COTCP

KNIHOVNA PROTOKOLU TCP

Příručka uživatele a programátora

***SofCon*®spol. s r.o.**
Střešovická 49
162 00 Praha 6
tel/fax: +420 220 180 454
E-mail: sofcon@sofcon.cz
www: http://www.sofcon.cz

Verze dokumentu 1.11

Informace v tomto dokumentu byly pečlivě zkontrolovány a SofCon věří, že jsou spolehlivé, přesto SofCon nenese odpovědnost za případné nepřesnosti nebo nesprávnosti zde uvedených informací.
SofCon negarantuje bezchybnost tohoto dokumentu ani programového vybavení, které je v tomto dokumentu popsané. Uživatel přebírá informace z tohoto dokumentu a odpovídající programové vybavení ve stavu, jak byly vytvořeny a sám je povinen provést validaci bezchybnosti produktu, který s použitím zde popsaného programového vybavení vytvořil.
SofCon si vyhrazuje právo změny obsahu tohoto dokumentu bez předchozího oznámení a nenese žádnou odpovědnost za důsledky, které z toho mohou vyplynout pro uživatele.

Datum vydání: 21.04.2004
Datum posledního uložení dokumentu: 21.04.2004
(Datum vydání a posledního uložení dokumentu musí být stejné)

Obsah :

1. O dokumentu 5
1.1. Revize dokumentu 5
1.2. Účel dokumentu 5
1.3. Rozsah platnosti 5
1.4. Související dokumenty 5
2. Termíny a definice 5
3. Úvod 6
3.1. Účel knihovny CoTCP 6
3.2. TCP protokol 7
3.2.1. Struktura TCP segmentu 7
3.2.2. Volitelné položky záhlaví 9
3.2.3. Fáze přenosu dat 9
3.2.4. Fáze navazování a závěru spojení 11
3.2.5. Technika zpožděné odpovědi 12
3.2.6. Technika zpožděného odesílání dat 13
3.2.7. Opakování ztracených segmentů 13
3.2.8. Vyhýbání se zahlcení 13
3.2.9. Porty 15
4. Konstanty a jednoduché typy 16
4.1. Konstanty 16
4.1.1. Návratové kódy CST_TCP_ERR_xxx 16
4.1.2. Identifikátory tříd zařízení DEV_CLASS_xxx 16
4.1.3. Identifikátory řídících operací IOCTL_TCP_xxx 17
4.1.4. Identifikátory parametrů zařízení COPT_TCP_xxx 17
4.1.5. Upřesňující příznaky volání operací COF_TCP_xxx 18
4.2. Typy 19
4.2.1. Třída TCoTcp 19
4.2.2. Třída TCoTcpMux 19
4.2.3. Struktura TTcpAddress 19
5. Textový konfigurační řetězec 20
6. Poznámky 22
6.1. Inicializace zařízení 22
6.2. Deinicializace zařízení 22
6.3. Navazování spojení 22
6.3.1. Navazování spojení pomocí CoConnect 22
6.3.2. Navazování spojení pomocí metod CoAccept a CoListen 22
6.4. Uzavírání spojení 23
6.5. Posílání a příjem dat 23
7. Příklad 24

1. O dokumentu

1.1. Revize dokumentu

Verze dokumentu	Verze SW	Autor	Datum vydání	Popis změn
1.00	1.XX	Čr		První vydání
1.10	1.XX	Tu	16.05.2003	Úprava dokumentu dle ISO9000
1.11	1.XX	Bin	21.04.2004	Oprava příkladu v kapitole 7 (doplnění uvozovek v parametrizačním řetězci).

1.2. Účel dokumentu

Tento dokument slouží jako popis jednotky implementující protokol TCP.

1.3. Rozsah platnosti

Určen pro programátory a uživatele programového vybavení SofCon.

1.4. Související dokumenty

Pro čtení tohoto dokumentu je potřeba seznámit se s manuálem CoBase.

Popis formátu verze knihovny a souvisejících funkcí je popsán v manuálu LibVer.

2. Termíny a definice

Používané termíny a definice jsou popsány v samostatném dokumentu „Termíny a definice“.

3. Úvod

3.1. Účel knihovny CoTCP

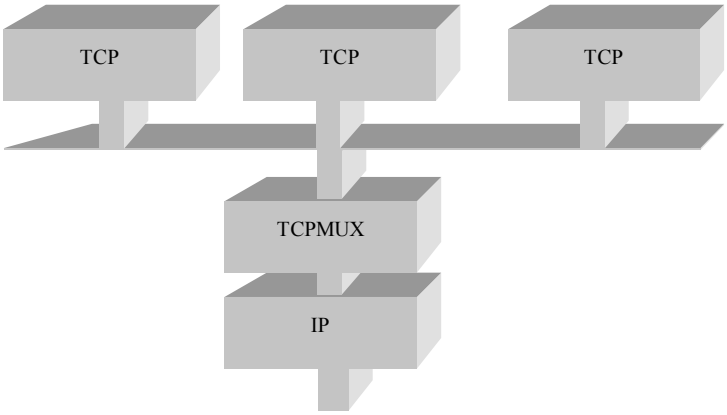
Knihovna **CoTCP** implementuje protokol TCP (Transmittion Control Protocol). Tj. jeden z transportních protokolů architektury TCP/IP zajišťující koncový přenos dat mezi dvěma stanicemi. Tento protokol je implementován pomocí zařízení TCoTCP. Z následujícího obrázku je zřejmé umístění transportní vrstvy v rámci TCP/IP architektury. Stručný popis architektury TCP/IP je uveden v dokumentaci ke knihovně CoBase.

TCP/IP

TCP protokol vytváří spojovou službu, která mezi dvěma aplikace před samotným přenosem dat naváže spojení a vytvoří tzv. virtuální okruh. Tento okruh je plně duplexní. Přenášené oktety jsou číslovány a v případě ztráty nebo poškození jsou znovu vyžádány. Integrita dat je zabezpečena kontrolním součtem. TCP protokol používá k řízení toku dat virtuálním okruhem heuristické algoritmy, které brání zahlcení jedné či druhé stanice, příp. jiného prvku na cestě v síti

TCP protokol je implementován pomocí dvou zařízení:

- TCP zásuvka (zařízení TCP, TCoTcp)
- TCP multiplexer (zařízení TCPMUX, TCoTcpMux)



Nad síťovou vrstvou (IP protokol) je vždy jedno zařízení TCP multiplexer (zařízení TCPMUX). K zařízení TCPMUX je možné připojit libovolný počet TCP zásuvek (zařízení TCP), které jsou charakterizovány číslem portu. Aplikace vždy přistupuje pouze k zařízení TCP, nikdy ne k TCPMUX.

Protokol TCP je implementovaný pomocí třídy TCoDevice. V této dokumentaci jsou popsány pouze odlišnosti a speciality protokolu TCP (tedy zařízení TCP a TCPMUX). Ostatní naleznete v dokumentaci ke knihovně CoBase.

