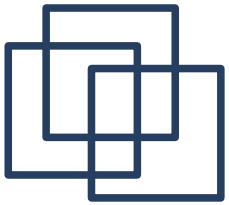


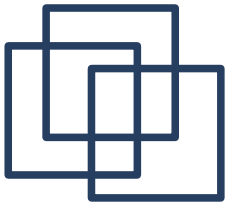
Tunelowanie, kapsułkowanie, XDR

1. Transmisja tunelowa i kapsułkowanie

- serwery proxy.

2. Zewnętrzna reprezentacja danych

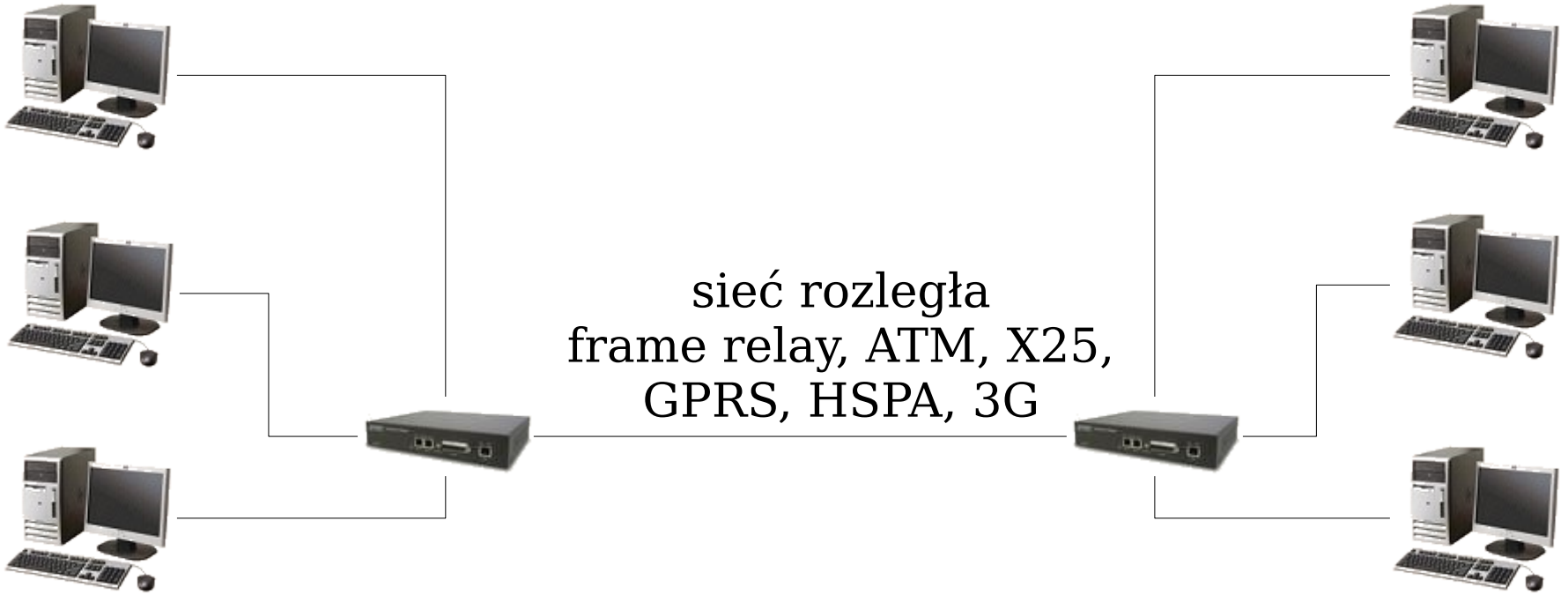
- XDR.



