TRANSPORTNI SLOJ

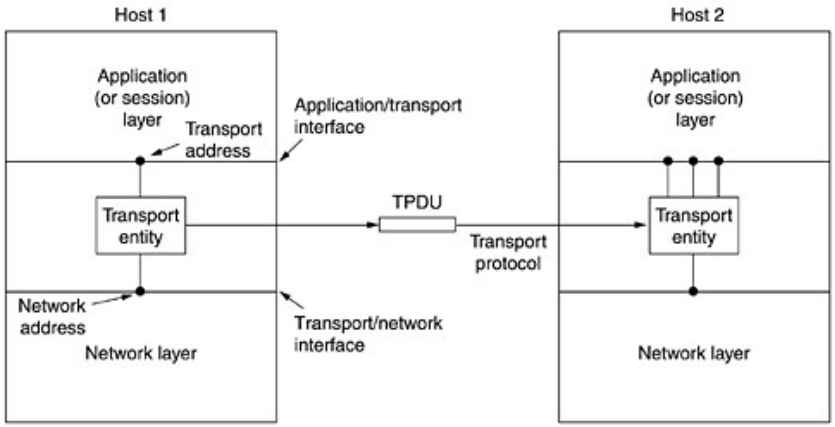
1. Namena transportnog sloja

- Transportni sloj nije još samo jedan od mnogih slojeva jer je njegova uloga sama srž hijerarhije protokola. On treba da **obezbedi pouzdan i isplativ prenos podataka** bez obzira na fizičku mrežu ili mreže koje se trenutno nalaze između izvorišnog i odredišnog čvora.
- Ovde će biti obrađene isključivo usluge prenosa koje transportni sloj pruža višim slojevima, a pre svega aplikacionom sloju, bez dodatnih detalja u vezi sa arhitekturom samog transportnog sloja.

2. Usluge koje se obezbeđuju za više slojeve

- Da bi se obavio zadatak obezbeđivanja efikasne i pouzdane usluge prenosa podataka, transportni sloj koristi usluge mrežnog sloja. Hardver i/ili softver unutar tranbsportnog sloja koji obavlja prenos zove se **transportnom jedinicom** (*transport entity*). Može da se nalazi u jezgru OS-a, u biblioteci ili na mrežnoj kartici
- Logički odnosi između slojeva prikazani su na slici:

Figure 6-1. The network, transport, and application layers.



- Kao što postoje dve vrste usluga mrežnog sloja, sa uspostavljanjem direktne veze i bez nje, tako postoje i **dve vrste usluga transportnog sloja** koje se oslanjaju na pomenute usluge mrežnog sloja. Rad se u oba slučaja svodi na **tri faze**: uspostavljanje veze, prenos podataka i raskidanje veze.
- Pošto usluga koju daje transportni sloj jako liči na uslugu koju daje mrežni sloj, **pitanje je zašto uopšte postoje dva različita sloja**?
- Odgovor proističe iz činjenice da **kod mrežnog sloja radi uglavnom u ruterima** koje održavaju provajderi, dok **kod transportnog sloja uglavnom radi na korisničkim računarima**.
- Pošto se unutar mrežnog sloja mogu javiti problemi u prenosu, a korisnici nemaju stvarnog uticaja na mrežni sloj, jedino rešenje je obezbeđivanje novog sloja koji bi poboljšao kvalitet usluge.
- **Postojanje transportnog sloja u suštini omogućava da usluga prenosa bude pouzdanije od mrežnog sloja na koji se oslanja**.

3. Osnovne operacije u uslugama prenosa

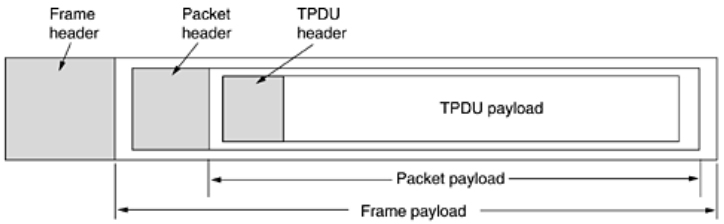
- Da bi korisnicima omogućio pristup uslugama prenosa, transportni sloj mora da obezbedi neke osnovne operacije (primitive), tj. interfejs ka uslugama prenosa.
- Osnovni transportni interfejs je zaista rudimentaran, ali većini aplikacija sasvim dovoljan. Evo kako izgleda:

Figure 6-2. The primitives for a simple transport service.

