Network Program Synthesis

Lecture 23, Computer Networks (198:552)
Fall 2019
Routing Algorithm

4.1 - OVERVIEW OF NETWORK LAYER 3

In this example, a routing algorithm runs in each and every router, and both forwarding and routing functions are contained within a router. As we'll see in Sections 5.3 and 5.4, the routing algorithm function in one router communicates with the routing algorithm function in other routers to compute the values for its forwarding table. How is this communication performed? By exchanging routing messages containing routing information according to a routing protocol! We'll cover routing algorithms and protocols in Sections 5.2 through 5.4.

The distinct and different purposes of the forwarding and routing functions can be further illustrated by considering the hypothetical (and unrealistic, but technically feasible) case of a network in which all forwarding tables are configured directly by human network operators physically present at the routers. In this case, no routing protocols would be required! Of course, the human operators would need to interact with each other to ensure that the forwarding tables were configured in such a way that packets reached their intended destinations. It's also likely that human configuration would be more error-prone and much slower to respond to changes in the network topology than a routing protocol. We're thus fortunate that all networks have both a forwarding and a routing function!
Review: RMT router data plane

Parser Program

```
parser parse_ethernet {
  extract(ethernet);
  return switch(ethernet.ethertype) {
    0x8100 : parse_vlan_tag;
    0x0800 : parse_ipv4;
    0x8847 : parse_mpls;
    default: ingress;
  }
}
```

Header and Data Declarations

```
header_type ethernet_t  { ... }
header_type 12_metadata_t { ... }

header ethernet_t ethernet;
header vlan_tag_t vlan_tag[2];
metadata 12_metadata_t 12_meta;
```

Tables and Control Flow

```
table port_table { ... }
control ingress {
  apply(port_table);
  if (l2_meta.vlan_tags == 0) {
    process_assign_vlan();
  }
}
```
Review: Verification

for all M, does N satisfy P?

Sequence of messages:
- Packets,
- Routing protocol
- Link failures

Network representation:
- Data plane
- Control plane

Property of interest:
- Loop freedom
- Blackholes
- Equivalence
- Many complex props...

Decision Procedure: An algorithm that answers yes/no
Review: Different guises, Similar question

- **Verification**: for all M, does N satisfy P?
- **Testing**: For the given M, does N satisfy P?
- **Synthesis**: Given P, can you produce an N that satisfies it
  - For a given set of M? (including for all M)
- Let N’ be another network representation
- **Equivalence checking**: For all M, do N and N’ behave in the same way with respect to P?, i.e.,
  - i.e., either both satisfy P or both violate it
Stateful data plane programs

A case study in programming networks
Stateful processing

• Action on a packet depends on previously seen packets
• Example: send every 100\textsuperscript{th} packet to a measurement server
• Example: Track DNS TTL changes

• Example: if (pkt.field > max) { counter++; max = pkt.field }

• Example: Flowlet-based load balancing
Flowlet load balancing

• Consider the time of arrival of the current packet and the last packet of the same flow

• If current packet arrives threshold later than the last packet did, consider rerouting the packet to balance load
  • Otherwise, use the same path as before
  • Q: Why?

if (pkt.ts − saved_ts > threshold) {
    pkt.outport = new_random(); saved_port = pkt.outport;
} else pkt.outport = saved_port;
saved_ts = pkt.ts;
How would you implement Flowlet LB?

• Recall that pipeline is clocked at 1 GHz
• i.e., a new packet admitted every 1 ns
• Where is the processing? Where is the state?
• How about interactions across packets?
Abstraction: Packet transaction

- A piece of code along with state that runs to completion on each packet before processing the next:
  - C-like domain-specific language, Domino [sigcomm’16]

- Assume every packet encounters the same set of actions
  - i.e., the match on each stage is the same: superset of our needs

- Implementation must respect abstraction
  - A compiler must enforce atomicity for the implementation
  - Challenge: transaction code may not fit in one pipeline stage!
  - Challenge: state updates must happen within 1 ns
Insight #1: Pipeline the stateless actions

- if (pkt.field1 + pkt.field2 == 10) { counter ++; }

Stateless operations (whose results depend only on the current packet) can execute over multiple stages

Only the stateful operation must run atomically in one pipeline stage
Insight #2: Fit state updates to ALUs

The atoms constitute the switch’s action instruction set. Run under 1 ns
All-or-nothing compilation

Problems with running stateful network programs
Implementing complex policies

• What if you have a very large P4 or Domino program?
  • Example: too many logical tables
    • Not enough physical stages to hold tables
  • Example: logical table keys are too wide
    • Sharing memory across stages leads to paucity in physical stages
  • Example: Action on each packet, e.g., domino pgm, is too large
    • Sharing compute across stages leads to paucity in physical stages

• Solution?
  • Re-circulation
Re-circulation “extends” the pipeline

If you do this for every packet, packet throughput drops by 2X!
Synthesis of stateful data plane programs

Chipmunk
Program synthesis for code generation

Auto-generate a program from a specification by exhaustive search

- Search space represented by partial program
  - Machine code skeletons

- Specification
  - High-level language program

Exhaustive Search Algorithm

SKETCH

- Completed program that satisfies specification

- Return infeasible if there is none

Program synthesis as an enabling technology for pkt-processing compilers
Example: Using SKETCH to do synthesis

int spec(int x) {
    return x + x + x;
}

Specification

int sketch1(int x) implements spec {
    return x * ??;
}

Partial program

int sketch1(int x) implements spec {
    return x * 3;
}

Completed program

int sketch2(int x) implements spec {
    return x + ??;
}

Infeasible partial program

Hole representing unknown constant

int sketch2(int x) implements spec {
    return x + ??;
}
if (count == 10):
    count = 0
    pkt.sample = 1
else:
    count++
    pkt.sample = 0

Program as a packet transaction in Domino

Partial program for PISA simulator

Chipmunk: An Overview

Chipmunk

Machine code for PISA simulator
Partial program representation: Atoms

```c
bit choice = ??;
int constant = ??;
if (choice) {
    x = x + constant;
} else {
    x = x - constant;
}
```
CEGIS: Finding PISA configurations (holes)

Counterexample-Guided Inductive Synthesis
Other apps of synthesis in networking

• Generating control plane rules with desired properties
• Data plane programs for other targets
  • e.g., Network processors (CPUs)
• Program repair and troubleshooting
• Approximation