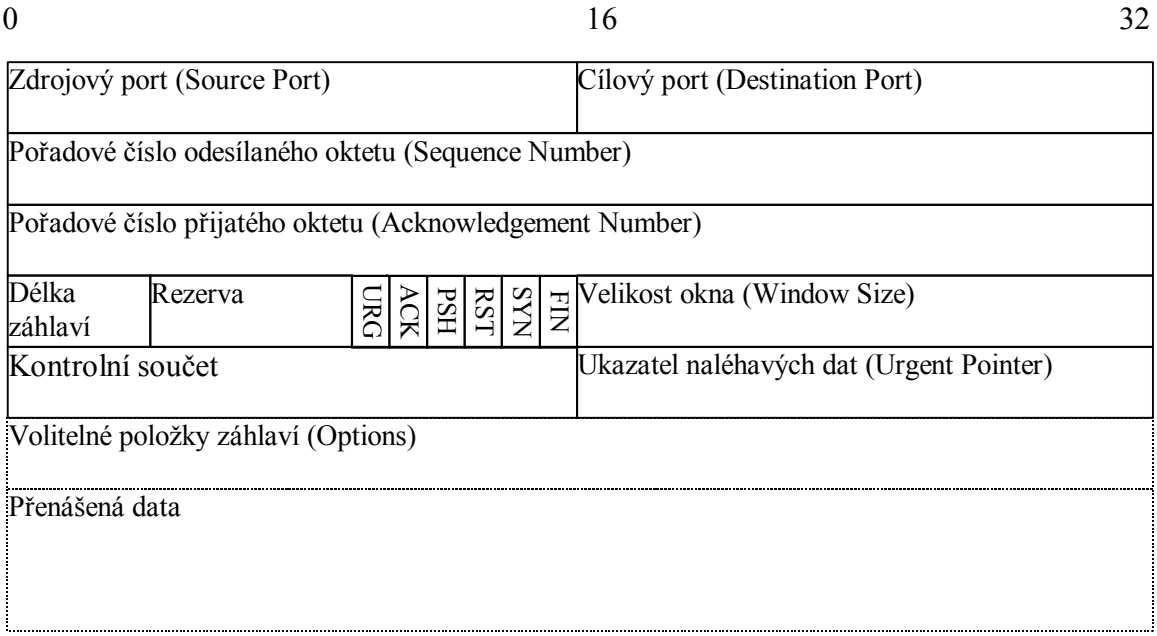
3.2. TCP protokol

TCP protokol vytváří spojovou službu, která mezi dvěma aplikace před samotným přenosem dat naváže spojení a vytvoří tzv. virtuální okruh. Tento okruh je plně duplexní. Přenášené oktety jsou číslovány a v případě ztráty nebo poškození jsou znovu vyžádány. Integrita dat je zabezpečena kontrolním součtem. TCP protokol používá k řízení toku dat virtuálním okruhem heuristické algoritmy, které brání zahlcení jedné či druhé stanice, příp. jiného prvku na cestě v síti. Tyto algoritmy budou nastíněny v kapitolách 3.2.7 a 3.2.8.

Posílaná data jsou vkládána do tzv. **segmentů** (viz. následující kapitola). Protože se nejedná o datagramovou službu (jako v případě protokolu UDP), mezi segmenty a požadavky vysílání aplikační vrstvy neexistuje přímá souvislost. Požadavek na vysílání může být rozdělen do více segmentů, nebo příp. více požadavků může být sceleno do jediného segmentu. Stejně tak příjemce není schopen rozeznat v přijatých datech hranice mezi bloky dat vyslanými aplikační vrstvou odesílatele.

3.2.1. Struktura TCP segmentu

Pakety posílané pomocí protokolu TCP se nazývají segmenty. Struktura segmentu je zobrazena na následujícím obrázku. Délka segmentu závisí na délce pole **Volitelné položky záhlaví** a může se pohybovat v rozsahu od 20 do 60 oktetů (bajtů).



Zdrojový port Jednoznačně identifikuje zdrojový aplikační proces odesílatele.

Cílový port Jednoznačně identifikuje cílový aplikační proces příjemce.

Potvrzení, Pořadové číslo odesílaného oktetu Číslo prvního oktetu dat v segmentu.

Pořadové číslo přijatého oktetu Specifikuje číslo oktetu, které se očekává v dalším přijatém segmentu. Položka je platná, pokud je v řídícím poli nastaven příznak ACK.

Délka záhlaví Délka záhlaví (včetně jeho volitelné části) v násobcích 32 bitů.

Řídící pole Šest příznaků určených pro navazování, závěr a udržování spojení.

URG - označuje naléhavá data pro přednostní doručení.
SYN - je žádostí o navázání spojení.

ACK - označuje platnost pole s Pořadovým číslem očekávaného oktetu. Všechny segmenty kromě prvního mají nastaven tento příznak.
RST - Odmítnutí TCP spojení.
PSH - požaduje okamžité doručení dat segmentu vyšší vrstvě.
FIN - je žádost o ukončení spojení

Velikost okna Počet oktetů, které může druhá stanice vyslat, bez průběžného potvrzování.

Kontrolní součet Zabezpečuje záhlaví a data celého segmentu

Ukazatel naléhavých dat Určuje poslední oktet naléhavých dat.

Volitelné položky záhlaví Doplňkové informace pro řízení toku dat, jako je např. maximální délka segmentu apod. Toto pole je podrobněji popsáno v kapitole 3.2.2.

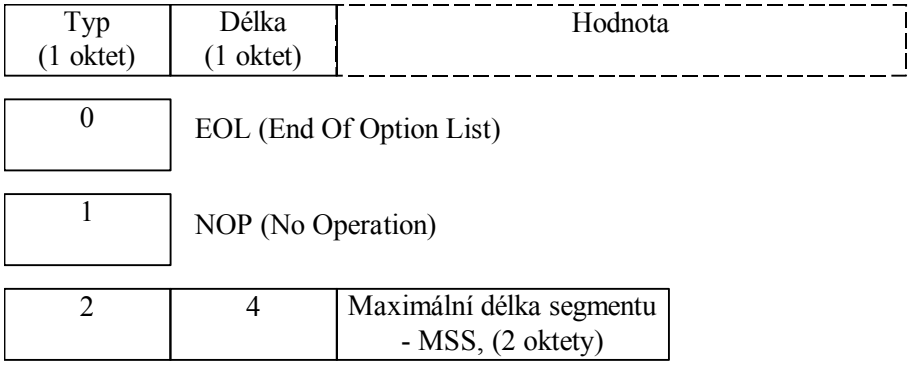
Přenášená data Velikost odesílaných dat by neměla přesáhnout 536 oktetů, pokud se stanice nedomluví jinak pomocí položky MSS volitelné části záhlaví (viz. kapitola 3.2.2).

3.2.2. Volitelné položky záhlaví

Volitelné položky se nacházejí mezi pevnými položkami záhlaví a daty datagramu. Snažíme se volitelným položkám spíše vyhýbat, protože prodlužují dobu zpracování segmentu. Existují dva typy volitelných položek:

- Speciální jednooktetové položky EOL (End of Option List) , NOP (No Operation), které se vkládají na konec seznamu volitelných položek, příp. jako výplň mezi jiné další položky.
- Víceoktetové položky složené z typu, délky a vlastní hodnoty položky.

