Topologia do Laboratório de Frame Relay

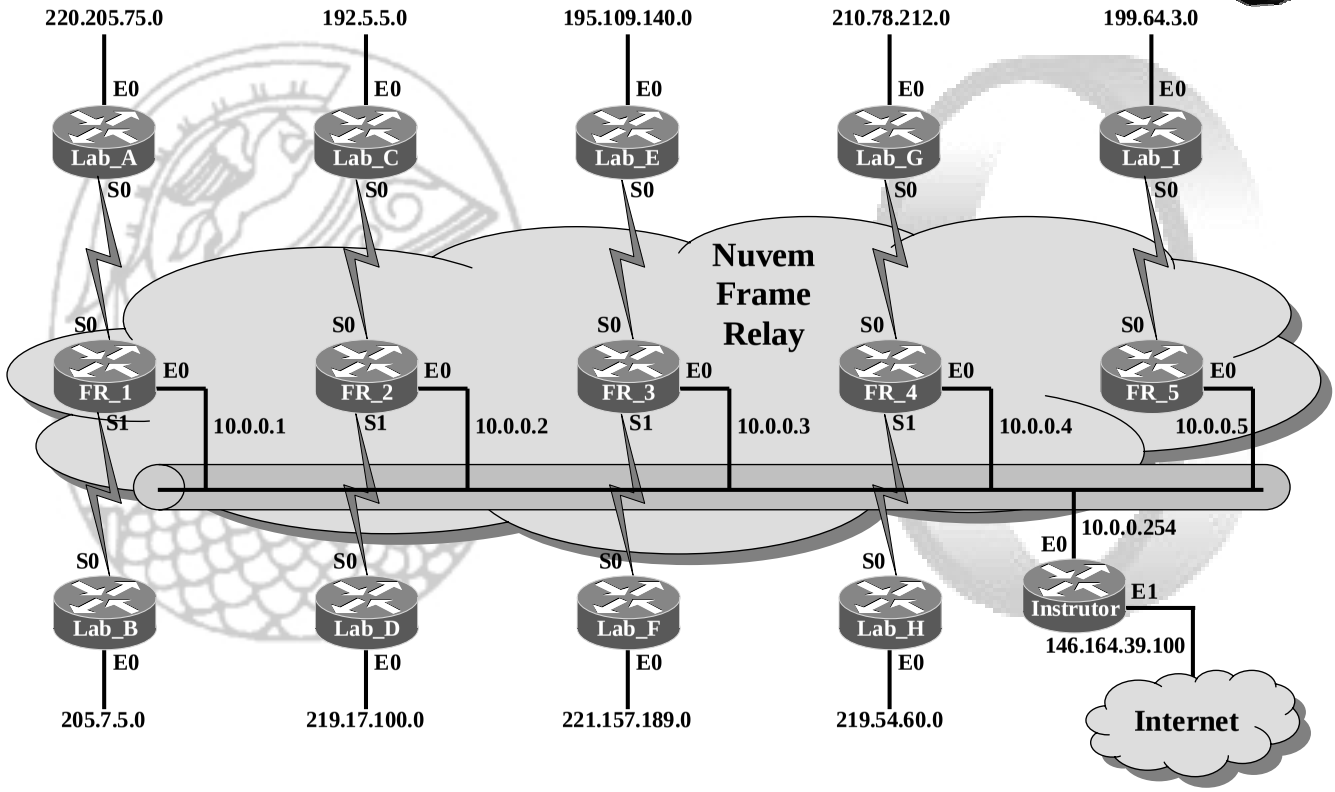


Tabela de DLCIs

Lab_A

Origem	Destino	DLCI
Lab_A	---	610
Lab_B	Lab_B	620
Lab_C	Lab_C	630
Lab_D	Lab_D	640
Lab_E	Lab_E	650
Lab_F	Lab_F	---
Lab_G	Lab_G	670
Lab_H	Lab_H	680
Lab_I	Lab_I	690

Lab_B

Origem	Destino	DLCI
Lab_B	Lab_A	710
Lab_B	Lab_B	720
Lab_C	Lab_C	730
Lab_D	Lab_D	740
Lab_E	Lab_E	750
Lab_F	Lab_F	760
Lab_G	Lab_G	---
Lab_H	Lab_H	780
Lab_I	Lab_I	790

Lab_C

Origem	Destino	DLCI
Lab_C	Lab_A	810
Lab_C	Lab_B	820
Lab_C	Lab_C	830
Lab_D	Lab_D	840
Lab_E	Lab_E	850
Lab_F	Lab_F	860
Lab_G	Lab_G	870
Lab_H	Lab_H	---
Lab_I	Lab_I	890

Lab_D

Origem	Destino	DLCI
Lab_D	Lab_A	910
Lab_D	Lab_B	920
Lab_D	Lab_C	930
Lab_D	Lab_D	940
Lab_D	Lab_E	950
Lab_D	Lab_F	960
Lab_D	Lab_G	970
Lab_D	Lab_H	980
Lab_D	Lab_I	---

Lab_E

Origem	Destino	DLCI
Lab_E	Lab_A	410
Lab_E	Lab_B	420
Lab_E	Lab_C	430
Lab_E	Lab_D	---
Lab_E	Lab_E	450
Lab_E	Lab_F	460
Lab_E	Lab_G	470
Lab_E	Lab_H	480
Lab_E	Lab_I	490

Lab_F

Origem	Destino	DLCI
Lab_F	Lab_A	510
Lab_F	Lab_B	520
Lab_F	Lab_C	530
Lab_F	Lab_D	540
Lab_F	Lab_E	---
Lab_F	Lab_F	560
Lab_F	Lab_G	570
Lab_F	Lab_H	580
Lab_F	Lab_I	590

Lab_G

Origem	Destino	DLCI
Lab_G	Lab_A	210
Lab_G	Lab_B	---
Lab_G	Lab_C	230
Lab_G	Lab_D	240
Lab_G	Lab_E	250
Lab_G	Lab_F	260
Lab_G	Lab_G	270
Lab_G	Lab_H	280
Lab_G	Lab_I	290

Lab_H

Origem	Destino	DLCI
Lab_H	Lab_A	310
Lab_H	Lab_B	320
Lab_H	Lab_C	---
Lab_H	Lab_D	340
Lab_H	Lab_E	350
Lab_H	Lab_F	360
Lab_H	Lab_G	370
Lab_H	Lab_H	380
Lab_H	Lab_I	390

Lab_I

Origem	Destino	DLCI
Lab_I	Lab_A	110
Lab_I	Lab_B	120
Lab_I	Lab_C	130
Lab_I	Lab_D	140
Lab_I	Lab_E	150
Lab_I	Lab_F	160
Lab_I	Lab_G	170
Lab_I	Lab_H	180
Lab_I	Lab_I	190

Configuração dos Roteadores da Nuvem Frame Relay

1. FR_1

```
hostname FR_1
enable secret cisco
enable password cisco1

frame-relay switching

line con 0
exec-timeout 0 0
logging synchronous
password cisco
login

line aux 0
exec-timeout 0 0
logging synchronous
password cisco
login

line vty 0 4
exec-timeout 0 0
password cisco
login

interface e 0
description tunnel com frame-relay encapsulado em IP
ip add 10.0.0.1 255.255.255.0
no shutdown

interface tunnel 12
description tunnel entre FR_1 e FR_2
tunnel source e0
tunnel destination 10.0.0.2

interface tunnel 13
description tunnel entre FR_1 e FR_3
tunnel source e0
tunnel destination 10.0.0.3

interface tunnel 14
description tunnel entre FR_1 e FR_4
tunnel source e0
tunnel destination 10.0.0.4

interface tunnel 15
description tunnel entre FR_1 e FR_5
tunnel source e0
tunnel destination 10.0.0.5

interface tunnel 19
description tunnel entre FR_1 e Instrutor
tunnel source e0
tunnel destination 10.0.0.254
```

```
interface s 0
description ligado ao roteador Lab_A
no ip address
encapsulation frame-relay
clock rate 56000
frame-relay intf-type dce

! Lab_A (120) ==> Lab_B (210)
frame-relay route 120 interface s1 210
! Lab_A (130) ==> tunnel12 (135) ==> Lab_C (310)
frame-relay route 130 interface tunnel12 135
! Lab_A (140) ==> tunnel12 (145) ==> Lab_D (410)
frame-relay route 140 interface tunnel12 145
! Lab_A (150) ==> tunnel13 (155) ==> Lab_E (510)
frame-relay route 150 interface tunnel13 155
! Lab_A (160) ==> tunnel13 (165) ==> Lab_F (610)
frame-relay route 160 interface tunnel13 165
! Lab_A (170) ==> tunnel14 (175) ==> Lab_G (710)
frame-relay route 170 interface tunnel14 175
! Lab_A (180) ==> tunnel14 (185) ==> Lab_H (810)
frame-relay route 180 interface tunnel14 185
! Lab_A (190) ==> tunnel15 (195) ==> Lab_I (910)
frame-relay route 190 interface tunnel15 195
no shutdown

interface s 1
description ligado ao roteador Lab_B
no ip address
encapsulation frame-relay
clock rate 56000
frame-relay intf-type dce

! Lab_B (210) ==> Lab_A (120)
frame-relay route 210 interface s0 120
! Lab_B (230) ==> tunnel12 (235) ==> Lab_C (320)
frame-relay route 230 interface tunnel12 235
! Lab_B (240) ==> tunnel12 (245) ==> Lab_D (420)
frame-relay route 240 interface tunnel12 245
! Lab_B (250) ==> tunnel13 (255) ==> Lab_E (520)
frame-relay route 250 interface tunnel13 255
! Lab_B (260) ==> tunnel13 (265) ==> Lab_F (620)
frame-relay route 260 interface tunnel13 265
! Lab_B (270) ==> tunnel14 (275) ==> Lab_G (720)
frame-relay route 270 interface tunnel14 275
! Lab_B (280) ==> tunnel14 (285) ==> Lab_H (820)
frame-relay route 280 interface tunnel14 285
! Lab_B (290) ==> tunnel15 (295) ==> Lab_I (920)
frame-relay route 290 interface tunnel15 295
no shutdown
```


2. FR_2

```
hostname FR_2
enable secret cisco
enable password cisco1

frame-relay switching

line con 0
exec-timeout 0 0
logging synchronous
password cisco
login

line aux 0
exec-timeout 0 0
logging synchronous
password cisco
login

line vty 0 4
exec-timeout 0 0
password cisco
login

interface e 0
description tunnel com frame-relay encapsulado em IP
ip add 10.0.0.2 255.255.255.0
no shutdown

interface tunnel 21
description tunnel entre FR_2 e FR_1
tunnel source e0
tunnel destination 10.0.0.1

interface tunnel 23
description tunnel entre FR_2 e FR_3
tunnel source e0
tunnel destination 10.0.0.3

interface tunnel 24
description tunnel entre FR_2 e FR_4
tunnel source e0
tunnel destination 10.0.0.4

interface tunnel 25
description tunnel entre FR_2 e FR_5
tunnel source e0
tunnel destination 10.0.0.5

interface tunnel 29
description tunnel entre FR_2 e Instrutor
tunnel source e0
tunnel destination 10.0.0.254
```

```
interface s 0
description ligado ao roteador Lab_C
no ip address
encapsulation frame-relay
clock rate 56000
frame-relay intf-type dce

! Lab_C (310) ==> tunnel21 (135) ==> Lab_A (130)
frame-relay route 310 interface tunnel21 135
! Lab_C (320) ==> tunnel21 (235) ==> Lab_B (230)
frame-relay route 320 interface tunnel21 235
! Lab_C (340) ==> Lab_D (430)
frame-relay route 340 interface s1 430
! Lab_C (350) ==> tunnel23 (355) ==> Lab_E (530)
frame-relay route 350 interface tunnel23 355
! Lab_C (360) ==> tunnel23 (365) ==> Lab_F (630)
frame-relay route 360 interface tunnel23 365
! Lab_C (370) ==> tunnel24 (375) ==> Lab_G (730)
frame-relay route 370 interface tunnel24 375
! Lab_C (380) ==> tunnel24 (385) ==> Lab_H (830)
frame-relay route 380 interface tunnel24 385
! Lab_C (390) ==> tunnel25 (395) ==> Lab_I (930)
frame-relay route 390 interface tunnel25 395
no shutdown

interface s 1
description ligado ao roteador Lab_D
no ip address
encapsulation frame-relay
clock rate 56000
frame-relay intf-type dce

! Lab_D (410) ==> tunnel21 (145) ==> Lab_A (140)
frame-relay route 410 interface tunnel21 145
! Lab_D (420) ==> tunnel21 (245) ==> Lab_B (240)
frame-relay route 420 interface tunnel21 245
! Lab_D (430) ==> Lab_C (340)
frame-relay route 430 interface s0 340
! Lab_D (450) ==> tunnel23 (455) ==> Lab_E (540)
frame-relay route 450 interface tunnel23 455
! Lab_D (460) ==> tunnel23 (465) ==> Lab_F (640)
frame-relay route 460 interface tunnel23 465
! Lab_D (470) ==> tunnel24 (475) ==> Lab_G (740)
frame-relay route 470 interface tunnel24 475
! Lab_D (480) ==> tunnel24 (485) ==> Lab_H (840)
frame-relay route 480 interface tunnel24 485
! Lab_D (490) ==> tunnel25 (495) ==> Lab_I (940)
frame-relay route 490 interface tunnel25 495
no shutdown
```

3. FR_3

```
hostname FR_3
enable secret cisco
enable password cisco1

frame-relay switching

line con 0
exec-timeout 0 0
logging synchronous
password cisco
login

line aux 0
exec-timeout 0 0
logging synchronous
password cisco
login

line vty 0 4
exec-timeout 0 0
password cisco
login

interface e 0
description tunnel com frame-relay encapsulado em IP
ip add 10.0.0.3 255.255.255.0
no shutdown

interface tunnel 31
description tunnel entre FR_3 e FR_1
tunnel source e0
tunnel destination 10.0.0.1

interface tunnel 32
description tunnel entre FR_3 e FR_2
tunnel source e0
tunnel destination 10.0.0.2

interface tunnel 34
description tunnel entre FR_3 e FR_4
tunnel source e0
tunnel destination 10.0.0.4

interface tunnel 35
description tunnel entre FR_3 e FR_5
tunnel source e0
tunnel destination 10.0.0.5

interface tunnel 39
description tunnel entre FR_3 e Instrutor
tunnel source e0
tunnel destination 10.0.0.254

interface s 0
description ligado ao roteador Lab_E
no ip address
encapsulation frame-relay
clock rate 56000
frame-relay intf-type dce

! Lab_E (510) ==> tunnel31 (155) ==> Lab_A (150)
frame-relay route 510 interface tunnel31 155
! Lab_E (520) ==> tunnel31 (255) ==> Lab_B (250)
frame-relay route 520 interface tunnel31 255
! Lab_E (530) ==> tunnel32 (355) ==> Lab_C (350)
frame-relay route 530 interface tunnel32 355
! Lab_E (540) ==> tunnel32 (455) ==> Lab_D (450)
frame-relay route 540 interface tunnel32 455
! Lab_E (560) ==> Lab_F (650)
frame-relay route 560 interface s1 650
! Lab_E (570) ==> tunnel34 (575) ==> Lab_G (750)
frame-relay route 570 interface tunnel34 575
! Lab_E (580) ==> tunnel34 (585) ==> Lab_H (850)
frame-relay route 580 interface tunnel34 585
! Lab_E (590) ==> tunnel35 (595) ==> Lab_I (950)
frame-relay route 590 interface tunnel35 595
no shutdown

interface s 1
description ligado ao roteador Lab_F
no ip address
encapsulation frame-relay
clock rate 56000
frame-relay intf-type dce

! Lab_F (610) ==> tunnel31 (165) ==> Lab_A (160)
frame-relay route 610 interface tunnel31 165
! Lab_F (620) ==> tunnel31 (265) ==> Lab_B (260)
frame-relay route 620 interface tunnel31 265
! Lab_F (630) ==> tunnel32 (365) ==> Lab_C (360)
frame-relay route 630 interface tunnel32 365
! Lab_F (640) ==> tunnel32 (465) ==> Lab_D (460)
frame-relay route 640 interface tunnel32 465
! Lab_F (650) ==> Lab_E (560)
frame-relay route 650 interface s0 560
! Lab_F (670) ==> tunnel34 (675) ==> Lab_G (760)
frame-relay route 670 interface tunnel34 675
! Lab_F (680) ==> tunnel34 (685) ==> Lab_H (860)
frame-relay route 680 interface tunnel34 685
! Lab_F (690) ==> tunnel35 (695) ==> Lab_I (960)
frame-relay route 690 interface tunnel35 695
no shutdown
```


4. FR_4

```
hostname FR_4
enable secret cisco
enable password cisco1

frame-relay switching

line con 0
exec-timeout 0 0
logging synchronous
password cisco
login

line aux 0
exec-timeout 0 0
logging synchronous
password cisco
login

line vty 0 4
exec-timeout 0 0
password cisco
login

interface e 0
description tunnel com frame-relay encapsulado em IP
ip add 10.0.0.4 255.255.255.0
no shutdown

interface tunnel 41
description tunnel entre FR_4 e FR_1
tunnel source e0
tunnel destination 10.0.0.1

interface tunnel 42
description tunnel entre FR_4 e FR_2
tunnel source e0
tunnel destination 10.0.0.2

interface tunnel 43
description tunnel entre FR_4 e FR_3
tunnel source e0
tunnel destination 10.0.0.3

interface tunnel 45
description tunnel entre FR_4 e FR_5
tunnel source e0
tunnel destination 10.0.0.5

interface tunnel 49
description tunnel entre FR_4 e Instrutor
tunnel source e0
tunnel destination 10.0.0.254

interface s 0
description ligado ao roteador Lab_G
no ip address
encapsulation frame-relay
clock rate 56000
frame-relay intf-type dce

! Lab_G (710) ==> tunnel41 (175) ==> Lab_A (170)
frame-relay route 710 interface tunnel41 175
! Lab_G (720) ==> tunnel41 (275) ==> Lab_B (270)
frame-relay route 720 interface tunnel41 275
! Lab_G (730) ==> tunnel42 (375) ==> Lab_C (370)
frame-relay route 730 interface tunnel42 375
! Lab_G (740) ==> tunnel42 (475) ==> Lab_D (470)
frame-relay route 740 interface tunnel42 475
! Lab_G (750) ==> tunnel43 (575) ==> Lab_E (570)
frame-relay route 750 interface tunnel43 575
! Lab_G (760) ==> tunnel43 (675) ==> Lab_F (670)
frame-relay route 760 interface tunnel43 675
! Lab_G (780) ==> Lab_H (870)
frame-relay route 780 interface s1 870
! Lab_G (790) ==> tunnel45 (795) ==> Lab_I (970)
frame-relay route 790 interface tunnel45 795
no shutdown

interface s 1
description ligado ao roteador Lab_H
no ip address
encapsulation frame-relay
clock rate 56000
frame-relay intf-type dce

! Lab_H (810) ==> tunnel41 (185) ==> Lab_A (180)
frame-relay route 810 interface tunnel41 185
! Lab_H (820) ==> tunnel41 (285) ==> Lab_B (280)
frame-relay route 820 interface tunnel41 285
! Lab_H (830) ==> tunnel42 (385) ==> Lab_C (380)
frame-relay route 830 interface tunnel42 385
! Lab_H (840) ==> tunnel42 (485) ==> Lab_D (480)
frame-relay route 840 interface tunnel42 485
! Lab_H (850) ==> tunnel43 (585) ==> Lab_E (580)
frame-relay route 850 interface tunnel43 585
! Lab_H (860) ==> tunnel43 (685) ==> Lab_F (680)
frame-relay route 860 interface tunnel43 685
! Lab_H (870) ==> Lab_G (780)
frame-relay route 870 interface s0 780
! Lab_H (890) ==> tunnel45 (895) ==> Lab_I (980)
frame-relay route 890 interface tunnel45 895
no shutdown
```

