Using Qualnet – Part II

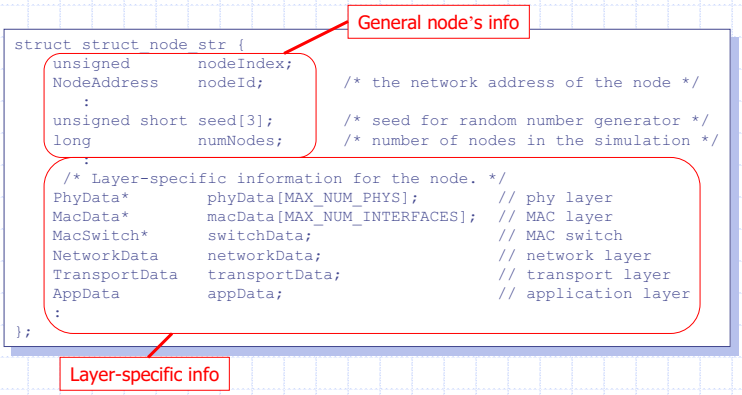
Adding a Custom Protocol

Qualnet's Directory Structure

- `$QUALNET_HOME`
  - `application` – code for the application layer protocols and traffic generators
  - `bin` – executable and configuration or input/output files
  - `gui` – the Visual Environment Toolset
  - `include` – common include files
  - `mac` – the code for the MAC layer protocols
  - `main` – the basic framework design
  - `mobility` – the code for the mobility models
  - `network` – the code for the network layer protocols and routing protocols
  - `phy` – the code for the physical layer models
  - `transport` – the code for the transport layer protocols (TCP/UDP, RSVP, etc)
  - `tcplib` – User models for simulating TCP applications such as FTP and Telnet
  - `verification` – Sample files and outputs to verify protocol correctness

Qualnet Layered Architecture

- The simulation is a collection of network nodes, each with its own protocol stack parameters and statistics
  - File `addons/seq/node.h`



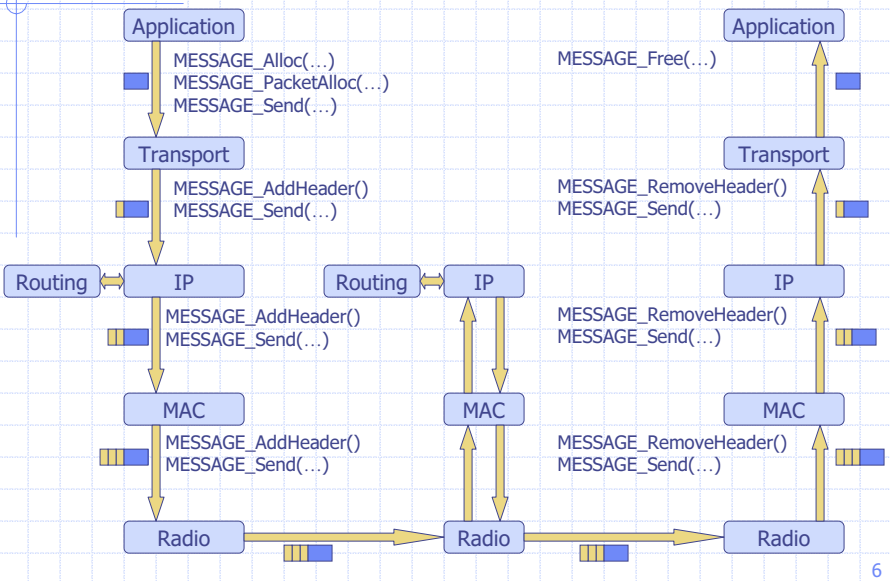
Messages, Packets, and Timers

- A message is a unit defining an interaction between protocols and between nodes
- Two types of messages
  - Packets – used for communication between nodes
  - Timers – allow protocols to schedule events in a future time

