Tivoli. software

IBM

IBM Tivoli NetView for z/OS 5.2: Prototype for Enhanced Monitoring of TCP/IP Connections

White Paper

Paul Quigley (quigleyp@us.ibm.com)

December 2006

Table of Contents

Introduction

White Paper : Prototype for Enhanced Monitoring of TCP/IP Connections

Introduction

About this Paper

IBM Tivoli NetView for z/OS (NetView) provides many functions to manage and automate network resources, including TCP/IP resources. NetView 5.1 introduced a policy-based function to monitor TCP/IP connections with the local TCP/IP stack. Included with the monitoring function you can define thresholds for connections that are idle or that exceed a minimum or maximum number of bytes that are sent across the connection.

Two actions are supported when a threshold condition is detected. The connection can be broken and *notification policy* can be invoked to notify operators and key support personnel of the threshold condition. Using the notification policy you can generate messages or e-mails to notify your personnel.

The connection monitoring and thresholding function uses SNMP MIB data to gather information about each connection with the local TCP/IP stack. The connection data is retrieved from the ibmTcpipMvsTcpConnTable MIB table.

There are several problems inherent with this approach. In general, SNMP can be slow and consume CPU cycles. Second, the ibmTcpipMvsTcpConnTable MIB table is extremely large. Detailed information about each connection is contained in 55 MIB variables within the ibmTcpipMvsTcpConnTable MIB table.

In an average customer environment there may be tens of thousands of active connections with applications such as TN3270, FTP, Web browser, SMTP, printer, CICS, and so on.

When using SNMP to **WALK** the ibmTcpipMvsTcpConnTable MIB table **all MIB variables for all connections** are retrieved. For example, for a small environment you may have 10,000 connections. The ibmTcpipMvsTcpConnTable MIB will contain 550,000 MIB variables. There is no way to filter the amount of data retrieved. The policy definitions allow you to look for a specific type of connection based on the application name but the code must still retrieve **all MIB variables for all connections**.

As a result, the NetView connection monitoring and thresholding function has not been implemented by many customers.

This paper will discuss a prototype written by the author that includes several improvements over the NetView code. The prototype will be made available to customers to use. Support for the prototype will be provided by the author *as time permits*.

Audience

This document is primarily intended for system programmers and network administrators responsible for the management of TCP/IP resources.

For example, many customers have printers connected to their z/OS systems using TCP/IP. The connection monitoring and thresholding function, provided by NetView and the prototype discussed in this document, can be used to ensure that connections to the printers are broken if idle to prevent a situation where all ports allocated to the printer application are in use even though no print jobs are scheduled.

Terminology

Throughout this document several terms will be used. They are summarized here:

- **IPCONN:** NetView policy definition statement for TCP/IP connection monitoring and thresholding.
- **TCPCONN:** NetView command to display TCP/IP connection data.
- **FKXECMON:** Name of the NetView routine used to perform TCP/IP connection monitoring and thresholding. FKXECMON is shipped as a compiled REXX EXEC.
- **CONNMON:** Can be used to refer to three items; the name of the prototype, the name of the REXX EXEC used by the prototype to perform TCP/IP monitoring and thresholding, or the name of the in-storage file name used by the prototype to hold port data.
  - The **CONNMON routine** will also be referenced as **CONNMON REXX EXEC** or **CONNMON EXEC**.
  - the CONNMON in-storage file name is referenced as the **CONNMON file**.
  - Any general discussion will be referenced as ***CONNMON*** prototype.

  The CONNMON routine is an enhanced version of FKXECMON and can be used in place of FKXECMON. It is provided as an interpreted REXX EXEC.

  The CONNMON routine replaces the FKXECMON routine.

Acronym Definitions

Several acronyms are used throughout this document and are defined here.

AON: Automated Operations Network (NetView for z/OS component)

AON/TCP: Automated Operations Network / Transmission Control Protocol

CICS: Customer Information Control System

CPU: central processing unit

FTP: File Transfer Protocol

MIB: Management Information Base

NMI: network management interface

REXX: Restructured Extended Executor programming language

SNMP: Simple Network Management Protocol

TCP/IP: Transmission Control Protocol / Internet Protocol

About the Author

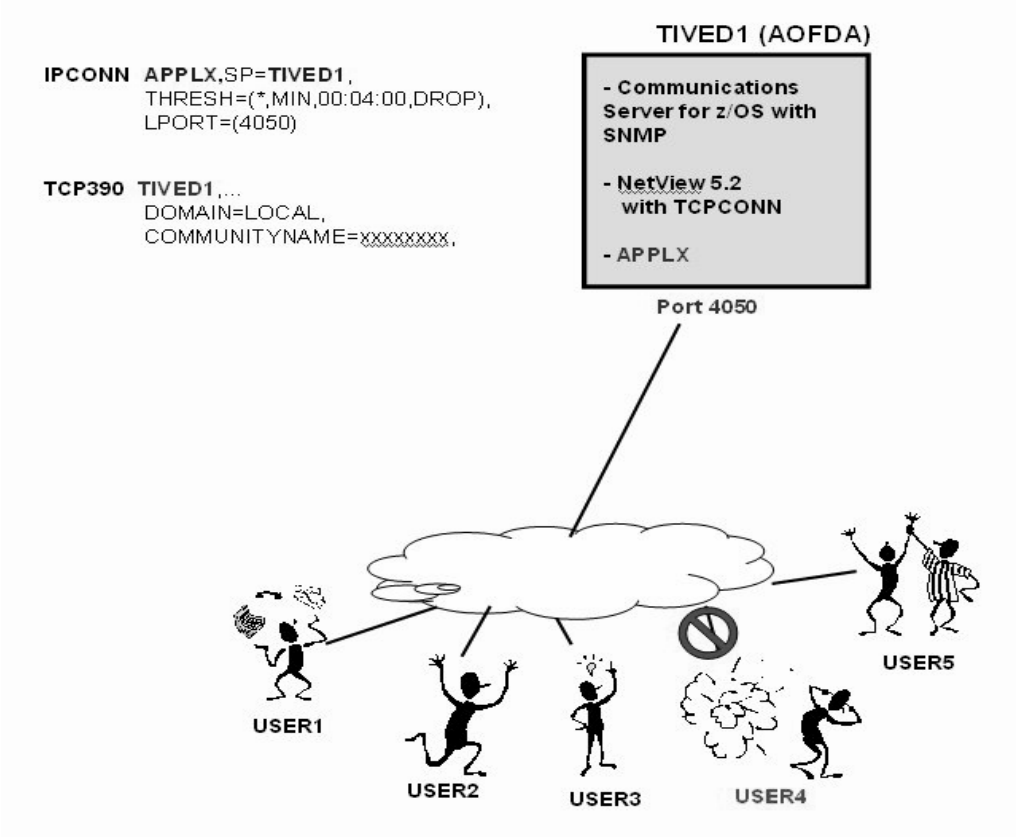
Paul Quigley is a member of the Tivoli Technical Training Course Development organization with responsibilities for developing customer courses for the IBM Tivoli NetView for z/OS and IBM Tivoli System Automation for z/OS products. He has worked on mainframe-based systems and network management, monitoring, and automation for 25 years with an extensive background using the IBM Tivoli NetView for z/OS platform and products. Paul has held various positions within IBM ranging from software test to product design and development to product support and has worked with many customers worldwide to understand their requirements and solve their problems.

White Paper: Prototype for Enhanced Monitoring of TCP/IP Connections

1 Overview and Comparison

The purpose of the connection monitoring and thresholding function is to identify TCP/IP connections that are either idle, sending more bytes than allowed, or sending fewer bytes than expected.

Without getting too deep into the policy definition at this point, an example can be used to illustrate the functionality.



In this example, connections into application APPLX on stack TIVED1 are being monitored from NetView domain AOFDA based on an IPCONN APPLX policy definition.

Note: The LPORT keyword shown in this example is only available with the *CONNMON* prototype. In addition, TCPCONN exploitation is shown and is only available with the *CONNMON* prototype.

User USER4 has established a connection with application APPLX using port 4050. All connections with APPLX will be monitored. Based on the THRESH keyword of the IPCONN APPLX policy, any connection with APPLX will automatically be broken (dropped) if the connection is idle for more than four minutes.

In the example, the connection between APPLX and USER4 has been idle for greater than four minutes and has been broken automatically by the connection monitoring code.

1.1 Usage Examples

Connection monitoring and thresholding can be used to:

Monitor TCP/IP connections to a z/OS printer application and break the connections when they have been idle to maximize printer availability.

Monitor FTP connections for sessions that exceed a maximum byte count and generate a message for key support personnel to increase awareness of large FTP transfers during first shift.

1.2 Overview of NetView Connection Monitoring

NetView provides function to monitor TCP/IP connections and perform thresholding **based on the name of the application.**

The NetView connection monitoring and thresholding is a function of the AON/TCP component.

The routine that is scheduled is known as FKXECMON. For simplicity, the NetView connection monitoring and thresholding will be referred to as FKXECMON in this document.

