The Open Network Lab (Part 3)

Ken Wong
Applied Research Laboratory
Computer Science and Engineering Department
http://www.arl.wustl.edu/~kenw
kenw@arl.wustl.edu
http://www.onl.wustl.edu (ONL)
National Science Foundation ANI-023826, CNS-0551651, REL-0632580

Topics
1) Introduction
2) Lab 4 overview
3) Standard plugins
   • ~onl/stdPlugins/
4) Lab 4 – Configuration A
5) ~kenw/new-plugins/discard-99/
6) Your own discard-99
   • ~YourLogin/myplugins/discard-99/
   • copy, recompile, test, debug mode
7) Lab 4 – Configuration B
8) Lab 4 – Configuration C

Plugin Ex: SYN Flood Mitigation
- SYN Flood Attacker
  » Sends many TCP SYN pkts to target
  • Signals new TCP connection
  » Never responds to server’s SYN-ACK
  » Floors new connections table
- Features of syndemo plugin
  » TCP connection monitoring
  » Triggered generation of TCP RST (Reset) pkt to terminate incomplete TCP connection
  » Dynamic installation of EM filter
  » On/Off control of plugin

SYN Demo Running
- Plugin ON
- Web Srvr sends HTTP requests across Internet thru n1p2
The Good, The Bad, The Ugly

- **The Good**
  - Customize pkt processing ... (at wire speed)
    - e.g., security, quality of service, network services
  - ONL provides programming framework
    - user is passed ptr to pkt buffer in handle_pkt routine
    - most programming confined to 2-3 functions
- **The Bad**
  - Like debugging remote embedded processor
    - NO glibc functions
    - Can easily lock up kernel ➔ Restart experiment
- **The Ugly**
  - Like kernel network programming
    - C, not C++
    - byte ordering important (ntohs, ntohl, htons, htonl)
    - Dynamic allocation ➔ Liberal use of pointers
- **The Hope:** Incremental development of simple plugin

Configuration A

- Lab 2 sndr and rcvr supplied
- send 2,790 1000-byte pkts
  - from n1p2 to n1p3 thru p6-p7 loop
  - bottleneck: port 3
    - 10 Mbps, 2 MB queue (qid 300)
  - acks from n1p3 to n1p2 direct
    - 800 msec delay at p2out
  - thruput? effect on traffic?

Configuration B

- discard plugin drops pkt when pktCnt == target[i]
  - modify existing discard plugin
  - thruput? effect on traffic?

Configuration C

- dupe pkts at p2in (simple FEC)
  - thruput? effect on traffic?

The Plugin Framework

- A plugin can extend the capabilities of a PP
  - Examine or modify packet headers and/or bodies
  - Delay packets
  - Produce additional packets
  - Modify packet shims

Plugins are installed along a packet's data path

- The plugin framework follows an OO paradigm
  - A plugin instance (object) has its own local variables (state) and is created from a plugin class
  - Filters can direct their matching packets to plugin instances (NOT plugin classes)
  - The code for a plugin implements the plugin abstraction (e.g., load, create, bind, handle pkt, handle msg)
The Plugin Abstraction

- RLI plugin functions
  - Add instance: loads plugin into PP; creates instance
  - Unload: unloads plugin (class) from a PP
  - Bind: a plugin instance to a filter
  - Send a message to a plugin instance

- Existing code handles user commands in a basic way

- A filter-plugin binding causes matching pkts to be sent to the plugin instance for processing
  - The user writes a Plugin_handle_packet function to handle packets passed to it by the FPX

- The abstraction is implemented by the plugin code in cooperation with the SPC kernel code

Lab 4 Endhost Software

- sn2r -r n1p3 -l 1000 -n N -x
  - sends 3 ping pkts to prime ARP caches
  - send N 1000-byte UDP pkts to n1p3 (port 2000)
    - send 3 new pkts for each ACK
    - send 5 FIN pkts (sn = -N) when done
  - -x: announce the beginning of retransmissions
  - -s: run in slow motion (sleep 1 sec before sending)
  - -v: display each pkt sn before sending

