APPENDIX B

Buffer Monitoring Code and Configuration Files

The following buffer monitoring code and configuration files are available for consideration:

buffer_check.py

```
#!/usr/bin/python
#
# A script for monitoring buffer utilization on the Cisco Nexus 3000
# platform. Tested with Nexus 3064 and Nexus 3048 switches. Intended
# to be run on the switch. Reports data to Graphite via pickled data
# over TCP (or any other data sink that can read pickle data).
#
# Written by Mark T. Voelker
# Copyright 2012 Cisco Systems, Inc.
#

import os
import sys
import re
import logging
import argparse
import time
```

```
import cPickle
import socket
import struct
import copy
import xml.parsers.expat
from cisco import CLI


def daemonize():
    """
    Daemonizes the process by forking the main execution off
    into the background.
    """
    try:
        pid = os.fork()
    except OSError, e:
        raise OSError("Can't fork(%d): %s" % (e.errno, e.strerror))

    if (pid == 0):
        # This is the child process.
        # Become the session leder/process group leader and ensure
        # that we don't have a controlling terminal.
        os.setsid()

        # Now do some work.
    else:
        # This is the parent.
        # Write the pid of the child before we quit.
        write_pidfile(pid)
        global logger
        logger.debug(
            "Parent (%d) spawned child (%d) successfully" % (os.getpid(), pid)
            )
        exit(0)


def write_pidfile(pid=os.getpid()):
    """
    Writes a pid file to /bootflash/buffer_check.py.pid.
    The file contains one line with the PID.
    """
    global args
    f = open(args.pidfile, 'w')
    f.write(str(pid))
    f.close()


def set_exit_code(value, current_code):
    """
    Returns an exit code taking into account any previous conditions
    that set an exit code.
    """
    if exit_code >= 2:
        # Nothing can change this.
        logger.debug("exit code is already set to 2")
        return 2
    else:
        logger.debug("exit code set to %s" % (current_code))
        return current_code


def start_element(name, attrs):
    """
    Callback routine to handle the start of a tag.
```

```
    """
    global current_tag
    current_tag = copy.copy(name)
    #logger.debug("Current tag: '%s'" % (current_tag))


def end_element(name):
    """
    Callback routine for handling the end of a tagged element.
    """
    global current_tag
    current_tag = ''


def char_data(data):
    """
    Callback routine to handle data within a tag.
    """
    global current_tag
    global current_int
    global parsed_data
    #logger.debug("char_data handler called [current_tag = %s] on '%s'" % (
    #    current_tag, data)
    #    )
    if current_tag == 'total_instant_usage':
        parsed_data['instant_cell_usage'] = int(copy.copy(data))
        logger.debug("FOUND TOTAL INSTANT CELL USAGE: %s" % (data))
    elif current_tag == 'max_cell_usage':
        parsed_data['max_cell_usage'] = int(copy.copy(data))
        logger.debug("FOUND TOTAL MAX CELL USAGE: %s" % (data))
    elif current_tag == 'rem_instant_usage':
        parsed_data['rem_instant_usage'] = int(copy.copy(data))
        logger.debug("FOUND REMAINING INSTANT USAGE: %s" % (data))
    elif current_tag == 'front_port':
        current_int = int(copy.copy(data))
        parsed_data[current_int] = 0
        logger.debug("Started a new front port: %s" % (data))
    elif re.search('^[m|u]cast_count_\d$', current_tag):
        logger.debug("Found queue counter (port %s): %s" % (current_int, data))
        if current_int in parsed_data:
            parsed_data[current_int] += int(copy.copy(data))
        else:
            parsed_data[current_int] = int(copy.copy(data))
        logger.debug("Added %s to counter for port %s (total: %s)" % (
            data, current_int, parsed_data[current_int])
            )


def int_char_data(data):
    """
    Callback routine to handle data within a tag.
    """
    global interface_rates
    global current_tag
    global current_int
    global get_cmd_timestamp
    global pickle_data
    global logger

    # List of the tags we care about.
    keepers = ['eth_outbytes', 'eth_inbytes', 'eth_outpkts', 'eth_inpkts']

    if current_tag in keepers:
        # Set up some data storage.
```

```
logger.debug("Working on %s for %s..." % (current_tag, current_int))
if current_int not in interface_rates:
    interface_rates[current_int] = dict()
    logger.debug("   allocating space for %s" % (current_int))
if 'last' not in interface_rates[current_int]:
    interface_rates[current_int]['last'] = dict()
    interface_rates[current_int]['last']['timestamp'] = 0.0
    logger.debug("Initializing %s last timestamp to 0." % (
        current_int))

# Before we start working on the data, go ahead and pickle
# the raw counter stat.
pickle_data.append(
    tuple(['iface_%s.1-%s.%s' % (
        current_tag, current_int, hostname),
        tuple([get_cmd_timestamp, data])]))
logger.debug("Pickled iface_%s.1-%s.%s: %s" % (
       current_tag, current_int, hostname, data
       ))

# Make sure we're set up to hold this data properly.
if current_tag not in interface_rates[current_int]['last']:
    interface_rates[current_int]['last'][current_tag] = 0

# Calculate the rates of change.
logger.debug("Calculating rate for %s/%s using (%s-%s)/(%s-%s)" % (
    current_tag, current_int, data,
    interface_rates[current_int]['last'][current_tag],
    get_cmd_timestamp,
    interface_rates[current_int]['last']['timestamp']
    ))
rate = (float(data) - \
    int(interface_rates[current_int]['last'][current_tag])) / \
    (get_cmd_timestamp - \
    interface_rates[current_int]['last']['timestamp'])
pickle_data.append(
    tuple(['iface_%s_rate.1-%s.%s' % (
        current_tag, current_int, hostname),
        tuple([get_cmd_timestamp, rate])]))