The initial delivery of FKXECMON was NetView 5.1:

- Policy-based using IPCONN policy
- Timer-driven using an interval defined in the policy
- Utilize SNMP commands only

  - WALK of ibmTcpipMvsTcpConnTable MIB table to collect the connection data

- Support definition of thresholds
  - Using the THRESH keyword of the IPCONN policy
- Supports definition of actions to take
  - Part of the THRESH keyword definition

There are some disadvantages which have limited the customer use of FKXECMON:

- SNMP use can consume CPU cycles.
- SNMP v1 and v2 implementation is not as secure as v3.
- Connection MIB table (ibmTcpipMvsTcpConnTable) can be *very* large.

  At the time of publishing this document, the ibmTcpipMvsTcpConnTable MIB table contains 55 MIB variables for each connection. FKXECMON must retrieve the entire ibmTcpipMvsTcpConnTable MIB table to determine the active connections and then use six (6) of the 55 MIB variables for the connections that FKXECMON is interested in. In other words, all 55 MIB variables are retrieved for every connection. FKXECMON ignores the remaining MIB variables as well as connections not being monitored.

- Processing the MIB data can be error prone as new connections are started and old connections end.
- Since the monitoring is at the application level there may be ports monitored that you do not want to be monitored.

  After the TCP/IP connection data is retrieved from the ibmTcpipMvsTcpConnTable MIB table, a comparison is done to filter the connection data based on the ibmMvsTcpConnResourceName MIB variable. The ibmMvsTcpConnResourceName MIB variable is compared to the name of the IPCONN policy.

Connections that do not match the IPCONN policy are ignored after retrieving and processing data for all of the connections. This may not be the best approach, but it was the only method available when the connection monitoring function was developed.

1.3 Overview of Prototype Connection Monitoring

The prototype of connection monitoring and thresholding will be called *CONNMON* in this document.

The *CONNMON* prototype is a subtower of the TCPCONN tower. If AON/TCP is enabled then the *CONNMON* prototype code will supersede the AON/TCP code.

CONNMON is a rewrite of FKXECMON that contains several enhancements:

- NetView 5.2 introduced the TCPCONN command to display active connection data based on an improved network management interface (NMI) with Communications Server for z/OS. *CONNMON* utilizes the TCPCONN command to gather active connection data by local stack TCP/IP address and local port instead of an SNMP WALK of the ibmTcpipMvsTcpConnTable MIB table.

  TCPCONN is a more robust interface. Using TCPCONN reduces the amount of data collected, parsed, and interrogated.

  TCPCONN does not store detailed connection data. Therefore, it is still necessary to use SNMP to retrieve the detailed connection data. The overall amount of SNMP data retrieved is much less with this approach and is therefore quicker.

- Enhance IPCONN policy with an additional parameter, LPORT, to identify one or more port numbers, including a range of ports to improve performance by reducing the amount of connection data to interrogate.

  Defining a range of ports to monitor may also simplify the amount of IPCONN policy definitions. For example, with FKXECMON you may have to define 10 IPCONN port statements for 10 different ports that an application may use. With *CONNMON* you only need to define one IPCONN policy definition and specify the port range with the LPORT keyword.

- Port data can be stored statically in NetView storage or retrieved dynamically. If all ports to be monitored are static then performance can be improved by loading static port data in NetView storage instead of retrieving port data dynamically from SNMP MIB variables every time the *CONNMON* routine is invoked.

- New CNMSTYLE common global definitions to control port data collection and logging to simplify policy definitions.

- *CONNMON* can be run independent of AON/TCP.

Compared with FKXECMON, the CONNMON prototype monitors connections **based on the local port or the application name.**

1.4 Comparison: FKXECMON versus CONNMON

Because of the differences in collecting the connection data (SNMP WALK ibmTcpipMvsTcpConnTable versus TCPCONN) there are functional differences between the NetView supplied code (FKXECMON) and the prototype code (*CONNMON*):

Task	Prototype	NetView 5.2
Communication to remote z/OS host	**Not applicable**	**SNMP**
Definitions	**Base policy plus CNMSTYLE**	**AON/TCP policy**
Data collection	**TCPCONN to determine active connections followed by SNMP GET commands for details**	**SNMP WALK of a *potentially huge MIB* table followed by SNMP GET commands**
Data filtering	**TCPCONN KEEP – very flexible**	**Hard-coded for loopback only**
Monitoring level	**Application or port**	**Application**
Performance Improvements	**• Static port data can be kept in storage** **• New LPORT parameter reduces amount of data collected** **• Data filtered with TCPCONN.KEEP**	**Not applicable**

The *CONNMON* prototype is functionally comparable to FKXECMON. The only function *CONNMON* does not support is monitoring connections on a remote TCP/IP stack.

- Monitoring connections on a remote TCP/IP stack can be accomplished through the use of RMTCMD communication between the NetView domains. This was not implemented in the prototype code but can be implemented at a later date by the NetView development team if they choose to adopt the *CONNMON* prototype.

The biggest advantage to using the *CONNMON* prototype is the improved performance and flexibility over the AON/TCP FKXECMON.

1.5 Restrictions

The *CONNMON* prototype does not support:

- Non-Netview connection to target host. FKXECMON uses SNMP so NetView is not required at the target host.

  This design uses TCPCONN, requiring monitoring to be done locally.

- IPv6 connections.

  IPv6 support requires version neutral MIBs from Communications Server for z/OS. Some of the detailed connection data is missing with the newer version neutral MIBs. Therefore, IPv6 support is not possible at this point in time.

  CONNMON does contain code to filter IPv6 connection data that may appear through the use of the TCPCONN command. This code allows CONNMON to be *IPv6 tolerant*.

  Note that FKXECMON does not support IPv6 either.

- SNMPv3 requests.

  Note that FKXECMON does not support SNMPv3 either.

- Web browser connections (default port 80).

  Web browser connections are highly transient, rendering any monitoring functions useless. These should be filtered with TCPCONN KEEP definitions in CNMSTYLE.

  FKXECMON retrieved all connections to all ports and all local addresses. Web browser connection data is retrieved but ignored since FKXECMON is only interested in a subset of the connections based on the IPCONN policy.

- Connections to the loopback address (127.0.0.1).

  CONNMON requires the loopback address can be filtered with TCPCONN KEEP definitions in CNMSTYLE.

  FKXECMON is hardcoded to filter these connections.

1.6 Overview of NetView Dynamic TCP/IP Stack Discovery

Connection monitoring and thresholding requires TCP/IP stack policy definitions to define the stack to NetView. Regardless of whether you are running FKXECMON or the *CONNMON* prototype it is important to understand what you need to define for your local TCP/IP stack.

With NetView 5.2, the TCP/IP stacks on every system where NetView is running are dynamically discovered. As the stacks are discovered the policy definitions (TCP390 definition statement) are also dynamically created.

There may be cases where you still need to define a small subset of the TCP/IP stack policy. For example, SNMP requests need the SNMP community name defined for the stack policy. You should define the stack policy and include the SNMP community name on the stack policy definition.

When NetView discovers the TCP/IP stack it will combine your static policy definitions such as the community name with the dynamically discovered policy definitions.

For example, suppose you have a stack whose short name is TIVED1 and you need to implement connection monitoring. You would then define the stack policy for TIVED1 with only the community name defined:

```
TCP390 TIVED1 COMMUNITYNAME=publicv2c
```

As NetView dynamically discovers the stack, it will add all other related information to the policy definition. You can verify this by issuing a **POLICY ENTRY=TCP390,TYPE=TIVED1** command:

```
POLICY ENTRY=TCP390,TYPE=TIVED1
EZL115I  TCP390 TIVED1 TCPNAME TCPIP
EZL115I  TCP390 TIVED1 IPADDR 10.44.15.200
EZL115I  TCP390 TIVED1 IPADDRTYPE 01
EZL115I  TCP390 TIVED1 PRIMARYINTERFACE 10.44.15.200
EZL115I  TCP390 TIVED1 PRIMARYINTERFACETYPE 01
EZL115I  TCP390 TIVED1 HOSTNAME TIVED1.TIVED.IBM.COM
EZL115I  TCP390 TIVED1 COMMUNITYNAME publicv2c
EZL115I  TCP390 TIVED1 DOMAIN LOCAL
EZL115I  TCP390 TIVED1 DYNDISC YES
EZL115I  TCP390 TIVED1 HIER2 SP-APPL
EZL115I  TCP390 TIVED1 HIER3 NETSP
EZL115I  TCP390 TIVED1 FORMAT STACK
EZL115I  TCP390 TIVED1 STATUS (NORMAL,DEGR*,THRESH*)
EZL115I  TCP390 TIVED1 IDSAUTO Y
EZL115I  TCP390 TIVED1 IDSINTVL 00:10
EZL115I  TCP390 TIVED1 UNIXSERV YES
EZL002I END
```

The static policy items such as COMMUNITYNAME are merged with the dynamic policy (TCPNAME, IPADDR, HOSTNAME, and so on).

The dynamic stack discovery feature means fewer policy definitions which can correlate to fewer keystroke errors and improved time to value. The policy will be updated every time the stack initializes to pick up changes to pertinent information.