- rcrv2 -x 2000
  - receive UDP pkts using in-order accept policy
    - ACK only a pkt with next expected sn (sequence number)
    - initially, expect sn = 0
    - if accept sn = x, next expected sn = x+1
    - reject all unexpected pkts (don't ACK)
  - -x 2000: announce whenever accept sn%2000 == 0
  - display [lo, hi] ranges of accept/reject sn's and pn's

Application Pkt Header

```c
struct pkthdr {
  char debug[8];  // debug string
  char tag[8];  // "UdpFlow"
  int sn;  // sequence number starting at 0
  int pn;  // packet number starting at 0
  tv_t ts[4];  // time stamps
};
typedef pkthdr_t struct pkthdr;
```

- see ~kenw/src/lab4y/lab.h
- sn2r
  - stores string "UdpFlow" into tag[] field
  - sn: sequence number in NBO (Network Byte Order)
  - pn: pkt number in NBO
  - plugin will see sn and pn in NBO, not HBO (Host Byte Order)
- rcrv2 converts sn and pn to HBO before displaying
Configurations A and B

Demo 1 (My Configuration B)
- Start with config A
- "Add GM filter" to port 6 (ingress)
  - Direct all UDP pkts to discard plugin (SPC queue 8)
- "Add instance" of discard-99 plugin to port 6 (ingress)
  - p6: Plugin Table → Edit → Add instance → [new-plugins] discard-99
- commit
  - p6: Plugin Table → Port 6 Plugins instance = 0
  - shows instance was created
- Start rcvr2 at n1p3
  - rcvr2 -x 50
- Start sndr2 at n1p2 and send 100 pkts
  - sndr2 -r n1p3 -l 1000 -n 100 -x

rcvr2 Output Example
- Enter REJECT: acc sn[0, 19], pn[0, 19], nxtsn = 20
- Enter REJECT: rej sn[21, 39], pn[21, 39], nxtsn = 20
- Enter ACCEPT: rej sn[41, 60], pn[41, 60], nxtsn = 20
- Enter KIENT: acc sn[20, 38], pn[61, 79], nxtsn = 39

Demo 1 (Your Configuration B)
- Same steps as "My Configuration B" except ...
- Follow instructions for your own plugin
  - see "Your Own discard-99" slide
- Use "File → Save as" to save new configuration
**No discard Plugin**

- send 100 pkts
- no drops
- effect of \(-x\) flag
- low bw (only 5 rounds)
- transfer time \(\approx 5 \times 0.8\) sec

**GM Filter Directs Pkts to Plugin**

- send pkts to queue 8 (SPC)
- need to tell plugin to get pkts from queue 8
- pkts return to FPX in queue 8+128
- don’t send ping pkts to plugin

**Add Plugin Instance**

- commit
- tell plugin pkts will be in queue 8
- instance 0

**Send 100 1000-Byte Pkts**

- bug (fixed)
- discard-99 plugin (drops sn 3)
- rejected pkts 4-9
~kenw/new-plugins/discard-99/

- **discard.h** (interface)
  - define plugin ID to be 99
  - define instance variables; e.g., pktCnt
- **discard.c** (implementation of interface)
  - discard_handle_packet, discard_create_instance, discard_handle_message, etc.
- **Makefile**
  - make clean; make non-debug module
  - make clean; make debug # make debug module
- **discard.o**
  - result of compiling: gcc -c discard.c ...
- **combined.o**
  - module loaded into SPC kernel

