Taking Tcl Networking to the Next Level

Reinhard Max <max@tclers.tk>

About myself

- First contact with Tcl around 1996
- 25 Years @ SUSE
- Tcl/Tk maintainer since 1998
- SQLite, PostgreSQL, ntp, chrony, OpenVPN, ClamAV etc.
- Worked on the [socket] command before

Questions (Ouster-Vote)

- Who has used the [socket] command before
- Who has done network communication in C at the syscall level (socket(), bind(), connect(), accept(), …)?
- Who wants UDP in the core?
- Who would like to see support for other types of sockets (unix domain, raw, …)?

Some History …
(from the changes file)

- 1995/96 – Jacob Levy implements the [socket] command to bring TCP communication to Tcl 7.6.
- Improvements by Scott Stanton, Mo DeJong and others.
- Mac improvements by Daniel Steffen and Jim Ingham
- 2009/10 – Reinhard Max rewrites parts of [socket] for 8.6 to support IPv6 and multiple IP addresses per Domain.
- Harald Oehlmann and others help to get -async and Windows notifications right.

What we have now (TCP)

- Client:
```
socket ?-async? host port
```
- Server:
```
socket -server callback port

proc callback {sock host port} {
	# put client handling here
}
```

Quote *(unknown author)*

“ Compared to other languages Tcl’s [socket] command makes network communication so easy that you always think you missed something. ”

Quote *(many Tclers)*

“ But there is still no UDP in the core in 2022! ”

Having a closer look

- [socket] pros
  - Very simple to use, more convenient than the socket() API in C
  - Cross platform abstraction
  - Well established
  - Has been used to implement all sorts of TCP based protocols from HTTP (in the core http package) to XMPP (in TkChat).

Having a closer look

- [socket] cons
  - TCP only
  - Implemented using stream oriented Tcl channels
    => not a good fit for datagram oriented communication.
  - Limited access to setsockopt() etc.
  - No separation between name resolution and socket handling
  - Name resolution is always synchronous

What we have now (UDP)

- Several UDP extensions, but not in the core:
  - TclUDP, Tcl-dp, Scotty, ceptcl, …
- Had no closer look at them, as I wanted a generic solution that goes beyond UDP.
- Some of them don't support IPv6 or support it only as an afterthought.

My idea to get there

- Create a thin layer of C code to make the the whole socket() API available in Tcl without any artificial limitations regarding protocols, etc.
- Not intended to be used directly by application code, but to...
  - reimplement the [socket] command in Tcl (at least as a PoC)
  - be a building block for similar convenience functions (in pure Tcl) for other protocols and use cases.

My idea to get there (2)

- Sockets are not channels, but can be wrapped as such (e.g. refchan).
- Instead, the send and receive functions work with raw byte strings.
- Name resolution is separate from actual socket handling
  - allows for asynchronous resolution by running getaddrinfo in a thread
  - allows using alternative resolvers, e.g. event-driven

Wrapping the socket API

```
% getaddrinfo host port
```

Returns the relevant members of struct addrinfo as a Tcl list:
address family, socktype, protocol, IP(v6) address, and port number.

```
% showaddrinfo addrinfo
```

Debugging function to turn the numbers back into symbolic names.

```
% socket addrinfo
```

Uses the first three members of an addrinfo list to create a socket

```
% connect socket addrinfo
```

```
bind, listen, accept, sendto, recvfrom, shutdown,
(get|set)sockopt, get(sock|peer)name, setblocking
```

Symbolic Constants
(preliminary)

- e.g. `SOCK_STREAM, AF_INET, IPPROTO_TCP`
- Get parsed from preprocessed system headers and generated into alias commands:
```
interp alias {} SOCK dict get {STREAM 1 DGRAM 2 …}
interp alias {} rSOCK dict get {1 STREAM 2 DGRAM …}
```
-
```
% SOCK DGRAM
2
% rSOCK 3
RAW
```
- Q&D solution that might change.

Examples

```
% getaddrinfo -type [SOCK STREAM] example.com 80
{10 1 6 2606:…:1946 80} {2 1 6 93.184.216.34 80}

% showaddrinfo {2 1 6 93.184.216.34 80}
INET STREAM TCP 93.184.216.34 80

% socket {2 1 6 93.184.216.34 80}
6

% connect 6 {2 1 6 93.184.216.34 80}

% getsockname 6
192.168.178.67 192.168.178.67 52020

% close 6
```

Reimplementing [socket] in Tcl

- Q&D code to build a [socket] drop-in replacement in Tcl
- Implements everything except for asynchronous connections
- Passes the core’s socket.test
- Revealed some missing pieces in refchan

Shortcomings of refchan

- Does not allow channels that are neither readable nor writable.
- Will be fixed in core shortly.

- Does not allow half-close of bidirectional channels (shutdown()).
- Will work with Andreas Kupries to get this in.

Finally: UDP and local sockets!

- => Live demo

Lessons learned & ToDo

- `getaddrinfo` only supports IP(v6), but no unix domain and other protocol families.
- `addrinfo` arguments are too inflexible in some cases, alternatives should be allowed.
- Use better tokens for sockets than the underlaying file descriptor number.
- UDP and UNIX domain sockets have only received minimal testing
- Out-of-band data (e.g. for TCP) has not been tested yet, but might just work.
- Linux supports way more socket options than OSX.

Asking for Help!

- Windows support, esp. notifications. (Harald?)
- Improving/fixing refchan. (Andreas?)
- Testing, esp. on non-Linux platforms
- Better ideas for the representation and conversion of symbolic constants
- Better ideas for the representation of socket tokens.
- Better ideas for the representation of addresses in places where getaddrinfo results are not always a good match.
- Performance comparison between the C-coded and Tcl-coded [socket] implementations and improvements on the Tcl code, if needed.
- More convenience code on top of the new API!

Questions & Answers
Comments & Suggestions

https://chiselapp.com/user/rmax/repository/sock

Reinhard Max <max@tclers.tk>