2 Defining Connection Monitoring and Thresholding

2.1 Overview

This section will discuss the definitions needed to implement connection monitoring and thresholding with the *CONNMON* prototype:

- TCP/IP stack policy (TCP390 policy definition)
- Connection monitoring and thresholding policy (IPCONN policy definition)
- Notification policy (NOTIFY policy definition) if running AON/TCP
- CNMSTYLE customization

In addition, an overview of customization for NetView and Communications Server for z/OS is provided.

2.2 Stack Policy

Each local TCP/IP stack must be defined for connection monitoring to occur. TCP/IP stacks are defined using the TCP390 policy definition statement. NetView 5.2 will dynamically discover your TCP/IP stacks and create the necessary TCP390 policy definitions for each stack. The TCP390 policy definition that is created will be merged with any user overrides (for example, COMMUNITYNAME).

The default NetView DSIPARM member for TCP/IP stack policy definition is CNMPOLCY.

The complete syntax for the TCP390 policy definition can be found in the *IBM Tivoli NetView for z/OS: Administration Reference* manual. Two keywords are pertinent to a discussion of the *CONNMON* prototype:

TCP390	***stack_name*,** **DOMAIN=LOCAL | *remote_domainID*,** **…** **COMMUNITYNAME=*community_name***

- COMMUNITYNAME=*community_name*

  Required to retrieve detailed connection data from SNMP MIBs. Detailed connection data is not known to TCPCONN for active connections so it must be retrieved from SNMP MIBs.

  – Where community_name is the SNMP community name defined to TCP/IP. community_name can be mixed case format.
  – Appropriate security measures should be taken to prevent unauthorized personnel from obtaining the community name.

- DOMAIN=LOCAL

  The DOMAIN keyword is used by *CONNMON* to identify the local TCP/IP stack.

  The local TCP/IP stack will be used as the default stack if one is not specified on the IPCONN policy.

  – The domain is determined by dynamic stack discovery. You do not need to code this keyword.

As discussed in Section 1.6 (Overview of NetView Dynamic TCP/IP Stack Discovery) you should define the SNMP community name and allow NetView to discover the local stack information dynamically.

2.3 Connection Monitoring and Thresholding Policy

Connection monitoring and thresholding is enabled with IPCONN policy definitions. IPCONN policy was introduced in NetView 5.1 and is enhanced for the *CONNMON* prototype.

IPCONN policy is also discussed in the *IBM Tivoli NetView for z/OS: Administration Reference* manual. The focus in this document will be on the enhancements to IPCONN policy for the *CONNMON* prototype.

A word of **caution** about the location of IPCONN policy statements: NetView provides sample IPCONN policy statements in FKXCFG01 that are commented-out. If you have implemented connection monitoring and thresholding using FKXECMON then you will need to move your IPCONN policy statements to CNMPOLCY for the *CONNMON* prototype.

```
IPCONN      proc_name,
            { SP=stack_policy_name, }
            { LPORT=(local_port,…,local_port), }
            { LPORT=(beg_port_num:end_port_num), }
            { MAXRECS=100 | nnn | -nnn, }
            { USEIPADDR=NO | YES,}
            THRESH=(byte_count,type,interval,actions),
            .
            .
            THRESH=(byte_count,type,interval,actions),
```

This is the same basic IPCONN policy definition used by FKXECMON with a few enhancements focusing on improving performance and ease of use:

- New keywords for the *CONNMON* prototype.
  - **LPORT** can be used to reduce the amount of port data searched by the *CONNMON* prototype.
  - **MAXRECS** can be used to limit the amount of connection data retrieved when the TCPCONN command is invoked.
  - **USEIPADDR** can be used to limit the amount of connection data retrieved by specifying YES to use only the IP address of the primary interface for the local TCP/IP stack when issuing the TCPCONN command. The TCPCONN command will be issued with the LADDR= parameter.

  All new keywords are **optional**.

- Changed keywords for the *CONNMON* prototype:
  - The SP=*stack_policy_name* is **optional** for the *CONNMON* prototype. If the SP keyword is not specified, the local TCP/IP stack will be used.

The supported IPCONN keyword values are:

- *proc_name*: The name of the start procedure associated with the application being monitored. For example, CICS01 or TCPIP.

  Wildcard values (*****) are supported.

- *stack_policy_name*: The name of the stack to use to monitor connections with this application. This should match a TCP390 policy definition name.

The *stack_policy_name* is optional for the *CONNMON* prototype. If not specified, CONNMON will use the local stack defined by TCP390 ... DOMAIN=LOCAL as the default stack.

- *local_port*: Optional, one or more ports the application will use. If specified, this will reduce the amount of port data interrogated by the monitoring routine, *CONNMON*, reducing overall CPU cycles.
- *nnn*: Optional, number of connection records retrieved with the TCPCONN command. 100 is the default. This will retrieve the 100 oldest matching (based on TCPCONN parameters) connections.

  This value can also be a negative number to retrieve the most recent *nnn* connections.

Note: The default value used by *CONNMON* differs from the TCPCONN command default. The TCPCONN command uses a default of negative 100 to retrieve connection data for the **most recent** 100 connections. For more details read the help for TCPCONN command.

- *interval*: Time interval, in hh:mm:ss format, defines the threshold for how long a session is allowed to be inactive.
- *byte_count*: The number of *bytes out* known to the TCP/IP stack with the ibmMvsTcpConnBytesOut MIB variable.

  byte_count can be an asterisk (*) to identify an idle connection threshold.
- *type*: The type of byte count threshold:
  - MINimum
  - MAXimum

  This parameter is ignored if the *byte_count* is an asterisk (*).
- *actions*: Actions to take:
  - NONE: Take no action: Do not issue any messages or drop connection. This is the default action.
  - NOTIFY: Invoke notification policy to notify the appropriate personnel.

  If running *CONNMON* without AON/TCP then only a message can be generated.
  - DROP: Break the connection, log the appropriate message to DSILOG.

  DROP and NOTIFY can be coded together.

Note: FKXECMON does not support the new keywords: LPORT, MAXRECS, and USEIPADDR.

2.3.1 LPORT Keyword

The LPORT keyword is not required on the IPCONN policy definition. It can be used to improve performance by specifying one or more local ports to use for the connection monitoring and thresholding function.

Specifying the LPORT keyword will reduce the amount of connection data retrieved when using the TCPCONN command.

LPORT can also be used to consolidate several existing IPCONN policy definitions for the same application (required by FKXECMON) into one IPCONN policy definition for *CONNMON*.

Examples:

- Single port number: LPORT=(23)
- List of port numbers: LPORT=(23,623,9090)
- Range of port numbers: LPORT=(4090:4999)

Specifying a range of port numbers can be useful for z/OS applications that do not bind to (open) a specific port number.

A **TCPCONN LADDR=*stack_IPaddr*,LPORT=*local_port*** command will be issued for every port specified with the LPORT keyword.

Caution: Specifying a large range of port numbers will result in processing overhead to build and issue the appropriate TCPCONN commands.

2.3.2 USEIPADDR Keyword

The USEIPADDR keyword is optional on the IPCONN policy definition. It can be used to limit the amount of connection data being retrieved with the TCPCONN command.

- **USEIPADDR=NO:** Scan all interfaces for connection data. Issue TCPCONN commands without LADDR parameter to retrieve connection data for all interfaces for the specified TCP/IP stack.

  This is the default value.

  This option is consistent with the functionality provided by FKXECMON.

- **USEIPADDR=YES:** Scan only the local stack IP address for connection data. Issue TCPCONN commands with the LADDR= parameter and a value of the local stack IP address.

This option potentially reduces the amount of connection data retrieved with the TCPCONN command.

It is anticipated this option will not be heavily utilized since connections with most TCP/IP stacks can be established using any interface defined to the stack.

2.3.3 IPCONN Example 1

This example will illustrate the basic IPCONN policy with the use of the LPORT USEIPADDR keywords:

```
IPCONN TCPIP*,
       SP=TIVED1,
       THRESH=(*,MIN,00:04:00,NOTIFY),
       THRESH=(2147,MIN,00:00:30,NOTIFY),
       USEIPDADDR=NO,
       LPORT=(23)
```

This will monitor local port 23 on stack TIVED1 for connections with any application name that begins with the character string of TCPIP.

If the session is idle for four minutes, regardless of the number of bytes sent, consult the NOTIFY policy to notify an operator.

If the session has had less than 2147 bytes received during the previous 30 seconds, consult the NOTIFY policy to notify an operator.

The timer will be scheduled to run using the interval defined by CONNMONINTVL in CNMSTYLE.

2.3.4 IPCONN Example 2

This example illustrates use of the default stack:

```
IPCONN FTP*,
       THRESH=(*,MIN,00:04:00,NOTIFY),
       THRESH=(2147,MIN,00:00:30,NOTIFY)
```

This will monitor connections to any local port opened by an application whose job name begins with the character string of FTP. Since no SP= keyword is defined, the local TCP/IP stack (TCP390 DOMAIN=LOCAL) will be used as a default.

If the session is idle for four minutes, regardless of the number of bytes sent, consult the NOTIFY policy to notify an operator.

If the session has had less than 2147 bytes received during the previous 30 seconds, consult the NOTIFY policy to notify an operator.