Formát některých z volitelných položek záhlaví je uveden na následujícím obrázku:



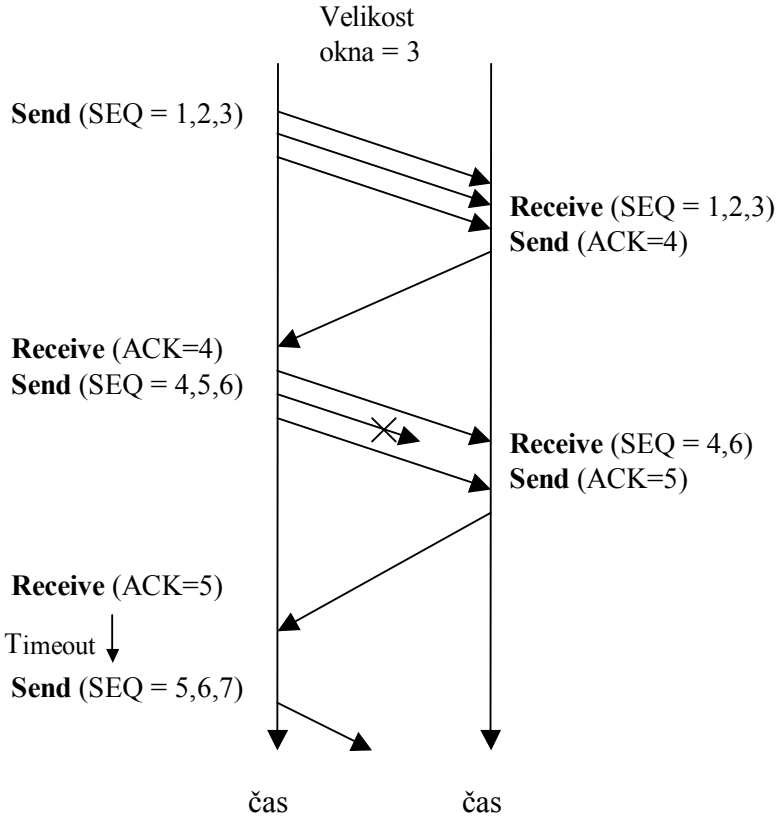
Pomocí položky **MSS** (Maximum Segment Size) se transportní vrstvy stanic domlouvají na maximální délce segmentu, který je možno odeslat. Položka MSS se obvykle uvádí pouze v prvním segmentu při navazování spojení. Pokud položka MSS není v záhlaví uvedena, předpokládá se standardní maximální velikost segmentu - 536 oktetů.

3.2.3. Fáze přenosu dat

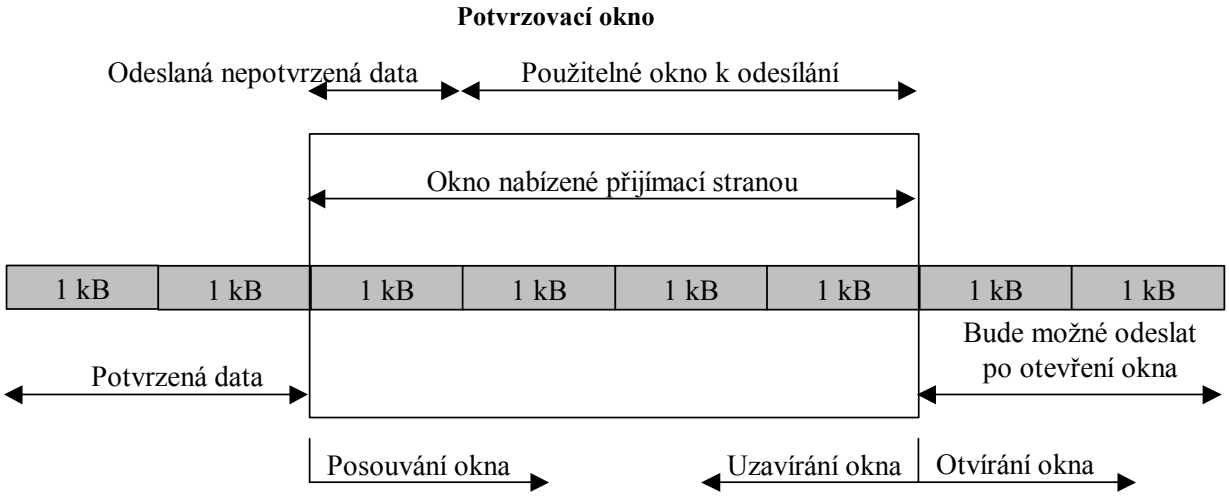
Nejprve je nutno poznamenat, že všechna odesílaná data jsou číslována. Každý odeslaný oktet má své jednoznačné číslo, a každý tak může být potvrzen. Potvrzovací mechanismus je kumulativní, tzn. že potvrzení N-tého oktetu potvrzuje všechny předchozí oktety (s nižším číslem). Číslo prvního oktetu v datech segmentu je uvedeno v záhlaví segmentu. Číslo potvrzovaného oktetu se také přenáší v záhlaví segmentu (který může nést data) a tak žádný speciální segment nesoucí pouze potvrzení tedy není potřeba (tzv. piggy-backing).
Číslo prvního odesílaného oktetu je zvoleno v průběhu navazování spojení. Tato hodnota musí být náhodná a doporučuje se, aby byla svázaná s hodinami stanice.
S každým odesílaným segmentem inzeruje odesílatel velikost svého přijímacího okna,

tj. počet oktetů, který je schopen v daném okamžiku akceptovat. Druhá strana, tak může teoreticky najednou odeslat tolik dat, bez toho aby očekávala potvrzení, kolik inzeruje příjemce velikostí přijímacího okna. Všechna data mimo toto okno jsou příjemcem zahazována. Okno napomáhá k zprůchodnění dlouhých cest v síti s velkým zpožděním. Potvrzení nejsou vysílána vždy po zaplnění okna, ale průběžně (obvykle nejhůře po přijetí dat o velikosti dvojnásobku MSS (Maximální velikost segmentu).

Na následujícím obrázku je uveden jednoduchý příklad přenosu dat mezi dvěma stanicemi. Pro zjednodušení se zde předpokládá, že data putují pouze jedním směrem a v každém segmentu je právě jeden oktet. Na obrázku je také znázorněna situace, kdy se jeden segment ztratí. V takovém případě příjemce potvrdí pouze souvislou část přijatých dat a odesílatel po uplynutí jisté doby, označované jako RTO (Retransmittion Timeout), vyšle data znovu. Dále je možné si všimnout, že číslo potvrzení je spíše číslem následujícího očekávaného oktetu. Je tedy o jedničku vyšší, než je pořadové číslo poslední přijatého oktetu.

