CS 6114
Course Overview

Nate Foster
Cornell University
Take I
The Internet is an enormous success—it's one of the "wonders" of the modern world.
The Internet is an enormous success—it's one of the "wonders" of the modern world. With a few notable exceptions, networking remains an **engineering discipline**...
Although many thorny problems remain...
Although many thorny problems remain...

...transporting results from theory to practice is typically not straightforward
This started to change ~10 years ago with the emergence of large-scale data centers...
This started to change ~10 years ago with the emergence of large-scale data centers...

- Complexity became unmanageable
- Growing need to deploy new features
- Big players unhappy with market dynamics
Take II
Many programming languages were designed in an age when computers looked like this...
But nowadays, computers look like this...
And applications are built like this...
• Stand-alone
• Centralized
• Sequential
• Stand-alone
• Centralized
• Sequential
- Stand-alone
- Centralized
- Sequential

- Networked
- Distributed
- Concurrent
We need new kinds of abstractions and tools for programming these networked systems!

- Stand-alone
- Centralized
- Sequential
- Networked
- Distributed
- Concurrent
Take III
Network Programming
Network Programming

**Data plane:** forward packets, balances load, implements monitoring, etc.
Control Plane: discovers topology, computes routes, enforces policies, etc.
Network Devices: implement packet processing, buffering, queueing, etc. at line rate
Network Programming Challenges

Networks are *distributed* systems with thousands of interacting nodes.

Networks enforce complex *security* policies that span trust boundaries.

Networks are expected to offer good *performance* with limited resources.
Software-Defined Networking
Conventional Networking
Conventional Networking
Conventional Networking

**Control Plane:** discovers topology, computes routes, manages policy, etc.

**Data plane:** forwards packets, enforces access control, monitors flows, etc.
Software-Defined Networking

1. Separate control plane and data plane
2. Use appropriate unit of abstraction for control plane
Software-Defined Networking

3. Generalize data plane and standardize configuration APIs
Software-Defined Networking

4. Implement control plane on general-purpose machine
Software-Defined Networking

4. Implement control plane on general-purpose machine
SDN Switch

Programmable device that can be used to implement switches, routers, gateways, firewalls, load balancers, etc.

<table>
<thead>
<tr>
<th>Match</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1</td>
<td>Drop</td>
</tr>
<tr>
<td>10.0.0.2</td>
<td>Forward 2</td>
</tr>
<tr>
<td>10.0.0.3</td>
<td>Forward 3</td>
</tr>
<tr>
<td>*</td>
<td>Controller</td>
</tr>
</tbody>
</table>
SDN Switch

Programmable device that can be used to implement switches, routers, gateways, firewalls, load balancers, etc.

Key construct is a flow table comprising a prioritized list of match-action forwarding rules

<table>
<thead>
<tr>
<th>Match</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.0.0.1</td>
<td>Drop</td>
</tr>
<tr>
<td>10.0.0.2</td>
<td>Forward 2</td>
</tr>
<tr>
<td>10.0.0.3</td>
<td>Forward 3</td>
</tr>
<tr>
<td>*</td>
<td>Controller</td>
</tr>
</tbody>
</table>
Switch to controller:
• switch_connected
• switch_disconnected
• port_status
• packet_in

Controller to switch:
• flow_mod
• packet_out
Repeater Hub
Repeater Hub
Repeater Hub

Floods packets out on all ports, except the ingress—i.e., the port it arrived on.
Ethernet Switch
Ethernet Switch

<table>
<thead>
<tr>
<th>Src</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nate</td>
<td>1</td>
</tr>
</tbody>
</table>
Ethernet Switch

<table>
<thead>
<tr>
<th>Src</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nate</td>
<td>1</td>
</tr>
<tr>
<td>Src</td>
<td>Port</td>
</tr>
<tr>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Nate</td>
<td>1</td>
</tr>
</tbody>
</table>
Ethernet Switch

<table>
<thead>
<tr>
<th>Src</th>
<th>Port</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nate</td>
<td>1</td>
</tr>
<tr>
<td>Dexter</td>
<td>2</td>
</tr>
</tbody>
</table>
• Learn locations of hosts in a table
• Flood packets going to unknown hosts; forward packets directly to known hosts
let known_hosts : (dlAddr, portId) Hashtbl.t = Hashtbl.create 101

let learning_packet_in (sw:switchId) (pktIn:packetIn) : unit =
  Hashtbl.add known_hosts pk.dlSrc pktIn.port

let routing_packet_in (sw:switchId) (pktIn : packetIn) : unit =
  try
  let out_port = Hashtbl.find known_hosts pk.dlDst in
  let sd = {match_all with dlDst=Some pk.dlDst; dlSrc=Some pk.dlSrc} in
  let ds = {match_all with dlDst=Some pk.dlSrc; dlSrc=Some pk.dlDst} in
  send_flow_mod sw 0l (add_flow 0 sd [Output out_port]);
  send_flow_mod sw 0l (add_flow 0 ds [Output pktIn.port]);
  send_packet_out sw 0l
    { output_payload = pktIn.input_payload;
      port_id = None;
      apply_actions = [Output out_port] }
  with Not_found ->
  send_packet_out sw 0l
    { output_payload = pktIn.input_payload;
      port_id = None;
      apply_actions = [Flood] }