The timer will be scheduled to run using the interval defined by CONNMONINTVL in CNMSTYLE.

2.3.5 IPCONN Example 3

This example illustrates the use of the default stack plus a range or ports:

```
IPCONN MYAPP*,
       LPORT=(9010:9019),
       USEIPADDR=YES,
       THRESH=(*,MIN,00:04:00,NOTIFY),
       THRESH=(2147,MIN,00:00:30,NOTIFY)
```

This will monitor connections to local ports 9010 through 9019 (opened by an application whose job name begins with the character string of MYAPP) and the IP address of the primary interface for the local stack only. Since no SP= keyword is defined, the local TCP/IP stack will be used.

If the session is idle for four minutes, regardless of the number of bytes sent, consult the NOTIFY policy to notify an operator.

If the session has had less than 2147 bytes received during the previous 30 seconds, consult the NOTIFY policy to notify an operator.

The timer will be scheduled to run using the interval defined by CONNMONINTVL in CNMSTYLE.

2.4 Notification Policy

2.4.1 Notification with AON

If you are running AON you have access to the Notification policy. You should see a NOTIFY policy definition statement in FKXCFG01 similar to:

```
NOTIFY IPCONN,ALERT=NO,MSG=YES,DDF=NO,INFORM=NO
```

Modify the NOTIFY IPCONN statement as appropriate. For example, define ALERT=YES if you want NPDA alerts generated from connection monitoring and thresholding messages.

Change INFORM from NO to the name of your Inform policy to generate e-mail notifications.

2.4.2 Notification without AON

If you are not running AON/TCP then the NOTIFY policy, if coded, is ignored and messages are generated from the *CONNMON* routine directly using the REXX SAY instruction. The message text will be the same as notification with AON but the message is incapable of being routed to support personnel in an e-mail.

2.5 ACTMON Policy

The ACTMON IPCONN policy definition is used by AON/TCP FKXECMON to schedule the connection monitoring timer. This definition is obsolete and is replaced by a CNMSTYLE definition with the CONNMON prototype.

Delete the ACTMON IPCONN policy definition on your system. It is typically found in DSIPARM member FKXCFG01.

2.6 AUTOOPS Policy

AON/TCP provides an AUTOOPS policy definition for connection monitoring and thresholding. This definition is obsolete and is replaced by a CNMSTYLE definition with the CONNMON prototype.

Delete the AUTOOPS CONNOPER,ID=AUTCMON policy definition on your system. It is typically found in DSIPARM member FKXCFG01.

If you are running AON/TCP you will see an EZL572I message and can ignore it:

```
EZL572I OPERATOR CGLOBALS NOT INITIALIZED - UNABLE TO ROUTE
COMMAND FKXECMON
```

This occurs when AON/TCP initializes. AON/TCP will detect the existence of the IPCONN policy definitions and attempt to schedule FKXECMON under the autotask defined by the AUTOOPS CONNOPER statement.

2.7 CNMSTYLE Customization

CNMSTYLE customization consists of two groupings:

- Enable and define TCPCONN parameters
- Enable and define *CONNMON* parameters

2.7.1 TCPCONN Definitions

TCPCONN requires definitions in CNMSTYLE.

1. Enable the TCPIPCOLLECT and TCPCONN towers:

   ```
   TOWER = *SA   *AON  *MSM  *Graphics  MVScmdMgt  NPDA *NLDM
   TCPIPCOLLECT *AMI *TARA  DVIPA

   TOWER.TCPIPCOLLECT = TCPCONN *PKTS
   ```

2. Define TCPCONN collection:

   ```
   INIT.TCPCONN = Yes
   function.autotask.TCPCONN.TCPIP = autotask_name
   ```

   Select a valid *autotask_name* for your environment.

3. Define TCPCONN.KEEP statements to filter connections to the loopback address and Web browser connections:

   ```
   * Do not keep loopback connection data
   TCPCONN.KEEP.TCPIP.A = NOT 127.0.0.1/*,*/*
   * Do not keep Web browser (port =80) connection data
   TCPCONN.KEEP.TCPIP.B = NOT */80,*/*
   * Keep data for all other connections
   TCPCONN.KEEP.TCPIP.Z = */*,*/*
   ```

   Define additional TCPCONN.KEEP statements for any other connections to be filtered.

2.7.2 Definitions for CONNMON Prototype

Several customization parameters can be found in DSIPARM member **CONNSTYL** which is available as part of the *CONNMON* prototype.

You will need to add this statement to your CNMSTYLE:


```
%INCLUDE CONNSTYL
```

This section will discuss each of the parameters in further detail.

2.7.2.1 Define CONNMON tower

By default, the CONNMON prototype assumes you are also running AON/TCP. If true then you can skip this definition.

If you are **not** running AON/TCP you need to define CONNMON as a subtower of the TCPCONN tower. Uncomment the TOWER.TCPIPCOLLECT.TCPCONN statement in CONNSTYL:

```
TOWER.TCPIPCOLLECT.TCPCONN = CONNMON
```

2.7.2.2 Define the autotask for monitoring

The default autotask used for connection monitoring is AUTCMON. Modify this statement to a valid autotask name in your environment:

```
function.autotask.CONNMON = OPER5
```

AUTCMON is the autotask also used by FKXECMON.

2.7.2.3 Define default interval for scheduling monitoring timer

The default interval for scheduling the *CONNMON* routine is one minute.

```
COMMON.CNMSTYLE.CONNMONINTVL = 00:01:00
```

2.7.2.4 Define name of in storage NetView file to store port data

Port data will be collected by the GETPORTS REXX EXEC and stored locally in NetView storage under the name defined by the CONNMONFILE variable:

```
COMMON.CNMSTYLE.CONNMONFILE = CONNMON | file_name
```

The default file name will be CONNMON.

2.7.2.5 Define TCP/IP port detection and logging

Define TCP/IP port detection and logging.

```
COMMON.CNMSTYLE.CONNMONPORTS = { STATIC | DYNAMIC |
DYNAMIC,hh:mm }
```

Each of these options can affect your overall performance. For example, STATIC is the best performance option since it will retrieve port data only once. However, if your applications open ports dynamically you may need to code DYNAMIC to query port data each time connection monitoring (*CONNMON*) is invoked. This will require additional CPU cycles to retrieve the current port data.

- **STATIC:** Retrieve TCP/IP port data once (initially) and store in NetView storage.

  This is the default value.

- **DYNAMIC:** Retrieve TCP/IP port data, each time monitoring is invoked, from the ibmTcpipMvsTcpListenerTable MIB table. No data is kept in storage.

  This should be used when active ports are dynamic.

- **DYNAMIC,hh:mm:** Retrieve TCP/IP port data periodically (for example, hourly) and store port data in NetView storage.

  This will schedule an additional timer based on the interval (hh:mm) to retrieve TCP/IP port data from the ibmTcpipMvsTcpListenerTable MIB and keep it in NetView storage.

Defining DYNAMIC with a time interval (hh:mm) will schedule the GETPORTS command based on the time interval. When *CONNMON* is invoked it will query the port data stored in the CONNMON file independent of the GETPORTS timer.

2.7.3 Command Definitions

DSIPARM member **CONNCMD** contains the necessary NetView command definitions for the CONNMON and GETPORTS REXX EXECs.

2.7.4 DSIAUTB Definitions

DSIPARM member **CONNAUTB** contains the necessary NetView command authorization definitions for the CONNMON and GETPORTS REXX EXECs.

2.8 Communications Server for z/OS Customization

To support use of the TCPCONN command Communications Server for z/OS customization may be required to enable the NMI:

- Add this statement to your TCP/IP profile member:

NETMONITOR TCPCONNSERVICE

- Check value of MINLIFETIME:

  Defined as: The minimum connection lifetime, specified in seconds, for connections reported by the TCP connection information server.

  The default is 3 seconds; you may want to change to be higher.

2.9 NetView SNMP Customization

SNMP MIBs are retrieved using the NetView SNMP command. There are several tasks for customizing NetView SNMP:

- Run CNMSJ032 to copy MIB source files to the appropriate USS directories.

  CNMSJ032 will perform tasks for other NetView functions. Review the commentary in the JCL before running the job.

  If CNMSJ032 ends with a return code of four you should check for existing MIB source files in /etc/netview/mibs directory.

- Review CNMSTYLE definitions:

  For example, if you modified the USS directories in CNMSJ032 then you may need to modify the MIBPATH definition:

  ```
  COMMON.CNMSNMP.MIBPATH = /usr/lpp/netview/v5r2/mibs:/etc/
  netview/mibs
  ```

- Customize OSNMPD:

  Modify your TCP/IP profile to start OSNMPD.

  Define SNMP community names.

2.10 NetView Security Considerations

Both CONNMON and GETPORTS routines are coded to bypass further security checks (to be consistent with FKXECMON) once they are invoked. This requires definitions for DSIAUTB that are provided in member CONNAUTB. This approach reduces the number of command authorization checks, resulting in less overhead.

The autotask running the CONNMON and GETPORTS routines will need to have sufficient authority to invoke them. One method is to permit the autotask access within the NetView Command Authorization Table (CAT).

