CESNET Technical Report 29/2004

"Bulk" utility for TCP performance debugging

Pavel Cimbál <xcimbal@cs.felk.cvut.cz>, Czech Technical University,
Sven Ubik, <ubik@cesnet.cz>, CESNET, Prague, Czech Republic

December 14, 2004

Keywords: performance monitoring, end-to-end performance, bandwidth measurement

1 Debugging TCP performance

Low TCP throughput can be caused not only by insufficient available bandwidth in the network, but also by many other aspects in network or end-host behaviour. In order to debug these issues most effectively, we need a detailed short timescale information about current throughput and about the values of runtime TCP variables (e.g., cwnd, ssthresh). Moreover, we need *synchronous* information about various TCP variables that is we need to check values of all these variables at the same time.

In this report we describe how to use our utility called "bulk", which allows synchronous measurement of TCP throughput and monitoring of runtime TCP variables. Bulk does not require root privileges and works with standard Linux kernel.

2 Bulk utility

Bulk is a command-line utility for TCP End-to-End performance analysis. Bulk produces a unidirectional TCP bulk traffic with the possibility of runtime access to the socket internal variables via setsockopt / getsockopt interface in the kernel. This possibility of synchronous and detailed data collecting is essential for further matching between captured flows and the real state of the participating socket.

setsockopt / getsockopt is a standard interface so we can use this utility on remote machines directly and without the necessity for kernel customisation. Bulk uses TCP_INFO socket option, compiled by default in all 2.4.x kernels, for a deeper insight into the TCP internals.

The general syntax is as follows:

```
bulk [optional switches] [host]
```

When no host is specified, bulk enters the server mode and waits for incoming connections. Otherwise, it enters the client mode and tries to connect to the server running at the specified host.

Requests for additional features are specified with optional switches, which must precede the host name. Switches may be used in any order and multiple times. If two switches affect the same parameter, then only the first is valid.

Bulk will produce a flow or a set of flows with desired parameters specified by `-s` switches and it will produce output about desired runtime connection properties specified by `-g` switches. This information can be then matched with the packet dump to check what was the cause of a possible performance problem.

3 Command line switches

The bulk utility accepts a number of command-line switches affecting its behaviour. All switches take the following form:

`-<switchname_letter><additional_parameters>[,<additional_parameters>]`

Additional parameters can contain numbers with decadic, octal or hexadecimal base. Mnemonics, when used instead of numbers, is not case sensitive. The following switches are implemented:

-v Verbose mode. All actions, warnings and messages will be displayed.

-d<number> Test duration in seconds. Has currently no effect for the server mode. Default is 10 seconds.

-b<number> Data block size (the memory area passed to write() / read() calls) in bytes. Default is 65535 bytes.

-c<number> Number of measurements until the utility exits. In multi-mode (see switch `-m`), these measurements can be done in parallel, otherwise they are in a sequence. Default is 1 measurement.

-m Multi-mode is selected. With multi-mode and `-cN`, the sender will open N flows in parallel. The receiver in multi-mode will be to serve up to N flows concurrently. Note that `-m` behaves differently even for N=1, because in the multi-mode the utility always throws all the work to its child processes. This mode is off by default.

-p<number> Chooses another port than the default one. Default port is 3141.

-<s|S><optionname>[,<initialiser1>[,<initialiser2> ...]]

-<s|S><optionlevel>,<optionname>[,¡initialiser1>[,<initialiser2> ...]] Sets the specified socket options. The first form is allowed only when the i mnemonic for `<optionname>` is unambiguous across all known socket levels. The `<initialiser>` is a number, optionally suffixed with a `.<size_specifier_letter>`. As size specifiers, the following letters can be used:

```
4iIdDlL=double word
2sSwW=word
1cCbB=byte
```

Without a size specifier, the number is considered to be long. Examples:

```
-sSO_RCVBUF,49152.4
-sTCP_AIMD,100,75,1,1
```

-<g|G><member1>[,<member2> ...]

-<g|G><optionname>[,<member1>[,<member2> ...]]

-<g|G><optionlevel>,<optionname>[,<member1>[,<member2> ...]] This switch defines the getsockopts() items (members) to be read. When the `<optionname>` or `<optionlevel>` is not suffixed with the size specifier and no member is specified, the switch applies to the member at offset 0 with the size of long. This is useful for pure atomic data types, used in most getsockopt() calls.

Otherwise, the desired members must be specified. All members in one switch must belong to the same option.

The member itself is a byte-offset (number) in the structure, optionally suffixed with the size specifier. These specifiers are the same as for `-s` switch, but may also use `.t<number>` for a textual string expression, containing no more than `<number>` of characters.