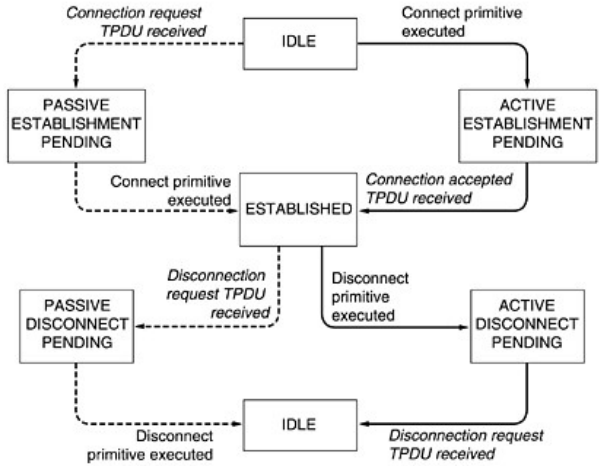
Primitive	Packet sent	Meaning
LISTEN	(none)	Block until some process tries to connect
CONNECT	CONNECTION REQ.	Actively attempt to establish a connection
SEND	DATA	Send information
RECEIVE	(none)	Block until a DATA packet arrives
DISCONNECT	DISCONNECTION REQ.	This side wants to release the connection

- Najpre server izvršava operaciju LISTEN koja sistemski blokira server dok se ne pojavi neki klijent. Kada se klijent pojavi, izvršava operaciju CONNECT, pri čemu je u koristan teret ovog paketa kapsulirana poruka za transportnu jedinicu servera.
- Jedinica prenosa u transportnom protokolu je tzv. **TPDU** (*Transport Protocol Data Unit*), a evo kako izgleda ugnežđivanje TPDU poruka u IP pakete, a potom u okvire:

Figure 6-3. Nesting of TPDUs, packets, and frames.



- Dakle, operacija CONNECT na klijentu šalje TPDU poruku CONNECTION REQUEST serveru. Server se deblokira i klijentu šalje TPDU poruku CONNECTION ACCEPTED.
- Sada, kada je veza uspostavljena strane međusobno razmenjuju poruke korišćenjem operacija SEND i RECEIVE. U najjednostavnijem slučaju, jedna strana može da se blokira operacijom RECEIVE i da čeka drugu stranu da izvrši SEND. Sve dobro radi dok obe strane vode računa o redosledu slanja.
- Kada veza više nije potrebna raskida se TPDU porukom DISCONNECT. Način raskidanja može biti **asimetričan** i **simetričan**. Na slici je dat dijagram stanja veze:



- Na slici je pretpostavljeno simetrično raskidanje veze, dok pune linije označavaju redosled stanja klijenta, dok isprekidane linije prikazuju redosled stanja servera.

4. Berkli utičnice (*Berkeley sockets*)

- Nakon baznog teorijskog modela osnovnih operacija prenosa, biće uvedene osnovne operacije u modelu Berkeley UNIX-a i današnji su standard za sve UNIX-olike operativne sisteme. Berkeley primitive su date u sledećoj tabeli:

Figure 6-5. The socket primitives for TCP.

Primitive	Meaning
SOCKET	Create a new communication end point
BIND	Attach a local address to a socket
LISTEN	Announce willingness to accept connections; give queue size
ACCEPT	Block the caller until a connection attempt arrives
CONNECT	Actively attempt to establish a connection
SEND	Send some data over the connection
RECEIVE	Receive some data from the connection
CLOSE	Release the connection

- Prve četiri operacije izvršavaju serveri navedenim redom. **Operacija SOCKET** pravi novu utičnicu, pri čemu se u parametrima zadaju format

adrese, vrsta usluge (npr. pouzdan tok bitova) i protokol. Uspešno izvedena operacija SOCKET vraća običan C-ovski fajl deskriptor.