The autotask is defined in CONNSTYL:

function.autotask.CONNMON = ***OPER5***

By default, commands not protected in the sample NetView CAT table are permitted.

If your organization follows the same philosophy then the CONNMON and GETPORTS routines will be permitted. If your organization protects all commands then you will need to permit access to the CONNMON and GETPORTS routines.

2.11 CONNMON Timer

When connection monitoring and thresholding is properly configured you can issue a LIST TIMER command to display the timer information. It will look similar to:

LIST TIMER=FKXECMON,OP=ALL

```
DISPLAY OF OUTSTANDING TIMER REQUESTS
TYPE: AFTER  TIME: 10/31/06 11:31:43
 COMMAND: FKXECMON TIVED1
  OP: OPER5     (OPER5   )   ID: FKXECMON
TIMEFMSG: NO    GMT
1 TIMER ELEMENT(S) FOUND FOR ALL
END OF DISPLAY
```

In this example, the autotask used for the monitoring is OPER5. The command is scheduled as an AFTER timer based on the interval defined with the COMMON.CNMSTYLE.CONNMONINTVL statement in CONNSTYL.

The command scheduled will be FKXECMON even when using the *CONNMON* prototype due to the way the CONNMON EXEC is defined (command synonym of FKXECMON).

3 Enable the CONNMON Prototype

This section is intended to be used as a checklist to complete the enablement of the *CONNMON* prototype. Topics discussed will be the necessary and optional tasks for enabling of the *CONNMON* prototype. The discussion of several tasks will be dependent on whether or not you have AON/TCP active.

Each CNMSTYLE statement and IPCONN policy definition shown in this section was discussed in detail in earlier sections of this document.

3.1 Download the CONNMON Prototype Package

1. Access NetView Downloads Web page:

   `http://www.ibm.com/software/sysmgmt/products/support/IBMTivoliNetViewforzOS.html`

2. Download the *CONNMON* package.

3. Unzip CONNMON.zip.

   This will result in several parts being created that you must FTP to the systems where you want connection monitoring and thresholding implemented.

4. Print CONNMON_White_Paper.PDF.

3.2 Copy NetView Files

The files included in the CONNMON prototype affect NetView DSICLD and DSIPARM:

1. FTP REXX EXECs to a NetView DSICLD data set:
   - GETPORTS.REX
   - CONNMON.REX

2. FTP definition members to a NetView DSIPARM data set:
   - CONNSTYL.PRM
   - CONNCMD.PRM
   - CONNAUTB.PRM
   - CONNMON.PRM

Choose data sets that are concatenated **before** the NetView-supplied DSICLD and DSIPARM data sets.

3.3 Edit DSIPARM Members

3.3.1 Edit CNMSTYLE Members

Several style sheet statements are provided with the CONNMON prototype. These instructions reference NetView member CNMSTGEN as an example.

- Edit CNMSTYLE (CNMSTGEN or CNMSTUSR):
  - Add **%INCLUDE CONNSTYL** to include the CONNMON definitions
- If you are not running AON/TCP, enable the CONNMON tower as a subtower of TCPCONN. If you are running AON/TCP you can skip this step.
  - Uncomment this statement in CONNSTYL

    TOWER.TCPIPCOLLECT.TCPCONN = CONNMON

    Note: If you are running AON/TCP it will schedule FKXECMON under the AUTCMON autotask by default.
- Review member CONNSTYL and determine if customization is necessary.
  - COMMON.CNMSTYLE.CONNMONINTVL
  - COMMON.CNMSTYLE.CONNMONPORTS
  - COMMON.CNMSTYLE.CONNMONFILE
- Customize NetView for TCPCONN.

If you are running AON/TCP you should have already customized the FKXCFG01 policy member to use AUTCMON as the autotask for the monitoring and thresholding code.

If you are *not* running AON/TCP you will need to define the autotask in CNMSTYLE:

- Select an autotask to use for the connection monitoring and thresholding code.

  function.autotask.CONNMON = ***OPER5***

  This example will use OPER5 as the CONNMON autotask. Choose a valid task name for your environment.

- Select a time interval on the supplied *initcmd* statement to schedule the initial invocation to FKXECMON:

  AUTOTASK.?CONNMON.InitCmd = AFTER **00:10:00**,FKXECMON

  This example uses a ten minute delay to allow sufficient time for NetView initialization to occur. You can choose a lower time interval. The time interval chosen is dependent on the amount of time it takes for NetView to dynamically discover the local TCP/IP stack.

3.3.2 Edit NetView Command Definitions (CNMCMD)

CNMCMD defines commands to NetView.

- Define *CONNMON* commands in CNMCMDU:
  - Add **%INCLUDE CONNCMD** to define the CONNMON prototype commands

If you are running AON/TCP you may see a DSI234I message related to the command definition for CONNMON to define it with a command synonym of FKXECMON:

```
DSI234I DUPLICATE COMMAND 'FKXECMON' DETECTED
```

You can ignore this message.

3.3.3 Command Authorization

- Define *CONNMON* commands in DSIAUTBU for authorization:
  - Add **%INCLUDE CONNAUTB**

3.3.4 Connection Monitoring and Thresholding Policy

Define IPCONN policy definitions to enable connection monitoring and thresholding function. You should code the IPCONN policy definitions in CNMPOLCY to avoid confusion with the samples provided by AON/TCP. The AON/TCP samples are commented out by default and should be left commented out.

If you are running AON/TCP:

- Remove (or comment out) any existing IPCONN policy definitions in FKXCFG01 or any other policy definition member.

This will ensure that you use the enhanced IPCONN policy definitions used by the *CONNMON* prototype. The *CONNMON* prototype supports the existing NetView IPCONN policy definitions but you will be unable to take further advantage of the performance and ease of use enhancements provided by the *CONNMON* prototype.

4 NetView TCPCONN Command

4.1 Overview of TCPCONN Usage

Information on the TCPCONN command can be found by accessing the NetView help (HELP TCPCONN) or by reading the *IBM Tivoli NetView for z/OS: Command Reference Volume 1* manual.

CONNMON uses TCPCONN command with these parameters

- **QUERY**

  Queries connection records matching the input criteria.

- **LADDR=***local_stack_ipaddr*

  Specifies the IP address of the local stack for the QUERY.

- **LPORT=***local_port*

  Specifies the local port number, in decimal format.

- **MAXRECS=***max_conn_records*

  Specifies the maximum number of connection records to return from TCPCONN QUERY.

  A positive value specifies the set of connection records ending with the oldest matching connection.

  A negative value specifies the set of records starting with the most recent matching connection.

  The default value is 100 for *CONNMON*.

Note: The default value used by *CONNMON* differs from the TCPCONN command default. The TCPCONN command uses a default of negative 100 to retrieve connection data for the **most recent** 100 connections. For more details read the help for TCPCONN command.

Default values are used for all other TCPCONN parameters.

Reminder: TCPCONN exploits an NMI with Communications Server for z/OS. Use of TCPCONN by itself should improve the performance of the connection monitoring and thresholding function.

4.2 TCPCONN Example

To retrieve connection data for a TCP/IP stack with local address of 10.44.15.200 and local port 23, the TCPCONN command would look like:

TCPCONN QUERY LPORT=23 LADDR=10.44.15.200

An example response would look similar to:

```
BNH772I NUMBER OF CONNECTIONS: 1 , MISSED BUFFERS: 0
TCPIP    10.44.15.200          23          10.44.15.200          1028       ×*ÌÅ>*è*
```

The response will contain six parameters:

- TCPNAME (TCPIP)

  Not used by *CONNMON*

- Local IP address of the stack (10.44.15.200)
- Local IP port on the stack (23)
- Remote IP address (10.44.15.200)
- Remote IP port (1028)
- Time stamp, in binary format (×*ÌÅ>*è*)

  Not used by *CONNMON*

After issuing a TCPCONN command, CONNMON has a list of four-tuples that represent the TCP/IP connection matching the IPCONN criteria. Using the four-tuples CONNMON will issue an SNMP GET command to retrieve the detailed connection data needed to perform its thresholding.

Note: TCPCONN supports IPv6. The response may contain IPv6 data. *CONNMON* will filter the IPv6 data.

5 TCP/IP MIB Connection Data

Once CONNMON has determined the four-tuple representation of a connection it will use the four-tuple to retrieve MIB variable data from the ibmTcpipMvsTcpConnTable MIB table.

The MIB variables used are:

- **ibmMvsTcpConnBytesIn**
- **ibmMvsTcpConnBytesOut**

  The number of bytes for the connection. This is compared to the maximum or minimum byte count threshold of the IPCONN policy definition.

- **ibmMvsTcpConnSndWnd**

  Used to determine if the connection is hung.

- **ibmMvsTcpConnResourceName**

  Contains the application name for the connection. For example, SMTP. This is compared to the IPCONN policy name if no LPORT parameter is specified. If LPORT is specified the resource name is ignored.

- **ibmMvsTcpConnResourceId**

  Decimal value that represents the connection. The resource ID is used in messages.

- **ibmMvsTcpConnLastActivity**

  When the byte count for the threshold is an asterisk (*) the last activity data is used to determine if the connection is idle.

