NETIO documentation v1.0.0

Doc Version

Version	Date	Author	Description
1.0.0	15.09.2020	mpetavy	Initial release

Description

NETIO is a performance testing tool for network or tty connections.

The following feature set is supported:

- With or without TLS usage
- Zero bytes, random bytes or file transfer
- Looping rounds with timeout or file transfer

TLS data encryption

NETIO supports TLS data encryption by version TLS 1.0 - TLS 1.3. By setting "-tls" a self signed certificate is automaticaly generated for the server side. Client connects with "-tls" also via TLS. Server verification is disable in default. Can be activated "-tls.verify"

Support for TLS 1.0 and TLS 1.1 is disabled in default ut can be enabled via "-tls.insecure".

NETIO TLS implementation is done by the GO default "BoringSSL" implementation: https://boringssl.googlesource.com/boringssl/

GO development

NETIO is developed with the Google GO tooling.

Current used version 1.15.2

By using the GO programming language (https://golang.org) multiple architectures and OS are supported. You can find a complete list of all supported architectures and OS at https://github.com/golang/go/blob/master/src/go/build/syslist.go

Currently these environments are tested in development

- Linux (x86_64)
  - Manjaro on x86_64 based PC https://www.manjaro.org
  - Raspian on ARM7 based Raspberry Pi Model 3 Model B+ https://www.raspberrypi.org/downloads/raspbian
- Windows 10 (x86_64)

Compiling NETIO

Before NETIO can be compiled please make sure the following tasks in this order.

1. i18n and opensource license files. To generate those files please execute inside the NETIO source code folder the following command "go run . -codegen".
   i. This generates an updated "static/netio.i18n" file with all i18n strings inside the NETIO source code files.
   ii. This generates an updated "static/netio-opensource.json" file with an listing if all used opensource modules along with their license infos.
2. BINPACK resources. All resources of NETIO must be transpiled to "binpack" source code files in order to be compiled statically to the NETIO executable. For that please use the BINPACK executable (https://github.com/mpetavy/binpack). Execute the transpile with the command "binoack -i static" inside the NETIO source code folder. After successfull execution an updated GO soource code file "binpack.go" is generated.
3. "vendor.tar.gz" file. When NETIO is compiled with Docker the compilation process uses GO's feature of "vendor"ing, That means the GO compiler in the Docker build does not use the standard GOPATH directory for 3d party modules source code files but uses the "vendor" directory in the NETIO source code folder. The "vendor" is generated automatically in the Docker build by untaring the TAR file "vendor.tag.gz". To update the "vendor.tar.gz" file to match the latest GO modules in the GOPATH of the development environment the batch job "update-vendor.bat" can be used.

Build with Docker

NETIO can be built either by the BUILD.SH (Linux) or BUILD.BAT (Windows) batch jobs. The Build uses Docker to generate an Image in which the complete packages for Windows and Linux are generated.

This is done by using GO's built-in feature to do cross-compiling to any supported plattform inside the Docker images.

After the docker image creatiion a temporaty docker container is built from which the following 3 software packages are extracted:

Sample for Version "1.0.0" and Build number "1234":

- netio-1.0.0-1234-linux-amd64.tar.gz
- netio-1.0.0-1234-linux-arm6.tar.gz

- netio-1.0.0-1234-windows-amd64.tar.gz

Those software packages containes everything for running NETIO on the defined platform.

Build manual

To build a binary executable for your preferred OS please do the following:

1. Install the GO programming language support (http://golang.org)
2. configure your OS env environment with the mandatory GO variables
   i. GOROOT (points to your)
   ii. GOPATH (points to your)
   iii. OS PATH (points to your /bin)
3. Open a terminal
4. CD into your GOPATH root directory
5. Create a "src" subdirectory
6. CD into the "src" subdirectory
7. Clone the netio repository
8. CD into the "netio" directory
9. Build manually:
   i. If you would like to cross compile to an other OS/architecture define the env variable GOOS and GOARCH along to the values defined here https://github.com/golang/go/blob/master/src/go/build/syslist.go
   ii. Build NETIO by "go install". Multiple dependent modules will be downloaded during the build
   iii. After a successful build you will find the NETIO executable in the "GOPATH\bin" directory
10. Build automatically with GoReleaser https://goreleaser.com/:
    i. Use just or modify the build configuration file.goreleaser.yml
    ii. Execute: `goreleaser --snapshot --skip-publish --rm-dist`
    iii. After a successful build you will find the NETIO executable in the ".dist" directory.

Installation as application

Like all other GO based application there is only the file `netio.exe` or `netio` which contains the complete application.

Just copy this executable into any installation directory you would like. Start the application by calling the executable `netio.exe` or `./netio`

Installation as OS service

Follow the instructions "Installation as application". To register NETIO as an OS service do the following steps.