Transmisja tunelowa i kapsułkowanie

Sieci komputerowe rozwijały się stopniowo przez wiele lat. Poszczególni producenci promowali własne systemy sieciowe, a protokoły TCP/IP nie zawsze były dostępne. Co więcej technologie przesyłu danych w sieciach rozległych zwykle wykorzystują odmienne protokoły transmisji niż te, które stosuje się w sieciach lokalnych. W związku z tym często zachodzi potrzeba przekazywania danych jednego protokołu poprzez połączenie obsługiwane przez inny protokół. W tym celu wykorzystuje się tzw. kapsułkowanie *(encapsulation)* lub tunelowanie *(tunnelling)*.

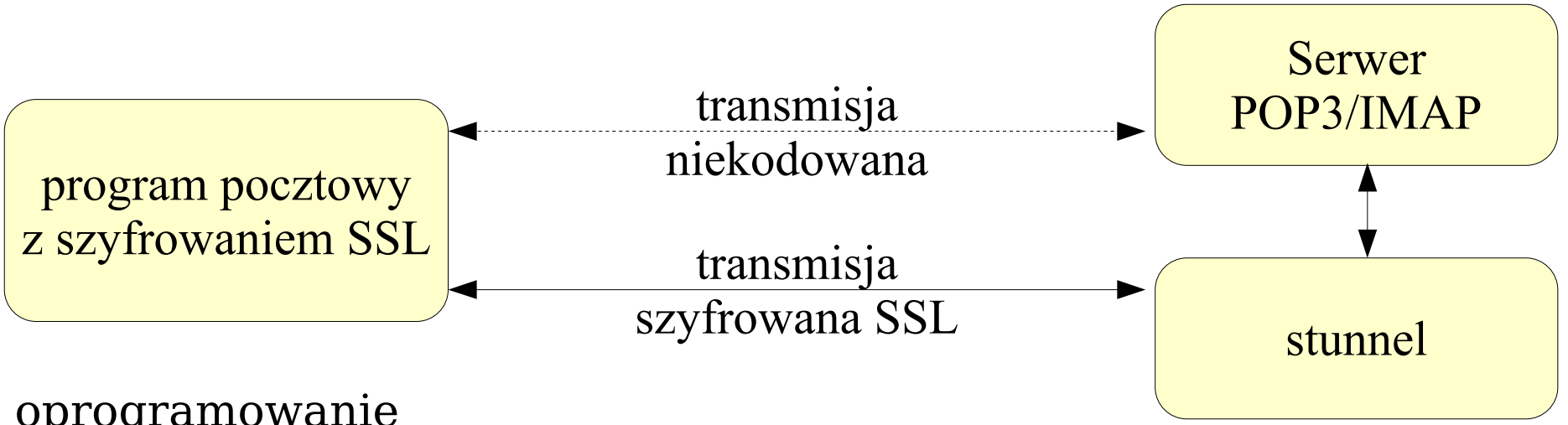
Transmisja tunelowa i kapsułkowanie



Transmisja tunelowa i kapsułkowanie

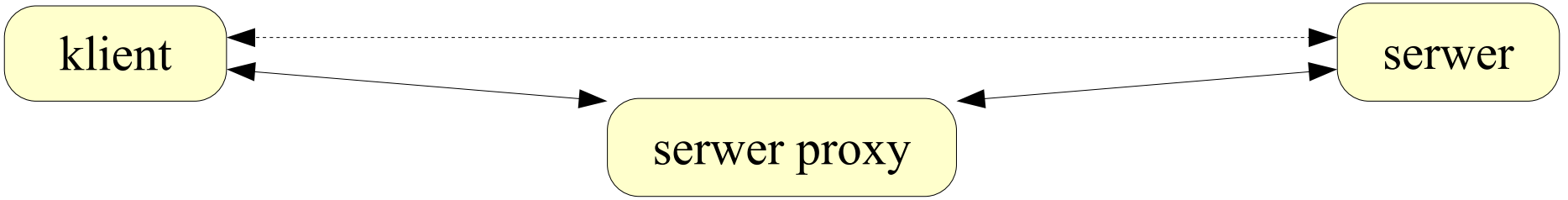
O transmisji kapsułkowanej datagramów IP mówimy wtedy, gdy protokół IP korzysta bezpośrednio z transmisji realizowanych przez sprzęt sieciowy, tzn. każdy datagram przeznaczony do wysłania jest umieszczany (kapsułkowany) w tzw. ramce *(frame)* lub pakiecie sieciowym. O transmisji tunelowej mówimy zaś wtedy, gdy protokół IP przesyła datagramy korzystając z usług protokołu wysokiego poziomu, np. korzysta z warstwy transportowej innego protokołu.

