A Bigjar Systems, LLC White Paper
July 2024

Discover Latent Network Problems with Automated Bandwidth Testing (ABT)

Executive Overview

Everyone in IT has experienced this situation: End users discover a latent network problem when they come into the office, and IT staff must scramble to fix the problem. This white paper shows how you can use the Network Utility Knife (from Bigjar Systems, LLC) in batch mode to automatically find these problems before the users arrive, giving IT staff time to start repairs.

Introduction

Latent network problems are the bane of IT staff. Many network management systems can identify if a connection is hard down. However, because routing protocols are designed to be forgiving of packet loss, a problem that involves a noisy circuit or one that's flooded can be difficult to detect, to give some examples.

The latest version of the Network Utility Knife (NUK) has the ability to run jobs (batch mode) and one type of job tests available bandwidth. With the help of some shell scripts (included in this document and available for download) this type of job can be launched automatically with actions taken based on the throughput measured. In this way, IT staff can receive notifications when employees are not in the office, giving them more time to fix the problem.

In conclusion, the NUK can help you identify problems before users arrive in the office, giving you time to effect repairs. Nothing in the market at this price point can provide this functionality.

Now the reason the enlightened prince and the wise general conquer the enemy ... is foreknowledge.

- Sun Tzu

Automated Bandwidth Testing (ABT)

Here we describe in detail the scripts and programs involved in ABT. But first, an overview of the technology.

NUK jobs

Jobs allow for automated use of the utilities on the NUK. The included scripts submit jobs to test throughput (sample job files included). The script then waits for the output from the job, and takes action (e.g. sending an email) if necessary. This process can be repeated for different sites or different networks and for different NUK's.

Script logic

The logic for the scripts is as follows:

The **master.sh** script goes through all files in a directory and runs **bandwidthcheck.sh** passing each file as an argument. This allows for one executable to test bandwidth to multiple sites in sequence.

The **bandwidthcheck.sh** script does the following:
Uploads the job file to the NUK specified in the ABTIP
Waits a set period of time for the job to finish
Downloads the output produced by the job
Compares a value in the job output to a value specified in the job file
If the value is less than the specified amount, then a third script is run (the script name is specified in the job file).

The **error.sh** script performs the action designated when bandwidth is too low. In the supplied script it sends an email to address specified in the ABTEMAIL variable containing debugging output from the job and the output generated by the NUK using the Unix **mail** command.

How jobs work

Job are launched on the NUK by uploading text files to a NUK via FTP or SFTP. The file is uploaded to */storage/job/intake* (or */config/storage/job/intake* if using SFTP). The specified utility is then run on the NUK and the output from this utility is stored on the NUK where it can be retrieved via FTP or HTTP.

Note: any extension is stripped from the job file and the output from the job is stored in *jobfilename.out* in the */storage/job/output/* directory (accessible via FTP).

The format of the job file is as follows

Job file format

The text file (or job file) consists of multiple lines of the form “X=Y”. The lines define which utility is to be run, and what arguments should be passed to the utility. A sample job file is included below:

```
SCRIPT=srv-nett2.sh
PROG='iperfclient'
IPADDR='192.168.10.193'                    # IP address of remote NUK
IPADDRVAR='iperfclient_ipaddr'
UOPTIONS='-w 800k -i .5 -O 1 -t 3 -J'      # Options to pass to iperf3
```

There are many more possible options, but the above file will launch iperf3 in client mode and tell it to connect to 192.168.10.193 with the following options '-w 800k -i .5 -O 1 -t 3 -J'. From left to right these options have the following meanings:

-w 800k	Set the TCP Window to 800k //**//
-i .5	Report throughput every .5 seconds
-O 1	Omit the first second of testing from the total
-t 3	Test for 3 seconds
-J	Generate output in JSON format (required)

Additional lines of the form A=B can be added as long as these variables are not used internally by the NUK. We will add these lines to the job file so that a single file contains instructions both for the NUK and for the included scripts. To avoid name collisions, we will prepend ABT to each variable name.

