ONL Plugin Exercises (Jan 2005)

Ken Wong
Washington University

kenw@wustl.edu
www.arl.wustl.edu/~kenw

Preliminaries (1)

- We assume you know basic Unix
- Use a 1 sec traffic polling interval, not 0.5 sec
- Put basic plugin source code into your home directory tree
  - » ssh to onl.arl
  - » Make a directory somewhere in your directory tree
    - mkdir plugins
  - » Copy plugin tar file to the directory and extract files
    - cp /users/onl/export/plugins.tar plugins
    - cd plugins
    - tar xf plugins.tar
- If your plugin wedges a port, move to another port
- NOTE: The exercises are only suggestions. Feel free to try whatever strikes your fancy

Preliminaries (2)

- Plugin management commands
  - » Normally in a CVS tree but we'll use /users/onl/bin/*
  - » Copy useful csh and bash commands to your home directory
    - cp /users/onl/export/=srcthis.{csh,bash} ~
  - » Set up aliases to commands
    - Put one of the following commands as the last line in your .cshrc or .bashrc file (which one depends on your login shell)
    - csh users: source ~/=srcthis.csh
    - bash users: source ~/=srcthis.bash
- Help
  - » sendcmd –h
  - » cfy –h
  - » /users/onl/bin/usage-*.txt are help files for some common commands

Overview of Exercises

- Observe the COUNTER plugin
- Modify COUNTER plugin to collect byte count
- Add more control to the COUNTER plugin
  - » Allow for msg types
    - CLR_COUNT: Zero the byte counter
    - GET_COUNT: Return the byte counter value
- Add debug msgs to the COUNTER plugin
  - » Add debug output
    - Display the src and dst IPv4 addresses in host byte order
- Build up a queue of 2 pkts and then start forwarding in FIFO order (a "push out queue")
  - » Use control msg to toggle between push and flush mode

Exercise 1, Observe … (1)

- RLI (No filters)
  - Get 1 cluster and set default routing at all ports
  - Start monitoring ingress and egress bandwidth at ports 2 and 3
- Traffic
  - Send 20 1000-byte ping pkts from n1p2 to n1p3
  - Repeat with 100-byte ping pkts
  - Do the traffic plots make sense?
- RLI (Add GM filter)
  - Add a GM filter at ingress port 2 to match ANY pkts from n1p2
    - 192.168.1.48/32, 0, 0.0.0.0/0, 0, proto=0, fwd=3, priority=58, plugin, no aux
  - Remember to commit

Exercise 1 (2)

- NOTE: You may elect to use the PM_demo tool
- Bind 1 instance of COUNTER plugin to GM filter
  - ssh CPnode # Call this the command window
  - cd <COUNTER plugin directory>
  - Quiet:
    - $SND -p 2 -c set_debug -l error -m all
    - $SND -p 2 -c policy -s set_dflags -d 2
  - Load: pluginDownload -d -e COUNTER -s combined.o -p 2
  - Create: $SND -c rp_pcu -s create -p 2 -i 432
  - Bind: cfy -p 2 -i 0 -q 8 -- bind
  - Probe Plugin: $SND -p 2 -c rp_inst -d 0 -d 0
    - Should get response from plugin
- Repeat earlier traffic
  - Are the traffic plots the same as before? Or different?

Exercise 1 (3)

- How many pkts has plugin seen?
  - » $SND –p 2 –c rp_inst –d 0 –d 0
    - Does the response make sense?
  - » Send 10 more pkts and repeat
    - Does the response still make sense?
- Observe debug messages
  - » Open another window to the CP
    - Start 'monmsgs'
  - » Command window
    - Noisy: $SND –p 2 –c set_debug –l verbose –m all
  - » Send 1 ping pkt from n1p2 to n1p3
    - Does the monmsgs window make sense?