Transmisja tunelowa na poziomie aplikacyjnym



Program stunnel (http://www.stunnel.org) typowo odbiera zgłoszenia na portach 995 (POP3) i 993 (IMAP). Po rozszyfrowaniu odebrane dane są przekazywane do serwera poczty (110, 143). Odpowiedz serwera jest szyfrowana i wysyłana do klienta.

Serwer proxy



Serwer proxy pośredniczy w transmisji pomiędzy klientem a serwerem. Serwerów proxy zwykle używa się aby:

• zabezpieczyć komputery w sieci lokalnej,

• przyspieszyć otrzymywanie odpowiedzi na powtarzające się zapytania (np. strony www – squid),

• umożliwić wielu komputerom korzystanie z jednego publicznego adresu IP *(Network Address Translation).*

Inne zastosowania „serwerów” pośredniczących: VPN, VOIP, GSM/GPRS, ...

Zewnętrzna reprezentacja danych

Dana jest liczba 32 bitowa

0	0	2	8

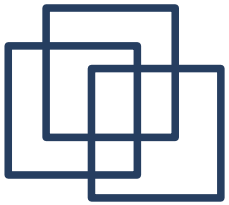
W zależności od „komputera” ten ciąg bajtów może być różnie interpretowany. Np.

$$2 \cdot 2^8 + 8 \cdot 2^0 = 520$$

lub

$$8 \cdot 2^{24} + 2 \cdot 2^{16} = 134217728 + 131072 = 134348800$$

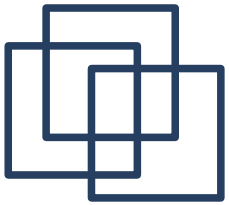
Dla programistów konstruujących oprogramowanie typu klient – serwer wybór sposobu reprezentacji danych przesyłanych między partnerami jest istotnym problemem. Jeśli komputery różnią się reprezentacją przesyłanych danych to muszą one zostać poddane konwersji.



Zewnętrzna reprezentacja danych

Asymetryczna konwersja danych **-** jedna ze stron (klient lub serwer) dokonują odpowiedniej konwersji danych. W tej technice programista musi uwzględnić konwersję dla każdej pary różnych typów architektury komputerów, na których system może być używany.

Jeśli w realizacji programów klient – serwer zakłada się asymetryczną konwersję danych między rodzimą reprezentacją klienta a rodzimą reprezentacją serwera to liczba potrzebnych wersji takiej pary programów wzrasta proporcjonalnie do kwadratu liczby różnych typów architektury (**problem N^2 dla konwersji danych**).



Standardowy sieciowy porządek bajtów

Symetryczna konwersja danych – obie strony wykonują konwersję danych z postaci rodzimej do standardowej reprezentacji danych przesyłanych przez sieć – **zewnętrznej reprezentacji danych** *(external data representation)*.

Zalety:

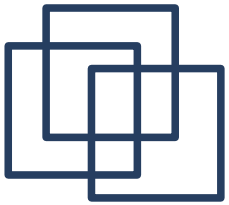
• brak troski o architekturę komputera, na którym działa partner komunikacji,

• łatwiejsza implementacja i konserwacja programu

Wady:

• dodatkowy koszt związany z konwersją danych, w przypadku gdy nie byłaby ona potrzebna,

• przesyłanie dodatkowych informacji przez sieć wymaganych przez specyfikację.



