Internet-Draft
Opstat Working Group
draft-ietf-opstat-client-server-02.txt
Expires: 30 Jul 1995

H. Clark
OARnet

30 Jan 1995

The Opstat Client-Server Model for Statistics Retrieval

Status of this Memo

This document is an Internet-Draft. Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet- Drafts as reference material or to cite them other than as ``work in progress.''

To learn the current status of any Internet-Draft, please check the ``1id-abstracts.txt'' listing contained in the Internet- Drafts Shadow Directories on ds.internic.net (US East Coast), nic.nordu.net (Europe), ftp.isi.edu (US West Coast), or munnari.oz.au (Pacific Rim).

Abstract

Network administrators gather data related to the performance, utilization, usability and growth of their data network. The amount of raw data gathered is usually quite large, typically ranging somewhere between several megabytes to several gigabytes of data each month. Few (if any) tools exist today for the sharing of that raw data among network operators or between a network service provider (NSP) and its customers. This document defines a model and protocol for a set of tools which could be used by NSPs and Network Operation Centers (NOCs) to share data among themselves and with customers.

1.0 Introduction

Network administrators gather data related to the performance, utilization, usability and growth of their data network. The primary goal of gathering the data is to facilitate near-term problem isolation and longer-term network planning within the organization. The amount of raw data gathered is usually quite large, typically ranging somewhere between several megabytes to several gigabytes of

data each month. From this raw data, the network administrator produces various types of reports. Few (if any) tools exist today for the sharing of that raw data among network operators or between a network service provider (NSP) and its customers. This document defines a model and protocol for a set of tools which could be used by NSPs and Network Operation Centers (NOCs) to share data among themselves and with customers.

1.1 The OPSTAT Model

Under the Operational Statistics model [1], there exists a common model under which tools exist for the collection, storage, retrieval and presentation of network management data.

This document defines a protocol which would allow a client on a remote machine to retrieve data from a central server, which itself retrieves from the common statistics database. The client then presents the data to the user in the form requested (maybe to a X-window, or to paper).

The basic model used for the retrieval methods defined in this document is a client-server model. This architecture envisions that each NOC (or NSP) should install a server which provides locally collected information for clients. Using a query language the client should be able to define the network object of interest, the interface, the metrics and the time period to be examined. Using a reliable transport-layer protocol (e.g. TCP), the server will transmit the requested data. Once this data is received by the client it could be processed and presented by a variety of tools including displaying the data in a X window, sending postscript to a printer, or displaying the raw data on the user's terminal.

The remainder of this document describes how the client and server interact, describes the protocol used between the client and server, and discusses a variety of other issues surrounding the sharing of data.

2.0 Client-Server Description

2.1 The Client

The basic function of the client is to retrieve data from the server. It will accept requests from the user, translate those requests into

the common retrieval protocol and transmit them to the server, wait for the server's reply, and send that reply to the user.

Note that this document does not define how the data should be presented to the user. There are many methods of doing this

including:

- use a X based tool that displays graphs (a line, a histogram, etc.)
- generate PostScript output to be sent to a printer
- dump the raw data to the user's terminal

Future documents based on the Operational Statistics model may define standard graphs and variables to be displayed, but this is work yet to be done (as of this writing).

2.2 The Server

The basic function of the server is to accept connections from a client, accept some series of commands from the client and perform a series of actions based on the commands, and then close the connection to the client.

The server must have some type of configuration file, which is left undefined in this document. The configuration file would list users that could access the server along with the authentication strings they would use. The configuration file should also allow the specification of the data items that the user should be permitted to access (and, by implication, not allowed to access). Server security concerns are specifically addressed in Section 4.

3.0 Protocol Commands

This section defines the commands which may be transmitted to the server and the server responses to those commands. The available commands are:

LOGIN - accept new connection
EXIT - disconnect

```
LIST   - show available variables
SELECT - mark data for retrieval
STATUS - show the state of the server
GET    - download data to the client
```

In addition, a state machine describing specific actions by the server is included. Server security concerns are addressed in Section 4.

Note that in some of the descriptions below, the term <ASCII-STRING> is used. This refers to printable ASCII characters, defined as all

letters, numbers, and special characters such as $, %, or *. It specifically excludes all special control characters in the lower parts of the character set (i.e. 0x00 - 0x1F).

3.1 Command Return Codes

The only responses a server may return to a client are:

```
RETURN-CODE  ::=  <OK-TYPE> | <ERROR-TYPE>
OK-TYPE      ::=  OK | OK <ASCII-STRING>
ERROR-TYPE   ::=  ERROR | ERROR <ASCII-STRING>
```

The server should strive to return text along with the "OK" or "ERROR" when appropriate to help guide the user in using the server. If a command is sent where the server should return data, the server must also return an OK (or ERROR, if appropriate) when if finishes sending the data.