- Novoformirani soketi nemaju mrežne adrese, već im se one dodeljuju **operacijom BIND**. Kada server soketu pridruži adresu na nju se tada mogu priključivati udaljeni klijenti.
- Zatim se poziva **operacija LISTEN** koja za dolazne pozive dodeljuje prostor u redu čekanja ukoliko više klijenata pokušava da uspostavi vezu. U ovom modelu LISTEN ne blokira server.
- Da bi se blokirao dok ne stigne neki poziv, server poziva **operaciju ACCEPT**. Ona vraća standardni deskriptor datoteka koji se može koristiti za čitanje i upisivanje.

- Šta se za to vreme dešava **kod klijenta**?
- I klijent mora da napravi soket pozivanjem **operacije SOCKET**, dok operacija BIND nije potrebna jer dodeljena adresa serveru ništa ne znači.
- **Operacija CONNECT** blokira pozivaoca i aktivno započinje proces povezivanja.
- Kada se proces povezivanja završi, klijent dobija odgovarauću TPDU poruku i deblokira se, tako da je veza uspostavljena.
- Obe strane sada mogu da šalju (**SEND**) i primaju (**RECEIVE**) podatke **potpunom dupleksnom vezom**.
- Umesto SEND i RECEIVE mogu se koristiti i standardni UNIX sistemski pozivi **WRITE** i **READ**.
- Veza se raskida simetrično, kada obe strane izvrše **operaciju CLOSE**.

5. Primer soket programiranja - server fajlova - serverski deo

Zadatak je isprogramirati sasvim elementarni server fajlova i jednog klijenta koji ga koristi. Program je napisan u C-u, a može se kompajlirati standardnim gcc kompajlerom uz linkovanje biblioteke *libnsl* (*Name Server Library*):

```
gcc -o server server.c -lnsl
```

Evo serverskog koda:

```
#include <sys/types.h>                          /* Serverski kod */
#include <sys/fcntl.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345       /* proizvoljno, ali isto za klijent i server */
#define BUF_SIZE 4096                          /* velicina bloka */
#define QUEUE_SIZE 10

int main(int argc, char *argv[])
{
 int s, b, l, fd, sa, bytes, on = 1;
 char buf[BUF_SIZE];                           /* bafer za fajl koji se salje */
 struct sockaddr_in channel;                   /* sadrzi IP adresu */

 /* Napravi adresnu strukturu i povezi je sa soketom. */
 memset(&channel, 0, sizeof(channel));         /* nulovanje channel-a */
 channel.sin_family = AF_INET;
 channel.sin_addr.s_addr = htonl(INADDR_ANY);
 channel.sin_port = htons(SERVER_PORT);

 /* Pasivno otvaranje soketa. Cekanje konekcije. */
 s = socket(AF_INET, SOCK_STREAM, IPPROTO_TCP);   /* pravljenje soketa */
 if (s < 0) fatal("socket failed");
 setsockopt(s, SOL_SOCKET, SO_REUSEADDR, (char *) &on, sizeof(on));

 b = bind(s, (struct sockaddr *) &channel, sizeof(channel));
 if (b < 0) fatal("bind failed");
```

```
 l = listen(s, QUEUE_SIZE);          /* specificiranje velicine reda cekanja */
 if (l < 0) fatal("listen failed");

 /* Soket je napravljen, cekaj konekciju */
 while (1) {
     sa = accept(s, 0, 0);                  /* blokiranje do poziva klijenta */
     if (sa < 0) fatal("accept failed");

     read(sa, buf, BUF_SIZE);               /* citanje imena fajla iz soketa */

     /* Get and return the file. */
     fd = open(buf, O_RDONLY);              /* otvori fajl i posalji klijentu */
     if (fd < 0) fatal("open failed");

     while (1) {
          bytes = read(fd, buf, BUF_SIZE); /* citanje iz fajla */
          if (bytes <= 0) break;            /* ispitivanje za kraj fajla */
          write(sa, buf, bytes);            /* pisanje bajtova u soket */
     }
     close(fd);                             /* zatvori fajl */
     close(sa);                             /* zatvori konekciju */
 }
}

fatal(char *string)
{
      printf("%s\n", string);
      exit(1);
}
```

Kod počinje nekim standardnim zaglavljima od kojih tri sadrže definicije entiteta vezanih za Internet. Zatim dolazi definicija SERVER_PORT koja definiše broj serverskog priključka. Analogija za portove (priključke) su lokali na telefonskoj centrali. Broj je proizvoljno izabran, a dobar je svaki od 1024 do 65535, ukoliko ga ne koristi neki drugi proces.

