Honeywell

Manage UCN Communications

L53682
UCN

Total Plant®

Copyright, Notices, and Trademarks

Revision 05 – April 7, 1998

Honeywell IAC courseware is subject to change without notice.

Honeywell
Industrial Automation and Control
Automation College
2820 West Kelton Lane
Phoenix, AZ 85053

1-800-852-3211

Table of Contents

Figures and Tables

Acronyms

AM.....Application Module
APM..... Advanced Process Manager
CL..... Control Language
CL/AM..... Control Language for Application Module
DDT .....Data Definition Table
DO..... Data Out
EB .....Exception Build
I/O.....Input/Output
IOP .....Input/Output Processor
LCN .....Local Control Network
LM ..... Logic Manager
NIM ..... Network Interface Module
PM..... Process Manager
PMM ..... Process Manager Module
TAC .....Technical Assistance Center
UCN.....Universal Control Network
US ..... Universal Station

Parameters

HIGHAL..... High Alarm
INITREQ.....Initialization Request
INITVAL.....Initialization Value
PTINAL .....Point in Alarm
PV ..... Process Variable
UCNSCANT.....Universal Control Network Scan Time

References

Publication Title	Publication Number	Binder Title	Binder Number
For R5xx :			
HPM Service	HP13-500	PM/APM/HPM Service-1	TPS 3061-1
PM/APM Service	AP13-500	PM/APM/HPM Service-1	TPS 3061-1
UCN Guidelines	UN20-500	Installation/UCN	TPS 3041
HPM Implementation Guidelines	HP12-500	Implementation/HPM-1	TPS 3066-1
APM Implementation Guidelines	AP12-500	Implementation/APM-1	TPS 3042-1
PM Implementation Guidelines	PM12-500	Implementation/PM-1	TPS 3040-1
For R4xx:			
APM Service	AP13-400	PM/APM Service	TPS 2061
PM Service	PM13-400	PM/APM Service	TPS 2061
UCN Guidelines	UN12-410	Installation/UCN	TPS 2041
APM Implementation Guidelines	AP12-400	Implementation/APM	TPS 2042-1
PM Implementation Guidelines	PM12-400	Implementation/PM	TPS 2040-1

Introduction

Module Overview

About this module

This course module provides information for understanding, planning, and managing peer-to-peer communication on the UCN.

Peer-to-peer communication is very easy to use—and abuse. To use this feature properly, you must have an understanding of how the LCN and UCN communicate, as well as an understanding of how UCN nodes communicate with each other. For example, a UCN overrun problem may appear as being peer-to-peer related, but the problem source may actually be from the LCN (such as many LCN schematics with unusually heavy parameter requests.)

Objectives

The objectives of this course module are to

- Interpret conditions on the LCN and UCN that contribute to UCN loading,
- Determine the load a UCN node places on the UCN,
- Resolve UCN peer-to-peer issues,
- Use effective techniques for peer-to-peer communication.

Who this module is for

This module is intended for personnel who are responsible for UCN configuration issues. This includes engineers who configure data points, system administrators, and engineering technicians.

Sample test items

This course module's Criterion Test includes the following items:

Your course manager may choose to hand out a hypothetical case study based on situations that have been diagnosed and resolved by TAC in the past. After reading the case study, be prepared to join in a class discussion on possible causes of the problems described, ways to analyze the problem further, and feasible resolutions.

Continued on next page

A note about our examples

Our examples will reference the Advanced Process Manager as in Figure 1, but also apply to the High Performance Process Manager, Process Manager, Safety Manager, and Logic Manager unless otherwise noted. The examples are kept as simple as possible to illustrate concepts that will help you configure and troubleshoot your system properly.

Summary

There are no quick easy answers on peer-to-peer communication, you must look at how the whole system interacts. Later in this course module, some guidelines and recommendations are given that have had successful results at various sites.

LCN to UCN Communication

Operation Concepts

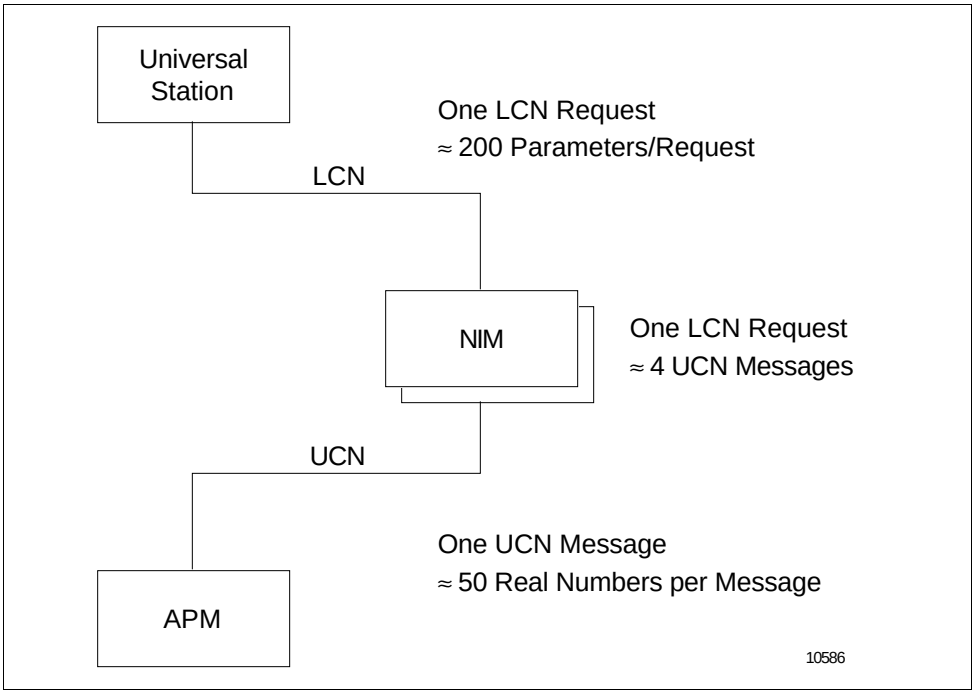
Introduction

This section provides a basic understanding of the NIM, and how it translates and transmits an LCN message to the UCN. After completing this section, you will have a better understanding of how the LCN contributes to UCN communication traffic.

LCN request and UCN message

Consider our example system where a Universal Station does a display update every 4 seconds and requests the data from an APM. Assuming that an LCN request in this example consists of approximately 200 real parameters, the NIM translates this request into approximately four UCN messages. Because a UCN message is smaller than an LCN request, each UCN message could consist of, for example, 50 real numbers.

Figure 1 LCN Request to UCN Message



Concept

The concept in Figure 1 shows that depending on the type of data in the LCN request (reals, booleans, enumerations), the data can be translated into one or more UCN messages. (Note that fewer UCN messages are required if the request includes enumerations and logicals.) It is by no accident that we chose a US request, because as you know, requests from the US are typically from schematics. You can see right away that optimizing LCN requests (such as US schematics) reduces UCN load.

Continued on next page

Calculations are not necessary

Because LCN requests can also include enumerations and booleans (which require less words in a UCN message), calculations of how LCN requests translate to UCN messages can become complex. You can spend a lot of time calculating how many UCN messages are created from an LCN request, but there is no need to do so. One reason is that diagnostic displays provide information (such as parameter requests, transactions). Another reason is that your main goal is to optimize LCN requests.

Part of the solution

Because your goal is optimizing requests, part of the your solution includes

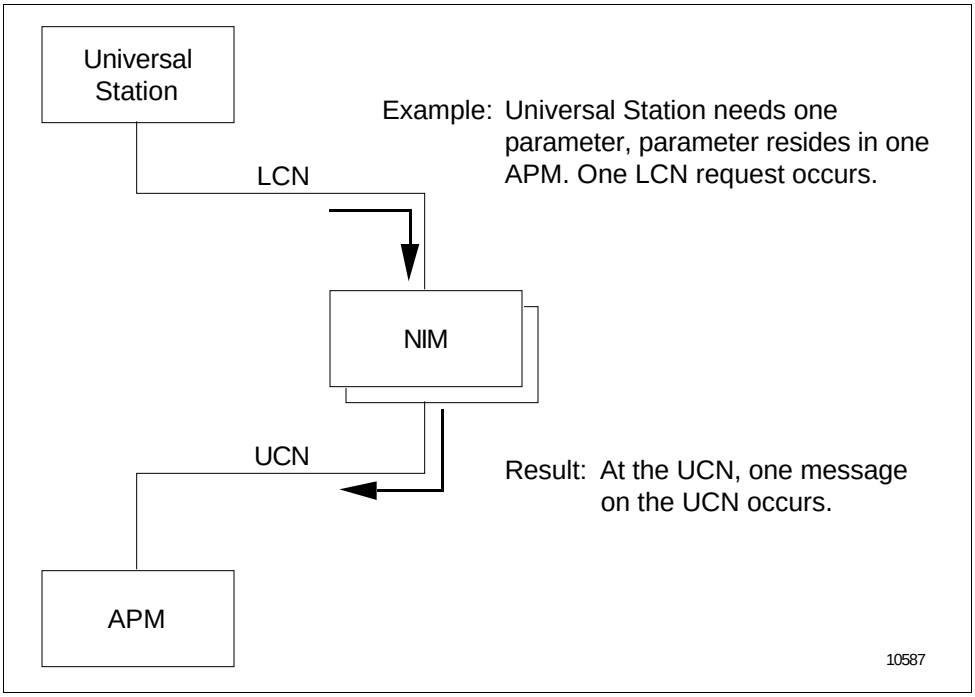
- Only requesting parameters that are needed. For example, a textual descriptor could be built into a schematic instead of collecting it from the UCN, or you can assign a collection rate of 0 (update on callup only).
- Controlling the update rate of the parameter requests. For example, a slow process may not require updates every 1 to 4 seconds. A temperature could be collected every 30 seconds.

Next, review what happens if more nodes are added to our example network. This presents another factor to consider in optimization.

Simplest case example

Returning to the example system, the simplest LCN request scenario is presented. For the sake of clarity in the later examples, assume that a US display requires only one parameter (such as a PV). As you would expect, an LCN request for one parameter generates one UCN message.

Figure 2 Example of LCN to UCN Request

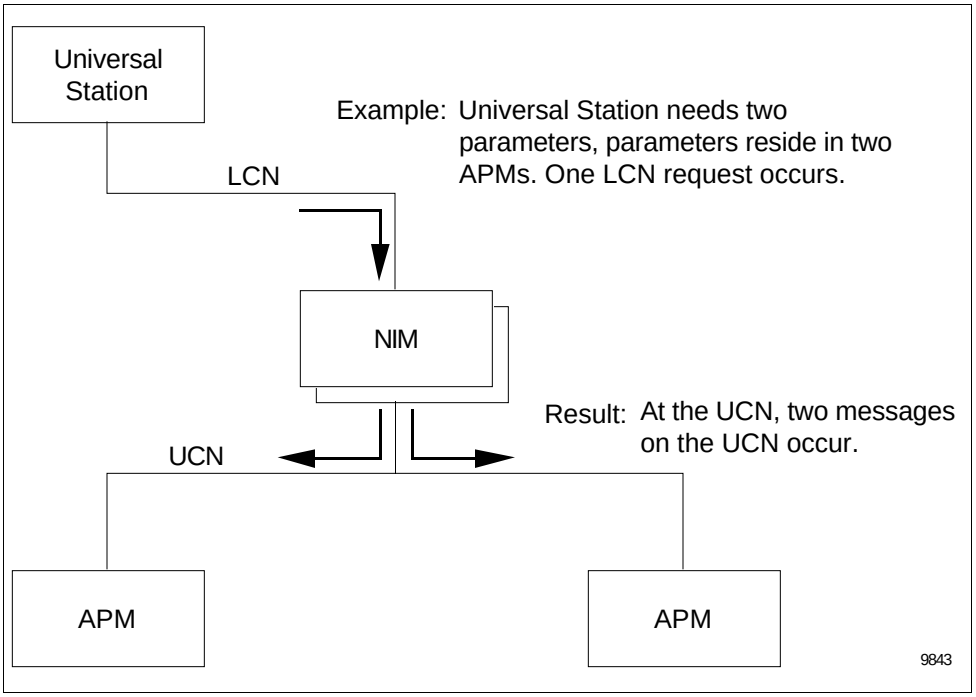


Continued on next page

Additional UCN node

Consider what happens if an LCN node needs one parameter each from two UCN nodes. One LCN request actually results in two UCN messages.

Figure 3 LCN Request to Multiple Nodes

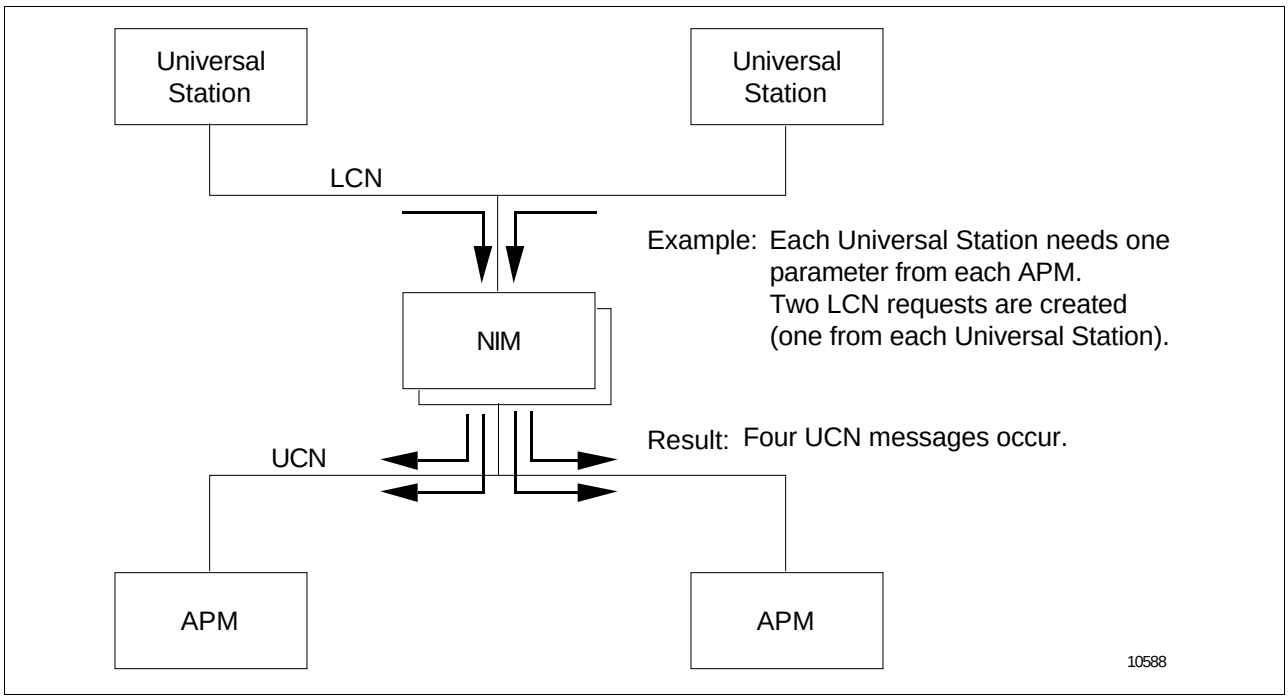


Continued on next page

Additional LCN node

Now add another LCN node. If both LCN nodes need one parameter from each UCN node, four UCN messages result as shown in Figure 4.

Figure 4 Multiple LCN Requests, Multiple UCN Nodes



NIM services LCN on demand

From our examples (Figure 5 and Figure 6), you should first note that the NIM does not “hold” the messages until there are enough for a destination node, mainly because of the scan time (how often the request is made) and network token-rotation time. In other words, the NIM services the LCN as soon as it gets a request from the LCN.

Continued on next page

Conclusion

As you can see from our examples, the amount of UCN traffic is also dependent on the destination of the LCN request. That is, the NIM sends the message to the UCN node that "owns" the data. You can draw three general conclusions from our examples in Figure 3 and Figure 4:

1. In Figure 3, LCN requests are split up into one or more UCN messages. For the engineer, this means that some optimization is possible if the LCN node requests only the data it needs. For example, one site had a point descriptor updated every 4 seconds. It is necessary to collect it only once, or better, enter the textual string into the schematic.
2. In Figure 4, LCN requests are sorted by the NIM to send it to the destination UCN node. For the engineer, this means that optimization is possible if the LCN node "groups" its requests by UCN node number. For example, history collection groups that are built by UCN node number result in more efficient requests to the UCN.
3. The update rate of requests also needs to be considered. A slow process, for example, may not need the data as often as every 4 seconds.
4. Use a parameter from as high a data owner as possible. For example, use FI3000.HIGHAL = PVHI (NIM resident instead of FI3000.PVHIFL = ON (IOP resident).

Use it or lose it

While our examples illustrate the simplest request scenarios, one further comment should be made. If you are going to open a communication path to the UCN, you should make best use of your LCN request. A request for one UCN parameter (as in our examples) is not any more efficient than a request for 50 real parameters. If you are going to request data, you should make the best use of that request. In other words, "use it or lose it."

Reducing schematic loading on UCN nodes

The following is an excerpt from the Picture Editor Reference manual, providing guidelines for reducing schematic loading on the NIM and UCN.

There are two concepts that particularly apply when optimizing the UCN data gathering efficiency of a custom schematic. One is grouping of the data access requests by processor type and the other is the use of parameters at the highest possible data owner level.

Continued on next page

Reducing schematic loading on UCN nodes (continued)

Grouping of Data Access Requests by Processor Type

Custom schematics request data from the network through LCN Data Access. The mechanism used to make the request is called an Intermediate Data Block (IDB), but in this document, it is called a Data Request message.

Each Data Request message consists of a single collection group. A completely homogeneous request is generated by a group containing parameters from

- a single UCN node, and
- a single processor type (Control or IOL) within that node

However, if this criteria is applied strictly to more than a few UCN nodes or in cases where there are multiple collection rates, it creates a large number of IDBs and places a heavy transaction load on the NIM. It also creates a bookkeeping problem for the developer.

Note: The NIM sorts by node so display builders only need to be concerned about grouping by processor type and collection rate.

The combination of user-grouping by processor type and NIM sorting by node creates a UCN request message that is resident-homogeneous (single node/single processor in the same node).

The need for homogeneous UCN Request Messages arises from the way in which
parameter requests are processed in the PM, APM, or HPM. Control parameters are (in effect) obtained directly from memory with relatively small cost in processing time in the node's Communications Processor. IOL parameters require that the Communications Processor create a second request to the IOL processor with an associated cost in processing time.

This overhead cost occurs each time a new IOL parameter is encountered in a message. In a message that consists of a mixture of control-resident and lOL-resident parameters, the Communications Processor uses a significant amount of time switching between the two types of parameters. If the message consists of IOL parameters only, the number of IOL processor requests is considerably reduced because the Communications Processor can include several parameters in a single IOL request.

Continued on next page

Reducing schematic loading on UCN nodes (continued)

You can determine the processor type from the point type. It isn't necessary to be aware of the actual residence of a given parameter. The relationship of point type to processor type is—

IOL Processor Point Types

- Digital Input
- Digital Output
- Analog Input
- Analog Output

Control Processor Point Types

- Regulatory Control
- Regulatory PV
- Digital Composite
- Logic
- Process Module

All point types have some parameters that are NlM-resident, but access to those parameters has no effect on UCN node communications loading.

The Basic Rules for Grouping

Using the update rate considerations previously discussed, set up the collection groups as follows:

- groups to be updated at target selection
- groups to be collected at multiples of the schematic base rate
- groups to be included in fast update

Split groups that will update into subgroups by processor type (control or IOL).

Summary

Now that you have a general understanding of how an LCN request is sent to the UCN go to the next section, which discusses how peer-to-peer communication occurs.

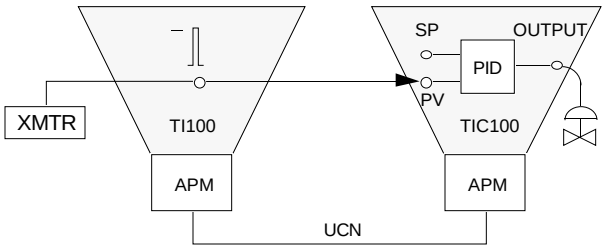
Operation Concepts

Quick review Peer-to-peer communication is accomplished through either point connections, or CL programs that use read or write statements.