1. Open a terminal
2. Switch to root/administrative rights
3. CD into your installation directory
4. Installation NETIO as an OS service:
   i. Windows: `netio -service install`
   ii. Linux: `./netio -service install`
5. Uninstallation NETIO as an OS service:
   i. Windows: `netio -service uninstall`
   ii. Linux: `./netio -service uninstall`

Running NETIO with Docker

The Linux amd64 package containes everything for running NETIO with Docker. Just use the provided "docker-compose-up.bat" and "docker-compose-down.bat". Here a sample Dockerfile:

```
FROM alpine:latest
RUN mkdir /app
WORKDIR /app
COPY ./NETIO .
EXPOSE 8443 15000-15050
EXPOSE 15000/udp
RUN apk --no-cache update \
    && apk --no-cache upgrade \
    && apk --no-cache add ca-certificates \
    && apk --no-cache add dbus \
    && apk --no-cache add tzdata \
    && cp /usr/share/zoneinfo/Europe/Berlin /etc/localtime \
    && echo "Europe/Berlin" >  /etc/timezone \
    && dbus-uuidgen > /var/lib/dbus/machine-id
ENTRYPOINT /app/NETIO
```

Running NETIO with Linux Container (LXC)

The Linux amd64 package containes everything for running NETIO with Linux container (LXC). Here a sample script to setup and install NETIO inside a Linux container based on Debian. Finally the LXC is exported to a tar.gz file.

- Used LXC version is 4.0.0 (compatible 2.x+)
- LXC container name is 'NETIO'
- NETIO is installed as service 'NETIO.service'
- NETIO service is enabled and started
- Assuming that the executable file "NETIO" ist in the current path.

Content of the file 'lxc.sh':

```
#/bin/sh
# lxc delete debian-netio --force
lxc launch images:debian/10 debian-netio
lxc exec debian-netio -- mkdir /opt/netio
lxc file push netio debian-netio/opt/netio/
lxc exec debian-netio -- /opt/netio/netio -log.verbose -service install
lxc exec debian-netio -- systemctl enable netio.service
lxc exec debian-netio -- systemctl start netio.service
lxc export debian-netio lxc-debian-netio.tar.gz --instance-only
```

The ending line 'lxc list debian-netio' prints out the IP address on which you can connect to the NETIO interface.

Serial interface parameter format

The format for defining a serial device is as follows:

```
<device>,<baud>,<databits,<parity>,<stopbits>
```

Parameter	Default value	Description
device		Defines the OS specific serial interface name like "COM3" (Windows) or "/dev/ttyUSB0" (Linux)
baud	9600	Defines the baud rate ("50", "75", "110", "134", "150", "200", "300", "600", "1200", "1800", "2400", "4800", "7200", "9600", "14400", "19200", "28800", "38400", "57600", "76800", "115200")
databits	8	Defines the amount of databits (1-8)
parity mode	N	Defines the partity mode ("N" no parity, "O" odd parity, "E", even parity)
stopbits	1	Defines the amount of stopbits ("1", "1.5", "2")

Shortage of the parameter definition is support, so for not defined parameter the default value is used.

```
# setting 115200,8,N,1
"/dev/ttyUSB1,115200"

# setting 28800,5,E
"/dev/ttyUSB1,28800,5,E"
```

Runtime parameter

Here you find the complete list of NETIO runtime parameters. Please substitute default filename parameters with the prefix "/home/ransom/go/src/NETIO" with your installaton directory of NETIO.

Parameter	Default value	Description
?	false	show usage
backup.count	3	amount of file backups
bs	32K	block size in bytes
c		client socket address or TTY port
cfg.file	/home/ransom/go/src/netio/netio.json	Configuration file
cfg.reset	false	Reset configuration file
cfg.timeout	0	rescan timeout for configuration change
f		filename to write (client)/read (server)
h		hash algorithm
language	en	language for messages

Parameter	Default value	Description
lc	10	loop count
log.file		filename to log logFile (use "." for /home/ransom/go/src/netio/netio.log)
log.filesize	5242880	max log file size
log.io	false	trace logging
log.json	false	JSON output
log.verbose	false	verbose logging
ls	0	loop sleep timeout between loop steps
lt	1000	loop timeout
nb	false	no copyright banner
r	false	write random bytes
rt	0	read throttled bytes/sec
s		server socket address
st	1000	serial read timeout for disconnect
tls	false	use TLS
tls.info	false	show TLS info
tls.insecure	false	Use insecure TLS versions and ciphersuites
tls.p12		TLS PKCS12 certificates & privkey container stream (P12,Base64 format)
tls.p12file		TLS PKCS12 certificates & privkey container file (P12 format)
tls.verify	false	TLS server verification/client verification
wt	0	write throttled bytes/sec

Samples

Here some sample usages.

```
# start NETIO as server on Port 15000 with TLS
netio -s :15000 -tls

# start client on COM3 with 9600 Baud, 8 Databist, no parity, 1 Stopbit and transfer the file "testfile.txt"
netio -c COM3,9600,8,N,1 -f testfile.txt
```