discard-99/discard.h

```c
# define discard_ID 99
enum { OFF = 0, ONCE, PERIODIC, VECTOR }; // modes
const int DEF_TARGET = 3; // default target
struct discard_instance {
    struct rp_instance rootinstance;
    int pktCnt; // #pkts seen recently
    int pktTot; // total #pkts seen
    int dropCnt; // drop counter
    int target; // target pkt
    int mode;   // OFF: don't drop any pkts
    // ONCE: drop when pktCnt==target
    . . .  vector mode code omitted  . . .
};
```

numeric id, unique within directory

if p points to plugin instance then p->pktCnt is pkt counter

user adds these

Under The Hood

- **RLI**
  - Add instance
    - (qid 8) discard_load(…)
    - instance 0 discard_create_instance(…)
  - Delete instance
    - discard_unbind_instance(…)
  - Unload class
    - discard_unload(…)

- **Plugin**
  - commit

commit

commit

commit
discard_create_instance(…)
- plugin instance initialization code
- called when you select "Add instance" in RLI

discard_handle_packet(…)
- pkt processing code
- called when a pkt arrives to SPC
- you are passed a single pkt buffer thru param list
- pkt is forwarded if you don’t remove pkt buffer from param list
- if want to drop pkt, then
  - remove pkt buffer from param list (msr_removeBuffer)
  - free buffer memory (msr_freeBuffer)

discard_handle_message(…)
- handle control messages to plugin instance

```
discard_create_instance...
struct rp_instance * discard_create_instance(
    struct rp_class *theClass,  // ptr to class structure
    u_int32_t instanceid // id for new instance
) {
    struct discard_instance *newInst;
    newInst = ... allocate memory for instance ...
    ... init standard instance variables ...
    newInst->pktCnt = 0; // Init per instance variables
    newInst->dropCnt = 0;
    newInst->target = DEF_TARGET;
    newInst->mode = ONCE;
    newInst->vec[0] = 121;
    newInst->vec[1] = 485;
    newInst->vec[8] = 5463;
    newInst->vec[9] = 6556;
    newInst->vec[10] = 0;
    return (struct rp_instance *) newInst;
}
```

Vector Mode Code in create_instance()
```
newInst->ndx = 0; // which vec[]
newInst->vec[0] = 121;
newInst->vec[1] = 485;
newInst->vec[8] = 5463;
newInst->vec[9] = 6556;
newInst->vec[10] = 0;
```

- May want to make these changes
  - Default mode to VECTOR
  - Default target to match newInst->vec[0]

discard_handle_packet(...)
```
discard_handle_packet(...
void discard_handle_packet(
    struct rp_instance *this, // ptr to instance structure
    void *bufferList // ptr to list of packet buffers
) {
    struct discard_instance *inst = ...
    msr_bufhdr_t *buffer = msr_firstBuffer(bufferList);
    struct ip *iph = msr_pkt_iph(buffer);
    int len = msr_iplen(iph); // not used: hdr+data
    int proto = msr_ipproto(iph); // protocol
    int sn; // application layer sequence number
    int hdrsize; // size of header
    int discardit = 0;
    ... lines deleted ...
    return;
}
```

- pkt will be forwarded unless you remove pkt buffer from bufferList
+ discard_handle_packet(...)

```c
++inst->pktTot; ++inst->pktCnt;
if ( inst->mode == OFF ) return; // fwd pkt
if ( inst->mode == ONCE ) {
  if ( (inst->pktCnt-1) == inst->target ) {
    ++inst->dropCnt;
    msr_removeBuffer(bufferList, buffer);
    msr_freeBuffer(buffer);
    return;
  }
}
else if ( inst->mode == PERIODIC ) {
  // insert code here if you wish
}
else if ( inst->mode == VECTOR ) {
  // insert code here if you wish
}
else {                    // bad mode
  MSR_DEBUG( ( . . . ) );
  // forward pkt anyway
}
return;
```

- incr counter
- remove buffer
- free buffer

otherwise, leave buffer alone; i.e., forward pkt

Your Own discard-99 (1)