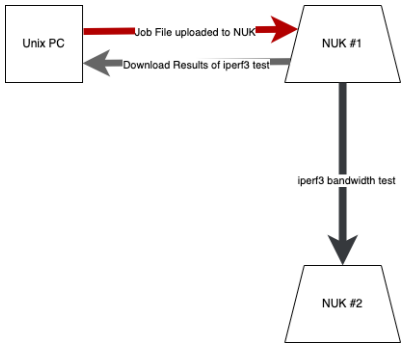
A full sample job file is included below:

```
SCRIPT=srv-nett2.sh
PROG='iperfclient'
IPADDR='192.168.10.193'
IPADDRVAR='iperfclient_ipaddr'
UOPTIONS='-w 800k -i .5 -O 1 -t 3 -J'
ABTIPADDR=192.168.10.203
ABTUSER=root                            # Username for upload. Usually root
ABTPASSWORD=password                    # Password for the above username
ABTFIELD=13                             # Field to numerically test against.
                                        # 13 = received bandwidth, 11 = sent bandwidth
ABTLESSTHAN=1                           # Bandwidth below which an action is taken (Mbit)
ABTSCRIPT=./error.sh                    # Script to run if field 13 is less than expected
                                        # An absolute path would probably be a good idea
ABTEMAIL="jmitchel@localhost"           # error.sh sends email to the following address
```

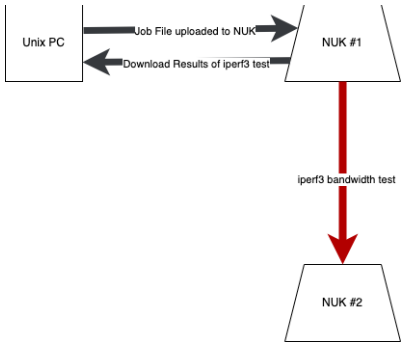
Diagrams

We know that some of you are visual learners, so here are diagrams showing the networking actions taken by the supplied scripts. In each diagram the step mentioned is shown in red.

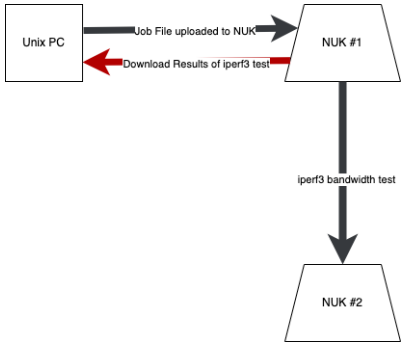
Step 1 – The job file is uploaded to the NUK



Step 2 – The iperf3 bandwidth test is performed



Step 3 – The results of the iperf3 test are retrieved



Conclusion

With the NUK and the supplied scripts, IT staff can get advance warning of problems before they impact users. This can improve productivity – especially at remote offices where problems may remain undetected until users arrive for work.

Appendix A

Here are the scripts described above. They are also on our website at: https://www.bigjar.com/abt-finalv2.tar.gz

master.sh

master.sh goes through every file in the directory specified by the ABTSITES variable (or in the first argument passed to **master.sh**) and launches **bandwidthcheck.sh** to process it.

```
#!/bin/sh
#
#
# Arguments
# 1: Directory to go through and send jobs to NUK
#

BANDWIDTHCHECK=./bandwidthcheck.sh
DIRECTORY=$1

#
#
# Load needed functions
. ./system.conf

DIRECTORY=$1
if [ "$DIRECTORY" = "" ]; then
  DIRECTORY=$ABTSITES
fi


for FILE in `ls -d -1 $DIRECTORY/** | grep -v \~`
do
  sh $BANDWIDTHCHECK $FILE
  RESULT=$?
  debugvar RESULT
  debugvar FILE
done
```