let packet_in (sw:switchId) _ (pk:packetIn) : unit =
  learning_packet_in sw pk; routing_packet_in sw pk
let known_hosts : (dlAddr, portId) Hashtbl.t = Hashtbl.create 101

let learning_packet_in (sw:switchId) (pktIn:packetIn) : unit =
  Hashtbl.add known_hosts pk.dlSrc pktIn.port

let routing_packet_in (sw:switchId) (pktIn : packetIn) : unit =
  try
    let out_port = Hashtbl.find known_hosts pk.dlDst in
    let sd = {match_all with dlDst=Some pk.dlDst; dlSrc=Some pk.dlSrc} in
    let ds = {match_all with dlDst=Some pk.dlSrc; dlSrc=Some pk.dlDst} in
    send_flow_mod sw 0l (add_flow 0 sd [Output out_port]);
    send_flow_mod sw 0l (add_flow 0 ds [Output pktIn.port]);
    send_packet_out sw 0l
      { output_payload = pktIn.input_payload;
        port_id = None;
        apply_actions = [Output out_port] } with Not_found ->
    send_packet_out sw 0l
      { output_payload = pktIn.input_payload;
        port_id = None;
        apply_actions = [Flood] }
  with Not_found ->
    send_packet_out sw 0l
      { output_payload = pktIn.input_payload;
        port_id = None;
        apply_actions = [Flood] }

let packet_in (sw:switchId) _ (pk:packetIn) : unit =
  learning_packet_in sw pk; routing_packet_in sw pk
let known_hosts : (dlAddr, portId) Hashtbl.t = Hashtbl.create 101

let learning_packet_in (sw:switchId) (pktIn:packetIn) : unit =
  Hashtbl.add known_hosts pkt.dlSrc pktIn.port

let routing_packet_in (sw:switchId) (pktIn : packetIn) : unit =
  try
    let out_port = Hashtbl.find known_hosts pk.dlDst in
    let sd = {match_all with dlDst=Some pk.dlDst; dlSrc=Some pk.dlSrc} in
    let ds = {match_all with dlDst=Some pk.dlSrc; dlSrc=Some pk.dlDst} in
    send_flow_mod sw 01 (add_flow 0 sd [Output out_port]);
    send_flow_mod sw 01 (add_flow 0 ds [Output pktIn.port]);
    send_packet_out sw 01
      { output_payload = pktIn.input_payload;
        port_id = None;
        apply_actions = [Output out_port] }
  with Not_found ->
    send_packet_out sw 01
      { output_payload = pktIn.input_payload;
        port_id = None;
        apply_actions = [Flood] }

let packet_in (sw:switchId) _ (pk:packetIn) : unit =
  learning_packet_in sw pk; routing_packet_in sw pk
let known_hosts : (dlAddr, portId) Hashtbl.t = Hashtbl.create 101

let learning_packet_in (sw:switchId) (pktIn:packetIn) : unit =
  Hashtbl.add known_hosts pk.dlSrc pktIn.port

let routing_packet_in (sw:switchId) (pktIn : packetIn) : unit =
  try
    let out_port = Hashtbl.find known_hosts pk.dlDst in
    let sd = {match_all with dlDst = Some pk.dlDst; dlSrc = Some pk.dlSrc} in
    let ds = {match_all with dlDst = Some pk.DlSrc; dlSrc = Some pk.dlDst} in
    send_flow_mod sw 01 (add_flow 0 sd [Output out_port]);
    send_flow_mod sw 01 (add_flow 0 ds [Output pktIn.port]);
    send_packet_out sw 01
      { output_payload = pktIn.input_payload;
        port_id = None;
        apply_actions = [Output out_port] }
  with Not_found ->
    send_packet_out sw 01
    { output_payload = pktIn.input_payload;
      port_id = None;
      apply_actions = [Flood] }

let packet_in (sw:switchId) _ (pk:packetIn) : unit =
  learning_packet_in sw pk; routing_packet_in sw pk
Learning Switch

let known_hosts : (dlAddr, portId) Hashtbl.t = Hashtbl.create 101

let learning_packet_in (sw:switchId) (pktIn:packetIn) : unit =
    Hashtbl.add known_hosts pk.dlSrc pktIn.port

let routing_packet_in (sw:switchId) (pktIn : packetIn) : unit =
    try
        let out_port = Hashtbl.find known_hosts pk.dlDst in
        let sd = {match_all with dlDst=Some pk.dlDst; dlSrc=Some pk.dlSrc} in
        let ds = {match_all with dlDst=Some pk.dlSrc; dlSrc=Some pk.dlDst} in
        send_flow_mod sw 0l (add_flow 0 sd [Output out_port]);
        send_flow_mod sw 0l (add_flow 0 ds [Output pktIn.port]);
        send_packet_out sw 0l
        { output_payload = pktIn.input_payload;
          port_id = None;
          apply_actions = [Output out_port] } with Not_found ->
            send_packet_out sw 0l
            { output_payload = pktIn.input_payload;
              port_id = None;
              apply_actions = [Flood] }

