RPort Final Assignment

COMP 8005

By Ramzi Chennafi

Table of Contents

About the Program

RPort is a simple, configurable port forwarder written in node.js.

To run the program, install node - instructions on installation can be found here.

https://github.com/joyent/node/wiki/Installing-Node.js-via-package-manager#debian-and-ubuntu-based-linux-distributions

On Fedora just execute the following commands as root.

```
yum install -y nodejs
```

Executing is done by typing this within the main directory.

```
Node main.js
```

Setting up the Rules

Rules are placed in the config file in json format as follows:

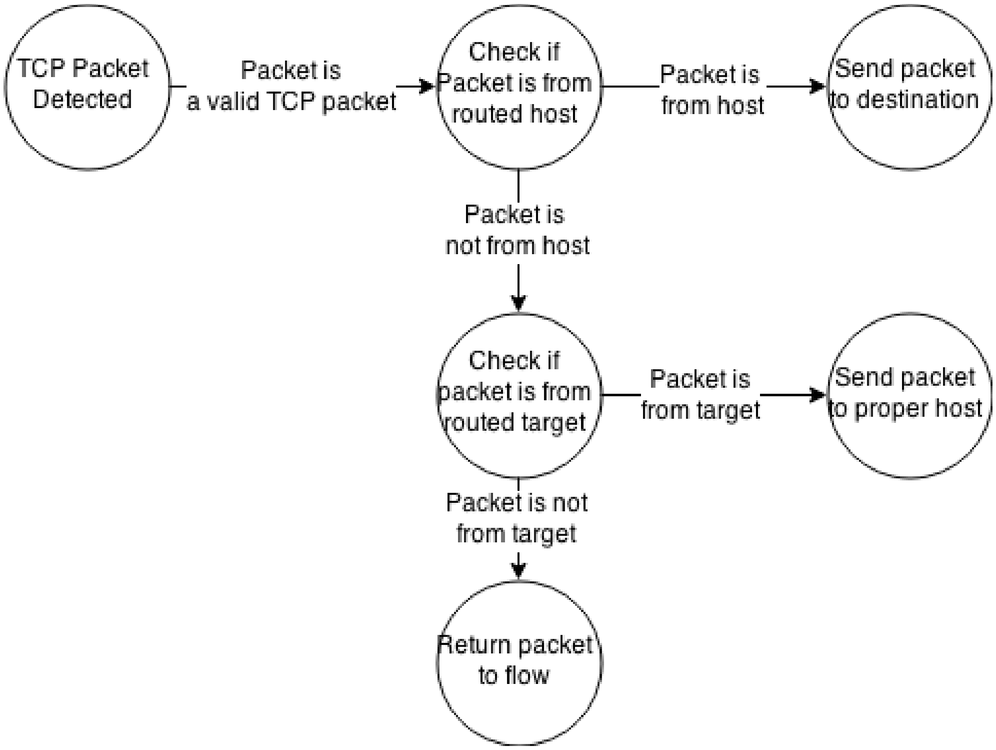
```
[
    {"fromPort":"7002","toPort":"88","toAddress":"localhost"}
]
```

Seperate each new entry with a comma like so

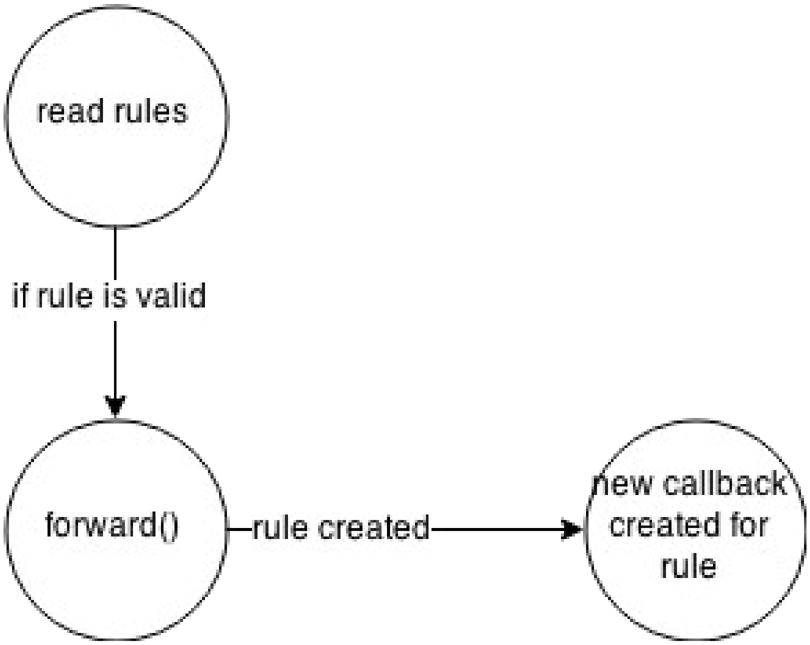
```
[
    {"fromPort":"7002","toPort":"88","toAddress":"localhost"},
    {"fromPort":"7002","toPort":"88","toAddress":"localhost"}
]
```

There is no limit on rules, but there may be a performance hit if the amount of rules get excessive.

