CS434/534: Topics in Network Systems

Network Function Virtualization: Programming, Composition

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson
Email: yry@cs.yale.edu

http://zoo.cs.yale.edu/classes/cs434/
Outline

- Admin and recap
- Network functions (carrier cloud)
  - overview
  - network function: programming
    - click (lower layer network function processing)
    - clickNP (click w/ hardware acceleration)
    - bro (L4-L7 middlebox programming)
    - Netbricks
  - network function composition
    - intra-device composition
    - inter-device (network-wide) composition
Admin

- Instructor office hours
  - Tuesday: 1:30-2:30
  - Thursday: 3:00-4:00
  - Fridays: 1:30-2:30 pm

- Projects
  - Milestones (exactly 4 weeks left)
    - 4/17 (T+1 week): Finish reading major related work; a google doc listing related papers (at least 4 papers)
    - 4/24 (T+2 weeks): Finish architecture design (slides/write up of architecture, including all key components)
    - 5/1 (T+3 weeks): Initial, preliminary evaluations (slides/write up, about experiment/analysis setup)
    - 5/8 (T+4 weeks) 5:30 pm, final report due
Recap: Data Parallel Applications

A communication stage cannot complete until all the data have been transferred.
Recap: Varys CoFlow API

<table>
<thead>
<tr>
<th>VarysClient Methods</th>
<th>Caller</th>
</tr>
</thead>
<tbody>
<tr>
<td>register(numFlows, [options]) → coflowId</td>
<td>Driver</td>
</tr>
<tr>
<td>put(coflowId, datalId, content, [options])</td>
<td>Sender</td>
</tr>
<tr>
<td>get(coflowId, datalId) → content</td>
<td>Receiver</td>
</tr>
<tr>
<td>unregister(coflowId)</td>
<td>Driver</td>
</tr>
</tbody>
</table>

Table 2: The Coflow API

```scala
def main { 
  val cId = client.register(6) 
  // Read from DFS, run user-written map method, 
  // and write intermediate data to disk. 
  // Now, invoke the coflow API. 
  for (r <- reducers) 
    client.put(cId, dId-m-r, content-m-r) 
  // Shuffle using the coflow API. 
  for (m <- mappers) 
    content-m-r = client.get(cId, dId-m-r) 
  // Now, sort, combine, and write to DFS. 
  client.unregister(cId) 
}
```
Recap: Varys Scheduling Architecture

- A two step algorithm
  - Ordering Heuristic: Typically scheduling algorithms are based on ordering, i.e., considering the jobs in some order (called permutation scheduling)

\[
\Gamma = \max \left( \max_i \frac{\sum_j d_{ij}}{\text{Rem}(P_{i}^{\text{in}})}, \max_j \frac{\sum_i d_{ij}}{\text{Rem}(P_{j}^{\text{out}})} \right)
\]

- Rate allocation algorithm: Allocates minimum required resources to each coflow to finish in minimum time
Recap: Varys -> Aalo

- Removes Varys assumptions:
  - Size of each flow is known
  - The total number of flows is known
  - The endpoints are known

- Framework
  - Dynamic, non-blocking scheduling
  - Priority discretization
    - FIFO within the same queue
    - Prioritization across queue
    - weighted sharing across queues to guarantees starvation avoidance
Recap: Major Network System Trend

- Programming and managing carrier network infrastructures in a similar way to programming and managing DC
  - Convert from expensive, hardware-centric network architecture to software-centric network architecture
    - Functions deployed called network functions
Recap: Network Functions

Dedicated hardware NFs are not flexible

Virtualized NFs on servers to maximize flexibility
Recap: Benefits and Challenges of Network Functions

- **Benefits**
  - Simplifies adding new functionality: Deploy new software
  - Simplifies developing new functionality: Write software vs design hardware
  - Reuse management tools from other domains.
  - **Consolidation**: Reduce number of hardware boxes in the network

- **Challenges**: new issues in
  - Programming individual NFs
  - Composing NFs
Recap: Click Programming Structure: A Graph of Network Elements

- Large number of small elements
  - Each performing a simple packet function
  - Introduce queue as an explicit element to clarify design
- Connected together in a directed graph
- Two modes for connected elements to interact: push or pull
Recap: Click Nice Features