5. FR_5

```
hostname FR_5
enable secret cisco
enable password cisco1

frame-relay switching

line con 0
exec-timeout 0 0
logging synchronous
password cisco
login

line aux 0
exec-timeout 0 0
logging synchronous
password cisco
login

line vty 0 4
exec-timeout 0 0
password cisco
login

interface e 0
description tunnel com frame-relay encapsulado em IP
ip add 10.0.0.5 255.255.255.0
no shutdown

interface tunnel 51
description tunnel entre FR_5 e FR_1
tunnel source e0
tunnel destination 10.0.0.1

interface tunnel 52
description tunnel entre FR_5 e FR_2
tunnel source e0
tunnel destination 10.0.0.2

interface tunnel 53
description tunnel entre FR_5 e FR_3
tunnel source e0
tunnel destination 10.0.0.3

interface tunnel 54
description tunnel entre FR_5 e FR_4
tunnel source e0
tunnel destination 10.0.0.4

interface tunnel 59
description tunnel entre FR_5 e Instrutor
tunnel source e0
tunnel destination 10.0.0.254

interface s 0
description ligado ao roteador Lab_I
no ip address
encapsulation frame-relay
clock rate 56000
frame-relay intf-type dce

! Lab_I (910) ==> tunnel51 (195) ==> Lab_A (190)
frame-relay route 910 interface tunnel51 195
! Lab_I (920) ==> tunnel51 (295) ==> Lab_B (290)
frame-relay route 920 interface tunnel51 295
! Lab_I (930) ==> tunnel52 (395) ==> Lab_C (390)
frame-relay route 930 interface tunnel52 395
! Lab_I (940) ==> tunnel52 (495) ==> Lab_D (490)
frame-relay route 940 interface tunnel52 495
! Lab_I (950) ==> tunnel53 (595) ==> Lab_E (590)
frame-relay route 950 interface tunnel53 595
! Lab_I (960) ==> tunnel53 (695) ==> Lab_F (690)
frame-relay route 960 interface tunnel53 695
! Lab_I (970) ==> tunnel54 (795) ==> Lab_G (790)
frame-relay route 970 interface tunnel54 795
! Lab_I (980) ==> tunnel54 (895) ==> Lab_H (890)
frame-relay route 980 interface tunnel54 895
no shutdown

interface s 1
description nao esta ligado
no ip address
shutdown
```


6. FR_Instrutor

```
hostname FR_Instrutor
enable secret cisco
enable password cisco1

line con 0
exec-timeout 0 0
logging synchronous
password cisco
login

line aux 0
exec-timeout 0 0
logging synchronous
password cisco
login

line vty 0 4
exec-timeout 0 0
password cisco
login

interface e 0
description ligado com os roteadores frame-relay 1 a 5
ip add 10.0.0.254 255.255.255.0
no shutdown

interface e 1
description ligado com a rede NCE
ip add 146.164.39.100 255.255.255.0
no shutdown

interface s 0
description nao ligado
no ip address
shutdown

interface s 1
description nao ligado
no ip address
shutdown
```

Frame Relay

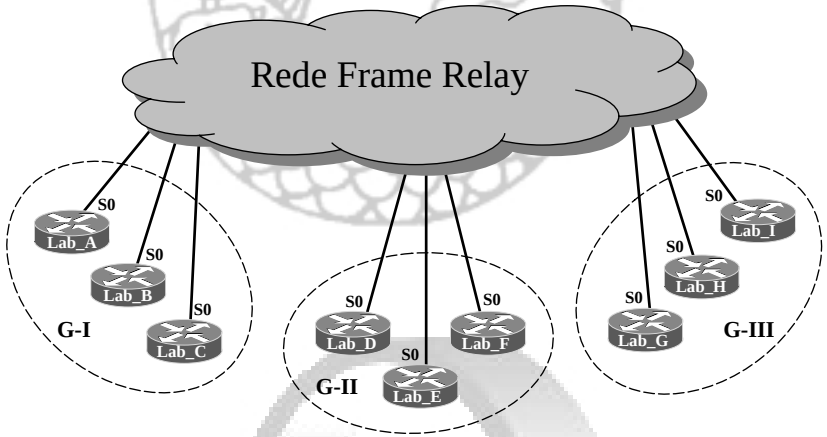
CISCO ACADEMY - NCE/UFRJ

Objetivos

Familiarizar com os procedimentos para efetuar a configuração para ligar roteadores Cisco em uma rede Frame Relay.

Montagem Inicial

Os 9 roteadores serão divididos em três grupos de três roteadores cada. Cada roteador utilizará apenas as interfaces Ethernet 0 e a Serial 0 e se ligarão à rede Frame Relay (nuvem) do laboratório Cisco conforme o esquema abaixo.



Cada roteador se comunicará, inicialmente, apenas com os roteadores do seu grupo, e os DLCIs a serem usados são os mesmos já divulgados.

Grupo	Roteadores
I	Lab_A, Lab_B e Lab_C
II	Lab_D, Lab_E e Lab_F
III	Lab_G, Lab_H e Lab_I

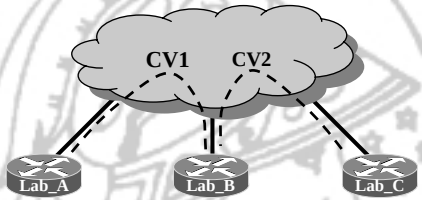
Configuração Inicial

Os endereços IP da interface Ethernet de cada roteador são os mesmos utilizados na topologia do laboratório Cisco original:

Roteador	IP da E0	Roteador	IP da E0	Roteador	IP da E0
Lab_A	220.205.75.0	Lab_D	219.17.100.0	Lab_G	210.78.212.0
Lab_B	205.7.5.0	Lab_E	195.109.140.0	Lab_H	219.54.60.0
Lab_C	192.5.5.0	Lab_F	221.157.189.0	Lab_I	199.64.3.0

Exercício 1 – Conexões Ponto-a-Ponto Hub-and-Spoke

Cada grupo será ligado conforme a figura abaixo e apenas dois circuitos virtuais foram contratados na Concessionária de Serviços: um que liga o primeiro roteador ao segundo (CV1) e outro que liga o segundo roteador ao terceiro (CV2).



Configure os roteadores utilizando os seguintes endereços IP para cada circuito virtual contratado:

Grupo	CV1	IP	CV2	IP
I	Lab_A – Lab_B	200.100.100.0	Lab_B – Lab_C	200.100.200.0
II	Lab_D – Lab_E	200.150.100.0	Lab_E – Lab_F	200.150.200.0
III	Lab_G – Lab_H	200.200.100.0	Lab_H – Lab_I	200.200.200.0

Como o roteador central tem 2 circuitos virtuais, será necessário usar subinterface para configurar a interface S0. Na configuração dos outros roteadores pode, ou não, ser usada a subinterface.

1. ***Programações Usando Interface na Serial 0 de Lab_A e Lab_C***

   a. ***Programação de Lab_A***

```
Lab_A# config terminal
Lab_A(config)# interface serial 0
Lab_A(config-if)# description CV1 para Lab_B
Lab_A(config-if)# encapsulation frame-relay
Lab_A(config-if)# ip address 200.100.100.1 255.255.255.0
Lab_A(config-if)# frame-relay interface-dlci 120
Lab_A(config-fr-dlci)# end
```

   b. ***Programação de Lab_B***

```
Lab_B# config terminal
Lab_B(config)# interface serial 0
Lab_B(config-if)# encapsulation frame-relay
Lab_B(config-if)# no ip address
Lab_B(config-if)# interface serial 0.1 point-to-point
Lab_B(config-subif)# description CV1 para Lab_A
Lab_B(config-subif)# ip address 200.100.100.2 255.255.255.0
Lab_B(config-subif)# frame-relay interface-dlci 210
Lab_B(config-fr-dlci)# interface serial 0.2 point-to-point
Lab_B(config-subif)# description CV2 para Lab_C
Lab_B(config-subif)# ip address 200.100.200.1 255.255.255.0
Lab_B(config-subif)# frame-relay interface-dlci 230
Lab_B(config-fr-dlci)# end
```

c. *Programação de Lab_C*

```
Lab_C# config terminal
Lab_C(config)# interface serial 0
Lab_C(config-if)# description CV2 para Lab_B
Lab_C(config-if)# encapsulation frame-relay
Lab_C(config-if)# ip address 200.100.200.2 255.255.255.0
Lab_C(config-if)# frame-relay interface-dlci 320
Lab_C(config-fr-dlci)# end
```

d. *Verificando os Circuitos Virtuais*

Verifique o mapeamento IP x DLCI entrando com o comando ***show frame-relay map***.

```
Lab_A# sh frame-relay map
Serial0 (up): ip 200.100.100.2 dlci 120(0x78,0x1C80), dynamic,
              broadcast,, status defined, active
Serial0 (up): ip 200.100.200.2 dlci 130(0x82,0x2020), dynamic,
              broadcast,, status defined, active
```

```
Lab_B# sh frame-relay map
Serial0.1 (up): point-to-point dlci, dlci 210(0xD2,0x3420), broadcast
                status defined, active
Serial0.2 (up): point-to-point dlci, dlci 230(0xE6,0x3860), broadcast
                status defined, active
```

```
Lab_C# sh frame-relay map
Serial0 (up): ip 200.100.100.1 dlci 310(0x136,0x4C60), dynamic,
              broadcast,, status defined, active
Serial0 (up): ip 200.100.200.1 dlci 320(0x140,0x5000), dynamic,
              broadcast,, status defined, active
```

Observe que o PVC entre Lab_A e Lab_C está ativo mesmo não estando definido na programação! Isso ocorre porque os DLCIs 130 (em Lab_A) e 310 (em Lab_C) estão definidos na nuvem Frame Relay e são informados pelo switch através das LMIs. Através do *Inverse ARP* o IP da interface do outro lado é descoberto.

Da mesma forma, como os roteadores dos outros grupos também estão programados, outros DLCIs devem aparecer (isso irá depender do estado em que estão as programações dos outros roteadores).

Dessa forma a configuração dos 3 roteadores (ou eventualmente dos 9 roteadores) seria *full-mesh* (totalmente conectada) e não *hub-and-spoke*. Para que o laboratório funcione como foi definido é necessário excluir da nuvem Frame Relay todos os DLCIs não utilizados pelos 3 grupos.

Quando usamos subinterfaces, como em Lab_B, não se atribui endereço IP à interface. Assim, os DLCIs não utilizados ficam associados à interface (sem endereço IP), de forma que o *Inverse ARP* não terá sucesso nesses DLCIs.

Através do comando ***show frame-relay pvc*** o estado de cada DLCI pode ser verificado. Executando esse comando em Lab_B, observe que os DLCIs 210 e 230 estão ativos e

associados às subinterfaces (Serial0.1 e Serial0.2) e os demais DLCIs estão inativos e associados à interface (Serial0).

```
Lab_B# sh frame-relay pvc
PVC Statistics for interface Serial0 (Frame Relay DTE)

DLCI=210, DLCI USAGE=LOCAL, PVC STATUS=ACTIVE, INTERFACE=Serial0.1
  input pkts 18          output pkts 10         in bytes 1154
  out bytes 1563         dropped pkts 0         in FECN pkts 0
  in BECN pkts 0         out FECN pkts 0        out BECN pkts 0
  in DE pkts 0           out DE pkts 0
  out bcast pkts 10      out bcast bytes 1563
  pvc create time 00:20:31, last time pvc status changed 00:20:22

DLCI=230, DLCI USAGE=LOCAL, PVC STATUS=ACTIVE, INTERFACE=Serial0.2
  input pkts 97          output pkts 7          in bytes 30090
  out bytes 1241         dropped pkts 0         in FECN pkts 0
  in BECN pkts 0         out FECN pkts 0        out BECN pkts 0
  in DE pkts 0           out DE pkts 0
  out bcast pkts 7       out bcast bytes 1241
  pvc create time 00:20:33, last time pvc status changed 00:11:13

DLCI=240, DLCI USAGE=UNUSED, PVC STATUS=ACTIVE, INTERFACE=Serial0
  input pkts 0           output pkts 0          in bytes 0
  out bytes 0            dropped pkts 0         in FECN pkts 0
  in BECN pkts 0         out FECN pkts 0        out BECN pkts 0
  in DE pkts 0           out DE pkts 0
  out bcast pkts 0       out bcast bytes 0      Num Pkts Switched 0
  pvc create time 00:20:34, last time pvc status changed 00:20:24

DLCI=250, DLCI USAGE=UNUSED, PVC STATUS=ACTIVE, INTERFACE=Serial0
  input pkts 0           output pkts 0          in bytes 0
  out bytes 0            dropped pkts 0         in FECN pkts 0
  in BECN pkts 0         out FECN pkts 0        out BECN pkts 0
  in DE pkts 0           out DE pkts 0
  out bcast pkts 0       out bcast bytes 0      Num Pkts Switched 0
  pvc create time 00:20:35, last time pvc status changed 00:20:25

DLCI=260, DLCI USAGE=UNUSED, PVC STATUS=INACTIVE, INTERFACE=Serial0
  input pkts 0           output pkts 0          in bytes 0
  out bytes 0            dropped pkts 0         in FECN pkts 0
  in BECN pkts 0         out FECN pkts 0        out BECN pkts 0
  in DE pkts 0           out DE pkts 0
  out bcast pkts 0       out bcast bytes 0      Num Pkts Switched 0
  pvc create time 00:20:36, last time pvc status changed 00:20:36

DLCI=270, DLCI USAGE=UNUSED, PVC STATUS=ACTIVE, INTERFACE=Serial0
  input pkts 21          output pkts 0          in bytes 630
  out bytes 0            dropped pkts 0         in FECN pkts 0
  in BECN pkts 0         out FECN pkts 0        out BECN pkts 0
  in DE pkts 0           out DE pkts 0
  out bcast pkts 0       out bcast bytes 0      Num Pkts Switched 0
  pvc create time 00:20:37, last time pvc status changed 00:20:27
```

```
DLCI=280, DLCI USAGE=UNUSED, PVC STATUS=ACTIVE, INTERFACE=Serial0
  input pkts 0            output pkts 0           in bytes 0
  out bytes 0             dropped pkts 0          in FECN pkts 0
  in BECN pkts 0          out FECN pkts 0         out BECN pkts 0
  in DE pkts 0            out DE pkts 0
  out bcast pkts 0        out bcast bytes 0       Num Pkts Switched 0
  pvc create time 00:20:38, last time pvc status changed 00:20:28

DLCI=290, DLCI USAGE=UNUSED, PVC STATUS=ACTIVE, INTERFACE=Serial0
  input pkts 23           output pkts 0           in bytes 690
  out bytes 0             dropped pkts 0          in FECN pkts 0
  in BECN pkts 0          out FECN pkts 0         out BECN pkts 0
  in DE pkts 0            out DE pkts 0
  out bcast pkts 0        out bcast bytes 0       Num Pkts Switched 0
  pvc create time 00:20:39, last time pvc status changed 00:20:29

Lab_B#
```

Assim, para não alterar a nuvem Frame Relay, é preferível usar a solução através de subinterfaces (item 2, a seguir).

e. Comando Ping na Própria Interface Frame Relay

Outro problema da solução através de interfaces é a impossibilidade de executar o comando **ping para o endereço IP da própria interface serial**, que é uma limitação da Cisco. Esse problema não ocorre quando subinterfaces são usadas.

```
Lab_A# ping 200.100.100.1                    ← ping na própria interface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.1, timeout is 2 seconds:
. . . . .
Success rate is 0 percent (0/5)

Lab_A# ping 200.100.100.2                    ← ping na interface do vizinho
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.2, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 68/68/68 ms
```

```
Lab_C# ping 200.100.200.2                    ← ping na própria interface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.200.2, timeout is 2 seconds:
. . . . .
Success rate is 0 percent (0/5)

Lab_C# ping 200.100.200.1                    ← ping na interface do vizinho
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.200.1, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 44/55/100 ms
```



```
Lab_B# ping 200.100.100.2                    ← ping na própria subinterface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.2, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 128/131/140 ms

Lab_B# ping 200.100.100.1                    ← ping na interface do vizinho
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.1, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 68/73/96 ms

Lab_B# ping 200.100.200.1                    ← ping na própria subinterface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.200.1, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 84/86/96 ms

Lab_B# ping 200.100.200.2                    ← ping na interface do vizinho
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.200.2, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 44/54/96 ms
```

2. Programações Usando Subinterface na Serial 0 de Lab_A e Lab_C

a. Programação de Lab_A

```
Lab_A# config terminal
Lab_A(config)# interface serial 0
Lab_A(config-if)# encapsulation frame-relay
Lab_A(config-if)# no ip address
Lab_A(config-if)# interface serial 0.1 point-to-point
Lab_A(config-subif)# description CV1 para Lab_B
Lab_A(config-subif)# ip address 200.100.100.1 255.255.255.0
Lab_A(config-subif)# frame-relay interface-dlci 120
Lab_A(config-fr-dlci)# end
```

b. Programação de Lab_B

```
Lab_B# config terminal
Lab_B(config)# interface serial 0
Lab_B(config-if)# encapsulation frame-relay
Lab_B(config-if)# no ip address
Lab_B(config-if)# interface serial 0.1 point-to-point
Lab_B(config-subif)# description CV1 para Lab_A
Lab_B(config-subif)# ip address 200.100.100.2 255.255.255.0
Lab_B(config-subif)# frame-relay interface-dlci 210
Lab_B(config-fr-dlci)# interface serial 0.2 point-to-point
Lab_B(config-subif)# description CV2 para Lab_C
Lab_B(config-subif)# ip address 200.100.200.1 255.255.255.0
Lab_B(config-subif)# frame-relay interface-dlci 230
Lab_B(config-fr-dlci)# end
```

c. *Programação de Lab_C*

```
Lab_C# config terminal
Lab_C(config)# interface serial 0
Lab_C(config-if)# encapsulation frame-relay
Lab_C(config-if)# no ip address
Lab_C(config-if)# interface serial 0.1 point-to-point
Lab_C(config-subif)# description CV2 para Lab_B
Lab_C(config-subif)# ip address 200.100.200.2 255.255.255.0
Lab_C(config-subif)# frame-relay interface-dlci 320
Lab_C(config-fr-dlci)# end
```

d. *Verificando os Circuitos Virtuais*

Verifique o mapeamento IP x DLCI entrando com o comando ***show frame-relay map***.

```
Lab_A# sh frame-relay map
Serial0.1 (up): point-to-point dlci, dlci 120(0x78,0x1C80), broadcast
                status defined, active
```

```
Lab_B# sh frame-relay map
Serial0.1 (up): point-to-point dlci, dlci 210(0xD2,0x3420), broadcast
                status defined, active
Serial0.2 (up): point-to-point dlci, dlci 230(0xE6,0x3860), broadcast
                status defined, active
```

```
Lab_C# sh frame-relay map
Serial0.1 (up): point-to-point dlci, dlci 320(0x140,0x5000), broadcast
                status defined, active
```

e. *Tabelas de Roteamento*

Libere o roteamento RIP nos roteadores e verifique se todos os 3 roteadores possuem rotas para as 3 redes ethernet (uma de cada roteador).

```
Lab_A# show ip route
R    200.100.200.0/24 [120/1] via 200.100.100.2, 00:00:10, Serial0.1
R    205.7.5.0/24 [120/1] via 200.100.100.2, 00:00:10, Serial0.1
C    200.100.100.0/24 is directly connected, Serial0.1
R    192.5.5.0/24 [120/2] via 200.100.100.2, 00:00:10, Serial0.1
C    220.205.75.0/24 is directly connected, Ethernet0
```

```
Lab_B# show ip route
C    200.100.200.0/24 is directly connected, Serial0.2
C    205.7.5.0/24 is directly connected, Ethernet0
C    200.100.100.0/24 is directly connected, Serial0.1
R    192.5.5.0/24 [120/1] via 200.100.200.2, 00:00:07, Serial0.2
R    220.205.75.0/24 [120/1] via 200.100.100.1, 00:00:14, Serial0.1
```



```
Lab_C# show ip route
C    200.100.200.0/24 is directly connected, Serial0.1
R    205.7.5.0/24 [120/1] via 200.100.200.1, 00:00:18, Serial0.1
R    200.100.100.0/24 [120/1] via 200.100.200.1, 00:00:18, Serial0.1
C    192.5.5.0/24 is directly connected, Ethernet0
R    220.205.75.0/24 [120/2] via 200.100.200.1, 00:00:18, Serial0.1
```

f. *Split Horizon*

O *Split Horizon* está automaticamente desativado nas interfaces com encapsulamento Frame Relay e nas subinterfaces Frame Relay ponto-a-ponto (apenas em subinterfaces multiponto ele está ativo). No modo EXEC privilegiado, entre com o comando "***debug ip rip***" e verifique as tabelas enviadas por Lab_A e Lab_C para Lab_B.

Devem constar nessas tabelas as redes ethernet dos outros roteadores: 205.7.5.0 e 192.5.5.0 são divulgadas por Lab_A e 220.205.75.0 e 205.7.5.0 são divulgadas por Lab_C. Isso pode ser visto de forma mais facilitada verificando-se o debug apenas no roteador hub (central), isto é, Lab_B.