Message-Related API Functions

MESSAGE_Alloc()	Allocate a message and provide it with standard event, layer, protocol info
MESSAGE_InfoAlloc()	Allocate additional user-specified space for optional information about the event
MESSAGE_PacketAlloc()	Allocate space for the packet within the message
MESSAGE_AddHeader()	Add a header to the packet (usually called by each layer in the protocol stack)
MESSAGE_RemoveHeader()	Remove a header from the packet
MESSAGE_AddVirtualPayload()	Add virtual payload to a Message (increase tx delay without increasing array size)
MESSAGE_Duplicate()	Copy the message, including its packet and user-specified space (info field)
MESSAGE_Send()	Send the message as an event to the specified layer and protocol
MESSAGE_Free()	Free the message, once it has reached its final destination

Packet Life Cycle



Creating Messages

```
Message*
MESSAGE_Alloc(
    Node *node,

    int layerType,

    int protocol,

    int eventType);
```

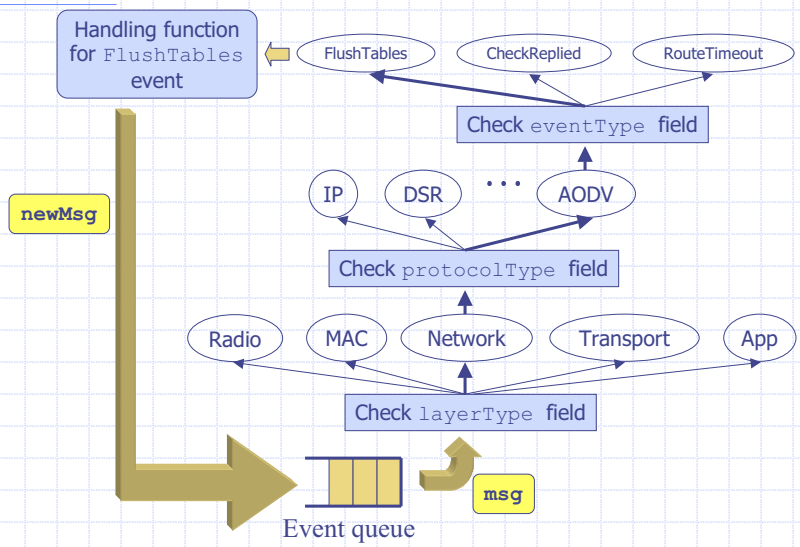
A pointer to the node creating the message

The stack layer at which this message will be processed next
e.g., NETWORK_LAYER

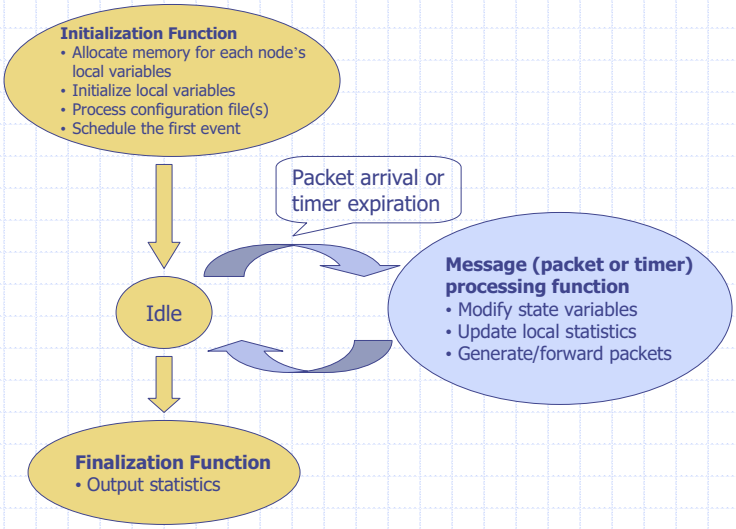
The specific protocol at the layer which will process this message
e.g., ROUTING_PROTOCOL_DSR

The event that this message represents
e.g., MSG_NETWORK_FlushTables

Message Processing



Qualnet's Protocol Modeling



Adding a Protocol to Qualnet

- Determine what layer your protocol will operate at
- Implement four/five main functions
  - Initialization function
  - Packet/event handling function
  - Router function (for routing protocol)
  - Finalization function
- Hook up the above functions to the protocol dispatching functions of the corresponding layer

Example: Adding a New Routing Protocol

- Simplified Routing Information Protocol (SRIP)
  - Table-driven, distance vector protocol
  - Using periodic route update, no triggered update, no split horizon
  - Working properly in static networks with only a small number of nodes (no node failure)
  - Supporting only one interface (wireless) per node

Distributed Bellman-Ford Algorithm

- What local information is maintained by each node?