Zatim se definiše veličina bafera za prenos i maksimalni broj veza koje čekaju na uspostavljanje.

Nakon definicija, sledi sam kod. Pre pravljenja soketa se inicijalizuje se nulom struktura koja čuva IP adresu - *channel*. Funkcije *htonl* i *htons* pretvaraju vrednosti u standardni format tako da mogu da ga koriste i računari sa *big-endian* i sa *little-endian* sistemima.

Posle toga, server konačno pravi soket i proverava greške. Pozivanje funkcije *setsockopt* neophodno je da bi se soket iznova mogao koristiti, tj.

server će prihvatati jedan zahtev za drugim dok se ne prekine. Sada se IP adresa pridružuje soketu funkcijom *bind*, da bi potom usledio poziv funkciji listen kojom se objavljuje spremnost servera da prihvati pozive i postavlja se veličina reda čekanja.

Sada je server spreman da uđe u glavnu petlju koju nikada ne napušta. Poziv proceduri *accept* blokira server sve dok se ne prijavi neki klijent. U kodu koji sledi se vidi da je rad sa soketima identičan radu sa fajlovima u C-u. Server prihvata (*read*) ime datoteke, otvara je u skladu sa dozvolama UNIX korisnika koji je pokrenuo server i preko bafera je šalje na izlaznu liniju.

6. Primer soket programiranja - server fajlova - klijentski deo

Klijentski deo koda se nakon kompajliranja i linkovanja koristi na sledeći način:

```
gcc -o client client.c -lnsl
./client 147.91.204.77 /home/student/fajl.txt
```

Gornji poziv ispisuje sadržaj fajla /home/student/fajl.txt koji se nalazi na serverskom računaru na konzolu klijentskog računara, naravno, ako navedeni fajl postoji. Evo samog koda i dodatnih komentara vezanih za klijentski deo:

```
/* Klijentski program za server fajlova */

#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

#define SERVER_PORT 12345       /* proizvoljno, ali isto za klijent i server */
#define BUF_SIZE 4096                           /* velicina bloka za transfer */

int main(int argc, char **argv)
{
 int c, s, bytes;
 char buf[BUF_SIZE];                            /* bafer za fajl koji se prima */
 struct hostent *h;                     /* struktura za informacije o serveru */
 struct sockaddr_in channel;                    /* struktura za IP adrese */

 if (argc != 3) fatal("Usage: client server-name file-name");
 h = gethostbyname(argv[1]);                    /* pretrazi adresu servera */
 if (!h) fatal("gethostbyname failed");

 s = socket(PF_INET, SOCK_STREAM, IPPROTO_TCP);
 if (s <0) fatal("socket");
 memset(&channel, 0, sizeof(channel));
 channel.sin_family= AF_INET;
 memcpy(&channel.sin_addr.s_addr, h->h_addr, h->h_length);
 channel.sin_port= htons(SERVER_PORT);
```

```
 c = connect(s, (struct sockaddr *) &channel, sizeof(channel));
 if (c < 0) fatal("connect failed");

 /* Konekcija je uspostavljena. Posalji ime fajla sa nulom na kraju. */
 write(s, argv[2], strlen(argv[2])+1);

/* Uzmi sadrzaj fajla sa soketa i ispisi ga na stdout */
 while (1) {
     bytes = read(s, buf, BUF_SIZE);             /* citaj iz soketa */
     if (bytes <= 0) exit(0);                    /* proveri da li je kraj */
     write(1, buf, bytes);                       /* pisi na stdout (ekran) */
 }
}

fatal(char *string)
{
      printf("%s\n", string);
      exit(1);
}
```

Klijentski deo počinje isto kao i serverski, navođenjem odgovarajućih zaglavlja. Izvršavanje počinje proverom sintakse poziva (*argc*==3 znači ime programa plus dva argumenta). Treba obratiti pažnju na funkciju *gethostbyname* koja kontaktira DNS server kako bi tekstualnu adresu pretvorila u IP adresu.