Figure 5 Peer-to-Peer Overview

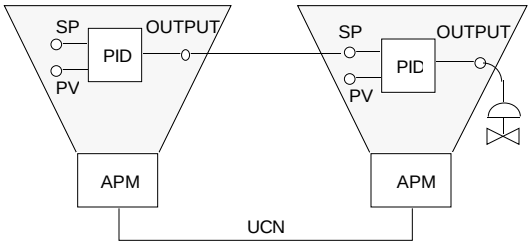
- Point Connection

  Input Connections

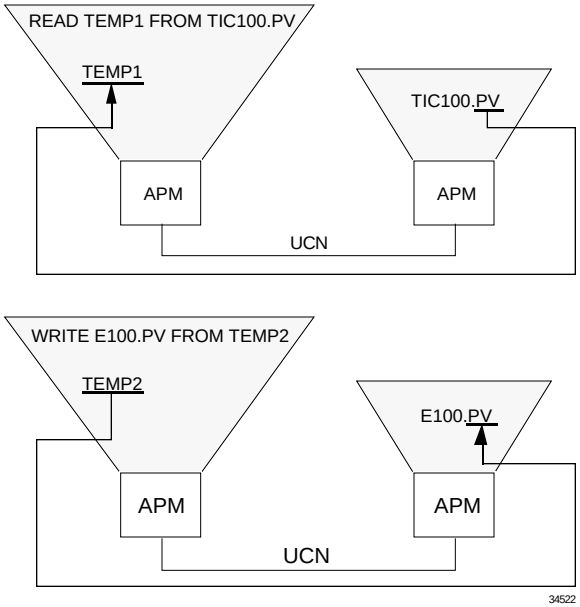


  Output Connections

PM = SP only
APM/HPM = SP and X1
(X1 for R5xx only)



- Read or Write Statements in CL



Continued on next page

Introduction

As Figure 5 illustrates, peer-to-peer point connections are either input or output connections. To understand how peer-to-peer point input connections are gathered (fetched) during peer-to-peer communication requires discussing the input scan table.

Configuration review

The configuration limits for peer-to-peer communications are summarized as follows:

- PM and APM—Up to 50 input connections (pulls)
- HPM—Up to 100 input connections (pulls)
- Unlimited output connections (pushes)
- Use of CL read/write statements that fetch or store at statement execution time.

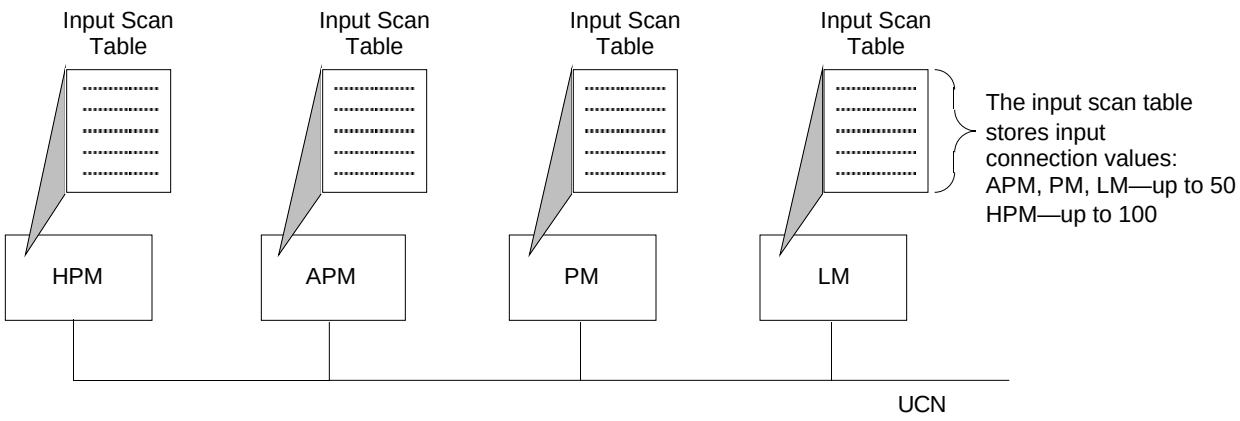
User preferences

Typically, batch application users prefer to use CL read/write statements for peer-to-peer communication. Continuous application users typically prefer using point connections for peer-to-peer communication.

Concept

As Figure 6 shows, each UCN node has an input scan table for its configured peer input connections.

Figure 6 Input Scan Table



Continued on next page

How the table is built

When a UCN point that has peer-to-peer connections is loaded to the node:

- the node checks and identifies when an input point connection is not part of the local node,
- the node places the point connection in its peer to peer input scan (prefetch) table (The table also has the point.parameter node number.),
- every 1/2 second the point.parameter is scanned (fetched), even if the requesting point is inactive.

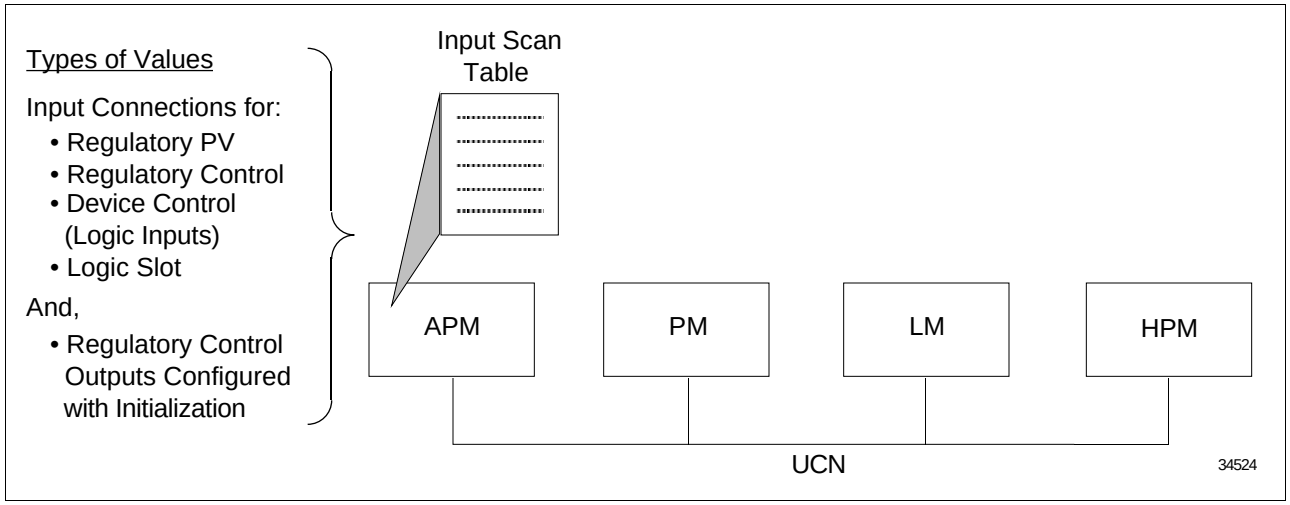
CL reads

CL reads are another type of peer input connection, but they are not handled by the input scan table. CL reads are handled at the time they are executed.

Values

Figure 7 shows the value types (discussed later) that are scanned.

Figure 7 Scan Table Values



Control outputs

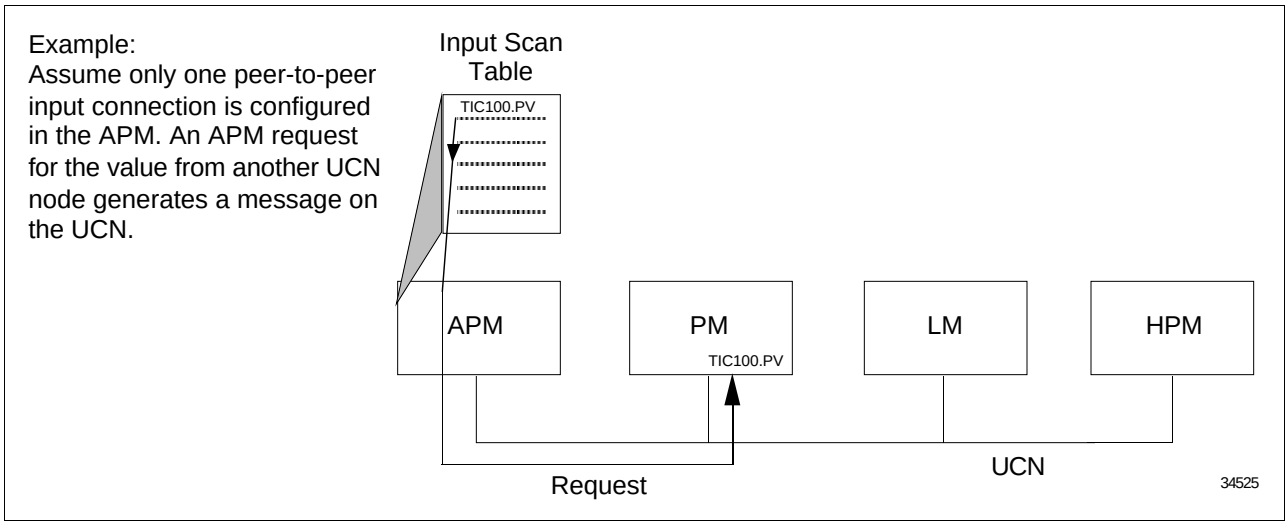
It may seem odd to find regulatory control outputs added as an input scan table entry. Note that this applies only to regulatory control outputs configured with initialization. This is because a regulatory control output would need to "know" when the downstream point is initializing it. In order to do this, the initialization request and initialization value are added to the input scan table.

Continued on next page

Input request

The input scan table is updated through a UCN node request. Again our example is simplified and shows only one input connection. The APM's request to the PM node that "owns" the point is also called a "message sent."

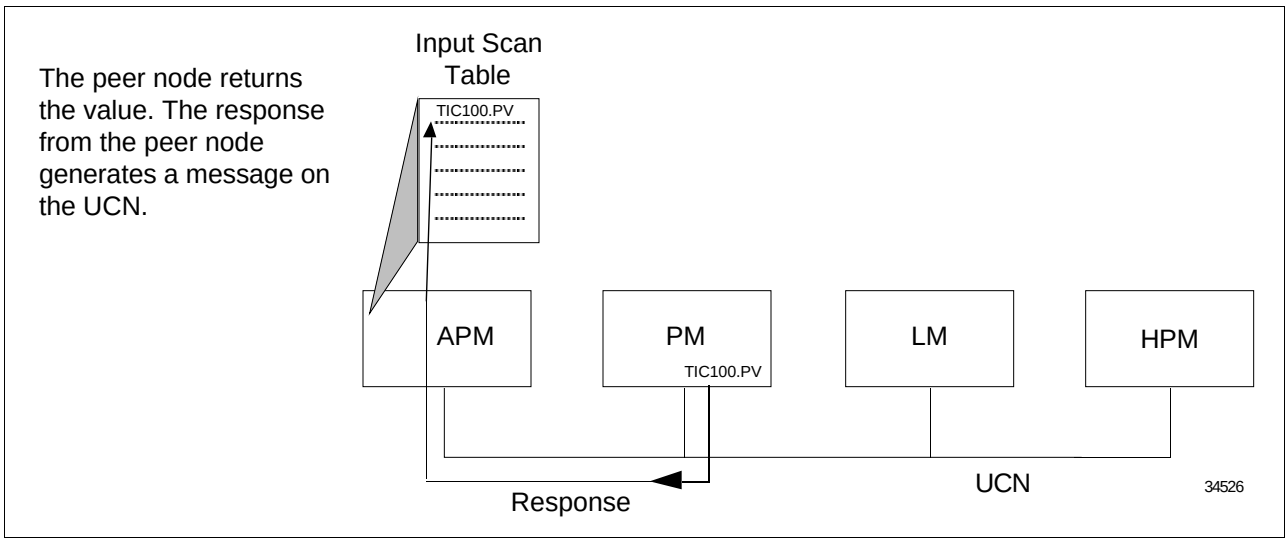
Figure 8 Input Request



Response

The peer node's response is to return the value. The APM's response from the PM node is also called a "message received." This completes one "transaction" on the UCN.

Figure 9 Response from Node



Continued on next page

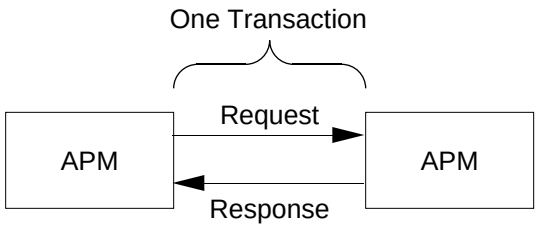
Key terms

The previous discussion, while easy to understand, introduces the terms

- Transaction
- Message sent
- Message received

The terms message sent and message received appear in various diagnostic displays and are used to determine the UCN load (number of transactions). Figure 10 summarizes these key terms.

Figure 10 Transaction Definition



Concept:
A transaction consists of a request and a response; or, in other words, a transaction consists of a message sent and a message received.

9847

Destination node contributes load

You may have raised the question, “Does the PM in Figure 8 and Figure 9 have a UCN transaction as well?” That is correct, it received a request for data (message received) and returned the value (message sent). From this observation you can say that

- A peer-to-peer transaction places a load on both the sending and receiving node. Later in calculating peer-to-peer load, you will see that you have to consider the load a UCN node places on other nodes, as well as the load placed on it by its peers.
- Messages sent and messages received can be part of your UCN load calculations.

Other transactions

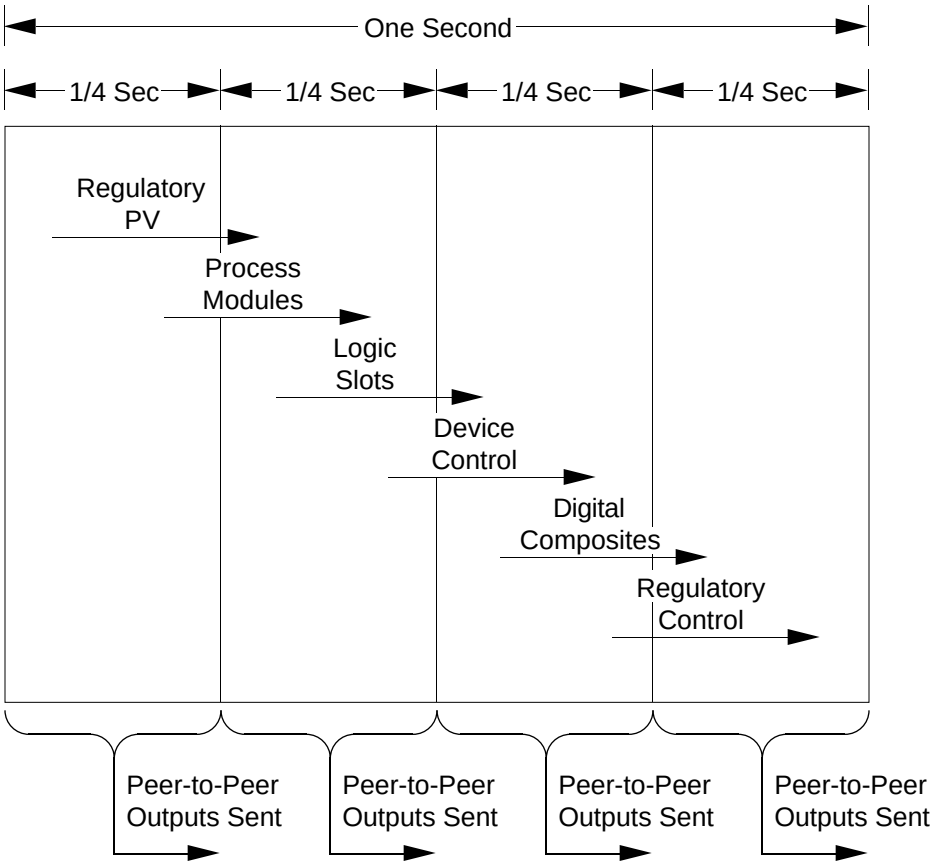
As you know, output connections and CL read/write statements contribute to the peer-to-peer transaction load. The next section shows how output connections are handled.

Continued on next page

Outputs sent at point execution

As Figure 11 illustrates, peer-to-peer outputs are generated when the point is executed. The point's execution is scheduled over the 1-second processing scan. Later in this course module, you will see how engineers often use the schedule to calculate peer-to-peer load. For now, note that when a point is executed at its scheduled interval, the peer-to-peer output is generated (message sent.)

Figure 11 Output Cycle Example for 1 Second Scan Configuration



Concept: Peer-to-peer outputs are generated (message sent) at the time the point is executed.

9848

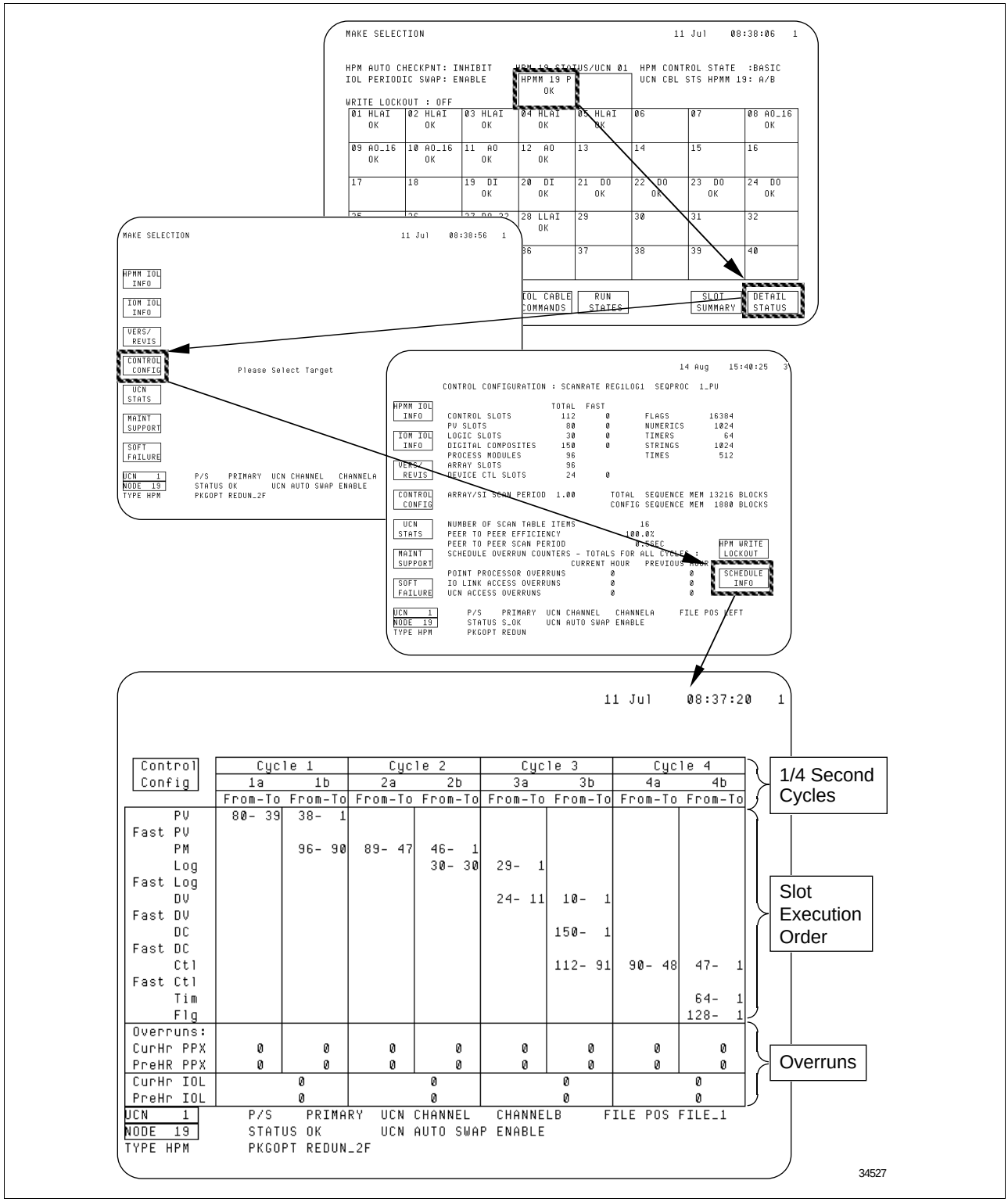
Example display

Figure 12 shows an example schedule. Note the slot execution order on their respective cycles. As you would expect, each site's schedule will vary, as the node automatically determines the optimum load for each cycle. Engineers use this display to calculate the peer-to-peer transaction load, as shown later in this course module.

Continued on next page

Example display An example schedule (and how to call it up) appears in Figure 12.

Figure 12 Load Schedule Example



Continued on next page

CL read/write

Calculations for peer-to-peer load also include CL read/writes. Again, the schedule is sometimes used to identify when the process module point is executed. Calculating peer load from CL could become complex, as you would need the program listing to determine when a CL read or write statement is executed. (Note: An exception would be where CL is used in a continuous fashion for tight loop control—in that case the CL load is fairly easy to calculate.)

Summary

In this section you have seen how peer-to-peer occurs. In the next sections you will see how to determine if a communication problem has occurred, and a way to resolve the problem.