O debug irá mostrar as rotas exportadas pelas interfaces serial 0 e ethernet 0. Para facilitar a visualização das rotas, coloque a interface ethernet 0 como ***passive-interface*** (não há exportação de rotas por essa interface).

```
Lab_A# debug ip rip
01:42:19: RIP: sending v1 update to 255.255.255.255 via Serial0.1 (200.100.100.1)
01:42:19:      network 200.100.200.0, metric 2
01:42:19:      network 205.7.5.0, metric 2
01:42:19:      network 200.100.100.0, metric 1
01:42:19:      network 192.5.5.0, metric 3
01:42:19:      network 220.205.75.0, metric 1
01:42:47: RIP: received v1 update from 200.100.100.2 on Serial0.1
01:42:47:      200.100.200.0 in 1 hops
01:42:47:      205.7.5.0 in 1 hops
01:42:47:      192.5.5.0 in 2 hops
Lab_A# undebug all
```

```
Lab_C# debug ip rip
01:43:47: RIP: sending v1 update to 255.255.255.255 via Serial0.1 (200.100.200.2)
01:43:47:      network 200.100.200.0, metric 1
01:43:47:      network 205.7.5.0, metric 2
01:43:47:      network 200.100.100.0, metric 2
01:43:47:      network 192.5.5.0, metric 1
01:43:47:      network 220.205.75.0, metric 3
01:43:48: RIP: received v1 update from 200.100.200.1 on Serial0.1
01:43:48:      205.7.5.0 in 1 hops
01:43:48:      200.100.100.0 in 1 hops
01:43:48:      220.205.75.0 in 2 hops
Lab_C# undebug all
```

```
Lab_B# debug ip rip
01:43:19: RIP: sending v1 update to 255.255.255.255 via Serial0.1 (200.100.100.2)
01:43:19:        network 200.100.200.0, metric 1
01:43:19:        network 205.7.5.0, metric 1
01:43:19:        network 192.5.5.0, metric 2
01:43:19: RIP: sending v1 update to 255.255.255.255 via Serial0.2 (200.100.200.1)
01:43:19:        network 205.7.5.0, metric 1
01:43:19:        network 200.100.100.0, metric 1
01:43:19:        network 220.205.75.0, metric 2
01:43:34: RIP: received v1 update from 200.100.100.1 on Serial0.1
01:43:34:        200.100.200.0 in 2 hops
01:43:34:        205.7.5.0 in 2 hops
01:43:34:        200.100.100.0 in 1 hops
01:43:34:        192.5.5.0 in 3 hops
01:43:34:        220.205.75.0 in 1 hops
01:43:37: RIP: received v1 update from 200.100.200.2 on Serial0.2
01:43:37:        200.100.200.0 in 1 hops
01:43:37:        205.7.5.0 in 2 hops
01:43:37:        200.100.100.0 in 2 hops
01:43:37:        192.5.5.0 in 1 hops
01:43:37:        220.205.75.0 in 3 hops
Lab_B# undebug all
```

Ative o *Split Horizon* nas interfaces seriais de Lab_A e Lab_C e veja as tabelas de roteamento enviadas por Lab_A e Lab_C para Lab_B. O que aconteceu? A tabela de roteamento nos 3 roteadores se alterou?

```
Lab_A# config terminal
Lab_A(config)# interface serial 0.1
Lab_A(config-subif)# ip split-horizon
Lab_A(config-subif)# end

Lab_A# debug ip rip
01:45:09: RIP: sending v1 update to 255.255.255.255 via Serial0.1 (200.100.100.1)
01:45:09:        network 220.205.75.0, metric 1
01:45:23: RIP: received v1 update from 200.100.100.2 on Serial0.1
01:45:23:        200.100.200.0 in 1 hops
01:45:23:        205.7.5.0 in 1 hops
01:45:23:        192.5.5.0 in 2 hops
Lab_A# undebug all

Lab_A# show ip route
R    200.100.200.0/24 [120/1] via 200.100.100.2, 00:00:00, Serial0.1
R    205.7.5.0/24 [120/1] via 200.100.100.2, 00:00:00, Serial0.1
C    200.100.100.0/24 is directly connected, Serial0.1
R    192.5.5.0/24 [120/2] via 200.100.100.2, 00:00:00, Serial0.1
C    220.205.75.0/24 is directly connected, Ethernet0
```



```
Lab_B# debug ip rip
01:50:55: RIP: sending v1 update to 255.255.255.255 via Serial0.1 (200.100.100.2)
01:50:55:        network 200.100.200.0, metric 1
01:50:55:        network 205.7.5.0, metric 1
01:50:55:        network 192.5.5.0, metric 2
01:51:05: RIP: received v1 update from 200.100.100.1 on Serial0.1
01:51:05:        220.205.75.0 in 1 hops
01:51:07: RIP: sending v1 update to 255.255.255.255 via Serial0.2 (200.100.200.1)
01:51:07:        network 205.7.5.0, metric 1
01:51:07:        network 200.100.100.0, metric 1
01:51:07:        network 220.205.75.0, metric 2
01:51:16: RIP: received v1 update from 200.100.200.2 on Serial0.2
01:51:16:        192.5.5.0 in 1 hops
Lab_B# undebug all

Lab_B# show ip route
C    200.100.200.0/24 is directly connected, Serial0.2
C    205.7.5.0/24 is directly connected, Ethernet0
C    200.100.100.0/24 is directly connected, Serial0.1
R    192.5.5.0/24 [120/1] via 200.100.200.2, 00:00:10, Serial0.2
R    220.205.75.0/24 [120/1] via 200.100.100.1, 00:00:14, Serial0.1
```

```
Lab_C# config terminal
Lab_C(config)# interface serial 0.1
Lab_C(config-subif)# ip split-horizon
Lab_C(config-subif)# end

Lab_C# debug ip rip
01:52:04: RIP: sending v1 update to 255.255.255.255 via Serial0.1 (200.100.200.2)
01:52:04:        network 192.5.5.0, metric 1
01:52:04: RIP: received v1 update from 200.100.200.1 on Serial0.1
01:52:04:        205.7.5.0 in 1 hops
01:52:16:        200.100.100.0 in 1 hops
01:52:16:        220.205.75.0 in 2 hops
Lab_C# undebug all

Lab_C# show ip route
C    200.100.200.0/24 is directly connected, Serial0.1
R    205.7.5.0/24 [120/1] via 200.100.200.1, 00:00:18, Serial0.1
R    200.100.100.0/24 [120/1] via 200.100.200.1, 00:00:18, Serial0.1
C    192.5.5.0/24 is directly connected, Ethernet0
R    220.205.75.0/24 [120/2] via 200.100.200.1, 00:00:18, Serial0.1
```

g. Comando Ping na Própria Interface Frame Relay

Como estão sendo usadas subinterfaces o comando **ping para o endereço IP da própria interface serial** deverá funcionar normalmente.

```
Lab_A# ping 200.100.100.1                    ← ping na própria subinterface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.1, timeout is 2 seconds:
! ! ! ! !
Success rate is 0 percent (0/5)

Lab_A# ping 200.100.100.2                    ← ping na interface do vizinho
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.2, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 68/68/68 ms
```

```
Lab_B# ping 200.100.100.2                    ← ping na própria subinterface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.2, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 128/131/140 ms

Lab_B# ping 200.100.100.1                    ← ping na interface do vizinho
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.1, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 68/73/96 ms

Lab_B# ping 200.100.200.1                    ← ping na própria subinterface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.200.1, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 84/86/96 ms

Lab_B# ping 200.100.200.2                    ← ping na interface do vizinho
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.200.2, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 44/54/96 ms
```

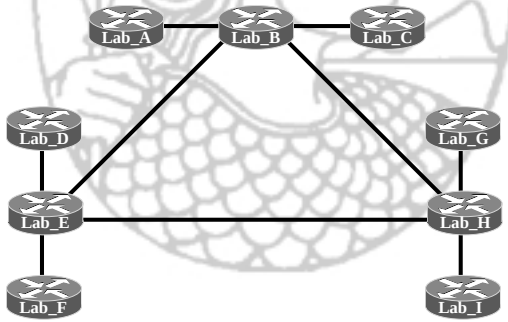
```
Lab_C# ping 200.100.200.2                    ← ping na própria subinterface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.200.2, timeout is 2 seconds:
! ! ! ! !
Success rate is 0 percent (0/5)

Lab_C# ping 200.100.200.1                    ← ping na interface do vizinho
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.200.1, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 44/55/100 ms
```


h. Interligando os 3 Grupos

Interligue os três grupos que estavam separados através dos roteadores centrais (Lab_B, Lab_E e Lab_H) usando os circuitos virtuais cujos endereços IP estão descritos na tabela a seguir.

Circuito Virtual	Endereço IP
Lab_B – Lab_E	200.250.100.0
Lab_B – Lab_H	200.250.150.0
Lab_E – Lab_H	200.250.200.0

A rede total, com nove roteadores, ficou montada conforme a figura abaixo.



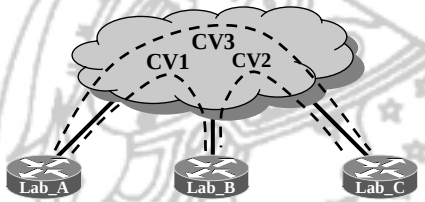
No roteador Lab_B a programação a seguir deve ser acrescentada.

```
Lab_B# config terminal
Lab_B(config)# interface serial 0.3 point-to-point
Lab_B(config-subif)# description CV para Lab_E
Lab_B(config-subif)# ip address 200.250.100.1 255.255.255.0
Lab_B(config-subif)# frame-relay interface-dlci 250
Lab_B(config-fr-dlci)# interface serial 0.4 point-to-point
Lab_B(config-subif)# description CV para Lab_H
Lab_B(config-subif)# ip address 200.250.150.1 255.255.255.0
Lab_B(config-subif)# frame-relay interface-dlci 280
Lab_B(config-fr-dlci)# end
```

Verifique as tabelas de roteamento para a rede completa.

Exercício 2 – Conexões Ponto-a-Ponto Full-Mesh

Cada grupo será ligado conforme a figura abaixo e três circuitos virtuais foram contratados na Concessionária de Serviços: um que liga o primeiro roteador ao segundo (CV1), outro que liga o segundo roteador ao terceiro (CV2) e, por último um que liga o primeiro roteador ao terceiro (CV3).



Configure os roteadores utilizando os endereços IP para cada circuito virtual contratado de acordo com a tabela a seguir.

Grupo	Circuito Virtual	IP
I	CV1 = Lab_A – Lab_B	200.100.100.0
	CV2 = Lab_B – Lab_C	200.100.200.0
	CV3 = Lab_A – Lab_C	200.100.150.0
II	CV1 = Lab_D – Lab_E	200.150.100.0
	CV2 = Lab_E – Lab_F	200.150.200.0
	CV3 = Lab_D – Lab_F	200.150.150.0
III	CV1 = Lab_G – Lab_H	200.200.100.0
	CV2 = Lab_H – Lab_I	200.200.200.0
	CV3 = Lab_G – Lab_I	200.200.150.0

Como todos os roteadores possuem 2 circuitos virtuais, será necessário usar subinterface para configurar a interface S0 de todos eles.

a. Programação de Lab_A

```
Lab_A# config terminal
Lab_A(config)# interface serial 0
Lab_A(config-if)# encapsulation frame-relay
Lab_A(config-if)# no ip address
Lab_A(config-if)# interface serial 0.1 point-to-point
Lab_A(config-subif)# description CV1 para Lab_B
Lab_A(config-subif)# ip address 200.100.100.1 255.255.255.0
Lab_A(config-subif)# frame-relay interface-dlci 120
Lab_A(config-fr-dlci)# interface serial 0.2 point-to-point
Lab_A(config-subif)# description CV3 para Lab_C
Lab_A(config-subif)# ip address 200.100.150.1 255.255.255.0
Lab_A(config-subif)# frame-relay interface-dlci 130
Lab_A(config-fr-dlci)# end
```


b. Programação de Lab_B

```
Lab_B# config terminal
Lab_B(config)# interface serial 0
Lab_B(config-if)# encapsulation frame-relay
Lab_B(config-if)# no ip address
Lab_B(config-if)# interface serial 0.1 point-to-point
Lab_B(config-subif)# description CV1 para Lab_A
Lab_B(config-subif)# ip address 200.100.100.2 255.255.255.0
Lab_B(config-subif)# frame-relay interface-dlci 210
Lab_B(config-fr-dlci)# interface serial 0.2 point-to-point
Lab_B(config-subif)# description CV2 para Lab_C
Lab_B(config-subif)# ip address 200.100.200.1 255.255.255.0
Lab_B(config-subif)# frame-relay interface-dlci 230
Lab_B(config-fr-dlci)# end
```

c. Programação de Lab_C

```
Lab_C# config terminal
Lab_C(config)# interface serial 0
Lab_C(config-if)# encapsulation frame-relay
Lab_C(config-if)# no ip address
Lab_C(config-if)# interface serial 0.1 point-to-point
Lab_C(config-subif)# description CV2 para Lab_B
Lab_C(config-subif)# ip address 200.100.200.2 255.255.255.0
Lab_C(config-subif)# frame-relay interface-dlci 320
Lab_C(config-fr-dlci)# interface serial 0.2 point-to-point
Lab_C(config-subif)# description CV3 para Lab_A
Lab_C(config-subif)# ip address 200.100.150.2 255.255.255.0
Lab_C(config-subif)# frame-relay interface-dlci 310
Lab_C(config-fr-dlci)# end
```

d. Tabelas de Roteamento

Libere o roteamento RIP nos roteadores e verifique se os 3 roteadores possuem rotas para as redes ethernet dos demais. Como a rede permite caminhos alternativos, na tabela de roteamento de cada roteador existe uma rede com 2 rotas de mesma métrica.

```
Lab_A# show ip route
R    200.100.200.0/24 [120/1] via 200.100.100.2, 00:00:02, Serial0.1
                              [120/1] via 200.100.150.2, 00:00:25, Serial0.2
R    205.7.5.0/24 [120/1] via 200.100.100.2, 00:00:02, Serial0.1
C    200.100.100.0/24 is directly connected, Serial0.1
R    192.5.5.0/24 [120/1] via 200.100.150.2, 00:00:25, Serial0.2
C    200.100.150.0/24 is directly connected, Serial0.2
C    220.205.75.0/24 is directly connected, Ethernet0
```

```
Lab_B# show ip route
C    200.100.200.0/24 is directly connected, Serial0.2
C    205.7.5.0/24 is directly connected, Ethernet0
C    200.100.100.0/24 is directly connected, Serial0.1
R    192.5.5.0/24 [120/1] via 200.100.200.2, 00:00:28, Serial0.2
R    200.100.150.0/24 [120/1] via 200.100.100.1, 00:00:03, Serial0.1
                              [120/1] via 200.100.200.2, 00:00:28, Serial0.2
R    220.205.75.0/24 [120/1] via 200.100.100.1, 00:00:03, Serial0.1
```

```
Lab_C# show ip route
C    200.100.200.0/24 is directly connected, Serial0.1
R    205.7.5.0/24 [120/1] via 200.100.200.1, 00:00:10, Serial0.1
R    200.100.100.0/24 [120/1] via 200.100.200.1, 00:00:10, Serial0.1
                      [120/1] via 200.100.150.1, 00:00:14, Serial0.2
C    192.5.5.0/24 is directly connected, Ethernet0
C    200.100.150.0/24 is directly connected, Serial0.2
R    220.205.75.0/24 [120/1] via 200.100.150.1, 00:00:14, Serial0.2
```

e. Verificando os Circuitos Virtuais

Verifique o mapeamento IP x DLCI entrando com o comando ***show frame-relay map***.

```
Lab_A# sh frame-relay map
Serial0.1 (up): point-to-point dlci, dlci 120(0x78,0x1C80), broadcast
                status defined, active
Serial0.2 (up): point-to-point dlci, dlci 130(0x82,0x2020), broadcast
                status defined, active
```

```
Lab_B# sh frame-relay map
Serial0.1 (up): point-to-point dlci, dlci 210(0xD2,0x3420), broadcast
        status defined, active
Serial0.2 (up): point-to-point dlci, dlci 230(0xE6,0x3860), broadcast
        status defined, active
```

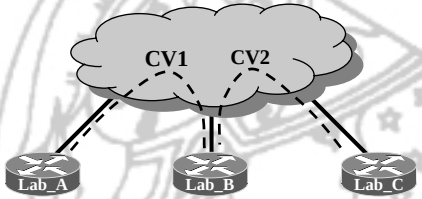
```
Lab_C# sh frame-relay map
Serial0.1 (up): point-to-point dlci, dlci 320(0x140,0x5000), broadcast
        status defined, active
Serial0.2 (up): point-to-point dlci, dlci 310(0x136,0x4C60), broadcast
        status defined, active
```

f. Comando Ping na Própria Interface Frame Relay

Como estão sendo usadas subinterfaces o comando **ping para o endereço IP da própria interface serial** deverá funcionar normalmente.

Exercício 3 – Conexões Multiponto

Cada grupo será ligado conforme a figura abaixo e apenas dois circuitos virtuais foram contratados na Concessionária de Serviços: um que liga o primeiro roteador ao segundo (CV1) e outro que liga o segundo roteador ao terceiro (CV2). Os 3 roteadores devem estar na mesma sub-rede, implementando, assim, uma conexão multiponto.



Configure os roteadores utilizando os seguintes endereços IP para cada sub-rede:

Grupo	IP
I	200.100.100.0
II	200.150.100.0
III	200.200.100.0

Como o roteador central tem um circuito virtual multiponto o uso de subinterface para configurar a interface S0 é obrigatório. Na configuração dos outros roteadores pode ser usada interface ou subinterface.

1. Programações Usando Interface na Serial 0 de Lab_A e Lab_C

Para evitar os problemas encontrados quando são usadas interfaces (aparecimento de outros DLCIs existentes na nuvem que forma o laboratório e o não funcionamento do comando ping na própria interface) serão usadas apenas subinterfaces.

Porém, ao usar interface é importante destacar que Lab_A não sabe qual DLCI deve ser usado para chegar em Lab_C (ele só sabe qual usar para chegar em Lab_B). O mesmo se aplica para Lab_C chegar em Lab_A. Isso não ocorre quando se usa subinterfaces.

