Implementing a Testbed for the Evaluation of FAST TCP in DOCSIS-based Access Networks

David Kennedy and Irena Atov
Centre for Advanced Internet Architectures. Technical Report 060119A
Swinburne University of Technology
Melbourne, Australia
{dkennedy,iatov}@swin.edu.au

***Abstract*-This technical report describes the process of implementing a testbed, which was used in [1] to experimentally evaluate the performance of FAST under typical 'edge of network' scenarios involving DOCSIS (Data Over Cable Service Interface Specification) cable system.**

Keywords- FAST TCP, testbed, DOCSIS cable network.

I. INTRODUCTION

Given the importance of controlling the congestion and stability of the Internet, there have been many TCP (Transmission Control Protocol) proposals aiming to improve the current standard version of TCP, known as NewReno [2], [3]. One such popular proposal that has received significant attention in recent years is FAST (Fast AQM Scalable TCP) [4], [5], which has been designed at Caltech (California Institute of Technology) to improve performance in high speed networks, especially those with long propagation delays.

IETF standardization and worldwide deployment requires that any new TCP variant must be tested and validated experimentally in real-world trials and in a wide variety of network environments. It is also crucial that independent groups repeat these tests. To date, FAST has been tested by Caltech and independent groups such as SLAC (Stanford Linear Accelerator Center) and CERN (The European Particle Physics Laboratory) in a wide range of high speed environments. However, there is a pressing need for testing in low speed environments which are more typical of the existing Internet. The current and medium term future of access networks is in the 1-10 Mbps range, using such technologies as xDSL, cable modems, and 802.11 wireless LANs. FAST needs to work in these environments, as well as being able to scale to the high speed regime.

The experimental study presented in [1] evaluated the performance of FAST in typical low-speed access network scenarios involving DOCSIS cable system [6]. This technical report explains our implementation of the FAST TCP testbed employed in [1]. The testbed uses real world DOCSIS equipment and is integrated with the existing Broadband Access Research Testbed at CAIA [7].

Section II describes the physical layout of the FAST TCP testbed. Detailed specification of its hardware and software components is provided in Section III and, finally, the configuration steps for each of the testbed components are outlined in Section IV.

II. TESTBED LAYOUT

The experimental work reported on in [1] required the setup of two different configurations of the FAST testbed in order to emulate two low-speed environments, respectively, each involving a typical access link. One contained a DOCSIS cable modem and the other was a simple rate-limited link. The setup containing the DOCSIS link is shown in Fig. 1, along with the interface and address information for each host/device; the setup for the simple link experiments is described in Section V.

The testbed consists of two end hosts: a sender (TCP server), which runs Linux with Caltech's FAST patches and a receiver, which runs standard Linux. In addition to the DOCSIS cable network – comprised of a cable modem (CM) and a cable modem termination system (CMTS) – a bridge running Dummynet [9] under FreeBSD and a standard Ethernet switch are used to emulate a typical ISP network. All the links in the network except for the bottleneck link have capacity 100 Mbps. The bottleneck DOCSIS link was configured with various bandwidths in the downstream (DS) and upstream (US) channels (in the range of 0.5-3Mbps) through adequate configuration of the CMTS, which governs transmission in the DOCSIS network. The buffering on the bottleneck link was set to the default value of the CMTS configuration (details of the CMTS channel capacity and buffer configuration are provided in Section IV.D). The Dummynet was configured to emulate a high-speed Wide Area Network (WAN) path without imposing any limitation on the downstream (DS) and usptream (US) channel capacity. Also, the Dummynet was configured with sufficiently large buffering in both DS and US in order to ensure that no packet loss occured in the core network (details of the Dummynet configuration are provided in Section IV.C).

For the experiments involving a simple low-speed link, the DOCSIS system was bypassed. Instead the same Dummynet that emulated the WAN delay was also