3.2 The LOGIN Command

The LOGIN command authenticates a user to the server. The format of the LOGIN command is:

```
LOGIN-CMD    ::=  LOGIN <username> <auth-string>
USERNAME     ::=  " <ASCII-STRING> "
AUTH-STRING  ::=  " <ASCII-STRING> "
```

A sample LOGIN command might look like:

LOGIN "user1" "cows"

The <auth-string> entry is just a simple ASCII string which helps to authenticate a user to the server. This could be, for example, a clear-text password or an encrypted/one-time password (preferred).

3.3 The EXIT Command

The EXIT command disconnects a current user from the server. The format of the EXIT command is:

EXIT

Note that upon reception of an EXIT command, the server MUST always close the connection, even if it would be appropriate to return an

ERROR return code.

3.4 The SELECT Command

The SELECT command is the function used to tag data for retrieval from the server. The SELECT command has the format:

```
SELECT-COM   ::=  SELECT <NETWORK> <ROUTER> <INTERFACE> <VARNAME>
                  <GRANULARITY> <START-DATE> <END-DATE> <AGG> <SELECT-COND>
NETWORK      ::=  <ASCII-STRING>
ROUTER       ::=  <ASCII-STRING>
INTERFACE    ::=  <ASCII-STRING>
VARNAME      ::=  <ASCII-STRING>
GRANULARITY  ::=  <ASCII-STRING>
START-DATE   ::=  <DATE-TYPE>
END-DATE     ::=  <DATE-TYPE>
DATE-TYPE    ::=  YYYY-MM-YY
AGG          ::=  <AGG-TYPE> | NULL
AGG-TYPE     ::=  TOTAL | PEAK
SELECT-COND  ::=  <SELECT-STMT> | NULL
SELECT-STMT  ::=  WITH DATA <COND-TYPE> <ASCII-STRING>
```

```
COND-TYPE    ::=  LE | GE | EQ | NE | LT | GT
```

If any conditional within the SELECT does not match existing data within the database (such as VARNAME, the S-DATE or E-DATE, or GRANULARITY), the server must return an ERROR (and hopefully a meaningful error message). The granularity should always be specified in seconds. A sample query might be:

```
SELECT net rtr1 eth-0 ifInOctets 900 1992-01-01 1992-02-01
```

which would select all data from network "net" router "rtr1" interface "eth-0" from Jan 1, 1992 to Feb 1, 1992.

Note that if the client requests some type of aggregation to be performed upon the data, then the aggregation field specifies how to perform the aggregration (i.e. total or peak) and the granularity specifies to what interval (in seconds) to agggregate the data to. If the server cannot perform the requested action, then it must return an error.

Upon completion of the data lookup, the SELECT must return the an indication of whether the lookup was successful and (if the search was successful) the tag associated with that data. If the lookup was successful, then information in the "OK" return code should be encoded as:


```
TAG <ASCII-STRING>
```

In this case, the use of the word TAG is used as a handle for the selected data on the server. Note that this single handle may refer to one or more specific SNMP variables (refer to [1] for a further explanation).

For example, if the tag "foobar" were assigned to the select example above, then the OK would be as:

```
OK TAG foobar
```

It is recommended that the return tag string be less than 10 bytes long (this gives many tag combinations), although the server (and client) should be capable of handling arbitrary length strings.

There is no requirement that the TAG have any particular meaning and may be composed of arbitrary strings.

The server must keep any internal information it needs during a session so that all SELECT tags can be processed by GET or other commands. If a server doesn't have the resources to process the given SELECT, it must return an error message.

It is the responsibility of the client to store information about the data that a particular tag corresponds to, i.e. if the client had requested all ifInOctet data for October 1993 to be placed in a tag called "1234", then the client must store that information someplace as the variables will not be listed for each tag.

3.5 The STATUS Command

The STATUS command shows the general state of the server plus listing all data sets which have been tagged via the SELECT command. The STATUS command has no arguments. The output from a STATUS command is:

```
STATUS-RETURN    ::=  START-STATUS <STATUS-DATA> END-STATUS
STATUS-DATA      ::=  <SERVER-STATUS> <SERVER-TAG-LIST>
SERVER-STATUS    ::=  "STATUS= " <STATUS-FIELDS>
STATUS-FIELDS    ::=  "OK" | "NOT-OK"
SERVER-TAG-LIST  ::=  <SERVER-TAG> | NULL
SERVER-TAG       ::=  "TAG" <TAG-ID> "SIZE" <NUMBER>
```

For example, a sample output might look like:


```
START-STATUS
STATUS= OK
TAG 1234 SIZE 123456
TAG ABCD SIZE 654321
END-STATUS
```

3.6 The GET Command

The GET command actually retrieves the data chosen via a previous SELECT command. The GET command has the format:

```
GET-CMD  ::=  GET <TAG>
TAG      ::=  <ASCII-STRING>
```

If the TAG matches a previously returned TAG from a SELECT statement, then the previously tagged data is returned. If the TAG is invalid (i.e. it hasn't been previously assigned by the server), then the server must return an error. If the server, while retrieving the data, cannot retrieve some portion of the data (i.e. some of the data previously found disappeared between the time of the SELECT and the time of the GET), then the server must return an error.