Exercise 1 (4)

- Unbind, Free, Unload
  - » Observe the monmsgs window
    - Does the output make sense?
  - » cfy –p 2 –I 0 –q 8 -- unbind
  - » $SND –c rp_pcu –s free –p 2 -i 0
  - » $SND –c rp_pcu –s unload –d 0 –p 2

Exercise 2, Modifying Plugin (1)

- Change COUNTER source code to determine the total number of IP pkt bytes the plugin receives
- In the COUNTER directory, save the original code
  - mkdir orig
  - cp COUNTER.[hc] orig
  - cp Makefile orig
- Length field (bytes) in IP header
  - Is defined in /usr/include/netinet/ip.h as the u_int16_t ip_len field in 'struct ip'
  - Ptr to IP header in COUNTER_handle_packet
    - HDRQ_t *hdrs = plist; // plist is 2nd handle_packet arg
    - struct msr_bufhdr_t *hdr = TAILQ_FIRST(hdrs);
    - struct ip *iph = (struct ip *) msr_pkt_iph(hdr);
  - Length field must be converted to Host Byte Order
    - nbytes = ntohs(iph->ip_len);

Exercise 2 (2)

- Define member 'byte_count' in COUNTER.h
  - int byte_count; // see struct COUNTER_instance{ }
- Modify COUNTER_handle_packet to sum the pkt length
  - inst->byte_count += nbytes; // defined earlier
- Modify COUNTER_create instance to initialize byte_count
  - myinst->byte_count = 0;
- Modify COUNTER_handle_msg to return both the byte count and the pkt count
- Recompile
  - ssh onlbsd2
  - cd <COUNTER Directory>; make

Exercise 2 (3)

- Test new plugin with ping traffic
  - » Before sending traffic start monmsgs at the CP
  - » Make the SPC at port 2 noisy (verbose)
  - » Send 1 ping pkt from n1p2 to n1p3
    - NOTE: Each default length ping pkt should be 20+64 = 84 bytes
    - 20 bytes IP hdr; 8 bytes ICMP hdr; 56 bytes data
  - » Watch the monmsgs window
  - » Repeat sending 1 ping pkt

Exercise 3, Adding Control ... (1)

- Add the ability to clear the byte counter
- Add another message type in COUNTER_handle_msg for resetting the byte counter
  - » See ../delay/pdelay.c or ../syn_demo/syn_demo.c for examples
  - » Involves writing a switch statement that will demultiplex based on the second word of the incoming buffer
    - 'data' points to this location

Exercise 4, Debug Msgs … (1)

- Display the src and dst IPv4 addresses in host byte order using the MSR_DEBUG macro
- IP header structure
  - » See Exercise 2
    - u_int32_t src_addr, dst_addr;
    - u_int16_t seq_no;
    - HDRQ_t *hdrs = plist; // plist is 2nd handle_packet arg
    - struct msr_bufhdr_t *hdr = TAILQ_FIRST(hdrs);
    - struct ip *iph = (struct ip *) msr_pkt_iph(hdr);
    - msr_ipsaddr(iph) and msr_ipdaddr(iph) are the src and dst addresses
- See any MSR_DEBUG call in COUNTER.c
- Call MSR_DEBUG( ) in COUNTER_handle_packet

Exercise 4 (1)

- Build up a queue of 2 pkts and then start forwarding in FIFO order (a "push out queue")
  - » Use control msg to toggle between pushd and flush mode
  - » In flush mode, all pkts in the queue are forwarded up arrival of the next pkt
  - » In push mode, pkts are held in the queue until the #pkts reaches 2; then every arrival pushes out the first pkt
- Canabalize the code in pdelay_handle_packet( ) found in delay/delay.c
  - » The 2nd argument plist is a ptr to a pkt list structure
  - » You should move the packet from the incoming pkt list onto your internal queue which uses the same type of pkt list structure

Exercise 4 (2)

- A pkt list is a doubly-linked list
  - Documented in NetBSD queue(3) man page as the tail queue facility
  - Uses 3 interface functions
    - TAILQ_FIRST(phdrs), TAILQ_REMOVE(phdrs, qhdr, qlist) and TAILQ_INSERT_TAIL(&inst->qhead, qhdr, qlist)
- Test the plugin with several consecutive low bandwidth UDP iperf streams
  - Client: iperf –u –c n1p3 –b 1k –l 100 –n 600
    - Send 6 UDP 100-byte pkts to n1p3 at 1 Kbps
  - Server: iperf –u -s