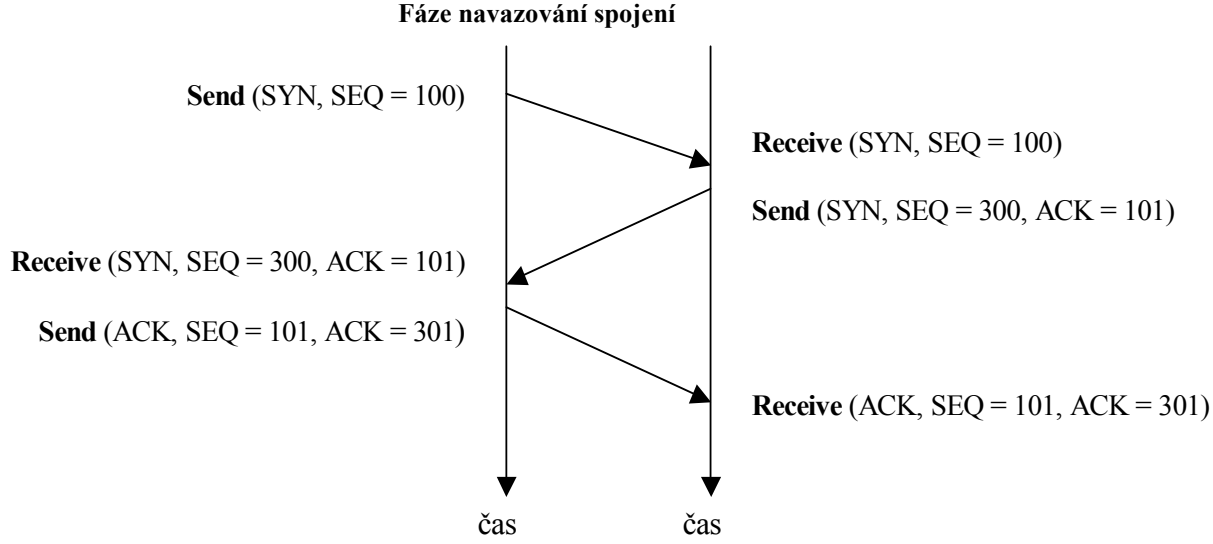


Princip funkce okna protokolu TCP ukazuje obrázek níže. Okno začíná oktetem následujícím za posledním potvrzeným oktetem. Společně s přijatými potvrzeními se začátek okna posouvá doprava. Pokud je dostatek paměti pro příjem dalších dat, okno se otvírá směrem doprava, v opačném případě se okno uzavírá směrem doleva. Algoritmy pro posouvání a otvírání okna nejsou triviální. Pomalé otvírání okna může vést k rozmělnění vysílaných dat do malých segmentů. Při prudkém uzavření okna dochází k zahazování segmentů. Přesný algoritmus pro nakládaní s oknem - SWS (Silly Window Syndrom) Avoidance Algorithm je popsaný v RFC-1122.



3.2.4. Fáze navazování a závěru spojení

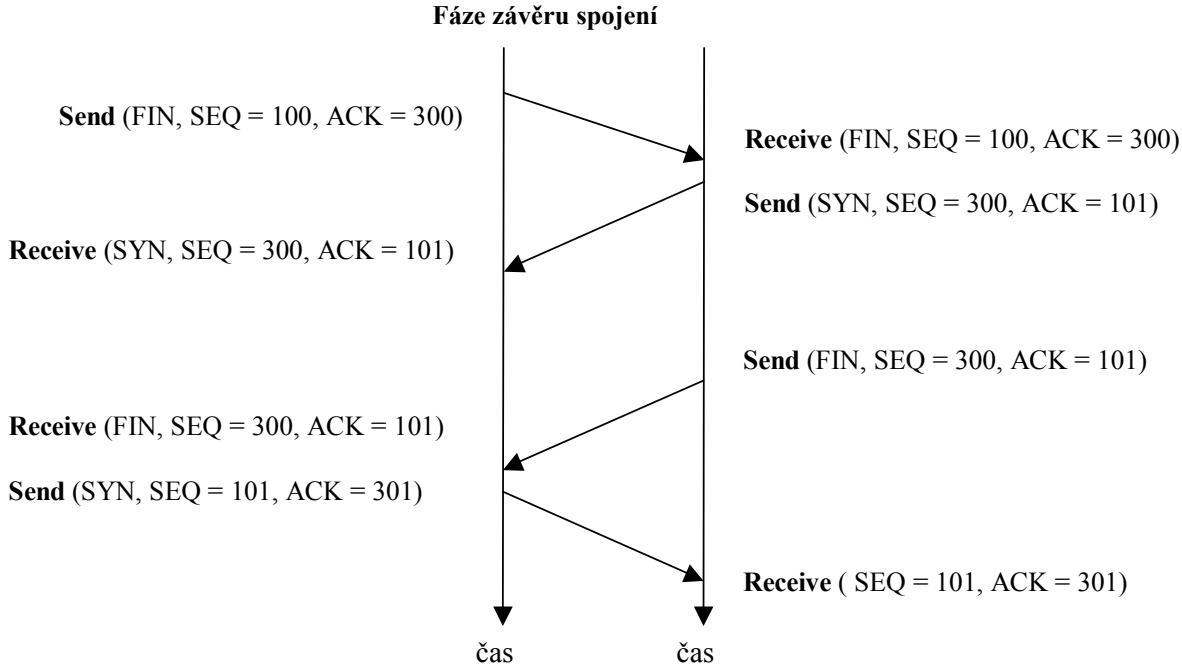
Navazování spojení probíhá ve třech fázích (tzv. Three Way Handshaning). Stanice navazující spojení vyšle první segment s nastaveným příznakem **SYN** (příznak řídícího pole). Pole záhlaví **Pořadové číslo vysílaného oktetu** (SEQ) vyplní náhodně zvoleným číslem, tzv. ISN (Initial Sequence Number), které se použije jako pořadové číslo prvního odesílaného oktetu. Stanice, která přijme tento segment, zhodnotí, zda přijme nabídku k navázání spojení, a v kladném případě odešle segment s nastavenými příznaky **SYN** a **ACK**. **Pole pořadové číslo vysílaného oktetu** (SEQ) vyplní náhodně zvoleným číslem (tzv. ISN) a pole **Pořadové číslo přijímaného oktetu** (ACK) vyplní hodnotou přijatou v poli SEQ zvýšenou o jedničku. Stanice iniciující navázání spojení provede dokončení navázání spojení vysláním potvrzení, tj. segmentu s nastaveným příznakem **ACK**. Pole ACK vyplní přijatou hodnotou v poli SEQ zvýšenou o jedničku. Celý proces ukazuje následující obrázek.



Výše uvedený postup navázání spojení je pouze jeden ze tří možných. Situace se komplikuje, pokud stanice navazují spojení současně. Podrobnější rozbor lze nalézt v

RFC-793.
Pokud stanice z jakéhokoli důvodu odmítne navázání spojení, vyšle segment s nastaveným příznakem **RST**.

Postup závěru spojení je podobný s postupem navazování spojení. Příznak **SYN** v záhlaví TCP segmentu je nahrazen příznakem **FIN**. Pro řádné ukončení spojení jsou potřeba čtyři segmenty. Pokud stanice ukončí spojení, pak již nemůže odesílat data. Druhá stanice však může vysílat do té doby, než sama potvrdí ukončení spojení.