logger.debug("Pickled iface_%s_rate.1-%s.%s: %s" % (
   current_tag, current_int, hostname, rate
   ))

# Per user request, we convert byte rates to bit rates.
if re.search('bytes$', current_tag):
    logger.debug(
        "Calculating bitrate for %s/%s using (%s-%s)/(%s-%s)*8" % (
        current_tag, current_int, data,
        interface_rates[current_int]['last'][current_tag],
        get_cmd_timestamp,
        interface_rates[current_int]['last']['timestamp']
        ))
    rate = (float(data) - \
        int(interface_rates[current_int]['last'][current_tag])) / \
        (get_cmd_timestamp - \
        interface_rates[current_int]['last']['timestamp']) * 8
    bitname = copy.copy(current_tag)
    bitname = re.sub('bytes$', 'bits', bitname)
    pickle_data.append(
        tuple(['iface_%s_rate.1-%s.%s' % (
            bitname, current_int, hostname),
            tuple([get_cmd_timestamp, rate])]))
    logger.debug("Pickled bitrate iface_%s_rate.1-%s.%s: %s" % (
       bitname, current_int, hostname, rate
```

```
                ))

            # Now store the current data.
            interface_rates[current_int]['last'][current_tag] = \
                int(copy.copy(data))


def get_show_queuing_int():
    """
    Parses output from 'show queuing interface' and reports stats.
    Unicast drop stats are reported for each interface given in the
    list of interfaces on the command line.  Drop stats for multicast,
    unicast, xon, and xoff are added up for all interfaces (including
    those not specified on the command line) to provide switch-level
    totals for each.

    Note that there is no XML output for 'show queuing interface' at
    present, so we're forced to parse plaintext from the CLI.  XML
    output does exist for 'show queuing interface x/y | xml', however
    this would require issuing one command for each interface on the box
    since we need to provide switch-level totals.  As this would be
    a performance bottleneck due to the number of commands to be issued
    and parsed, we've avoided that approach here.
    """
    global pickle_data
    global logger
    global args

    logger.debug("Issuing 'show queuing interface' command...")
    get_cmd_timestamp = time.time()
    cli_obj = CLI('show queuing interface', False)
    cli_output = cli_obj.get_output()

    # As we parse, remember what interface we're working with.
    current_int = ''

    # Set up switch-level total counters.
    switch_counters = dict()
    switch_counters['ucast_pkts_dropped'] = 0
    switch_counters['ucast_bytes_dropped'] = 0
    switch_counters['mcast_pkts_dropped'] = 0
    switch_counters['mcast_bytes_dropped'] = 0
    switch_counters['xon'] = 0
    switch_counters['xoff'] = 0

    for line in cli_output:
        match = re.match('Ethernet(\d+\/\d+) queuing information', line)
        if match:
            current_int = match.group(1).replace('/', '-')
            logger.debug("Working on queuing stat for int %s" % (current_int))
            continue
        match = re.search('drop-type:\s+drop,\s+xon:\s*(\d+),\s+xoff:\s*(\d+)',
            line)
        if match:
            # As of this revision, we don't collect individual
            # interface counters per interface.
            switch_counters['xon'] += int(match.group(1))
            switch_counters['xoff'] += int(match.group(2))
            continue
        match = re.search('([UM]cast) (pkts|bytes) dropped\s+:\s*(\d+)', line)
        if match:
            stat_name = "%s_%s_dropped.%s.%s" % (
                match.group(1).lower(), match.group(2).lower(),
                current_int, hostname
```

```
                    )
                switch_stat_name = "%s_%s_dropped" % (match.group(1).lower(),
                    match.group(2).lower())

                # If it's a unicast stat and this interface is
                # in the list given in the CLI, pickle it.
                if re.match('ucast_', stat_name):
                    int_on_lc = re.match('\d+\-(\d+)', current_int)
                    int_on_lc = int(int_on_lc.group(1))
                    if int_on_lc in args.interfaces:
                        pickle_data.append(
                            tuple([
                                stat_name,
                                tuple(
                                    [get_cmd_timestamp,
                                    int(match.group(3))])]))
                        logger.debug("Pickled %s: %s" % (stat_name,
                            match.group(3)))

                # Add to our switch-level counters.
                switch_counters[switch_stat_name] += int(match.group(3))

        # Output parsing complete...pickle the switch-level stats.
        for stat_name in switch_counters:
            pickle_data.append(
                tuple([
                    stat_name + '.' + hostname,
                    tuple([get_cmd_timestamp, switch_counters[stat_name]])]))
            logger.debug("Pickled %s.%s: %s" % (stat_name, hostname,
                switch_counters[stat_name]))


def get_int_counters():
    """
    Parses stats from the output of 'show interface x/y | xml'.
    """
    global args
    global pickle_data
    global logger
    global interface_rates
    global current_int
    global current_tag
    global get_cmd_timestamp

    # Sift through each interface.
    for port_num in args.interfaces:
        # Now handle any interface-specific counters for this port.
        try:
            current_int = port_num
            get_cmd_timestamp = time.time()
            cli_obj = CLI("show int e1/%s | xml" % (port_num), False)

            # Get the reply.
            get_cmd_reply = cli_obj.get_raw_output()
            logger.debug("-----\nReply received:\n-----" + str(get_cmd_reply))

            # Clean off trailing junk...is this an NX-OS bug?