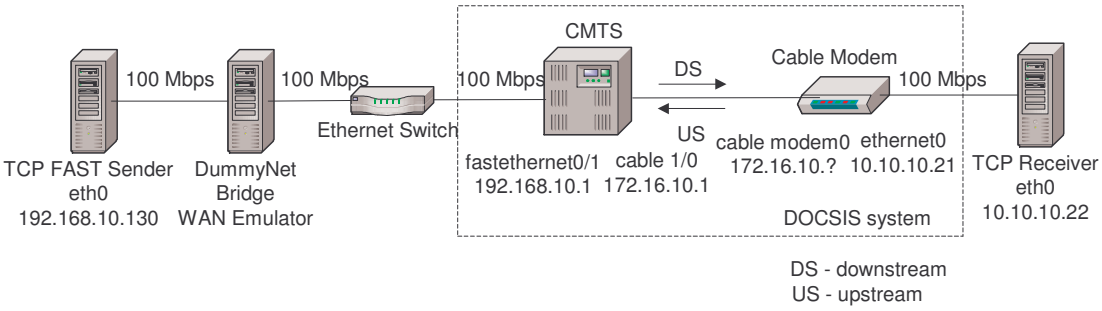

Figure 1 – FAST testbed layout with DOCSIS cable link

configured to emulate the bottleneck capacity limits in both the DS and US, and the limited buffering on the bottleneck link.

III. Testbed Hardware and Software

The systems used for the sender, receiver and the Dummynet bridge consisted of:

- Intel Celeron 2.4GHz Single Processor
- ASUS P4P800VM ATX Motherboard
- Intel 82801BA 10/100 Mbps on-board NIC
- 256MB PC3200 DDR RAM

These machines also had an extra NIC added to their PCI slots, which were either Intel 1000MT (82540em chipset) or Intel 100/Pro (82559 chipset).

The cable modem was a Cisco uBR900 series, which also acts as a router, the CMTS was a Cisco uBR7111 series router [10], and the switch was a Cisco Catalyst 3550 series. All Cisco devices ran Cisco's Internetworking Operating System (IOS). The CMTS and the switch ran version 12.1, and the cable modem ran version 12.2.

The TCP receiver was running Suse Linux 9.1 with kernel version 2.6.4, and used standard TCP. Since the latest version of FAST at the time of this experimental work was written for Linux kernel version 2.4.22, the FAST TCP sender was running Suse Linux 9.0, with the 2.4.22 version of the kernel. FAST TCP is available for download for research and educational use from [5]. The bridge was running FreeBSD 4.10 [11] with Dummynet [9], to emulate WAN paths of various delays.

The program that we used to generate TCP traffic and measure its throughput was `iperf` 1.7.0, which can be downloaded from [8]. We set up and ran different experiments from the TCP sender (on the downlink) by using an automatic script generator to start multiple `iperf` sessions to emulate multiple TCP connections.

IV. Configuration

A. FAST *TCP Sender*

Since the FAST patches were written for Linux kernel version 2.4.22, for the TCP sender we required a Linux distribution that ran on kernel version 2.4.x. To achieve this, we used Suse Linux 8.0, which came with kernel version 2.4.27, and we patched it with 2.4.22 kernel version.

The FAST TCP download consists of two patch files: an operating system independent file, and an operating system dependent file [5]. These files are called "os.ind.patch" and "os.dep.patch" respectively. It is required that the kernel is patched with both of these files. Also, FAST TCP must be enabled by executing the following command:

```
echo "1" > /proc/sys/net/ipv4/tcp_fast
```

Alpha tuning is a feature of FAST that dynamically sets the alpha value according to the network environment [5]. In our experiments, it was required that we have control of the alpha parameter, so alpha tuning was disabled by executing the following command:

```
echo "0" > /proc/sys/net/ipv4/tcp_fast_at
```

Burstiness control is a feature of FAST that spreads large packet transmission bursts over time so that the instantaneous transmission rate is smoother. This feature was enabled for our experiments by executing the following command:

```
echo "1" > /proc/sys/net/ipv4/tcp_fast_bc
```

The ideal TCP window size is approximately equal to the bandwidth-delay product. For some of the experiments, the window size required is higher than the default allowed by Linux. To be able to use the proper window sizes required for the experiments, the TCP buffers must be tuned. The five Linux configuration values tcp_mem, tcp_rmem, tcp_wmem, wmem_max and rmem_max must be modified to accommodate the higher window sizes. This is done by executing the following commands on both the sender and the receiver.

```
   echo "48128 48640 49152" >
/proc/sys/net/ipv4/tcp_mem

   echo 4096 33554432 33554432 >
/proc/sys/net/ipv4/tcp_rmem

   echo 4096 33554432 33554432 >
/proc/sys/net/ipv4/tcp_wmem

   echo "33554432" > /proc/sys/net/core/wmem_max

   echo "33554432" > /proc/sys/net/core/rmem_max
```

Both the sender and the receiver were configured with two interfaces, one for traffic flow within the testbed, and another to enable access to the Swinburne University network. Therefore, they were configured with a default route to the Swinburne gateway (IP address 136.186.229.1) and separate routes were set for within the testbed. To configure the sender with the these routes and IP addresses one needs to enter the following commands:

```
ifconfig eth1 136.186.229.130 netmask
255.255.255.0 broadcast 136.186.229.255

route add -net default 136.186.229.1 eth1

route add -net 10.10.10.20 gw 192.168.10.1 eth0

route add -net 172.16.10.0 gw 192.168.10.1 eth0
```

All of these commands must be executed again after a reboot of the machine. It would be possible to put these commands into a script such as /etc/rc.d/network so that they are executed automatically upon boot up.