3.2.5. Technika zpožděné odpovědi

Některé interaktivní aplikace, jako například Telnet (klient), reagují na stisk klávesy vysláním jejího kódu. Protější strana (server) musí jednak potvrdit přijatý znak a vyslat znak zpět (echo), aby klientův software zobrazil znak na obrazovce. Přijatý znak musí klient opět potvrdit. Pokud by takto aplikace pracovaly, přenesly by se v každém směru dva segmenty (tj. cca 81 oktetů). Ve snaze snížit objem přenášených dat, se používá techniky zpožděné odpovědi.
Princip techniky zpožděné odpovědi spočívá ve zpoždění odeslání segmentu s potvrzením o několik stovek milisekund (podle RFC-1122 by tento čas by neměl přesáhnout 500ms). Aplikace Server ve výše uvedeném příkladě má tedy dostatek času, aby zareagovala na přijatý znak a vyslala odpověď, jejíž součástí je potvrzení přijatého znaku. Podobně na straně klienta může dojít k úspoře. Pokud uživatel stiskne po přijetí odpovědi do zmíněné doby další klávesu, bude její kód vyslán současně s potvrzením. Takto lze tedy snížit celkový objem přenášených dat na polovinu.

3.2.6. Technika zpožděného odesílání dat

Jak bylo uvedeno v předchozí kapitole, režie spojená s odesíláním krátkých segmentů je obrovská. Dalším postupem k jejímu snížení je technika zpožděného odesílání uživatelských dat. Přesný algoritmus je uveden v RFC-1122 kap. 4.2.3.4.

Data jsou posílána v následujících případech:

- Ve vysílacím bufferu je dostatek dat, aby bylo možné odeslat segment plné velikosti
- Aplikační vrstva explicitně určila, že data mají být okamžitě odeslána
- Všechna předchozí data byla potvrzena a je zaplněna alespoň polovina vysílacího bufferu.
- Vyprší jistý časový limit (obvykle několik stovek milisekund, maximálně však 1s).

Uvedený postup je součástí algoritmu nazývaného SWS Avoidance Algorithm. Pro některé realtimeové aplikace je zpožděné odesílání dat nevhodné, a proto implementace protokolu TCP umožňuje vypnutí této funkce.

3.2.7. Opakování ztracených segmentů

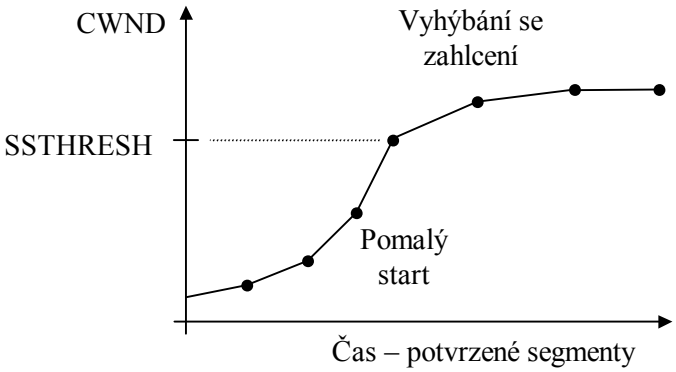
Všechna odeslaná data musí být do doby nazývané RTO (Retransmittion Timeout) příjemcem potvrzeny. Pokud potvrzení nedorazí (což může být způsobeno ztrátou segmentu s daty, nebo také ztrátou segmentu s potvrzením), odesílatel data zopakuje. Vysílání dat se opakuje, dokud nepřijde potvrzením, nebo nevyprší maximální čas pro provedení operace vysílání. Po každém opakování se RTO zvýší na dvojnásobek.
Vzhledem k různorodosti prostředí, může být zpoždění přenosu dat různé: Od milisekund na lokální síti po desítky sekund v Internetu. Nabízejí se dvě řešení nastavení RTO: RTO nastavit staticky na dostatečně velkou hodnotu, a nebo RTO určovat dynamicky pro každé spojení. Podle RFC-1122 musí každá implementace podporovat druhou uvedenou variantu.
TCP provádí odhad zpoždění měřením doby mezi odesláním dat a příjmem potvrzení. Tento čas se nazývá RTT (Round Trip Time). Měření se provádí neustále po dobu spojení a v jednom okamžiku se provádí maximálně jedno měření. Po každém měření se upraví hodnota RTO. Úplný algoritmus výpočtu RTO a měření RTT je uveden v RFC-2988.

3.2.8. Vyhýbání se zahlcení

Příjemce by měl být vždy schopen přijmout najednou tolik dat, kolik inzeruje v záhlaví TCP segmentu v položce **Velikost okna**. Odesílatel tedy může vyslat najednou data až do velikosti přijímacího okna druhé stanice. Problém může nastat například v případě, kdy odesílatel je na rychlé síti a příjemce na pomalé. Směrovače na cestě si sice ukládají příchozí datagramy do vyrovnávací paměti, ale ta není neomezená. Pokud se směrovač zahltí, začne datagramy přirozeně zahazovat. Protože k odesílateli nedorazí potvrzení, začne vysílání dat po nějaké době opakovat, což vede ke značnému zpomalení. Proto se snažíme takové situaci raději vyhnout.

Pro tyto účely byly vyvinuty algoritmy nazývané jako Slow Start (Pomalý start) a Congestion Avoidance (Vyhýbání se zahlcení). Oba algoritmy jsou detailně popsané v RFC-2581.
Zmíněné algoritmy zavádí další okno na straně odesílatele, tzv. CWND (Congestion Window). Velikost CWND nahrazuje inzerovanou velikost okna příjemce (RWND). Odesílatel může vždy odeslat data maximálně do velikosti minima z hodnot CWND a RWND. Další proměnnou, která se v souvislosti s těmito algoritmy zavádí, je SSTHRESH (Slow Start Threshold) – práh pomalého startu. Proměnná SSTRESH udává hranici, za kterou je větší pravděpodobnost zahlcení. Pokud je CWND menší než SSTHRESH, použije se algoritmus pomalého startu. V opačném případě se použije algoritmus vyhýbání se zahlcení.

Algoritmus pomalého startu slouží mj. k prvotnímu zjištění hodnoty CWND. Počáteční hodnota CWND je nastavena na dvojnásobek hodnoty MSS (Maximální velikost segmentu) a počáteční hodnota SSTRESH může být nastavena např. na první inzerovanou velikost okna příjemce – RWND. S každým přijatým potvrzením, tj. s každým segmentem obsahujícím příznak ACK, se hodnota CWND zvyšuje o hodnotu MSS. To vede k exponenciálnímu růstu CWND. Algoritmus pomalého startu končí v okamžiku, kdy hodnota CWND dosáhne hodnoty SSTRESH, nebo v případě že dojde k zahlcení (vyprší RTO – Retransmittion Timeout).
Při použití algoritmu vyhýbání se zahlcení se z každým přijatým potvrzením zvyšuje hodnota CWND o hodnotu MSS * MSS / CWND, což vede k postupnému zvyšovaní CWND o čím dál menší hodnotu (hyperbolicky klesající).



Pokud stanice detekuje ztrátu segmentu pomocí vypršení RTO, nastaví hodnotu SSTRESH na maximum z poloviny odeslaných a nepotvrzených (FlightSize) dat a dvojnásobku MSS (SSTHRESS = Max(FlightSize / 2, 2 * MSS)). Proměnná CWND je v takovém případě nastavena na hodnotu MSS. Po ztrátě segmentu se tedy vždy opakuje algoritmus rychlého startu.

K uvedeným algoritmům dále patří algoritmy Fast Retransmit (Rychlé opakování) a Fast Recovery (Rychlé zotavení). Tyto algoritmy slouží k předčasnému opakování vysílaní segmentu před vypršením RTO, a tedy k celkovému zrychlení komunikace. Oba algoritmy jsou popsány v RFC-2581 a dále se jimi nebudeme zabývat, protože nejsou současnou verzí knihovny podporovány.