            get_cmd_reply = get_cmd_reply.rstrip(">]\n") + '>'

            # Set up an XML parser and parse.
            int_xml_parser = xml.parsers.expat.ParserCreate()
            int_xml_parser.StartElementHandler = start_element
            int_xml_parser.EndElementHandler = end_element
            int_xml_parser.CharacterDataHandler = int_char_data
```

```
            int_xml_parser.Parse(get_cmd_reply, 1)

            # Remember the timestamp of this command for the next go around.
            if port_num not in interface_rates:
                interface_rates[port_num] = dict()
            if 'last' not in interface_rates[port_num]:
                interface_rates[port_num]['last'] = dict()
                logger.debug("Initializing %s last dict for " % (
                    port_num))
            interface_rates[port_num]['last']['timestamp'] = \
                copy.copy(get_cmd_timestamp)
            logger.debug("Set last timestamp for %s to %s" % (
                port_num, get_cmd_timestamp))
        except SyntaxError:
            print """
    WARNING: can't get output for interface 1/%s.  Does it exist?
    """ % (port_num)


def get_buffer_stats():
    """
    Parses stats from the output of 'show hardware internal buffer pkt-stats detail |
xml'.
    """
    global args
    global pickle_data
    global logger
    global interface_rates
    global exit_code

    # Frame up the command snippet we need to send to the switch.
    get_message = "show hardware internal buffer info pkt-stats detail | xml"

    # Set up the CLI object and issue the command.
    get_cmd_timestamp = time.time()
    cli_obj = CLI(get_message, False)

    # Before we process the reply, send another message to clear the counters
    # unless we've been told not to do so.
    if args.clear_counters:
        clear_obj = CLI(clear_message)
        clear_cmd_reply = clear_obj.get_raw_output()
        logger.debug("Result of clear command:\n%s" % (clear_cmd_reply))

    # Parse the reply.
    get_cmd_reply = cli_obj.get_raw_output()
    logger.debug("-----\nReply received:\n-----" + str(get_cmd_reply))

    # Clean off trailing junk...is this an NX-OS bug?
    get_cmd_reply = get_cmd_reply.rstrip(">]\n") + '>'

    # Start up an expat parser to quickly grock the XML.
    xml_parser = xml.parsers.expat.ParserCreate()
    xml_parser.StartElementHandler = start_element
    xml_parser.EndElementHandler = end_element
    xml_parser.CharacterDataHandler = char_data
    xml_parser.Parse(get_cmd_reply, 1)

    # Form a pickle-protocol data structure.
    output_string = ""

    # Pickle max buffer usage if necessary.
    if args.get_max_buf:
        logger.debug("Max cell usage is %s" % (parsed_data['max_cell_usage']))
```

```
        output_string += "Max cell usage: %s" % (parsed_data['max_cell_usage'])
        pickle_data.append(tuple(['max_cell_usage.%s' % (hostname),
            tuple([get_cmd_timestamp, parsed_data['max_cell_usage']])])
            )
        exit_code = set_exit_code(
            int(parsed_data['max_cell_usage']), exit_code
            )

    # Now do instant cell usage.
    if args.get_instant_buf:
        logger.debug("Instant cell usage is %s" % (
            parsed_data['instant_cell_usage'])
            )
        if args.get_max_buf:
            output_string += ', '
        output_string += "Instant cell usage: %s" % (
            parsed_data['instant_cell_usage']
            )
        pickle_data.append(tuple(['instant_cell_usage.%s' % (hostname),
            tuple([get_cmd_timestamp, parsed_data['instant_cell_usage']])]))
            )
        exit_code = set_exit_code(
            int(parsed_data['instant_cell_usage']), exit_code
            )

    # Now get per-port stats.  We add together each of the
    # 8 buffer queues for simplicity here, if that doesn't
    # suit your purposes please feel free to modify.
    for port_num in args.interfaces:
        if int(port_num) in parsed_data:
            # Pickle that port data.
            pickle_data.append(
                tuple(
                    ['iface_instant_cell_usage.1-%s.%s' % (port_num, hostname),
                tuple([get_cmd_timestamp, int(parsed_data[int(port_num)])])])
                )
            logger.debug("Pickled instant cell usage for 1/%s: %s" % (
                port_num, parsed_data[int(port_num)]
                ))

            # We're also doing a metric for the percentage of
            # alpha threshhold used.  In a nutshell, a packet
            # is only admitted to the buffer if an threshhold is
            # not exceeded.  The threshhold is the remaining
            # instant usage (taken from the <rem_instant_usage> tag
            # in the output we parsed) times 2.  Because this threhshold
            # is dependent on how much buffer is actually in use at
            # any given time, we graph the current buffer utilization
            # on the port as a percentage of the threshhold.  When
            # we hit 100%, no more packets will be admitted to the buffer
            # on this port even if there is buffer available on the box.
            percent_used = float(parsed_data[int(port_num)]) / (
                int(parsed_data['rem_instant_usage']) * 2) * 100
            pickle_data.append(
                tuple([
                    'percent_buf_threshold.1-%s.%s' % (port_num, hostname),
                    tuple([get_cmd_timestamp, percent_used])
                    ]))
            logger.debug("Pickled percent of threshhold for 1/%s: %f" % (
                port_num, percent_used
                ))
        else:
            print """
    WARNING: requested interface %s not found in command output.
```

```
    """ % (port_num)


def do_switch_commands():
    """
    A hook function for executing any switch-level command necessary.