Design Diagrams


TCP Packet Detected
Packet is a valid TCP packet
Check if Packet is from routed host
Packet is from host
Send packet to destination
Packet is not from host
Check if packet is from routed target
Packet is from target
Send packet to proper host
Packet is not from target
Return packet to flow

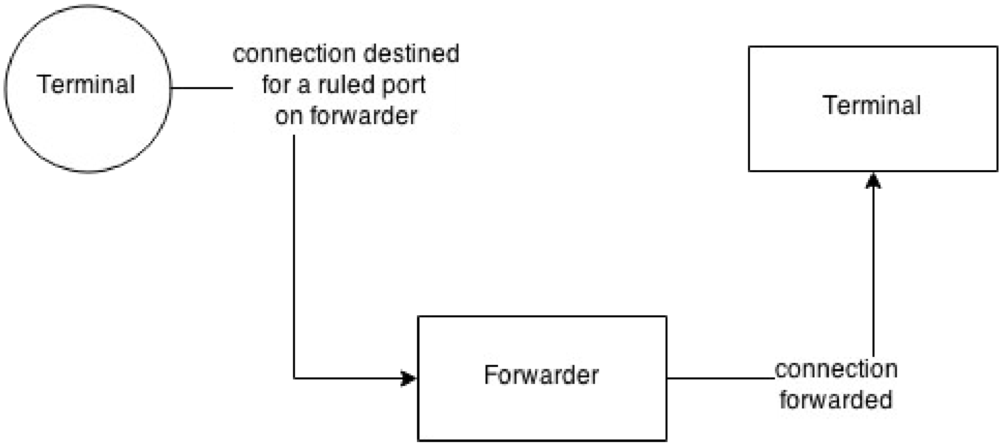
An overview of the program from a more theoretical perspective.



A look at the application programmatically.

Testing Setup

I used both 3 terminals, one terminal hosted the port forwarding program. While another machine tried to connect to a service on the port forwarder, the port forwarder would then forward that connection to the proper destination based on the rule.



PseudoCode

Forward(sourcePort, destinationPort, destinationIP)

```
Create a new network server
create a destination connection to destination port and destination ip
create a listen socket on the sourcePort
create callback
    if data comes in on listen socket
    write to destination connection socket
```

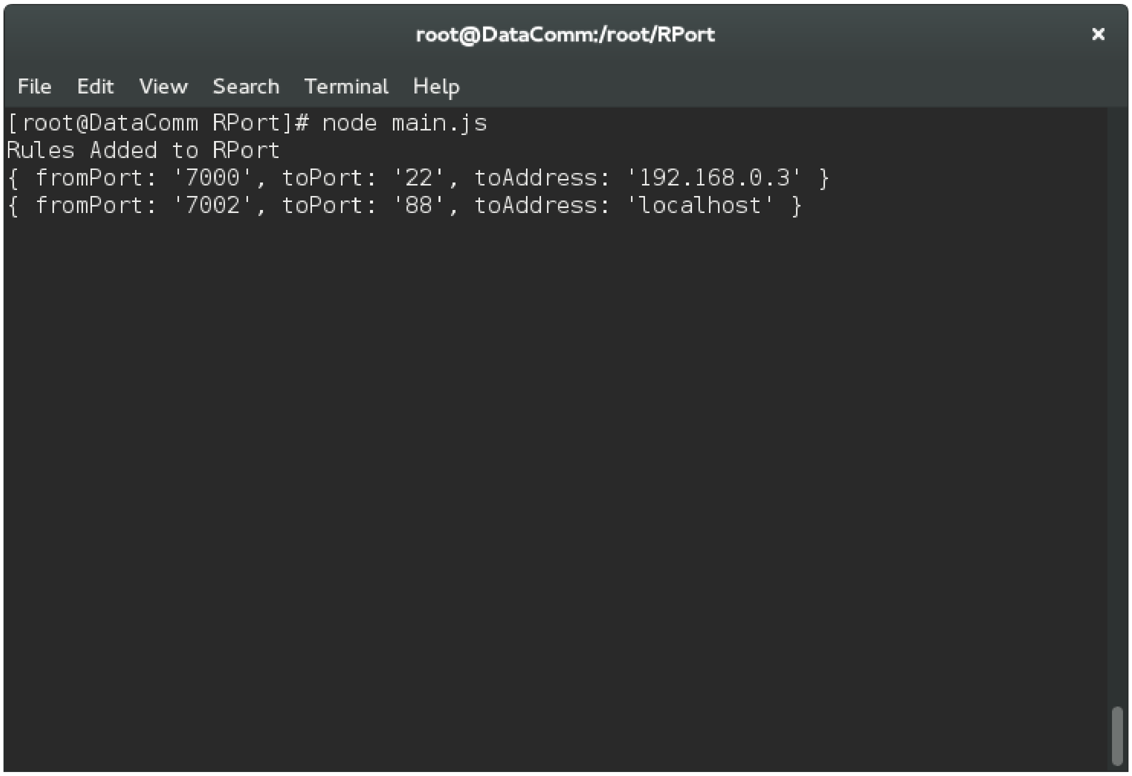
main body

```
Read in config file as json
for each entry in json file
    forward(file.toPort, file.fromPort, file.toAddress)
```