Zatim se pravi i inicijalizuje soket, posle čega klijent pokušava da uspostavi vezu koristeći proceduru *connect*. Pošto je veza uspostavljena, na soket se šalje ime datoteke koja se traži. Broj poslatih bajtova za jedan je veći od dužine stringa, jer se šalje i završna nula kako bi server znao gde je kraj imena datoteke.

Klijent sada ulazi u petlju gde blok po blok prihvata podatke koje server šalje i ispisuje ih na stdout. Kada to uradi, samo završi program, dok server na drugom kraju ostaje i dalje aktivan i spreman da prihvati nove klijente.

7. Soketi u Javi - serverski deo

```
//: tcpServer0.java
//  SYNOPSIS:      tcpServer0
//  DESCRIPTION:   TProgram kreira tcp soket na internet domenu
//                 povezuje ga na port 12345, slusa i prihvata
//                 konekciju od TcpClient-a, nakon cega mu on salje
//                 ime fajla. Server otvara trayeni fajl i salje
//                njegov sadrzaj klijentu.
///////////////////////////////////////////////////////////////////////

import java.io.*;
import java.net.*;

public class TcpServer {
	public static final int PORT = 12345;

	public static void main(String[] args) throws IOException {

		ServerSocket serverSocket = new ServerSocket(PORT);
		BufferedReader fileReader = null;
		BufferedReader in = null;
		PrintWriter out = null;

		while (true) {
			// Blokira se do klijentskog zahteva za konekcijom
			Socket socket = serverSocket.accept();

			try {
				in = new BufferedReader(new InputStreamReader(socket.getInputStream()));

				out = new PrintWriter(new BufferedWriter(
						new OutputStreamWriter(socket.getOutputStream())), true);

				// uzmi ime fajla
				String strLine = in.readLine();

				// Otvori fajl kao BufferedReader
				fileReader = new BufferedReader(new InputStreamReader(
						new DataInputStream(new FileInputStream(strLine))));

				// Citaj fajl liniju po liniju i salji na soket
				while ((strLine = fileReader.readLine()) != null)
					out.println(strLine);

			} finally {
				socket.close();
				fileReader.close();
			}
		} // while true
```

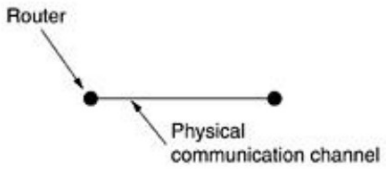
```
        }
}
```

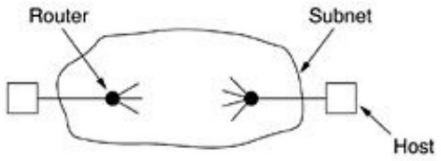
8. Soketi u Javi - klijentski deo

```
import java.net.*;
import java.io.*;

public class TcpClient {

	// Port je konstanta, mora da bude isti za klijent i server
	private static final int PORT = 12345;

	public static void main(String[] args) throws IOException {

		if (args.length != 2) {
			System.out.println("Koriscenje: java TcpClient hostname filename");
			System.exit(0);
		}

		// Konekcija na dati host
		InetAddress address = InetAddress.getByName(args[0]);

		// Pokusaj konekcije na TcpServer
		Socket socket = new Socket(address, PORT);

		try {
			PrintWriter out = new PrintWriter(new BufferedWriter(
					new OutputStreamWriter(socket.getOutputStream())), true);

			BufferedReader in = new BufferedReader(new InputStreamReader(socket
					.getInputStream()));

			// posalji ime fajla
			out.println(args[1]);
			String strLine;

			// citaj liniju po liniju i pisi na ekranu
			while ((strLine = in.readLine()) != null)
				System.out.println(strLine);

		} finally {
			socket.close();
		}
	}
}
```