B. TCP Receiver

On the receiver machine, it is necessary to tune the TCP buffers and set the IP addresses and routes, just as on the sender. The commands for the TCP buffer tuning are the same as on the sender:

```
echo "48128 48640 49152" >
/proc/sys/net/ipv4/tcp_mem

echo 4096 33554432 33554432 >
/proc/sys/net/ipv4/tcp_rmem

echo 4096 33554432 33554432 >
/proc/sys/net/ipv4/tcp_wmem

echo "33554432" > /proc/sys/net/core/wmem_max

echo "33554432" > /proc/sys/net/core/rmem_max
```

The commands for setting the IP address and routes will be slightly different. On the receiver, one needs to enter the following commands:

```
ifconfig eth1 136.186.229.129 netmask
255.255.255.0 broadcast 136.186.229.255

route add -net default 136.186.229.1 eth1

route add -net 192.168.10.0 gw 10.10.10.21 eth0

route add -net 172.16.10.0 gw 10.10.10.21 eth0
```

As with the sender, all of these commands must be executed after each reboot of the machine.

C. FreeBSD Dummynet Bridge

The Dummynet bridge should have the firewall rules and pipes set up to induce a 50ms delay each way. The maximum queue size of a Dummynet pipe is 1024Kbyte. We desired a larger queue size than this, to make sure that we had an ample amount of buffering, so we implemented a setup with two pipes connected in serial. By doing this, we achieved a larger amount of buffering than was otherwise possible with Dummynet. In order to have the data traverse both pipes, the sysctl IPFW variable "one pass" must be set to 0 by issuing the following command:

```
sysctl net.inet.ip.fw.one_pass=0.
```

The Dummynet pipes and IPFW rules were set up as follows:

```
ipfw add 1 pipe 1 ip from any to any layer2

ipfw add 2 pipe 2 ip from any to any layer2

ipfw add 65534 allow ip from any to any layer2

ipfw pipe 1 config delay 25ms queue 1024k

ipfw pipe 2 config delay 25ms queue 1024k
```

These IFPW commands may be put into a script in /usr/local/etc/rc.d so that they are executed upon boot up.

D. CMTS

The CMTS is where the upstream and downstream bandwidth, as well as the amount of buffering the CMTS itself provides is configured. The CMTS contains a file called a boot file. This boot file contains the maximum upstream and downstream bandwidth allowed for each cable modem. When a cable modem boots up and connects to the CMTS, the CMTS sends this boot file to the modem, telling it the maximum upstream and downstream bandwidth.

The CMTS is configured via telnet. This can be done from either the sender or the receiver, and since the CMTS has two IP addresses (192.168.10.1 and 172.16.10.1), either one can be used for this. The following commands show how this can be done:

```
telnet 192.168.10.1

Password:

BartCMTS>enable

Password:

BartCMTS# show running-configuration
```

The `show running-configuration` command will display the running configuration of the CMTS. The relevant information regarding the bandwidth and buffering can be found in here. For brevity, only the relevant lines are shown below. For complete running configurations of the CMTS and the cable modem, see Appendix 1.

```
…
cable config-file test2.cm
…
 service-class 1 max-upstream 512
 service-class 1 max-downstream 3000
…
!
ip dhcp pool BARTip
   bootfile test2.cm
…
interface Cable1/0
 cable downstream rate-limit token-bucket shaping
max-delay 512
```