Standardowy sieciowy porządek bajtów

XDR – najpopularniejszy standard sieciowej reprezentacji danych opracowany przez firmę SUN [RFC 1014]. Podstawowy blok danych – 4 bajty.

• **`int`** – [-2147483648, 2147483647] – 4 bajty w formacie „big endian” (bajt najbardziej znaczący pod najniższym adresem):

0	0	2	8

= 520

Liczby ujemne – notacja uzupełnieniowa do dwóch. Zapis bitów w ramach jednego bajta definiowany przez inne standardy. Zwykle „little endian”.

• **`unsigned int`** – [0, 4294967295] – 4 bajty, “big endian”.

• **`enum`** – równoważnie jak zbiór `int`'ów,

• **`bool`** – równoważne do **`enum { FALSE = 0, TRUE = 1 }`**.

Standardowy sieciowy porządek bajtów

- **`hyper`** (**`unsigned hyper`**) – 8 bajtowy **`int`** (**`unsigned int`**), format „big endian”.
- **`float`** – 4 bajtowa liczba zmiennoprzecinkowa:

1 bajt		2 bajt	3 bajt	4 bajt
S	E (8 bitów)	F (23 bity)		

$$\text{liczba} = (-1)^S \cdot 1.F \cdot 2^{E-B}. \qquad (B=127)$$

Przykład:

$$(5.78125)_{10} = (4 + 1 + 1/2 + 1/4 + 1/32)_{10} = (101.11001)_2 = (-1)^0 \cdot$$

$$(1.0111001)_2 \cdot 2^2 =$$

$$= 0\ 10000001\ 01110010000000000000000$$

Pierwszy bit najbardziej znaczący. Stałe typu **`NaN`** (*not a number*) nie powinny być przesyłane przez sieć,

Standardowy sieciowy porządek bajtów

- **`double`** – 8 bajtowa liczba zmiennoprzecinkowa: E – 11 bitów, F – 52 bity, B = 1023,

1 bajt		2 bajt	3 bajt	4 bajt	5 bajt	6 bajt	7 bajt	8 bajt
S	E (11 bitów)	F (52 bity)						

$$\text{liczba} = (-1)^{S} \cdot 1.F \cdot 2^{E-B}. \qquad (B=1023)$$

- **`void`** – 0 bajtów,

Standardowy sieciowy porządek bajtów

- **`opaque identyfikator[n]`** – dane nie poddane konwersji (n bajtów). Ewentualnie uzupełniane na końcu zerowymi bajtami (tak aby długość ciągu n+r była wielokrotnością 4).

bajt 0	bajt 1	...	bajt n-1	...	bajt n+r-1

- **`opaque identyfikator<n>`** - ciąg danych nie poddanych konwersji o nieustalonej długości (maksymalnie n = 2^{32}-1 bajtów). Długość kodowana jako **`unsigned int`**,

długość (4 bajty)	kolejne bajty uzupełnione zerami na końcu (jak w opaque)

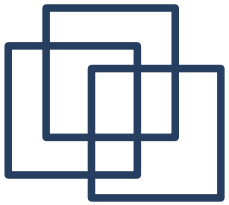
Standardowy sieciowy porządek bajtów

- **`string identyfikator<n>`** - tekst kodowany jest podobnie do opaque o nieustalonej długości. Znaki tekstu są kolejno umieszczane w obszarze danych,
- **`tablica`** o ustalonej długości – kolejno zapisane elementy tablicy (każdy element może mieć inną długość – **`string`**),

element 0	element 1	...	element n-1

- **`tablica`** o nieustalonej długości (max n = 2^{32}-1 kodowane jako **`unsigned int`**)

n	element 0	element 1	...	element n-1



Standardowy sieciowy porządek bajtów

- **struktury – struct {**

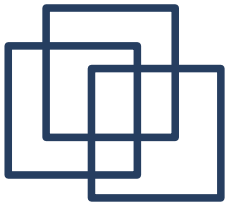
```
   komponent A;
   komponent B;
   ...
};
```

komponent A	komponent B	...