Esse problema é solucionado usando-se o comando ***frame-relay map*** de forma a se criar uma entrada estática na tabela IP x DLCI. Esse comando deve ser dado em Lab_A e Lab_C. A programação de Lab_B não se altera.

a. Programação de Lab_A

```
Lab_A# config terminal
Lab_A(config)# interface serial 0
Lab_A(config-if)# description CV1 para Lab_B
Lab_A(config-if)# encapsulation frame-relay
Lab_A(config-if)# ip address 200.100.100.1 255.255.255.0
Lab_A(config-if)# frame-relay interface-dlci 120
Lab_A(config-fr-dlci)# frame-relay map ip 200.100.100.3 120 broadcast
Lab_A(config-if)# end
```

b. *Programação de Lab_B*

Como nos exercícios anteriores as subinterfaces S0.1 e S0.2 foram usadas como tipo ponto-a-ponto. Para usar uma dessas subinterfaces como tipo multiponto seria necessário excluir essas subinterfaces (comando ***no interface s0.1*** e ***no interface s0.2***) e dar um ***reload*** no roteador Lab_B. Isso ocorre porque o IOS não permite trocar o tipo de uma subinterface. Para evitar isso será usada uma subinterface nova.

```
Lab_B# config terminal
Lab_B(config)# interface serial 0
Lab_B(config-if)# encapsulation frame-relay
Lab_B(config-if)# no ip address
Lab_B(config-if)# interface serial 0.3 multipoint
Lab_B(config-subif)# description CV multiponto para Lab_A e Lab_C
Lab_B(config-subif)# ip address 200.100.100.2 255.255.255.0
Lab_B(config-subif)# frame-relay interface-dlci 210
Lab_B(config-fr-dlci)# frame-relay interface-dlci 230
Lab_B(config-fr-dlci)# end
```

c. *Programação de Lab_C*

```
Lab_C# config terminal
Lab_C(config)# interface serial 0
Lab_C(config-if)# description CV2 para Lab_B
Lab_C(config-if)# encapsulation frame-relay
Lab_C(config-if)# ip address 200.100.100.3 255.255.255.0
Lab_C(config-if)# frame-relay interface-dlci 320
Lab_C(config-fr-dlci)# frame-relay map ip 200.100.100.1 320 broadcast
Lab_C(config-if)# end
```

d. *Verificando os Circuitos Virtuais*

Verifique o mapeamento IP x DLCI entrando com o comando ***show frame-relay map***. Para esse laboratório funcionar é necessário excluir da nuvem Frame Relay todos os circuitos virtuais não utilizados pelos 3 grupos.

```
Lab_A# sh frame-relay map
Serial0 (up): ip 200.100.100.2 dlci 120(0x78,0x1C80), dynamic,
              broadcast,, status defined, active
Serial0 (up): ip 200.100.100.3 dlci 120(0x78,0x1C80), static,
              broadcast,
              CISCO, status defined, active
```

```
Lab_B# sh frame-relay map
Serial0.3 (up): ip 200.100.100.1 dlci 210(0xD2,0x3420), dynamic,
                broadcast,, status defined, active
Serial0.3 (up): ip 200.100.100.3 dlci 230(0xE6,0x3860), dynamic,
                broadcast,, status defined, active
```

```
Lab_C# sh frame-relay map
Serial0 (up): ip 200.100.100.1 dlci 320(0x140,0x5000), static,
              broadcast,
              CISCO, status defined, active
Serial0 (up): ip 200.100.100.2 dlci 320(0x140,0x5000), dynamic,
              broadcast,, status defined, active
```


e. *Comando Ping na Própria Interface Frame Relay*

O comando ping na própria interface Frame Relay não funciona em Lab_A e Lab_C porque está sendo usada interface (e não subinterface). No caso de Lab_B, o ping na própria subinterface Frame Relay tipo multiponto não funciona, conforme definido no IOS da Cisco. Isso pode ser visto no artigo técnico da Cisco "*Frame Relay Characteristics*" que está disponível link abaixo:
http://www.cisco.com/en/US/tech/tk713/tk237/technologies_tech_note09186a00800942ab.shtml

```
Lab_A# ping 200.100.100.1                    ← ping na própria interface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.1, timeout is 2 seconds:
. . . . .
Success rate is 0 percent (0/5)

Lab_A# ping 200.100.100.2                    ← ping na interface de Lab_B
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.2, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 64/65/68 ms

Lab_A# ping 200.100.100.3                    ← ping na interface de Lab_C
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.3, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 108/108/108 ms
```

```
Lab_B# ping 200.100.100.1                    ← ping na interface de Lab_A
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.1, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 64/64/64 ms

Lab_B# ping 200.100.100.2                    ← ping na própria subinterface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.2, timeout is 2 seconds:
. . . . .
Success rate is 0 percent (0/5)

Lab_B# ping 200.100.100.3                    ← ping na interface de Lab_C
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.3, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 44/44/44 ms
```

```
Lab_C# ping 200.100.100.1                    ← ping na interface de Lab_A
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.1, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 108/110/120 ms

Lab_C# ping 200.100.100.2                    ← ping na interface de Lab_B
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.2, timeout is 2 seconds:
! ! ! ! !
Success rate is 100 percent (5/5), round-trip min/avg/max = 44/44/44 ms

Lab_C# ping 200.100.100.3                    ← ping na própria interface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.3, timeout is 2 seconds:
. . . . .
Success rate is 0 percent (0/5)
```

2. *Programações Usando Subinterface na Serial 0 de Lab_A e Lab_C*

a. *Programação de Lab_A*

Como está sendo usada subinterface, o comando ***frame-relay map*** não é necessário.

```
Lab_A# config terminal
Lab_A(config)# interface serial 0
Lab_A(config-if)# encapsulation frame-relay
Lab_A(config-if)# no ip address
Lab_A(config-if)# interface serial 0.1 point-to-point
Lab_A(config-subif)# description CV1 para Lab_B
Lab_A(config-subif)# ip address 200.100.100.1 255.255.255.0
Lab_A(config-subif)# frame-relay interface-dlci 120
Lab_A(config-fr-dlci)# end
```

b. *Programação de Lab_B*

```
Lab_B# config terminal
Lab_B(config)# interface serial 0
Lab_B(config-if)# encapsulation frame-relay
Lab_B(config-if)# no ip address
Lab_B(config-if)# interface serial 0.3 multipoint
Lab_B(config-subif)# description CV multiponto para Lab_A e Lab_C
Lab_B(config-subif)# ip address 200.100.100.2 255.255.255.0
Lab_B(config-subif)# frame-relay interface-dlci 210
Lab_B(config-fr-dlci)# frame-relay interface-dlci 230
Lab_B(config-fr-dlci)# end
```


c. *Programação de Lab_C*

Como está sendo usada subinterface, o comando ***frame-relay map*** não é necessário.

```
Lab_C# config terminal
Lab_C(config)# interface serial 0
Lab_C(config-if)# encapsulation frame-relay
Lab_C(config-if)# no ip address
Lab_C(config-if)# interface serial 0.1 point-to-point
Lab_C(config-subif)# description CV2 para Lab_B
Lab_C(config-subif)# ip address 200.100.100.3 255.255.255.0
Lab_C(config-subif)# frame-relay interface-dlci 320
Lab_C(config-fr-dlci)# end
```

d. *Tabelas de Roteamento*

Libere o roteamento RIP nos roteadores e verifique as tabelas de roteamento.

```
Lab_A# show ip route
R    205.7.5.0/24 [120/1] via 200.100.100.2, 00:00:02, Serial0.1
C    200.100.100.0/24 is directly connected, Serial0.1
C    220.205.75.0/24 is directly connected, Ethernet0
```

```
Lab_B# show ip route
C    205.7.5.0/24 is directly connected, Ethernet0
C    200.100.100.0/24 is directly connected, Serial0.3
R    192.5.5.0/24 [120/1] via 200.100.100.3, 00:00:28, Serial0.3
R    220.205.75.0/24 [120/1] via 200.100.100.1, 00:00:05, Serial0.3
```

```
Lab_C# show ip route
R    205.7.5.0/24 [120/1] via 200.100.100.2, 00:00:08, Serial0.1
C    200.100.100.0/24 is directly connected, Serial0.1
C    192.5.5.0/24 is directly connected, Ethernet0
```

Observe que o roteador Lab_A está sem rota para a ethernet de Lab_C e vice-versa, embora Lab_B possua todas as rotas. Isso ocorre porque na subinterface Frame Relay multiponto existente em Lab_B o *split horizon* está ativo por default. Assim Lab_B não divulga pela interface serial 0 as rotas aprendidas por essa interface.

Desative o *split horizon* (comando no ***ip split-horizon***) na subinterface de Lab_B (esse comando não irá funcionar se dado na interface) e verifique as novas tabelas de roteamento.

```
Lab_B# config terminal
Lab_B(config-if)# interface serial 0.3
Lab_B(config-subif)# no ip split-horizon
Lab_B(config-subif)# end
```

Como a subinterface serial 0.3 já está criada não é necessário colocar o tipo (multipoint) ao fazer a programação.

```
Lab_A# show ip route
R    205.7.5.0/24 [120/1] via 200.100.100.2, 00:00:15, Serial0.1
C    200.100.100.0/24 is directly connected, Serial0.1
R    192.5.5.0/24 [120/2] via 200.100.100.2, 00:00:15, Serial0.1
C    220.205.75.0/24 is directly connected, Ethernet0
```

```
Lab_B# show ip route
C    205.7.5.0/24 is directly connected, Ethernet0
C    200.100.100.0/24 is directly connected, Serial0.3
R    192.5.5.0/24 [120/1] via 200.100.100.3, 00:00:28, Serial0.3
R    220.205.75.0/24 [120/1] via 200.100.100.1, 00:00:05, Serial0.3
```

```
Lab_C# show ip route
R    205.7.5.0/24 [120/1] via 200.100.100.2, 00:00:07, Serial0.1
C    200.100.100.0/24 is directly connected, Serial0.1
C    192.5.5.0/24 is directly connected, Ethernet0
R    220.205.75.0/24 [120/2] via 200.100.100.2, 00:00:08, Serial0.1
```

e. Verificando os Circuitos Virtuais

Verifique o mapeamento IP x DLCI entrando com o comando ***show frame-relay map***.

```
Lab_A# sh frame-relay map
Serial0.1 (up): point-to-point dlci, dlci 120(0x78,0x1C80), broadcast
                status defined, active
```

```
Lab_B# sh frame-relay map
Serial0.3 (up): ip 200.100.100.1 dlci 210(0xD2,0x3420), dynamic,
                broadcast,, status defined, active
Serial0.3 (up): ip 200.100.100.3 dlci 230(0xE6,0x3860), dynamic,
                broadcast,, status defined, active
```

```
Lab_C# sh frame-relay map
Serial0.1 (up): point-to-point dlci, dlci 320(0x140,0x5000), broadcast
                status defined, active
```

Como o tipo da interface em Lab_B é multiponto, não é possível saber em Lab_A e em Lab_C qual o endereço IP que está associado à interface de Lab_B do outro lado do circuito virtual.

f. Comando Ping na Própria Interface Frame Relay

No caso de Lab_B, o ping na própria subinterface Frame Relay tipo multiponto não funciona conforme definido no IOS da Cisco. Isso pode ser visto no artigo técnico da Cisco "*Frame Relay Characteristics*".


```
Lab_A# ping 200.100.100.1                    ← ping na própria subinterface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 128/132/140 ms

Lab_A# ping 200.100.100.2                    ← ping na interface de Lab_B
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 64/64/64 ms

Lab_A# ping 200.100.100.3                    ← ping na interface de Lab_C
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 104/109/120 ms
```

```
Lab_B# ping 200.100.100.1                    ← ping na interface de Lab_A
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 64/64/64 ms

Lab_B# ping 200.100.100.2                    ← ping na própria subinterface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.2, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)

Lab_B# ping 200.100.100.3                    ← ping na interface de Lab_C
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 44/44/44 ms
```

```
Lab_C# ping 200.100.100.1                    ← ping na interface de Lab_A
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.1, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 108/111/124 ms

Lab_C# ping 200.100.100.2                    ← ping na interface de Lab_B
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.2, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 44/44/44 ms

Lab_C# ping 200.100.100.3                    ← ping na própria subinterface
Type escape sequence to abort.
Sending 5, 100-byte ICMP Echos to 200.100.100.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round-trip min/avg/max = 84/84/84 ms
```

Configuring Frame Relay

This chapter describes the tasks for configuring Frame Relay on a router or access server. For a complete description of the commands mentioned in this chapter, refer to the "Frame Relay Commands" chapter in the *Wide-Area Networking Command Reference*.

Although Frame Relay access was originally restricted to leased lines, dial-up access is now supported. For more information, see the "Configure DDR over Frame Relay" section in the "Configuring DDR" chapter of this manual.

To install software on a new router or access server by downloading software from a central server over an interface that supports Frame Relay, see the "Loading System Images, Microcode Images, and Configuration Files" chapter in the *Configuration Fundamentals Configuration Guide*.

To configure access between SNA devices over a Frame Relay network, see the "Configuring SNA Frame Relay Access Support" chapter in the *Bridging and IBM Networking Configuration Guide*.

Frame Relay Hardware Configurations

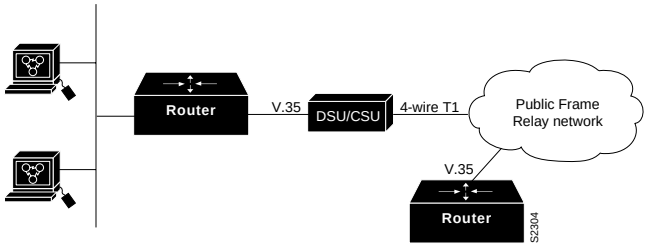
One of the following hardware configurations is possible for Frame Relay connections:

- Routers and access servers can connect directly to the Frame Relay switch.
- Routers and access servers can connect directly to a channel service unit/digital service unit (CSU/DSU), which then connects to a remote Frame Relay switch.

Note A Frame Relay network is not required to support only routers that are connected directly or only routers connected via CSU/DSUs. Within a network, some routers can connect to a Frame Relay switch through a direct connection and others through connections via CSU/DSUs. However, a single router interface configured for Frame Relay can be only one or the other.

The CSU/DSU converts V.35 or RS-449 signals to the properly coded T1 transmission signal for successful reception by the Frame Relay network. Figure 28 illustrates the connections between the different components.

Figure 28 Typical Frame Relay Configuration



The Frame Relay interface actually consists of one physical connection between the network server and the switch that provides the service. This single physical connection provides direct connectivity to each device on a network, such as a StrataCom FastPacket wide-area network (WAN).

Frame Relay Configuration Task List

There are required, basic steps you must follow to enable Frame Relay for your network. In addition, you can customize Frame Relay for your particular network needs and monitor Frame Relay connections. The following sections outline these tasks. The tasks in the first two sections are required.

- Enable Frame Relay Encapsulation on an Interface
- Configure Dynamic or Static Address Mapping
- Configure the LMI
- Configure Frame Relay Switched Virtual Circuits
- Configure Frame Relay Traffic Shaping
- Customize Frame Relay for Your Network
- Monitor the Frame Relay Connections

See the "Frame Relay Configuration Examples" section at the end of this chapter for ideas of how to configure Frame Relay on your network. See the "Frame Relay Commands" chapter in the *Wide-Area Networking Command Reference* for information about the commands listed in the tasks.

Enable Frame Relay Encapsulation on an Interface

To set Frame Relay encapsulation at the interface level, perform the following tasks beginning in global configuration mode:

Task	Command
Specify the serial interface, and enter interface configuration mode.	**interface serial** *number*[1]
Enable Frame Relay, and specify the encapsulation method.	**encapsulation frame-relay** [**ietf**]

1. This command is documented in the "Interface Commands" chapter in the *Configuration Fundamentals Command Reference*.

Frame Relay supports encapsulation of all supported protocols in conformance with RFC 1490, allowing interoperability between multiple vendors. Use the Internet Engineering Task Force (IETF) form of Frame Relay encapsulation if your router or access server is connected to another vendor's equipment across a Frame Relay network. IETF encapsulation is supported either at the interface level or on a per-virtual circuit basis.

For an example of how to enable Frame Relay and set the encapsulation method, see the sections "IETF Encapsulation Examples" and "Static Address Mapping Examples" later in this chapter.

Configure Dynamic or Static Address Mapping

Dynamic address mapping uses Frame Relay Inverse ARP to request the next hop protocol address for a specific connection, given its known DLCI. Responses to Inverse ARP requests are entered in an address-to-DLCI mapping table on the router or access server; the table is then used to supply the next hop protocol address or the DLCI for outgoing traffic.

Inverse ARP is enabled by default for all protocols it supports, but can be disabled for specific protocol-DLCI pairs. As a result, you can use dynamic mapping for some protocols and static mapping for other protocols on the same DLCI. You can explicitly disable Inverse ARP for a protocol-DLCI pair if you know that the protocol is not supported on the other end of the connection. See the "Disable or Reenable Frame Relay Inverse ARP" section later in this chapter for more information.

Configure Dynamic Mapping

Inverse ARP is enabled by default for all protocols enabled on the physical interface. Packets are not sent out for protocols that are not enabled on the interface.

Because Inverse ARP is enabled by default, no additional command is required to configure dynamic mapping on an interface.

Configure Static Mapping

A static map links a specified next hop protocol address to a specified DLCI. Static mapping removes the need for Inverse ARP requests; when you supply a static map, Inverse ARP is automatically disabled for the specified protocol on the specified DLCI.

You must use static mapping if the router at the other end either does not support Inverse ARP at all or does not support Inverse ARP for a specific protocol that you want to use over Frame Relay.

To establish static mapping according to your network needs, perform one of the following tasks in interface configuration mode:

Task	Command
Define the mapping between a next hop protocol address and the DLCI used to connect to the address.	**frame-relay map** *protocol protocol-address dlci* [**broadcast**] [**ietf**] [**cisco**]
Define a DLCI used to send International Organization for Standardization (ISO) Connectionless Network Service (CLNS) frames.	**frame-relay map clns** *dlci* [**broadcast**]
Define a DLCI used to connect to a bridge.	**frame-relay map bridge** *dlci* [**broadcast**] [**ietf**]

The supported protocols and the corresponding keywords to enable them are as follows:

- IP—**ip**
- DECnet—**decnet**
- AppleTalk—**appletalk**
- XNS—**xns**
- Novell IPX—**ipx**
- VINES—**vines**
- ISO CLNS—**clns**

You can greatly simplify the configuration for the Open Shortest Path First (OSPF) protocol by adding the optional **broadcast** keyword when doing this task. See the **frame-relay map** command description in the *Wide-Area Networking Command Reference* and the examples at the end of this chapter for more information about using the **broadcast** keyword.

For examples of how to establish static address mapping, see the "Static Address Mapping Examples" section later in this chapter.

Configure the LMI

Beginning with Cisco IOS Release 11.2, the software supports Local Management Interface (LMI) *autosense*, which enables the interface to determine the LMI type supported by the switch. Support for LMI autosense means that you are no longer required to configure the Local Management Interface (LMI) explicitly.

Allow LMI Autosense to Operate

LMI autosense is active in the following situations:

- The router is powered up or the interface changes state to up.
- The line protocol is down but the line is up.
- The interface is a Frame Relay DTE.
- The LMI type is not explicitly configured.

When LMI autosense is active, it sends out a full status request, in all 3 LMI flavors, to the switch. The order is ANSI, ITU, cisco but is done in rapid succession. Unlike previous software capability, we can now listen in on both DLCI 0 (cisco LMI) and DLCI 1023 (ANSI and ITU) simultaneously.

One or more of the status requests will elicit a reply (status message) from the switch. The router will decode the format of the reply and configure itself automatically. If more than one reply is received, the router will configure itself with the type of the last received reply. This is to accommodate intelligent switches that can handle multiple formats simultaneously.

If LMI autosense is unsuccessful, an intelligent retry scheme is built in. Every N391 interval (default is 60 seconds, which is 6 keep exchanges at 10 seconds each), LMI autosense will attempt to ascertain the LMI type. For more information about N391, see the **frame-relay lmi-n391dte** command in the "Frame Relay Commands" chapter of the *Wide-Area Networking Command Reference*.

The only visible indication to the user that LMI autosense is underway is when "debug frame lmi" is turned on. Every N391 interval, the user will now see 3 rapid status enquiries coming out of the serial interface. One in ANSI, one in ITU and one in cisco LMI-type.

No configuration options are provided; this is transparent to the user. You can turn off LMI autosense by explicitly configuring an LMI type. The LMI type must be written into NVRAM so that next time the router powers up, LMI autosense will be inactive. At the end of autoinstall, a "frame-relay lmi-type xxx" statement is included within the interface configuration. This configuration is not automatically written to NVRAM; you must do so explicitly.

Explicitly Configure the LMI

Our Frame Relay software supports the industry-accepted standards for addressing the Local Management Interface (LMI), including the Cisco specification. If you want to configure the LMI and thus deactivate LMI autosense, complete the tasks in the following sections. The tasks in the first two sections are required if you choose to configure the LMI.

- Set the LMI Type
- Set the LMI Keepalive Interval
- Set the LMI Polling and Timer Intervals

Set the LMI Type

If the router or access server is attached to a public data network (PDN), the LMI type must match the type used on the public network. Otherwise, the LMI type can be set to suit the needs of your private Frame Relay network.

You can set one of three types of LMIs on our devices: ANSI T1.617 Annex D, Cisco, and ITU-T Q.933 Annex A. To do so, perform the following task in interface configuration mode:

Task	Command
Set the LMI type.	**frame-relay lmi-type** {**ansi** | **cisco** | **q933a**}
Write the LMI type to NVRAM.	**copy runnning-config** *destination*[1]

1. Use the command form that is appropriate to your router platform. See the "Image and Configuration File Load Commands" chapter in the *Configuration Fundamentals Command Reference*.

For an example of how to set the LMI type, see the "Pure Frame Relay DCE Example" section later in this chapter.

Set the LMI Keepalive Interval

A keepalive interval must be set to configure the LMI. By default, this interval is 10 seconds and, per the LMI protocol, must be less than the corresponding interval on the switch. To set the keepalive interval, perform the following task in interface configuration mode:

Task	Command
Set the keepalive interval.	**keepalive** *number*
Turn off keepalives on networks without an LMI.	**no keepalive**

For an example of how to specify an LMI keepalive interval, see the "Two Routers in Static Mode Example" section later in this chapter.