    Commands for individual interfaces are handled elsewhere.
    """
    global args
    # TODO (mvoelker): add CLI options here to determine which
    # commands get run.
    if args.get_queuing_stats:
        get_show_queuing_int()
    if args.get_buffer_stats:
        get_buffer_stats()


def do_interface_commands():
    """
    A hook function for executing any per-interface command necessary.
    Commands for handling switch-level stats and commands which
    provide data for multiple interfaces are generally handled in
    do_switch_commands().
    """
    global args
    if args.get_int_counters:
        get_int_counters()

# Provide usage and parse command line options.
usage = "\n%prog [options] [arg1 arg2 ...]"
usage += """

Arguments are the numbers of the ports you want to collect
buffer queue stats for.  If unspecified, no per-port stats
will be displayed.

This script is intended to be run on a Cisco Nexus 3000-series switch
(tested on 3064 and 3048 models).  It can be run manually or via
the NX-OS scheduler.  It will report buffer utilization stats
parsed from the output of "show hardware internal buffer pkt-stats detail"
via the pickle protocol over TCP to Graphite (or another data
sink of your choice that can grock pickled data).

Example:
%prog -H myN3K.mydomain.com -l admin -p password \\
   -m -i 46 47 48
"""
parser = argparse.ArgumentParser(description=usage)
parser.add_argument("-H", "--hostname", dest="hostname",
    help="Hostname or IP address", required=True)
parser.add_argument("-p", "--pidfile", dest="pidfile",
    help="File in which to write our PID", default="/bootflash/buffer_check.py.pid")
parser.add_argument("-v", "--verbose", dest="verbosity", action="count",
    help="Enable verbose output.", default=0)
parser.add_argument("-b", "--clear_buffer_counters", dest="clear_counters",
    help="Clear buffer counters after checking", default=False,
    action="store_true")
parser.add_argument("-m", "--max_buffer", dest="get_max_buf",
    help="Show max buffer utilization", default=False,
    action="store_true")
parser.add_argument("-i", "--instant_buffer", dest="get_instant_buf",
    help="Show instant buffer utilization", default=False,
    action="store_true")
parser.add_argument("interfaces", metavar="N", type=int, nargs='*',
```

```
    help='List of interfaces to check.')
parser.add_argument("-s", "--sleep_interval", dest="sleep_interval",
    help="Interval to sleep between polls (higher reduces CPU hit)",
    type=float, default=0)
parser.add_argument("-q", "--queuing_stats", dest="get_queuing_stats",
    help="Get stats from 'show queuing interface'", default=False,
    action="store_true")
parser.add_argument("-c", "--interface_counters", dest="get_int_counters",
    help="Get stats from 'show interface x/x'", default=False,
    action="store_true")
parser.add_argument("-f", "--buffer_stats", dest="get_buffer_stats",
    help="Get stats from 'show hardware internal buffer pkts-stats detail'",
    default=False, action="store_true")
args = parser.parse_args()

# Set up a logger.
logger = logging.getLogger('n3k_buffer_check')
logging.basicConfig()

# Since this started out purely as a script for buffer monitoring commands,
# certain command options imply others.  Fix things up here.
if args.get_instant_buf:
    args.get_buffer_stats = True
    logger.debug("CLI: assuming -f because I received -i.")
if args.get_max_buf:
    args.get_buffer_stats = True
    logger.debug("CLI: assuming -f because I received -m.")
if args.clear_counters:
    args.get_buffer_stats = True
    logger.debug("CLI: assuming -f because I received -b.")

# Daemonize ourself if we've gotten this far.
daemonize()

# Set the hostname.
hostname = socket.gethostname()

# If we're doing verbose output, set that up.
if args.verbosity == 3:
    # Hmm...do....something?
    logger.setLevel(logging.DEBUG)
if args.verbosity >= 2:
    # Hmm...do...something else?
    logger.setLevel(logging.DEBUG)
if args.verbosity >= 1:
    logger.setLevel(logging.DEBUG)
if args.verbosity == 0:
    # Redirect standard I/O streams to /dev/null.
    os.close(0)
    os.close(1)
    os.close(2)
    os.open(os.devnull, os.O_RDWR)
    os.dup2(0, 1)
    os.dup2(0, 2)

# Add a message for clearing the counters.
clear_message = "clear counters buffers"

# Set up some data holders to be used by XML parsing callback routines.
parsed_data = dict()
interface_rates = dict()
current_int = 0
current_tag = ''
charbuff = ''
```

```
port_num = 0
get_cmd_timestamp = 0.0

# A place to hold data we'll send back over the wire.
pickle_data = list()

# Set up a default exit code.
exit_code = 0

# Start up a socket over which to send data.
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect((args.hostname, 2004))

while True:
    # Clear out old pickled data.
    pickle_data = list()

    # If we have other interface-level commands to do, do them
    # here.
    logger.debug("Doing interface-level commands...")
    do_interface_commands()

    # If we have other switch-level commands to do, do them here.
    logger.debug("Doing switch-level commands...")
    do_switch_commands()

    if args.verbosity > 0:
        logger.debug(pickle_data)

    # Pickle the data.
    payload = cPickle.dumps(pickle_data)
    logger.debug("Size of picked data: %s" % sys.getsizeof(payload))
    header = struct.pack("!L", len(payload))
    message = header + payload

    # Batch the data off to Graphite.
    # Unfortunately Carbon doesn't listen for pickle data on UDP
    # sockets. =(  If we can fix that, uncomment the next two lines
    # and comment out the three after that to use UDP instead of TCP.
    #sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
    #sock.sendto(message, (args.hostname, 2004))
    sock.sendall(message)

    # Go to sleep if we've been told to do so.
    time.sleep(args.sleep_interval)
```