- Composability
  - Graph model

- Extensibility
  - extend existing elements to construct new types
  - define new graph for new settings
Recap: Click Issues

- Potentially low performance (software based on x86 is slow)

- Simple non-blocking per-packet programming model

- Focusing on packet level processing, but many middleboxes need stream level processing
## Limited processing capacity

### Number of CPU cores needed for 40 Gbps line rate

<table>
<thead>
<tr>
<th>Network function</th>
<th>Implementation</th>
<th>1500B pkt @ 40 Gbps (normal case)</th>
<th>64B pkt @ 40 Gbps (worst-case estimate)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NVGRE tunnel encapsulation</td>
<td>Hyper-V virtual switch</td>
<td>5</td>
<td>100</td>
</tr>
<tr>
<td>Firewall (8K rules)</td>
<td>Linux iptables</td>
<td>21</td>
<td>480</td>
</tr>
</tbody>
</table>
Discussion: How to Accelerate Individual NF?
**Potential Direction: FPGA**

*Dirk (VMWare VP):* If you are starting today, what would you do?

*Linus Torvalds:* When I created Linux, the programming of hardware was much more easier than when I was a child...

I would consider FPGA if you are interested in chip design, because the price of FPGAs is more affordable these years, and there have been a lot of free tools to program them...

Source: LC3 Conference, Beijing, June 2017
One Slide FPGA

- Can contain combination logic and registers

![Diagram of a NOT gate and a D flip-flop]

```verilog
module myModule_tb();
  wire out;
  reg clock;

  always begin
    #1 clock = !clock;
  end

  initial begin
    //Initialize clock
    clock = 0;

    //End simulation
    #10
    $finish;
  end

  myModule notGate(clock, out);
endmodule

module myModule(A, B);
  input wire A;
  output wire B;
  assign B = !A;
endmodule
```

https://en.wikipedia.org/wiki/Register-transfer_level
Why FPGA

- Massive parallelism
  - Millions of logic elements => Thousands of “cores” in parallel
  - Thousands of memory blocks => TB/s memory bandwidth

- Low power consumption (~20W)

- General computing platform (vs. GPU, NP)

- Mature technology with a reasonable price
FPGA Challenge: Programmability

Hardware description language (HDL): push many software developers away

always @ (posedge SYSCLK or negedge RST_B) begin
  if (!RST_B)
    DMA_TX_DATA <= `UD 8’hff;
  else
    DMA_TX_DATA <= `UD DMA_TX_DATA_N;
end

//send: “hello world!”
always @ (posedge SYSCLK) begin
  if (rst) begin
    dma_data <= 88’h0;
    dma_valid <= 1’b0;
  end
end

if (dma_start) begin
  dma_data <= 88’h68656C6C20776F726C64;
  dma_valid <= 1’b1;
end
else begin
  dma_data <= 88’h0;
  dma_valid <= 1’b0;
end

Ahhhhhhhhhhhhhh!
**ClickNP**

- **Basic idea:**
  - Programmers write Click programs
  - System compiles to joint CPU/FPGA packet processing
Element: Single-threaded Core

- **states** (reg/mem)
- **Process handler** (main thread)
- **Signal handler** (ISR)

**Input channels** (I/O) → **Process handler** (main thread) → **Signal handler** (ISR) → **Output channels** (I/O)

- **Input channels** (I/O)
- **Output channels** (I/O)
- **Signal/control from host** (interrupt)
Element Interaction

cores (elements) running in parallel

communicate via channels, not shared memory (why?)