Routing Table

Destination	Next hop	Cost
:	:	:

Initial routing table for *A*

Destination	Next hop	Cost
A	*A*	0
B	-	∞
:	-	∞

- What information is exchanged between neighboring nodes?

Route Advertisement

Destination	Cost
:	:

- How a node processes a route advertisement?
  - A node *A* updates its entry for destination *D* only when the advertised cost to *D* is lower than its current cost

SRIP Header File (network/srip.h)

```
#ifndef _SRIP_H_
#define _SRIP_H_

#define SRIP_INFINITY  16

typedef struct srip_table_entry
{
   NodeAddress  destination;
   NodeAddress  nextHop;
   unsigned int distance;
} SripTableEntry;

typedef struct srip_str
{
   clocktype       updateInterval;
   SripTableEntry* routingTable;

   /* statistic */
   unsigned int numRouteUpdatesBroadcast;
} SripData;

void SripInit(Node* node, SripData** sripPtr,
              const NodeInput* nodeInput, int interfaceIndex);
void SripHandleProtocolEvent(Node* node, Message* msg);
void SripHandleProtocolPacket(Node* node, Message* msg,
                              NodeAddress sourceAddress);
void SripFinalize(Node *node);
void SripRouterFunction(Node* node, Message* msg, NodeAddress destAddr,
                        NodeAddress previousHopAddress, BOOL* packetWasRouted);

#endif
```

Routing table entry

Local SRIP variables per nodes

Function prototypes to be recognized by Qualnet's network layer (IP)

SRIP Initialization Function

- Called when each node is initialized by Qualnet

```
void SripInit(Node*            node,
              SripData**       sripPtr,
              const NodeInput* nodeInput,
              int              interfaceIndex)
{
   int i; BOOL retVal;
   Message* newMsg;
   SripData* srip;

   if (MAC_IsWiredNetwork(node, interfaceIndex))
      ERROR_ReportError("SRIP supports only wireless interfaces");

   if (node->numberInterfaces > 1)
      ERROR_ReportError("SRIP only supports one interface of node");

   /* allocate memory for SRIP's variables for this node */
   (*sripPtr) = (SripData*) MEM_malloc(sizeof(SripData));
   srip = *sripPtr;

   /* read parameter from the configuration file */
   IO_ReadTime(node->nodeId,
               ANY_ADDRESS,
               nodeInput,
               "SRIP-UPDATE-INTERVAL",
               &retVal,
               &(srip->updateInterval));
   if (retVal == FALSE)
      ERROR_ReportError("SRIP-UPDATE-INTERVAL not specified!");
```

Make sure the node has exactly one wireless interface

Allocate memory for local SRIP variables within each node

Read SRIP parameter from the main config file

(to be continued)

SRIP Initialization Function

(continued)

```
    /* allocate and initialize the routing table for this node */
    /* Note: (n+1) entries are allocated for convenience       */
    srip->routingTable = (SripTableEntry*)
        MEM_malloc(sizeof(SripTableEntry)*node->numNodes + 1);
    for (i = 1; i <= node->numNodes; i++) {
       srip->routingTable[i].destination = i;
       srip->routingTable[i].nextHop     = INVALID_ADDRESS;
       srip->routingTable[i].distance    = SRIP_INFINITY;
    }
    srip->routingTable[node->nodeId].nextHop  = node->nodeId;
    srip->routingTable[node->nodeId].distance = 0;

    /* Initialize statistic */
    srip->numRouteUpdatesBroadcast = 0;

    /* Tell IP to use our function to route packets */
    NetworkIpSetRouterFunction(node,
                               &SripRouterFunction,
                               interfaceIndex);

    /* schedule the very first route update broadcast */
    newMsg = MESSAGE_Alloc(node, NETWORK_LAYER,
                  ROUTING_PROTOCOL_SRIP, MSG_NETWORK_RTBroadcastAlarm);
    MESSAGE_Send(node, newMsg, pc_nrand(node->seed)%srip->updateInterval);
}
```

