```
    // Bind the channel and open our local send port.
    mcapi_pktchan_connect_i(work_request_out_endpoint,
work_request_remote_endpoint, &request, &status);
    CHECK_STATUS(status);
    mcapi_pktchan_send_open_i(&work_requests_out_hndl[i],
work_request_out_endpoint, &request, &status);
    CHECK_STATUS(status);


    // Repeat the process to create an ack scalar channel from the
    // worker back to us.
    mcapi_endpoint_t ack_in_endpoint = mcapi_endpoint_create(MCAPI_PORT_ANY,
&status);
    CHECK_STATUS(status);
    mcapi_endpoint_t ack_remote_endpoint = mcapi_endpoint_get(DOMAIN_0,
worker_node, WORKER_ACK_PORT_ID, &status);
    CHECK_STATUS(status);

    mcapi_sclchan_connect_i(ack_remote_endpoint, ack_in_endpoint, &request,
&status);
    CHECK_STATUS(status);

    mcapi_sclchan_recv_open_i(&acks_in_hndl[i], ack_in_endpoint, &request,
&status);
    CHECK_STATUS(status);
  }
}


void dispatch_packets()
{
  PacketInfo packet_info;
  mcapi_status_t status;

  while(get_next_packet(&packet_info))
  {
    // Because we maintain "session state" across packets, each
    // incoming packet is associated with a particular worker.
    int worker = packet_info.worker;

    // Each worker sends back acks when it is ready for more work.
    if (mcapi_sclchan_available(acks_in_hndl[worker], &status))
    {
      // An ack is available; pull it off and send more work
      int unused_result = mcapi_sclchan_recv_uint8(acks_in_hndl[worker],
&status);
      mcapi_pktchan_send(work_requests_out_hndl[worker], &packet_info,
sizeof(packet_info), &status);
      CHECK_STATUS(status);
    }
    else
    {
      // No ack available; drop the packet (or queue, or do some other
      // form of exception processing) and move on.
      drop_packet(&packet_info);
    }
  }
}
```