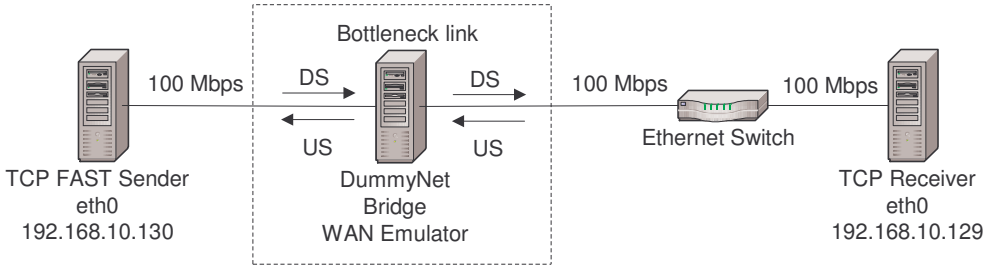

Figure 2 – FAST testbed layout with simple low-speed link

The line `cable config-file test2.cm` means that the following indented lines apply to the "test2.cm" boot file. These indented lines show that the file "test2.cm" contains an upstream bandwidth setting of 512kbps and a downstream bandwidth setting of 3000kbps. The following lines indicate that the CMTS is currently using "test2.cm" as its boot file, or the file that it sends to the cable modems when they boot up and connect to the CMTS. And finally, the last two lines displayed show that on the cable 1/0 interface, the buffer is set to hold packets for a maximum of 512ms.

The `configure-terminal` command will activate configuration mode, shown by the prompt `BartCMTS(config)#`. Once in this mode, commands such as those seen above can be entered. To alter settings that are indented in the running-config, the first non-indented command before them must be entered first. The prompt will then change accordingly.

For example, to configure the buffering capacity to 512ms, the `interface Cable1/0` command must be entered before the `cable downstream rate-limit token-bucket shaping max-delay 512` command.

To configure the bandwidth settings, it is necessary to make sure that in the running-configuration, it stated that the bootfile in use was test2.cm. If not, it must be changed by entering these commands at the prompt:

```
BartCMTS(config)#ip dhcp pool BARTip

BartCMTS(dhcp-config)#bootfile test2.cm
```

For another example, to change the upstream bandwidth to 128kbps and the downstream bandwidth to 1536kbps, the following commands would be issued:

```
BartCMTS(config)# cable config-file test2.cm

BartCMTS(config-file)# service-class 1 max-upstream 128

 BartCMTS(config-file)# service-class 1 max-downstream 1536
```

If the bandwidth settings such as in this case, or anything to do with the boot file were modified, the cable modem must be rebooted. This is done via the Cisco terminal server, which can be accessed via telnet from any computer in the lab (not just the sender and receiver).

To connect to the terminal server and reload the cable modem, the following commands must be issued:

```
telnet 136.186.229.89

CommServer2610>connect bartCableModem6

Password:

BartCM6>en

BartCM6#reload
```

Once the cable modem has reloaded, the DOCSIS system is now ready to be used again.

There were a number of housekeeping protocols running on the BART network that produce unnecessary traffic. Since our experiments were based on realistic home-user conditions, it was required to have these (unnecessary) protocols disabled. Routing Information Protocol (RIP) ran on the CMTS and the cable modem, and was disabled on these devices. Cisco Discovery Protocol (CDP) ran on all three Cisco devices and needed to be disabled on each of them.

We disabled CDP on the CMTS by simply entering the `no cdp run` command in configuration mode.

RIP was disabled by typing `no router rip` in configuration mode. Doing this meant that the CMTS required static routes. We set these up by typing `ip route 10.10.10.0 255.255.255.0 Cable1/0 permanent` in configuration mode.

E. Cable Modem

The cable modem we used was a Cisco, just like the CMTS, so the interface is the same. The configuration of our cable modem is included in Appendix 1. As in the CMTS case, we disabled CDP on the cable modem by entering the `no cdp run` command.

RIP was also disabled on the cable modem. This was done by entering the `no router rip` command. The cable modem does not require any static routes to be configured as a result of this.

F. Switch

The Cisco switch that we used also ran CDP, but on top of that, it also ran Spanning Tree Protocol (STP), Virtual Trunking Protocol (VTP), and Dynamic Trunking Protocol (DTP). CDP was disabled on this device in the same way as on the CMTS and the cable modem by using the `no cdp run` command.

The switch was configured with a separate VLAN specifically for FAST, so that other traffic from

Swinburne University network would not interfere. The relevant lines from the configuration are shown below:

```
...
vlan 123
 name TCPFAST
...
no spanning-tree vlan 123
...
interface FastEthernet0/<port-number (1-24)>
 switchport access vlan 123
 switchport mode access
 switchport nonegotiate
 no ip address
 no keepalive
 no cdp enable
...
interface Vlan123
 no ip address
...
```

STP runs on a per-VLAN basis. To disable it for the FAST VLAN, in configuration mode in the switch, we typed `no spanning-tree vlan <vlan ID>`. Each port that is being used for the FAST testbed was assigned to the FAST VLAN by issuing the `switchport access vlan 123` command for each port. For each of these ports, all the attributes displayed above were applied also. These ensured that DTP was also disabled, and that keepalives and CDP messages were not being sent on these ports.