bandwidthcheck.sh

bandwidthcheck.sh starts the job on the NUK that tests bandwidth between the two NUK's. It launches the script specified by the ABTSCRIPT variable.

```
#!/bin/sh
#
# arguments are:
#
# 1: IP Address or DNS name of NUK to upload the file to
# 2: Password of NUK
# 3: filename
# 4: field#
# 5: value to check against. If less than
# 6: Action Script to run if bandwidth is less
#
# Error codes:
# 1: the upload of the job file to the NUK fails
#
# Script Logic:
#
# 0.5) Create filename with random hex digits added
# 1) Upload job file to IP address of NUK
# 2) Wait for job file to finish
# 3) Download output from job
# 4) Run output through python script sending output to a variable
# 5) Parse resulting CSV output
# 6) If value is less than the value passed to this script, then the action
script is run with the job file as the only argument
#
#
# The script returns the following values
# 0: Success
# 1: Error returned by curl
# 2: Error returned by wget
# 4: Bandwidth is lower than threshold
# 5: Bandwidth can't be derived from output from iperf3 output
#
# Function(s)
#
callactionscript () {

# Arguments passed to this function
# 1) Error String
# 2) filename of job
# 3) filename containing OUTPUT from commands run
```

```
  debug "Error in ABT (Automated Bandwidth Testing). Running specified
script"
  CASSTRING=$1
  JOBFILE=$2
  OUTPUTFILE=$3
  ERRORSTRING="$CASSTRING Running $ACTIONSCRIPT"
  debug " sh $ACTIONSCRIPT "$ERRORSTRING" $JOBFILE $OUTPUTFILE"
  debug "`sh $ACTIONSCRIPT "$ERRORSTRING" $JOBFILE $OUTPUTFILE`"
  RESULT=$?
  if [ $RESULT != "0" ]; then
        debug "Error running $ACTIONSCRIPT"
  fi
}

# Step 0: Initialization
#
# Set location of executables
#
#CURL="/usr/pkg/bin/curl"
DATE="/bin/date"

. ./system.conf                                                 # Load
needed functions into memory

NUMARGS=$#
ARGTEST=`expr $NUMARGS \>\= 6`
RESULT=$?
debugvar RESULT

# Text output from ARGTEST is 0 if NOT true, and 1 if TRUE
# For some perverse reason (probably historial/hysterical) The error codes
returned are the reverse :(
#
# expr only works for integers!!!!!!!!!!!!!!!!
#
if [ "$ARGTEST" = "1" ]; then
  IPADDR=$1
  PASSWORD=$2
  FILENAME=$3
  FIELD=$4
  LESSTHAN=$5
  ACTIONSCRIPT=$5
  USER=root
else
  FILENAME=$1
  . $FILENAME
  IPADDR=$ABTIPADDR
  USER=$ABTUSER
  PASSWORD=$ABTPASSWORD
```

```
  FIELD=$ABTFIELD
  LESSTHAN=$ABTLESSTHAN
  ACTIONSCRIPT=$ABTSCRIPT
fi

#CURL=/usr/pkg/bin/curl

RANDHEX=`randhex`

if [ -e $FILENAME ]; then
  SHORTJOBFILENAME=`stripdir $FILENAME`