Testing

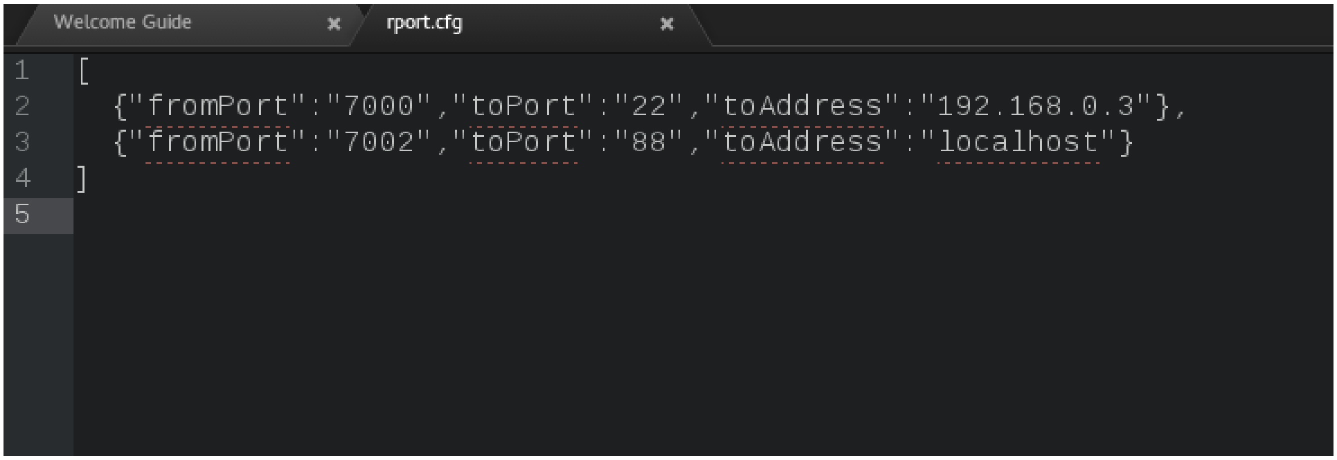
Test Number	Test Name	Test Description	Tools Used	Pass/Fail?
1	Rule Configurability	A new rule can be placed within the rules config file and followed.	program	Pass
2	Forwarding Packets	A message sent from a host to be routed is sent properly.	Wireshark, ssh program	Pass
3	Receiving Data	A message that was routed results in a response back to the original host.	Wireshark, netcat program	Pass
4	Packet Data	The packet data matches what it should be.	Wireshark, program	Pass
5	Handshaking	A TCP handshake can on a forwarded operation.	Wireshark, program	Pass

Test 1 – Rule Configurability


root@DataComm:/root/RPort
File Edit View Search Terminal Help

```
[root@DataComm RPort]# node main.js
Rules Added to RPort
{ fromPort: '7000', toPort: '22', toAddress: '192.168.0.3' }
{ fromPort: '7002', toPort: '88', toAddress: 'localhost' }
```