ELEMENTI TRANSPORTNIH PROTOKOLA

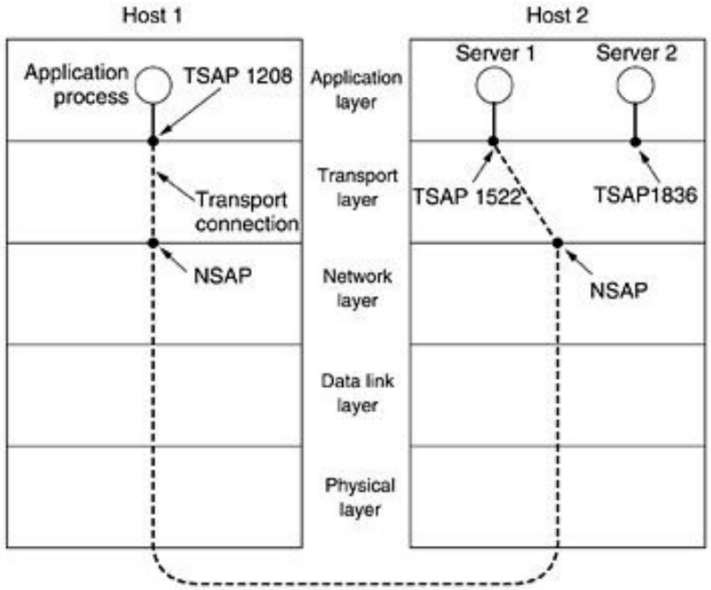
- Koje su sličnosti a koje razlike između veze ostvarene u sloju veze podataka i veze u transportnom sloju?
  1. Adresiranje
  2. Uspostavljanje veze
  3. Potencijalni skladišni kapacitet podmreže – pojavljivanje paketa nakon nekog vremena
  4. Zbog većeg broja veza u transportnom sloju, potrebno je dinamički dodeljivati bafere umesto statički kako je to rađeno u sloju veze podataka



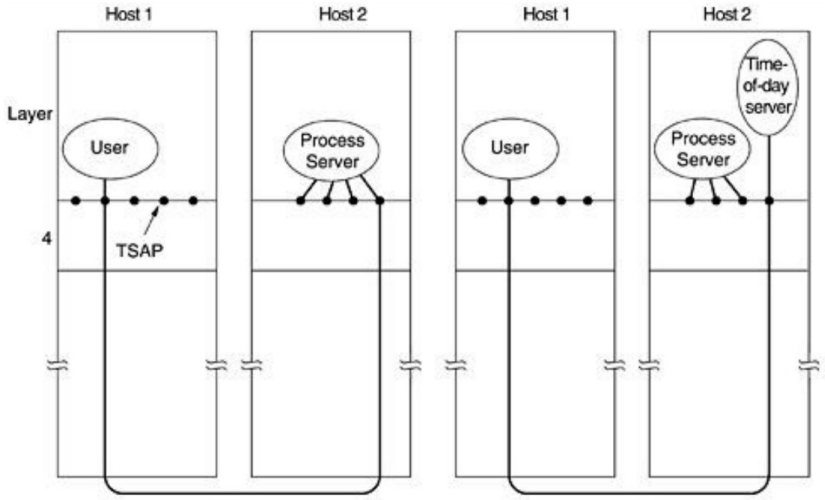


Adresiranje

- Krajnje tačke za povezivanje u mrežnom sloju zovu se **NSAP (*Network Service Access Point*)**. Primer NSAP-a je IP adresa.
- Sa druge strane, transportna jedinica poseduje **portove** (priključke), gde svaki od njih predstavlja po jedan **TSAP (*Transport Service Access Point*)**
- Dakle, potrebno je multipleksiranje jer više TSAP-ova koristi jedan jedini NSAP. O tome se brine transportna jedinica



- Kako klijent da zna na koji port servera treba da se poveže da bi dobio određenu uslugu? Na UNIX-u postoji **/etc/services** koji definiše portove za pojedine opšte-poznate servere
- Da li je potrebno da svi serveri koji ikad mogu da zatrebaju zauzimaju sisetmske resurse? I za to postoji rešenje: tzv. **Superserver**. Uloga superservera je da sluša zahteve na predefinisanim portovima, a zatim po potrebi pokreće i, nakon nekog perioda neaktivnosti, gasi serverske procese. Na UNIX-u tu ulogu igra **xinetd** (***extended Internet daemon***).


Host 1
Host 2
Host 1
Host 2
Layer
4
User
Process Server
TSAP
User
Process Server
Time-of-day server