check_process.py

```
#!/usr/bin/python
import os
import sys
import re
import argparse
from cisco import CLI

usage = "\n%prog [options]"
usage += """

This script checks to see if the buffer_check.py script
is running by reading it's pidfile and pinging the listed pid.
If buffer_check.py isn't running, this script will start it.
```

```
If buffer_check.py is running, this script can optionally
kill it if run with the -k option.

Example:
%prog -k
"""

parser = argparse.ArgumentParser(description=usage)
parser.add_argument("-k", "--kill", dest="kill",
    help="Kill buffer_check.py if running", default=False,
    action="store_true")
args = parser.parse_args()


def start_process():
    """
    Starts the buffer_check.py script.  For our implementation, we
    start three instances: one to check buffer stats, one to check
    interface stats on server-facing ports, and one to check interface
    stats on other ports and queuing stats (at lower granularity).
    """

    cmd = 'python bootflash:buffer_check.py -H 172.18.117.181 -m -i -b -s 0.8 ' \
        '-f -p /bootflash/buffer_check.py.pid 1 2 3 4 5 6 7 8 17 18 19 20 21 ' \
        '33 34 35 36 37 46 47 48 57 58 59 60 61 62 63 64 '
    cli_obj = CLI(cmd, False)
    print "Started %s" % (cmd)

    cmd = 'python bootflash:buffer_check.py -H 172.18.117.181 -c -s 0.8 ' \
        '-p /bootflash/buffer_check.py-1.pid 33 34 35 36 37'
    cli_obj = CLI(cmd, False)
    print "Started %s" % (cmd)

    cmd = 'python bootflash:buffer_check.py -H 172.18.117.181 -c -q -s 5 ' \
        '-p /bootflash/buffer_check.py-2.pid 1 2 3 4 5 6 7 8 17 18 19 20 21' \
        '33 34 35 36 37 46 47 48 57 58 59 60 61 62 63 64 '
    cli_obj = CLI(cmd, False)
    print "Started %s" % (cmd)


def check_pid():
    """
    Checks to see if the buffer_check.py script is running.
    """

    pidfiles = ['/bootflash/buffer_check.py.pid', '/bootflash/buffer_check.py-1.pid',
                '/bootflash/buffer_check.py-2.pid']

    retval = True

    for pf in pidfiles:
        # Try to open our pidfile.
        try:
            f = open(pf, 'r')
        except IOError:
            print "No pidfile %s found!" % (pf)
            retval = False

        # Read the pid from the file and grock it down to an int.
        pid = f.readline()
        pidmatch = re.search('^(\d+)\s*$', pid)
        if pidmatch:
            pid = pidmatch.group(1)
```

```
            print "Pid from pidfile is %s" % (pid)
            global options
            try:
                if args.kill:
                    os.kill(int(pid), 9)
                    print "Killed %s" % (pid)
                else:
#!/usr/bin/python
import os
import sys
import re
import argparse
from cisco import CLI

usage = "\n%prog [options]"
usage += """

This script checks to see if the buffer_check.py script
is running by reading it's pidfile and pinging the listed pid.
If buffer_check.py isn't running, this script will start it.

If buffer_check.py is running, this script can optionally
kill it if run with the -k option.

Example:
%prog -k
"""

parser = argparse.ArgumentParser(description=usage)
parser.add_argument("-k", "--kill", dest="kill",
    help="Kill buffer_check.py if running", default=False,
    action="store_true")
args = parser.parse_args()


def start_process():
    """
    Starts the buffer_check.py script.  For our implementation, we
    start three instances: one to check buffer stats, one to check
    interface stats on server-facing ports, and one to check interface
    stats on other ports and queuing stats (at lower granularity).
    """

    cmd = 'python bootflash:buffer_check.py -H 172.18.117.181 -m -i -b -s 0.8 ' \
        '-f -p /bootflash/buffer_check.py.pid 1 2 3 4 5 6 7 8 17 18 19 20 21 ' \
        '33 34 35 36 37 46 47 48 57 58 59 60 61 62 63 64 '
    cli_obj = CLI(cmd, False)
    print "Started %s" % (cmd)

    cmd = 'python bootflash:buffer_check.py -H 172.18.117.181 -c -s 0.8 ' \
        '-p /bootflash/buffer_check.py-1.pid 33 34 35 36 37'
    cli_obj = CLI(cmd, False)
    print "Started %s" % (cmd)

    cmd = 'python bootflash:buffer_check.py -H 172.18.117.181 -c -q -s 5 ' \
        '-p /bootflash/buffer_check.py-2.pid 1 2 3 4 5 6 7 8 17 18 19 20 21' \
        '33 34 35 36 37 46 47 48 57 58 59 60 61 62 63 64 '
    cli_obj = CLI(cmd, False)
    print "Started %s" % (cmd)


def check_pid():
    """
    Checks to see if the buffer_check.py script is running.
```

```
    """

    pidfiles = ['/bootflash/buffer_check.py.pid', '/bootflash/buffer_check.py-1.pid',
                '/bootflash/buffer_check.py-2.pid']

    retval = True

    for pf in pidfiles:
        # Try to open our pidfile.
        try:
            f = open(pf, 'r')
        except IOError:
            print "No pidfile %s found!" % (pf)
            retval = False

        # Read the pid from the file and grock it down to an int.
        pid = f.readline()
        pidmatch = re.search('^(\d+)\s*$', pid)
        if pidmatch:
            pid = pidmatch.group(1)
            print "Pid from pidfile is %s" % (pid)
            global options
            try:
                if args.kill:
                    os.kill(int(pid), 9)
                    print "Killed %s" % (pid)
                else:
                    os.kill(int(pid), 0)
            except OSError:
                print "%s is dead." % (pid)
                retval = False
            else:
                if not args.kill:
                    print "%s is alive." % (pid)
        else:
            print "No pid found!"
            retval = False

    return retval

if check_pid():
    # We can exit, the scripts are running.
    exit(0)
else:
    # We need to start the scripts.
    if not args.kill:
        start_process()
    exit(1)
```