  JOBFILENAME=$SHORTJOBFILENAME-$RANDHEX
  cp $FILENAME $ABTOUTPUT/$JOBFILENAME
  debugvar JOBFILENAME
  debugvar SHORTJOBFILENAME
fi
#
# Step 0.5: Set up Temporary file to receive output. This will only be used
in case of low bandwidth or an error
#

TEMPFILE=$ABTOUTPUT/bwcheck.$RANDHEX
DEBUGFILE=$TEMPFILE

echo "" > $DEBUGFILE

debugvar SHORTJOBFILENAME
debugvar JOBFILENAME
debugvar FILENAME
debugvar IPADDR
debugvar USER
debugvar PASSWORD
debugvar FIELD
debugvar LESSTHAN
debugvar ACTIONSCRIPT

# Step 1: Upload job file to NUK

URL="ftp://$IPADDR/storage/job/intake"

if [ "$ABTUSER" = "" ]; then
  ABTUSER=root
fi

debug "$CURL -s --user $ABTUSER:$PASSWORD --upload-file
$ABTOUTPUT/$JOBFILENAME $URL/$JOBFILENAME"
debug `$CURL -s --user $ABTUSER:$PASSWORD --upload-file
$ABTOUTPUT/$JOBFILENAME $URL/$JOBFILENAME`
```

```
RESULT=$?
if [ "$RESULT" != "0" ]; then
  callactionscript "Error uploading job file!" $FILENAME $TEMPFILE
  exit 1
fi

# Step 2: Wait for job to finish and print countdown if DEBUG is set

if [ "$DEBUG" != "" ]; then
  echo "Sleeeping for 10 seconds"
  sleep 10
else
  sleep 10 >/dev/null
fi

# Step 3: Download Output from Job file
#
#
# Save starting directory so we can go back

SAVEPWD=$PWD
cd $ABTOUTPUT
$WGET --user=root --password=$PASSWORD -nv
ftp://$IPADDR/storage/job/output/$JOBFILENAME.out 2>> $TEMPFILE
#debugvar WGETOUTPUT
RESULT=$?
cd $SAVEPWD
if [ "$RESULT" = "1" ]; then
  callactionscript "Error returned by wget!"  $JOBFILENAME $TEMPFILE
  exit 2
fi

JSONFILENAME=$ABTOUTPUT/$JOBFILENAME.out
debugvar JSONFILENAME

CSV=`cat $JSONFILENAME | $PYTHON iperf3tocsv.py`
debugvar CSV
echo $CSV | grep "Error"
RESULT=$?
if [ $RESULT = "0" ]; then
  callactionscript "Error received from iperf3tocsv.py!"  $JOBFILENAME
$TEMPFILE
  exit 4
else
  VALUE=`echo $CSV | cut -f $FIELD -d","`

  debugvar VALUE
  debugvar LESSTHAN
# MEGAVALUE=`echo $VALUE \* 1000000 | bc`
```

```
# MEGALESSTHAN=`echo $LESSTHAN \* 1000000 | bc`
# debugvar MEGAVALUE
# debugvar MEGALESSTHAN


  if [ "$VALUE" != "" ]; then
        debug "echo $VALUE \< $LESSTHAN | bc "
        RESULT=`echo $VALUE \< $LESSTHAN | bc `
        debugvar RESULT
        if [ "$RESULT" = "1" ]; then
              callactionscript "Bandwidth is lower than threshold given."
$JOBFILENAME $TEMPFILE
              exit 4
        else
              debug "Bandwidth tested is greater than value given"
        fi
  else
        callactionscript "Error: No value returned from numerical
comparison. iperf3 not set to output in JSON format?" $JOBFILENAME
$TEMPFILE
        exit 5
  fi
fi
```