Communication Problem Identification

Symptoms

Introduction

When a UCN communication problem occurs and peer-to-peer communication is configured, a common tendency is to believe the problem is caused by a peer-to-peer fault. Actually, symptoms of communication problems can indicate a problem with

- UCN network itself, for example, a bad hardware connection;
- LCN loading, for example, requests from a busy schematic; or
- UCN loading, for example, excessive peer-to-peer loading.

Symptoms of problem

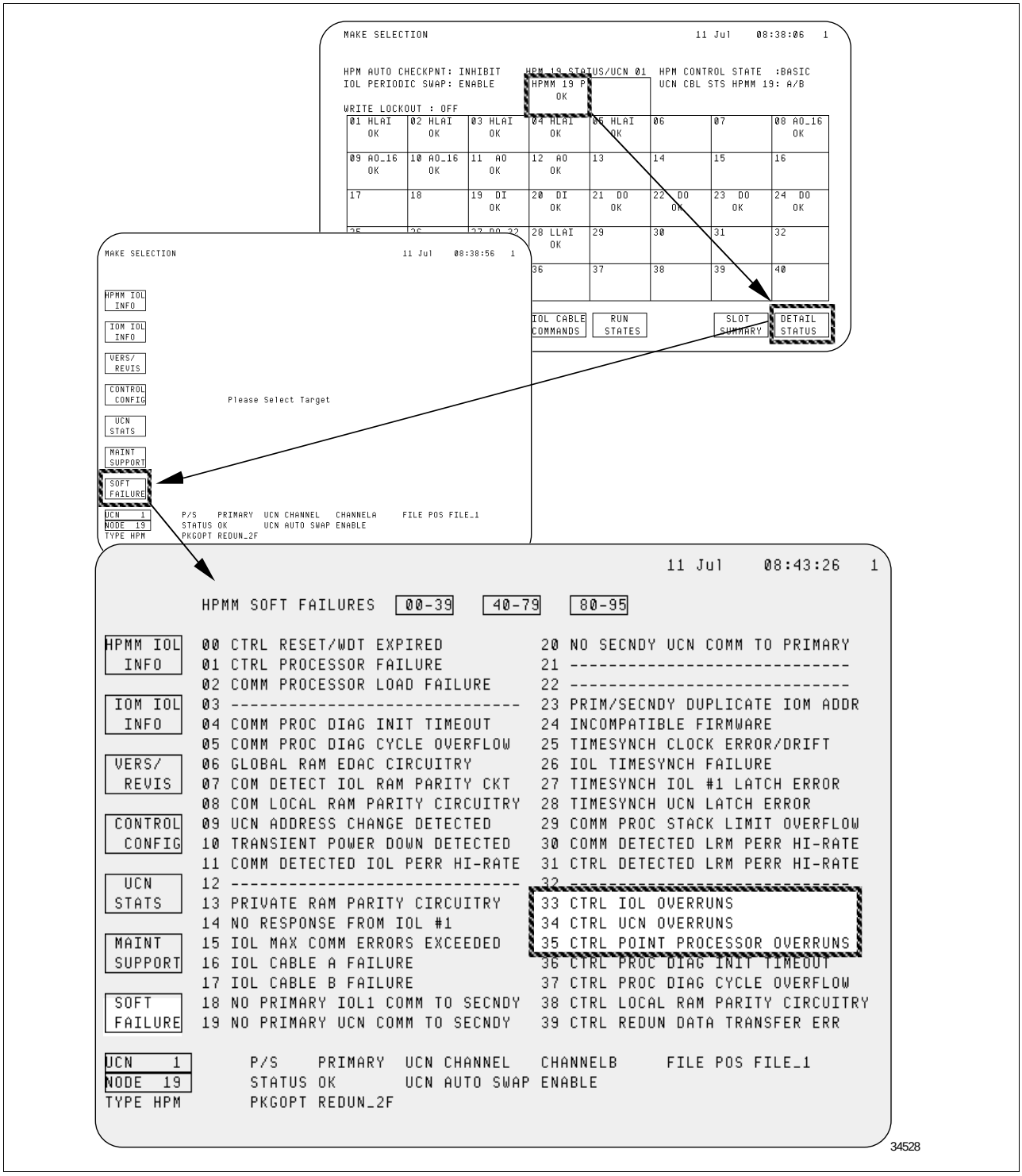
Communication problems and excessive UCN traffic symptoms can appear in several ways:

- Question marks or blanks on operator displays (operators experience temporary loss of view caused by timeout occurring while waiting for UCN node to update the graphic update requests).
- Universal Station receives frequent system alarms (UCN overrun soft failures, see Figure 13).
- Bad PV alarms on points that get PVs from points in another node (and the other node actually has a good PV).
- Overruns on Application Module points.
- The UCN node experiencing an excessive load problem causes a load on other UCN nodes because their messages cannot be serviced.
- On systems operating on R410 and later, indications are that peer-to-peer efficiency is reduced and CPUFREE values are low.

Continued on next page

Overrun indications UCN overrun failures are annunciated as soft failures. Figure 13 shows where the soft failure overrun indications are annunciated.

Figure 13 Overrun Symptoms Locations

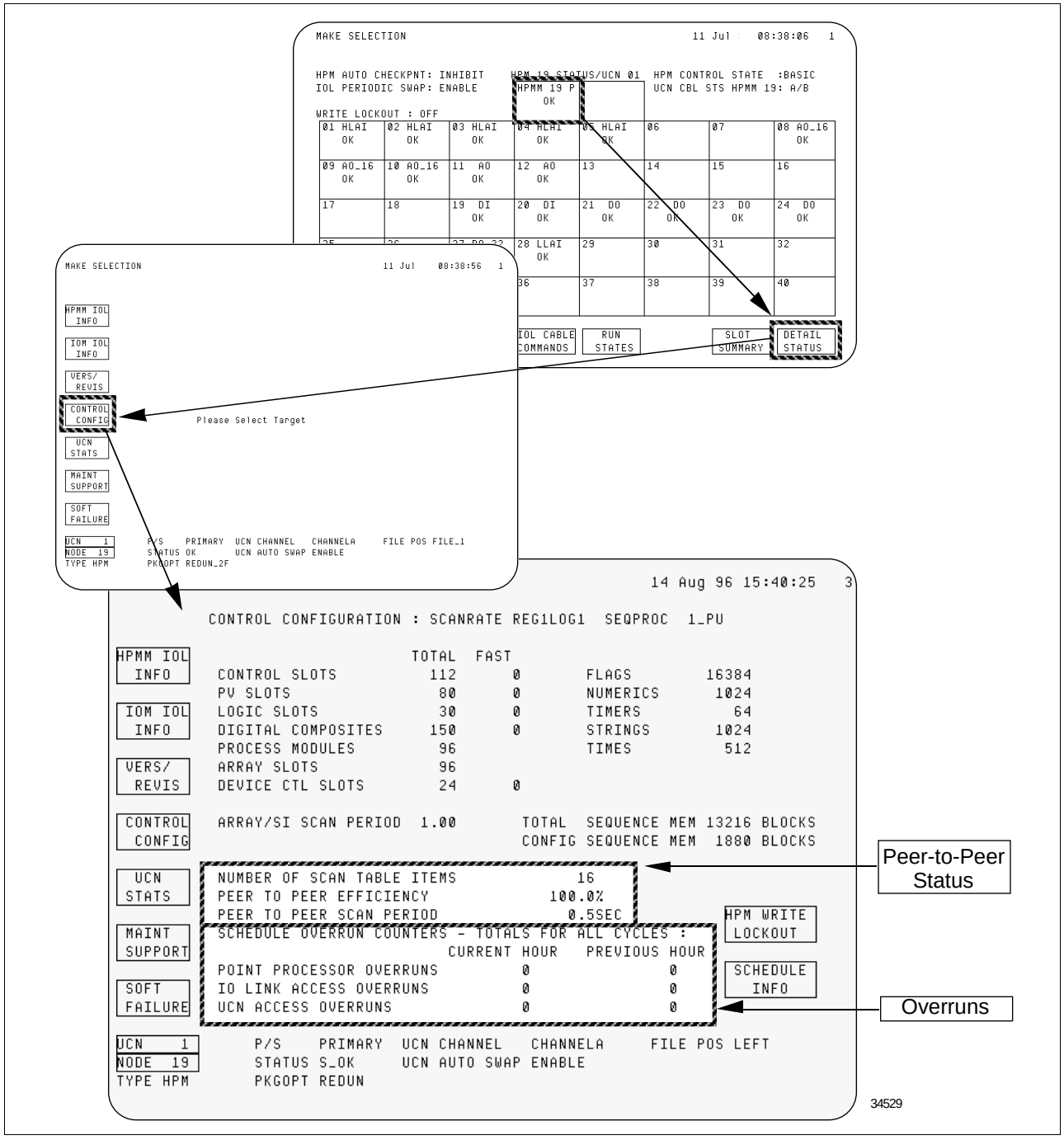


Continued on next page

Hourly counters and peer to peer status

The overrun counters for the previous and current hours is shown in Figure 14. The hourly overrun counters are used to determine whether an unacceptable rate of overruns is occurring. On R410 and later, a value for peer-to-peer efficiency, configurable scan period, and number of input scan table items is also provided.

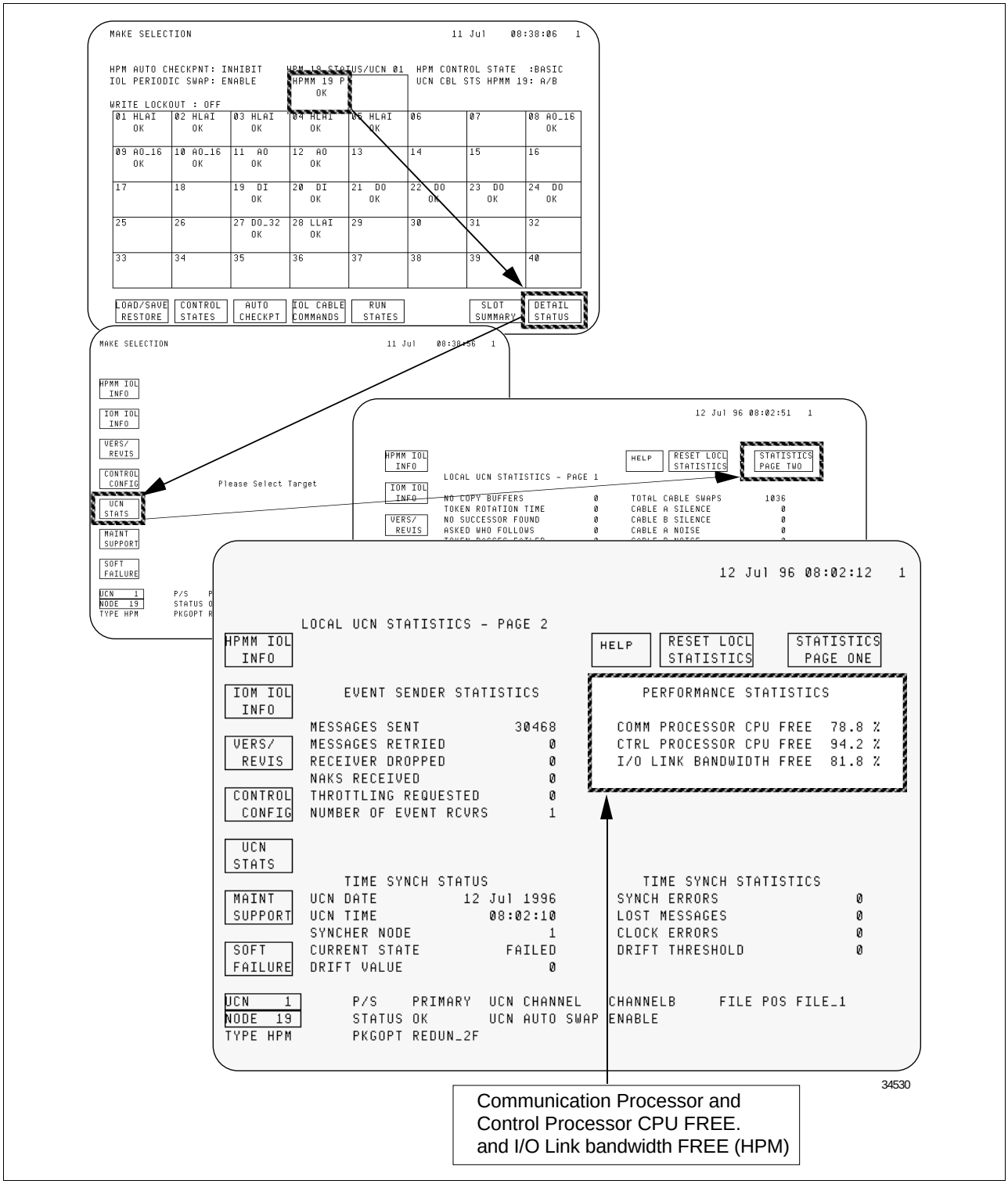
Figure 14 Hourly Overrun Counts and Peer-to-Peer Efficiency



Continued on next page

CPUFREE displays R410 and later provides CPUFREE statistics on page 2 of the Local Statistics display, as shown in Figure 15.

Figure 15 CPUFREE Statistics



Continued on next page

Peer-to-peer statuses

The peer-to-peer status information shown in Figure 14 and Figure 15 is described as follows:

- number of scan table items—the number of peer-to-peer fetches (pulls) configured in the input scan table.
- peer-to-peer efficiency—a percentage derived from the number of successful peer-to-peer accesses (no overruns) over a period of 15 minutes.
- peer-to-peer scan period—how often the input scan table values are fetched. Configurable as 0.5 second or 1 second.

Summary

Communication problems can be caused by one or more factors. The symptoms can manifest themselves in several ways. While the problem may indicate that an excessive peer-to-peer communication load has been configured, an investigative procedure usually starts with the UCN network cables and connections.

Check UCN Health

Introduction

Your investigative procedure should consider other factors before reviewing the peer-to-peer load. A broad outline of your procedure would include:

- Investigating UCN health.
- Investigating LCN load
- Investigating UCN load
- Reducing the load and recovering

UCN problem indications

The health of the communication system and UCN statistics should be studied first, as communication and cable problems may be the cause of the above symptoms. If cables are marked as bad or UCN statistics (Figure 16) such as

Ⓐ Total Cable Swaps and Auto Reconnects,

Ⓑ Noise bits,

Ⓒ Checksum Error,

Ⓓ Partial Frame,

Ⓔ Cable Noise,

Ⓕ Cable Silence, or

Ⓖ No Response errors are increasing steadily, fix UCN problems first.

No response errors exceptions

There are several scenarios that cause the No Response error count (refer to Ⓖ in Figure 16) to steadily increase.

- Secondary PMMs in redundant backplanes that are powered down will cause the No Response error count to increment.
- Any nonredundant APM will generate No Response errors because redundancy hardware is included on the Control card, even for APMs in a nonredundant configuration.
- Nonredundant PMs with redundancy hardware options present will increment the No Response error count.

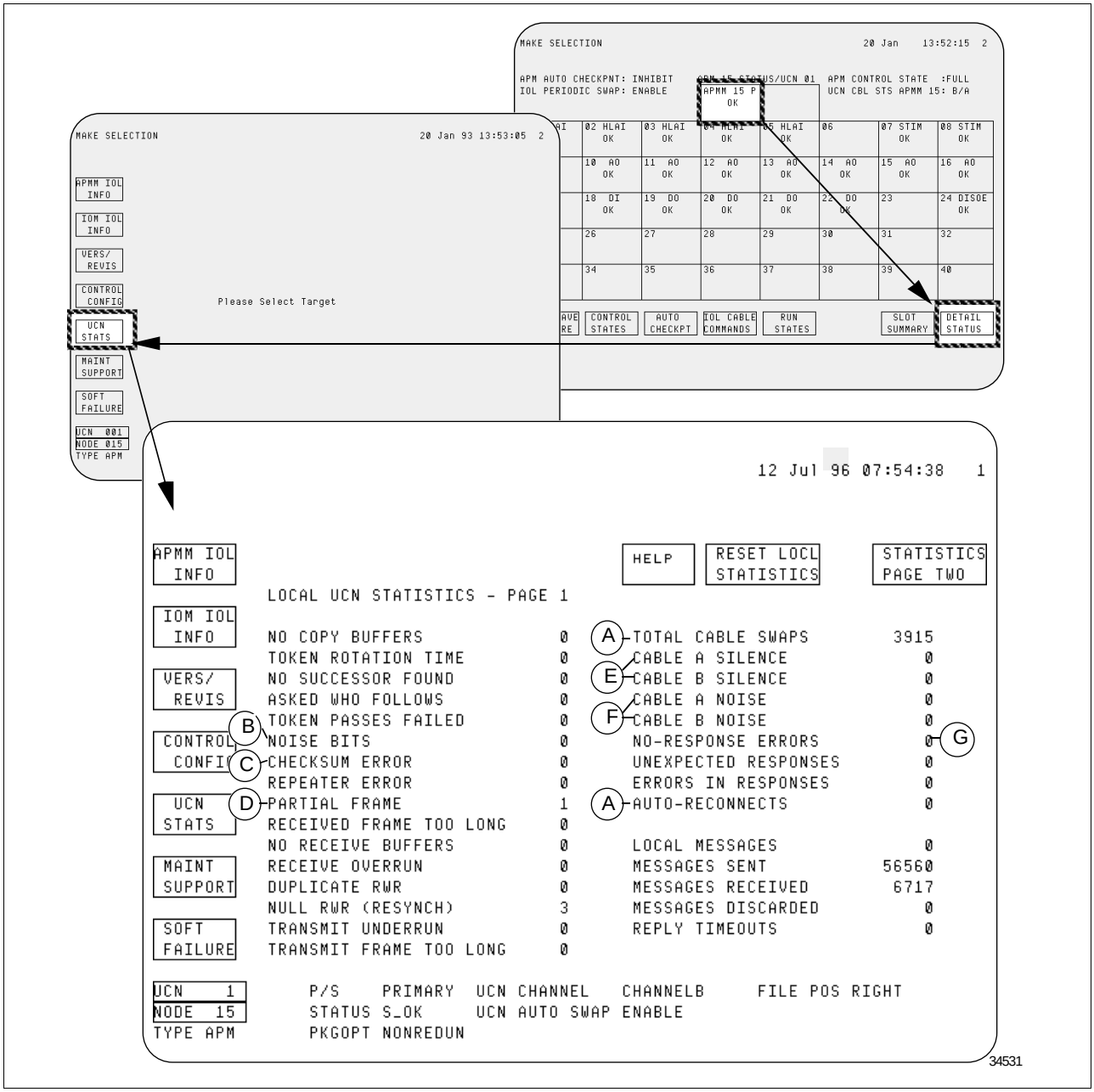
If any of these scenarios exist on your system, you will see the No Response error statistic continually incrementing in the primary (approximately eight counts per second). Keep this in mind when trying to interpret UCN statistics on your UCN.

Continued on next page

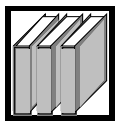
Example display

Figure 16 is an example display showing where the UCN health indications are located. A good practice is to first select the target RESET LOCAL STATISTICS , take a display print for a beginning reference point, at a fixed time interval take another display print for an ending reference point, and then calculate the statistics for acceptable or unacceptable error rates.

Figure 16 UCN Health Indications



Continued on next page



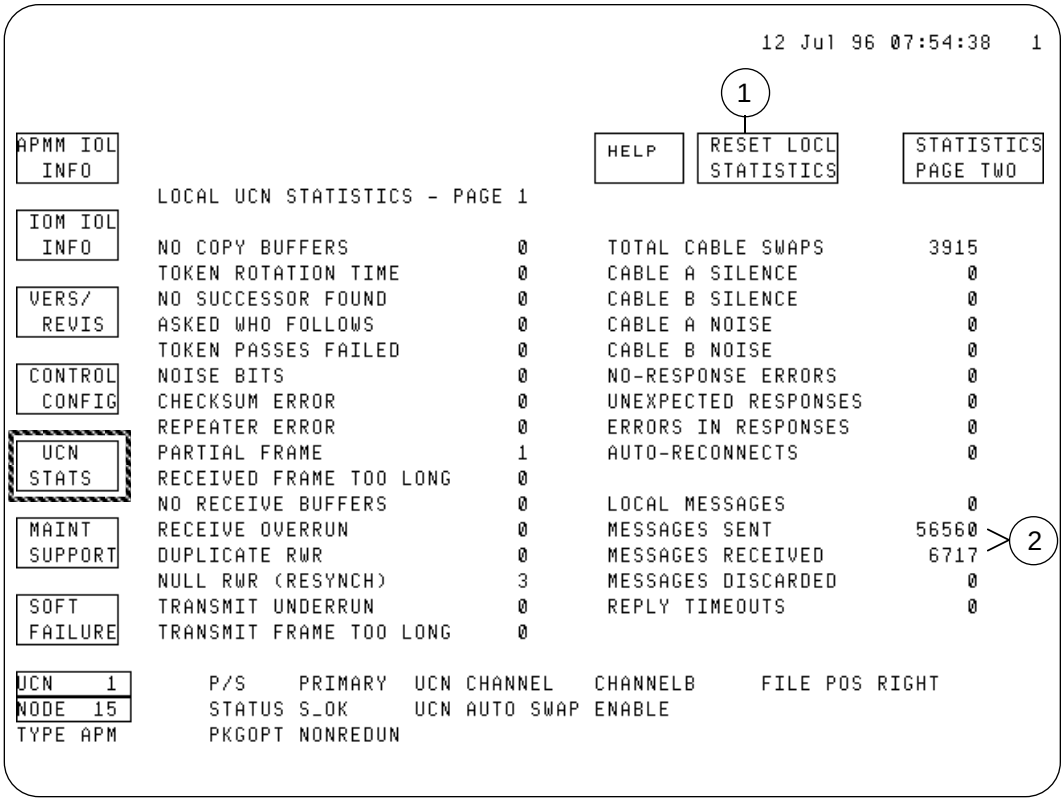
REFERENCE—Table 4-5 in the publication *Universal Control Network Guidelines* provides more detail on the acceptable error rates for various statistics.