Figure 3: (a) Two ClickNP elements are connected through a channel. (b) The format of a flit.
Example: ClickNP Element

```c
.element Count <1, 1> {
    .state{
        ulong count;
    }
    .init{
        count = 0;
    }
    .handler{
        if (get_input_port() != PORT_1) {
            return (PORT_1);
        }
        flit x;
        x = read_input_port(PORT_1);
        if (x.fd.sop) count = count + 1;
        set_output_port(PORT_1, x);
        return (PORT_1);
    }
    .signal{
        ClSignal p;
        p.Sig.LParam[0] = count;
        set_signal(p);
    }
}
```
Offline Read

- Read ClickNP (detect and schedule parallelism)

- Potential research project: Offload to ASIC (e.g., RMT ASIC, NP)
Recap: Click Issues

- Potentially low performance (software based on x86 is slow)
- Simple non-blocking per-packet programming model
- Focusing on packet level processing, but many middleboxes need stream level processing
Outline

- Admin and recap
- Network functions (carrier cloud)
  - overview
  - network function programming
    - click (lower layer network function processing)
    - clickNP (click w/ hardware acceleration)
    - bro (L4-L7 middlebox programming)
Bro

- One of the first major, middlebox programming environment

- Highly influential for later designs in its domain
Bro Design Goals

- Real-time intrusion detection system (IDS) of attacks, e.g., worm attacks, DoS
  - observes network traffic, focus on detection

- Extensibility and reusability: New and different attack techniques are being constantly developed, an IDS that is hard to develop to keep up is useless

- Separating policy from mechanisms for modularity and performance
Bro Programming Framework

- Three-level hierarchy:
  - Network level: libpcap in the OS to filter unwanted packets and pass the rest up to the next level (benefit?)
  - Event level: Tracks state such as TCP hand-shake state, generates events and places events on a queue (benefit using queue?) to the next layer, e.g.,
    - SYN-ACK: connection_established
    - RST: connection_rejected
    - FIN: connection_finished
  - Policy level: A “policy script interpreter” reads events off the queue asynchronously and evaluates them. It records statistics, generates new events, and logging real-time alerts
Example: Using Bro to Monitor the Finger Protocol

- **FingerConn** derived from **TCP_Connection**
  - When Bro encounters a new connection w/ port 79, it instantiates a **FingerConn** object
  - **FingerConn** redefines virtual function **BuildEndpoints**, which is invoked when a connection object is first created

```cpp
void FingerConn::BuildEndpoints()
{
    resp = new TCP_Endpoint(this, 0);
    orig = new TCP_EndpointLine(this, 1, 1, 0);
}
```

Example: Using Bro to Monitor the Finger Protocol

```c
int FingerConn::NewLine(TCP_Endpoint* /* s */,
    double /* t */, char* line)
{
    line = skip_whitespace(line);

    // Check for /W.
    int is_long = (line[0] == '/' &&
        toupper(line[1]) == 'W');

    if (is_long) line = skip_whitespace(line+2);
    val_list* vl = new val_list;
    vl->append(BuildConnVal());
    vl->append(new StringVal(line));
    vl->append(new Val(is_long, TYPE_BOOL));
    mgr.QueueEvent(finger_request, vl);
    return 0;
}
```
Policy Script

global hot_names = { "root", "lp", "uucp"};
global finger_log =
  open(getenv("BRO_ID") == "" ?
    "finger.log",
    fmt("finger.%s", getenv("BRO_ID")));

event finger_request(c:connection,
  request: string,
  full: bool)
{
  if ( byte_len(request) > 80 ) {
    request = fmt("%s...",
                  sub_bytes(request, 1, 80));
    ++c$hot;
  }
  if ( request in hot_names )
    ++c$hot;

  local req = request == "" ?
    "ANY" : fmt("\"%s\"", request);
  if ( c$addl != "" )
    # This is an additional request.
    req = fmt("(%s)", req);
  if ( full )
    req = fmt("%s (/W)", req);

  local msg = fmt("%s > %s %s",
                  c$mid$orig_h,
                  c$mid$resp_h,
                  req);
  if ( c$hot > 0 )
    log fmt("finger: %s", msg);
  print finger_log,
    fmt("%.6f %s", c$start_time, msg);

  c$addl = c$addl == "" ?
    req : fmt("%s, %s", c$addl, req);
Discussion: Benefits of 3 Layers

- Allows integration of fast and slow (expensive policy) asynchronous executions.
Discussion

- What do you like about Bro design?
- What are missing in Bro?
Outline

- Admin and recap
- Cloud data center (CDC) applications/services
- Network functions (carrier cloud)
  - overview
  - network function programming
    - click (lower layer network function processing)
    - clickNP (click w/ hardware acceleration)
    - bro (L4-L7 middlebox programming)
    - Netbricks
NetBricks: Motivation