error.sh

The **error.sh** script sends an email to the user specified by the ABTEMAIL variable in the site file supplied to **bandwidthcheck.sh**:

```
#!/bin/sh
#
#
#
# Arguments passed to this script
#
# 1) Error string
# 2) Job file
# 3) iPerf output file (output from commands goes here)
#
# Load needed functions
. ./system.conf

# DEBUG=ON

TEMPFILE=`randfile $TEMPDIR/error`
TEMPSTRING=$1
STRING="$TEMPSTRING Target IP is $UOPTIONS"
TEMPFILENAME=$2
OUTPUTFILE=$3
RANDHEX=`echo $TEMPFILE | cut -f 2 -d "-"`

SHORTFILENAME=`stripdir $TEMPFILENAME`
FILENAME=$ABTOUTPUT/$SHORTFILENAME
IPERFOUTPUT=$ABTOUTPUT/$SHORTFILENAME.out

debugvar STRING
debugvar FILENAME
debugvar OUTPUTFILE
debugvar IPERFOUTPUT

. $FILENAME

echo "$STRING" > $TEMPFILE
echo >> $TEMPFILE
echo "******************Job file included below******************" >>
$TEMPFILE
echo >> $TEMPFILE
if [ -f $FILENAME ]; then
  cat $FILENAME >> $TEMPFILE
else
  debug "File: "$FILENAME" Does not exist"
fi
```

```
echo >> $TEMPFILE
echo "************************End Job File************************" >>
$TEMPFILE
echo >> $TEMPFILE
echo "*******************Debugging Output included
below*******************" >> $TEMPFILE
echo >> $TEMPFILE

if [ -f $OUTPUTFILE ]; then
  cat $OUTPUTFILE >> $TEMPFILE
else
  debug "File: "$OUTPUTFILE" Does not exist"
fi

echo >> $TEMPFILE
echo "************************End
Debugging*******************************" >> $TEMPFILE
echo >> $TEMPFILE
echo >> $TEMPFILE
echo "*******************iperf3 output included below*******************"
>> $TEMPFILE
echo >> $TEMPFILE

if [ -f $IPERFOUTPUT ]; then
  cat $IPERFOUTPUT >> $TEMPFILE
else
  debug "File: "$IPERFOUTPUT" Does not exist"
fi

echo >> $TEMPFILE
echo "************************End iperf3 output************************"
>> $TEMPFILE
echo >> $TEMPFILE
echo >> $TEMPFILE

if [ -f $TEMPFILE ]; then
  debug "Time to make the biscuits. (Sending email)"
# debugvar TEMPFILE
  debugvar TEMPFILE
# cat $TEMPFILE
  cat $TEMPFILE | mail -s "$STRING" $ABTEMAIL
  RESULT=$?
  debug "Here's the Error Code from sending mail: $RESULT"
else
  debug "Temporary File does not exist!!!!!!!! The filename is: $TEMPFILE"
fi

#showvar STRING
```

```
#showvar FILENAME
#showvar OUTPUTFILE

exit 0
```