Check LCN Load

NIM UCN statistics

After determining whether the UCN is healthy, your next step is to check the LCN load. One starting point is to check the number of transactions (messages sent, messages received) a UCN node is experiencing. (Note that the transaction load may be LCN-related, so you also need to monitor what the LCN is contributing to the load.) Again, a good practice is to first select the target RESET LOCAL STATISTICS , take display prints, and then monitor the statistics.

Figure 17 Collect Message Sent and Messages Received



(1) Reset local statistics.

(2) Track messages sent, messages received, to calculate transactions.

34532

Values to monitor

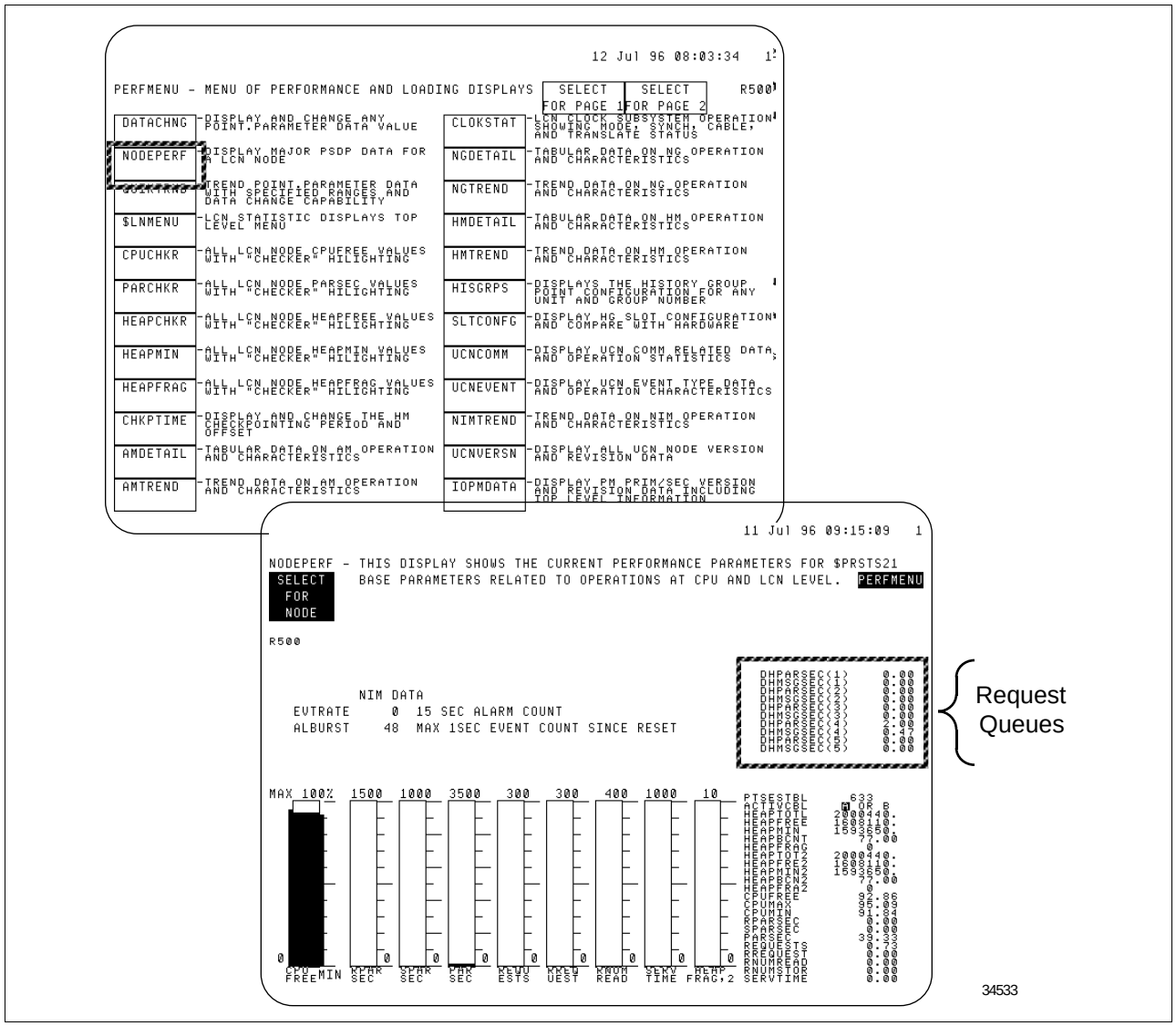
NIM values (Figure 18) to monitor include CPUFREE, CPUMIN, PARSEC, HEAPFREE, DHPARSEC(i), DHMSGSEC(i). How to interpret the values for DHPARSEC(i), DHMSGSEC(i) is discussed further in the course material, Interpret NIM Loading.

Continued on next page

NIM LCN statistics

While the NIM has the capability of handling up to 1200 parameters per second from the LCN, Toolkit displays show the parameter per second requests as well as message request load (DHMSGSEC(i)). This represents the load from the “LCN side of the system.”

Figure 18 NIM Parameter Request Display



Summary

Assuming that the UCN is healthy and the LCN loading is not a factor, your final step is to check the UCN load. This requires an understanding of UCN node performance constraints, the topic of the next section.

Transaction Load

Introduction

Having background on the performance constraints of a UCN node helps you properly configure your UCN node, as well as troubleshoot any UCN loading problems.

Definition

UCN nodes have a performance constraint (on the UCN) estimated at

PM/APM/LM/SM	HPM
400 parameters per second	1000 parameters per second
50 transactions per second	100 transactions per second

Experienced users state that the transaction load limit (or messages limit) is usually exceeded before the parameter per second limit. The transaction load is calculated as follows:

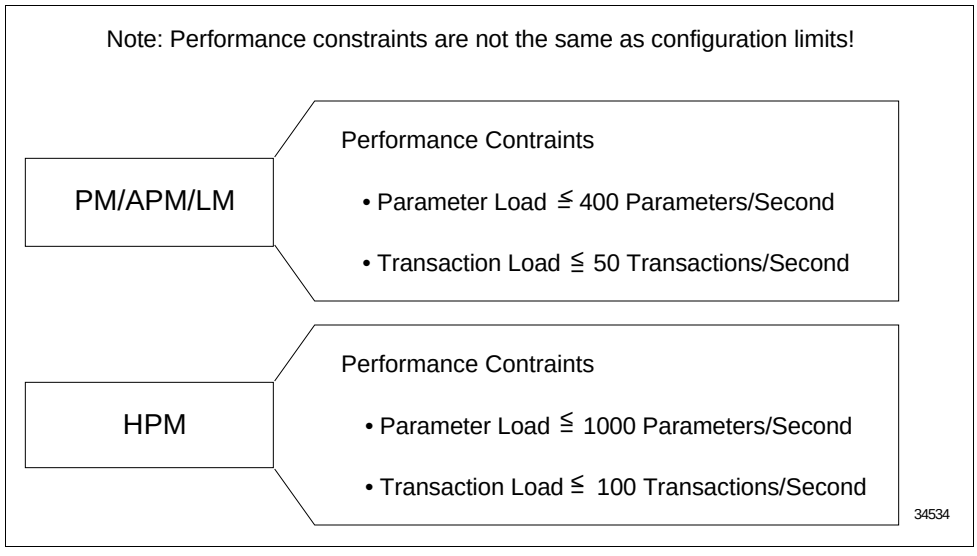
- Number of Transactions = (messages sent + messages received)/2

The Toolkit displays for R410 and later provide this information in formats such as transactions, messages sent, or messages received.

Performance versus configuration

Performance constraints are not the same as configuration limits! Don't get the configuration limits (50 input connections, unlimited output connections) confused with transactions. One transaction can contain a large number of real parameters (over 50 reals, or even a higher number of parameters in a transaction if logical parameters are used.)

Figure 19 Performance Constraints Definition



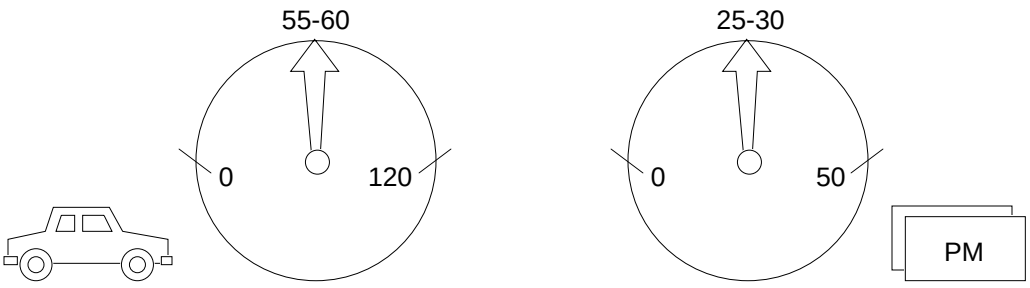
Continued on next page

Transaction Load, Continued

Performance analogy

An analogy of performance constraints is shown in Figure 20.

Figure 20 Performance Analogy



Just as an automobile has a performance range of 0 to 120 miles per hour but operates most efficiently at 55-60 miles per hour, a PM has a performance range of up to 50 transactions per second but, typically, users aim for 25-30 transactions per second.

9860

Burst analogy

Continuing with the automobile analogy, you could say your car should handle a “burst” of up to 120 mph, but that does not mean you would run at that speed constantly. Likewise, a UCN node can handle bursts of 50 or 100 transactions per second, but you would not want to run that way continuously. This is because you need to allow for UCN requests such as diagnostics and event handling.

Summary

To avoid overruns, the UCN node performance constraints should not be exceeded. Typically, users strive for a transaction load of 25 (if the limit is 50) or 50 (if the limit is 100). This is to allow for situations that may temporarily increase the transaction load. The transaction limits apply to all UCN nodes—HPM, APM, PM, LM, and SM.

Check UCN Load

Introduction

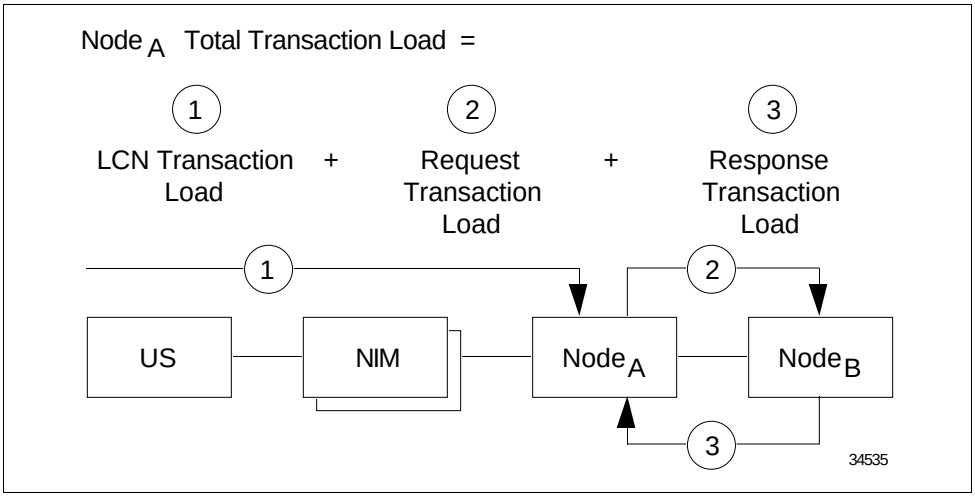
After checking the UCN health and LCN load, the UCN load can be investigated. To do that, a good starting point is to understand what the elements of a transaction load are.

Definition

Figure 21 illustrates that the total transaction load for a UCN node is made up of three elements:

1. The LCN transaction load, such as schematic parameter requests.
2. The request transaction load, or the requests "this" node makes to "other" nodes on the UCN.
3. The response transaction load, or the requests made on "this" node from "other" nodes on the UCN.

Figure 21 Transaction Load Definition

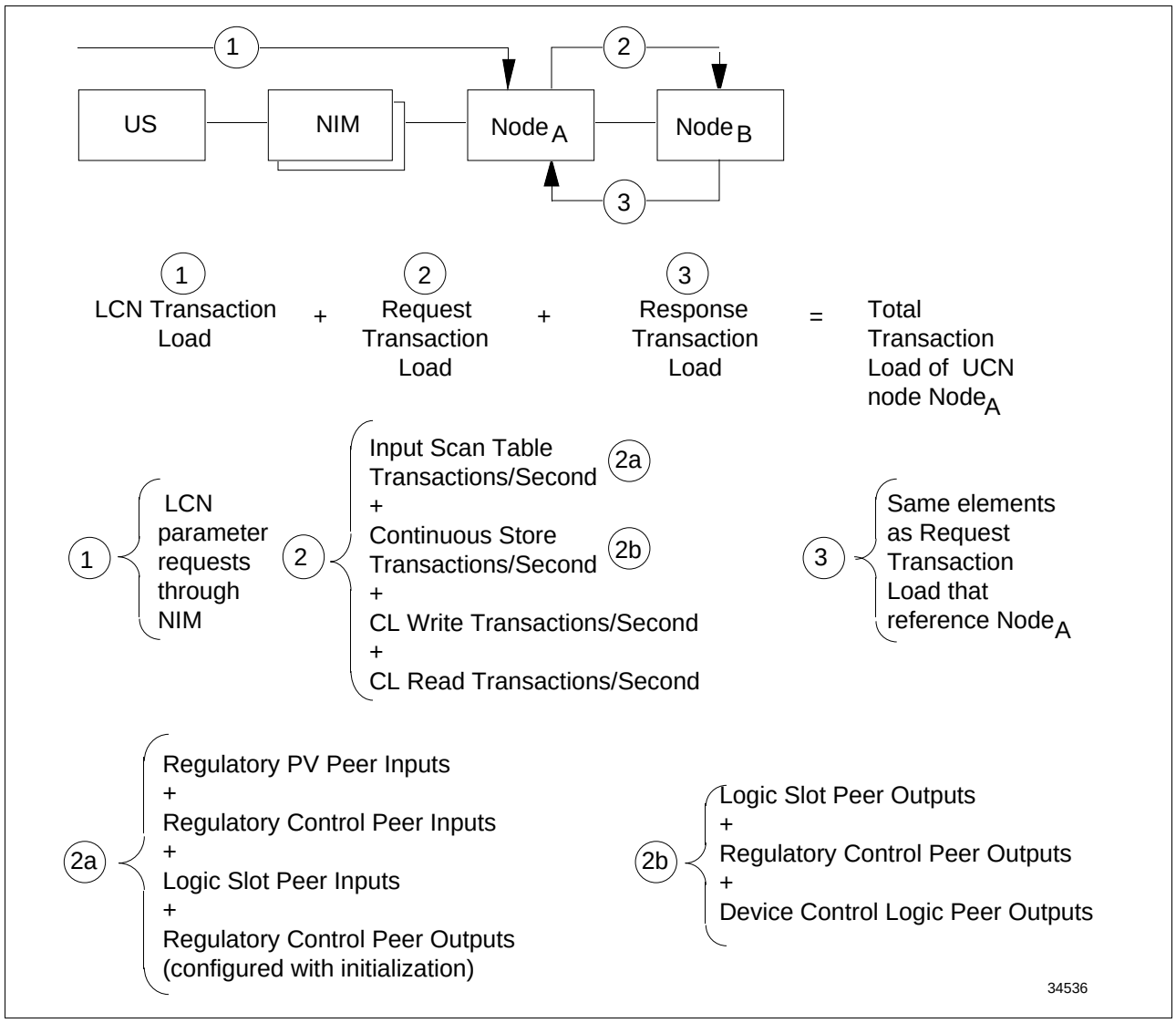


Continued on next page

Overview

Figure 22 provides an overview of the elements that make up the transaction load of a UCN node, which are discussed in more detail in the following section.

Figure 22 Transaction Load Roadmap



Continued on next page

LCN transaction load

The LCN transaction load has parameter requests from the LCN as its source. These include

- Application Module—for example, supervisory control requests
- Computing Module—for example, Data Definition Table (DDT) requests
- History Module—for example,
  - auto checkpointing
  - history collection groups
- Universal Station—for example,
  - Group displays
  - Alarm summary
  - Alarm annunciators
  - Schematics
  - Unit Trend
  - Sequence displays
  - Reports, logs, journals
- User initiated requests—for example,
  - Documentation Tool node queries
  - Point building, reconstituting
- Archive Replay Module—for example, fast point value history collection
- Network Gateway requests

Searches and queries

Find Names and Documentation Tool **file** searches are done at the LCN level. No UCN access occurs; however, when possible, a good practice is to copy the file to local media then do the search to reduce the LCN load. This also avoids the possibility of accidentally corrupting your master copy.

Documentation Tool can also be used to perform node queries. These queries are requests to the UCN. You should carefully restrict the use of "wild card" searches when making node queries if your system is already heavily loaded.

Common sense applies

If it is necessary to reduce the load from the LCN, the optimization rules listed earlier apply. They are again briefly summarized below:

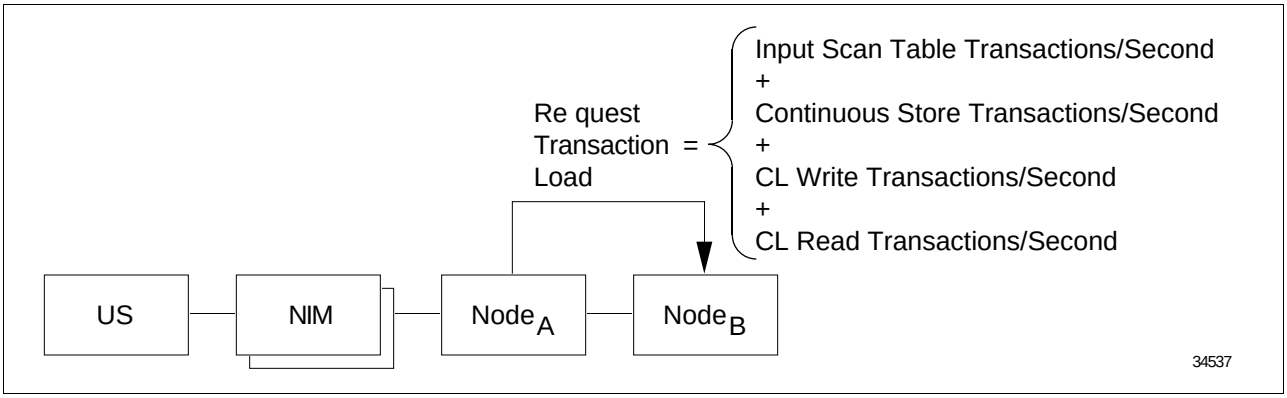
- Request only what you need.
- Group your requests by destination node.
- Schedule your requests at a realistic update rate.

Continued on next page

Request transaction load

The request transaction load consists of the four elements shown in Figure 23. You would need to know your system's configuration to calculate it. Later releases provide tools (such as Find Names) to identify your system connections. To calculate the request transaction load you need the four elements shown in Figure 23, starting with input scan table entries.

Figure 23 Calculation of Request Transaction Load



Input scan table entries

Figure 24 and Figure 25 provide additional detail about calculating the transaction load from the input scan table. Two considerations are

1. Point.parameter input connections that are repeated (used on more than one point) count as only one input connection in the scan table.
2. Regulatory control outputs (configured with initialization) appear in the input scan table once. This is because of the need to "initialize" the regulatory control slot in the event the output store cannot take place. This means the INITVAL and INITREQ are stored in the input scan table.3. Requests from the same destination nodes are grouped together in one message.