VTP was also disabled on the switch. This was done by entering the `vtp mode transparent` command in configuration mode.

V. BYPASSING DOCSIS

In order to evaluate the impact of DOCSIS on FAST, we created an equal rate system to the one we had, but without running DOCSIS. To achieve this, we ran the same experiments over a simple, rate limited link by bypassing the DOCSIS system. This was done by configuring the Dummynet bridge with the same capacity and buffering as the CMTS. A number of configuration changes are required in order to do this.

First of all, the receiver must be unplugged from the cable modem and plugged straight into the same switch that the bridge is connected to. Since there is no routing done between the sender and receiver in this configuration, the receiver must also change its IP address to be on the 192.168.10.0 subnet. This is done by using the following command on the receiver:

```
ifconfig eth0 192.168.10.129 netmask
255.255.255.0 broadcast 192.168.10.255
```

To swap it back to the DOCSIS system, simply change the IP address to what it was and add the routes again by issuing the following commands on the receiver:

```
ifconfig eth0 10.10.10.21 netmask 255.255.255.252
broadcast 10.10.10.23

route add -net default gw 10.10.10.22

route add -net 192.168.10.0 gw 10.10.10.21 eth0

route add -net 172.16.10.0 gw 10.10.10.21 eth0
```

When bypassing the DOCSIS system, it is also necessary to modify the traffic shaping settings on the bridge to match the settings of the DOCSIS system. This is done by issuing the following commands on the Dummynet bridge:

```
ipfw flush

ipfw add 1 pipe 1 ip from 192.168.10.130 to
192.168.10.129 layer2

ipfw add 2 pipe 2 ip from 192.168.10.129 to
192.168.10.130 layer2

ipfw add 65534 allow mac-type 2054 layer2

ipfw pipe 1 config bw <downstream bandwidth>
delay 50ms queue <formula for queue size>k

ipfw pipe 2 config bw <upstream bandwidth> delay
50ms queue 1024k

sysctl net.inet.ip.fw.one_pass=1
```

REFERENCES

[1] L. H. Andrew, I. Atov, D. Kennedy, B. Wydrowski, Evaluation of FAST TCP in Low-Speed DOCSIS-based Access Networks, *in Proc. of IEEE Tencon'05*, pp. 1680-1685, Melbourne, Australia, November 2005.

[2] M. Allman, V. Paxson, and W. Stevens, *TCP Congestion Control*, IETF RFC 2581, April 1999. Available at http://ietf.org/rfc/rfc2581.txt.

[3] S. Floyd and T. Henderson, *The NewReno Modification to TCP's Fast Recovery Algorithm*, IETF RFC 2582, April 1999, Available at http://ietf.org/rfc/rfc2582.txt.

[4] C. Jin, D. X. Wei, and S. H. Low, FAST TCP: Motivation, Architecture, Algorithms, Performance, *in Proc. of IEEE INFOCOM 2004*, pp. 2490-2501, Hong Kong, March 2004.

[5] FAST, http://netlab.caltech.edu/FAST/index.html.

[6] CableLabs, Data-Over-Cable Service Interface Specifications Radio Frequency Interface Specification SP –RFIv1.1-101-990311,1999.

[7] Broadband Access Research Testbed, Centre for Advanced Internet Architectures, Swinburne University of Technology, http://caia.swin.edu.au/bart.

[8] A. Tirumala, F. Qin, J. Dugan, J. Ferguson, K. Gibbs, "Iperf", [Online]. Available: http://dast.nlanr.net/Projects/Iperf.

[9] L. Rizzo, "Dummynet", [Online]. Available: http://info.iet.unipi.it/~luigi/ip_dummynet/.

[10] Cisco uBR7100 Series Software Configuration giude. [Online]. Available: http://www.cisco.com/en/US/products/hw/cable/ps2211/products_configuration_guide_chapter09186a00801b355a.html#wp1021916.