system.conf

system.conf contains some global variables that will apply across all tested sites as well as some functions

```
# These are the variables that need to be changed

TEMPDIR="/tmp"
DEBUGFILE=./debug.txt
ABTSITES=/home/jmitchel/abt/sites
CURL="/usr/bin/curl"
WGET=/usr/bin/wget
BANDWIDTHCHECK=/home/jmitchel/abt/bandwidthcheck.sh
PYTHON=/usr/bin/python3.7
#
# WARNING! The ABTOUTPUT variable must contain an absolute path!!!!!!!!
#
ABTOUTPUT="/home/jmitchel/abt/output"

# Changing things below this line could cause serious problems!!!!!!!!!

echoerr() { printf "%s\n" "$*" 1>&2;}
echoerr2() { echo "$*" 1>&2; }

sanfilename() { echo $* | sed s+/+_+g; }

gethash() { hmac256 "`grep HOSTNAME /config/nuk.conf`" $*; }

stripdir() {
  echo $1 | rev | cut -d'/' -f 1 | rev
}

sanarg () { bash /script/sanitizearg $1; }

# Display contents of variable $1 is the variable name (no $!)
showvar () {
  NAMEOFVAR=$1
  eval VALUEOFVAR=\$$NAMEOFVAR
  printf '%s=' "$NAMEOFVAR" 1>&2
  printf "\'" 1>&2
  echoerr $VALUEOFVAR\'
  unset NAMEOFVAR
  unset NAMEOFVAR2
}

#dialogvar () {
# sanarg $1
```

```
# logmessage file "message to be logged"
logmessage () {
  NUMARG=$#
  SYSFILE=$1
  i=1
  LMMESSAGE=""

  while [ $i -lt $NUMARG ]
  do
        j=$(($i+1))
        eval LMARG=\$$j
        TMPLMMESSAGE="$LMMESSAGE $LMARG"
        LMMESSAGE=$TMPLMMESSAGE
        i=$(($i+1))
  done
# showvar SYSFILE
# showvar LMMESSAGE
  echoerr $LMMESSAGE
  echo $LMMESSAGE >> $SYSFILE
}

#logvar <file to log to> <variable to output>
# logvar outputs VARIABLE=$VARIABLE to the file specified
logvar () {
  LOGSYSFILE=$1
  LMVAR=$2
  showvar $LMVAR 2>> $LOGSYSFILE
}

# Remove $1 from $2
stripname () {
  TOBEREMOVED=$1
  REMOVEFROM=$2
  echo $REMOVEFROM | sed s+$TOBEREMOVED++
}

#debug "String to debug" - output goes to $DEBUGFILE
debug () {
# echoerr "Arguments to debug are $*"
# showvar DEBUG
  if [ ! "$DEBUG" = "" ]; then
#       showvar DEBUG
        logmessage $DEBUGFILE "$*"
#       echoerr "Done with Logmessage"
        if [ "$DEBUGSLEEP" = "" ]; then
              DEBUGSLEEP=0.4
#             showvar DEBUGSLEEP
        fi
        if [ $DEBUGSLEEP = 0 ]; then
```

```
                printf '%s' "No sleep till Brooklyn! "
                intsleep 0
        else
                DEBUGSTRING="Sleeping for $DEBUGSLEEP seconds"
                DEBUGCHAR=${#DEBUGSTRING}
#               showvar DEBUGCHAR
                i=1
                #printf --
                printf '%s %s %s' $DEBUGSTRING 1>&2
                sleep $DEBUGSLEEP >/dev/null 2>/dev/null
#               DC=$(($DEBUGCHAR*2))
                DC=$(($DEBUGCHAR+1))
#               showvar DC
                while [ $i -lt $DC ]
                do
                        printf '\b \b' 1>&2
                        i=$(($i+1))
                done
                printf '\b' 1>&2
        fi
  fi
}

debugvar () {
  #$1 = VARIABLE Name and value to write to debug file
  DVVAR=$1
  DVTEXT=`showvar $DVVAR 2>&1`
# showvar DVTEXT
  if ! [ "$DEBUG" = "" ]; then
  #     showvar DVVAR
  #     showvar DVTEXT
  #     debug "Debugvar is running"
        debug "$DVTEXT"
  fi
  unset DVVAR
  unset DVTEXT
}

debugvar2 () {
  if [ ! $DEBUG = "" ]; then
        #$1 = VARIABLE whose value were outputing
        DV2=$1
        showvar $DV2
        logvar $DEBUGFILE $DV2
        sleep $DEBUGSLEEP
        unset DV2
  fi
}
```

```
getsavedvar () {
        GSVVAR=$1
        GSVCONFIGVAR=$2
        VALUE=\$$GSVVAR
        if [ $VALUE == "" ]; then
                eval $GSVVAR="$2"
                debugvar GSVVAR
                debugvar VALUE
                debugvar GSVCONFIGVAR
        fi
}

delayeddelete () {
  DDFILE=$1
  DDDELAY=$2
  if [ "$DDDELAY" = "0" ]; then
        DDDELAY=60
  fi
  if [ "$DDDELAY" = "" ]; then
        DDDELAY=60
  fi
  /script/delaydel.sh $DDFILE $DDDELAY &
}

sethn () {
  HOSTNAME=`/bin/hostname`
  if [ "$HOSTNAME" = "" ]; then
        HOSTNAME="NoName"
  fi
  hostname $HOSTNAME
  export HOSTNAME
}

randhex () { openssl rand -hex 6; }

randfile () {
  RANDFILE=$1
  RANDRANDOM=`randhex`
  echo $RANDFILE.$RANDRANDOM
}

intsleep () {
  TIME=$1
  sh /script/looppressq.sh $TIME
}

countdown() {
  TIMESLEEP=$1
  msg="Sleeping for ... "
```

```
        echo -n "$msg"
        for I in `seq $TIMESLEEP 1`
        do
          printf "$I"
                sleep 1
        ILENGTH=`echo $I | wc -c`
        for J in `seq $ILENGTH 2`
        do
              printf "\b \b"
        done
        done
  printf "\b done!\n"
}
```

iperf3tocsv.py

iperf3tocsv.py was written and is copyrighted by Kirth Gersen (https://nspeed.app/) and is available at https://github.com/kgersen/iperf3protect/tree/master. It takes the JSON output from iperf3 and converts it to CSV format which can then be parsed by **bandwidthcheck.sh**. Unlike the other scripts, it is written in python.