  When the byte count for the threshold is a numeric the last activity data is compared to the interval specified on the THRESH keyword of the IPCONN policy definition.

Using the four-tuple from the TCPCONN example, 10.44.15.200.23 representing the local stack and local port and 10.44.15.200.1028 representing the remote client and the remote port, an SNMP GET command will be issued for the MIB details:

- ibmMvsTcpConnBytesIn.10.44.15.200.23.10.44.15.200.1028
- ibmMvsTcpConnBytesOut.10.44.15.200.23.10.44.15.200.1028
- ibmMvsTcpConnSndWnd.10.44.15.200.23.10.44.15.200.1028

- ibmMvsTcpConnResourceName.10.44.15.200.23.10.44.15.200.1028
- ibmMvsTcpConnResourceId.10.44.15.200.23.10.44.15.200.1028
- ibmMvsTcpConnLastActivity.10.44.15.200.23.10.44.15.200.1028

6 Retrieving Port Data

The GETPORTS REXX EXEC uses SNMP MIB data to determine the active ports for the local TCP/IP stack. GETPORTS can be scheduled on a timer basis depending upon the CONNMONPORTS definition in CONNSTYL:

```
COMMON.CNMSTYLE.CONNMONPORTS = ( STATIC | DYNAMIC |
DYNAMIC,hh:mm:ss )
```

STATIC: This is the default value. GETPORTS will be called once during NetView initialization to collect port data and store for later use by the CONNMON EXEC. This option provides the best overall performance and should be used if the TCP/IP ports you are monitoring do not change.

DYNAMIC: GETPORTS will be called from the CONNMON EXEC every time CONNMON is driven to dynamically collect data for the currently active ports. This option should be used if the TCP/IP ports you are monitoring change frequently. This option is also consistent with the functionality provided by FKXECMON.

DYNAMIC,*hh:mm:ss*: GETPORTS will be scheduled on a timer basis using the *hh:mm:ss* interval specified. GETPORTS will collect the port data and store for later use by the CONNMON EXEC. This option can be used if the TCP/IP ports you are monitoring change but not frequently.

6.1 TCP/IP MIB Port Data

Local TCP/IP port data is retrieved from the ibmTcpipMvsTcpListenerTable MIB table. There are two MIB variables within the ibmTcpipMvsTcpListenerTable MIB table that are used:

- **ibmMvsTcpListenerResourceName**

  Contains the application name (job name) of the listener on the local port. For example:

  – ibmMvsTcpListenerResourceName.16 = TCPIP

- **ibmMvsTcpListenerLocalPort**

  Contains the local port number for the listener. For example:

  – ibmMvsTcpListenerLocalPort.16 = 23

The GETPORTS EXEC will correlate the two MIB variables such that the application known as TCPIP is listening on local port 23.

GETPORTS will update the list of ports in the file in NetView storage (defined by COMMON.CNMSTYLE.CONNMONFILE = CONNMON in CONNSTYL). You can view the list of ports by browsing the file.

BROWSE CONNMON

```
 NETVIEW.BRWS ------ BROWSE CONNMON  (DSIPARM ) --- LINE 00000 TO 00017 OF 00017
                                                          SCROLL ==> CSR
----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----
 ****************************** TOP OF DATA ******************** DATASET:  0
*** 10/26/06 09:38:25
1024 TCPIP
23 TCPIP
512 RXSERVE
514 RXSERVE
1026 OSNMPD
21 FTPD1
10007 BPXOINIT
80 AUTONETV
4022 AUTONETV
9999 AUTONETV
4020 AUTONETV
25 SMTP
612 INETD4
513 INETD4
614 INETD4
623 INETD4
 ***************************** BOTTOM OF DATA ******************************
```

GETPORTS has detected 16 active ports:

- TCPIP has opened two ports
- RXSERVE has opened two ports
- OSNMPD has opened one port
- FTPD1 has opened one port
- BPXOINIT has opened one port
- AUTONETV has opened four ports
- SMTP has opened one port
- INETD4 has opened four ports

6.2 Correlate GETPORTS Data with IPCONN Policy

This section will attempt to tie the IPCONN policy definition with the port data retrieved by GETPORTS.

The example IPCONN policy definition used in this section will be:

```
IPCONN TCPIP*,SP=TIVED1,
   THRESH=(*,MIN,00:04:00,NOTIFY),
   THRESH=(2147,MIN,00:00:30,NOTIFY)
```

6.2.1 IPCONN Policy without LPORT

If you specify IPCONN TCPIP* without the LPORT keyword to monitor connections with TCPIP, CONNMON will retrieve all ports from the CONNMON file and build a list of ports with a matching resource name. For example, ports 23 and 1024 will be used from the 16 active ports in the GETPORTS example.

That translates into two TCPCONN commands being issued:

- TCPCONN LPORT=23 LADDR=*nnn.nnn.nnn.nn*n
- TCPCONN LPORT=1024 LADDR=*nnn.nnn.nnn.nnn*

This is a case where additional overhead will occur by retrieving the connection data for port 1024. To avoid that code LPORT=(23) on the IPCONN TCPIP* policy.

Note: The LADDR parameter of the TCPCONN command is shown here for completeness. Its value is not relevant to this discussion so it is shown as *nnn.nnn.nnn.nnn*.

6.2.2 IPCONN Policy with LPORT

Using the LPORT keyword will reduce wasted CPU cycles if you know what the local port is.

By adding LPORT=(23) to the existing IPCONN TCPIP* policy definition the CONNMON EXEC will look for connections only to local port 23 and ignore any local port data saved by the GETPORTS EXEC.

This is the option for optimum performance.

6.3 GETPORTS Timer

If you specify that you want ports to be discovered dynamically for COMMON.CNMSTYLE.CONNMONPORTS and provide an interval, GETPORTS will be scheduled based on the interval provided with a timer ID of GETPORTS.

The NetView LIST TIMER command can be used to display the timer information:

LIST TIMER=GETPORTS,OP=ALL

```
DISPLAY OF OUTSTANDING TIMER REQUESTS
TYPE: AFTER   TIME: 10/26/06 10:00:27
  COMMAND:  GETPORTS TIVED1
  OP: AUTCMON  (AUTCMON )   ID: GETPORTS
TIMEFMSG: NO    GMT
1 TIMER ELEMENT(S) FOUND FOR ALL
END OF DISPLAY
```

In this example, GETPORTS is scheduled as an AFTER timer on the AUTCMON task.

7 Discussion of Customer Modifiable Variables

Each REXX EXEC supplied with the *CONNMON* prototype package, CONNMON and GETPORTS, contains a subroutine (**Customer_Vars**) that defines default values for variables used within the EXEC. The defaults are used when policy definitions or CONNSTYL statements are not defined properly.

In general, you should specify as many keywords and parameters as possible on the policy definitions or CONNSTYL statements.

7.1 CONNMON Customer_Vars

```
/* Define a default SNMP community name in case it is not defined on */
/* a stack policy definition (TCP390). The preferred method is to use */
/* the COMMUNITYNAME keyword on your TCP390 definition.            */
Default_Cname    = 'publicv2c'

/* Define a default file name to contain port data. The preferred    */
/* method is to define the file name in CNMSTYLE with:              */
/* COMMON.CNMSTYLE.CONNMONFILE = CONNMON                            */
Default_filename = 'CONNMON'

/* Define a default time interval used to schedule this routine.     */
/* The preferred method to define the interval is in CNMSTYLE with:  */
/* COMMON.CNMSTYLE.CONNMONINTVL = 00:10:00                          */
Default_interval = '00:10:00'

/* Define a default value for the MAXRECS parm of the TCPCONN cmd.   */
/* The preferred method to define MAXRECS on the IPCONN policy       */
/* definition.                                                      */
Default_MaxRecs = 100

/* Define a default value for the USEIPADDR keyword of IPCONN    */
/* The preferred method to define USEIPADDR on the IPCONN policy     */
/* policy definition.                                               */
/* definition.                                                      */
Default_UseIpAddr = 'NO'
```

7.2 GETPORTS Customer_Vars

```
/* MaxPorts value is used on SNMP GETBULK command: -Cr<MaxPorts>.    */
/* By default this setting will retrieve data for "up to" 100 ports. */
/* Adjust higher if you are running more than 100 ports.            */
```