[11] The FreeBSD Project, http://www.freebsd.org

APPENDIX I

Running configuration for the cable modem:

```
bartCM6#show running-configuration
Building configuration...

Current configuration : 1013 bytes
!
! NVRAM config last updated at 15:37:11 - Sat Nov
16 2002
!
version 12.2
no parser cache
no service pad
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
service internal
!
hostname BartCableModem1
!
enable secret 5 $1$V4lI$nsRplYMFdD4QavCpqShNG1
enable password bart
!
clock timezone - 0
ip subnet-zero
ip tftp source-interface cable-modem0
no ip domain-lookup
!
ip audit notify log
ip audit po max-events 100
!
interface Ethernet0
 ip address 10.10.10.21 255.255.255.252
 no ip mroute-cache
 no cdp enable
!
interface cable-modem0
 no ip mroute-cache
 no cable-modem compliant bridge
 cable-modem boot admin 2
 cable-modem boot oper 5
!
router rip
 version 2
 redistribute connected
 network 172.16.0.0
 no auto-summary
!
ip classless
ip pim bidir-enable
no ip http server
no ip http cable-monitor
!
no cdp run
snmp-server manager
call rsvp-sync
!
!
mgcp profile default
!
!
line con 0
line vty 0 4
 login
!
scheduler max-task-time 5000
end
```

Running configuration for the CMTS:

```
BartCMTS#show running-configuration
Building configuration...

Current configuration : 2651 bytes
!
version 12.1
no service single-slot-reload-enable
service timestamps debug uptime
service timestamps log uptime
no service password-encryption
service udp-small-servers max-servers no-limit
!
hostname BartCMTS
!
enable secret 5 $1$VNoT$7CbYtv0URbYYoXLz15I1l1
enable password bart
```

```
!
no cable qos permission create
no cable qos permission update
cable qos permission modems
cable time-server
!
cable config-file test.cm
 service-class 1 priority 1
 service-class 1 max-upstream 3000
 service-class 1 max-downstream 10000
 service-class 1 max-burst 1600
 cpe max 99
 timestamp
!
cable config-file test2.cm
 service-class 1 priority 1
 service-class 1 max-upstream 512
 service-class 1 max-downstream 3000
 service-class 1 max-burst 1600
 cpe max 99
 timestamp
!
ip subnet-zero
no ip domain-lookup
no ip dhcp conflict logging
ip dhcp excluded-address 172.16.10.1
!
ip dhcp pool BARTip
   network 172.16.10.0 255.255.255.0
   bootfile test2.cm
   next-server 172.16.10.1
   option 4 ip 172.16.10.1
   option 2 hex 0000.0000
   option 7 ip 172.16.10.1
   default-router 172.16.10.1
   dns-server 136.186.1.111
   lease 0 7 30
!
ip ssh time-out 120
ip ssh authentication-retries 3
!
!
!
!
!
!
interface FastEthernet0/0
 ip address 136.186.229.80 255.255.255.0
 ip nat outside
 no keepalive
 duplex auto
 speed 100
 no cdp enable
!
interface FastEthernet0/1
 ip address 192.168.10.1 255.255.255.0
 ip nat inside
 no keepalive
 duplex auto
 speed auto
 no cdp enable
!
interface Cable1/0
 ip address 172.16.10.1 255.255.255.0
 ip nat inside
 cable downstream rate-limit token-bucket shaping
max-delay 1024
 cable downstream annex B
 cable downstream modulation 64qam
 cable downstream interleave-depth 32
 cable downstream frequency 550000000
 cable downstream channel-id 0
 no cable downstream rf-shutdown
 cable upstream 0 frequency 38000000
 cable upstream 0 power-level 0
 cable upstream 0 channel-width 200000
 cable upstream 0 minislot-size 32
 no cable upstream 0 concatenation
 no cable upstream 0 shutdown
!
ip default-gateway 136.186.229.1
ip nat inside source static 10.10.10.10
136.186.229.222
ip nat inside source static 10.10.10.6
136.186.229.221
ip nat inside source static 192.168.10.2
136.186.229.83
ip nat inside source static 10.10.10.2
136.186.229.220
```

```
   ip nat inside source static 192.168.10.1
136.186.229.82
   ip nat inside source static 10.10.10.30
136.186.229.224
   ip classless
   ip route 10.10.10.0 255.255.255.0 Cable1/0
permanent
   no ip http server
   !
   no cdp run
   snmp-server community caiapublic RO
   banner motd ^CCWelcome to Broadband Access
Research Testbed (BART) ^C
   !
   line con 0
   line aux 0
   line vty 0 4
    password bart
    login
   !
   end
```