Initialize routing table

Initialize statistic

Register router function with IP

Schedule the first route advertisement timer

SRIP Event Handling Function

- Called when a node's timer expires

```
void SripHandleProtocolEvent(Node* node, Message* msg)
{
   int i, numEntries = 0, pktSize;
   Message*  newMsg;
   char*     pktPtr;

   /* Obtain a pointer to the local variable space */
   SripData* srip = (SripData*)
      NetworkIpGetRoutingProtocol(node, ROUTING_PROTOCOL_SRIP);

   for (i = 0; i < node->numNodes; i++) {
      if (srip->routingTable[i].distance < SRIP_INFINITY)
         numEntries++;
   }

   newMsg = MESSAGE_Alloc(node, 0, 0, 0);
   pktSize = sizeof(unsigned int) + sizeof(SripTableEntry)*numEntries;
   MESSAGE_PacketAlloc(node, newMsg, pktSize, TRACE_ANY_PROTOCOL);
   pktPtr = newMsg->packet;
   memcpy(pktPtr, &numEntries, sizeof(unsigned int)); /* number of entries */
   pktPtr += sizeof(unsigned int);

   /* Fill the packet with the valid table entries */
   for (i = 1; i <= node->numNodes; i++) {
      if (srip->routingTable[i].distance < SRIP_INFINITY) {
         memcpy(pktPtr, &(srip->routingTable[i]), sizeof(SripTableEntry));
         pktPtr += sizeof(SripTableEntry);
      }
   }
```

(to be continued)

Obtain pointer to local variable space

Prepare a route advertisement packet

Count the number of valid entries

SRIP Event Handling Function

(continued)

```
    /* Send the route update packet to MAC layer */
    NetworkIpSendRawMessageToMacLayer(
        node,               /* node pointer */
        newMsg,             /* raw message */
        node->nodeId,       /* source address */
        ANY_DEST,           /* destination address */
        CONTROL,            /* priority */
        IPPROTO_SRIP,       /* IP Protocol */
        1,                  /* TTL */
        DEFAULT_INTERFACE,  /* output interface */
        ANY_DEST);          /* next hop address */

    /* update statistic */
    srip->numRouteUpdatesBroadcast++;

    /* schedule the next route update broadcast */
    newMsg = MESSAGE_Alloc(node, NETWORK_LAYER,
                ROUTING_PROTOCOL_SRIP, MSG_NETWORK_RTBroadcastAlarm);
    MESSAGE_Send(node, newMsg, srip->updateInterval);
}
```

Ask IP to add header and send packet to MAC layer

Update local statistic

Schedule the next broadcast event

SRIP Packet Handling Function

- Called when a node receives a route advertisement

```
void SripHandleProtocolPacket(Node*       node,
                              Message*    msg,
                              NodeAddress sourceAddress)
{
    SripData* srip = (SripData*)
        NetworkIpGetRoutingProtocol(node, ROUTING_PROTOCOL_SRIP);
    int i, numEntries;
    char *pktPtr;
    SripTableEntry entry;

    pktPtr = msg->packet;
    memcpy(&numEntries, pktPtr, sizeof(unsigned int));
    pktPtr += sizeof(unsigned int);

    /* scan the entry list and update routing table only with entries with
     * shorter distance */
    for (i = 0; i < numEntries; i++)
    {
        memcpy(&entry, pktPtr, sizeof(SripTableEntry));
        entry.distance++;
        if (entry.distance < srip->routingTable[entry.destination].distance) {
            srip->routingTable[entry.destination].distance = entry.distance;
            srip->routingTable[entry.destination].nextHop  = sourceAddress;
        }
        pktPtr += sizeof(SripTableEntry);
    }

    MESSAGE_Free(node, msg);
}
```