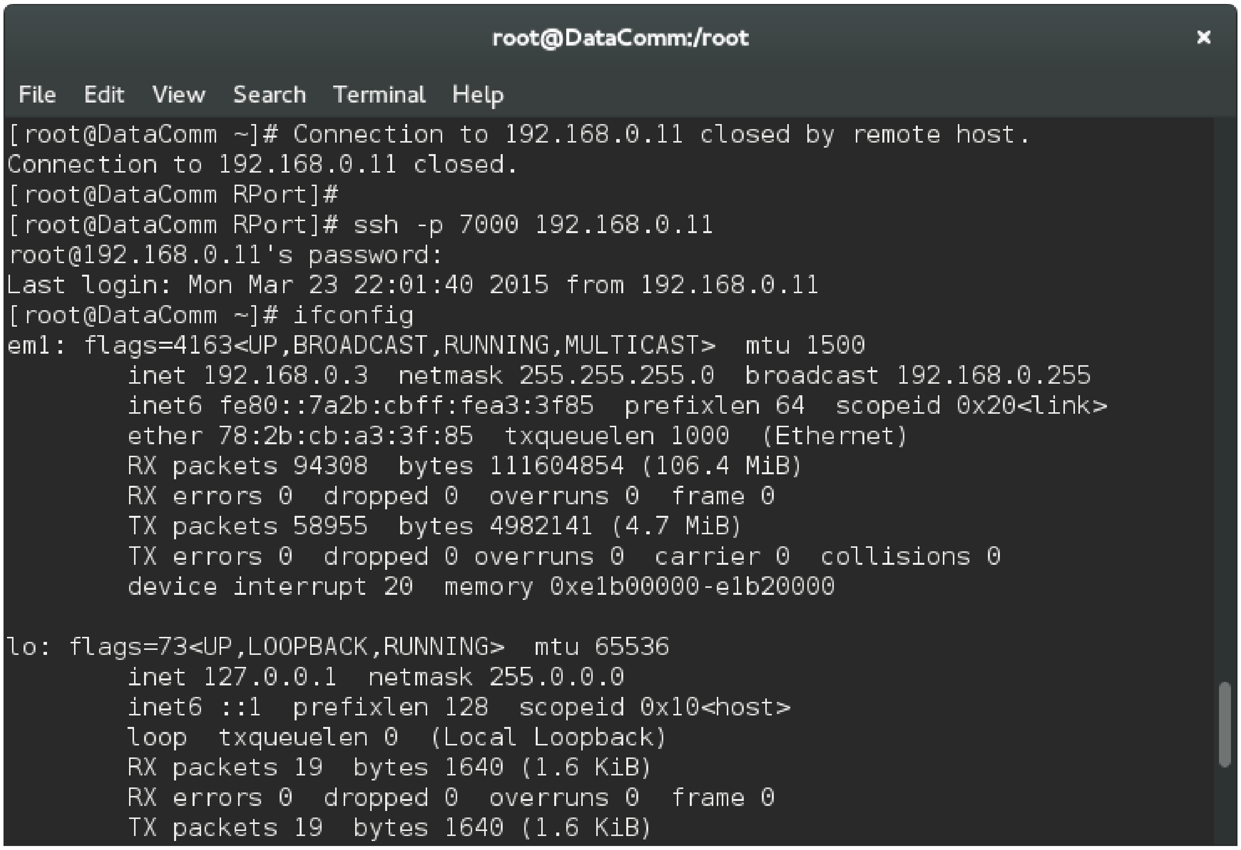
Above is the rules as demonstrated by the program. If we take a look at the config file we can see that these same rules exist here. This test is a success, for proof of these rules in action refer to the next few tests where you can see them forwarding.



```
[
  {"fromPort":"7000","toPort":"22","toAddress":"192.168.0.3"},
  {"fromPort":"7002","toPort":"88","toAddress":"localhost"}
]
```