NX-OS Scheduler Example

```
milliways-3k-1# show scheduler config
config terminal
  feature scheduler
  scheduler logfile size 16
end


config terminal
 scheduler job name buffer_check
python bootflash:/check_process.py
```

```
 show scheduler config

end

config terminal
  scheduler schedule name every_minute
    time start 2012:09:10:09:58 repeat 1
```

Collectd Configuration

The following collectd configurations are available for consideration:

- collectd.conf, page B-15
- Puppet Manifest, page B-16

collectd.conf

```
BaseDir     "/var/lib/collectd"
PIDFile     "/var/run/collectd.pid"
PluginDir   "/usr/lib64/collectd"
TypesDB     "/usr/share/collectd/types.db"
Interval     1
ReadThreads  5
LoadPlugin syslog
LoadPlugin cpu
LoadPlugin disk
LoadPlugin ethstat
LoadPlugin libvirt
LoadPlugin load
LoadPlugin memory
LoadPlugin write_graphite
<Plugin disk>
    Disk "/^[hs]d[a-f][0-9]?$/"
    IgnoreSelected false
</Plugin>

<Plugin ethstat>
    Interface "eth0"
        Interface "eth1"
        Map "rx_packets" "pkt_counters" "rx_packets"
        Map "tx_packets" "pkt_counters" "tx_packets"
        Map "rx_bytes" "byte_counters" "rx_bytes"
        Map "tx_bytes" "byte_counters" "tx_bytes"
        Map "rx_errors" "error_counters" "rx_errors"
        Map "tx_errors" "error_counters" "tx_errors"
        Map "rx_dropped" "drop_counters" "rx_dropped"
        Map "tx_dropped" "drop_counters" "tx_dropped"
        Map "collisions" "error_counters" "collisions"
        Map "rx_over_errors" "error_counters" "rx_over_errors"
        Map "rx_crc_errors" "error_counters" "rx_crc_errors"
        Map "rx_frame_errors" "error_counters" "rx_frame_errors"
        Map "rx_fifo_errors" "error_counters" "rx_fifo_errors"
        Map "rx_missed_errors" "error_counters" "rx_missed_errors"
        Map "tx_aborted_errors" "error_counters" "tx_aborted_errors"
        Map "tx_carrier_errors" "error_counters" "tx_carrier_errors"
        Map "tx_fifo_errors" "error_counters" "tx_fifo_errors"
        Map "tx_heartbeat_errors" "error_counters" "tx_heartbeat_errors"
        Map "rx_pkts_nic" "pkt_counters" "rx_pkts_nic"
```

```
        Map "tx_pkts_nic" "pkt_counters" "tx_pkts_nic"
        Map "rx_bytes_nic" "byte_counters" "rx_bytes_nic"
        Map "tx_bytes_nic" "byte_counters" "tx_bytes_nic"
        Map "lsc_int" "misc_counters" "lsc_int"
        Map "tx_busy" "error_counters" "tx_busy"
        Map "non_eop_descs" "misc_counters" "non_eop_descs"
        Map "broadcast" "pkt_counters" "broadcast"
        Map "rx_no_buffer_count" "error_counters" "rx_no_buffer_count"
        Map "tx_timeout_count" "error_counters" "tx_timeout_count"
        Map "tx_restart_queue" "error_counters" "tx_restart_queue"
        Map "rx_long_length_errors" "error_counters" "rx_long_length_errors"
        Map "rx_short_length_errors" "error_counters" "rx_short_length_errors"
        Map "tx_flow_control_xon" "misc_counters" "tx_flow_control_xon"
        Map "rx_flow_control_xon" "misc_counters" "rx_flow_control_xon"
        Map "tx_flow_control_xoff" "misc_counters" "tx_flow_control_xoff"
        Map "rx_flow_control_xoff" "misc_counters" "rx_flow_control_xoff"
        Map "rx_csum_offload_errors" "error_counters" "rx_csum_offload_errors"
        Map "alloc_rx_page_failed" "error_counters" "alloc_rx_page_failed"
        Map "alloc_rx_buff_failed" "error_counters" "alloc_rx_buff_failed"
        Map "rx_no_dma_resources" "error_counters" "rx_no_dma_resources"
        Map "hw_rsc_aggregated" "misc_counters" "hw_rsc_aggregated"
        Map "hw_rsc_flushed" "misc_counters" "hw_rsc_flushed"
    MappedOnly true
</Plugin>

<Plugin libvirt>
    Connection "qemu:///system"
    RefreshInterval 5
    IgnoreSelected false
          HostnameFormat hostname name
</Plugin>

<Plugin write_graphite>
  <Carbon>
    Host "voyager-graphite.hosts.voyager.cisco.com"
    Port "2003"
    Prefix "collectd"
    Postfix "collectd"
    StoreRates false
    AlwaysAppendDS false
    EscapeCharacter "_"
  </Carbon>
</Plugin>

Include "/etc/collectd.d"
```

Puppet Manifest

```
class collectd {

    package { "collectd":
       name     => "collectd",
       ensure   => 'latest',
       require  => [File['/etc/yum.conf']]
    }

    package { "collectd-graphite":
       name     => "collectd-graphite",
       ensure   => 'latest',
       require  => [File['/etc/yum.conf']]
    }
```

```
    package { "collectd-ethstat":
       name     => "collectd-ethstat",
       ensure   => 'latest',
       require  => [File['/etc/yum.conf']]
    }

    package { "collectd-libvirt":
       name     => "collectd-libvirt",
       ensure   => 'latest',
       require  => [File['/etc/yum.conf']]
    }

    service { "collectd":
        enable  => 'true',
        ensure  => 'running',
        start   => '/etc/init.d/collectd start',
        stop    => '/etc/init.d/collectd stop',
        require => [Package['collectd'], Package['collectd-graphite'],
Package['collectd-ethstat'], File['/etc/collectd.conf']]
    }

    if $fqdn =~ /^r05+-p0[1-5]\.hosts\.voyager\.cisco\.com$/ {
        file { '/etc/collectd.conf':
            #source  => 'puppet:///modules/collectd/collectd.conf.enabled',
            source  => 'puppet:///modules/collectd/collectd.conf',
            owner   => 'root',
            group   => 'root',
            mode    => '644',
            notify  => Service['collectd'],
            require => Package['collectd']
        }
    } else {
        file { '/etc/collectd.conf':
            source  => 'puppet:///modules/collectd/collectd.conf',
            owner   => 'root',
            group   => 'root',
            mode    => '644',
            notify  => Service['collectd'],
            require => Package['collectd']
        }
    }
}
```

Graphite Configuration

The following graphite configurations are available for consideration:

carbon.conf

```
[cache]
USER =
MAX_CACHE_SIZE = inf
MAX_UPDATES_PER_SECOND = 50000
MAX_CREATES_PER_MINUTE = 500
LINE_RECEIVER_INTERFACE = 0.0.0.0
LINE_RECEIVER_PORT = 2103
ENABLE_UDP_LISTENER = True
UDP_RECEIVER_INTERFACE = 0.0.0.0
UDP_RECEIVER_PORT = 2103
PICKLE_RECEIVER_INTERFACE = 0.0.0.0
PICKLE_RECEIVER_PORT = 2104
USE_INSECURE_UNPICKLER = False
CACHE_QUERY_INTERFACE = 0.0.0.0
CACHE_QUERY_PORT = 7102
USE_FLOW_CONTROL = True
LOG_UPDATES = False
WHISPER_AUTOFLUSH = False

[cache:b]
LINE_RECEIVER_PORT = 2203
PICKLE_RECEIVER_PORT = 2204
CACHE_QUERY_PORT = 7202
UDP_RECEIVER_PORT = 2203

[cache:c]
LINE_RECEIVER_PORT = 2303
PICKLE_RECEIVER_PORT = 2304
CACHE_QUERY_PORT = 7302
UDP_RECEIVER_PORT = 2303

[cache:d]
LINE_RECEIVER_PORT = 2403
PICKLE_RECEIVER_PORT = 2404
CACHE_QUERY_PORT = 7402
UDP_RECEIVER_PORT = 2403

[cache:e]
LINE_RECEIVER_PORT = 2503
PICKLE_RECEIVER_PORT = 2504
CACHE_QUERY_PORT = 7502
UDP_RECEIVER_PORT = 2503

[cache:f]
LINE_RECEIVER_PORT = 2603
PICKLE_RECEIVER_PORT = 2604
CACHE_QUERY_PORT = 7602
UDP_RECEIVER_PORT = 2603

[cache:g]
LINE_RECEIVER_PORT = 2703
PICKLE_RECEIVER_PORT = 2704
CACHE_QUERY_PORT = 7702
UDP_RECEIVER_PORT = 2703

[cache:h]
LINE_RECEIVER_PORT = 2803
PICKLE_RECEIVER_PORT = 2804
CACHE_QUERY_PORT = 7802
UDP_RECEIVER_PORT = 2803
```

```
[cache:i]
LINE_RECEIVER_PORT = 2903
PICKLE_RECEIVER_PORT = 2904
CACHE_QUERY_PORT = 7902
UDP_RECEIVER_PORT = 2903

[cache:j]
LINE_RECEIVER_PORT = 3003
PICKLE_RECEIVER_PORT = 3004
CACHE_QUERY_PORT = 8002
UDP_RECEIVER_PORT = 3003

[relay]
LINE_RECEIVER_INTERFACE = 0.0.0.0
LINE_RECEIVER_PORT = 2003
PICKLE_RECEIVER_INTERFACE = 0.0.0.0
PICKLE_RECEIVER_PORT = 2004
RELAY_METHOD = rules
REPLICATION_FACTOR = 1
DESTINATIONS = 127.0.0.1:2104:a, 127.0.0.1:2204:b, 127.0.0.1:2304:c, 127.0.0.1:2404:d,
127.0.0.1:2504:e, 127.0.0.1:2604:f, 127.0.0.1:2704:g, 127.0.0.1:2804:h,
127.0.0.1:2904:i, 127.0.0.1:3004:j
MAX_DATAPOINTS_PER_MESSAGE = 500
MAX_QUEUE_SIZE = 10000
USE_FLOW_CONTROL = True

[aggregator]
LINE_RECEIVER_INTERFACE = 0.0.0.0
LINE_RECEIVER_PORT = 2023
PICKLE_RECEIVER_INTERFACE = 0.0.0.0
PICKLE_RECEIVER_PORT = 2024
DESTINATIONS = 127.0.0.1:2004
REPLICATION_FACTOR = 1
MAX_QUEUE_SIZE = 10000
USE_FLOW_CONTROL = True
MAX_DATAPOINTS_PER_MESSAGE = 500
MAX_AGGREGATION_INTERVALS = 5
```

graphite.wsgi

```
import os, sys
sys.path.append('/opt/graphite/webapp')
os.environ['DJANGO_SETTINGS_MODULE'] = 'graphite.settings'
import django.core.handlers.wsgi
application = django.core.handlers.wsgi.WSGIHandler()
from graphite.logger import log
log.info("graphite.wsgi - pid %d - reloading search index" % os.getpid())
import graphite.metrics.search
```

graphite-vhost.conf

```
<IfModule !wsgi_module.c>
    LoadModule wsgi_module modules/mod_wsgi.so
</IfModule>

WSGISocketPrefix run/wsgi

<VirtualHost *:80>
        ServerName voyager-graphite
```

```
        ServerAlias voyager-graphite.cisco.com
        DocumentRoot "/opt/graphite/webapp"
        ErrorLog /opt/graphite/storage/log/webapp/error.log
        CustomLog /opt/graphite/storage/log/webapp/access.log common

        WSGIDaemonProcess graphite processes=16 threads=16 display-name='%{GROUP}'
inactivity-timeout=120
        WSGIProcessGroup graphite
        WSGIApplicationGroup %{GLOBAL}
        WSGIImportScript /opt/graphite/conf/graphite.wsgi process-group=graphite
application-group=%{GLOBAL}
        WSGIScriptAlias / /opt/graphite/conf/graphite.wsgi

        Alias /content/ /opt/graphite/webapp/content/
        <Location "/content/">
                SetHandler None
        </Location>

        Alias /media/ "@DJANGO_ROOT@/contrib/admin/media/"
        <Location "/media/">
                SetHandler None
        </Location>

        <Directory /opt/graphite/conf/>
                Order deny,allow
                Allow from all
        </Directory>

</VirtualHost>
```

local_settings.py

```
TIME_ZONE = 'America/New_York'
DEBUG = True
USE_LDAP_AUTH = True
LDAP_SERVER = "ldap.cisco.com"
LDAP_PORT = 389
LDAP_SEARCH_BASE = "OU=active,OU=employees,ou=people,o=cisco.com"
LDAP_USER_QUERY = "(uid=%s)"  #For Active Directory use "(sAMAccountName=%s)"
CARBONLINK_HOSTS = ["127.0.0.1:7102:a", "127.0.0.1:7202:b", "127.0.0.1:7302:c",
"127.0.0.1:7402:d", "127.0.0.1:7502:e", "127.0.0.1:7602:f", "127.0.0.1:7702:g",
"127.0.0.1:7802:h", "127.0.0.1:7902:i", "127.0.0.1:8002:j"]
```

relay-rules.conf

```
[collectd01-02]
pattern = collectdr01.*
destinations = 127.0.0.1:2104:a

[collectd03-04]
pattern = collectdr03.*
destinations = 127.0.0.1:2204:b

[collectd05-06]
pattern = collectdr05.*
destinations = 127.0.0.1:2304:c

[collectd07-08]
pattern = collectdr07.*
```

```
destinations = 127.0.0.1:2404:d

[collectd09-10]
pattern = collectdr09.*
destinations = 127.0.0.1:2504:e

[collectd11-12]
pattern = collectdr11.*
destinations = 127.0.0.1:2604:f

[collectd13-14]
pattern = collectdr13.*
destinations = 127.0.0.1:2704:g

[collectd15-16]
pattern = collectdr15.*
destinations = 127.0.0.1:2804:h

[iface_eth_inb]
pattern = iface_eth_inb.*
destinations = 127.0.0.1:2904:i

[iface_eth_inp]
pattern = iface_eth_inp.*
destinations = 127.0.0.1:2904:i

[iface_eth_outb]
pattern = iface_eth_outb.*
destinations = 127.0.0.1:3004:j

[iface_eth_outp]
pattern = iface_eth_outp.*
destinations = 127.0.0.1:3004:j

[max_cell]
pattern = max_cell.*
destinations = 127.0.0.1:2504:e

[instant_cell]
pattern = instant_cell.*
destinations = 127.0.0.1:2604:f

[percent_buf]
pattern = percent_buf.*
destinations = 127.0.0.1:2704:g

[carbon]
pattern = carbon.*
destinations = 127.0.0.1:3004:j

[default]
default = true
destinations = 127.0.0.1:2904:i
```