The data to be returned may be in any of several formats. Currently, the only defined format is RFC1404. To handle these different types of data, the following constructs are used:

```
RETURN-DATA-TYPE   ::=   START-DATA  <RETURN-TYPE>  <DATA>   END-DATA
RETURN-TYPE        ::=  1404
```

An example would be:

START-DATA 1404 1404 data stream here... END-DATA

After the end of data, an OK (or ERROR) must still be returned. If an ERROR is returned, the client must discard all data received from the server.

3.7 The LIST Command

The LIST command allows the client to query the server about available data residing on the server. The LIST command has the format:

```
LIST-CMD  ::=  LIST <net> <rtr> <intf> <var> <gran> <sdate> <edate>
```



```
<net>     ::=  <ASCII-STRING> | *
```

```
<rtr>     ::=  <ASCII-STRING> | *
<intf>    ::=  <ASCII-STRING> | *
<var>     ::=  <ASCII-STRING> | *
<gran>    ::=  <ASCII-STRING> | *
<sdate>   ::=  <DATE-TYPE>    | *
<edate>   ::=  <DATE-TYPE>    | *
```

For example, to get a list of networks that the server has data for, you would use the command

```
LIST * * * * * * *
```

The command

```
LIST netx rtry * * * * *
```

will list all interfaces for rtry. The command

```
LIST netx rtry * ifInOctets * 1993-02-01 *
```

will get the list of interfaces on router "rtry" in network "netx" which have values for the variable "ifInOctets" after the start date of Februrary 1, 1993.

To process wildcards in a LIST command, follow these rules:

1) Only the leftmost wildcard will be serviced for a given LIST command
2) If all fields to the right of the leftmost wildcard are wildcards, then all values for the wildcard being processed will be returned.
3) If specific values are given for fields to the right of the wildcard being serviced, then the specific values must match a known value

The output from the LIST command is formatted as follows:

```
LIST-RETURN  ::=  START-LIST <LIST-ENTRY> END-LIST
LIST-ENTRY   ::=  <net> <rtr> <intf> <var> <gran> <sdate> <edate>
<net>        ::=  <ASCII-STRING>
<rtr>        ::=  <ASCII-STRING> | <NULL>
<intf>       ::=  <ASCII-STRING> | <NULL>
<var>        ::=  <ASCII-STRING> | <NULL>
```

```
<gran>       ::=  <ASCII-STRING> | <NULL>
<sdate>      ::=  <DATE-TYPE>    | <NULL>
<edate>      ::=  <DATE-TYPE>    | <NULL>
```

Note that only the fields with values in them will be returned by the server. For example, the query to find the interfaces on rtry:

```
LIST netx rtry * * * * *
```

might return

```
START-LIST
netx rtry intf1
netx rtry intf2
netx rtry intf3
END-LIST
```

The query to find interfaces having ifInOctets data with a 15 minute granularity:

```
LIST netx rtry * ifInOctets 15min * *
```

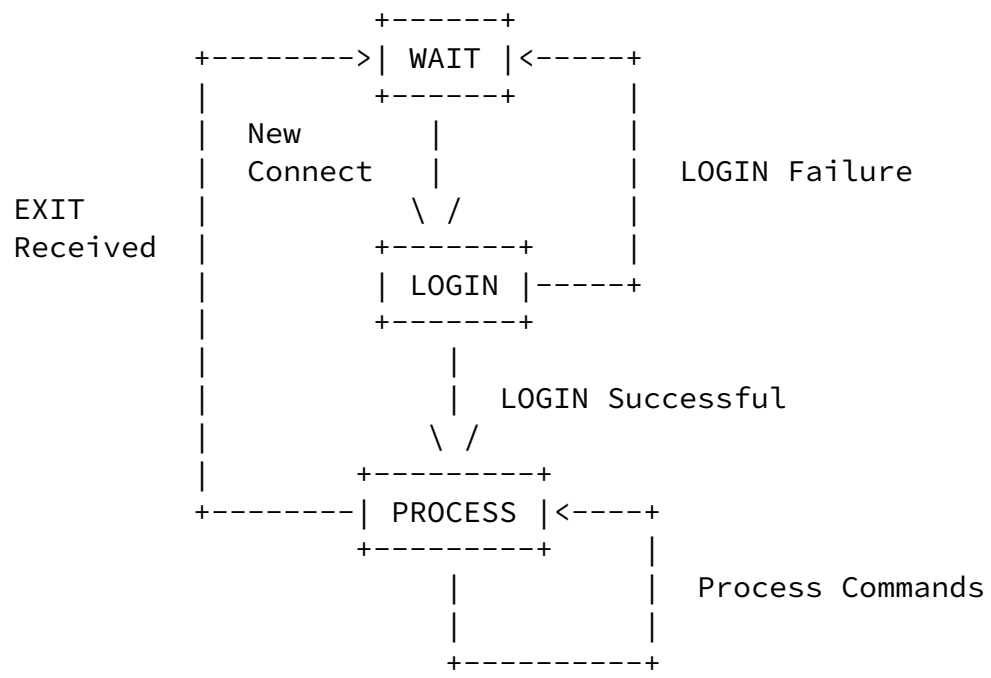
might return

```
START-LIST
netx rtry intf1
netx rtry intf2
netx rtry intf3
END-LIST
```