Test 2 – Forwarding Packets

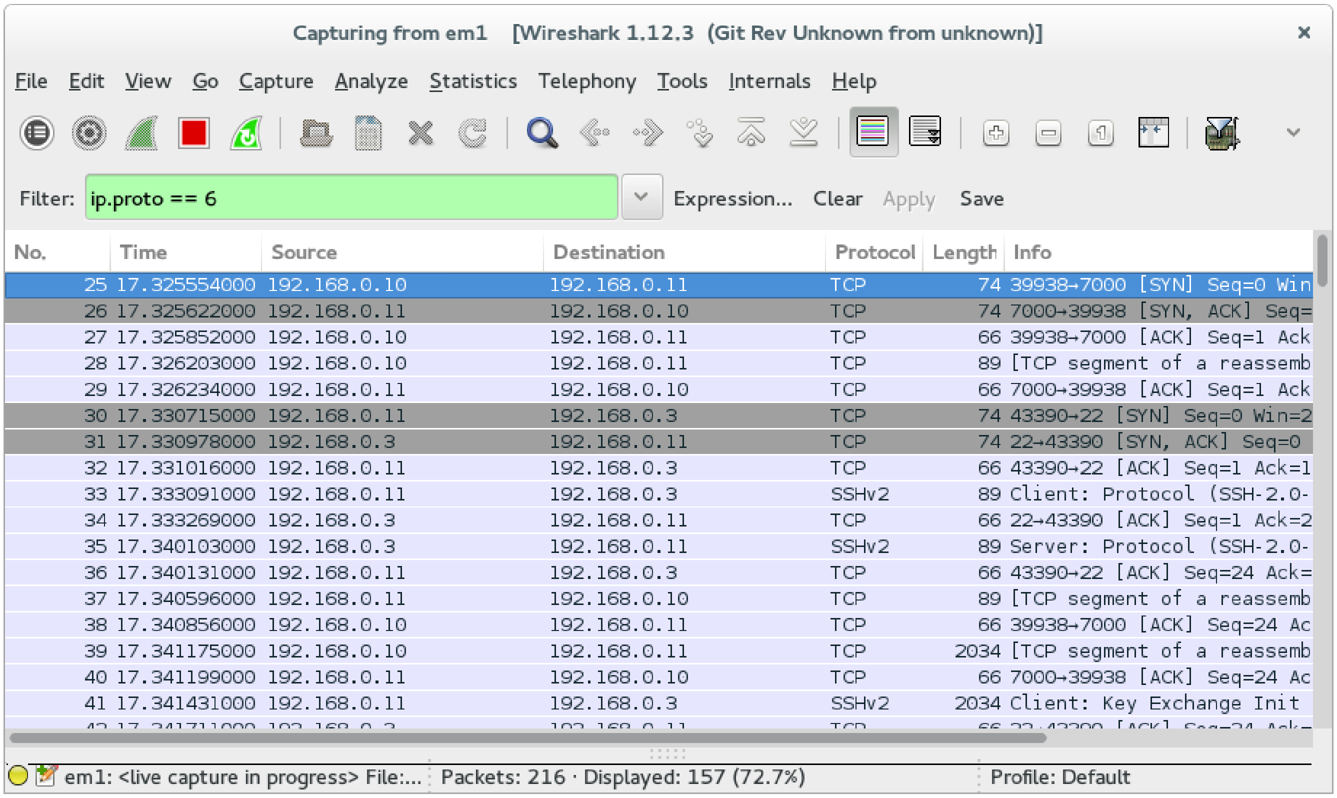
In the above we saw 2 rules for forwarding packets. One of which is for SSH to be forwarded from port 7000 to 22 on 192.168.0.3.



```
[root@DataComm ~]# Connection to 192.168.0.11 closed by remote host.
Connection to 192.168.0.11 closed.
[root@DataComm RPort]#
[root@DataComm RPort]# ssh -p 7000 192.168.0.11
root@192.168.0.11's password:
Last login: Mon Mar 23 22:01:40 2015 from 192.168.0.11
[root@DataComm ~]# ifconfig
em1: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.0.3  netmask 255.255.255.0  broadcast 192.168.0.255
        inet6 fe80::7a2b:cbff:fea3:3f85  prefixlen 64  scopeid 0x20<link>
        ether 78:2b:cb:a3:3f:85  txqueuelen 1000  (Ethernet)
        RX packets 94308  bytes 111604854 (106.4 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 58955  bytes 4982141 (4.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 20  memory 0xe1b00000-e1b20000

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 0  (Local Loopback)
        RX packets 19  bytes 1640 (1.6 KiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 19  bytes 1640 (1.6 KiB)
```

As you can see here, we connected from a terminal to the port forwarder at 192.168.0.11 on port 7000. The port forwarder then sent us to 192.168.0.3, according to the forwarding rule as you can see in IFCONFIG. Below is a transfer demonstrating the actual forward in action, this is taken from the port forwarder.


Capturing from em1 [Wireshark 1.12.3 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