Each member can be also suffixed with `%<label>`. The label replaces the standard delimiter in the output with a user-specified text. This text is terminated by ',', '.' or '

The first mentioned form of `-g` can be used when the member name determines both the socket name and level, and is unambiguous across all known socket names and levels.

The second form is allowed only when the mnemonic for `<optionname>` is unambiguous across all known socket levels.

Examples:

```
-gSnd_Cwnd%\ Congestion\ window:\
-gSO_SNDBUF
-gato,rtt,rto
-gTCP_INFO,rcv_ssthresh%Receiver's\ ssthresh:\
```

Hint:

You can use the `-v` switch and then type `?` whenever you are not sure what expression or mnemonic is appropriate. Bulk will show you all possible mnemonics and expressions.

For example, if you want to set some option, but you do not know its name or numbers, you can use `-v` and `-s?` and bulk will tell you all known option levels, together with their numbers, which can be used for the kernel interface.

Or, if you know the option level (e.g., SOL_TCP, number 6), but you do not know possible option names, you can use `-v` and `-sSOL_TCP,?` or `-s6,?` and bulk will suggest possible option names.

This context help works with `-g` switch as well. Furthermore, you can tahe advantage of the fact that one definite unambiguous member determines the rest of the options to retrieve all other possible members. For instance, you can type `-v` and `-gsnd_cwnd,?` to help with all the rest of TCP_INFO option with their offsets.

4 Predefined mnemonics

The purpose of mnemonics is to ease the access to the setsockopt() / getsockopt() interface, which is strictly number-based because of its low-level nature. All mnemonics can be divided into three classes - option levels, option names (always existing within a certain option level) and members (always existing within a certain option name and option level). Unique member names are used to imply their option names and option leves and unique option names are used to imply their option levels. These dependencies are evaluated by bulk so that the user can specify only the necessary minimum syntax.

Mnemonics for option levels and option names is case insensitive and does not contain characters '.', '.' and 'within the option names and within the option levels (but the same name can be used for an option name and for an option level). The strings are translated to corresponding numbers as defined in the `bulknames.h` file, which is derived from standard kernel symbols.

A member specified by its symbolic name is translated to the byte offset (relative to the beginning of the structure) and its size. However, this size is taken in account only when no size specifier was used.

4.1 Option levels

The following option levels are recognized:

- "SOL_IP"
- "SOL_TCP"
- "SOL_UDP"
- "SOL_IPV6"
- "SOL_ICMPV6"
- "SOL_SCTP"
- "SOL_RAW"

- "SOL_IPX"
- "SOL_AX25"
- "SOL_ATALK"
- "SOL_NETROM"
- "SOL_ROSE"
- "SOL_DECNET"
- "SOL_X25"
- "SOL_PACKET"
- "SOL_ATM"
- "SOL_AAL"
- "SOL_IRDA"
- "SOL_NETBEUI"
- "SOL_LLC"
- "SOL_SOCKET"

The option level names are usually translated to corresponfing protocol numbers, but number 1 is used for SOL_SOCKET (as a Linux convention) instead of ICMP protocol.

4.2 Option names

The following option names are recognized as belonging to SOL_SOCK option level:

- "SO_DEBUG"
- "SO_REUSEADDR"
- "SO_TYPE"
- "SO_ERROR"
- "SO_DONTROUTE"
- "SO_BROADCAST"
- "SO_SNDBUF"
- "SO_RCVBUF"
- "SO_KEEPALIVE"
- "SO_OOBINLINE"

- "SO_NO_CHECK"
- "SO_PRIORITY"
- "SO_LINGER"
- "SO_BSDCOMPAT"
- "SO_PASSCRED"
- "SO_PEERCRED"
- "SO_RCVLOWAT"
- "SO_SNDLOWAT"
- "SO_RCVTIMEO"
- "SO_SNDTIMEO"
- "SO_SECURITY_AUTHENTICATION"
- "SO_SECURITY_ENCRYPTION_TRANSPORT"
- "SO_SECURITY_ENCRYPTION_NETWORK"
- "SO_BINDTODEVICE"
- "SO_ATTACH_FILTER"
- "SO_DETACH_FILTER"
- "SO_PEERNAME"
- "SO_TIMESTAMP"
- "SO_ACCEPTCONN"
- "SO_DONTLINGER"

These names can be used for generic manipulation with the socket. With the exception of SO_PEERNAME and SO_DONTLINGER they all consist of one signed longint number. Their implementation may modify this number, so when in doubt, always look in `linux/net/core/sock.c` for their detailed implementation, which can even vary between different kernel releases.