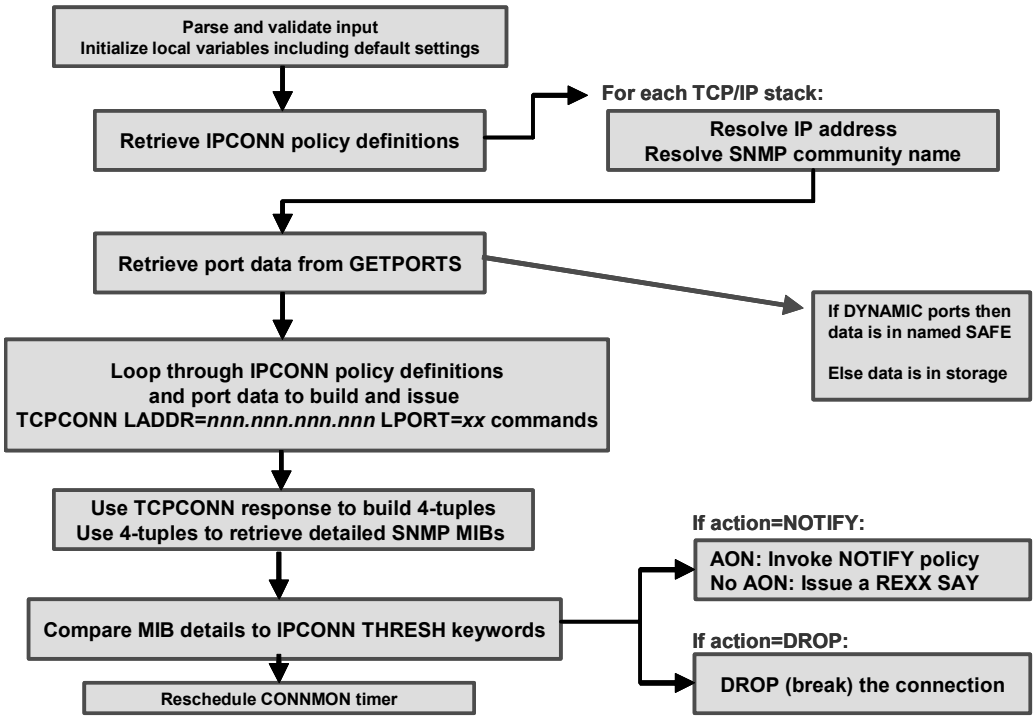
```
/* Adjust lower if you are running less than 100 ports. Otherwise,   */
/* extraneous data will be retrieved with the GETBULK and ignored in */
/* the GETBULK PIPE. Extraneous data means additional cycles.        */
MaxPorts = 100

/* Define a default SNMP community name in case it is not defined on */
/* a stack policy definition (TCP390). The preferred method is to use */
/* the COMMUNITYNAME keyword on your TCP390 definition.              */
Default_Cname    = 'publicv2c'
```


8 CONNMON Prototype Design

8.1 CONNMON EXEC

This section provides an overall flow of the logic in the CONNMON EXEC.



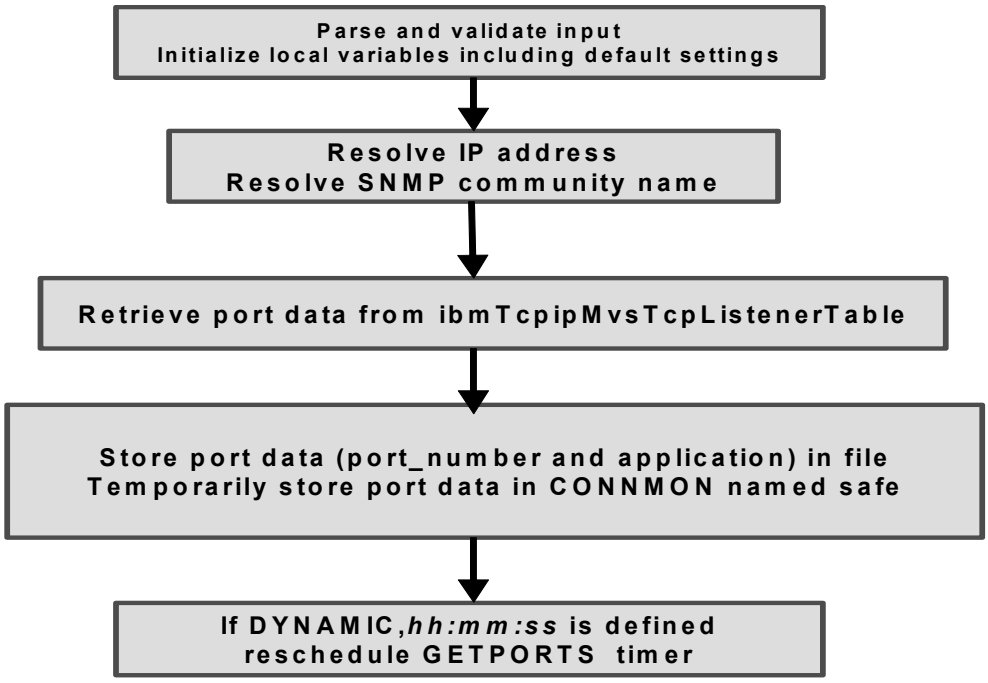
The primary changes for CONNMON (compared to FKXECMON) are related to the processes to:

- Retrieve the IPCONN policy definitions
- Retrieve the port data
- Use the port data in conjunction with the IPCONN policy to build a list of TCPCONN commands with a local TCP/IP stack address and port
- The TCPCONN response is then formatted into a list of four-tuples (4-tuples) that are used to retrieve the detailed SNMP MIB variables such as ibmMvsTcpConnLastActivity

The logic to compare the MIB details and take the appropriate actions (NOTIFY and DROP) is relatively unchanged.

8.2 GETPORTS EXEC

This section provides an overall flow of the logic in the GETPORTS EXEC.



GETPORTS is new with the *CONNMON* prototype. It will retrieve port data from the ibmTcpipMvsTcpListenerTable MIB table and store the data in the CONNMON file (default file name is CONNMON) as well as a named safe (using the same name as the in-storage file, CONNMON).

If a time interval is specified on the COMMON.CNMSTYLE.CONNMONPORTS statement in CONNSTYL then GETPORTS will reschedule itself.

The in-storage file name (and named safe) is controlled by the COMMON.CNMSTYLE.CONNMONFILE statement in CONNSTYL.

9 Performance Tests

9.1 Overview of Tests Run

Several sample tests were run to validate the performance of the *CONNMON* prototype versus FKXECMON. The results are discussed in this section of the document.

Overall parameters for the measurements:

- All tests were performed on a second level z/OS system.
- All tests were performed with all required REXX EXECs loaded into NetView storage: FKXECMON, CONNMON, and GETPORTS.
- All tests were run on an autotask that was dedicated to run only these measurements. No other workload was scheduled on the autotask.
- All tests were run using **five TN3270** connections into local port 23 and **one FTP** connection. Each TN3270 connection was logged on to TSO.
- Only IPCONN policy for TCPIP connections was defined. There was no IPCONN policy for FTP connections defined.
- All connections were established in an identical manner. All tests were run using an IDLE threshold specification.
- DROP was not used during these tests to keep the sessions active and IDLE.
- All NetView timers were purged to prevent any interference.
- No entry, exit, or program tracing was allowed.
- A null Command Authorization Table was loaded to eliminate any overhead due to security checking.
- NOTIFY policy determines the types of notifications to occur for a given event. All notifications were disabled when running FKXECMON:

  NOTIFY IPCONN,ALERT=NO,**MSG=NO**,DDF=NO,INFORM=NO

No tests were performed for:

- Missing SP= keyword and using the default local stack
- USEIPADDR=YES

The z/OS system used for these measurements is not a production system and is not overly utilized as a result. Furthermore, it may not be a well tuned z/OS system.

There were two measurements taken for each test. A test EXEC was written to invoke each connection monitoring routine (FKXECMON and COMMON) repeatedly, inside a program loop. The test EXEC called the respective connection monitoring routine 200 times for each specific test run.

- First, the REXX TIME() function was used to evaluate the overall elapsed time it took a connection monitoring routine to complete.

  The elapsed time represents the actual clock time used by the task within the given time period.

  An average elapsed time was then derived for each test.

- Second, the NetView TASKUTIL command was used to evaluate the NetView CPU time used by the task running the connection monitoring routine. TASKUTIL was invoked before the program loop and afterwards.

  The NetView CPU time represents the amount of CPU cycles used by the task within the given time period.

  An average NetView CPU time was then derived for each test.

The unit of measurement is seconds.

9.2 FKXECMON Base Measurements

To establish a baseline, the base measurements for FKXECMON will be taken using IPCONN policy supported by FKXECMON:

```
IPCONN TCPIP*,SP=TIVED1,
      THRESH=(*,MIN,00:04:00,NOTIFY),
      THRESH=(2147,MIN,00:00:30,NOTIFY),
      ACTMON=IPCONN
```

FKXECMON will be used in this test to establish a baseline for comparisons with the *CONNMON* prototype.

Test results:

TESTRUN 200

Total Clock Time was: 421.431374

Average Clock Time was: 2.10715687

Total NetView TASKUTIL CPU Time was: 185.49

Average NetView TASKUTIL CPU Time was: **0.92745**

Summary of test results:

This test was performed solely to establish a baseline for the remaining tests.

This baseline is representative of a controlled test environment. In a production environment there would be tens of thousands of connections retrieved by FKXECMON and this baseline would actually be much worse.

9.3 Test 1: Basic NetView IPCONN Policy

The first test will use identical IPCONN policy as the FKXECMON baseline, with no enhancements for the *CONNMON* prototype. This will establish a *CONNMON* prototype baseline. The remaining tests will measure the effects of the additional enhancements provided by the *CONNMON* prototype.