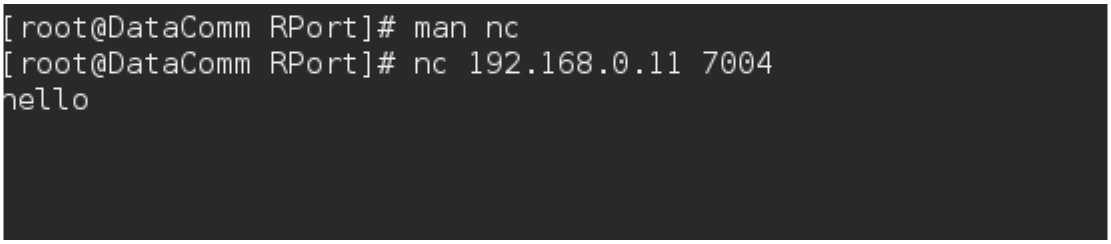
Filter: ip.proto == 6

No.	Time	Source	Destination	Protocol	Length	Info
25	17.325554000	192.168.0.10	192.168.0.11	TCP	74	39938→7000 [SYN] Seq=0 Win
26	17.325622000	192.168.0.11	192.168.0.10	TCP	74	7000→39938 [SYN, ACK] Seq=
27	17.325852000	192.168.0.10	192.168.0.11	TCP	66	39938→7000 [ACK] Seq=1 Ack
28	17.326203000	192.168.0.10	192.168.0.11	TCP	89	[TCP segment of a reassemb
29	17.326234000	192.168.0.11	192.168.0.10	TCP	66	7000→39938 [ACK] Seq=1 Ack
30	17.330715000	192.168.0.11	192.168.0.3	TCP	74	43390→22 [SYN] Seq=0 Win=2
31	17.330978000	192.168.0.3	192.168.0.11	TCP	74	22→43390 [SYN, ACK] Seq=0
32	17.331016000	192.168.0.11	192.168.0.3	TCP	66	43390→22 [ACK] Seq=1 Ack=1
33	17.333091000	192.168.0.11	192.168.0.3	SSHv2	89	Client: Protocol (SSH-2.0-
34	17.333269000	192.168.0.3	192.168.0.11	TCP	66	22→43390 [ACK] Seq=1 Ack=2
35	17.340103000	192.168.0.3	192.168.0.11	SSHv2	89	Server: Protocol (SSH-2.0-
36	17.340131000	192.168.0.11	192.168.0.3	TCP	66	43390→22 [ACK] Seq=24 Ack=
37	17.340596000	192.168.0.11	192.168.0.10	TCP	89	[TCP segment of a reassemb
38	17.340856000	192.168.0.10	192.168.0.11	TCP	66	39938→7000 [ACK] Seq=24 Ac
39	17.341175000	192.168.0.10	192.168.0.11	TCP	2034	[TCP segment of a reassemb
40	17.341199000	192.168.0.11	192.168.0.10	TCP	66	7000→39938 [ACK] Seq=24 Ac
41	17.341431000	192.168.0.11	192.168.0.3	SSHv2	2034	Client: Key Exchange Init
42	17.341711000	192.168.0.3	192.168.0.11	TCP	66	22→43390 [ACK] Seq=24 Ack=

em1: <live capture in progress> File:... Packets: 216 · Displayed: 157 (72.7%) Profile: Default

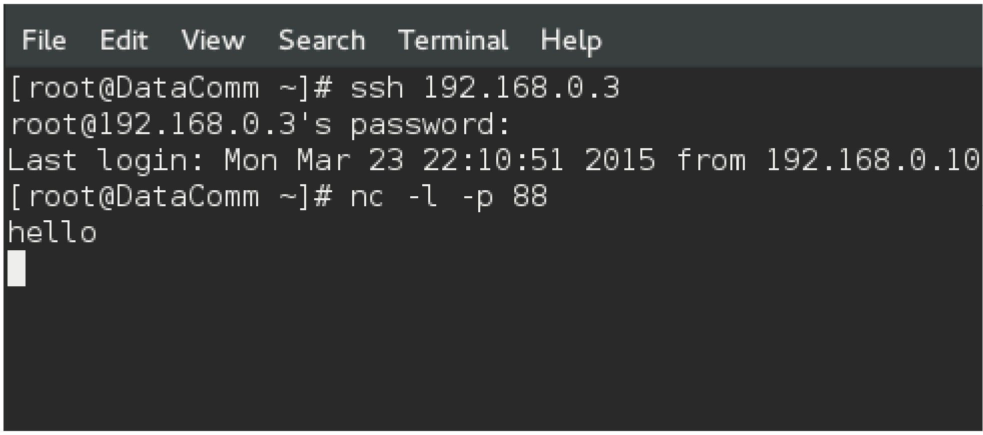
As you can see, the packets bounce from .10 to .11 to .3, this is the forwarding in action.