Set the LMI Polling and Timer Intervals

You can set various optional counters, intervals, and thresholds to fine-tune the operation of your LMI DTE and DCE devices. Set these attributes by performing one or more of the following tasks in interface configuration mode:

Task	Command
Set the DCE and Network-to-Network Interface (NNI) error threshold.	**frame-relay lmi-n392dce** *threshold*
Set the DCE and NNI monitored events count.	**frame-relay lmi-n393dce** *events*
Set the polling verification timer on a DCE or NNI interface.	**frame-relay lmi-t392dce** *timer*
Set a full status polling interval on a DTE or NNI interface.	**frame-relay lmi-n391dte** *keep-exchanges*
Set the DTE or NNI error threshold.	**frame-relay lmi-n392dte** *threshold*
Set the DTE and NNI monitored events count.	**frame-relay lmi-n393dte** *events*

See the "Frame Relay Commands" chapter in the *Wide-Area Networking Command Reference* for details about commands used to set the polling and timing intervals.

Configure Frame Relay Switched Virtual Circuits

Currently, access to Frame Relay networks is made through private leased lines at speeds ranging from 56 kbps to 45 Mbps. Frame Relay is a connection-oriented, packet-transfer mechanism that establishes virtual circuits between endpoints.

Switched virtual circuits (SVCs) allow access through a Frame Relay network by setting up a path to the destination endpoints only when the need arises and tearing down the path when it is no longer needed.

SVCs can coexist with PVCs in the same sites and routers. For example, routers at remote branch offices might set up PVCs to the central headquarters for frequent communication, but set up SVCs with each other as needed for intermittent communication. As a result, any-to-any communication can be set up without any-to-any PVCs.

On SVCs, quality of service (QOS) elements can be specified on a call-by-call basis to request network resources.

SVC support is offered in the Enterprise image on Cisco platforms that include a serial or HSSI interface (Cisco 7000 and 7500 series, Cisco 4500 and 4700, Cisco 4000, Cisco 3000, and Cisco 2500 platforms).

You must have the following services before Frame Relay SVCs can operate:

- Frame Relay SVC support by the service provider—The service provider's switch must be capable of supporting SVC operation.
- Physical loop connection—A leased line or dedicated line must exist between the router (DTE) and the local Frame Relay switch.

SVC operation requires that the Data Link layer (Layer 2) be set up, running ITU-T Q.922 Link Access Procedures to Frame mode bearer services (LAPF), prior to signalling for an SVC. Layer 2 sets itself up as soon as SVC support is enabled on the interface, if both the line and the line protocol are up. When the SVCs are configured and demand for a path occurs, the Q.933 signalling sequence is initiated. Once the SVC is set up, data transfer begins.

Q.922 provides a reliable link layer for Q.933 operation. All Q.933 call control information is transmitted over DLCI 0; this DLCI is also used for the management protocols specified in ANSI T1.617 Annex D or Q.933 Annex A.

You must enable SVC operation at the interface level. Once it is enabled at the interface level, it is enabled on any subinterfaces on that interface. One signalling channel, DLCI 0, is set up for the interface, and all SVCs are controlled from the physical interface.

To enable Frame Relay SVC service and set up SVCs, complete the tasks in the following sections. The subinterface tasks are not required, but offer additional flexibility for SVC configuration and operation. The LAPF tasks are not required and not recommended unless you understand thoroughly the impacts on your network.

- Configure SVCs on a Physical Interface
- Configure SVCs on a Subinterface (optional)
- Configure a Map Class
- Configure a Map Group with E.164 or X.121 Addresses
- Associate the Map Class with Static Protocol Address Maps
- Configure LAPF Parameters

See the "SVC Configuration Examples" section at the end of this chapter.

Configure SVCs on a Physical Interface

To enable SVC operation on a Frame Relay interface, perform the following tasks beginning in global configuration mode:

Task	Command
Specify the physical interface.	**interface serial** *number* [1]
Specify the interface IP address, if needed.	**ip address** *ip-address mask* [2]
Enable Frame Relay encapsulation on the interface.	**encapsulation frame-relay**
Assign a map group to the interface.	**map-group** *group-name*
Enable Frame Relay SVC support on the interface.	**frame-relay svc**

1. This command is documented in the "Interface Commands" chapter in the *Configuration Fundamentals Command Reference*.
2. This command is documented in the "IP Commands" chapter in the *Network Prototols Command Reference, Part 1*.

Map-group details are specified with the **map-list** command.

Configure SVCs on a Subinterface

To configure Frame Relay SVCs on a subinterface, perform all the tasks in the previous section, except assigning a the map group. After the physical interface is configured, complete the following tasks beginning in global configuration mode:

Task	Command
Specify a subinterface of the main interface configured for SVC operation.	**interface serial** *number.subinterface-number* {**multipoint** | **point-to-point**} [1]
Specify the subinterface IP address, if needed.	**ip address** *ip-address mask* [2]
Assign a map group to the subinterface.	**map-group** *group-name*

1. This command is documented in the "Interface Commands" chapter in the *Configuration Fundamentals Command Reference*.
2. This command is documented in the "IP Commands" chapter in the *Network Prototols Command Reference, Part 1*.

Configure a Map Class

To configure a map class, you can perform the following tasks. Only the first task is required.

- Specify the map class name.
- Specify a custom queue list for the map class.
- Specify a priority queue list for the map class.
- Enable BECN feedback to throttle the output rate on the SVC for the map class.
- Set nondefault QOS values for the map class.

  You are not required to set the QOS values; default values are provided.

To configure a map class, perform the following tasks beginning in global configuration mode:

Task	Command
Specify the Frame Relay map class name and enter map class configuration mode.	**map-class frame-relay** *map-class-name*
Specify a custom queue list to be used for the map class.	**frame-relay custom-queue-list** *list-number*
Assign a priority queue to virtual circuits associated with the map class.	**frame-relay priority-group** *list-number*
Enable BECN feedback to throttle the frame-transmission rate.	**frame-relay becn-response-enable**
Specify the inbound committed information rate (CIR).	**frame-relay cir in** *bps*
Specify the outbound committed information rate (CIR).	**frame-relay cir out** *bps*
Set the minimum acceptable incoming CIR	**frame-relay mincir in** *bps*
Set the minimum acceptable outgoing CIR.	**frame-relay mincir out** *bps*
Set the incoming committed burst size (Bc).	**frame-relay bc in** *bits*
Set the outgoing committed burst size (Bc).	**frame-relay bc out** *bits*
Set the incoming excess burst size (Be).	**frame-relay be in** *bits*

Task	Command
Set the outgoing excess burst size (Be).	**frame-relay be out** *bits*
Set the idle timeout interval.	**frame-relay idle-timer** *duration*

You can define multiple map classes. A map class is associated with a static map, not with the interface or subinterface itself. Because of the flexibility this association allows, you can define different map classes for different destinations.

Configure a Map Group with E.164 or X.121 Addresses

After you have defined a map group for an interface, you can associate the map group with a specific source and destination address to be used. You can specify E.164 addresses or X.121 addresses for the source and destination. To specify the map group to be associated with a specific interface, perform the following task in global configuration mode:

Specify the map group to be associated with specific source address and destination address for the SVC.	**map-list** *group-name* **source-addr** {**e164** | **x121**} *source-address* **dest-addr** {**e164** | **x121**} *destination-address*

Associate the Map Class with Static Protocol Address Maps

To define the protocol addresses under a **map-list** command and associate each protocol address with a specified map class, use the **class** command. Use this command for each protocol address to be associated with a map class. To associate a map class with a protocol address, perform the following task in map class configuration mode:

Specify a destination protocol address and a Frame Relay map class name from which to derive QOS information.	*protocol protocol-address* **class** *class-name* [**ietf**] [**broadcast** [**trigger**]]

The **ietf** keyword specifies RFC 1490 encapsulation; the **broadcast** keyword specifies that broadcasts must be carried. The **trigger** keyword, which can be configured only if **broadcast** is also configured, enables a broadcast packet to trigger an SVC. If an SVC already exists that uses this map class, the SVC will carry the broadcast.

Configure LAPF Parameters

Frame Relay Link Access Procedure for Frame Relay (LAPF) commands are used to tune Layer 2 system parameters to work well with the Frame Relay switch. Normally, you do not need to change the default settings.

However, if the Frame Relay network indicates that it does not support the Frame Reject frame (FRMR) at the LAPF Frame Reject procedure, complete the following task in interface configuration mode:

Task	Command
Select not to send FRMR frames at the LAPF Frame Reject procedure.	**no frame-relay lapf frmr**

By default, the Frame Reject frame is sent at the LAPF Frame Reject procedure.

Note Manipulation of Layer 2 parameters is not recommended if you do not know well the resulting functional change. For more information, refer to the ITU-T Q.922 specification for LAPF.

If you must change Layer 2 parameters for your network environment and you know well the resulting functional change, complete the following tasks as needed:

Task	Command
Set the LAPF window size *k*.	**frame-relay lapf k** *number*
Set the LAPF maximum retransmission count *N200*.	**frame-relay lapf n200** *retries*
Set the maximum length of the Information field of the LAPF I frame *N201*.	**frame-relay lapf n201** *number*
Set the LAPF retransmission timer value *T200*.	**frame-relay lapf t200** *tenths-of-a-second*
Set the LAPF link idle timer value *T203* of DLCI 0.	**frame-relay lapf t203** *seconds*

Configure Frame Relay Traffic Shaping

Beginning with Release 11.2, Cisco IOS supports Frame Relay traffic shaping, which provides the following:

- Rate enforcement on a per-virtual circuit basis—The peak rate for outbound traffic can be set to the CIR or some other user-configurable rate.
- Dynamic traffic throttling on a per-virtual circuit basis—When BECN packets indicate congestion on the network, the outbound traffic rate is automatically stepped down; when congestion eases, the outbound traffic rate is stepped up again. This feature is enabled by default.
- Enhanced queuing support on a per-virtual circuit basis—Either custom queuing or priority queuing can be configured for individual virtual circuits.

By defining separate virtual circuits for different types of traffic and specifying queuing and an outbound traffic rate for each virtual circuit, you can provide guaranteed bandwidth for each type of traffic. By specifying different traffic rates for different virtual circuits over the same line, you can perform virtual time division multiplexing. By throttling outbound traffic from high-speed lines in central offices to lower-speed lines in remote locations, you can ease congestion and data loss in the network; enhanced queuing also prevents congestion-caused data loss.

Traffic shaping applies to both PVCs and SVCs. For information about creating and configuring SVCs, see the “Configure Frame Relay Switched Virtual Circuits” section of this chapter.

To configure Frame Relay traffic shaping, perform the tasks in the following sections:

- Enable Frame Relay Encapsulation on an Interface (earlier in this chapter)
- Enable Frame Relay Traffic Shaping on the Interface
- Specify a Traffic-Shaping Map Class for the Interface
- Define a Map Class with Queuing and Traffic Shaping Parameters
- Define Access Lists
- Define Priority Queue Lists for the Map Class
- Define Custom Queue Lists for the Map Class

Enable Frame Relay Traffic Shaping on the Interface

Enabling Frame Relay traffic shaping on an interface enables both traffic shaping and per-virtual circuit queuing on all the interface's PVCs and SVCs.

To enable Frame Relay traffic shaping on the specified interface, complete the following task in interface configuration mode:

Task	Command
Enable Frame Relay traffic shaping and per-virtual circuit queuing.	**frame-relay traffic-shaping**

Specify a Traffic-Shaping Map Class for the Interface

If you specify a Frame Relay map class for a main interface, all the virtual circuits on its subinterfaces inherit all the traffic shaping parameters defined for the class.

To specify a map class for the specified interface, complete the following task in interface configuration mode:

Task	Command
Specify a Frame Relay map class for the interface.	**frame-relay class** *map-class-name*

You can override the default for a specific DLCI on a specific subinterface by using the **class** virtual circuit configuration command to assign the DLCI explicitly to a different class. See the "Configure Frame Relay Subinterfaces" section for information about setting up subinterfaces. For an example of assigning some subinterface DLCIs to the default class and assigning others explicitly to a different class, see the "Frame Relay Traffic Shaping Example" section.

Define a Map Class with Queuing and Traffic Shaping Parameters

When you define a map class for Frame Relay, you can define the average and peak rates (in bits per second) allowed on virtual circuits associated with the map class. You can also, optionally, specify *either* a custom queue-list or a priority queue-group to use on virtual circuits associated with the map class.

To define a map class, complete the following tasks beginning in global configuration mode:

Task	Command
Specify a map class to define.	**map-class frame-relay** *map-class-name*
Define the traffic rate for the map class.	**frame-relay traffic-rate** *average* [*peak*]
Specify a custom queue-list.	**frame-relay custom-queue-list** *number*
Specify a priority queue-list.	**frame-relay priority-group** *number*

Define Access Lists

You can specify access lists and associate them with the custom queue-list defined for any map class. The list number specified in the access list and the custom queue list tie them together.

See the appropriate protocol chapters for information about defining access lists for the protocols you want to transmit on the Frame Relay network.

Define Priority Queue Lists for the Map Class

You can define a priority list for a protocol and you can also define a default priority list. The number used for a specific priority list ties the list to the Frame Relay priority group defined for a specified map class.

For example, if you enter the **frame relay priority-group 2** command for the map class *fast_vcs* and then you enter the **priority-list 2 protocol decnet high** command, that priority list is used for the *fast_vcs* map class. The average and peak traffic rates defined for the *fast_vcs* map class are used for DECnet traffic.

Define Custom Queue Lists for the Map Class

You can define queue list for a protocol and a default queue list. You can also specify the maximum number of bytes to be transmitted in any cycle. The number used for a specific queue list ties the list to the Frame Relay custom-queue list defined for a specified map class.

For example, if you enter the **frame relay custom-queue-list 1** command for the map class *slow_vcs* and then you enter the **queue-list 1 protocol ip list 100** command, that queue list is used for the *slow_vcs* map class; **access-list 100** definition is also used for that map class and queue. The average and peak traffic rates defined for the *slow_vcs* map class are used for IP traffic that meets the **access list 100** criteria.

Customize Frame Relay for Your Network

Perform the tasks in the following sections to customize Frame Relay:

- Configure Frame Relay Subinterfaces
- Configure Frame Relay Switching
- Disable or Reenable Frame Relay Inverse ARP (multipoint communication only)
- Create a Broadcast Queue for an Interface
- Configure Payload Compression
- Configure TCP/IP Header Compression
- Configure Discard Eligibility
- Configure DLCI Priority Levels

Configure Frame Relay Subinterfaces

To understand and define Frame Relay Subinterfaces, perform the tasks in the following sections:

- Understand Frame Relay Subinterfaces
- Define Frame Relay Subinterfaces
- Define Subinterface Addressing

After these tasks are completed, you can also perform the following optional tasks:

- Configure Transparent Bridging for Frame Relay
- Configure a Backup Interface for a Subinterface

For an example of how to define a subinterface, see the section "Subinterface Examples" later in this chapter.

Understand Frame Relay Subinterfaces

Frame Relay subinterfaces provide a mechanism for supporting partially meshed Frame Relay networks. Most protocols assume *transitivity* on a logical network; that is, if station A can talk to station B, and station B can talk to station C, then station A should be able to talk to station C directly. Transitivity is true on LANs, but not on Frame Relay networks unless A is directly connected to C.

Additionally, certain protocols such as AppleTalk and transparent bridging cannot be supported on partially meshed networks because they require "split horizon," in which a packet received on an interface cannot be transmitted out the same interface even if the packet is received and transmitted on different virtual circuits.

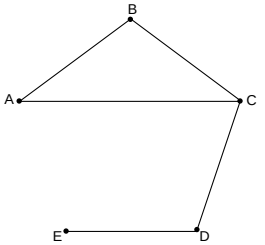
Configuring Frame Relay subinterfaces ensures that a *single physical interface* is treated as *multiple virtual interfaces.* This capability allows us to overcome split horizon rules. Packets received on one virtual interface can now be forwarded out another virtual interface, even if they are configured on the same physical interface.

Subinterfaces address the limitations of Frame Relay networks by providing a way to subdivide a partially meshed Frame Relay network into a number of smaller, fully meshed (or point-to-point) subnetworks. Each subnetwork is assigned its own network number and appears to the protocols as if it is reachable through a separate interface. (Note that point-to-point subinterfaces can be unnumbered for use with IP, reducing the addressing burden that might otherwise result.)

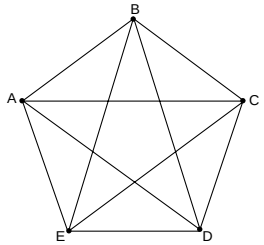
For example, suppose you have a five-node Frame Relay network (see Figure 29) that is partially meshed (Network A). If the entire network is viewed as a single subnetwork (with a single network number assigned), most protocols assume that node A can transmit a packet directly to node E, when in fact it must be relayed through nodes C and D. This network can be made to work with certain protocols (for example, IP) but will not work at all with other protocols (for example, AppleTalk) because nodes C and D will not relay the packet out the same interface on which it was received. One way to make this network work fully is to create a fully meshed network (Network B), but doing so requires a large number of PVCs, which may not be economically feasible.

Using subinterfaces, you can subdivide the Frame Relay network into three smaller subnetworks (Network C) with separate network numbers. Nodes A, B, and C are connected to a fully meshed network, and nodes C and D, as well as nodes D and E are connected via point-to-point networks. In this configuration, nodes C and D can access two subinterfaces and can therefore forward packets without violating split horizon rules. If transparent bridging is being used, each subinterface is viewed as a separate bridge port.

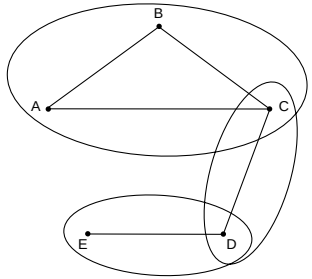
Figure 29 Using Subinterfaces to Provide Full Connectivity on a Partially Meshed Frame Relay Network



Network A: Partially Meshed Frame Relay Network without Full Connectivity



Network B: Fully Meshed Frame Relay Network with Full Connectivity



Network C: Partially Meshed Frame Relay Network with Full Connectivity (configuring subinterfaces)

S3299

Define Frame Relay Subinterfaces

To configure subinterfaces on a Frame Relay network, perform the following tasks:

Task		Command
Step 1	Specify a serial interface.	**interface serial** *number*[1]
Step 2	Configure Frame Relay encapsulation on the serial interface.	**encapsulation frame-relay**
Step 3	Specify a subinterface.	**interface serial** *number.subinterface-number* {**multipoint** | **point-to-point**}[1]

1. This command is documented in the "Interface Commands" chapter in the *Configuration Fundamentals Configuration Guide*.

Subinterfaces can be configured for multipoint or point-to-point communication. (There is no default.)

Define Subinterface Addressing

For point-to-point subinterfaces, the destination is presumed to be known and is identified or implied in the **frame-relay interface-dlci** command. For multipoint subinterfaces, the destinations can be dynamically resolved through the use of Frame Relay Inverse ARP or can be statically mapped through the use of the **frame-relay map** command.

Addressing on Point-to-Point Subinterfaces

If you specified a point-to-point subinterface in Step 3 of the previous procedure, perform the following task in interface configuration mode:

Task	Command
Associate the selected point-to-point subinterface with a DLCI.	**frame-relay interface-dlci** *dlci* [*option*]

For an explanation of the many available options, refer to this command in the *Wide-Area Networking Command Reference*. For an example of how to associate a DLCI with a subinterface, see the section "Subinterface Examples" later in this chapter.

If you define a subinterface for point-to-point communication, you cannot reassign *the same subinterface number* to be used for multipoint communication without first rebooting the router or access server. Instead, you can simply avoid using that subinterface number and use a different subinterface number instead.

Addressing on Multipoint Subinterfaces

If you specified a multipoint subinterface in Step 3 under "Define Frame Relay Subinterfaces," perform the tasks in one or both of the following sections:

- Accept Inverse ARP for Dynamic Address Mapping on Multipoint Subinterfaces
- Configure Static Address Mapping on Multipoint Subinterfaces

You can configure some protocols for dynamic address mapping and others for static address mapping.

Accept Inverse ARP for Dynamic Address Mapping on Multipoint Subinterfaces

Dynamic address mapping uses Frame Relay Inverse ARP to request the next hop protocol address for a specific connection, given a DLCI. Responses to Inverse ARP requests are entered in an address-to-DLCI mapping table on the router or access server; the table is then used to supply the next hop protocol address or the DLCI for outgoing traffic.

Since the physical interface is now configured as multiple subinterfaces, you must provide information that distinguishes a subinterface from the physical interface and associates a specific subinterface with a specific DLCI.