- Click allows NF development by assembling various modules
- Modules support only limited customization, through parameters, and developers often need to implement & optimize new modules
- NetBricks goal: identify more explicit, generic, reusable NF structures?
NetBricks Abstractions

Packet Processing

Byte Stream

Control Flow

State
### Packet Processing Abstraction

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Process/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parse</td>
<td>Header type and packet structure</td>
<td>Parses the payload using header type and pushes resulting headers onto stack, removes the header bytes from payload</td>
</tr>
<tr>
<td>Deparse</td>
<td>-</td>
<td>Pops bottom most header back to payload</td>
</tr>
<tr>
<td>Transform</td>
<td>Packet structure, UDF</td>
<td>Modifies header/payload as per UDF</td>
</tr>
<tr>
<td>Filter</td>
<td>Packet, UDF</td>
<td>Allows packets meeting some criteria (as defined by UDF) to be dropped</td>
</tr>
</tbody>
</table>
Example: Decrement TTL

```rust
pub fn ttl_nf<T: 'static + NbNode>(input: T) -> CompositionNode {
    input.parse::<MacHeader>()
      .parse::<IpHeader>()
      .transform(box | pkt | {
        let ttl = pkt.hdr().ttl() - 1;
        pkt.mut_hdr().set_ttl(ttl);
      })
      .filter(box | pkt | {
        pkt.hdr().ttl() != 0
      })
      .compose()
}
```

**Listing 1:** NetBricks NF that decrements TTL, dropping packets with TTL=0.

```rust
// cfg is configuration including
// the set of ports to use.
let ctx = NetbricksContext::from_cfg(cfg);
ctxqueues.map(|p| ttl_nf(p).send(p));
```

**Listing 2:** Operator code for using the NF in Listing 1
# Bytestream Processing Abstraction

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Process/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Window</strong></td>
<td>Window size, Timeout, sliding increment, stream UDF</td>
<td>Waits till timeout or window size packets collected. Operates on available bytes. Responsible for receiving, reordering and buffering packets to reconstruct TCP stream.</td>
</tr>
<tr>
<td><strong>Packetize</strong></td>
<td>Packet structure</td>
<td>Given header stack and byte array, converts data into packets with appropriate headers attached</td>
</tr>
</tbody>
</table>

---

**Notes:**
- Window size, Timeout, sliding increment, stream UDF
- Waits till timeout or window size packets collected. Operates on available bytes. Responsible for receiving, reordering and buffering packets to reconstruct TCP stream.
- Packet structure
- Given header stack and byte array, converts data into packets with appropriate headers attached
## Control Flow Abstractions

<table>
<thead>
<tr>
<th>Operation</th>
<th>Input</th>
<th>Process/Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Group By</td>
<td>Packet, No. of target groups, packet based UDF</td>
<td>Branch control flow within NF or across NF chains. The UDF function returns the ID of the group that the packet will go to</td>
</tr>
<tr>
<td>Shuffle</td>
<td>Packet</td>
<td>Similar to Group By, except that #target groups is based on #active cores. Shuffle outputs are processed on other cores</td>
</tr>
<tr>
<td>Merge</td>
<td>Packets from different branches</td>
<td>A single group of packets</td>
</tr>
</tbody>
</table>
Shuffle

Spread packets across cores for scaling

Mux

Input  Counter  Demux  Output

Core 1  Core 2  Core 3  Core 4
Example: Maglev

- Maglev: Load balancer from Google (NSDI'16).

```
1 pub fn maglev_nf<T: 'static + NbNode>(
2     input: T
3     backends: &[str],
4     ctx: nb_ctx,
5     lut_size: usize)
6     -> Vec<CompositionNode> {
7 let backend_ct = backends.len();
8 let lookup_table =
9     Maglev::new_lut(ctx,
10     backends,
11     lut_size);
12 let mut flow_cache =
13     BoundedConsistencyMap::<usize, usize>::new();
14
15 let groups =
16     input.shuffle(BuiltInShuffle::flow)
17         .parse::<MacHeader>()
18         .group_by((backend_ct, ctx,
19         box move |pkt| {
20         let hash =
21             ipv4_flow_hash(pkt, 0);
22         let backend_group =
23             flow_cache.entry(hash)
24                 .or_insert_with(|| {
25                     lookup_table.lookup(hash)));
26         backend_group
27         }));
28     groups.iter().map(|g| g.compose()).collect()
29 }
```