3.2.9. Porty

Rozhraní mezi transportní a aplikační vrstvou, tj. identifikace aplikačního protokolu, který bude poskytovanou transportní službu používat se označuje **číslem portu**. Port je abstrakce, kterou využívají protokoly transportní vrstvy k rozlišení konkrétního cílového aplikačního procesu běžícího na daném počítači.

Typy portů:

- **Známé porty** (well known) se pohybují v rozsahu 0 až 1023 a jsou protokoly pevně dané. Konkrétní hodnoty jsou uvedeny v RFC-1700.
- **Registrované porty** v intervalu 1024 až 49151 jsou užívány pro běžné procesy a aplikace.
- **Dynamické** nebo **soukromé porty** se pohybují v rozmezí 49151 až 65535

Při výběru čísla portu TCP zásuvky v rozsahu 0 až 1023, příp. 1024 až 49151 je vhodné se podívat do normy RFC-1700.

4. Konstanty a jednoduché typy

4.1. Konstanty

4.1.1. Návratové kódy CST_TCP_ERR_xxx

Knihovna CoTCP definuje několik nových návratových kódu s prefixem CST_TCP_ERR_:

CST_TCP_ERR_CONNCLOSING
Tento chybový kód mohou vrátit některé metody třídy TCoTCP v případě, že jsou volány v průběhu uzavírání spojení s cílovou stanicí. Např. tuto chybu vrátí metoda CoSendBuffer, pokud je zavolána těsně po volání metody CoDisconnect.

CST_TCP_ERR_CONNREFUSED
Chybový kód navracený metodou CoConnect v případě, že cílová stanice z jakéhokoli důvodu odmítla navázat spojení.

CST_TCP_ERR_CONNRESET
Tento návratový kód může vrátit libovolná metoda, pokud cílová stanice provede reset spojení odeslání segmentu s nastaveným příznakem RST (viz. kapitola 3.2.1).

CST_TCP_ERR_NORESPONSE
Druhá stanice neodpovídá. Tuto hodnotu mohou vrátit metody CoConnect, CoDisconnect, CoSendBuffer v případě, že byl překročen maximální počet opakovaní nepotvrzeného segmentu (viz. parametry COPT_TCP_RTLIMIT a COPT_TCP_RTSYNLIMIT v kapitole 4.1.4).

CST_TCP_ERR_PORTENGAGED
Port je obsazen. Tuto chybu vrací metoda CoConnect v případě, že spojení s aplikací na cílové stanicí a zadaném portu bylo je v současné době realizováno jinou zásuvkou se stejným lokálním portem.

Tyto návratový kódy je možné přeložit pomocí metody **CoStatusToStr** na textový řetězec.

4.1.2. Identifikátory tříd zařízení DEV_CLASS_xxx

Třída TCoTcp (protokol TCP) resp. TCoTcpMux (multiplexer protokolu TCP) má přidělen identifikátor **DEV_CLASS_TCP** resp. **DEV_CLASS_TCPMUX**. Tyto identifikátory určují zařízení v řetězci při volání funkcí **CoIoctl**, **CoGetOption**, **CoSetOption** apod. Zaregistrované jména tříd pro použití s funkcí **CoCreateDevice** jsou **TCP** a **TCPMUX**.

4.1.3. Identifikátory řídících operací IOCTL_TCP_xxx

Současná verze knihovny CoTCP nedefinuje žádné vlastní identifikátory IOCTL_TCP_xxx.

4.1.4. Identifikátory parametrů zařízení COPT_TCP_xxx

Identifikátory s prefixem **COPT_** specifikují parametr zařízení ve funkci **CoGetOption** příp. **CoSetOption**. Knihovna CoTCP definuje několik nových konstant s prefixem COPT_UDP_:

COPT_TCP_LPORT

Implicitní TCP port, na kterém bude TCP zařízení přijímat příchozí spojení a kterým budou označeny odchozí segmenty. Tato hodnota koresponduje s parametrem LPORT v konfiguračním textovém řetězci. Parametr lze měnit v libovolném okamžiku, změna se však projeví až v okamžiku volání metody CoBind (pokud místo adresy uvedeme hodnotu **nil**)

Formát dat: Word Implicitní hodnota: 5000

COPT_TCP_RADDR

Implicitní adresa protistanice. Parametr lze měnit v libovolném okamžiku, změna se však projeví až v okamžiku volání metody CoConnect (pokud na místě parametru AAddress uvedeme místo konkrétní adresy uvedeme hodnotu **nil**).

Formát dat: TTcpAddress Implicitní hodnota: 0.0.0.0:5000

COPT_TCP_DACK

Zpoždění potvrzení segmentu v milisekundách (viz. kapitola 3.2.5). Parametr lze měnit v libovolném stavu zařízení TCP. V případě, že je parametr nastaven na nulu, jsou segmenty potvrzovány ihned.

Formát dat: TCoTime Implicitní hodnota: 200
Min: 0
Max: 500

COPT_TCP_NAGLE

Zpoždění odeslání dat v ms (viz. kapitola 3.2.6). Parametr lze měnit v libovolném stavu zařízení TCP. V případě, že je parametr nastaven na nulu, algoritmus zpožděného odesílání dat se neuplatní.

Formát dat: TCoTime Implicitní hodnota: 400
Min: 0
Max: 1000

COPT_TCP_TIMEWAIT

Čekání ve stavu TIME_WAIT v sekundách. Podle standardu RFC-795 by TCP modul měl v některých případech po uzavření spojení čekat po dobu minimálně dvou minut, než bude možné navázat spojení na stejném portu. Tento požadavek je ve většině případů příliš přísný, a proto jej knihovna

nedodržuje. Implicitně je toto čekání nastaveno na 0ms. Parametr lze měnit v libovolném stavu zařízení TCP

Formát dat: Word
Implicitní hodnota: 0
Min: 0
Max: 240

COPT_TCP_RTLIMIT

Počet opakování segmentu při nedoručení potvrzení, než je indikována chyba CST_TCP_ERR_NORESPONSE. Parametr lze měnit v libovolném stavu zařízení TCP

Formát dat: Word
Implicitní hodnota: 10
Min: 3
Max: 32

COPT_TCP_RTSYNLIMIT

Počet opakování segmentu při nedoručení potvrzení, při navazování spojení s cílovou stanicí, než je indikována chyba CST_TCP_ERR_NORESPONSE. Parametr lze měnit v libovolném stavu zařízení TCP

Formát dat: Word
Implicitní hodnota: 3
Min: 1
Max: 32

COPT_TCP_MINWND
COPT_TCP_MAXWND

Minimální a maximální velikost přijímacího okna v bajtech. Viz. kapitola 3.2.3. Současná implementace protokolu neprovádí dynamickou změnu přijímacího okna, okno má pevnou velikost a oba dva parametry musí být nastaveny na stejnou hodnotu.

Formát dat: Word
Implicitní hodnota: 4096
Min: 1024
Max: 32768

4.1.5. Upřesňující příznaky volání operací COF_TCP_xxx

Knihovna CoTCP definuje dva nové příznaky COF_TCP_PUSH a COF_TCP_COPY, které lze použít k upřesnění chování metody **CoSendBuffer** (parametr AFlags).