If, while processing a LIST command, the server encounters an error, then the server must return an "ERROR" message.

3.8 The Server State Machine

The state machine pictured below describes how a server should interact with a client:



The server normally stays in WAIT (after starting and initialization) until a new connection is made from a client. The first command a client must issue is a LOGIN command, otherwise the server must immediately close the connection. If a client issues a LOGIN command using a invalid username or password, then the server must immediately close the connection.

Once a successful LOGIN is received, the server enters the PROCESS state where it processes some number of LIST, GET, STATUS, and SELECT commands. Any other command received while in this state must be ignored, except for the EXIT command. Once an EXIT command is received, the server exits immediately (after perfoming any needed internal bookkeeping) and returns to the WAIT state.

4.0 Security Issues

There are legal, ethical and political concerns of data sharing. For this reason there is a need to insure integrity and confidentiality of any shared data. Although not specified in this standard, mechan-

isms to control a user's access to specific data about specific objects may need to be included in server implementations. This could potentially be done in several ways, including a configuration file that listed the objects a user was allowed to access or limiting file access by using file permissions within a given file system. At a minimum, the server should not allow default access to all data on the server.

Additionally, the server should strictly follow the state diagram

shown in section 3.8. The server should be tested with arbitrary strings in the command fields to ensure that no unexpected security problems will be caused by the server. The server should specifically discard illegal ASCII characters as discussed in section 3.0. If the server executes other programs, then the server must verify that no unexpected side-effects will occur as the result of the invocation or the arguments given to that program.

5.0 Summary

This document defines a protocol which could be used in a client-server relationship to retrieve statistics from a remote database server.

Much work remains to be done in the area of Operational Statistics including questions such as:

- what "standard" graphs or "variables" should always be made available to the user?
- what additions to the standard MIB would make the network manager's job easier?

6.0 References

[1] RFC1404 A Model for Common Operational Statistics, B. Stockman

Appendix A: Sample Client-Server Sessions

Session 1: Failed LOGIN

```
LOGIN henryc foobar
ERROR LOGIN Incorrect
```

Session 2: Check available variables on router rtr1 interface eth0

```
LOGIN henry foobar
OK
LIST OARnet rtr1 eth0 * * * *
START-LIST
OARnet rtr1 eth0 ifInOctets
OARnet rtr1 eth0 ifOutOctets
OARnet rtr1 eth0 ifInErrors
OARnet rtr1 eth0 ifOutErrors
```



```
END-LIST
OK
EXIT
OK
```

Session 3: Retrieve a bit of data from the server

```
LOGIN henryc foobar
OK
SELECT OARnet rtr1 eth0 InBytes 15min 1993-02-01 1993-03-01
OK TAG blah
STATUS
START-STATUS
STATUS= OK
TAG blah SIZE 654321
END-STATUS
OK
GET blah
START-DATA 1404
START-DATA
BEGIN_LABEL,,
[InBytes],19930201000000,19930301000000,
END_LABEL
```

```
   BEGIN_DEVICE,
   OARnet,rtr1,eth0,1536000,IP,1.2.3.4,+0500,
   [InBytes,peak,[ifInOctets,900,900]],
   END_DEVICE
   BEGIN_DATA
   19920730000000,InBytes,300,1,
   19920730000300,InBytes,300,2,
   19920730000600,InBytes,300,3,
   END-DATA
   END-DATA
   OK
   EXIT
   OK
```

Author's Address

Henry Clark
OARnet
2455 North Star Rd

Columbus, OH 43221

Phone: (614) 728-8100 x 212

EMail: henryc@oar.net