Continued on next page

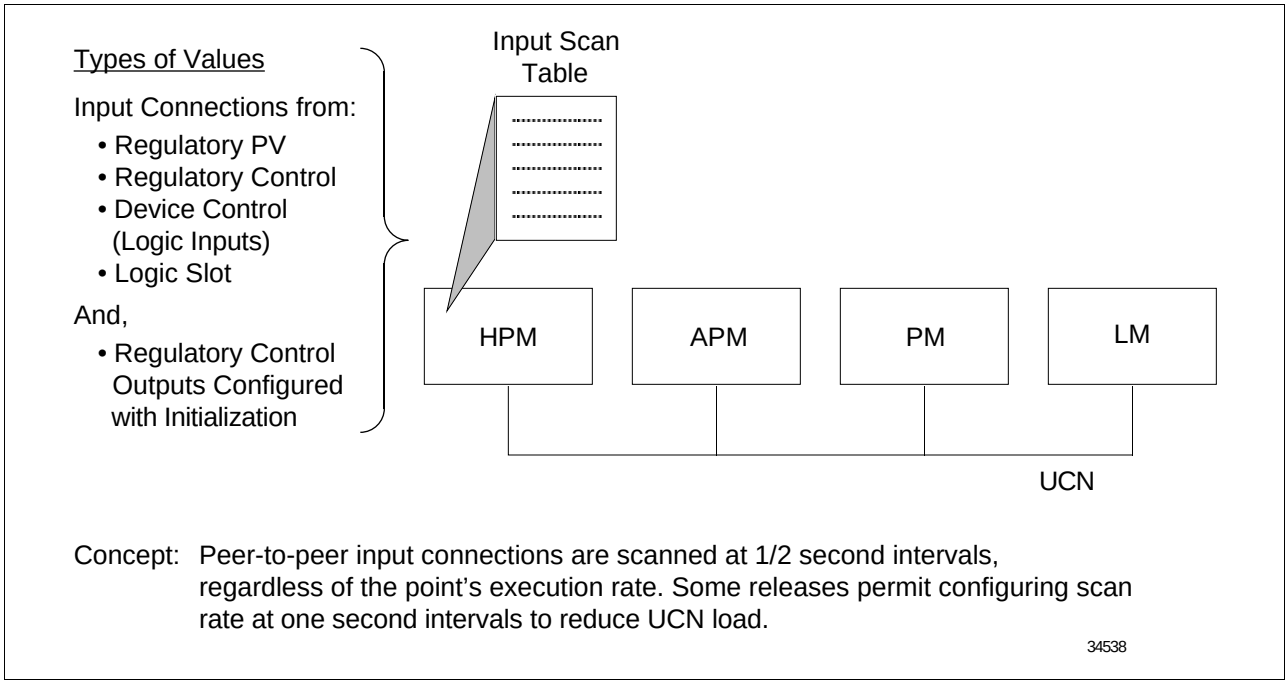
Scan fixed

The input scan table is scanned every half second, regardless of the point's execution schedule. In R410 and later releases, the scan table can be configured instead for 1 second by changing the box ($NMxxByy) parameter UCNSCANT. This can be done from the Control Configuration diagnostic display by selecting the value displayed next to "PEER-TO-PEER SCAN PERIOD." The only values accepted for input are .5 and 1.

Be aware that changing the scan interval can alter the characteristics of the existing control strategy. Analyze the effect before making any changes.

The scan period can be changed at any time; however, unless you are certain that the change will not disrupt control dynamics, it is preferable to make the change while the UCN node is in the IDLE state.

Figure 24 Input Scan Interval



ATTENTION

ATTENTION—The input scan tables fetches values whether the point is active or inactive.

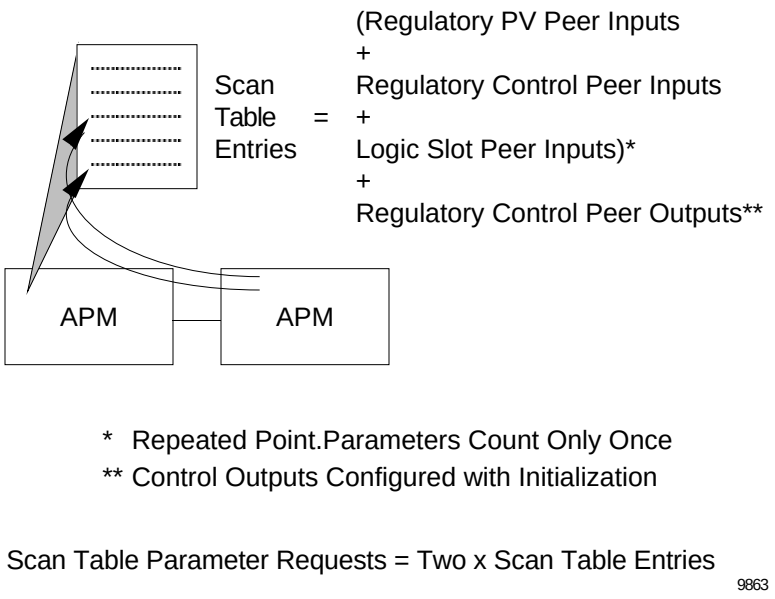
Continued on next page

Calculation

The transaction load from the input scan table is calculated as follows:

- Transaction load = number of scan table requests per node x 2. (Note: Multiply by 2 if the scan table is at half-second scan, multiply by 1 if scan is at 1 second.)

Figure 25 Scan Table Entries



ATTENTION

ATTENTION—Input scan table requests to the same node become part of one message.

Where we are

Recall that we are calculating the request transaction load (Figure 23), which is made up of

- Input scan table entries (as shown in Figure 25)

plus the topics covered next

- Continuous stores
- CL write transactions
- CL read transactions

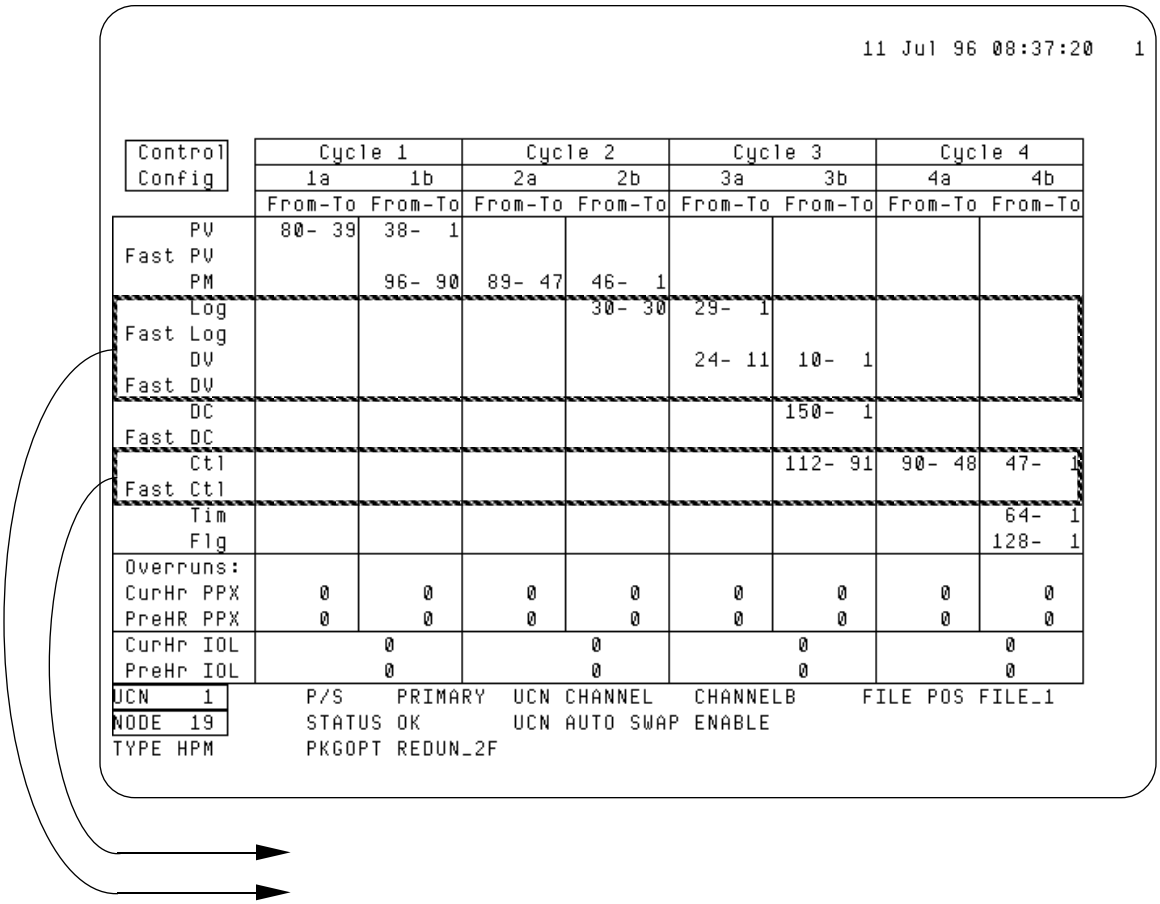
Continued on next page

Continuous stores

Figure 26 provides an example display of the APM's point execution schedule. Continuous stores occur on peer-to-peer output connections from

- Logic points (displayed as Log and Fast Log)
- Regulatory control points (Ctl and Fast Ctl)
- Device control point logic outputs (DV and Fast DV)

Figure 26 Continuous Stores



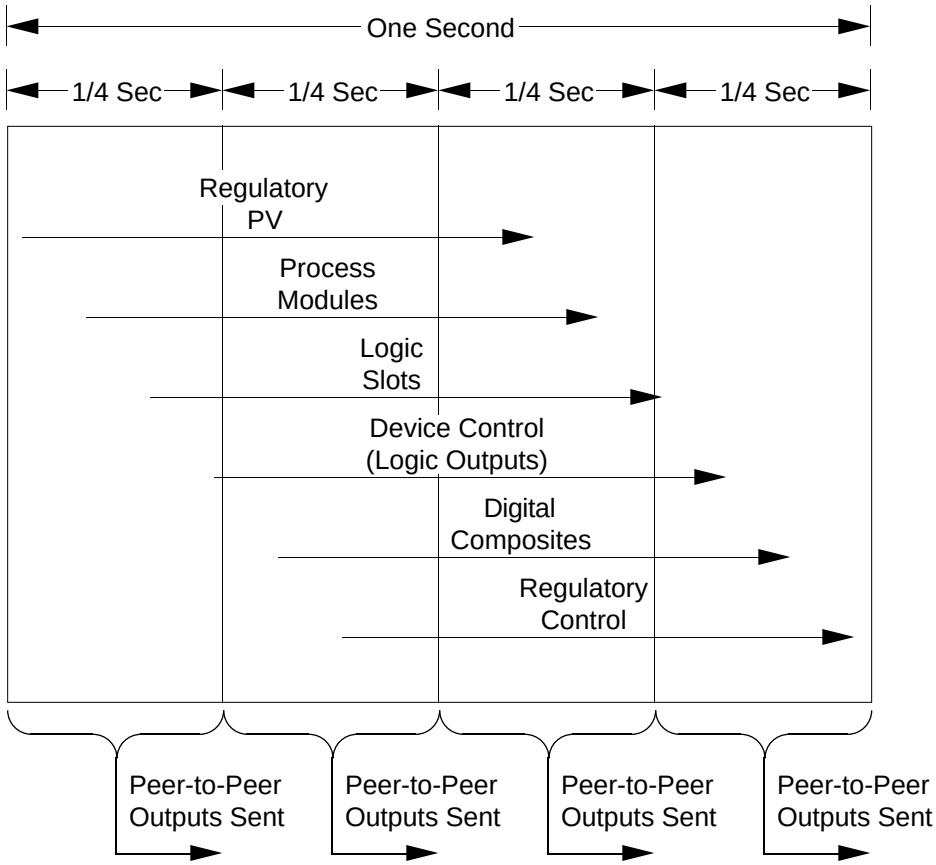
ATTENTION

ATTENTION—Later in this course and course module, this page is discussed in more detail. For now notice that this page is important to note in which cycle an output connection is stored in calculating your peer-to-peer load. For example, output stores to the same node become part of one message.

Continued on next page

Same destination Continuous stores occur at the interval that the point is executed. If the scheduled interval contains point connections with the same node destination, the output connections are sent in one message.

Figure 27 Outputs to Same Destination



Concept: Peer-to-peer output connections to the same UCN node are grouped together in a single message if they are sent on the same cycle.

10589

Continued on next page

Where we are

In calculating the request transaction load (Figure 23), you saw how to calculate the transaction load from

- Input scan table entries (Figure 25),
- Continuous stores (Figure 26),

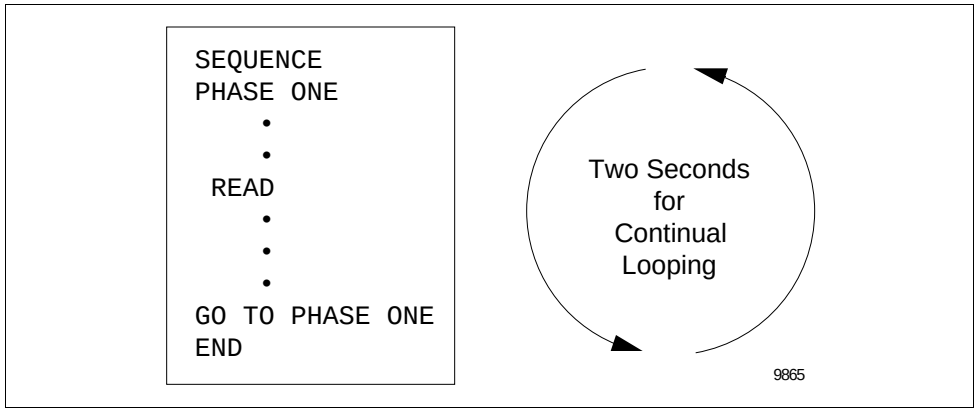
and now need to consider the

- CL write transactions,
- CL read transactions.

CL load

Figure 28 illustrates the worst case CL load, a program that continuously cycles on a READ (or WRITE) statement. The statement is executed every 2 seconds because of preemptions.

Figure 28 CL Read Write Load



Where we are

So far we have discussed two of the three factors that make up transaction load:

- The LCN transaction load
- The request transaction load, or the requests “this” node makes to “other” nodes on the UCN.

Next consider the third factor,

- The response transaction load, or the requests made on “this” node from “other” nodes on the UCN.

Continued on next page

Check UCN Load, Continued

Response transaction load

To calculate the response transaction load you could do the following:

- Response transaction load =
  Input scan table requests from other nodes
  + Continuous store transactions from other nodes
  + CL write transactions per second from other nodes
  + CL read transactions per second from other nodes

Alarming and UCN bandwidth

Alarming on the UCN is handled by a different mechanism (Type 1 message) and normally does not add to the transaction load; however, chattering alarms use up UCN bandwidth. This has an effect on the response times of other UCN nodes, which ultimately affects all UCN communication.

Where users typically go wrong

Typically transaction limits are exceeded before parameter limits on the UCN. Note that your experience may differ.

Parameters for peer-to-peer monitoring

R410 and later provides parameters that help in monitoring peer-to-peer communication on the UCN. Some of these parameters are the number of items in the input scan table, parameter and transaction throughput statistics, and node access rates for a UCN node.

REFERENCE—Section 6.7 in the HPM, APM, or PM Implementation Guidelines manual describes the Peer to Peer statistical parameters.

Conclusion

After seeing what contributes to a transaction load, the next topic shows how to calculate a UCN load.

Calculate UCN Load

Introduction

Assume for the moment that you know what your peer-to-peer connections are. The following example shows a way to calculate the peer-to-peer load.

Calculate the number of messages

A table (or personal computer spreadsheet) is created that shows the number of requests that each node is sending to the other node. The table can be setup to

- Add the rows to get the number of messages “this” node is making of “other” UCN nodes.
- Add the columns to get the number of messages the “other” nodes are making of “this” node.
- Summarize the results in another table (refer to Table 2).

Table 1 Total Requests Table

	Destination Node Transactions per second = (sum of input scan table values, output store, CL reads and writes)				
Originating node	**9**	**11**	**13**	**15**	**Total requests from this node**
9	—	5	2	5	12
11	1	—	1	3	5
13	5	2	—	0	7
15	4	2	4	—	10
Total requests from other nodes	10	9	7	8	

Continued on next page

Calculate the load

Take the results and put into the following table to calculate the load.

Table 2 Total Transactions

UCN node	requests from this node	requests from other nodes	Totals
9	12	10	22
11	5	9	14
13	7	7	14
15	10	8	18

These values are within limits, so these nodes should have no peer-to-peer problems.

Continued on next page

A slight change in terminology

You may have noticed in the last calculation that instead of using the term "transactions," the term "requests" is used. Requests from this node can also be referred to as "messages sent" and requests from other nodes can be called "messages received." Some engineers use the terms message and transaction to mean the same thing; however, by definition

- A transaction consists of a message sent and a message received

Assuming that

- Each *request* from this node (message sent) results in a response (message received), and
- Each *request* from other nodes (message received) results in a response (message sent),

the total transactions can be calculated by dividing the total messages (messages sent + messages received) by 2 as shown in Table 3.

Table 3 Total Transactions as Messages

UCN node	requests from this node (message sent)	response to requests from this node (message received)	requests from other nodes (message received)	response to requests from other nodes (message sent)	total messages	Total transactions (messages/2)
9	12	12	10	10	44	22
11	5	5	9	9	28	14
13	7	7	7	7	28	14
15	10	10	8	8	36	18

You can see from Table 3 that each *request* (from this node or from other nodes) is equivalent to one transaction for the purpose of calculating peer-to-peer load. You can use in your calculations the number of requests (messages) instead of transactions. How to calculate the number of requests is shown in the next section, along with additional UCN strategies.

Summary

You have seen what contributes to the UCN health, LCN load, UCN load. The next section provides some strategies for improving UCN communication performance.

Strategies for Optimal UCN Communication

Evaluate the Source

Introduction

Throughout this course module it has been implied that a thorough understanding of your system configuration is required before you can begin to improve performance at the UCN level. You may have already seen some solutions in the problems themselves. This section will provide suggestions and expand on earlier ones. As you review this section, you may even come across solutions that are not listed here, but would coincide with the concepts you now understand about peer-to-peer communication.

Evaluate the source

Evaluate your NIM/UCN loading by using the TOOLKIT displays; then pursue one or more strategies.

Overview of strategies

Your strategies should consider three areas:

1. LCN to UCN communication requests,
2. Local UCN communication requests,
3. UCN health monitoring

LCN to UCN Request Strategies

Introduction

This section summarizes ways that LCN to UCN requests can be optimized. They include:

- Optimizing graphics
- Evaluating plant operating characteristics
- Using diagnostic and utilities judiciously

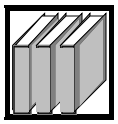
Optimize graphics

Honeywell provides guidelines for optimizing graphics, helping to make them more efficient and less demanding on the UCN. While these guidelines are covered in more detail in the publications, they are summarized here for your convenience.

Here's what you can do

A summary of techniques include

- Request only the data that is needed. As more sites have operations groups build graphics, their need for engineering guidance is significant. It is not unusual to hear of displays built requesting graphics that contain static information (that is, information that does not change) such as range limits, alarm limits, or point descriptors that are updated every 4 seconds. In the case of point limits, you may want to have the display update the information once. Point descriptor text could be entered into the schematic, or updated once at screen callup.
- Enter a realistic update rate. For example 8 seconds for temperature data or level data may be just as relevant as 4-second updates.
- Access the NIM parameter HIGHAL rather than the PM parameter PTINAL when possible. This keeps the access at the LCN level, eliminating the need for a UCN request.
- Group your data so the requests are more efficient. The requests can be grouped by node type and processor type. Even within processor types, requests can be grouped. Refer to the publications for various examples of doing this.
- Add another NIM to increase throughput. For example, a site requires that UCN nodes be on the same physical UCN hiway for peer-to-peer communication. The site decides that the amount of LCN requests through one NIM is too much of a load, leading to a decision to add a NIM.



REFERENCE—Refer to the *Picture Editor Reference manual,* Appendix I, Binder TPS 3032-2 for information on optimizing graphics. This publication provides various examples that are studied in detail.

Continued on next page

LCN to UCN Request Strategies, Continued

Evaluate plant characteristics

Evaluate your plant operating characteristics for occasions where the Universal Station(s) may be accessing large amounts of data along with other LCN nodes (such as auto-checkpointing, reporting). Two examples follow that illustrate "what not to do."