storage-schemas.conf

```
[carbon]
pattern = ^carbon\.
retentions = 60:90d

[interface_max_buffer]
pattern = ^max_cell_usage*
```

```
retentions = 1s:10d,1m:30d

[interface_instant_buffer]
pattern = ^instant_cell_usage*
retentions = 1s:10d,1m:30d

[interface_percent_threshhold]
pattern = ^iface_instant_cell_usage*
retentions = 1s:10d,1m:30d

[collectd]
pattern = ^collectd*
retentions = 1s:10d,1m:30d

[selective_in_byte_count]
pattern = ^iface_eth_inbytes+?\.1-3\d*
retentions = 1s:10d,1m:30d

[selective_out_byte_count]
pattern = ^iface_eth_outbytes+?\.1-3\d*
retentions = 1s:10d,1m:30d

[selective_in_bit_count]
pattern = ^iface_eth_inbits_rate\.1-3\d*
retentions = 1s:10d,1m:30d

[selective_out_bit_count]
pattern = ^iface_eth_outbits_rate\.1-3\d*
retentions = 1s:10d,1m:30d

[default_1min_for_1day]
pattern = .*
retentions = 10s:10d,1m:30d
```

Puppet Manifest (init.pp)

```
class graphite {
    package { "mod_wsgi":
       name     => "mod_wsgi",
       ensure   => "installed"
    }

    package { "gcc":
       name     => "gcc",
       ensure   => "installed",
    }

    package { "pycairo":
       name     => "pycairo",
       ensure   => 'installed',
    }

    package { "mod_python":
       name     => "mod_python",
       ensure   => 'installed',
    }

    package { "Django":
       name     => "Django",
       ensure   => 'installed',
    }
```

```
    package { "django-tagging":
       name     => "django-tagging",
       ensure   => 'installed',
    }

    package { "python-ldap":
       name     => "python-ldap",
       ensure   => 'installed',
    }

    package { "python-memcached":
       name     => "python-memcached",
       ensure   => 'installed',
    }

    package { "python-sqlite2":
       name     => "python-sqlite2",
       ensure   => 'installed',
    }

    package { "bitmap":
       name     => "bitmap",
       ensure   => 'installed',
    }

    package { "bitmap-fixed-fonts":
       name     => "bitmap-fixed-fonts",
       ensure   => 'installed',
    }

    package { "bitmap-fonts-compat":
       name     => "bitmap-fonts-compat",
       ensure   => 'installed',
    }

    package { "python-devel":
       name     => "python-devel",
       ensure   => 'installed',
    }

    package { "python-crypto":
       name     => "python-crypto",
       ensure   => 'installed',
    }

    package { "pyOpenSSL":
       name     => "pyOpenSSL",
       ensure   => 'installed',
    }

    package { "graphite-web":
       name     => "graphite-web",
       ensure   => 'installed',
       provider => 'pip',
       require  => [Package['pycairo'], Package['mod_python'], Package['Django'],
Package['python-ldap'], Package['python-memcached'], Package['python-sqlite2'],
Package['bitmap'], Package['bitmap-fonts-compat'], Package['bitmap-fixed-fonts']]
    }

    package { "carbon":
       name     => "carbon",
       ensure   => 'installed',
       provider => 'pip',
```

```
       require  => [Package['pycairo'], Package['mod_python'], Package['Django'],
Package['python-ldap'], Package['python-memcached'], Package['python-sqlite2'],
Package['bitmap'], Package['bitmap-fonts-compat'], Package['bitmap-fixed-fonts']]
    }

    package { "whisper":
       name     => "whisper",
       ensure   => 'installed',
       provider => 'pip',
       require  => [Package['pycairo'], Package['mod_python'], Package['Django'],
Package['python-ldap'], Package['python-memcached'], Package['python-sqlite2'],
Package['bitmap'], Package['bitmap-fonts-compat'], Package['bitmap-fixed-fonts']]
    }

    file { '/opt/graphite/conf/carbon.conf':
        source  => 'puppet:///modules/graphite/carbon.conf',
        owner   => 'apache',
        group   => 'root',
        mode    => '644',
        require => Package['carbon']
    }

    file { '/opt/graphite/conf/storage-schemas.conf':
        source  => 'puppet:///modules/graphite/storage-schemas.conf',
        owner   => 'apache',
        group   => 'root',
        mode    => '644',
        require => Package['whisper']
    }

    file { '/opt/graphite/conf/graphite.wsgi':
        source  => 'puppet:///modules/graphite/graphite.wsgi',
        owner   => 'apache',
        group   => 'root',
        mode    => '655',
        require => Package['graphite-web']
    }

    file { '/opt/graphite/webapp/local_settings.py':
        source  => 'puppet:///modules/graphite/local_settings.py',
        owner   => 'apache',
        group   => 'root',
        mode    => '655',
        require => Package['graphite-web']
    }

    file { '/etc/httpd/conf.d/graphite-vhost.conf':
        source  => 'puppet:///modules/graphite/graphite-vhost.conf',
        owner   => 'root',
        group   => 'root',
        mode    => '655',
        require => Package['graphite-web'],
        notify  => Service['httpd']
    }

    service { "httpd":
        enable  => 'true',
        ensure  => 'running',
        start   => '/etc/init.d/httpd start',
        stop    => '/etc/init.d/httpd stop',
        require => [Package['graphite-web'],
File['/etc/httpd/conf.d/graphite-vhost.conf']]
    }
}
```