- Make a personal plugin directory (~/myplugins)
  - mkdir ~/myplugins
- Copy discard-99 code to ~/myplugins
  - cp –R ~kenw/new-plugins/discard-99 ~/myplugins
- Test non-debug version
  - Recompile on onlusr
    - cd ~/myplugins/discard-99
    - make clean; make # should be NO undefined symbols
    - “ls –l” should show a new combined.o file
  - Install plugin at port 6 (ingress)
  - Test by sending 10 1000-byte pkts
    - rcvr: rcvr2 –x 5
    - sndr: sndr2 –r n1p3 –l 1000 –n 10 –x

Your Own discard-99 (2)

- Test debug version (only if you need to)
  - Recompile (debug version)
    - make clean; make debug # should see MSRDEBUG used
  - Delete instance from port 6 (ingress)
  - Unload plugin
  - Install plugin (debug version) at port 6 (ingress)
  - RLI: turn on debugging output from port 6
  - Test by sending 10 1000-byte pkts (slow mo)
    - rcvr: rcvr2 –x 5
    - sndr: sndr2 –r n1p3 –l 1000 –n 10 –x -s
  - /tmp/debug.log file on CP host
  - SSN to CP
    - cd /tmp
  - more debug.log or # look at output
  - tail –f debug.log # real-time

Compile-Time Error Examples

- undeclared variable name foo
  - /usr/local/xcomp/bin/i386--netbsdelf-gcc -DMSR . . . discard.c
discard.c: In function ‘discard_handle_packet’:
discard.c:122: error: ‘foo’ undeclared (first use in this function)
discard.c:122: error: (Each undeclared identifier is reported only once
discard.c:122: error: for each function it appears in.)
make: *** [discard.o] Error 1

- undeclared function call goofunc
  - /usr/local/xcomp/bin/i386--netbsdelf-gcc -DMSR . . . discard.c
discard.c: In function ‘discard_handle_packet’:
discard.c:122: error: ’foo’ undeclared (first use in this function)
```
**Debugging With MSRDEBUG(...)**

MSR_DEBUG( MSR_DEBUG_PLUGIN | MSR_DEBUG_LEVEL_INFO, "handle_packet BEGIN: Got pkt with iplen %d, proto %d\yn", len, proto );

- MSR_DEBUG is a macro
  - Note extra set of parentheses
- printf-like facility
- Output shows up in /tmp/debug.log of CP
  - if turn on debugging in RLI for port
- MSR_DEBUG_LEVEL_XXX controls verbosity
  - VERBOSE, INFO, WARNING, ERROR, etc.
- Can only be used with slow traffic (1 pkt/sec)

---

**Turning On Debugging**

- nsrpcp3:/tmp/debug.log is log file
  - your CP will depend on result of commit
- Output of MSRDEBUG output

---

**Strategy: Modifying discard-99/*.hc**

- Core strategy
  - Change default mode to VECTOR
    - disc.c: discard_create_instance(...)
  - Define target list in an array (instance variable)
    - disc.c: discard_create_instance(...)
    - ALREADY DONE FOR YOU (except for initial target, mode)
  - Insert code to detect target hit
    - disc.c: discard_handle_packet(...)
    - drop pkt if hit; forward pkt otherwise
- Add on strategy
  - Reset plugin when sn == 0
    - plugin can also be reset manually
  - Forward pkt unless pkt is UDP with "UdpFlow" tag
    - Cheap Solution: Look just for "U" in apphdr->tag[0]

---

**discard_create_instance(...)**