- **unie – union{**

```
   komponent A;
   komponent B;
};
```

numer komponentu (4 bajty)	komponent



Procedury konwersji XDR

Do konwersji można wykorzystać funkcje biblioteczne **<rpc/xdr.h>**

Najpierw trzeba utworzyć obiekt XDR. W przypadku konwersji strumieniowej należy użyć funkcji:

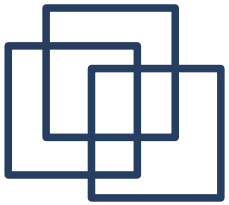
```
void xdrmem_create(XDR *xdrs, char *buf, int blen, int op);
```

xdrs – wskaźnik do utworzonej struktury zarządzającej konwersją strumieniową,

buf – wskaźnik do bufora dla konwertowanego strumienia danych,

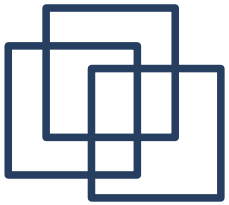
blen – rozmiar bufora,

op – tryb pracy konwertera: **XDR_ENCODE**, **XDR_DECODE** lub **XDR_FREE**.



Procedury konwersji XDR

Poszczególne procedury konwersji XDR mogą wykonywać konwersję w obydwu kierunkach. Wywołana procedura stwierdza w którym kierunku należy przeprowadzić konwersję, badając odpowiednią informację w strumieniu XDR, na którym działa. Jeśli **`*xdrs`** został utworzony z parametrem **`XDR_DECODE`**, odczytana zmienna zostanie wpisana w komórkę wskazaną przez **`pi`**.



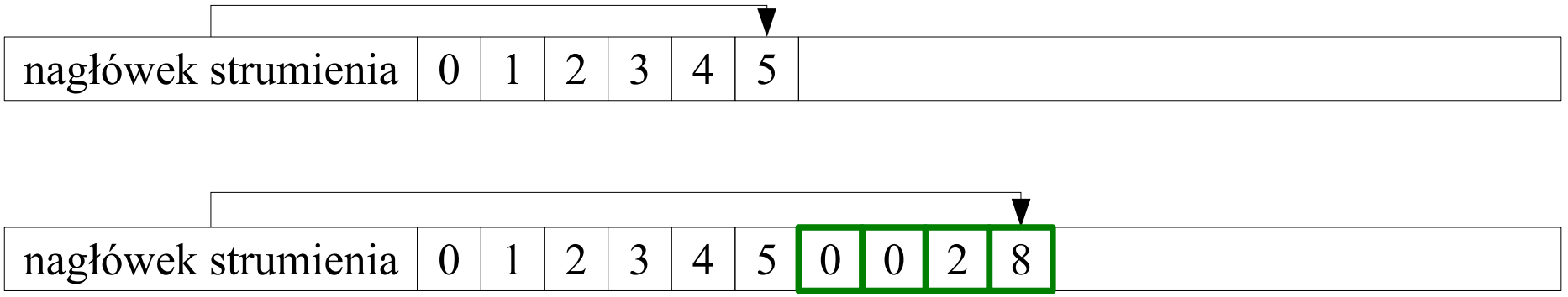
Procedury konwersji XDR

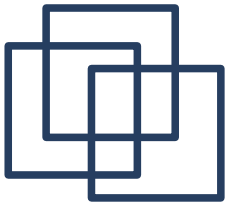
Jeśli (**`*xdrs`**) został utworzony w trybie **`XDR_ENCODE`** do konwersji liczby należy użyć jednej z funkcji konwertujących np:

```
int xdr_int(XDR *xdrs, int *pi);
```