Example #1

At one site a plant was going through a startup process that eventually was to be monitored by eight Universal Stations; however, an additional console had been cutover to help in the startup observation, bringing the number of Universal Stations to 16. This had the effect of doubling the load on the NIM. If scenarios such as this one are intended to be supported, you should consider optimizing schematics.

Example #2

At another site, a regular pattern of UCN overruns was observed. Careful investigation revealed that all the free format logs were set up to print at the same time, sending a request burst to the UCN.

Here's what you can do

These two examples illustrate that you can implement plant procedures that

- Determine operating procedures for various situations.
- Plan when LCN requests (auto-checkpointing, journals, etc.) are scheduled.

Use diagnostics and utilities judiciously

As you have seen in the earlier Figures, the diagnostic displays are parameter-intensive. Often it is not necessary for all stations in a console to have the same display invoked; only one station needs to have the display invoked when a troubleshooting session is underway.

Additionally, you should be aware that Documentation Tool node queries add requests to the UCN; therefore, "wild card" searches can add additional loads to the UCN.

Here's what you can do

You can do the following:

- Determine proper troubleshooting procedures. For example, only one US needs to have the diagnostic display invoked when troubleshooting a communication problem. Remember, if numerous stations invoke the displays, it affects or contributes to the data you are trying to monitor.
- Build your own schematic that calls in four or five key statistics. We will show you how to do that in the NIM-loading course module.

Summary

You may find other ways to optimize LCN requests that are not listed here. Next consider UCN strategies.

Local UCN Strategies

Introduction

The local UCN communication strategies include the following:

- Locate points within the same UCN node to reduce peer-to-peer communication.
- Limit communication mechanisms to a single transaction mechanism.
- Analyze your system for peer-to-peer “ghost” points or inactive points
- Use Honeywell’s pack and unpack routines.
- Limit peer-to-peer communication to three or four nodes.
- Use control point-to-control point peer-to-peer, not control point-to-I/O point peer-to-peer.
- Group your messages to the same cycle
- Configure a longer input scan interval
- Design star-type configurations carefully.
- Look for chattering alarms.
- Push when you need fast communication.

Locate points in same PM

The overall peer-to-peer load is reduced by moving some I/O within the same UCN node. This decision usually is made during the overall design phase.

Single transaction mechanism

Some users prefer limiting peer-to-peer transactions to one or two types of mechanisms using logic block pulls as the main peer-to-peer transaction mechanism.

A general rule is that if transactions are less than 50/second, use pulls. That way you do not have to be concerned about which 1/4-second cycle the slot is executed in. Using pulls also means you have a fixed rate of input scanning.

If output connections are used, an effective strategy is to use logic block pushes. Note that the APM and HPM logic slot has the ability to write on output change only (boolean data, through the use of flag1). With any HPM/APM/PM logic slot you can use a logic slot flag that permits write access, or a digital input flag as a logic output enable source. You can then set up your peer output strategy to send its output only on demand.

Continued on next page

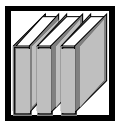
"Ghost" or inactive points

"Ghost" points or inactive points still can have an affect on peer-to-peer load.

- Note that if a point is configured for peer-to-peer and is deleted at the NIM level only, a "ghost" point results. It will still cause peer-to-peer transactions to occur.
- Inactive points with peer input connections continue to have their inputs scanned into the input scan table. Inactivating points does not disable peer-to-peer input scanning. (This often happens when a PM/APM/HPM database is rebuilt, but the node's database was not initialized.)

Limit communication

Limit communication to 3 or 4 nodes. Briefly, capacity is inversely proportional to the number of nodes.



REFERENCE—Refer to the publication *HPM or APM Implementation Guidelines* for more details on transaction load and number of nodes on a UCN.

Refer to Application Note SL-53-252 for an example of pack and unpack routines.

Use control to control

Use control-to-control peer-to-peer communication as opposed to control-to-I/O peer-to-peer communication connections. This is because control-to-I/O peer-to-peer connections require the destination node's communication processor to make a request to its I/O Link processor for the information. Less time is required if the connection is control-to-control.

Group your output connections

If you decide to use output connections, use the schedule display (see Figure 23) to determine when your point is being executed. Points that have output connections that have the same node destination are combined into one UCN message. This helps to reduce the number of transactions.

If output connections are sent to different node destinations across the four cycles, determine whether it is feasible to reconfigure the output connections so that they are sent to the same node destination.

Longer scan interval

Beginning in R410, you can configure a 1-second scan interval instead of 1/2-second for the input scan table. Be aware that changing the scan interval can alter the characteristics of the existing control strategy. Analyze the effect before making any changes.

Continued on next page

Local UCN Strategies, Continued

Star configurations	Design star-type configurations carefully in peer-to-peer communication. This configuration is one where multiple nodes all get and store data from one node. In this type of configuration, be sure to include loading factors.
Look for chattering alarms	Periodically call up the PV retrieval display for each unit and look for chattering alarms. A chattering alarm could be an open thermocouple or digital contact that needs a proper delay time configured. Because your point could be on a 1/4 cycle, you can see how an alarm would add to the UCN traffic.
When should you push?	When should you push? Push if your system needs communication at a 1/4-second rate (most users do not require this speed). One effective way to do this is to use the APM/HPM's array point as your platform for pushing. Group the array point data into logic blocks and push them at the same time.
Optimization of schematics	When assigning Collection Rates, assign control points and IOP points to separate sets. Refer to schematic optimization guidelines for additional details.

UCN Health Strategies

Introduction

In addition to periodic maintenance recommendations listed in the publications, you can do the following:

- Build a diagnostic display that focuses on the key parameters listed earlier. This reduces the parameter load on the UCN.
- Historize overrun counters. A sample program is provided to do this.

REFERENCE—Refer to the publication *Universal Control Network Guidelines*, which contains maintenance recommendations for the UCN.

Trend data

While the UCN statistics display provides a wealth of information, you can also trend the parameters listed in Table 4. For your convenience, we can provide all the parameters should you decide additional parameters are needed.

UCN parameters

You can build a display that monitors the health of the communication system. UCN statistics (Figure 29) can be called into a custom display by referencing the following parameters

Table 4 Diagnostic Parameters

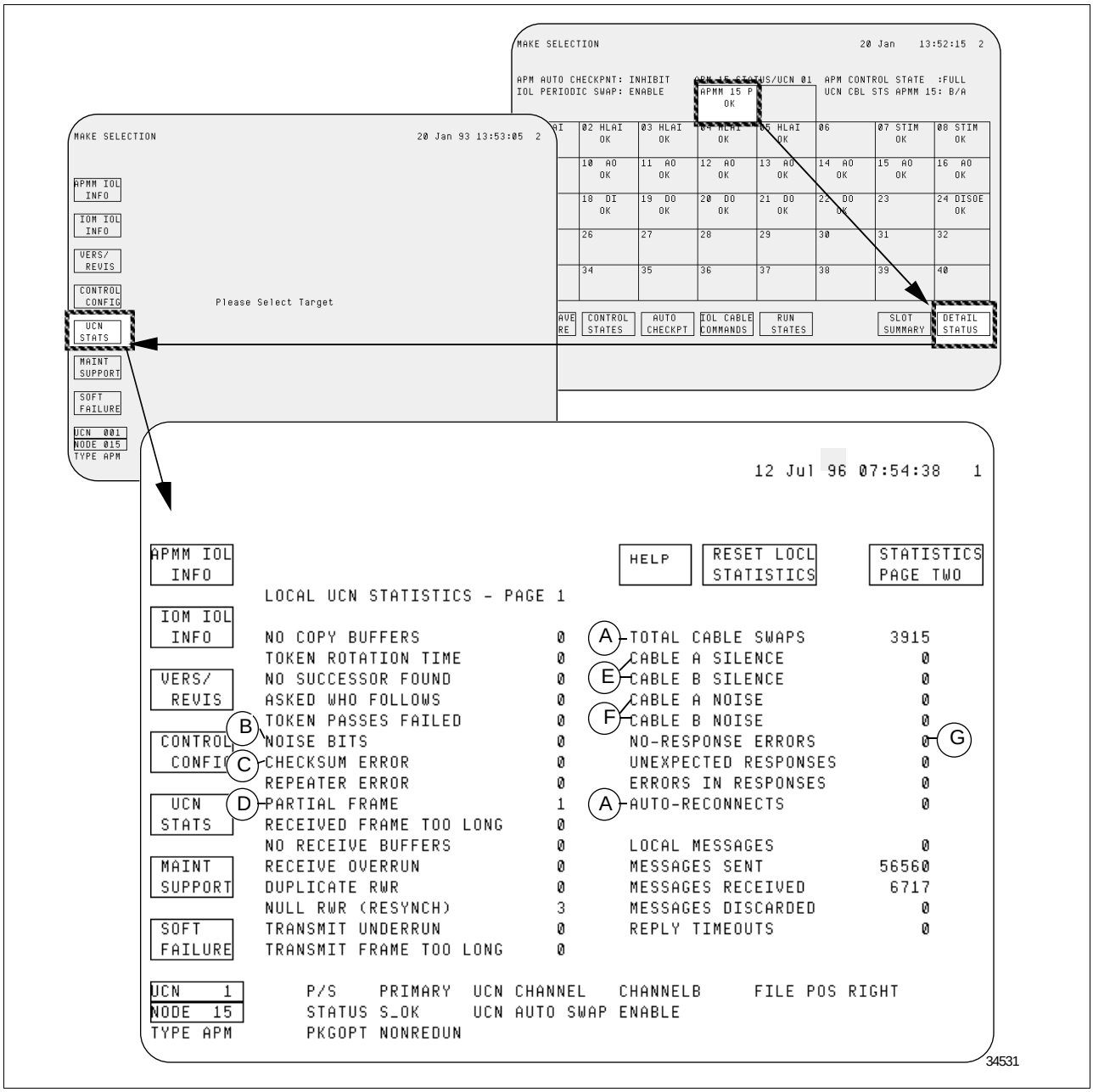
Item	Description	Parameter
A	Total Cable Swaps Total Auto Reconnects	$UCNLSB(17) $UCNLSB(43)
B	Noise bits	$UCNLSB(6)
C	Checksum Error	$UCNLSB(7)
D	Partial Frame	$UCNLSB(9)
E	Cable A Silence	$UCNLSB(18)
E	Cable B Silence	$UCNLSB(19)
F	Cable A Noise	$UCNLSB(20)
F	Cable B Noise	$UCNLSB(21)
G	No response errors	$UCNLSB(22)

Continued on next page

Reference display

Recall that the parameters listed in Table 4 correspond to those shown in the UCN statistics display (Figure 29.)

Figure 29 Diagnostic Display with Parameters Referenced



Continued on next page

Historize messages Some sites historize the transaction loads with a program similar to Figure 30.

Figure 30 Message Historization Program

```
UCNCOM.CL
01/12/96 10:20:21

PACKAGE
CUSTOM
--
PARAMETER   BOXREAD     “BOX READS”
VALUE       0.0
PARAMETER   BOXWRIT     “BOX WRITES”
VALUE       0.0
PARAMETER   BOXRDCG     “BOX READ CHANGE PER MINUTE”
VALUE       0.0
PARAMETER   BOXWRCG     “BOX WRITE CHANGE PER MINUTE”
VALUE       0.0
PARAMETER   BOXLSRD     “BOX LAST READS”
VALUE       0.0
PARAMETER   BOXLSWR     “BOX LAST WRITES”
VALUE       0.0
--
END CUSTOM
--
BLOCK UCNCOM (GENERIC; AT PV_ALG)
--
--    UCN MONITOR PROGRAM TO KEEP TRACK OF UCN
--    MESSAGES
--
PARAMETER   PVCALC
PARAMETER   PVAUTOST : PVVALST

--CALCULATE READ MESSAGES PER SECOND
IF BOXLSRD > BOXREAD THEN SET BOXLSRD = BOXLSRD - 65535
SET BOXRDCG = (BOXREAD - BOXLSRD)/60
SET BOXLSRD = BOXREAD

--CALCULATE WRITE MESSAGES PER SECOND
IF BOXLSWR > BOXWRIT THEN SET BOXLSWR = BOXLSWR - 65535
SET BOXWRCG = (BOXWRIT - BOXLSWR)/60
SET BOXLSWR = BOXWRIT

--CALCULATE TRANSACTIONS PER SECOND
CALL ALLOW_BAD (PVCALC, (BOXRDCG+BOXWRCG)/2)

END UCNCOM
END PACKAGE
```

Continued on next page

AM Point Definition

Figure 31 shows a sample AM point definition used in conjunction with the CL/AM message historization program.

Figure 31 AM Point Definition

```
SYSTEM ENTITY UCNMON09( )

NAME            =“UCNMON09         “
UNIT            = AM
PTDESC       =“UCN BOX 09 MONITOR      “
EUDESC       =“MSG/SEC”
KEYWORD      =“UCN09  “
PRIMMOD      =“UCN09  “
PTDISCL      = FULL
PVALGID      = CL
CTLALDIG     = NULL
OVERVAL      = 25
SUPPIO       = NOSUPPR
$IPPASN      = OFF
PERIOD       = 1MIN
BEFAFT       = NO
PVFORMAT     = D1
PVEUHI       = 100.0000
PVEULO       = 0.000000
PVEXEUHI     = 100.0000
PVEXEULO     = 0.000000
PVCLAMP      = CLAMP
PVSRCOPT     = ONLYAUTO
PVFLTOPT     = NONE
PVTV         = --------
PVALDB       = ONE
PVHITP       = 30.00000
PVLOTP       = --------
PVHHTP       = --------
PVROCPTP     = --------
PVROCNTP     = --------
ALENBST      = ENABLE
ALPRIOR      = LOW     (Before R500)
(R500 and later—Separate Alarm Priorities:)
   ADVDEVPR  = LOW     CLEALMPR  = LOW     PVHHPR  = LOW
   BADCTLPR  = LOW     CLFALMPR  = LOW     PVHIPR  = LOW
   BADPVPR   = LOW     CNFERRPR  = LOW     PVLLPR  = LOW
   BCLEALPR  = LOW     DEVHIPR   = LOW     PVLOPR  = LOW
   BCLFALPR  = LOW     DEVLOPR   = LOW     PVROCNPR = LOW
                                           PVROCPPR = LOW
                                           PVSGCHPR = LOW
```

Continued on next page

AM Point Definition, continued

Figure 31 AM Point Definition, *continued*

```
CCINPT       = NO
CLSLOTS      = 1
NOPKG        = 1
NUMSWTCH     = 0
NOGINPTS     = 2
NOGOPTS      = 0
PKGNAME(1)   =“UCNMON  “
BOXREAD        = 0.0000
BOXWRIT        = 0.0000
BOXRDCG        = 0.0000
BOXWRCG        = 0.0000
BOXLSRD        = 0.0000
BOXLSWR        = 0.0000
GISRC(1)       = $NM12B09.$UCNLSB(27)
GIDSTN(1)      = BOXREAD
GIACTSTS(1)    = ACTIVE
GISRC(2)       = $NM12B09.$UCNLSB(28)
GIDSTN(2)      = BOXWRIT
GIACTSTS(2) = ACTIVE
```

Purpose of program

The CL/AM program in Figure 30 gives you the number of messages in a UCN node. The CL program collects the number of reads and number of writes. The program subtracts the “old” reading from the “current” reading to show the changes. Because the counters are totalizers, the program also accounts for situations where the counter is reset.

Peer-to-Peer Load Identification

Collect and Collate the Data

Introduction

This section introduces techniques to determine the transaction load placed on the UCN. Using the methodology shown here, you can determine the transaction load caused by peer-to-peer communication.

Overview of procedure

The procedure is pretty straightforward:

- Collect a node's peer-to-peer connections.
- Determine how many messages are sent
  - If output connections are used, refer to the point execution schedule to determine how many output requests are made
  - If input connections are used, calculate the number of messages
- Repeat this for remaining nodes
- Collate the data from "other" nodes that send messages to "this" node.

Collect connections

More than one approach is available to collect data showing the peer-to-peer connections, depending on the release your system is presently on.

- On R400 and earlier systems, you can do the following:
  - Command a Data Out (DO) to a file
  - Use a Find Names command with a wild card (*) to list entity references.
  - Use a Find Names command with a wild card (*) to list points built.
  - Sort the results by tagname
- On R400 and earlier systems, you can also
  - Take a copy of your Exception Build (EB) files to Workbook
  - Convert the file to PC format
  - Search your EB files with a sort program
  - Sort the peer-to-peer connections in a tabular printout
- On R410 and later systems, you can request peer connections by using Find Names. An example of this display is in Figure 32.

Statistics

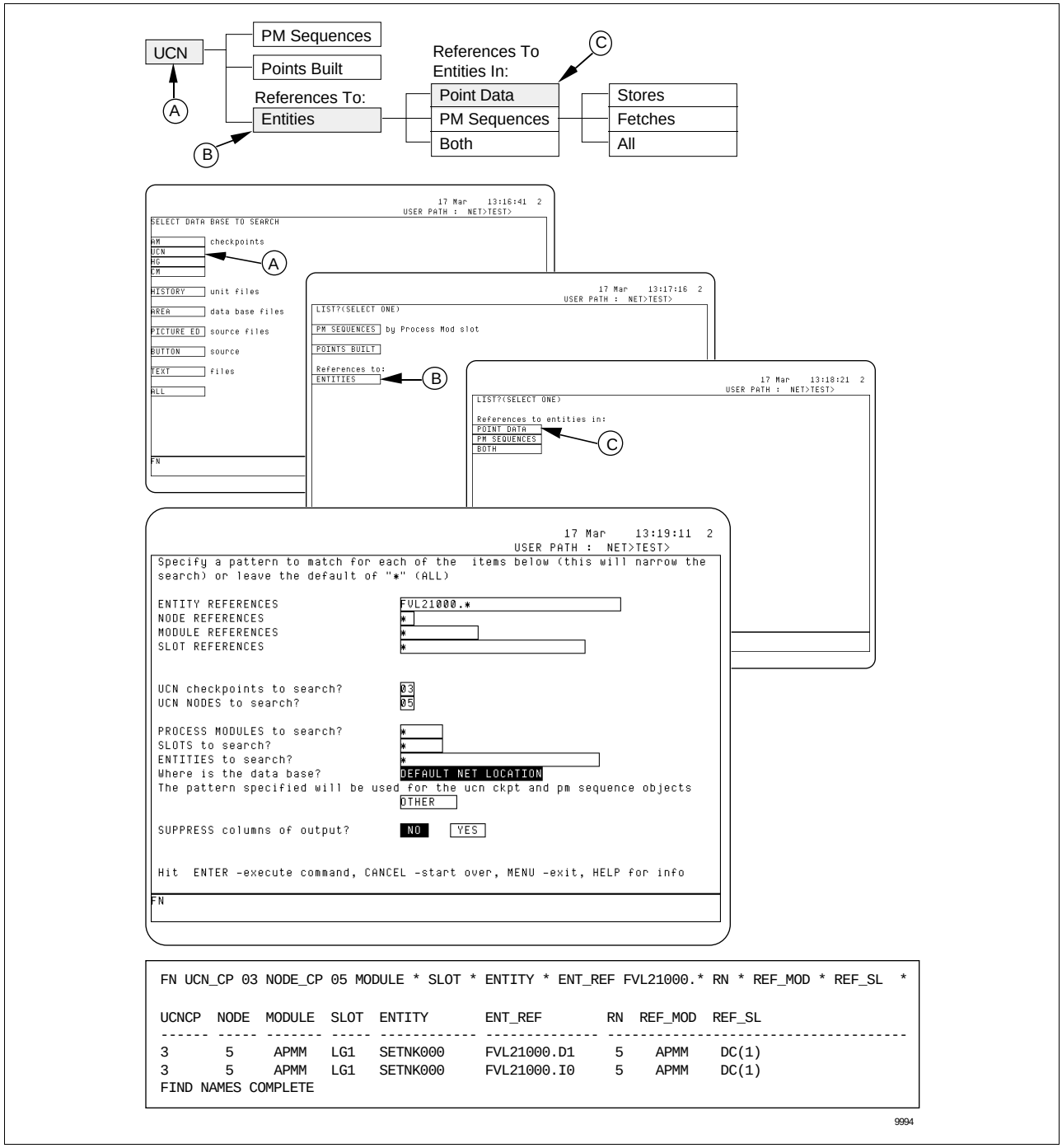
R410 and later provides statistics for transactions, parameter requests, and CPUFREE. These displays conveniently provide data on a box level and network level.

Continued on next page

Example display

Figure 32 shows the ports you would select to do a search for peer-to-peer connections. (Using Find Names for UCN management tasks is covered in more detail in another course module; however, you will get a chance to use Find Names to do a peer-to-peer search in the lab exercise.)