Summary: NetBricks Abstractions

Packet Processing
- Parse/Deparse
- Transform
- Filter

Control Flow
- Group By
- Shuffle
- Merge

Byte Stream
- Window
- Packetize

State
- Bounded
- Consistency

UDF: user defined function
Outline

- Admin and recap
- Network functions (carrier cloud)
  - overview
  - network function: programming
    - click (lower layer network function processing)
    - clickNP (click w/ hardware acceleration)
    - bro (L4-L7 middlebox programming)
    - Netbricks (high-level programming structures)
  - network function composition
Setting

- NFs can be placed
  - On the same device
    - Intra-device composition
  - Distributed on different devices
    - Inter-device composition
Traditional approaches are to use VMs or containers, but during network I/O packets must cross a h/w memory isolation boundary, which needs a context switch/syscall, which incurs significant overheads.
Penalty of Isolation Composition
Offline Read

- NetBricks: using type system to enforce intra-device composition
Outline

- Admin and recap
- Network functions (carrier cloud)
  - overview
  - network function: programming
    - click (lower layer network function processing)
    - clickNP (click w/ hardware acceleration)
    - bro (L4-L7 middlebox programming)
    - Netbricks
  - network function composition
    - intra-device composition
    - inter-device (network-wide) composition
Challenge: Composition Ambiguity

Policy Chain: *

Firewall → IDS → Proxy

Oops! Forward Pkt to IDS or Dst?

“Loops”
Traditional flow rules may not suffice!
Challenge: Resource Constraints

where to split?

Firewall

Proxy

IDS1 = 50%

IDS2 = 50%

Is there enough resource to set up "feasible" forwarding rules?
Challenge: Dynamic Modifications

User1: Proxy $\rightarrow$ Firewall
User2: Proxy

Proxy may modify flows

Are forwarding rules at S2 correct?
Outline

- Admin and recap
- Cloud data center (CDC) applications/services
- Network functions (carrier cloud)
  - overview
  - network function: programming
    - click (lower layer network function processing)
    - clickNP (click w/ hardware acceleration)
    - bro (L4-L7 middlebox programming)
    - Netbricks
  - network function composition
    - challenges
    - SIMPLE
Handling Composition Ambiguity

Policy Chain: *

Firewall → IDS → Proxy

Idea: Distinguish different instances of the same packet
Handling Resource Constraints

- Topology & Traffic
- Middlebox Capacity + Footprints
- Switch TCAM
- Policy Spec

Resource Manager

Optimal & Feasible load balancing
Approach: Decompose Optimization: Offline + Online Steps

Policy Spec

Enumerate Physical Sequences

Rule Model

Prune for Feasible Configs

Offline Pruning

Traffic Matrix Mbox Capacity

LP with PrunedSet

Network Topology

Online Load Balancing
Enumerating Physical Sequences

<table>
<thead>
<tr>
<th>Physical Sequence</th>
<th>Physical Sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>FW1-IDS1-Proxy1</td>
<td>S1 S2 FW1 S2 S4 S5 IDS1 S5 S4 S2 Proxy1 S2 S4 S5 S6</td>
</tr>
<tr>
<td>FW2-IDS1-Proxy1</td>
<td>S1 S3 FW2 S3 S5 IDS1 S5 S4 S2 Proxy1 S2 S4 S5 S6</td>
</tr>
</tbody>
</table>
Handling Modifications

Correlate flows → Install rules

Payload Similarity

User 1: Proxy → Firewall
User 2: Proxy
Benefits: Load balancing

4-7X better load balancing and near optimal
A More Complete NF Framework

- placement (which NF runs where)
- elastic scaling (adapting the number of NF instances and balancing load across them)
- service composition
- resource isolation
- fault-tolerance
- energy management
- Monitoring

Offline read: the E2 paper.
Question to Think About

- How to design an NFV based network architecture for 5G?