Obtain pointer to local variable space

Packet format

#entries	1st entry	2nd entry	…

Free the message since this is its final destination

SRIP Router Function

- Called when IP layer receives a data packet from MAC or transport

```
void SripRouterFunction(Node*       node,
                        Message*    msg,
                        NodeAddress destAddr,
                        NodeAddress previousHopAddress,
                        BOOL*       packetWasRouted)
{
   IpHeaderType *ipHeader = (IpHeaderType *) msg->packet;

   /* Obtain a pointer to the local variable space */
   SripData* srip = (SripData*)
      NetworkIpGetRoutingProtocol(node, ROUTING_PROTOCOL_SRIP);

   /* do not route any SRIP packet, or any packets destined to myself */
   if (ipHeader->ip_p == IPPROTO_SRIP || ipHeader->ip_dst == node->nodeId)
      return;

   /* route the packet only when the destination is considered reachable */
   if (srip->routingTable[ipHeader->ip_dst].distance < SRIP_INFINITY)
   {
      *packetWasRouted = TRUE;
      NetworkIpSendPacketToMacLayer(
           node,
           msg,
           DEFAULT_INTERFACE,
           srip->routingTable[ipHeader->ip_dst].nextHop);
   }
}
```

Ignore SRIP packets and my own packets

Route the packet if the destination is reachable

SRIP Finalizing Function

- Called at each node when Qualnet is terminating

```
void SripFinalize(Node *node)
{
   char buf[MAX_STRING_LENGTH];

   /* Obtain a pointer to the local variable space */
   SripData* srip = (SripData*)
      NetworkIpGetRoutingProtocol(node, ROUTING_PROTOCOL_SRIP);

   sprintf(buf, "Number of Route Updates Broadcast = %u",
      srip->numRouteUpdatesBroadcast);
   IO_PrintStat(node, "Network", "SRIP", ANY_DEST, -1, buf);
}
```

Report statistic

Make SRIP Recognized by Qualnet

- Let Qualnet know SRIP as a network layer protocol
  - In the file `include/network.h`

```
typedef
enum
{
    NETWORK_PROTOCOL_IP = 0,
    NETWORK_PROTOCOL_MOBILE_IP,
    :
    :
    ROUTING_PROTOCOL_IGRP,
    ROUTING_PROTOCOL_SRIP,
//InsertPatch ROUTING_PROTOCOL_TYPE
    :
    ROUTING_PROTOCOL_ALL,
    ROUTING_PROTOCOL_NONE
}
NetworkRoutingProtocolType;
```


Make SRIP Recognized by Qualnet

- Let IP module know SRIP as an IP protocol
  - In the file `network/ip.h`

```
//
// IP protocol numbers for network- and transport-layer protocols.
//

#define IPPROTO_ICMP            1
#define IPPROTO_IGMP            2
:
#define IPPROTO_DVMRP           200
#define IPPROTO_SRIP            234
//InsertPatch ROUTING_IPPROTO
```

Make SRIP Recognized by Qualnet

- Have IP module recognize the five entry functions of SRIP
  - Have `network/ip.c` include `srip.h`

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
:
:
#include "pim.h"
#include "access_list.h"

//#define DEBUG_FIX

#include "srip.h"
//InsertPatch HEADER_FILES
:
```

(file `network/ip.pc`)

Make SRIP Recognized by Qualnet

- Have IP initialize SRIP if specified in the configuration file
  - In the file `network/ip.c`, function `NetworkIpInit()`

```
void
NetworkIpInit(Node *node, const NodeInput *nodeInput)
{
    NetworkDataIp *ip = (NetworkDataIp *) node->networkData.networkVar;
            :
        IO_ReadString(
            node->nodeId,
            NetworkIpGetInterfaceAddress(node, i),
            nodeInput,
            "ROUTING-PROTOCOL",
            &retVal,
            protocolString);

        if (retVal)
        {
            :
            else
            if (strcmp(protocolString, "SRIP") == 0) {
                NetworkIpAddUnicastRoutingProtocolType(node, ROUTING_PROTOCOL_SRIP, i);

                if (!NetworkIpGetRoutingProtocol(node, ROUTING_PROTOCOL_SRIP)) {
                    SripInit(node,
                             (SripData **) &ip->interfaceInfo[i]->routingProtocol,
                             nodeInput, i);
                }
                else {
                    NetworkIpUpdateUnicastRoutingProtocolAndRouterFunction(
                       node, ROUTING_PROTOCOL_SRIP, i);
                }
            }
//InsertPatch NETWORK_INIT_CODE
            :
```

Make SRIP Recognized by Qualnet

- When the network layer receives an event for SRIP, dispatch it to SRIP's event handling function
  - In the file, `network/ip.c`, function `NetworkIpLayer()`