Test 3 – Receiving Data



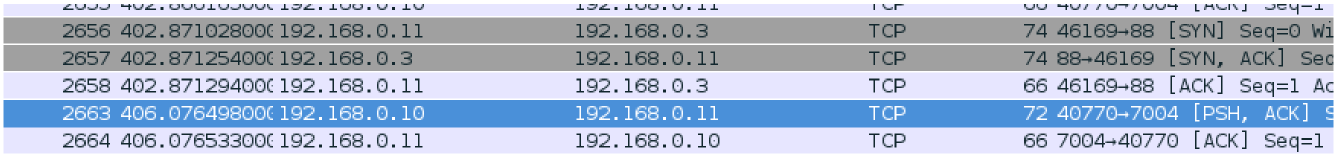
```
[root@DataComm RPort]# man nc
[root@DataComm RPort]# nc 192.168.0.11 7004
hello
```

In this test we used netcat to send data along a forwarder port. As you can see above, I used netcat on . 11 at 7004, a port we have a rule for. This rule will forward it to the netcat server waiting at .3.



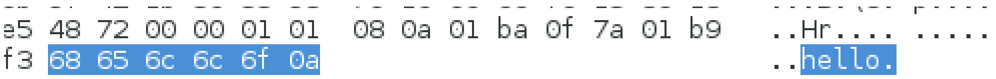
```
[root@DataComm ~]# ssh 192.168.0.3
root@192.168.0.3's password:
Last login: Mon Mar 23 22:10:51 2015 from 192.168.0.10
[root@DataComm ~]# nc -l -p 88
hello
```

This picture shows the server waiting on the terminal being forwarded to. As you can see, hello was sent between the two programs.



```
2656 402.87102800( 192.168.0.11   192.168.0.3    TCP   74 46169→88 [SYN] Seq=0 Wi
2657 402.87125400( 192.168.0.3    192.168.0.11   TCP   74 88→46169 [SYN, ACK] Sec
2658 402.87129400( 192.168.0.11   192.168.0.3    TCP   66 46169→88 [ACK] Seq=1 Ac
2663 406.07649800( 192.168.0.10   192.168.0.11   TCP   72 40770→7004 [PSH, ACK] S
2664 406.07653300( 192.168.0.11   192.168.0.10   TCP   66 7004→40770 [ACK] Seq=1
```

Here is a transfer of what occurred. As you can see the psh is highlighted here. This was the data contained within. This test was a success.

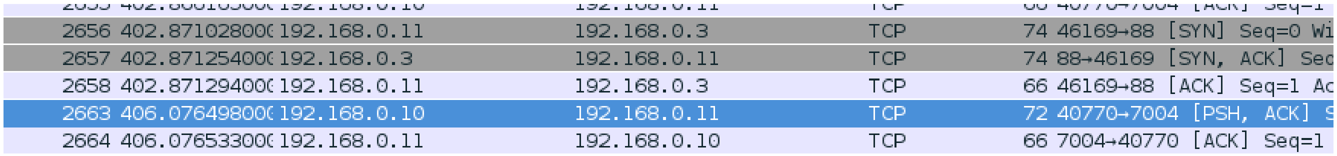


```
e5 48 72 00 00 01 01  08 0a 01 ba 0f 7a 01 b9   ..Hr.... .....
f3 68 65 6c 6c 6f 0a                            ..hello.
```

Test 4 – Packet Data

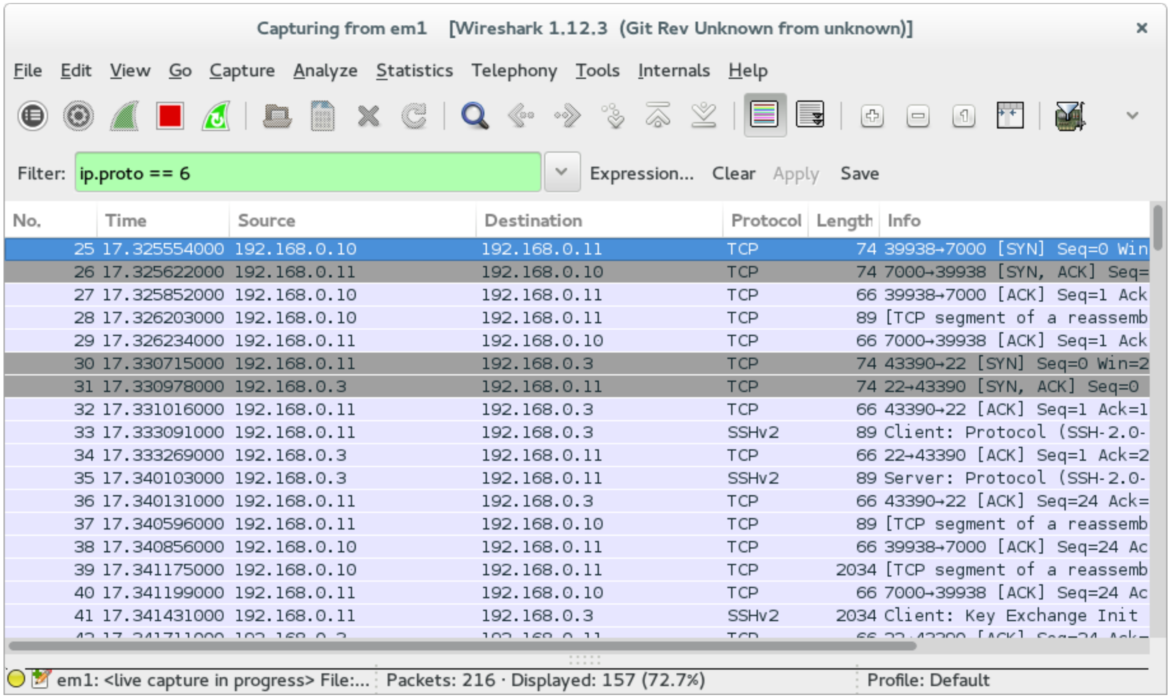
For this one, refer to the previous test as it demonstrates the existence of valid packet data.