COF_TCP_PUSH

Poslední segment dat odesílaných metodou CoSendBuffer bude označen příznakem PSH, tzn. že bude v cílové stanici okamžitě po přijetí předán aplikační vrstvě.

COF_TCP_COPY

Odesílaná data budou při volání metody CoSendBuffer překopírována do vyrovnávací paměti. Metoda CoSendBuffer se vrátí prakticky okamžitě poté, co jsou data překopírována a vrátí návratový kód CST_SUCCESS.

4.2. Typy

4.2.1. Třída TCoTcp

```
TCoTcp = object( TCoDevice );
```

Pokud použijeme knihovnu CoTCP (uvedeme ji v seznamu za klíčovým slovem **uses**). Bude tato třída zaregistrovaná v globálním seznamu tříd zařízení pod jménem **TCP**. Toto jméno lze použít v případě vytváření třídy pomocí funkce CoCreateDevice. Číselný identifikátor třídy pro použití s metodami CoIoctl, CoGetOption, CoSetOption apod. je **DEV_CLASS_TCP**.

4.2.2. Třída TCoTcpMux

```
TCoTcpMux = object( TCoTcpMux );
```

Pokud použijeme knihovnu CoTCP (uvedeme ji v seznamu za klíčovým slovem **uses**). Bude tato třída zaregistrovaná v globálním seznamu tříd zařízení pod jménem **TCPMUX**. Toto jméno lze použít v případě vytváření třídy funkcí **CoCreateDevice**. Číselný identifikátor třídy je **DEV_CLASS_TCPMUX**.

4.2.3. Struktura TTcpAddress

```
PTcpAddress = ^TTcpAddress;
TTcpAddress = packed record
  Size    : Byte;
  Address : TIpAddress;
  Port    : Word;
end;
```

Struktura **TTcpAddress** popisuje adresu na transportní vrstvě pro protokol TCP, která se použije při volání aplikačních metod třídy TCoTcp na místech, kde se vyskytuje typ **TCoAddress** (tj. v metodách CoSendBufferTo, CoRecvBufferFrom, CoBind a CoConnect).

Položka **Size** udává velikost struktury TTcpAddress. Před použitím struktury se vždy nezapomeňte ujistit, že položka **Size** obsahuje správnou hodnotu, tedy SizeOf(TTcpAddress). V položce **Address** je obsažena adresa uzlu, v položce **Port** je uloženo číslo Portu.

5. Textový konfigurační řetězec

Parametry zařízení TCoTcp lze nastavovat pomocí textového konfiguračního řetězce metodami **CoSetOptionString**.

V následujícím seznamu jsou uvedeny všechny povolené identifikátory parametrů:

LPORT Implicitní TCP port, který se použije v případě, že se zavolá metoda **CoBind** bez specifikované adresy. Parametr LPORT lze měnit v kterémkoli okamžiku, změna se však promítne až při volání metody **CoBind**.

Formát dat: číslo
Implicitní nastavení: 5000

RADDR Implicitní adresa cílové stanice v případě, že se zavolá metoda **CoConnect** bez specifikované adresy. Parametr RADDR lze měnit v kterémkoli okamžiku, změna se však promítne až při volání metody **CoConnect**.

Formát dat: text, IP adresa
Implicitní nastavení: 0.0.0.0

RPORT Implicitní port na cílové stanici v případě, že se zavolá metoda **CoConnect** bez specifikované adresy. Parametr RPORT lze měnit v kterémkoli okamžiku, změna se však promítne až při volání metody **CoConnect**.

Formát dat: číslo
Implicitní nastavení: 5000

DACK Zpoždění odeslání dat v milisekundách (viz. kapitola 3.2.6). Parametr lze měnit v libovolném stavu zařízení TCP. V případě, že je parametr nastaven na nulu, algoritmus zpožděného odesílání dat se neuplatní.

Formát dat: číslo
Implicitní nastavení: 200 Min: 0 Max: 500

NAGLE Zpoždění odeslání dat v milisekundách (viz. kapitola 3.2.6). Parametr lze měnit v libovolném stavu zařízení TCP. V případě, že je parametr nastaven na nulu, algoritmus zpožděného odesílání dat se neuplatní.

Formát dat: číslo
Implicitní nastavení: 400 Min: 0 Max: 1000

TIMEWAIT Čekání ve stavu TIME_WAIT v sekundách. Podle standardu RFC-795 by TCP modul měl v některých případech po uzavření spojení čekat po dobu minimálně dvou minut, než bude možné navázat spojení na stejném portu. Tento požadavek je ve většině případů příliš přísný, a proto jej knihovna nedodržuje. Implicitně je toto čekání nastaveno na 0ms. Parametr lze měnit v libovolném stavu zařízení TCP

Formát dat: číslo
Implicitní nastavení: 0 Min: 0 Max: 240

RTLIMIT Počet opakování segmentu při nedoručení potvrzení, než je indikována chyba CST_TCP_ERR_NORESPONSE. Parametr lze měnit v libovolném stavu zařízení TCP.

Formát dat: číslo
Implicitní nastavení: 10, Min: 3,Max: 32

RTSYNLIMIT Počet opakování segmentu při nedoručení potvrzení, při navazování spojení s cílovou stanicí, než je indikována chyba CST_TCP_ERR_NORESPONSE. Parametr lze měnit v libovolném stavu zařízení TCP

Formát dat: číslo
Implicitní nastavení: 3, Min: 1,Max: 32

MINWND
MAXWND Minimální a maximální velikost přijímacího okna v bajtech. Viz. kapitola 3.2.3. Současná implementace protokolu neprovádí dynamickou změnu přijímacího okna, okno má pevnou velikost a oba dva parametry musí být nastaveny na stejnou hodnotu.

Formát dat: číslo
Implicitní nastavení: 4096 Min: 1024 Max: 32768

Současná verze zařízení TCoTcpMux nemá žádné parametry a tudíž jí není potřeba konfigurovat pomocí textového konfiguračního řetězce.

Příklad:

```
var
  Sock : PCoDevice;

.
.

Sock^.CoSetOptionString( DEV_CLASS_UDP,  'LPORT=5001', nil );
```

6. Poznámky

6.1. Inicializace zařízení

Metoda **CoBind** provádí inicializace instance zařízení TCP protokolu a zaregistruje vyžádaný lokální port u zařízení nižší vrstvy (TCP multiplexeru).

První parametr metody **CoBind**, tj. AAddress je ukazatel na lokální adresu (strukturu TTcpAddress). Protože, lokální IP adresu udává vrstva IP protokolu. Pouze položky Protocol a Size struktury TTcpAddress musí být vyplněny, položku Address lze naplnit hodnotu IP_ADDR_ANY. Pokud místo ukazatele na strukturu TTcpAddress předáme hodnotu **nil**, pak se použije implicitní nastavení lokálního portu podle implicitního nastavení adresy cílové stanice (viz. COPT_TCP_LPORT nebo parametr konfiguračního textového řetězce LPORT).

6.2. Deinicializace zařízení

Metoda **CoUnbind** slouží k zrušení lokální adresy zařízení, uvolnění alokovaných prostředků a přepnutí zařízení do neaktivního stavu.