To associate a specific multipoint subinterface with a specific DLCI, perform the following task in interface configuration mode:

Task	Command
Associate a specified multipoint subinterface with a DLCI.	**frame-relay interface-dlci** *dlci*

Inverse ARP is enabled by default for all protocols it supports, but can be disabled for specific protocol-DLCI pairs. As a result, you can use dynamic mapping for some protocols and static mapping for other protocols on the same DLCI. You can explicitly disable Inverse ARP for a protocol-DLCI pair if you know the protocol is not supported on the other end of the connection. See the "Disable or Reenable Frame Relay Inverse ARP" section later in this chapter for more information.

Because Inverse ARP is enabled by default for all protocols that it supports, no additional command is required to configure dynamic address mapping on a subinterface.

For an example of configuring Frame Relay multipoint subinterfaces with dynamic address mapping, see the "Frame Relay Multipoint Subinterface with Dynamic Addressing Example" section.

Configure Static Address Mapping on Multipoint Subinterfaces

A static map links a specified next hop protocol address to a specified DLCI. Static mapping removes the need for Inverse ARP requests; when you supply a static map, Inverse ARP is automatically disabled for the specified protocol on the specified DLCI.

You must use static mapping if the router at the other end either does not support Inverse ARP at all or does not support Inverse ARP for a specific protocol that you want to use over Frame Relay.

To establish static mapping according to your network needs, perform one of the following tasks in interface configuration mode:

Task	Command
Define the mapping between a next hop protocol address and the DLCI used to connect to the address.	**frame-relay map** *protocol protocol-address dlci* [**broadcast**] [**ietf**] [**cisco**]
Define a DLCI used to send ISO CLNS frames.	**frame-relay map clns** *dlci* [**broadcast**]
Define a DLCI used to connect to a bridge.	**frame-relay map bridge** *dlci* [**ietf**] **broadcast**

The supported protocols and the corresponding keywords to enable them are as follows:

- IP—**ip**
- DECnet—**decnet**
- AppleTalk—**appletalk**
- XNS—**xns**
- Novell IPX—**ipx**
- VINES—**vines**
- ISO CLNS—**clns**

The **broadcast** keyword is required for routing protocols such as OSI protocols and the Open Shortest Path First (OSPF) protocol. See the **frame-relay map** command description in the *Wide-Area Networking Command Reference* and the examples at the end of this chapter for more information about using the **broadcast** keyword.

For an example of how to establish static address mapping, see the sections "Two Routers in Static Mode Example," "DECnet Routing Example," and "IPX Routing Example" later in this chapter.

Configure Transparent Bridging for Frame Relay

Transparent bridging for Frame Relay encapsulated serial and HSSI interfaces is supported on our routers. Transparent bridging for Frame Relay encapsulated serial interfaces is supported on our access servers.

You can configure transparent bridging for point-to-point or point-to-multipoint subinterfaces.

Note All PVCs configured on a subinterface belong to the same bridge group.

Point-to-Point Subinterfaces

To configure transparent bridging for point-to-point subinterfaces, complete the following tasks in interface configuration mode:

Task		Command
Step 1	Specify a serial interface.	**interface serial** *number*[1]
Step 2	Configure Frame Relay encapsulation on the serial interface.	**encapsulation frame-relay**
Step 3	Specify a subinterface.	**interface serial** *number.subinterface-number* **point-to-point**[1]
Step 4	Associate a DLCI with the subinterface.	**frame-relay interface-dlci** *dlci* [*option*]
Step 5	Associate the subinterface with a bridge group.	**bridge-group** *bridge-group*[2]

1. This command is documented in the "Interface Commands" chapter in the *Configuration Fundamentals Command Reference*.
2. This command is documented in the *Bridging and IBM Networking Command Reference*.

Point-to-Multipoint Interfaces

To configure transparent bridging for point-to-multipoint subinterfaces, complete the following tasks in interface configuration mode:

Task		Command
Step 1	Specify a serial interface.	**interface serial** *number*[1]
Step 2	Configure Frame Relay encapsulation on the serial interface.	**encapsulation frame-relay**
Step 3	Specify a subinterface.	**interface serial** *number.subinterface-number* **multipoint** [1]
Step 4	Define the mapping between a next hop protocol address and the DLCI used to connect to the address.	**frame-relay map bridge** *dlci* [**broadcast**] [**ietf**]
Step 5	Associate the subinterface with a bridge group.	**bridge-group** *bridge-group*[2]

1. This command is documented in the "Interface Commands" chapter in the *Configuration Fundamentals Command Reference*.
2. This command is documented in the *Bridging and IBM Networking Command Reference*.

Configure a Backup Interface for a Subinterface

Both point-to-point and multipoint Frame Relay subinterfaces can be configured with a backup interface. This approach allows individual PVCs to be backed up in case of failure rather than depending on the entire Frame Relay connection to fail before the backup takes over. You can configure a subinterface for backup on failure only, not for backup based on loading of the line.

If the serial interface has a backup interface, it will have precedence over the subinterface's backup interface in the case of complete loss of connectivity with the Frame Relay network. As a result, a subinterface backup is activated only if the serial interface is up, or if the serial interface is down and does not have a backup interface defined. If a subinterface has failed while its backup is in use, and then the serial interface goes down, the subinterface backup stays connected.

To configure a backup interface for a Frame Relay subinterface, perform the following tasks, beginning in global configuration mode:

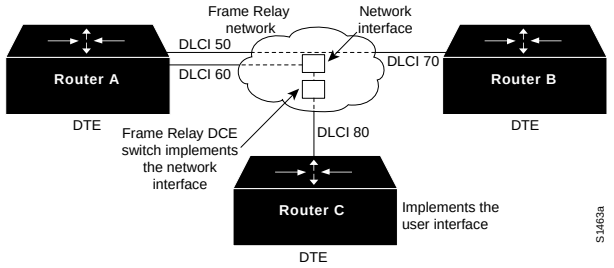
Task		Command
Step 1	Specify the interface.	**interface serial** *number*[1]
Step 2	Configure Frame Relay encapsulation.	**encapsulation frame-relay**
Step 3	Configure the subinterface.	**interface serial** *number.subinterface-number* **point-to-point**[1]
Step 4	Specify a DLCI for the subinterface.	**frame-relay interface-dlci** *dlci*
Step 5	Specify a backup interface for the subinterface.	**backup interface serial** *number*[1]
Step 6	Specify backup enable and disable delay.	**backup delay** *enable-delay disable-delay*[1]

1. This command is documented in the "Interface Commands" chapter in the *Configuration Fundamentals Command Reference*.

Configure Frame Relay Switching

Frame Relay switching is a means of switching packets based upon the DLCI, which can be looked upon as the Frame Relay equivalent of a MAC address. You perform the switching by configuring your router or access server as a Frame Relay network. There are two parts to a Frame Relay network: a Frame Relay DTE (the router or access server) and a Frame Relay DCE switch. Figure 30 illustrates this concept.

Figure 30 Frame Relay Switched Network



In Figure 30, Routers A, B, and C are Frame Relay DTEs connected to each other via a Frame Relay network. Our implementation of Frame Relay switching allows our devices to be used as depicted in this Frame Relay network.

Perform the tasks in the following sections, as necessary, to configure Frame Relay switching:

- Enable Frame Relay Switching
- Configure a Frame Relay DTE Device, DCE Switch, or NNI Support
- Specify the Static Route

These tasks are described in the following sections.

Enable Frame Relay Switching

You must enable packet switching before you can configure it on a Frame Relay DTE or DCE, or with Network-to-Network Interface (NNI) support. Do so by performing the following task in global configuration mode before configuring the switch type:

Task	Command
Enable Frame Relay switching.	**frame-relay switching**

For an example of how to enable Frame Relay switching, see the switching examples later in this chapter.

Configure a Frame Relay DTE Device, DCE Switch, or NNI Support

You can configure an interface as a DTE device or a DCE switch, or as a switch connected to a switch to support NNI connections. (DCE is the default.) To do so, perform the following task in interface configuration mode:

Task	Command
Configure a Frame Relay DTE device or DCE switch.	**frame-relay intf-type** [**dce** | **dte** | **nni**]

For an example of how to configure a DTE device or DCE switch, see the section "Hybrid DTE/DCE PVC Switching Example" later in this chapter.

For an example of how to configure NNI support, see the section "Pure Frame Relay DCE Example" later in this chapter.

Specify the Static Route

You must specify a static route for PVC switching. To do so, perform the following task in interface configuration mode:

Task	Command
Specify a static route for PVC switching.	**frame-relay route** *in-dlci out-interface out-dlci*

For an example of how to specify a static route, see the section "Pure Frame Relay DCE Example" later in this chapter.

Disable or Reenable Frame Relay Inverse ARP

Frame Relay Inverse ARP is a method of building dynamic address mappings in Frame Relay networks running AppleTalk, Banyan VINES, DECnet, IP, Novell IPX, and XNS. Inverse ARP allows the router or access server to discover the protocol address of a device associated with the virtual circuit.

Inverse ARP creates dynamic address mappings, as contrasted with the **frame-relay map** command, which defines static mappings between a specific protocol address and a specific DLCI (see the section "Configure Dynamic or Static Address Mapping" earlier in this chapter for more information).

Inverse ARP is enabled by default but can be disabled explicitly for a given protocol and DLCI pair. Disable or reenable Inverse ARP under the following conditions:

- Disable Inverse ARP for a selected protocol and DLCI pair when you know that the protocol is not supported on the other end of the connection.
- Reenable Inverse ARP for a protocol and DLCI pair if conditions or equipment change and the protocol is then supported on the other end of the connection.

Note If you change from a point-to-point subinterface to a multipoint subinterface, then change the subinterface number. Frame Relay Inverse ARP will be on by default, and no further action is required.

You do not need to enable or disable Inverse ARP if you have a point-to-point interface, because there is only a single destination and discovery is not required.

To select Inverse ARP or disable it, perform one of the following tasks in interface configuration mode:

Task	Command
Enable Frame Relay Inverse ARP for a specific protocol and DLCI pair, only if it was previously disabled.	**frame-relay inverse-arp** *protocol dlci*
Disable Frame Relay Inverse ARP for a specific protocol and DLCI pair.	**no frame relay inverse-arp** *protocol dlci*

Create a Broadcast Queue for an Interface

Very large Frame Relay networks might have performance problems when many DLCIs terminate in a single router or access server that must replicate routing updates and service advertising updates on each DLCI. The updates can consume access-link bandwidth and cause significant latency variations in user traffic; the updates can also consume interface buffers and lead to higher packet rate loss for both user data and routing updates.

To avoid such problems, you can create a special broadcast queue for an interface. The broadcast queue is managed independently of the normal interface queue, has its own buffers, and has a configurable size and service rate.

A broadcast queue is given a maximum transmission rate (throughput) limit measured in both bytes per second and packets per second. The queue is serviced to ensure that no more than this maximum is provided. The broadcast queue has priority when transmitting at a rate below the configured

maximum, and hence has a guaranteed minimum bandwidth allocation. The two transmission rate limits are intended to avoid flooding the interface with broadcasts. The actual transmission rate limit in any second is the first of the two rate limits that is reached.

To create a broadcast queue, complete the following task in interface configuration mode:

Task	Command
Create a broadcast queue for an interface.	**frame-relay broadcast-queue** *size byte-rate packet-rate*

Configure Payload Compression

You can configure payload compression on point-to-point or multipoint interfaces or subinterfaces. Payload compression uses the stac method to predict what the next character in the frame will be. Because the prediction is done packet-by-packet, the dictionary is not conserved across packet boundaries.

Payload compression on each virtual circuit consumes approximately 40 kilobytes for dictionary memory.

To configure payload compression on a specified multipoint interface or subinterface, complete the following task:

Task	Command
Enable payload compression on a multipoint interface.	**frame-relay map** *protocol protocol-address dlci* **payload-compress packet-by-packet**

To configure payload compression on a specified point-to-point interface or subinterface, complete the following task:

Task	Command
Enable payload compression on a point-to-point interface.	**frame-relay payload-compress packet-by-packet**

Configure TCP/IP Header Compression

TCP/IP header compression, as described by RFC 1144, is designed to improve the efficiency of bandwidth utilization over low-speed serial links. A typical TCP/IP packet includes a 40-byte datagram header. Once a connection is established, the header information is redundant and need not be repeated in every packet that is sent. Reconstructing a smaller header that identifies the connection and indicates the fields that changed and the amount of change reduces the number of bytes transmitted. The average compressed header is 10 bytes long.

For this algorithm to function, packets must arrive in order. If packets arrive out of order, the reconstruction will appear to create regular TCP/IP packets but the packets will not match the original. Because priority queuing changes the order in which packets are transmitted, enabling priority queueing on the interface is not recommended.

You can configure TCP/IP header compression in either of two ways, as described in the following sections:

- Configure an Individual IP Map for TCP/IP Header Compression
- Configure an Interface for TCP/IP Header Compression

The “Disable TCP/IP Header Compression” section describes how to disable this feature.

Note If you configure an interface with Cisco encapsulation and TCP/IP header compression, Frame Relay IP maps inherit the compression characteristics of the interface. However, if you configure the interface with IETF encapsulation, the interface cannot be configured for compression. Frame Relay maps will have to be configured individually to support TCP/IP header compression.

Configure an Individual IP Map for TCP/IP Header Compression

TCP/IP header compression requires Cisco encapsulation. If you need to have IETF encapsulation on an interface as a whole, you can still configure a specific IP map to use Cisco encapsulation and TCP header compression.

In addition, even if you configure the interface to perform TCP/IP header compression, you can still configure a specific IP map not to compress TCP/IP headers.

You can specify whether TCP/IP header compression is active or passive. Active compression subjects every outgoing packet to TCP/IP header compression. Passive compression subjects an outgoing TCP/IP packet to header compression only if the packet had a compressed TCP/IP header when it was received.

To configure an IP map to use Cisco encapsulation and TCP/IP header compression, perform the following task in interface configuration mode:

Task	Command
Configure an IP map to use Cisco encapsulation and TCP/IP header compression.	**frame-relay map ip** *ip-address dlci* [**broadcast**] **cisco tcp header-compression** {**active** | **passive**}

The default encapsulation is **cisco**.

Note An interface that is configured to support TCP/IP header compression cannot also support priority queuing or custom queuing.

For an example of how to configure TCP header compression on an IP map, see the “TCP/IP Header Compression Examples” section later in this chapter.

Configure an Interface for TCP/IP Header Compression

You can configure the interface with active or passive TCP/IP header compression. Active compression, the default, subjects all outgoing TCP/IP packets to header compression. Passive compression subjects an outgoing packet to header compression only if the packet had a compressed TCP/IP header when it was received on that interface.

To apply TCP/IP header compression to an interface, you must perform the following tasks in interface configuration mode:

Task	Command
Configure Cisco encapsulation on the interface.	**encapsulation frame-relay**
Enable TCP/IP header compression on the interface.	**frame-relay ip tcp header-compression** [**passive**]

Note If an interface configured with Cisco encapsulation is later configured with IETF encapsulation, all TCP/IP header compression characteristics are lost. To apply TCP/IP header compression over an interface configured with IETF encapsulation, you must configure individual IP maps, as described in the section "Configure an Individual IP Map for TCP/IP Header Compression."

For an example of how to configure TCP header compression on an interface, see the "TCP/IP Header Compression Examples" section later in this chapter.

Disable TCP/IP Header Compression

You can disable TCP/IP header compression by using either of two commands that have different effects, depending on whether Frame Relay IP maps have been explicitly configured for TCP/IP header compression or have inherited their compression characteristics from the interface.

Frame Relay IP maps that have explicitly configured TCP/IP header compression must also have TCP/IP header compression explicitly disabled.

To disable TCP/IP header compression, perform one of the following tasks in interface configuration mode:

Task	Command
Disable TCP/IP header compression on all Frame Relay IP maps that are not explicitly configured for TCP header compression.	**no frame-relay ip tcp header-compression**
or	
Disable TCP/IP header compression on a specified Frame Relay IP map.	**frame-relay map ip** *ip-address dlci* **nocompress tcp header-compression**

For examples of how to turn off TCP/IP header compression, see the section "Disabling Inherited TCP/IP Header Compression Example" and the section "Disabling Explicit TCP/IP Header Compression Example."

Configure Discard Eligibility

You can specify which Frame Relay packets have low priority or low time sensitivity and will be the first to be dropped when a Frame Relay switch is congested. The mechanism that allows a Frame Relay switch to identify such packets is the discard eligibility (DE) bit.

This feature requires that the Frame Relay network be able to interpret the DE bit. Some networks take no action when the DE bit is set. Other networks use the DE bit to determine which packets to discard. The most desirable interpretation is to use the DE bit to determine which packets should be dropped first and also which packets have lower time sensitivity.

You can define DE lists that identify the characteristics of packets to be eligible for discarding, and you can also specify DE groups to identify the DLCI that is affected.

To define a DE list specifying the packets that can be dropped when the Frame Relay switch is congested, perform the following task in global configuration mode:

Task	Command
Define a DE list.	**frame-relay de-list** *list-number* {**protocol** *protocol* | **interface** *type number*} *characteristic*

You can specify DE lists based on the protocol or the interface, and on characteristics such as fragmentation of the packet, a specific TCP or User Datagram Protocol (UDP) port, an access list number, or a packet size. See the **frame-relay de-list** command in the *Wide-Area Networking Command Reference* for arguments and other information.

To define a DE group specifying the DE list and DLCI affected, perform the following task in interface configuration mode:

Task	Command
Define a DE group.	**frame-relay de-group** *group-number dlci*

Configure DLCI Priority Levels

DLCI priority levels allow you to separate different types of traffic and can provide a traffic management tool for congestion problems caused by following situations:

- Mixing batch and interactive traffic over the same DLCI
- Traffic from sites with high-speed access being queued at destination sites with lower speed access.

Before you configure the DLCI priority levels, complete the following tasks:

- Define a global priority list.
- Enable Frame Relay encapsulation, as described earlier in this chapter.
- Define static or dynamic address mapping, as described earlier in this chapter.

  Make sure that you define each of the DLCIs to which you intend to apply levels. You can associate priority-level DLCIs with subinterfaces.
- Configure the LMI, as described earlier in this chapter.

Note DLCI priority levels provide a way to define multiple parallel DLCIs for different types of traffic. DLCI priority levels do not assign priority queues within the router or access server; in fact, they are independent of the device's priority queues. However, if you enable queuing and use the same DLCIs for queuing, then high-priority DLCIs can be put into high-priority queues.

To configure DLCI priority levels, perform the following task in interface configuration mode:

Task	Command
Enable multiple parallel DLCIs for different types of Frame Relay traffic, associate specified DLCIs with the same group, and define their levels.	**frame-relay priority-dlci-group** *group-number high-dlci medium-dlci normal-dlci low-dlci*

Note If you do not explicitly specify a DLCI for each of the priority levels, the last DLCI specified in the command line is used as the value of the remaining arguments. However, you must provide at least the high-priority and the medium-priority DLCIs.

Monitor the Frame Relay Connections

To monitor Frame Relay connections, perform any of the following tasks in EXEC mode:

Task	Command
Clear dynamically created Frame Relay maps, which are created by the use of Inverse ARP.	**clear frame-relay-inarp**
Display information about Frame Relay DLCIs and the LMI.	**show interfaces serial** *number*
Display LMI statistics.	**show frame-relay lmi** [*type number*]
Display the current Frame Relay map entries.	**show frame-relay map**
Display PVC statistics.	**show frame-relay pvc** [*type number* [*dlci*]]
Display configured static routes.	**show frame-relay route**
Display Frame Relay traffic statistics.	**show frame-relay traffic**
Display information about the status of LAPF.	**show frame-relay lapf**
Display all the SVCs under a specified map list.	**show frame-relay svc maplist**

Frame Relay Configuration Examples

This section provides examples of Frame Relay configurations. It includes the following sections:

- IETF Encapsulation Examples
- Static Address Mapping Examples
- SVC Configuration Examples
- Frame Relay Traffic Shaping Example
- Subinterface Examples
- Configuration Providing Backward Compatibility Example
- Booting from a Network Server over Frame Relay Example
- Frame Relay Switching Examples
- TCP/IP Header Compression Examples
- Disabling TCP/IP Header Compression Examples

IETF Encapsulation Examples

The first example that follows sets IETF encapsulation at the interface level. The second example sets IETF encapsulation on a per-DLCI basis. In the first example, the keyword **ietf** sets the default encapsulation method for all maps to IETF.

```
encapsulation frame-relay IETF
frame-relay map ip 131.108.123.2 48 broadcast
frame-relay map ip 131.108.123.3 49 broadcast
```

In the following example, IETF encapsulation is configured on a per-DLCI basis. This configuration has the same result as the configuration in the first example.

```
encapsulation frame-relay
frame-relay map ip 131.108.123.2 48 broadcast ietf
frame-relay map ip 131.108.123.3 49 broadcast ietf
```

Static Address Mapping Examples

The following sections provide examples of static address mapping for the IP, AppleTalk, DECnet, and IPX protocols.