Test 5 – Handshake



```
2656 402.87102800( 192.168.0.11   192.168.0.3    TCP   74 46169→88 [SYN] Seq=0 Wi
2657 402.87125400( 192.168.0.3    192.168.0.11   TCP   74 88→46169 [SYN, ACK] Sec
2658 402.87129400( 192.168.0.11   192.168.0.3    TCP   66 46169→88 [ACK] Seq=1 Ac
2663 406.07649800( 192.168.0.10   192.168.0.11   TCP   72 40770→7004 [PSH, ACK] S
2664 406.07653300( 192.168.0.11   192.168.0.10   TCP   66 7004→40770 [ACK] Seq=1
```

As you can see here, a 3 way handshake did succesfully work between 3 computers within the forwarding chain. This demonstrates a successful TCP Handshake, the picture below also shows a successful handshake.



Capturing from em1 [Wireshark 1.12.3 (Git Rev Unknown from unknown)]
File Edit View Go Capture Analyze Statistics Telephony Tools Internals Help
Expression... Clear Apply Save

Filter: ip.proto == 6

```
No. Time         Source        Destination   Protocol Length Info
 25 17.325554000 192.168.0.10  192.168.0.11  TCP        74 39938→7000 [SYN] Seq=0 Win
 26 17.325622000 192.168.0.11  192.168.0.10  TCP        74 7000→39938 [SYN, ACK] Seq=
 27 17.325852000 192.168.0.10  192.168.0.11  TCP        66 39938→7000 [ACK] Seq=1 Ack
 28 17.326203000 192.168.0.10  192.168.0.11  TCP        89 [TCP segment of a reassemb
 29 17.326234000 192.168.0.11  192.168.0.10  TCP        66 7000→39938 [ACK] Seq=1 Ack
 30 17.330715000 192.168.0.11  192.168.0.3   TCP        74 43390→22 [SYN] Seq=0 Win=2
 31 17.330978000 192.168.0.3   192.168.0.11  TCP        74 22→43390 [SYN, ACK] Seq=0
 32 17.331016000 192.168.0.11  192.168.0.3   TCP        66 43390→22 [ACK] Seq=1 Ack=1
 33 17.333091000 192.168.0.11  192.168.0.3   SSHv2      89 Client: Protocol (SSH-2.0-
 34 17.333269000 192.168.0.3   192.168.0.11  TCP        66 22→43390 [ACK] Seq=1 Ack=2
 35 17.340103000 192.168.0.3   192.168.0.11  SSHv2      89 Server: Protocol (SSH-2.0-
 36 17.340131000 192.168.0.11  192.168.0.3   TCP        66 43390→22 [ACK] Seq=24 Ack=
 37 17.340596000 192.168.0.11  192.168.0.10  TCP        89 [TCP segment of a reassemb
 38 17.340856000 192.168.0.10  192.168.0.11  TCP        66 39938→7000 [ACK] Seq=24 Ac
 39 17.341175000 192.168.0.10  192.168.0.11  TCP      2034 [TCP segment of a reassemb
 40 17.341199000 192.168.0.11  192.168.0.10  TCP        66 7000→39938 [ACK] Seq=24 Ac
 41 17.341431000 192.168.0.11  192.168.0.3   SSHv2    2034 Client: Key Exchange Init
```

em1: <live capture in progress> File:... Packets: 216 · Displayed: 157 (72.7%) Profile: Default