16 er_src;    // Error source
                        //   0: Other
                        //   1: Bridge
                        //   2: ARCNET
      uint32 er_code;   // Error code
      char   er_msg[];  // Error text null-terminated (optional)
};
```

Error codes (er_code) and error messages (er_msg[]) are described in the header file.

A typical response is illustrated by the Config Query response structure; Config Report:

```
//
//   CR - Config Report
//
struct op_cr
{
    uint16 op_len;      // Length of this operation (incl. header)
    uint16 op_type;     // Operation type 'CR' (0x4352)

    struct config cr_config;

    uint16 cr_fifo;     // FIFO depth
    uint16 cr_hwver;    // Hardware-Version Hi/Lo-byte
    uint16 cr_swver;    // Software-Version Hi/Lo-byte
    char   cr_vendor[]; // Vendor-Name, null-terminated
};
```

COMMANDS AND RESPONSES

Configuration Query

You request a configuration report by sending the following command structure:

```
//
//    CQ - Config Query
//
struct op_cq
{
    uint16 op_len;   // Length of this operation (incl. header)
    uint16 op_type;  // Operation type:  'CQ' (0x4351)
    uint8  op_security_code[4];
};
```

If successful, the AI-SRVR will respond with the following structure:

```
//
//    CR - Config Report
//
struct op_cr
{
    uint16 op_len;      // Length of this operation (incl. header)
    uint16 op_type;     // Operation type 'CR' (0x4352)

    struct config cr_config;  // Config structure

    uint16 cr_fifo;     // FIFO depth
    uint16 cr_hwver;    // Hardware-Version Hi/Lo-byte
    uint16 cr_swver;    // Software-Version Hi/Lo-byte
    char   cr_vendor[]; // Vendor-Name, null-terminated
};
```

The 'cr_config' member of the CR response packet is described below:

```
//
//    Config structure
//
struct config
{
    uint8  cf_node;     // Node number bridge (0: not initialized)
    uint8  cf_intf;     // Interface:  0,1
    uint8  cf_speed;    // ARCNET baudrate 0..9
    uint8  cf_timeout;  // ARCNET Timeout (ET1, ET2: 0..3)
    uint8  cf_rcntm;    // RECON Timeout (0..3)
    uint8  cf_rcvall;   // Receive all. 0: off, else: active
    uint16 cf_nltim;    // Timeout Node-List  [ 1/10 s].
};
```

Configuration Set

You send a new ARCNET configuration to the AI-SRVR with the following command structure:

```
//
//    CS - Config Set
//
struct op_cs
{
    uint16 op_len;      // Length of this operation (incl. header)
    uint16 op_type;     // Operation type  'CS' (0x4353)
    uint8  op_security_code[4];
    struct config cs_config;
};
```

The AI-Server code returns "ERR_OP_NOT_ALLOWED" if a configuration operation is attempted during, or within 3 seconds after, a recon.

For compatibility with the existing codebase, this feature must be explicitly enabled in the request. It is suggested to enable this feature for all new applications.

Recon Lockout enabled when bit 7 (0x80) of the 'rcvall' value is set in `struct config`.

Promiscuous Mode Query

You request the status of the 'promiscuous mode' setting with the following command structure:

```
//
// PQ - Promiscuous (Receive-all) Mode Query
//
struct op_pq
{
    uint16 op_len;  // Length of this operation (incl. header)
    uint16 op_type; // Operations type  'PQ' (0x5051)
    uint8  op_security_code[4];
};
```

If successful, the AI-SRVR will respond with the following structure:

```
//
//    PR - Promiscuous (Receive-All) Mode Report
//
struct op_pr
{
    uint16 op_len;    // Length of this operation (incl. header)
    uint16 op_type;   // Operations type 'PR' (0x5052)
    uint8  pr_rcvall; // Receive All. 0: off, else: on
};
```

Promiscuous Mode Set

You send a new promiscuous mode setting with the following command structure:

```
//
//    PS - Promiscuous (Receive-All) Mode Set
//
struct op_ps
{
    uint16 op_len;    // Length of this operation (incl. header)
    uint16 op_type;   // Operation type: 'PS' (0x5053)
    uint8  op_security_code[4];
    uint8  ps_rcvall; // Receive All. 0: off, else: on
};
```

If successful, the AI-SRVR will respond with a PR (Promiscuous Mode Report) as described above.

Status Query

You request a status report with the following command structure:

```
//
// SQ - Status Query
//
struct op_sq
{
    uint16 op_len;  // Length of this operation (incl. header)
    uint16 op_type; // Operations type  'SQ' (0x5351)
    uint8  op_security_code[4];
};
```

If successful, the AI-SRVR will respond with the following structure:

```
//
//   SR - Status Report
//
// Direction: GW --> IP
//
// Possible response: ER, SR
//
struct op_sr
{
    uint16 op_len;            // Length of this operation (incl. header)
    uint16 op_type;           // Operations-typ 'SR' (0x5352)
    //
    uint32 arc_rxPacketCnt;   // Number of received ARCNET packets.
    uint32 arc_txPacketCnt;   // Number of transmitted ARCNET packets.
    //
    uint32 eth_rxPacketCnt;   // Number of received ethernet packets.
    uint32 eth_txPacketCnt;   // Number of transmitted ethernet packets.
    uint32 eth_clientCnt;     // Number of ethernet clients connected.
};
```

Counter Reset

You can reset the packet counters returned in the status query with the following command structure:

```
//
//    CRESQ - Counter reset command.
//
struct op_cresq
{
    uint16 op_len;  // Length of this operation (incl. header)
    uint16 op_type; // Operations type  'OP_CRESQ'
    uint8  op_security_code[4];
};
```

If successful, the AI-SRVR will respond with an 'OpHdr' structure with 'op_type' set to OP_CRESR.

Write Request

You can send an ARCNET packet to a node on the ARCNET network with the following command structure:

```
//
//   WQ - Write reQuest
//
struct op_wq
{
    uint16 op_len;      // Length of this operation (incl. header)
    uint16 op_type;     // Operations type 'WQ' (0x5751)
    uint8  op_security_code[4];
    uint8  wq_data[];   // ARCNET-Telegram
                        // Format: struct arc_pac
};
```

The 'wq_data[]' member is an ARCNET packet as described in the header file (struct arc_pac).

The AI-SRVR will respond with the following structure:

```
//
//    WO - Write O.K.
//
// The 'op_result' member shows the result of the transmit operation.
//            0: Tx complete, Tx acknowledged.
//            1: Timeout, EXCNAK, or RECON.
//            2: Tx not complete.
//            3: ARCNET write failed.
//
struct op_wo
{
    uint16 op_len;      // Length of this operation (incl. header)
    uint16 op_type;     // Operations type  'WO' (0x574F)
    uint32 op_result;   // Status of transmit operation
};
```

Upon receiving a Write Request command, the AI-SRVR can either return immediately after the ARCNET packet is sent (with a return value of '2'), or it can wait for completion of the transmission process (return values '0' or '1'). See the "Flag Status Query" and "Wake On Transmit Complete" commands below.

Data Advise

You can request to receive ARCNET packets sent by particular nodes with the following command structure:

```
//
//    DA - Data Advise
//
struct op_da
{
    uint16 op_len;      // Length of this operation (incl. header)
    uint16 op_type;     // Operation type  'DA' (0x4441)
    uint8  op_security_code[4];
    uint8  da_map[32];  // Bit-Flags for all ARCNET nodes
                        // Bit 7 in da_map[0] = node 0,
                        // Bit 0 in da_map[31] = node 255
    uint16 da_time;     // After this time [1/10 s], the request will be
                        // deleted automatically
                        // 0 = forever
    uint8  da_rcvall;   // 0: only telegrams sent to the bridge,
                        // else: all telegrams
};
```

Former DA operations will be overwritten. A zeroed 'da_map' resets all advise requests so that no data will be sent; 'da_time' and 'da_rcvall' are ignored in this case.

If UDP is used, 'da_time' should not be set to '0'. A 'da_time' value of '0' with UDP packets would result in the Bridge sending packets to 'dead' IP nodes. UDP nodes should repeat this operation periodically.

The 'da_rcvall' flag, when set, indicates that the client is interested in all packets sent by the nodes marked in the 'da_map' array, not just those sent by the nodes to the bridge. It is only meaningful if the bridge is in promiscuous mode.

The 'da_map' is a bitmap; each bit represents one node. The 32 bytes (8 nodes/byte) in the array can represent all 256 nodes on the ARCNET network. If a bit is set, messages sent from the corresponding node will be forwarded to you. If the bridge is in promiscuous mode, you can receive all packets sent by the nodes by setting the 'rcv_all' flag. The bitmap is shown below:

'da_map' Index	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
da_map[0]	0	1	2	3	4	5	6	7
da_map[1]	8	9	10	11	12	13	14	15
				o o o				
da_map[31]	248	249	250	251	252	253	254	255

After sending a 'Data Advise' command, packets will automatically be forwarded to you as they are received. See the 'Data Response' packet below for the format of the received data.

Data Request

When an ARCNET packet is received by the bridge it is automatically stored in a fifo, based on the source ID. You can request ARCNET packets sent by a particular node with the following command structure:

```
//
//    DQ - Data reQuest
//
struct op_dq
{
    uint16 op_len;  // Length of this operation (incl. header)
    uint16 op_type; // Operation type  'DQ' (0x4451)
    uint8  op_security_code[4];
    uint8  dq_node; // ARCNET node number of interest
    uint8  dq_fifo; // number of telegrams in FIFO
};
```

The 'dq_fifo' member specifies the number of packets you'd like to retrieve. If there are fewer packets in a fifo than are requested, the bridge sends a DR packet with 0 bytes and a 'dr_telid' value of 0.

Data Response

ARCNET packets are sent by the bridge to a client in the following response structure:

```
//
//    DR - Data Response
//
struct op_dr
{
    uint16 op_len;    // Length of this operation (incl. header)
    uint16 op_type;   // Operation type  'DR' (0x4452)
    uint32 dr_telid;  // unique ID  of the ARCNET-Telegram
    uint8  dr_fifo;   // FIFO Position of the ARCNET-Telegram (0: latest)
    uint8  dr_data[]; // ARCNET telegram of struct arc_pac
};
```

If a DR packet is sent as a result of a DA (Data Advise), the 'dr_fifo' member is 0.

If a DQ (Data Request) command requests more packets than are stored in a node's fifo, a DR packet is sent with 'dr_telid' = 0, and no data. In this case 'op_len' is 9.

The bridge will respond with DR (Data Response) packets as described below. When using TCP, these packets will be sent back-to-back, but may be spread across multiple TCP packets. Therefore, you must take responsibility for treating DR packets as a byte stream and keep track of packet boundaries, as ARCNET packets will often bridge consecutive TCP packets.

TCP NODELAY Setting

When using a TCP connection, multiple ARCNET packets sent from the AI-SRVR to you (the client) may be combined into a single TCP packet. The TCP stack decides how long to wait before a packet is sent. If you want ARCNET packets to be sent immediately upon receipt, the TCP_NODELAY option should be set.

You can clear, set, or query the TCP_NODELAY setting on the AI-SRVR for a TCP connection with this command structure:

```
//
//    TCP_NODELAY Query/set/clear.
//
struct op_ndq
{
      uint16      op_len;     // Length of this operation (incl.header).
      uint16      op_type;    // Operation: 'NDQ'.
      uint8       op_security_code[4];
      uint16      action;     // Action: 0=set OFF, 1=set ON, 2=report
                              // current status.
};
```

If successful, the AI-SRVR will respond with the following structure:

```
//
// TCP_NODELAY Response.
//
struct op_ndr
{
      uint16      op_len;     // Length of this operation (incl.header).
      uint16      op_type;    // Operation: 'NDR'.
      uint16      status;     // TCP_NODELAY: off=0 on=1.
};
```

ARCNET Register Read/Write

You can read and write individual registers within the COM20022 on the AI-SRVR with the following command structure:

```
//
//    ARCNET Register Read/Write Query.
//
struct op_regq
{
      uint16      op_len;     // Length of this operation (incl.header).
      uint16      op_type;    // Operation: 'REGQ'.
      uint8       op_security_code[4];
      uint8 write;            // 0=read , 1=write.
      uint8 reg;              // ARCNET register number.
      uint8 value;            // Read/write value.
};
```

The AI-SRVR will respond with the following structure:

```
//
//    ARCNET Register Read/Write Response.
//
struct op_regr
{
      uint16      op_len;     // Length of this operation (incl.header).
      uint16      op_type;    // Operation: 'REGR'.
      uint8       value;      // Read/write value.
};
```

The AI-Server code returns "ERR_OP_NOT_ALLOWED" if a register operation is attempted during, or within 3 seconds after, a recon.

For compatibility with the existing codebase, this feature must be explicitly enabled in the request. It is suggested to enable this feature for all new applications.

Recon Lockout enabled when bit 7 (0x80) of the 'reg' value is set.

Flag Status Query

You can request the status of a pending ARCNET transmission with the following command structure:

```
//
//    ARCNET flags query.
//
struct op_flagq
{
      uint16      op_len;     // Length of this operation (incl.header).
      uint16      op_type;    // Operation: 'FLAGQ'.
      uint8       op_security_code[4];
};
```

The AI-SRVR will respond with the following structure:

```
//
//    ARCNET flags response.
//
struct op_flagr
{
      uint16      op_len;     // Length of this operation (incl.header).
      uint16      op_type;    // Operation: 'FLAGR'.
      uint8       tx_complete;
      uint8       tx_acknowledged;
      uint32      recon_count;
};
```

The 'tx_complete' and 'tx_acknowledged' flags are set at the same time. If the 'tx_acknowledged' flag is not set when the 'tx_complete' flag is set, the packet was transmitted, but not acknowledged (as would happen with a broadcast message).

The 'recon_count' member shows the number of recons that have occurred since the last time this query was sent.

Wake On TX Complete

You can request that the AI-SRVR wait until transmission is complete before responding to a Write Request with the following command structure:

```
//
//    Wake on tx complete query.
//
struct op_txwaitq
{
      uint16      op_len;     // Length of this operation (incl.header).
      uint16      op_type;    // Operation: 'TXWAITR'.
      uint8       op_security_code[4];
      uint8       op_value;   // 0=disable , 1=enable , 2= report value.
};
```

The AI-SRVR will respond with the following structure:

```
//
// Wake on tx complete response.
//
// Direction: GW -> IP
//
struct op_txwaitr
{
      uint16      op_len;     // Length of this operation (incl.header).
      uint16      op_type;    // Operation: 'TXWAITR'.
      uint8       op_value;   // 0=disabled , 1=enabled.
};
```

When Wake On TX Complete is enabled, the AI-SRVR will not send a response to a Write Request until the transmission is complete.

Wake On Recon

You can request that the AI-SRVR send a message when a recon occurs with the following command structure:

```
//
//    Wake on recon query.
//
struct op_reconq
{
      uint16      op_len;     // Length of this operation (incl.header).
      uint16      op_type;    // Operation: 'RECONR'.
      uint8       op_security_code[4];
      uint8       op_value;   // 0=disable , 1=enable , 2=report value.
};
```

The AI-SRVR will respond with the following structure:

```
//
//    Wake on recon response.
//
struct op_reconr
{
      uint16      op_len;     // Length of this operation (incl.header).
      uint16      op_type;    // Operation: 'RECONR'.
      uint8       op_value;   // 0=disabled , 1=enabled , 2=error.
};
```

When Wake On Recon is enabled, the AI-SRVR will send a response using an 'OpHdr' structure with the 'op_type' member set to OP_RECONM (Recon Message), whenever a recon occurs.

Transmit Timeout

You can set the ARCNET transmit timeout value with the following command structure:

```
//
//    Set ARCNET TX timeout value.
//
struct op_txtimeoq
{
      uint16      op_len;     // Length of this operation (incl.header).
      uint16      op_type;    // Operation: 'TXTIMEOQ'.
      uint8       op_security_code[4];
      uint32      op_timeout; // Timeout in milliseconds.
};
```

The AI-SRVR will respond with the following structure:

```
//
//    TX timeout response.
//
struct op_txtimeor
{
      uint16      op_len;     // Length of this operation (incl.header).
      uint16      op_type;    // Operation: 'TXTIMEOR'.
      uint32      op_result;  // 0=Success 1=failure.
};
```

When you send an ARCNET packet using the Write Request command, the AI-SRVR attempts to send the ARCNET packet using the ARCNET driver. When the driver has loaded the packet into the COM20022 chip and transmission has been requested, the driver tells the stack to wait while transmission is in process. When transmission completes normally, the interrupt service routine tells the stack it's ok to begin sending ARCNET packets again. If recons are occurring, or if the destination node keeps NAKing the free buffer inquiries, the transmit interrupt will not occur, and the stack will detect that a timeout has occurred. The TX timeout value is the number of milliseconds the stack will wait for an ARCNET packet to complete transmission.