Two Routers in Static Mode Example

The following example illustrates how to configure two routers for static mode.

Configuration for Router 1

```
interface serial 0
ip address 131.108.64.2 255.255.255.0
encapsulation frame-relay
keepalive 10
frame-relay map ip 131.108.64.1 43
```

Configuration for Router 2

```
interface serial 0
ip address 131.108.64.1 255.255.255.0
encapsulation frame-relay
keepalive 10
frame-relay map ip 131.108.64.2 43
```

AppleTalk Routing Example

The following example illustrates how to configure two routers to communicate with each other using AppleTalk over a Frame Relay network. Each router has a Frame Relay static address map for the other router. The use of the **appletalk cable-range** command indicates that this is extended AppleTalk (Phase II).

Configuration for Router 1

```
interface Serial0
ip address 172.21.59.24 255.255.255.0
encapsulation frame-relay
appletalk cable-range 10-20 18.47
appletalk zone eng
frame-relay map appletalk 18.225 100 broadcast
```

Configuration for Router 2

```
interface Serial2/3
ip address 172.21.177.18 255.255.255.0
encapsulation frame-relay
appletalk cable-range 10-20 18.225
appletalk zone eng
clockrate 2000000
frame-relay map appletalk 18.47 100 broadcast
```

DECnet Routing Example

The following example sends all DECnet packets destined for address 56.4 out on DLCI 101. In addition, any DECnet broadcasts for interface serial 1 will be sent on that DLCI.

```
decnet routing 32.6
!
interface serial 1
encapsulation frame-relay
frame-relay map decnet 56.4 101 broadcast
```

IPX Routing Example

The following example illustrates how to send packets destined for IPX address 200.0000.0c00.7b21 out on DLCI 102:

```
ipx routing 000.0c00.7b3b
!
interface ethernet 0
ipx network 2abc
!
interface serial 0
ipx network 200
encapsulation frame-relay
frame-relay map ipx 200.0000.0c00.7b21 102 broadcast
```

Subinterface Examples

The following sections provide basic Frame Relay subinterface examples and variations appropriate for different routed protocols and for bridging.

Basic Subinterface Examples

In the following example, subinterface 1 models a point-to-point subnet and subinterface 2 models a broadcast subnet. For emphasis, the **multipoint** keyword is used for serial subinterface 2, even though a subinterface is multipoint by default.

```
interface serial 0
encapsulation frame-relay
interface serial 0.1 point-to-point
ip address 10.0.1.1 255.255.255.0
frame-relay interface-dlci 42

interface serial 0.2 multipoint
ip address 10.0.2.1 255.255.255.0
frame-relay map 10.0.2.2 18
```


Frame Relay Multipoint Subinterface with Dynamic Addressing Example

The following example configures two multipoint subinterfaces for dynamic address resolution. Each subinterface is provided with an individual protocol address and subnet mask, and the **interface-dlci** command associates the subinterface with a specified DLCI. Addresses of remote destinations for each multipoint subinterface will be resolved dynamically.

```
interface Serial0
 no ip address
 encapsulation frame-relay
 frame-relay lmi-type ansi
!
interface Serial0.103 multipoint
 ip address 172.21.177.18 255.255.255.0
 frame-relay interface-dlci 300
!
interface Serial0.104 multipoint
 ip address 172.21.178.18 255.255.255.0
 frame-relay interface-dlci 400
```

IPX Routes over Frame Relay Subinterfaces Example

The following example configures a serial interface for Frame Relay encapsulation and sets up multiple IPX virtual networks corresponding to Frame Relay subinterfaces:

```
ipx routing 0000.0c02.5f4f
!
interface serial 0
encapsulation frame-relay
interface serial 0.1 multipoint
ipx network 1
frame-relay map ipx 1.000.0c07.d530 200 broadcast
ipx network 2
frame-relay map ipx 2.000.0c07.d530 300 broadcast
```

For subinterface serial 0.1, the router at the other end might be configured as follows:

```
ipx routing
interface serial 2 multipoint
ipx network 1
frame-relay map ipx 1.000.0c02.5f4f 200 broadcast
```

Unnumbered IP over a Point-to-Point Subinterface Example

The following example sets up unnumbered IP over subinterfaces at both ends of a point-to-point connection. In this example, Router A functions as the DTE, and Router B functions as the DCE. Routers A and B are both attached to Token Ring networks.

Configuration for Router A

```
frame-relay switching
!
interface token-ring 0
ip address 131.108.177.1 255.255.255.0
!
interface serial 0
no ip address
encapsulation frame-relay IETF
!
```

```
interface Serial0.2 point-to-point
ip unnumbered TokenRing0
ip pim sparse-mode
frame-relay interface-dlci 20
```

Configuration for Router B:

```
frame-relay switching
!
interface token-ring 0
ip address 131.108.178.1 255.255.255.0
!
interface serial 0
no ip address
encapsulation frame-relay IETF
bandwidth 384
clockrate 4000000
frame-relay intf-type dce
!
interface serial 0.2 point-to-point
ip unnumbered TokenRing1
ip pim sparse-mode

bandwidth 384
frame-relay interface-dlci 20
```

Transparent Bridging Using Subinterfaces Example

In the following example, Frame Relay DLCIs 42, 64, and 73 are to be used as separate point-to-point links with transparent bridging running over them. The bridging spanning tree algorithm views each PVC as a separate bridge port, and a frame arriving on the PVC can be relayed back out a separate PVC. Be sure that routing is not enabled when configuring transparent bridging using subinterfaces.

```
interface serial 0
encapsulation frame-relay
interface serial 0.1 point-to-point
bridge-group 1
frame-relay interface-dlci 42
interface serial 0.2 point-to-point
bridge-group 1
frame-relay interface-dlci 64
interface serial 0.3 point-to-point
bridge-group 1
frame-relay interface-dlci 73
```

SVC Configuration Examples

The following examples provide SVC configuration examples for interfaces and subinterfaces.

Interface Example

The following example configures a physical interface, applies a map-group to the physical interface, and then defines the map-group.

```
interface serial 0
ip address 172.10.8.6
encapsulation frame-relay
map-group bermuda
```



```
frame-relay lmi-type q933a
frame-relay svc

map-list bermuda source-addr E164 123456 dest-addr E164 654321
 ip 131.108.177.100 class hawaii
 appletalk 1000.2 class rainbow

map-class frame-relay rainbow
frame-relay idle-timer 60

map-class frame-relay hawaii
frame-relay cir in 64000
frame-relay cir out 64000
```

Subinterface Example

The following example configures a point-to-point interface for SVC operation. This example assumes that the main serial 0 interface has been configured for signalling, and that SVC operation has been enabled on the main interface.

```
int s 0.1 point-point
! Define the map-group; details are specified under the map-list holiday command.
map-group holiday

! Associate the map-group with a specific source and destination.
map-list holiday local-addr X121 <X121-addr> dest-addr E164 <E164-addr>
! Specify destination protocol addresses for a map-class.
 ip 131.108.177.100 class hawaii IETF
 appletalk 1000.2 class rainbow IETF broadcast

! Define a map class and its QOS settings.
map-class hawaii
 frame-relay cir in 2000000
 frame-relay cir out 56000
 frame-relay be 9000

! Define another map class and its QOS settings.
map-class rainbow
 frame-relay cir in 64000
 frame-relay idle-timer 2000
```

Frame Relay Traffic Shaping Example

The following comprehensive example illustrates a Frame Relay interface with three point-to-point subinterfaces.

In this example, the virtual circuits on subinterfaces Serial0.1 and Serial0.2 inherit class parameters from the main interface, namely those defined in *slow_vcs*, but the virtual circuit defined on subinterface Serial0.2 (DLCI 102) is specifically configured to use map class *fast_vcs*.

Map class *slow_vcs* uses a peak rate of 9600 and average rate of 4800 bps. Because BECN feedback is enabled by default, the output rate will be cut back as low as 4800bps in response to received BECNs. This map class is configured to use custom queuing using queue-list 1. In this example, queue-list 1 has 3 queues, with the first two being controlled by access lists 100 and 115.

Map class *fast_vcs* uses a peak rate of 64000 and average rate of 16000 bps. Because BECN feedback is enabled by default, the output rate will be cut back as low as 4800bps in response to received BECNs. This map class is configured to use priority-queuing using priority-group 2.

```
interface Serial0
 no ip address
 encapsulation frame-relay
 frame-relay lmi-type ansi
 frame-relay traffic-shaping
 frame-relay class slow_vcs
!
interface Serial0.1 point-to-point
 ip address 10.128.30.1 255.255.255.248
 ip ospf cost 200
 bandwidth 10
 frame-relay interface-dlci 101
!
interface Serial0.2 point-to-point
 ip address 10.128.30.9 255.255.255.248
 ip ospf cost 400
 bandwidth 10
 frame-relay interface-dlci 102
   class fast_vcs
!
interface Serial0.3 point-to-point
 ip address 10.128.30.17 255.255.255.248
 ip ospf cost 200
 bandwidth 10
 frame-relay interface-dlci 103
!
map-class frame-relay slow_vcs
 frame-relay traffic-rate 4800 9600
 frame-relay custom-queue-list 1
!
map-class frame-relay fast_vcs
 frame-relay traffic-rate 16000 64000
 frame-relay priority-group 2
!
access-list 100 permit tcp any any eq 2065
access-list 115 permit tcp any any eq 256
!
priority-list 2 protocol decnet high
priority-list 2 ip normal
priority-list 2 default medium
!
queue-list 1 protocol ip 1 list 100
queue-list 1 protocol ip 2 list 115
queue-list 1 default 3
queue-list 1 queue 1 byte-count 1600 limit 200
queue-list 1 queue 2 byte-count 600 limit 200
queue-list 1 queue 3 byte-count 500 limit 200
```

Configuration Providing Backward Compatibility Example

The following configuration provides backward compatibility and interoperability with earlier versions that are not compliant with RFC 1490. The **ietf** keyword is used to generate RFC 1490 traffic. This configuration is possible because of the flexibility provided by separately defining each map entry.

```
encapsulation frame-relay
frame-relay map ip 131.108.123.2 48 broadcast ietf
! interoperability is provided by IETF encapsulation
frame-relay map ip 131.108.123.3 49 broadcast ietf
frame-relay map ip 131.108.123.7 58 broadcast
! this line allows the router to connect with a
! device running an older version of software
frame-relay map decnet 21.7 49 broadcast
```


Configure IETF based on map entries and protocol for more flexibility. Use this method of configuration for backward compatibility and interoperability.

Booting from a Network Server over Frame Relay Example

When booting from a Trivial File Transfer Protocol (TFTP) server over Frame Relay, you cannot boot from a network server via a broadcast. You must boot from a specific TFTP host. Also, a **frame-relay map** command must exist for the host that you will boot from.

For example, if file *gs3-bfx* is to be booted from a host with IP address 131.108.126.2, the following commands would need to be in the configuration:

```
boot system gs3-bfx 131.108.126.2

interface Serial 0
encapsulation frame-relay
frame-relay map IP 131.108.126.2 100 broadcast
```

The **frame-relay map** command is used to map an IP address into a DLCI address. To boot over Frame Relay, you must explicitly give the address of the network server to boot from, and a **frame-relay map** entry must exist for that site. For example, if file *gs3-bfx.83-2.0* is to be booted from a host with IP address 131.108.126.111, the following commands must be in the configuration:

```
boot system gs3-bfx.83-2.0 131.108.13.111
!
interface Serial 1
ip address 131.108.126.200 255.255.255.0
encapsulation frame-relay
frame-relay map ip 131.108.126.111 100 broadcast
```

In this case, 100 is the DLCI that can get to host 131.108.126.111.

The remote router must have the following **frame-relay map** entry:

```
frame-relay map ip 131.108.126.200 101 broadcast
```

This entry allows the remote router to return a boot image (from the network server) to the router booting over Frame Relay. Here, 101 is a DLCI of the router being booted.

Frame Relay Switching Examples

The following sections provide several examples of configuring one or more routers as Frame Relay switches:

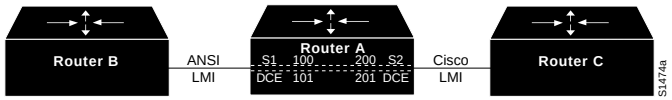
- PVC Switching Configuration Example—In this example, one router has two interfaces configured as DCEs; the router switches frames from the incoming interface to the outgoing interface on the basis of the DLCI alone.
- Pure Frame Relay DCE Example—In this example, a Frame Relay network is set up with two routers functioning as switches; standard NNI signaling is used between them.
- Hybrid DTE/DCE PVC Switching Example—In this example, one router is configured with both DCE and DTE interfaces (a hybrid DTE/DCE Frame Relay switch). It can switch frames between two DCE ports and between a DCE port and a DTE port.
- Switching over an IP Tunnel Example—In this example, two routers are configured to switch Frame Relay PVCs over a point-to-point IP tunnel.

PVC Switching Configuration Example

You can configure your router as a dedicated, DCE-only Frame Relay switch. Switching is based on DLCIs. The incoming DLCI is examined, and the outgoing interface and DLCI are determined. Switching takes place when the incoming DLCI in the packet is replaced by the outgoing DLCI, and the packet is sent out the outgoing interface.

In the following example, the router switches two PVCs between interface serial 1 and 2. Frames with DLCI 100 received on serial 1 will be transmitted with DLCI 200 on serial 2 (see Figure 31).

Figure 31 PVC Switching Configuration



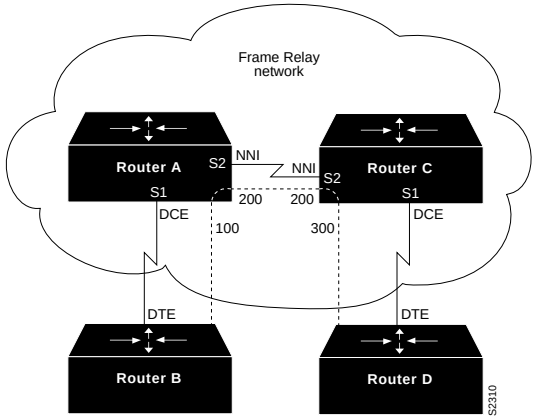
Configuration for Router A

```
frame-relay switching
!
interface Ethernet0
ip address 131.108.160.58 255.255.255.0
!
interface Serial1
no ip address
encapsulation frame-relay
keepalive 15
frame-relay lmi-type ansi
frame-relay intf-type dce
frame-relay route 100 interface Serial2 200
frame-relay route 101 interface Serial2 201
clockrate 2000000
!
interface Serial2
encapsulation frame-relay
keepalive 15
frame-relay intf-type dce
frame-relay route 200 interface Serial1 100
frame-relay route 201 interface Serial1 101
clockrate 64000
```

Pure Frame Relay DCE Example

Using the PVC switching feature, it is possible to build an entire Frame Relay network using our routers. In the following example, Router A and Router C act as Frame Relay switches implementing a two-node network. The standard Network-to-Network Interface (NNI) signaling protocol is used between Router A and Router C (see Figure 32).

Figure 32 Frame Relay DCE Configuration



Configuration for Router A

```
frame-relay switching
!
interface ethernet 0
no ip address
shutdown
!
interface ethernet 1
no ip address
shutdown
!
interface ethernet 2
no ip address
shutdown
!
interface ethernet 3
no ip address
shutdown
!
interface serial 0
ip address 131.108.178.48 255.255.255.0
shutdown
!
interface serial 1
no ip address
encapsulation frame-relay
frame-relay intf-type dce
frame-relay lmi-type ansi
frame-relay route 100 interface serial 2 200
!
interface serial 2
no ip address
encapsulation frame-relay
frame-relay intf-type nni
```

```
frame-relay lmi-type q933a
frame-relay route 200 interface serial 1 100
clockrate 2048000
!
interface serial 3
no ip address
shutdown
```

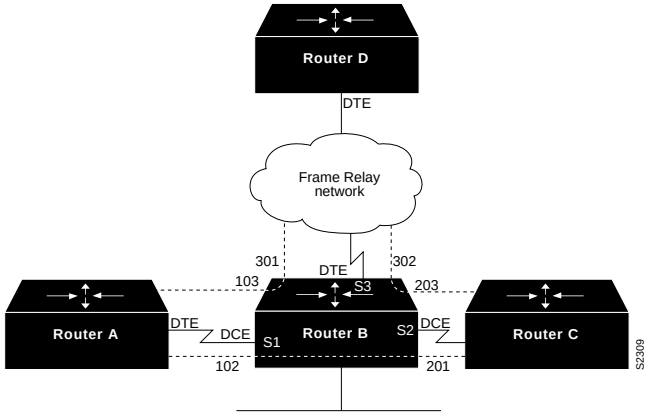
Configuration for Router C

```
frame-relay switching
!
interface ethernet 0
no ip address
shutdown
!
interface ethernet1
no ip address
shutdown
!
interface ethernet 2
no ip address
shutdown
!
interface ethernet 3
no ip address
shutdown
!
interface serial 0
ip address 131.108.187.84 255.255.255.0
shutdown
!
interface serial 1
no ip address
encapsulation frame-relay
frame-relay intf-type dce
frame-relay route 300 interface serial 2 200
!
interface serial 2
no ip address
encapsulation frame-relay
frame-relay intf-type nni
frame-relay lmi-type q933a
frame-relay route 200 interface serial 1 300
!
interface serial 3
no ip address
shutdown
```


Hybrid DTE/DCE PVC Switching Example

Routers can also be configured as hybrid DTE/DCE Frame Relay switches (see Figure 33).

Figure 33 Hybrid DTE/DCE PVC Switching



In the following example, Router B acts as a hybrid DTE/DCE Frame Relay switch. It can switch frames between the two DCE ports and between a DCE port and a DTE port. Traffic from the Frame Relay network can also be terminated locally. In the example, three PVCs are defined, as follows:

- Serial 1, DLCI 102 to serial 2, DLCI 201—DCE switching
- Serial 1, DLCI 103 to serial 3, DLCI 301—DCE/DTE switching
- Serial 2, DLCI 203 to serial 3, DLCI 302—DCE/DTE switching

DLCI 400 is also defined for locally terminated traffic.

Configuration for Router B

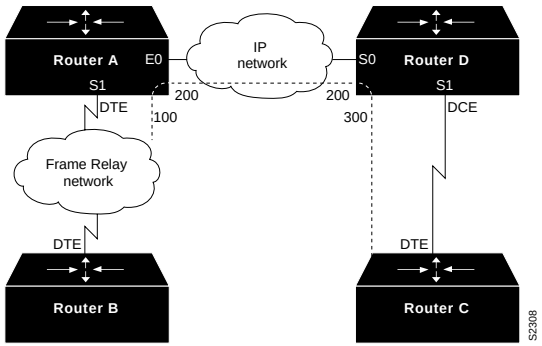
```
frame-relay switching
!
interface ethernet 0
ip address 131.108.123.231 255.255.255.0
!
interface ethernet 1
ip address 131.108.5.231 255.255.255.0
!
interface serial 0
no ip address
shutdown
!
interface serial 1
no ip address
encapsulation frame-relay
frame-relay intf-type dce
```

```
frame-relay route 102 interface serial 2 201
frame-relay route 103 interface serial 3 301
!
interface serial 2
no ip address
encapsulation frame-relay
frame-relay intf-type dce
frame-relay route 201 interface serial 1 102
frame-relay route 203 interface serial 3 302
!
interface serial 3
ip address 131.108.111.231
encapsulation frame-relay
frame-relay lmi-type ansi
frame-relay route 301 interface serial 1 103
frame-relay route 302 interface serial 1 203
frame-relay map ip 131.108.111.4 400 broadcast
```

Switching over an IP Tunnel Example

You can achieve switching over an IP tunnel by creating a point-to-point tunnel across the internetwork over which PVC switching can take place (see Figure 34).

Figure 34 Frame Relay Switch over IP Tunnel



The following configurations illustrate how to create the IP network depicted in Figure 34.