Figure 32 Find Names Peer-to-Peer Support (R410 and later)



Example Peer-to-Peer Sort

Calculation procedure

An outline of your procedure includes the following:

- Note the UCN node's input scan cycle and point execution cycle for later calculations.
- Calculate the number of pull messages "this" node is making from "other" nodes.
- Calculate the number of push messages "this" node is sending to "other" nodes.
- Calculate the number of pull messages "other" nodes are making from "this" node.
- Calculate the number of push messages "other" nodes are sending to "this" node.
- Total the numbers for the peer-to-peer transaction load for the node.
- Repeat the calculation for the remaining nodes.

Calculation approach

Your calculation can be summarized as follows:

Table 5 Summary of Calculations

Request description	Calculation	Result
Pulls to this node	# requests * input scan /sec	
Pushes from this node	# requests * (cycle/second)	
Pulls from other nodes	# requests * input scan /sec	
Pushes from other nodes	# requests * (cycle/second)	
Total transactions		

Example calculation

In this example, calculate the load for UCN node #9. In an actual application, you would calculate the peer-to-peer load for all UCN nodes. For the example, assume that the

- Input scan cycle is 1/2 second,
- Point execution cycle for all points is 1/2 second, and
- Assume that no CL read/writes are occurring.

Continued on next page

Example printout

Table 6 shows an example spreadsheet format you can list your points into. Having the list of peer-to-peer connections, you can begin your calculations of peer-to-peer loads.

Table 6 Sample Peer-to-Peer Listing

Name	Point type	Node #	Slot #	Pulled tagname	Pulled parameter	Pull node #	Pushed tagname	Pushed parameter	Push node #
RRL01	LOGICNIM	09	28	FDL01	SO(1)	11			
RRL01	LOGICNIM	09	28	IDL01	SO(1)	11			
RRL02	LOGICNIM	09	29	TFF01	PVFL	17			
RRL02	LOGICNIM	09	29	TFF02	PVFL	17			
UML01	LOGICNIM	09	2	DIMT1	PVFL	19			
RRL01	LOGICNIM	09	28	FWL01	SO(10)	21			
RRL02	LOGICNIM	09	29	FWL02	SO(13)	21			
RRL02	LOGICNIM	09	29	FWL03	PVFL	21			
FFAC04	REGCLNIM	17	23	BMC05	OP	09			
FFAC04	REGCLNIM	17	25	BMC05	OP	09			
WPL11	LOGICNIM	19	28	UMF01	PVFL	09			
WPL21	LOGICNIM	19	27	UMF02	PVFL	09			
FWL01	LOGICNIM	21	17	BMC05	OP	09			
FFAL01	LOGICNIM	17	7				BMC02	OP	09
WPL01	LOGICNIM	21	27				WPF1	PVFL	09
WPL01	LOGICNIM	21	27				WPF2	PVFL	09

Continued on next page

Calculate pulls to“this” node

Because you are calculating the number of pull messages node 9 is making, you need the tags in node 9 that are requesting data. The point connections that are requesting data in Table 6 are listed in Table 7 for your convenience.

Table 7 Points Requesting Data

Name	Point type	Node #	Slot #	Pulled tagname	Pulled parameter	Pull node #	Pushed tagname	Pushed parameter	Push node #
RRL01	LOGICNIM	09	28	FDL01	SO(1)	**11**			
RRL01	LOGICNIM	09	28	IDL01	SO(1)	**11**			
RRL02	LOGICNIM	09	29	TFF01	PVFL	**17**			
RRL02	LOGICNIM	09	29	TFF02	PVFL	**17**			
UML01	LOGICNIM	09	2	DIMT1	PVFL	**19**			
RRL01	LOGICNIM	09	28	FWL01	SO(10)	**21**			
RRL02	LOGICNIM	09	29	FWL02	SO(13)	**21**			
RRL02	LOGICNIM	09	29	FWL03	PVFL	**21**			

Because you know that

- inputs are scanned twice a second,
- requests to the same node are combined in one message, and
- requests for the same point.parameter count as one input in the scan table,

you can say that eight message requests are made. The results are based on the following:

- Four message requests are sent every 1/2 second. This is because the data resides in four “other” nodes and the parameter requests are combined in four separate messages.
- The four requests are sent out twice a second, resulting in eight messages.

Continued on next page

Example Peer-to-Peer Sort, Continued

Calculation approach

Your calculation thus far is

Request description	Calculation	Result
Pulls to this node	# requests * input scan /sec 4 requests * 2 scan/sec =	 8
Pushes from this node	# requests * (cycle/second)	
Pulls from other nodes	# requests * input scan /sec	
Pushes from other nodes	# requests * (cycle/second)	
Total transactions		

Calculate pushes

As you review Table 6, note that node #9 is not pushing any data.

Calculation approach

Your calculation thus far is

Request description	Calculation	Result
Pulls to this node	# requests * input scan /sec 4 requests * 2 scan/sec =	 8
Pushes from this node	# requests * (cycle/second) 0 requests * 4 =	 0
Pulls from other nodes	# requests * input scan /sec	
Pushes from other nodes	# requests * (cycle/second)	
Total transactions		

Continued on next page

Calculate pulls from "other" nodes

Since you are calculating the number of pull messages "other" nodes are making from node 9, you need the tags from "other" nodes that are requesting data. The point connections that are requesting data in Table 6 are listed in Table 8 for your convenience.

Table 8 Points Pulling Data

Name	Point type	Node #	Slot #	Pulled tagname	Pulled parameter	Pull node #	Pushed tagname	Pushed parameter	Push node #
FFAC04	REGCLNIM	**17**	23	BMC05	OP	09			
FFAC04	REGCLNIM	**17**	25	BMC05	OP	09			
WPL11	LOGICNIM	**19**	28	UMF01	PVFL	09			
WPL21	LOGICNIM	**19**	27	UMF02	PVFL	09			
FWL01	LOGICNIM	**21**	17	BMC05	OP	09			

Because you know that

- input connections are scanned twice a second, and
- requests to the same node are combined in one message,

you can observe that the number of "pull" messages from "other" nodes is six. This is because

- three message requests are sent every 1/2 second. This is because three "other" nodes are requesting the data from "this" node.
- The three requests are sent out twice a second, resulting in six messages.

Continued on next page

Calculation approach

Your calculation thus far is

Request description	Calculation	Result
Pulls to this node	# requests * input scan cycle/sec 4 requests * 2 scan/sec =	8
Pushes from this node	# requests * (cycle/second) 0 requests * 4 =	0
Pulls from other nodes	# requests * input scan cycle/sec 3 requests * 2 scan/sec =	6
Pushes from other nodes	# requests * (cycle/second)	
Total transactions		

Calculate pushes from "other" nodes

Because you are calculating the number of push messages "other" nodes are sending to node 9, you need the tags from "other" nodes that are sending data. The point connections that are requesting data in Table 6 are listed in Table 9 for your convenience.

Table 9 Requests from Other Nodes

Name	Point type	Node #	Slot #	Pulled tagname	Pulled parameter	Pull node #	Pushed tagname	Pushed parameter	Push node #
FFAL01	LOGICNIM	**17**	7				BMC02	OP	09
WPL01	LOGICNIM	**21**	27				WPF1	PVFL	09
WPL01	LOGICNIM	**21**	27				WPF2	PVFL	09

Because you know that

- output connections are sent twice a second, and
- requests to the same node are combined in one message,

you can observe that the number of "push" messages from "other" nodes is four. This is because

- two message requests are sent every 1/2 second. This is because two "other" nodes are sending the data to "this" node.
- The two requests are sent out twice a second, resulting in four messages.

Continued on next page

Example Peer-to-Peer Sort, Continued

Calculation approach

Your calculation results are

Request description	Calculation	Result
Pulls to this node	# requests * input scan/sec 4 requests * 2 scan/sec =	8
Pushes from this node	# requests * (cycle/second) 0 requests * 2 =	0
Pulls from other nodes	# requests * input scan /sec 3 requests * 2 scan/sec =	6
Pushes from other nodes	# requests * (cycle/second) 2 requests * 2=	4
Total transactions		18

Summary

In this example, the transactions for UCN node #9 are 18, well below the transaction limit. Of course, you would have to repeat the calculations for the remaining nodes on your UCN system to calculate their peer load as well. Additionally, the calculations did not include any LCN requests.

Continued on next page

Example Peer-to-Peer Sort, Continued

Spreadsheet

For your convenience, blank spreadsheets are provided on the next three pages.

Name	Point type	Node #	Slot #	Pulled tagname	Pulled parameter	Pull node #	Pushed tagname	Pushed parameter	Push node #

Request description	Calculation	Result
Pulls to this node	# requests * input scan /sec	
Pushes from this node	# requests * (cycle/second)	
Pulls to other nodes	# requests * input scan /sec	
Pushes from other nodes	# requests * (cycle/second)	
Total transactions		

Continued on next page

Example Peer-to-Peer Sort, Continued

Spreadsheet

Name	Point type	Node #	Slot #	Pulled tagname	Pulled parameter	Pull node #	Pushed tagname	Pushed parameter	Push node #

Request description	Calculation	Result
Pulls to this node	# requests * input scan /sec	
Pushes from this node	# requests * (cycle/second)	
Pulls to other nodes	# requests * input scan /sec	
Pushes from other nodes	# requests * (cycle/second)	
Total transactions		

Continued on next page

Example Peer-to-Peer Sort, Continued

Spreadsheet

Name	Point type	Node #	Slot #	Pulled tagname	Pulled parameter	Pull node #	Pushed tagname	Pushed parameter	Push node #

Request description	Calculation	Result
Pulls to this node	# requests * input scan /sec	
Pushes from this node	# requests * (cycle/second)	
Pulls to other nodes	# requests * input scan /sec	
Pushes from other nodes	# requests * (cycle/second)	
Total transactions		

Lab Exercise

Lab 1—Demonstration of Chattering Alarms

Overview

In the following lab exercise, the effects of chattering alarms are demonstrated on the UCN system. Although a chattering alarm can be disabled or inhibited, it adds to the UCN communication load.

Lab prerequisites

The following are lab prerequisites:

- This lab must be conducted as a group exercise to monitor the effects.
- A “quiet” UCN system is desirable to monitor the effects of chattering alarms.

Cause alarms to occur

Step	Action
1	Cause an alarm to occur; observe the effect on Type 1 messages sent.
2	Disable alarm reporting.
3	Cause an alarm to occur again; observe the effect on Type 1 messages sent.
4	Inhibit alarm reporting.
5	Cause an alarm to occur again; observe the effect on Type 1 messages sent.

Lab 2—Build Simple Peer-to-Peer Connection

Introduction

In the following lab exercise, a simple peer-to-peer connection is built on a logic slot. This demonstrates a common approach to peer communication, using logic block pulls.

Lab prerequisites

The following are some lab prerequisites:

- An existing logic slot with available logic inputs.
- More than one UCN node on the network capable of peer-to-peer.

Add peer connection

Step	Action
1	Inactivate the logic slot SETNK###, where ### represents your partition number.
2	Reconstitute SETNK###.
3	Find an unused logic input connection on SETNK###. Record the index # of the logic input connection you plan to use. _____ Make an entry that represents a box numeric from a peer UCN device. For example: Your entry could look like this: $NMxxByy. NN(##), where xx is the UCN number, yy is the UCN address, and ## represents the last 2 numbers of your partition.
4	Load SETNK### to the UCN node.
5	Activate SETNK###. You can activate SETNK### from the Slot Summary display, the DATACHNG schematic, or the SETNK### Detail display.
6	Verify that your point pulls the data correctly by making changes to the box numeric and observing the changes at SETNK###. NOTE: You can do this from the DATACHNG display.
7	• Call up the Toolkit DATACHNG display. • Enter SETNK###.L(index#) where index# is the # recorded in Step 3. • Try to change the numeric you were pulling. • Inactivate SETNK### from DATACHNG by – Entering in SETNK###.PTEXECST – After the value of PTEXECST is displayed, change it to INACTIVE • Observe that - - - - appears. • Observe that - - - - appears also in SETNK###'s detail display. Although your schematic and detail display do *not* show the most current data, the input scan table continues to pull data.

Lab 3—Find a Peer-to-Peer Connection

Introduction

In the following lab exercise, the Find Names utility is used to locate the peer connection you had previously configured. While Find Names is covered in more detail later in this course, you will see one of the tools you can use to locate peer connections.

Find the connection

Step	Action
1	Demand a checkpoint (from the UCN node that has your logic slot) to get the latest data on your checkpoint file.
2	Call up the Command Processor.
3	Copy the checkpoint for your assigned UCN node to S###. (Note: Find Names will later search your checkpoint. Having it on removable media or another HM directory helps reduce the possibility of accidentally corrupting our lab checkpoint.)
4	From the Command Processor, type in FN, press [ENTER]. This will invoke the Find Names function.
5	Select the UCN target from the checkpoint field.
6	Select the ENTITIES target (do not select POINTS BUILT target.)
7	After the screen updates, select the POINT DATA target.
8	After the screen updates, enter the following: • In the ENTITY REFERENCES port, enter a system identifier with wild card characters. For example, enter $nm01B??.nn* in the port. This will capture all system numeric references, including your lab partner's. • Enter the UCN checkpoint number. For example, enter 01 for UCN network #1. • Enter the UCN node number to search. For example, 07 if your UCN node address is #7. • To search your student directory, select the target next to the prompt "Where is the database?" Select the OTHER target. Enter "NET>S###", where ### is your partition number. • The remaining entries can be default entries. Press [ENTER], and any system reference defined in that node's checkpoint is returned.

Lab 4—Trend UCN Communication

Introduction

This exercise demonstrates using trend displays or other Toolkit displays to trend or monitor UCN communication.

Trend UCN

Step	Action
1	Call up the Toolkit schematic for data change. Press the [SCHEM] key and enter DATACHNG.
2	Enter the system references for parameters of interest to monitor them. Some example parameters are: • Messages sent ($NMxxByy.$UCNLSB(27)) • Messages received ($NMxxByy.$UCNLSB(28)) Refer to the module Interpret UCN Communications for a parameter listing of other UCN statistics of interest.
3	Repeat some of the same entries from the Toolkit schematic for quick trends. Press the [SCHEM] key and enter QUIKTRND.

Lab 5—Build a UCN Message Monitor Program (Before R410)

Overview

In the following lab exercise, build a CL/AM program that monitors UCN message statistics. The program calculates messages per second based on a one minute average. Note that R410 and later provides convenient displays that track similar data. One advantage this program has for any release is that you can set a goal for message counts (through an alarm trip point) and monitor your system's performance. Systems operating on releases earlier than R410 would find this program essential to tracking UCN traffic.

Program constraints

Resetting the statistics in the local statistics display is not necessary; however, if you do reset statistics, it may temporarily give you a false message count calculation. The false reading occurs because the program accounts for the message counter rolling over at 65535.

Also note that Type 1 event messages are not collected by this program. You can add Type 1 message counters to this program, or to another point that is set up to monitor event traffic.

Sample program

A sample program is already provided for you on one of the system directories. If the program directory and file differs from the one listed in the lab, your course manager will inform you as to which file you can use.

Sample point data

The sample program uses a regulatory control point in an AM.

```
NAME      =“UCNMS###“
UNIT      = AM
PTDESC    =“UCN PM 07 MONITOR      “
EUDESC    =“MSG/SEC”
KEYWORD   =“PM07  “
PRIMMOD   = ---
PTDISCL   = FULL
PVALGID   = CL
CTLALDIG  = NULL
OVERVAL   = 0
SUPPIO    = NOSUPPR
$IPPASN   = OFF
PERIOD    = 1MIN
BEFAFT    = NO
PVFORMAT  = D1
PVEUHI    = 100.0000
PVEULO    = 0.000000
PVEXEUHI  = 100.0000
PVEXEULO  = 0.000000
```

Sample point data,
continued

```
PVCLAMP    = CLAMP
PVSRCOPT   = ONLYAUTO
PVFLTOPT   = NONE
PVTV       = --------
PVALDB     = ONE
PVHITP     = 30.00000
PVLOTP     = --------
PVHHTP     = --------
PVROCPTP   = --------
PVROCNTP   = --------
ALENBST    = ENABLE
ALPRIOR    = LOW   (Before R500)
(R500 and later—Separate Alarm Priorities:)
   ADVDEVPR  = LOW     CLEALMPR  = LOW     PVHHPR  = LOW
   BADCTLPR  = LOW     CLFALMPR  = LOW     PVHIPR  = LOW
   BADPVPR   = LOW     CNFERRPR  = LOW     PVLLPR  = LOW
   BCLEALPR  = LOW     DEVHIPR   = LOW     PVLOPR  = LOW
   BCLFALPR  = LOW     DEVLOPR   = LOW     PVROCNPR = LOW
                                           PVROCPPR = LOW
                                           PVSGCHPR = LOW
CCINPT     = NO
CLSLOTS    = 1
NOPKG      = 1
NUMSWTCH   = 0
NOGINPTS   = 2
NOGOPTS    = 0
PKGNAME(1)  =“UCNMON“
BOXREAD    = 0.0000000
BOXWRIT    = 0.0000000
BOXRDCG    = 0.0000000
BOXWRCG    = 0.0000000
BOXLSRD    = 0.0000000
BOXLSWR    = 0.0000000
GISRC(1)    =  $NM12B09.$UCNLSB(27)
GIDSTN(1)   =  BOXREAD
GIACTSTS(1) =  ACTIVE
GISRC(2)    =  $NM12B09.$UCNLSB(28)
GIDSTN(2)   =  BOXWRIT
GIACTSTS(2) =  ACTIVE
```

Continued on next page

Lab 5—Build a UCN Message Monitor Program (Before R410), Continued

Example program

Your sample program is similar to the following example program. You need to copy the program from our lab directory to your directory, then edit (to change ### to your partition number), compile, and link your program to an AM point.

```
BLOCK UCNMN### (POINT UCNMS###; AT PV_ALG)
--
--    UCN MONITOR PROGRAM TO KEEP TRACK OF UCN NODE MESSAGES
--
--
PARAMETER   PVCALC
PARAMETER   PVAUTOST : PVVALST

--CALCULATE READ MESSAGES PER SECOND
IF BOXLSRD > BOXREAD THEN SET BOXLSRD = BOXLSRD - 65535
SET BOXRDCG = (BOXREAD - BOXLSRD)/60
SET BOXLSRD = BOXREAD

--CALCULATE WRITE MESSAGES PER SECOND
IF BOXLSWR > BOXWRIT THEN SET BOXLSWR = BOXLSWR - 65535
SET BOXWRCG = (BOXWRIT - BOXLSWR)/60
SET BOXLSWR = BOXWRIT

--CALCULATE TRANSACTIONS PER SECOND
CALL ALLOW_BAD (PVCALC, (BOXRDCG+BOXWRCG)/2)

END UCNMN###
```

Example segment

Your AM point uses a custom data segment that is already built for you. You do *not* need to recompile this segment, just add it to your AM point as UCNMON.

```
CUSTOM
PARAMETER   BOXREAD     “BOX READS”
VALUE       0.0
PARAMETER   BOXWRIT     “BOX WRITES”
VALUE       0.0
PARAMETER   BOXRDCG     “BOX READ CHANGE PER MINUTE”
VALUE       0.0
PARAMETER   BOXWRCG     “BOX WRITE CHANGE PER MINUTE”
VALUE       0.0
PARAMETER   BOXLSRD     “BOX LAST READS”
VALUE       0.0
PARAMETER   BOXLSWR     “BOX LAST WRITES”
VALUE       0.0
END CUSTOM
```

Continued on next page

Lab 5—Build a UCN Message Monitor Program (Before R410), Continued

Build it, and it will run

Build an AM point, using a reference to the custom data segment on our system. Copy an existing program to a file referencing your partition and then edit, compile, and link the program to your AM point. Run the program to monitor the messages from an APM or PM.

Step	Action
1	Build the AM point UCNMS### referenced in the data sheets.
2	Load the AM point UCNMS### to the AM.
3	Call up the Command Processor.
4	Copy the CL program UCNMSG.CL to your assigned partition; be sure to reference your partition number. Example copy command: COPY NET>CL>UCNMSG.CL NET>S###>UCNMN###.CL, where ### is your partition number.
5	Edit the program to change ### to your partition number.
6	Compile your CL program. Example compile command: CL NET>S###>UCNMN###.CL, where ### is your partition number.
7	Link the CL program UCNMN###.CL to your assigned point. Example link command: LK NET>S###>UCNMN### UCNMS###, where ### is your partition number.
8	Call up a detail display of your AM point, UCNMS###.
9	Activate your AM point. (Initially, the point may show a high value until the program executes on its interval.)
10	Your program is based on a 1-minute interval. You can however, demand a point process special (note that this will give you less than 1-minute data).
11	Note your PVHITP value. This value arbitrarily identifies a message count that when exceeded, is annunciated as an alarm.

Go to the next procedure when complete.