The following option names are recognized as belonging to SOL_TCP option:

- "TCP_NODELAY"
- "TCP_MAXSEG"
- "TCP_CORK"
- "TCP_KEEPIDLE"

- "TCP_KEEPINTVL"
- "TCP_KEEPCNT"
- "TCP_SYNCNT"
- "TCP_LINGER2"
- "TCP_DEFER_ACCEPT"
- "TCP_WINDOW_CLAMP"
- "TCP_INFO"
- "TCP_QUICKACK"

These names belongs to all known TCP-related options at the SOL_TCP level. Implementation details of the current meaning of each option should be checked in `linux/net/ipv4/tcp.c`.

The following option names are recognized as belonging to TCP_INFO option:

- "state"
- "ca_state"
- "retransmits"
- "probes"
- "backoff"
- "options"
- "wscales"
- "rto"
- "ato"
- "snd_mss"
- "rcv_mss"
- "unacked"
- "sacked"
- "lost"
- "retrans"
- "fackets"
- "last_data_sent"
- "last_ack_sent"

- "last_data_recv"
- "last_ack_recv"
- "pmtu"
- "rcv_ssthresh"
- "rtt"
- "rttvar"
- "snd_ssthresh"
- "snd_cwnd"
- "advmss"
- "reordering"

These mnemonics determine each particular part of the big TCP_INFO option (it is a recondite read-only option, but default), which is a part of the SOL_TCP level. Each name will be translated to the desired offset to the variable itself and the size of the variable will be determined (for its proper handling). The size can be overriden by the size specifier, if necessary. When no member is specified, bulk always takes a longint from the beginning of the structure. Also non-structured options can be unscrabled to smaller parts, but we must specify their unknown offsets numerically. You can simply type any of these names and bulk will know what option level, option name, offset and size should be used. For detailed meaning of these options look at `linux/net/ipv4/tcp.c` in the corresponding part of `tcp_getsockopt()` function.

4.3 Supporting new options

When a new option is added to the kernel, we can access it directly by its numbers (in which case no changes to bulk are required) or we can add new mnemonics to `bulknames.h` and let the bulk know their mnemonics.

Currently the following non-standard options are recognized by bulk (implemented by our AIMD kernel patch):

- "TCP_AIMD"
- "TCP_COUNTERS"

These options are implemented by the AIMD kernel patch (available for 2.4 and also 2.6 kernels) and contains a structure similar to the TCP_INFO option (which is however not read-only). These options extend the SOL_TCP level and their internal structure is also known to the bulk under the following names:

- For TCP_AIMD: "slope", "ratio", "cwven" and "tqcwr"
- For TCP_COUNTERS: "cwr_ss", "cwr_ot", "cwv_dec", "cwv_adj" and "cwnd_mod"

Detailed meaning of these variables can be found in the `AIMD_README.txt` file, supplied with the AIMD kernel patch.

5 Examples of use

For instance, we may start a test with 10 parallel flows with sender socket buffer size and block size of 10 MB and we may be interested whether the initial handshaking works properly (looking at agreed window scale factor), what is rcv_ssthresh, current rtt and receiver socket buffer size. We can use the following command to start bulk as throughput test server with the appropriate parameters:

```
./bulk -v -m -c50 -sSO_RCVBUF,10000000 -b10000000\
  -gwscales%Scales:\ ,rcv_ssthresh%Recv\ thresh:\ ,rtt%RTT:\
  -gSO_RCVBUF%Real\ RCVBUF\ size:\ >outfile_r.txt
```

The output will the look like the following (for 5th flow in progress):

```
...
[5] Scales: 16Recv thresh: 81919RTT: 10000Real RCVBUF size: 131070
[5] Scales: 16Recv thresh: 81919RTT: 10000Real RCVBUF size: 131070
...
```

In another scenario, we may be interested in information about cwnd, snd_ssthresh and CWR and CVW adjustements. We will also adjust AIMD parameters for the particular connection (requires our AIMD patch). The resulting command will look as follows:

```
./bulk -v -b10000000 -sTCP_AIMD,100,75,1,1 -gcwnd%CWND:\
  ,snd_ssthresh%Thresh:\ and -gcwr_ss%SS_Cwrs:\ ,cwr_ot%Other_CWRs:\
  ,cwv_dec%CWV\ actions: ,cwv_adj%CWV\ adjusts:\ ,cwnd_mod%Moderations:\
  remote.machines.org >outfile_s.txt
```

6 Conclusion

We described a new TCP performance debugging utility called "bulk", which permits synchronous monitoring of TCP throughput and various runtime TCP variables using socket options including support for all future socket options.

We plan to implement bulk functionality as iperf extension and to provide possibility to read current TCP sequence numbers, which are not available from the TCP_INFO option and which can be useful for correlation with an external packet dump (such as from tcpdump).