```
void
NetworkIpLayer(Node *node, Message *msg)
{
    switch (msg->protocolType)
    {
        :
        case ROUTING_PROTOCOL_ALL:
        {
            ERROR_Assert(FALSE, "IP event error");
            //HandleSpecialMacLayerStatusEvents(node, msg);
            break;
        }
        case ROUTING_PROTOCOL_SRIP:
        {
            SripHandleProtocolEvent(node, msg);
            break;
        }
//InsertPatch NETWORK_IP_LAYER
        :
```


Make SRIP Recognized by Qualnet

- When IP receives an SRIP route advertisement packet, dispatch it to SRIP's packet handling function
  - In the file, `network/ip.pc`, function `DeliverPacket()`

```
static void //inline//
DeliverPacket(Node *node, Message *msg,
              int interfaceIndex, NodeAddress previousHopAddress)
{
    NetworkDataIp *ip = (NetworkDataIp *) node->networkData.networkVar;
    :
    ipHeader = (IpHeaderType *) msg->packet;
    ipProtocolNumber = ipHeader->ip_p;

    if (ipHeader->ip_tos & IPTOS_CE) {
        aCongestionExperienced = TRUE;
    }

    switch (ipProtocolNumber)
    {
        :
        case IPPROTO_SRIP:
        {
            SripHandleProtocolPacket(
                node,
                msg,
                sourceAddress);

            break;
        }
//InsertPatch NETWORK_HANDLE_PACKET
        :
```

Make SRIP Recognized by Qualnet

- Call SRIP's finalizing function when IP is terminating
  - In the file `network/ip.pc`, function `NetworkIpFinalize()`

```
void
NetworkIpFinalize(Node *node)
{
    NetworkDataIp *ip = (NetworkDataIp *) node->networkData.networkVar;
    :
    for (i = 0; i < node->numberInterfaces; i++)
    {
        switch (NetworkIpGetUnicastRoutingProtocolType(node, i))
        {
            case ROUTING_PROTOCOL_LAR1:
            {
                Lar1Finalize(node);
                break;
            }
            :
            case ROUTING_PROTOCOL_SRIP:
            {
                SripFinalize(node);
                break;
            }
//InsertPatch FINALIZE_FUNCTION
            :
```


Compiling Qualnet with SRIP

- Edit `main/Makefile-common`
  - Add `srip.h` and `srip.c` to SIM_HDRS and SIM_SRCS macros, respectively

```
SIM_HDRS = \
$(ADDON_HDRS) \
\
:
../mac/aloha.h  \
\
../network/srip.h \
#InsertPatch HEADER_FILES

:
SIM_SRCS = \
$(ADDON_SRCS) \
\
:
\
../mac/aloha.c  \
\
../network/srip.c \
#InsertPatch SOURCE_FILES
```

Compiling Qualnet with SRIP

- Rebuild the .h dependencies:

```
cd $QUALNET_HOME/main
make depend
```

- Rebuild Qualnet executable:

```
make
```


Creating a configuration file for SRIP

- In `$QUALNET_HOME/bin` directory, copy `default.config` into `srip.config`, then modify/add the following parameters:

```
EXPERIMENT-NAME       srip
:
ROUTING-PROTOCOL   SRIP
SRIP-UPDATE-INTERVAL 10S
:
```

Testing the Protocol

- In $QUALNET_HOME/bin directory, run qualnet with SRIP configuration file:

```
cd $QUALNET_HOME/bin
./qualnet srip.config
```

- Check `srip.stat` file to see applications' statistics

Obtaining SRIP Example

- Two files, `srip.h` and `srip.c`, are located in `/m/buckwheat/qualnet/examples`
- Check `README` file for instructions

Programming Tips

- `MESSAGE_Free()` must be called only once per message
- Run `make depend` to rebuild .h dependencies whenever different header files are included
- Filling data into a packet or a packet header can be tricky
  - A field may span across a word boundary, causing '*bus error*' in some systems
  - Use `memcpy()` instead of an assignment operation (i.e., =)
- Use message's info field to carry extra information internally (within the same node)
- A good way to learn Qualnet is to study the code of some provided protocols