V případě, že zařízení je ve stavu s navázaným spojením, pak metoda **CoBind** způsobí odeslání segmentu s nastaveným RST příznakem. Tj. okamžité zrušení spojení bez řádného ukončení.

6.3. Navazování spojení

TCP protokol umožňuje dva způsoby navazování spojení:

- Aktivní navázání spojení pomocí **CoConnect**
- Pasivní čekaní na spojení a jeho potvrzení pomocí **CoAccept**

6.3.1. Navazování spojení pomocí CoConnect

Pomocí metody **CoConnect** se navazuje spojení s cílovou stanicí. Tuto metodu volá ta stanice, která iniciuje navázání spojení. Pokud spojení navazuje cílová stanice použijte metodu **CoAccept**. Spojení je jednoznačné identifikováno čtveřicí: lokální adresou, lokálním portem, adresou cílové stanice a portem na cílové stanici.

První parametr metody **CoConnect** je ukazatel na adresu cílové stanice (strukturu TTcpAddress). Struktura TTcpAddress musí mít před voláním metody **CoConnect** vyplněné všechny položky, tj. Size, Address a Port. Pokud místo ukazatele na strukturu TTcpAddress předáme hodnotu **nil**, pak se použije implicitní nastavení adresy cílové stanice (viz. COPT_TCP_RADDR, nebo parametry konfiguračního textového řetězce RADDR, RPORT).

6.3.2. Navazování spojení pomocí metod CoAccept a CoListen

Metoda **CoAccept** slouží k potvrzení spojení navazovaného jinou stanicí. Před

voláním metody **CoAccept**, je potřeba převést zařízení TCP do stavu pasivního čekání na příchozí spojení pomocí volání metody **CoListen**.

Jedním z parametrů metody **CoListen** je ABackLog. Parametr ABackLog udává maximální délku fronty čekajících požadavků na navázání spojení nepotvrzených metodou **CoAccept**. Úspěšné volání metody **CoAccept** vytvoří novou TCP zásuvku, která dokončí navázání spojení. Zásuvka jejíž metoda **CoAccept** byla zavolána, zůstává ve stavu pasivního čekání na příchozí spojení. V případě, že hodnota parametru ABackLog je nulová, pak spojení naváže přímo ta zásuvka jejíž metoda **CoAccept** byla zavolána.

6.4. Uzavírání spojení

Uzavření dříve navázaného spojení pomocí metody **CoConnect** nebo **CoAccept** se provádí pomocí metody **CoDisconnect**. Volání metody Disconnect zajistí odvysílání všech dat předaných TCP zásuvce pomocí metody **CoSendBuffer**. Neblokující operace přijmu dat (**CoReceiveBuffer**) jsou ukončeny a vrací návratový kód CST_TCP_ERR_CONNCLOSING.

6.5. Posílání a příjem dat

K posílání a příjmu dat jsou určené metody **CoSendBuffer** a **CoRecvBuffer**. Data lze přijímat a posílat pouze v případě navázaného spojení. Metody **CoSendBufferTo** a **CoRecvBufferFrom** nelze použít. Chování metody CoSendBuffer lze modifikovat pomocí příznaků AFlags – COF_TCP_PUSH a COF_TCP_COPY (viz. kapitola 4.1.5).

7. Příklad

Následující příklad ukazuje jak použít TCP zásuvku nad TCP/IP zásobníkem. Příklad naváže spojení se s cílovou stanicí, odešle připravená data a čeká na příjem libovolných dat. Poté ukončí spojení.

```
program TcpSample;

uses
  CoBase,
  CoEth01,
  CoIPv4,
  CoTCP,
  CoINET;


var
  Status    : TCoStatus;            { Stav vysledku operace         }
  Sock      : PCoDevice;            { Ukazatel na instanci zarizeni }
  Data      : array[0..1023] of Byte; { Data ramce                  }
  RecvLen   : Word;                 { Delka prijateho ramce         }
  SAddress  : TTcpAddress;          { Cilova adresa                 }


  { Procedura pripravujici data odesilaneho ramce }
  procedure ProduceData;
  begin
    { ... }
  end;

  { Procedura zpracovavajici data prijateho ramce }
  procedure ConsumeData;
  begin
    { ... }
  end;

begin

  { Vytvoreni TCP/IP zasobniku nad zarizenim ETH01 s bazovou adresou
    $2300. IP protokol bude nakonfigurovan na IP adresu 192.168.1.210
    a masku 255.255.255.0 }

  Status := NetOpenStack( 'ETH01', 'IOBASE=$2300',
             'IPADDR=”192.168.1.210” NETMASK=”255.255.255.0”', '' );

  if Status <> CST_SUCCESS then
  begin
    WriteLn( 'NetOpenStack() failed: ', CoStatusToStr( Status ) );
  end
  else
  begin
    { TCP/IP stack byl uspesne vytvoren }

    { Vytvorime TCP zasuvku naslouchajici na lokalni portu 5000 }
    Status := NetOpenSocket( 'TCP', 'LPORT=5000', '', Sock );

    if Status <> CST_SUCCESS then
    begin
    end
    else
    begin
```

```
{ Inicializace vsech vrstev TCP/IP zasobniku }
Status := Sock^.CoBind( nil, COF_PROPAGATE, nil );

if Status = CST_SUCCESS then
begin
  { Navazani spojeni }

  with SAddress do
  begin
    Size    := SizeOf( TTcpAddress );
    Address := MakeIpAddr( 192, 168, 1, 1 ); { Adresa stanice }
    Port    := 5000;                         { Port           }
  end;

  Status := Sock^.CoConnect( @SAddress, 0, nil );

  if Status <> CST_SUCCESS then
  begin
    WriteLn( 'CoConnect() failed: ',
      Sock^.CoStatusToStr( Status ) );
  end
  else begin
   { Spojeni bylo navazano }

   { Naplneni dat urcenych k odvysilani }
   ProduceData;

   { Odesleme blok dat }

   Status := Sock^.CoSendBuffer( @Data, SizeOf(Data), 0, nil );

   if Status <> CST_SUCCESS then
   begin
     WriteLn( 'CoSendBuffer() failed: ',
       Sock^.CoStatusToStr( Status ) );
   end;

   Status := Sock^.CoRecvBuffer( @Data, SizeOf(Data),
               @RecvLen, 0, nil );

   if Status <> CST_SUCCESS then
   begin
     WriteLn( 'CoRecvBuffer() failed: ',
       Sock^.CoStatusToStr( Status ) );
   end
   else begin
     { Zpracovani prijatych dat }
     ConsumeData;
   end;

   { Zaver spojeni }

   Status := Sock^.CoDisconnect( 0, nil );

   if Status <> CST_SUCCESS then
   begin
     WriteLn( 'CoDisconnect() failed: ',
       Sock^.CoStatusToStr( Status ) );
   end;
  end;

   { Deinicializace vsech vrstvev TCP/IP zasobniku }
  Status := Sock^.CoUnbind( COF_PROPAGATE, nil );
end;
```

```
      { Uvolneni zarizeni TCP zasuvky }
      NetCloseSocket( Sock );
    end;

    { Uvolneni vsech ostatnich zarizeni TCP/IP stacku }
    NetCloseStack( '' );
  end;

end.
```