This script is licensed under the MIT license: (from:
https://github.com/kgersen/iperf3protect/blob/master/LICENSE.md) :


```
#!/usr/bin/env python

"""
    Version: 1.1

    Author: Kirth Gersen
    Date created: 6/5/2016
    Date modified: 9/12/2016
    Python Version: 2.7

"""

from __future__ import print_function
import json
import sys
import csv

db = {}

def eprint(*args, **kwargs):
    print(*args, file=sys.stderr, **kwargs)

def main():
    global db
    """main program"""

    csv.register_dialect('iperf3log', delimiter=',',
quoting=csv.QUOTE_MINIMAL)
```

```
    csvwriter = csv.writer(sys.stdout, 'iperf3log')

    if len(sys.argv) == 2:
        if (sys.argv[1] != "-h"):
            sys.exit("unknown option")
        else:
            csvwriter.writerow(["date", "ip", "localport", "remoteport",
"duration", "protocol", "num_streams", "cookie", "sent", "sent_mbps",
"rcvd", "rcvd_mbps", "totalsent", "totalreceived"])
            sys.exit(0)

    # accummulate volume per ip in a dict
    db = {}

    # highly specific json parser
    # assumes top { } pair are in single line

    jsonstr = ""
    i = 0
    m = False
    for line in sys.stdin:
        i += 1
        if line == "{\n":
            jsonstr = "{"
            #print("found open line %d",i)
            m = True
        elif line == "}\n":
            jsonstr += "}"
            #print("found close line %d",i)
            if m:
                process(jsonstr,csvwriter)
            m = False
            jsonstr = ""
        else:
            if m:
                jsonstr += line
            #else:
                #print("bogus at line %d = %s",i,line)

def process(js,csvwriter):
    global db
    #print(js)
    try:
        obj = json.loads(js)
    except:
        eprint("bad json")
        pass
        return False
    try:
```

```
        # caveat: assumes multiple streams are all from same IP so we take
the 1st one
        # todo: handle errors and missing elements
        ip = (obj["start"]["connected"][0]["remote_host"]).encode('ascii',
'ignore')
        local_port = obj["start"]["connected"][0]["local_port"]
        remote_port = obj["start"]["connected"][0]["remote_port"]

        sent = obj["end"]["sum_sent"]["bytes"]
        rcvd = obj["end"]["sum_received"]["bytes"]
        sent_speed = obj["end"]["sum_sent"]["bits_per_second"] / 1000 /
1000
        rcvd_speed = obj["end"]["sum_received"]["bits_per_second"] / 1000 /
1000



        reverse = obj["start"]["test_start"]["reverse"]
        time = (obj["start"]["timestamp"]["time"]).encode('ascii',
'ignore')
        cookie = (obj["start"]["cookie"]).encode('ascii', 'ignore')
        protocol = (obj["start"]["test_start"]["protocol"]).encode('ascii',
'ignore')
        duration = obj["start"]["test_start"]["duration"]
        num_streams = obj["start"]["test_start"]["num_streams"]
        if reverse not in [0, 1]:
            sys.exit("unknown reverse")

        s = 0
        r = 0
        if ip in db:
            (s, r) = db[ip]

        if reverse == 0:
            r += rcvd
            sent = 0
            sent_speed = 0
        else:
            s += sent
            rcvd = 0
            rcvd_speed = 0

        db[ip] = (s, r)

        csvwriter.writerow([time, ip, local_port, remote_port, duration,
protocol, num_streams, cookie, sent, sent_speed, rcvd, rcvd_speed, s, r])
        return True
    except:
       eprint("error or bogus test:", sys.exc_info()[0])
       pass
```

```
       return False

def dumpdb(database):
    """ dump db to text """
    for i in database:
        (s, r) = database[i]
        print("%s, %d , %d " % (i, s, r))

if __name__ == '__main__':
    main()
```

Discovering Latent Network Problems
with Automated Bandwidth Testing
July 2024
Author: Jason C. Mitchell

Bigjar Systems, LLC
5305 Village Center Drive
Columbia, MD U.S.A.

Phone: +1.855-4BIGJAR
+1-855 424-4527