`xdrs` – wskaźnik do struktury zarządzającej konwersją strumieniową,

`pi` – wskaźnik do konwertowanej liczby;





Procedury konwersji XDR

Program może zażądać, aby procedury XDR po przekształceniu każdego elementu danych do postaci zewnętrznej automatycznie wysyłały ten element przez połączenie TCP. W tym celu należy użyć funkcji **`fdopen()`** i **`xdrstdio_create()`**:

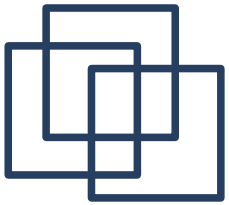
```
FILE * fdopen(int fd, char* mode);
```

`fd` – deskryptor istniejącego pliku (gniazda),

`mode` – typ dostępu do pliku (gniazda): **`r, r+, w, w+, a, a+`**

Wartość zwracana: wskaźnik do strumienia związanego z plikiem (gniazdem) lub **`NULL`** w przypadku błędu (**`errno`**).

Funkcja **`fdopen()`** pozwala związać istniejące gniazdo TCP ze strumieniem **`FILE`**.



Procedury konwersji XDR

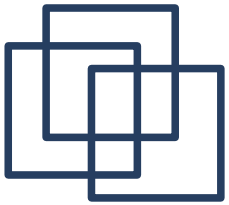
Do powiązania konwertera XDR z dowolnym strumieniem danych służy funkcja **xdrstdio_create().**

```
void xdrstdio_create(XDR *xdrs, FILE *file, int op);
```

xdrs - wskaźnik do utworzonej struktury zarządzającej konwersją strumieniową,

FILE – wskaźnik do struktury opisującej strumień danych,

op – tryb pracy konwertera: **XDR_ENCODE**, **XDR_DECODE** lub **XDR_FREE**.



Procedury konwersji XDR

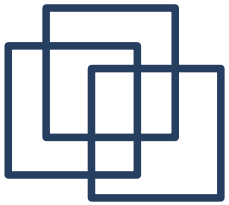
Aby funkcje XDR mogły współpracować z oprogramowaniem używającym transmisji datagramowej opracowano alternatywny interfejs.

```
void xdrrec_create(XDR *xdrs, u_int sendsize,
   u_int recvsize, char *handle, int (*readit)(),
   int (*writeit)();
```

xdrs - wskaźnik do utworzonej struktury zarządzającej konwersją strumieniową,

sendsize, recvsize – rozmiary buforów: wyjściowego i wejściowego,

readit, writeit – funkcje wywoływane po opróżnieniu / zapełnieniu bufora w celu pobrania/wysłania danych. Posiadają trzy argumenty: **void *handle, void *buffer, void *len**.



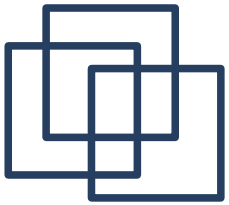
Procedury konwersji XDR

Użycie rekordów umożliwia wykorzystywanie dodatkowych funkcji:

`bool_t xdrrec_endofrecord(XDR *xdrs, bool_t flushnow)` - wstawia znacznik końca rekordu

`bool_t xdrrec_skiprecord(XDR *xdrs)` - pozwala pominąć rekord

`bool_t xdrrec_eof(XDR *xdrs)` - zwraca **`true`** jeśli osiągnięto koniec pliku w powiązanym ze strukturą **`*xdrs`** strumieniu.



Podsumowanie

Poza klasycznymi zastosowaniami, model klient serwer może być wykorzystany także do innych celów. Przedstawione przykłady odnoszą się do przekazywania i konwersji pakietów. We wszystkich zastosowaniach należy pamiętać o konwersji danych pomiędzy różnymi architekturami sprzętowymi klienta i serwera. Najpowszechniej używany jest tu standard XDR zakładający symetryczną konwersję danych.