```
struct rp_instance * discard_create_instance(
    struct rp_class *theClass,  // ptr to class structure
    u_int32_t instanceid // id for new instance)
{
    struct discard_instance *newInst;
    newInst = ... allocate memory for instance ...
    ... init standard instance variables ...
    newInst->pktCnt = 0;    // Init per instance variables
    newInst->pktTot = 0;
    newInst->dropCnt = 0;
    newInst->mode = ONCE;
    newInst->target = DEF_TARGET;  
    ... vector code omitted ...
    return (struct rp_instance *) newInst;
}
```
void discard_handle_packet(
    struct rp_instance *this, // ptr to instance structure
    void *bufferList       // ptr to list of packet buffers
) {
    struct discard_instance *inst =
        (struct discard_instance *) this;
    msr_bufhdr_t *buffer = msr_firstBuffer(bufferList);
    struct ip *iph = msr_pkt_iph(buffer);
    int len    = msr_iplen(iph);    // not used: hdr+data
    int proto   = msr_ipproto(iph);  // protocol
    int sn;    // application layer sequence number
    int hdrsz; // size of header
    int discardit = 0;
    ...
    . . .  lines deleted . . .
    return;
}

pkt will be forwarded unless you remove pkt buffer from bufferList

++inst->pktTot;        ++inst->pktCnt;
if ( inst->mode == OFF )    return;  // fwd pkt
if ( inst->mode == ONCE ) {
    if  ( (inst->pktCnt-1) == inst->target ) {
        ++inst->dropCnt;
        msr_removeBuffer(bufferList, buffer);
        msr_freeBuffer(buffer);
        return;
    }
} else if ( inst->mode == PERIODIC ) {
    // insert code here if you wish
} else if ( inst->mode == VECTOR ) {
    // insert code here if you wish
} else {                    // bad mode
    MSR_DEBUG( ( . . . ) );
    // forward pkt anyway
}
return;

• incr counter
• remove buffer
• free buffer

insert your code here

otherwise, leave buffer alone; i.e., forward pkt

Code Sketch (Core)

Example
- Want to drop pkt when pktCnt == 20, 40 or 80
  - i.e., drop 20th, 40th, 80th pkts
- Want a drop vector
  - e.g., inst->vec[0,1,2,3] = { 20, 40, 80, 0 }
    - define in discard.h
    - initialize in discard_create_instance(...)  
  - need index:  inst->nxt = 0
    - initialize in discard_create_instance(...)  
  - increment after each drop but no more than 3 times in discard_handle_packet(...)  
  - set inst->target to new target
    - in discard_handle_packet
      + inst->nxt = inst->target = inst->vec[inst->nxt]
    - except when done with drop vector !!!

Reinitialize Plugin Instance

Method 1 (crude)
- RLI: delete instance and then add instance
  - discard_create_instance is called

Method 2 (manual)
- send msg to plugin
  - operation = 3; operands = 3  20 (mode 3, target 20)
  - But need to update code in discard_handle_message(...) to reset inst->nxt and inst->target

Method 3 (auto)
- insert code into discard_handle_packet(...)
  - when apphdr->sn == 0  
    - inst->nxt = 0; inst->target = inst->vec[0];
    - inst->pktCnt = 0; inst->dropCnt = 0;
  - But how to find application header? i.e., apphdr = ?
Configuration C

- Start with Configuration B
- Add an auxiliary filter at ingress port 2
  - duplicates every udp pkt it sees; forwards to port 7
  - RLI: Select "Port 2 ➔ Ingress Filters"
  - Port 2 Ingress Filter Window
    - "Edit ➔ Add General Match Filter"
    - Set protocol to "udp"
    - Set priority to 50
    - Select "aux" box (for auxiliary)
    - In "forward to voq" field, select "7"

* ALERT
  - May not want to use ~x flag in rcvr2 command
  - high number of duplicate pkts which will be rejected
  - end up with lots (max 20) of 1-pkt reject ranges

Sending Messages to Plugin Instance

- Primitive message system
  - command code
  - operands
    - sequence of white-space separated integers
  - semantics defined in handle_message code
  - euphemism for "ad hoc"

To set mode to VECTOR (3) and reset target list:
- command id = 3
- parameters: 3

To set mode to ONCE (1) and target to 7:
- command id = 3
- parameters: 1 7