The policy is consistent with FKXECMON in NetView 5.2 including the retrieval of port data dynamically:

```
IPCONN TCPIP*,SP=TIVED1,
      THRESH=(*,MIN,00:04:00,NOTIFY),
      THRESH=(2147,MIN,00:00:30,NOTIFY)

COMMON.CNMSTYLE.CONNMONPORTS = (DYNAMIC)
```

Test results:

TESTRUN 200

Total Clock Time was: 658.153965

Average Clock Time was: 3.29076983

Total NetView TASKUTIL CPU Time was: 47.86

Average NetView TASKUTIL CPU Time was: **0.2393**

Summary of test results:

Use of the TCPCONN command to gather connection data represents, on average, a .68815 second improvement from the previous test. This translates into a **huge 74.2% improvement** over the FKXECMON baseline measurement.

If this was a production system the improvement would be even greater because the FKXECMON baseline would be much worse.

9.4 Test 2: IPCONN with LPORT, no Port Data in Storage

The second test will add the LPORT keyword to the IPCONN policy for one local port. The retrieval of port data will remain dynamic. This will test the benefits of using the LPORT keyword on the overall connection monitoring and thresholding process.

IPCONN with LPORT coded and dynamic port data

```
IPCONN TCPIP*,SP=TIVED1,
      THRESH=(*,MIN,00:04:00,NOTIFY),
      THRESH=(2147,MIN,00:00:30,NOTIFY),
      LPORT=(23)

COMMON.CNMSTYLE.CONNMONPORTS = (DYNAMIC)
```

Test results:

TESTRUN 200

Total Clock Time was: 603.727995

Average Clock Time was: 3.01863998

Total NetView TASKUTIL CPU Time was: 41.99

Average NetView TASKUTIL CPU Time was: **0.20995**

Summary of test results:

Use of the LPORT keyword on the IPCONN policy definition represents, on average, a .02935 second improvement from the previous test. This translates into a 12.3% improvement.

In the test environment TCPIP has two local ports. By specifying one, LPORT=(23), the amount of data was reduced affecting the overall time by 12%.

This example will vary based on the application involved. If you have an application with several ports open but only one or two of the ports is used for connections then you should code the LPORT keyword.

9.5 Test 3: IPCONN with LPORT, Port Data in Storage

The third test will add the use of static port data and use the IPCONN policy for a single local port. This will test the benefits of using the LPORT keyword and static port data on the overall connection monitoring and thresholding process.

IPCONN with LPORT coded and static port data in storage

```
IPCONN TCPIP*,SP=TIVED1,
      THRESH=(*,MIN,00:04:00,NOTIFY),
      THRESH=(2147,MIN,00:00:30,NOTIFY),
      LPORT=(23)

COMMON.CNMSTYLE.CONNMONPORTS = (STATIC)
```

Test results:

TESTRUN 200

Total Clock Time was: 499.671340

Average Clock Time was: 2.4983567

Total NetView TASKUTIL CPU Time was: 31.18

Average NetView TASKUTIL CPU Time was: **0.1559**

Summary of test results:

Use of static port data represents, on average, a .05405 second improvement from the previous test. This translates into a **25.7%** improvement.

As anticipated, this is a measurable improvement.

10 Messages

10.1 Messages Issued by CONNMON EXEC

The messages issued by the CONNMON routine are the same as those issued by FKXECMON.

In this example FKX101I message the connection between the client 99.44.15.209:1027 and the TIVED1 stack 10.44.15.200:23 has been idle longer than the defined threshold on the IPCONN TCPIP* policy definition:

```
FKX101I IDLE TIME THRESHOLD EXCEEDED FOR CONNECTION 2744/
0000000AB8 BETWEEN 10.44.15.200:23 AND 99.44.15.209:1027.
ACTION=NOTIFY SP=TIVED1 POLICY=TCPIP*.
```

The session identifier is 2744 (decimal) or 0000000AB8 (hex). The action defined in the policy is NOTIFY which causes the FKX101I message to be issued.

You can use the NetView HELP command to display information pertaining to each message.

```
FKX101I IDLE TIME THRESHOLD EXCEEDED FOR CONNECTION
connection_ID BETWEEN local_addr_and_port AND
remote_addr_and_port. ACTION=NOTIFY SP=local_stack_name
POLICY=IPCONN_policy.

FKX102I IDLE TIME THRESHOLD EXCEEDED FOR CONNECTION
connection_ID BETWEEN local_addr_and_port AND
remote_addr_and_port. ACTION=DROP SP=local_stack_name
POLICY=IPCONN_policy.

FKX104I MINIMUM BYTES THRESHOLD EXCEEDED FOR CONNECTION
connection_ID BETWEEN local_addr_and_port AND
remote_addr_and_port. ACTION=NOTIFY SP=local_stack_name
POLICY=IPCONN_policy.

FKX105I MINIMUM BYTES THRESHOLD EXCEEDED FOR CONNECTION
connection_ID BETWEEN local_addr_and_port AND
remote_addr_and_port. ACTION=DROP SP=local_stack_name
POLICY=IPCONN_policy.

FKX107I MAXIMUM BYTES THRESHOLD EXCEEDED FOR CONNECTION
connection_ID BETWEEN local_addr_and_port AND
remote_addr_and_port. ACTION=NOTIFY SP=local_stack_name
POLICY=IPCONN_policy.
```



```
FKX108I MAXIMUM BYTES THRESHOLD EXCEEDED FOR CONNECTION
connection_ID BETWEEN local_addr_and_port AND
remote_addr_and_port. ACTION=DROP SP=local_stack_name
POLICY=IPCONN_policy.
```

10.2 Messages Issued by GETPORTS EXEC

GETPORTS is new and does not issue messages recognized with a NetView product prefix such as FKX.

GETPORTS will use a REXX SAY instruction to display an error message relating to the type of data collection (STATIC or DYNAMIC):

CONNMON - Invalid collection type definition:' *Type*

GETPORTS encountered an error with the CNMSTYLE.CONNMONPORTS definition statement. It did not contain a type of STATIC or DYNAMIC.

A default collection type of STATIC will be used.

11 Common Errors

This section summarizes some common errors and their possible causes.

11.1 No Port Data

If the GETPORTS routine is not scheduled issue a BROWSE CONNMON command to verify that you have the sample file properly defined. You should see the NetView browse panel with one line:

```
* sample file for CONNMON prototype - do not delete
```

If you see a different display, check the COMMON.CNMSTYLE.CONNMONFILE statement in CONNSTYL.

11.2 Command Authorization Errors

If you receive command authorization error messages:

- You did not code a %INCLUDE CONNAUTB in DSIAUTBU
- DSIAUTB has been modified and is missing the %INCLUDE DSIAUTBU

11.3 Monitoring Is not Scheduled

If you do not see a timer ID of FKXECMON then there is an error in your CONNSTYL definitions for the autotask or the initcmd:

- function.autotask.CONNMON = OPER5
- AUTOTASK.?CONNMON.InitCmd = AFTER 00:10:00,FKXECMON

11.4 DSI651I Message Issued

Message DSI651I is issued from the CONNMON EXEC when it is unable to determine the local stack IP address or host name:

```
DSI651I KEYWORD MISSING - ONE OF FOLLOWING REQUIRED: IPADDR
```


The most likely cause of this error is a very short interval defined on the COMMON.CNMSTYLE.CONNMONINTVL statement in CONNSTYL. This error will occur when the CONNMON EXEC is driven before the dynamic stack discovery code has completed.

To correct this error change the interval to a larger value.

11.5 EZL572I Message Issued

Message EZL572I is issued by AON initialization processing when AON is unable to route a request to an autotask:

```
EZL572I OPERATOR CGLOBALS NOT INITIALIZED - UNABLE TO ROUTE
COMMAND FKXECMON
```

You can ignore this message. It occurs because AON has detected the occurrence of IPCONN policy definitions and attempted to schedule its FKXECMON EXEC but could not find an appropriate autotask.

The parameters defined in member CONNSTYL will be used to schedule the CONNMON EXEC.

To eliminate the EZL572I message, delete the **AUTOOPS CONNOPER,ID=AUTCMON** policy definition on your system. It is typically found in DSIPARM member FKXCFG01.

12 Support for the CONNMON Prototype

The *CONNMON* prototype was developed and tested on a NetView 5.2 running on z/OS 1.6 and Communications Server for z/OS 1.6.

Support for the *CONNMON* prototype is available *as time permits* by contacting the author of this document.

Conclusion

Summary

The primary purpose for writing this White Paper was to conceptually prove that the use of TCPCONN to retrieve TCP/IP connection data improved the performance of the NetView connection monitoring and thresholding function. The performance tests validated the author's assumptions.

Enhancements were suggested for the IPCONN policy definition which further improved the performance of the NetView connection monitoring and thresholding.

The prototype REXX routine (CONNMON) will also be available on the NetView for z/OS downloads Web page.

Resources

Consult the following manuals for further information:

IBM Tivoli NetView for z/OS 5.2:

- Command Reference Volume 1
- Administration Reference
- Security Reference
- Messages and Codes
- Installation: Configuring Additional Components

Communications Server for z/OS 1.6:

- IP Configuration Reference