Lab 6—Build a CPUFREE Monitor (Before R410)

Overview

In the following lab exercise, build an AM point that monitors CPUFREE. Note that R410 provides statistics and displays that monitor CPUFREE. An advantage of this program is that you can monitor the CPUFREE value and alarm it at a user-chosen value (typically 20%).

Program constraints

R410 is required to run this exercise.

Sample point data

The sample AM point uses a regulatory control point in an AM.

```
NAME        =“CPUFR###“
UNIT        = AM
PTDESC      =“UCN PM 09 CPUFREE      “
EUDESC      =“CPUFREE”
KEYWORD     =“PM09  “
PRIMMOD     = ---
PTDISCL     = FULL
PVALGID     = CL
CTLALDIG    = NULL
OVERVAL     = 0
SUPPIO      = NOSUPPR
$IPPASN     = OFF
PERIOD      = 1HR
BEFAFT      = NO
PVFORMAT    = D1
PVEUHI      = 100.0000
PVEULO      = 0.000000
PVEXEUHI    = 100.0000
PVEXEULO    = 0.000000
PVCLAMP     = CLAMP
PVSRCOPT    = ONLYAUTO
PVFLTOPT    = NONE
PVTV        = --------
PVALDB      = ONE
PVHITP      = --------
PVLOTP      = 20.00000
PVHHTP      = --------
PVROCPTP    = --------
PVROCNTP    = --------
```

Continued on next page

Sample point data, continued

```
ALENBST      = ENABLE
ALPRIOR      = LOW   (Before R500)
(R500 and later—Separate Alarm Priorities:)
  ADVDEVPR     = LOW     CLEALMPR  = LOW     PVHHPR  = LOW
  BADCTLPR     = LOW     CLFALMPR  = LOW     PVHIPR  = LOW
  BADPVPR      = LOW     CNFERRPR  = LOW     PVLLPR  = LOW
  BCLEALPR     = LOW     DEVHIPR   = LOW     PVLOPR  = LOW
  BCLFALPR     = LOW     DEVLOPR   = LOW     PVROCNPR = LOW
                                             PVROCPPR = LOW
                                             PVSGCHPR = LOW
CCINPT       = NO
CLSLOTS      = 0
NOPKG = 1
NUMSWTCH     = 0
NOGINPTS     = 7
NOGOPTS      = 0
PKGNAME(1)   =“CPUMON“
AVG_COM      = 0.0000000
MAX_COM      = 0.0000000
MIN_COM      = 0.0000000
AVG_CTL      = 0.0000000
MAX_CTL      = 0.0000000
MIN_CTL      = 0.0000000
GISRC(1)     = $NM12B09.COMCFAVG
GIDSTN(1)    = AVG_COM
GIACTSTS(1) = ACTIVE
GISRC(2)     = $NM12B09.COMCFMAX
GIDSTN(2)    = MAX_COM
GIACTSTS(2) = ACTIVE
GISRC(3)     = $NM12B09.COMCFMIN
GIDSTN(3)    = MIN_COM
GIACTSTS(3) = ACTIVE
GISRC(4)     = $NM12B09.CTLCFAVG
GIDSTN(4)    = AVG_CTL
GIACTSTS(4) = ACTIVE
GISRC(5)     = $NM12B09.CTLCFMAX
GIDSTN(5)    = MAX_CTL
GIACTSTS(5) = ACTIVE
GISRC(6)     = $NM12B09.CTLCFMIN
GIDSTN(6)    = MIN_CTL
GIACTSTS(6) = ACTIVE
GISRC(7)     = $NM12B09.COMCFAVG
GIDSTN(7)    = PVCALC
GIACTSTS(7) = ACTIVE
```

Continued on next page

Lab 6—Build a CPUFREE Monitor (Before R410), Continued

Example segment

Your AM point uses a custom data segment that is already built for you. You do *not* need to recompile this segment, just add it to your AM point as CPUMON.

```
CUSTOM
--
PARAMETER   AVG_COM     "AVERAGE CPUFREE COMM PROC"
VALUE       0.0
PARAMETER   MAX_COM     "MAXIMUM CPUFREE COMM PROC"
VALUE       0.0
PARAMETER   MIN_COM     "MINIMUM CPUFREE COMM PROC"
VALUE       0.0
PARAMETER   AVG_CTL     "AVERAGE CPUFREE CTL PROC"
VALUE       0.0
PARAMETER   MAX_CTL     "MAXIMUM CPUFREE CTL PROC"
VALUE       0.0
PARAMETER   MIN_CTL     "MINIMUM CPUFREE CTL PROC"
VALUE       0.0
--
END CUSTOM
--
```

Build it, and it will run

Build an AM point, then monitor the CPUFREE from an APM or PM.

Step	Action
1	From your assigned Universal Station, call up the Engineering Main Menu
2	Build the AM point, CPUFR###, referenced in the data sheets.
3	Load the AM point to the AM.
4	Call up a detail display of your AM point, CPUFR###.
5	Activate your AM point
6	Your program is based on a 1-hour interval. You can however, demand a point process special.
7	Note your PVLOTP value. This value arbitrarily identifies a CPUFREE value (usually 20%) that can be annunciated as an alarm.

Go to the next procedure when complete.

Lab 7—Build a UCN Overrun Monitor

Overview

In the following lab exercise, build an AM point that monitors UCN overruns. Note that overrun occurrences do *not* always mean you have a problem, they may occur in optimally operating systems. When overruns become excessive, you should consider further investigation of your UCN performance. An advantage of this program is that you can monitor the total network overruns value and alarm it at a user-chosen value.

Sample program

A sample program is already provided for you on one of the system directories. If the program directory and file differs from the one listed in the lab, your course manager will inform you as to which file you can use.

Sample point data

The sample AM point uses a regulatory control point in an AM.

```
NAME        =“OVMN###“
UNIT        = AM
PTDESC      =“UCN 12 OVERRUNS      “
EUDESC      =“OVERRUN”
KEYWORD     =“    “
PRIMMOD     = ---
PTDISCL     = FULL
PVALGID     = CL
CTLALDIG    = NULL
OVERVAL     = 0
SUPPIO      = NOSUPPR
$IPPASN     = OFF
PERIOD      = 1HR
BEFAFT      = NO
PVFORMAT    = D1
PVEUHI      = 100.0000
PVEULO      = 0.000000
PVEXEUHI    = 100.0000
PVEXEULO    = 0.000000
PVCLAMP     = CLAMP
PVSRCOPT    = ONLYAUTO
PVFLTOPT    = NONE
PVTV        = --------
PVALDB      = ONE
PVHITP      = --------
PVLOTP      = --------
PVHHTP      = --------
PVROCPTP    = --------
PVROCNTP    = --------
```

Continued on next page

Sample point data, continued

```
ALENBST      = ENABLE
ALPRIOR      = LOW   (Before R500)
(R500 and later—Separate Alarm Priorities:)
  ADVDEVPR     = LOW     CLEALMPR  = LOW     PVHHPR  = LOW
  BADCTLPR     = LOW     CLFALMPR  = LOW     PVHIPR  = LOW
  BADPVPR      = LOW     CNFERRPR  = LOW     PVLLPR  = LOW
  BCLEALPR     = LOW     DEVHIPR   = LOW     PVLOPR  = LOW
  BCLFALPR     = LOW     DEVLOPR   = LOW     PVROCNPR = LOW
                                             PVROCPPR = LOW
                                             PVSGCHPR = LOW
CCINPT       = NO
CCINPT       = NO
CLSLOTS      = 1
NOPKG = 1
NUMSWTCH     = 0
NOGINPTS     = 0
NOGOPTS      = 0
PKGNAME(1)   =“OVRMON“
OVRUN07      = 0.0000000
OVRUN17      = 0.0000000
```

Example program

Your sample program is similar to the following example program. You need to copy the program from our lab directory to your directory, then edit (to change ### to your partition number), compile, and link your program to an AM point.

```
BLOCK OVRMN### (POINT OVMN###; AT PV_ALG)
--
--    UCN MONITOR PROGRAM TO KEEP TRACK OF UCN OVERRUNS
--
EXTERNAL $NM12B09
EXTERNAL $NM01B19

PARAMETER   PVCALC
PARAMETER   PVAUTOST : PVVALST

--COLLECT OVERRUN COUNTS
SET OVRUN09 = $NM12B09.LSUCNORN
SET OVRUN19 = $NM01B19.LSUCNORN

--CALCULATE TOTAL UCN NETWORK OVERRUNS
SET PVCALC = OVRUN09 + OVRUN19

END OVRMN###
```

Continued on next page

Lab 7—Build a UCN Overrun Monitor, Continued

Example segment

Your AM point uses a custom data segment that is already built for you. You do *not* need to recompile this segment, just add it to your AM point as OVRMON.

```
CUSTOM
PARAMETER   OVRUN09     “LAST HOUR UCN OVERRUN”
VALUE       0.0
PARAMETER   OVRUN19     “LAST HOUR UCN OVERRUN”
VALUE       0.0
END CUSTOM
```

Continued on next page

Lab 7—Build a UCN Overrun Monitor, Continued

Build it, and it will run

Build an AM point, using a reference to the custom data segment on our system. Copy an existing program to a file referencing your partition and then edit, compile, and link the program to your AM point. Run the program to monitor the overruns from an APM or PM.

Step	Action
1	Build the AM point OVMN### referenced in the data sheets.
2	Load the AM point OVMN### to the AM..
3	Call up the Command Processor.
4	Copy the CL program OVRMN.CL to your assigned partition; be sure to reference your partition number. Example copy command: COPY NET>CL>OVRMN.CL NET>S###>OVRMN###.CL, where ### is your partition number.
5	Edit the program to change ### to your partition number.
6	Compile your CL program. Example compile command: CL NET>S###>OVRMN###.CL, where ### is your partition number.
7	Link the CL program OVRMN###.CL to your assigned point. Example link command: LK NET>S###>OVRMN### OVMN###, where ### is your partition number.
8	Call up a detail display of your AM point, OVMN###.
9	Activate your AM point. You may see 0 for your overrun count on our lab system, because the UCN communication load is very light.
1 0	Your program is based on a 1-hour interval. You can however, demand a point process special.

Go to the next procedure when complete.

Lab 8—Collect Your Performance Monitoring Alarms

Overview

In the following lab exercise, build a point that collects the alarms for your system performance monitors. Your point provides a convenient mechanism to do this, through the use of the parameter PRIMMOD. Instead of collecting alarms on a unit basis, they can be collected on "batch" or "alarm group" basis.

You can use any type of point as a PRIMMOD point. For this exercise, use a process module point. You will reuse the process module point in the CL exercises that follow.

Sample point data

The point that you will build is similar to the following.

```
NAME        ="PACK###"
NODETYP     = HPM
PNTFORM     = FULL
PTDESC      =  "   "
UNIT        = AM
NTWKNUM     = 12
NODENUM     = 09
SLOTNUM     = 24
PRIMMOD     = --
USERID      = --
SEQSLTSZ    = 20
CNTLLOCK    = OPERATOR
SPLOCK      = OPERATOR
RSTROPT     = OFF
ACP         = --
ALPRIOR     = LOW (before R500)
SEQPR       = LOW (R500 and later)
```

Lab prerequisites

Your course manager will assign to you an unused sequence slot (SLOTNUM) to use in the lab exercise.

Continued on next page

Build it, and it will collect

Build a process module point, then cause your AM monitor points to go into alarm. Call up the event retrieval display to observe that PRIMMOD provides a convenient mechanism for collecting alarms.

Step	Action
1	Build the point PACK###. Use your assigned unit and node number for the slot nunber, use the last 2 digits of your partition number.
2	Load the point PACK### to the node.
3	Call up your AM performance monitor points (CPUFR### and UCNMS###) and make them INACTIVE.
4	Reconstitute your AM performance monitor points (CPUFR### and UCNMS###) and enter the tagname PACK### for the parameter PRIMMOD.
5	Load your AM performance monitor points (CPUFR### and UCNMS###) and make them ACTIVE.
6	Call up your AM performance monitor points (CPUFR### and UCNMS###) and cause them to go into alarm. (Note: You may have to change alarm limits, then do point process specials to cause an alarm trip.)
7	From the System Menu, do the following: • Call up the Event History Menu • Select the PROCESS ALARMS target. • Select the PRIMMOD target. • Select a PRIMMOD# target. • Enter your tagname, PACK###. • Select the DISPLAY target.
8	You should now observe the alarms you had created. Using the point as an alarm collector provides you an additional tool in managing your system.

Go to the next procedure when complete.

Lab 9—Build a Packing Routine

Introduction

In this lab exercise you will use a technique (based on Application Note SL-53-252) to pack flag array values into a numeric. Packing and unpacking methods optimize the use of the input scan table, particularly when logical (boolean) values are requested and the limit of 50 inputs must be exceeded.

Lab overview

The lab exercise uses only the pack portion of Application Note SL-53-252. If you can pack an array, it is fairly easy to add code to unpack.To minimize your programming efforts, a pack routine is already provided for you. Use the point, PACK###, built in an earlier exercise.

Example program

Your program is similar to the following example. Copy the program from our lab directory to your directory, then edit (to change ### to your partition number), compile, and load your program.

```
SEQUENCE PAKFL### (HPM; POINT PACK###)
--    PROGRAM TO PACK FLAG ARRAY
EXTERNAL !BOX
LOCAL PACKBIT : LOGICAL ARRAY(1..16) AT FL(01)
LOCAL PACKNUM AT NN(01)

PHASE DECODE
S1: CALL PACK (!BOX.NN(###), PACKBIT)
GOTO S1
END PAKFL###
SUBROUTINE PACK(UNPKEDNM:OUT NUMBER;PKBIT:IN LOGICAL ARRAY(1..16))
IF PKBIT(01)= ON THEN SET PACKNUM = 1
            ELSE SET PACKNUM = 0
IF PKBIT(02)= ON THEN SET PACKNUM = PACKNUM + 2
IF PKBIT(03)= ON THEN SET PACKNUM = PACKNUM + 4
IF PKBIT(04)= ON THEN SET PACKNUM = PACKNUM + 8
IF PKBIT(05)= ON THEN SET PACKNUM = PACKNUM + 16
IF PKBIT(06)= ON THEN SET PACKNUM = PACKNUM + 32
IF PKBIT(07)= ON THEN SET PACKNUM = PACKNUM + 64
IF PKBIT(08)= ON THEN SET PACKNUM = PACKNUM + 128
IF PKBIT(09)= ON THEN SET PACKNUM = PACKNUM + 256
IF PKBIT(10)= ON THEN SET PACKNUM = PACKNUM + 512
IF PKBIT(11)= ON THEN SET PACKNUM = PACKNUM + 1024
IF PKBIT(12)= ON THEN SET PACKNUM = PACKNUM + 2048
IF PKBIT(13)= ON THEN SET PACKNUM = PACKNUM + 4096
IF PKBIT(14)= ON THEN SET PACKNUM = PACKNUM + 8192
IF PKBIT(15)= ON THEN SET PACKNUM = PACKNUM + 16384
IF PKBIT(16)= ON THEN SET PACKNUM = PACKNUM + 32768
SET UNPKEDNM = PACKNUM
END PACK
```

Continued on next page

Lab 9—Build a Packing Routine, Continued

Build it, and it will pack

Step	Action
1	Call up the Command Processor.
2	Copy the CL program PACK.CL to your assigned partition; be sure to reference your partition number. Example copy command: COPY NET>CL>PACK.CL NET>S###>PACK###.CL, where ### is your partition number.
3	Edit the program to change ### to your partition number.
4	Compile your CL program. Example compile command: CL NET>S###>PACK###.CL, where ### is your partition number. (Note: a -UL compile option may be required to get your program name into the NIM library.)
5	Copy the CL program object to the volume &E##, where ## is the UCN network number. Example copy command: COPY NET>S###>########.xx NET>&E##>= -D where xx = .PO for PM or .NO for APM/HPM
6	Call up a detail display of your point, PACK###.
7	Load and start your CL program: • Call up a detail display of PACK### • Select the LOAD SAVE target, then LOAD target. • Select the DEFAULT SOURCE target, then EXECUTE COMMAND target. • Select the PAKFL### target that represents your program, then the START target. • Select the MESSAGES target. • Select the text LOAD that appears next to your program name in the upper part of the display. • Select the START target then ENTER target that appears. • Select the RETURN target.

Continued on next page

Build it, and it will pack, continued

8	Your program should at this point be writing to box numeric ###. You can now modify the process module flags (1-16) and verify that the program writes correctly to your box numeric. For example: • Select the MOD FLAGS target to call up your flag array. • Set flag 1 (F0001) and flag 9 (F0009) = on. This will be packed as number 257 into your numeric. • Select the BOX NUMERICS target. • Select the SLOT SELECT target and enter ### for the numeric you were writing to. You should see the number 257 sent by your program. • Change your flags 1-16 to other values. (Select the MOD FLAGS target, make changes, then BOX NUMERICS target to see the changes.
9	Note that your program is not yet performing any peer-to-peer communication. You can use the logic slot SETNK### to push the packed numeric to another node. For example: • Call up SETNK### and inactivate the point. • Reconstitute SETNK###. • Press CTL and F8. Select the NIM INPUT CONNECTIONS target. • Add !BOX.nn(###) as Logic input 12, LISRC (12). Press ENTER. • Press CTL and F8, select NIM LOGIC OUTPUT CONNECTIONS target. • Add $NMxxByy.NN(###) as a logic output destination 6, LODSTN(6). (Note that xx = UCN network number, yy = node address, and ### = partition number. For example, $NM01B17.NN(865). • Add L12 as LOSRC(6). • Change LOENBL(6) to FL2. Press ENTER. • Load SETNK###. • Activate SETNK###.
1 0	Observe from SETNK###'s input connections detail display that values in the process module flag array are passed to your box numeric. From an earlier lab exercise, you had configured an input to pull the same numeric you are pushing to. This helps you to easily verify that your peer-to-peer is taking place.

Go to the next procedure when complete.

Lab 10—Use Configuration Display

Overview

In the following lab exercise, call up the configuration display for your assigned node. Note that R410 and later provides information in this standard display that monitors peer-to-peer performance.

Lab constraint

R410 or later is required.

Display orientation

Call up the display and become familiar with peer-to-peer data.

Step	Action
1	Call up the UCN Status display.
2	Select a node assigned to you.
3	Select the DETAIL STATUS target to call up the Status Display for your UCN node.
4	Select the primary node assigned to you.
5	Select the DETAIL STATUS target.
6	Select the CONTROL CONFIG target.
7	While in this display, identify and enter the following: • Number of items in peer-to-peer input scan table? _____ • Peer-to-peer efficiency? _____ • Scan cycle time? _____
8	Observe that you can change scan cycle time. Because this is a group exercise, not everyone can do this at the same time!! What precautions should be noted when scan times are changed? _____ _____

This concludes the lab exercises. If you have any questions, your course manager can review them with you.

Lab 11—Push Outputs on Demand

Introduction

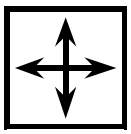
In this lab exercise, you will use a technique to push outputs on demand. You can do this through the use of the logic output enable source, such as turning a flag on or off when you need to push an output. This technique can be used to reduce communication load.

Lab overview

Change the logic slot SETNK### you used in earlier lab exercises. This way you will be able to easily verify that the push occurs only on demand, since the SETNK### point is pushing and pulling to the same numeric.

Step	Action
1	Inactivate your SETNK### point.
2	Reconstitute your SETNK### point.
3	Change the logic slot SETNK### logic output enable source to use a boolean variable (such as flag 12) that you can turn on or off. • Press [CTL] and [F8] Select NIM LOGIC OUTPUT CONNECTIONS target. • Change the logic output enable source for the numeric you are pushing. For example, change LOENBL(6) to FL12. Press [ENTER] • Load SETNK### • Activate SETNK###
4	Observe from SETNK###'s input connections detail display that values are pushed to your box numeric only when the flag is on. From an earlier lab exercise, you had configured an input to pull the same numeric you are pushing to. This helps you easily verify your peer to peer is taking place on demand.

Directions



DIRECTIONS—This is the end of the study material for this module. Discuss questions concerning the study material or the lab activities with a colleague or a course manager

If you are satisfied that you have achieved the objectives of this module, continue with the next section, the Student Proficiency Evaluation.

Student Proficiency Evaluation

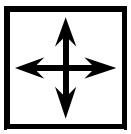
Criterion Test

Case Study

Your course manager may choose to hand out a hypothetical case study based on situations that have been diagnosed and resolved by TAC in the past. After reading the case study, be prepared to join in a class discussion on possible causes of the problems described, ways to analyze the problem further, and feasible resolutions.

Notes

Directions



DIRECTIONS—This is the end of this module.

Use your course map to

- Get your course manager to sign off this module.
- Choose your next eligible module.

If you have a question

- Ask your course manager.

LAST PAGE