Configuration for Router A

```
frame-relay switching
!
interface Ethernet0
ip address 108.131.123.231 255.255.255.0
!
interface Ethernet1
ip address 131.108.5.231 255.255.255.0
!
```



```
interface Serial0
no ip address
shutdown
!
interface Serial1
ip address 131.108.222.231 255.255.255.0
encapsulation frame-relay
frame-relay map ip 131.108.222.4 400 broadcast
frame-relay route 100 interface Tunnel1 200
!
interface Tunnel1
tunnel source Ethernet0
tunnel destination 150.150.150.123
```

Configuration for Router D

```
frame-relay switching
!
interface Ethernet0
ip address 131.108.231.123 255.255.255.0
!
interface Ethernet1
ip address 131.108.6.123 255.255.255.0
!
interface Serial0
ip address 150.150.150.123 255.255.255.0
encapsulation ppp
!
interface Tunnel1
tunnel source Serial0
tunnel destination 108.131.123.231
!
interface Serial1
ip address 131.108.7.123 255.255.255.0
encapsulation frame-relay
frame-relay intf-type dce
frame-relay route 300 interface Tunnel1 200
```

TCP/IP Header Compression Examples

The following examples show various combinations of TCP/IP header compression and encapsulation characteristics on the interface and the effect on the inheritance of those characteristics on a Frame Relay IP map.

IP Map with Inherited TCP/IP Header Compression Example

The following example shows an interface configured for TCP/IP header compression and an IP map that inherits the compression characteristics. Note that the Frame Relay IP map is not explicitly configured for header compression.

```
interface serial 1
encapsulation frame-relay
ip address 131.108.177.178 255.255.255.0
frame-relay map ip 131.108.177.177 177 broadcast
frame-relay ip tcp header-compression passive
```

Use of the **show frame-relay map** command will display the resulting compression and encapsulation characteristics; the IP map has inherited passive TCP/IP header compression:

```
Router> show frame-relay map
Serial 1   (administratively down): ip 131.108.177.177
           dlci 177 (0xB1,0x2C10), static,
           broadcast,
           CISCO
           TCP/IP Header Compression (inherited), passive (inherited)
```

Using an IP Map to Override TCP/IP Header Compression Example

The following example shows the use of a Frame Relay IP map to override the compression set on the interface:

```
interface serial 1
encapsulation frame-relay
ip address 131.108.177.178 255.255.255.0
frame-relay map ip 131.108.177.177 177 broadcast nocompress
frame-relay ip tcp header-compression passive
```

Use of the **show frame-relay map** command will display the resulting compression and encapsulation characteristics; the IP map has not inherited TCP header compression:

```
Serial 1   (administratively down): ip 131.108.177.177
           dlci 177 (0xB1,0x2C10), static,
           broadcast,
           CISCO
```

Disabling TCP/IP Header Compression Examples

The following examples show the use of two different commands to disable TCP/IP header compression.

Disabling Inherited TCP/IP Header Compression Example

In this first example, the initial configuration is the following:

```
interface serial 1
encapsulation frame-relay
ip address 131.108.177.179 255.255.255.0
frame-relay ip tcp header-compression passive
frame-relay map ip 131.108.177.177 177 broadcast
frame-relay map ip 131.108.177.178 178 broadcast tcp header-compression
```

You enter the following commands:

```
serial interface 1
no frame-relay ip tcp header-compression
```

Use of the **show frame-relay map** command will display the resulting compression and encapsulation characteristics:

```
Router> show frame-relay map
Serial 1   (administratively down): ip 131.108.177.177 177
           dlci 177(0xB1, 0x2C10), static,
           broadcast
           CISCO
```



```
Serial 1   (administratively down): ip 131.108.177.178 178
           dlci 178(0xB2,0x2C20), static
           broadcast
           CISCO
           TCP/IP Header Compression (enabled)
```

As a result, header compression is disabled for the first map (with DLCI 177), which inherited its header compression characteristics from the interface. However, header compression is not disabled for the second map (DLCI 178), which is explicitly configured for header compression.

Disabling Explicit TCP/IP Header Compression Example

In this second example, the initial configuration is the same as the previous example, but you enter the following commands:

```
serial interface 1
no frame-relay ip tcp header-compression
frame-relay map ip 131.108.177.178 178 nocompress
```

Use of the **show frame-relay map** command will display the resulting compression and encapsulation characteristics:

```
Router> show frame-relay map
Serial 1   (administratively down): ip 131.108.177.177 177
           dlci 177(0xB1,0x2C10), static,
           broadcast
           CISCO
Serial 1   (administratively down): ip 131.108.177.178 178
           dlci 178(0xB2,0x2C20), static
           broadcast
           CISCO
```

The result of the commands is to disable header compression for the first map (with DLCI 177), which inherited its header compression characteristics from the interface, and also explicitly to disable header compression for the second map (with DLCI 178), which was explicitly configured for header compression.

Frame Relay Characteristics

Next Previous Contents

This section explains several Frame Relay characteristics of which you should be aware.

IP Split Horizon Checking

IP split horizon checking is disabled by default for Frame Relay encapsulation so routing updates will come in and out the same interface. The routers learn the data−link connection identifiers (DLCIs) they need to use from the Frame Relay switch via Local Management Interface (LMI) updates. The routers then use Inverse ARP for the remote IP address and create a mapping of local DLCIs and their associated remote IP addresses. Additionally, certain protocols such as AppleTalk, transparent bridging, and IPX cannot be supported on partially meshed networks because they require "split horizon," in which a packet received on an interface cannot be transmitted out the same interface, even if the packet is received and transmitted on different virtual circuits. Configuring Frame Relay subinterfaces ensures that a single physical interface is treated as multiple virtual interfaces. This capability allows us to overcome split horizon rules. Packets received on one virtual interface can now be forwarded out another virtual interface, even if they are configured on the same physical interface.

Ping Your Own IP Address on a Multipoint Frame Relay

You are not able to ping your own IP address on a multipoint Frame Relay interface. This is because Frame Relay multipoint (sub)interfaces are non−broadcast, (unlike Ethernet and point−to−point interfaces High−Level Data Link Control [HDLC]), and Frame Relay point−to−point sub−interfaces.

Furthermore, you are not able to ping from one spoke to another spoke in a hub and spoke configuration. This is because there is no mapping for your own IP address (and none were learned via Inverse ARP). But if you configure a static map (using the **frame−relay map** command) for your own IP address (or one for the remote spoke) to use the local DLCI, you can then ping your devices.

```
aton#ping  3.1.3.3
Type escape sequence to abort.
Sending 5, 100−byte ICMP Echos to 3.1.3.3, timeout is 2 seconds:
.....
Success rate is 0 percent (0/5)

aton#configure terminal
Enter configuration commands, one per line.  End with CNTL/Z.
aton(config)#interface serial 1
aton(config−if)#frame−relay map ip 3.1.3.3 160
aton(config−if)#

aton#show frame−relay map
Serial1 (up): ip 3.1.3.1 dlci 160(0xA0,0x2800), dynamic,
                broadcast,, status defined, active
Serial1 (up): ip 3.1.3.2 dlci 160(0xA0,0x2800), static,
                CISCO, status defined, active
```



```
Serial1 (up): ip 3.1.3.3 dlci 160(0xA0,0x2800), static,
                CISCO, status defined, active
aton#ping 3.1.3.3

Type escape sequence to abort.
Sending 5, 100−byte ICMP Echos to 3.1.3.3, timeout is 2 seconds:
!!!!!
Success rate is 100 percent (5/5), round−trip min/avg/max = 64/68/76 ms
aton#
aton#show running−config
!
interface Serial1
ip address 3.1.3.3 255.255.255.0
no ip directed−broadcast
encapsulation frame−relay
frame−relay map ip 3.1.3.2 160
frame−relay map ip 3.1.3.3 160
frame−relay interface−dlci 160
!
```

The Keyword "broadcast"

The broadcast keyword provides two functions: it forwards broadcasts when multicasting is not enabled, and it simplifies the configuration of Open Shortest Path First (OSPF) for non−broadcast networks that use Frame Relay.

The broadcast keyword might also be required for some routing protocols −− for example, AppleTalk −− that depend on regular routing table updates, especially when the router at the remote end is waiting for a routing update packet to arrive before adding the route.

By requiring selection of a designated router, OSPF treats a non−broadcast, multi−access network such as Frame Relay in much the same way as it treats a broadcast network. In previous releases, this required manual assignment in the OSPF configuration using the **neighbor interface router** command. When the **frame−relay map** command is included in the configuration with the broadcast keyword, and the **ip ospf network** command (with the broadcast keyword) is configured, there is no need to configure any neighbors manually. OSPF now automatically runs over the Frame Relay network as a broadcast network. (See the **ip ospf network interface** command for more detail.)

Note: The OSPF broadcast mechanism assumes that IP class D addresses are never used for regular traffic over Frame Relay.

Example

The following example maps the destination IP address 172.16.123.1 to DLCI 100:

```
interface serial 0
 frame−relay map IP 172.16.123.1 100 broadcast
```

OSPF uses DLCI 100 to broadcast updates.

Reconfiguring a Subinterface

Once you create a specific type of subinterface, you cannot change it without a reload. For example, you cannot create a multipoint subinterface serial0.2, then change it to point−to−point. To change it, you need to either reload the router or create another subinterface. This is the way the Frame Relay code works in Cisco

IOS® software.

DLCI Limitations

DLCI Address Space

Approximately 1000 DLCIs can be configured on a single physical link, given a 10−bit address. Because certain DLCIs are reserved (vendor−implementation−dependent), the maximum is about 1000. The range for a Cisco LMI is 16−1007. The stated range for ANSI/ITU is 16−992. These are the DLCIs carrying user−data.

However, when configuring Frame Relay VCs on subinterfaces, you need to consider a practical limit known as the IDB limit. The total number of interfaces and subinterfaces per system is limited by the number of interface descriptor blocks (IDBs) that your version of Cisco IOS supports. An IDB is a portion of memory that holds information about the interface such as counters, status of the interface, and so on. IOS maintains an IDB for each interface present on a platform and maintains an IDB for each subinterface. Higher speed interfaces require more memory than lower speed interfaces. Each platform contains different amounts of maximum IDBs and these limits may change with each Cisco IOS release.

For more information, see Maximum Number of Interfaces and Subinterfaces for Cisco IOS Software Platforms: IDB Limits.

LMI Status Update

The LMI protocol requires that all permanent virtual circuit (PVC) status reports fit into a single packet and generally limits the number of DLCIs to less than 800, depending on the maximum transmission unit (MTU) size.

The default MTU on serial interfaces is 1500 bytes, yielding a maximum of 296 DLCIs per interface. You can increase the MTU to support a larger full status update message from the Frame Relay switch. If the full status update message is larger than the interface MTU, the packet is dropped, and the interface giant counter is incremented. When changing the MTU, ensure the same value is configured at the remote router and intervening network devices.

Please note that these numbers vary slightly, depending on the LMI type. The maximum DLCIs per router (not interface) platform guideline, based on extrapolation from empirical data established on a Cisco 7000 router platform, are listed below:

- Cisco 2500: 1 X T1/E1 link @ 60 DLCIs per interface = 60 total

- Cisco 4000: 1 X T1/E1 link @ 120 DLCIs per interface = 120 total
- Cisco 4500: 3 X T1/E1 links @ 120 DLCIs per interface = 360 total
- Cisco 4700: 4 X T1/E1 links @ 120 DLCIs per interface = 480 total
- Cisco 7000: 4 X T1/E1/T3/E3 links @ 120 DLCIs per interface = 480 total
- Cisco 7200: 5 X T1/E1/T3/E3 links @ 120 DLCIs per interface = 600 total
- Cisco 7500: 6 X T1/E1/T3/E3 links @ 120 DLCIs per interface = 720 total

Note: These numbers are guidelines only, and assume that all traffic is fast−switched.

Other Considerations

A practical DLCI limit also depends on whether the VCs are running a dynamic or static routing protocol. Dynamic routing protocols, and other protocols like IPX SAP that exchange database tables, send hellos and forwarding information messages which must be seen and processed by the CPU. As a general rule, using static routes will allow you to configure a larger number of VCs on a single Frame Relay interface.

IP/IPX/AT Address

If you are using subinterfaces, don't put an IP, IPX or AT address on the main interface. Assign DLCIs to their subinterfaces before you enable the main interface to ensure that **frame−relay inverse−arp** works properly. In case it does malfunction, follow the steps below:

1. Turn off Inverse Address Resolution Protocol (ARP) for that DLCI by using the **no frame−relay inverse−arp ip 16** and the **clear frame−relay−inarp** commands.
2. Fix your configuration.
3. Turn the **frame−relay inverse−arp** command on again.

RIP and IGRP

Routing Information Protocol (RIP) updates flow every 30 seconds. Each RIP packet can contain up to 25 route entries, for a total of 536 bytes; 36 bytes of this total are header information, and each route entry is 20 bytes. Therefore, if you advertise 1000 routes over a Frame Relay link configured for 50 DLCIs, the result is 1 MB of routing update data every 30 seconds, or 285 kbps of bandwidth consumed. On a T1 link, this bandwidth represents 18.7 percent of the bandwidth, with each update duration being 5.6 seconds. This amount of overhead is considerable, and it is borderline acceptable, but the committed information rate (CIR) would have to be in the region of the access speed. Obviously, anything less than a T1 would incur too much overhead. For example:

- 1000/25 = 40 packets X 36 = 1440 header bytes
- 1000 X 20 bytes = 20,000 bytes of route entries
- Total 21,440 bytes X 50 DLCIs = 1072 MB of RIP updates every 30 seconds

- 1,072,000 bytes / 30 sec X 8 bits = 285 kbps

Interior Gateway Routing Protocol (IGRP) updates flow every 90 seconds (this interval is configurable). Each IGRP packet can contain 104 route entries, for a total of 1492 bytes, 38 of which are header information, and each route entry is 14 bytes. If you advertise 1000 routes over a Frame Relay link configured with 50 DLCIs, the request is approximately 720 KB of routing update data every 90 seconds, or 64 kbps of bandwidth consumed. On a T1 link, this bandwidth would represent 4.2 percent of the bandwidth, with each update duration being 3.7 seconds. This overhead is an acceptable amount:

- 1000/104 = 9 packets X 38 = 342 header bytes
- 1000 X 14 = 14,000 bytes of route entries
- Total = 14,342 bytes X 50 DLCIs = 717 KB of IGRP updates every 90 seconds
- 717,000 bytes / 90 X 8 bits = 63.7 kbps

Routing Table Maintenance Protocol (RTMP) routing updates occur every 10 seconds (this interval is configurable). Each RTMP packet can contain up to 94 extended route entries, for a total of 564 bytes, 23 bytes of header information, and each route entry is 6 bytes. If you advertise 1000 AppleTalk networks over a Frame Relay link configured for 50 DLCIs, the result is approximately 313 KB of RTMP updates every 10 seconds, or 250 kbps of bandwidth consumed. To remain within an acceptable level of overhead 15 percent or less), a T1 rate is required. For example:

- 1000/94 = 11 packets X 23 bytes = 253 header bytes
- 1000 X 6 = 6000 bytes of route entries
- Total = 6253 X 50 DLCIs = 313 KB of RTMP updates every 10 seconds
- 313,000 / 10 sec X 8 bits = 250 kbps

IPX RIP packet updates occur every 60 seconds (this interval is configurable). Each IPX RIP packet can contain up to 50 route entries for a total of 536 bytes, 38 bytes of header information, and each route entry is 8 bytes. If you advertise 1000 IPX routes over a Frame Relay link configured for 50 DLCIs, the result is 536 KB of IPX updates every 60 seconds, or 58.4 kbps of bandwidth consumed. To remain within an acceptable level of overhead (15 percent or less), a rate of 512 kbps is required. For example:

- 1000/50 = 20 packets X 38 bytes = 760 bytes of header
- 1000 X 8 = 8000 bytes of route entries
- Total = 8760 X 50 DLCIs = 438,000 bytes of IPX updates every 60 seconds
- 438,000 / 60 sec X 8 bits = 58.4 kbps

IPX service access point (SAP) packet updates occur every 60 seconds (this interval is configurable). Each IPX SAP packet can contain up to seven advertisement entries for a total of 536 bytes, 38 bytes of header information, and each advertisement entry is 64 bytes. If you broadcast 1000 IPX advertisements over a Frame Relay link configured for 50 DLCIs, you would end up with 536 KB of IPX updates every 60 seconds, or 58.4 kbps of bandwidth consumed. To remain within an acceptable level of overhead (15 percent or less), a rate of greater than 2 Mbps is required. Obviously, SAP filtering is required in this scenario. Compared to all

other protocols mentioned in this section, IPX SAP updates require the most bandwidth:

- 1000/7 = 143 packets X 38 bytes = 5434 bytes of header
- 1000 X 64 = 64,000 bytes of route entries
- Total = 69,434 X 50 DLCIs = 3,471,700 bytes of IPX service advertisements every 60 seconds
- 3,471,700 / 60 sec X 8 bits = 462 kbps

Keepalive

In some cases, the keepalive on the Cisco device needs to be set slightly shorter (about 8 seconds) than the keepalive on the switch. You'll see the need for this if the interface keeps coming up and down.

Serial Interfaces

Serial interfaces, which are by default multipoint, are non−broadcast media, while point−to−point subinterfaces are broadcast. If you are using static routes, you can point to either the next hop or the serial subinterface. For multipoint, you need to point to the next hop. This concept is very important when doing OSPF over Frame Relay. The router needs to know that this is a broadcast interface for OSPF to work.

OSPF and Multipoint

OSPF and multipoint can be very troublesome. OSPF needs a Designated Router (DR). If you start losing PVCs, some routers may lose connectivity and try to become a DR even though other routers still see the old DR. This causes the OSPF process to malfunction.

Overhead associated with OSPF is not as obvious and predictable as that with traditional distance vector routing protocols. The unpredictability comes from whether or not the OSPF network links are stable. If all adjacencies to a Frame Relay router are stable, only neighbor hello packets (keepalives) will flow, which is comparatively much less overhead than that incurred with a distance vector protocol (such as RIP and IGRP). If, however, routes (adjacencies) are unstable, link−state flooding will occur, and bandwidth can quickly be consumed. OSPF also is very processor−intensive when running the Dijkstra algorithm, which is used for computing routes.

In earlier releases of Cisco IOS software, special care had to be taken when configuring OSPF over multiaccess nonbroadcast medias such as Frame Relay, X.25, and ATM. The OSPF protocol considers these media like any other broadcast media such as Ethernet. Nonbroadcast multiaccess (NBMA) clouds are typically built in a hub and spoke topology. PVCs or switched virtual circuits (SVCs) are laid out in a partial mesh and the physical topology does not provide the multiaccess that OSPF believes is there. For the case of point−to−point serial interfaces, OSPF always forms an adjacency between the neighbors. OSPF adjacencies exchange database information. In order to minimize the amount of information exchanged on a particular segment, OSPF elects one router to be a DR, and one router to be a backup designated router (BDR) on each multiaccess segment. The BDR is elected as a backup mechanism in case the DR goes down.

The idea behind this setup is that routers have a central point of contact for information exchange. The selection of the DR became an issue because the DR and BDR needed to have full physical connectivity with all routers that exist on the cloud. Also, because of the lack of broadcast capabilities, the DR and BDR needed to have a static list of all other routers attached to the cloud. This setup is achieved using the **neighbor** command:

neighbor ip−address [priority number] [poll−interval seconds]

In later releases of Cisco IOS software, different methods can be used to avoid the complications of configuring static neighbors and having specific routers becoming DRs or BDRs on the nonbroadcast cloud. Which method to use is influenced by whether the network is new or an existing design that needs modification.

A subinterface is a logical way of defining an interface. The same physical interface can be split into multiple logical interfaces, with each subinterface being defined as point−to−point. This scenario was originally created in order to better handle issues caused by split horizon over NBMA and vector based routing protocols.

A point−to−point subinterface has the properties of any physical point−to−point interface. As far as OSPF is concerned, an adjacency is always formed over a point−to−point subinterface with no DR or BDR election. OSPF considers the cloud a set of point−to−point links rather than one multiaccess network. The only drawback for the point−to−point is that each segment belongs to a different subnet. This scenario might not be acceptable because some administrators have already assigned one IP subnet for the whole cloud. Another workaround is to use IP unnumbered interfaces on the cloud. This scenario also might be a problem for some administrators who manage the WAN based on IP addresses of the serial lines.

Next Previous Contents

Updated: Sep 02, 2002 Document ID: 15442