let packet_in (sw:switchId) _ (pk:packetIn) : unit =
    learning_packet_in sw pk; routing_packet_in sw pk
<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst=Nate</td>
<td>Allow</td>
</tr>
<tr>
<td>dst=Dexter</td>
<td>Deny</td>
</tr>
<tr>
<td>dst=Alexandra</td>
<td>Log</td>
</tr>
</tbody>
</table>
Firewall

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst=Nate</td>
<td>Allow</td>
</tr>
<tr>
<td>dst=Dexter</td>
<td>Deny</td>
</tr>
<tr>
<td>dst=Alexandra</td>
<td>Log</td>
</tr>
</tbody>
</table>
Firewall

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst=Nate</td>
<td>Allow</td>
</tr>
<tr>
<td>dst=Dexter</td>
<td>Deny</td>
</tr>
<tr>
<td>dst=Alexandra</td>
<td>Log</td>
</tr>
<tr>
<td>Pattern</td>
<td>Action</td>
</tr>
<tr>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>dst=Nate</td>
<td>Allow</td>
</tr>
<tr>
<td>dst=Dexter</td>
<td>Deny</td>
</tr>
<tr>
<td>dst=Alexandra</td>
<td>Log</td>
</tr>
</tbody>
</table>
Firewall

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst=Nate</td>
<td>Allow</td>
</tr>
<tr>
<td>dst=Dexter</td>
<td>Deny</td>
</tr>
<tr>
<td>dst=Alexandra</td>
<td>Log</td>
</tr>
</tbody>
</table>
Firewall

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>dst=Nate</td>
<td>Allow</td>
</tr>
<tr>
<td>dst=Dexter</td>
<td>Deny</td>
</tr>
<tr>
<td>dst=Alexandra</td>
<td>Log</td>
</tr>
</tbody>
</table>

Drops (or logs) packets from potentially malicious hosts
<table>
<thead>
<tr>
<th>Dst</th>
<th>Port</th>
<th>Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nate</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Dst</td>
<td>Port</td>
<td>Dist</td>
</tr>
<tr>
<td>-----</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Nate</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
I can reach Alexandra in 2 hop and Dexter in 1 hop

<table>
<thead>
<tr>
<th>Dst</th>
<th>Port</th>
<th>Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nate</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Dst</td>
<td>Port</td>
<td>Dist</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Nate</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Alexandra</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Dexter</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
I can reach Alexandra in 0 hops and Dexter in 1 hop

<table>
<thead>
<tr>
<th>Dst</th>
<th>Port</th>
<th>Dist</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nate</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Alexandra</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Dexter</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>Dst</td>
<td>Port</td>
<td>Dist</td>
</tr>
<tr>
<td>----------</td>
<td>------</td>
<td>------</td>
</tr>
<tr>
<td>Nate</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>Alexandra</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>Dexter</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
Forwards packets using tables that are computed and maintained by a distributed routing protocol.
Another History Lesson...
Another History Lesson...
Another History Lesson...
Operating Systems
Programming Languages

Operating Systems
A hardware model describes behavior in terms of concepts like pipelines of lookup tables.
A hardware model describes behavior in terms of concepts like pipelines of lookup tables.

A language model describes behavior in terms of concepts like mathematical functions on packets.
Administrativa
Staff

**Background**
- BA Williams College
- MPhil Cambridge University
- PhD University of Pennsylvania
- Postdoc Princeton University

**Office Hours**
Gates 432
Tuesdays 1-2pm

**Contact**
jnfoster@cs.cornell.edu
http://www.cs.cornell.edu/~jnfoster/
Schedule

Lecture
• Tuesdays & Thursdays
• Hollister B14
• 8:40am-9:55am

Expectations
• Attend!
• Read papers
• Contribute to discussions

Breakfast
• I will buy bagels and coffee...
• ...if you pick it up from CTB ;-)
Coursework and Grades

Participation (20%)
- Attend lectures
- Contribute to discussions

Problem Sets (3 x 20% each)
- Mostly programming assignments, with some pencil-and-paper problems mixed in.
- I'll supply a virtual machine, starter code, etc.

Mini Project (20%)
- We'll build a basic router from scratch in P4
- You'll extend this router with some advanced feature (that you choose)
Academic Integrity

• Cheating is a lot easier to detect than you might imagine (I use automated tools to find similarities between submissions)

• Violations are unpleasant and painful for everyone involved

• To avoid pressure, start assignments early

• A simple guideline: provide attribution for everything you obtain from an outside source

• If you run into difficulty, speak up!
Rough Outline

Introduction

NetKAT Language

P4 Language

Advanced Topics
Rough Outline

- Introduction
- NetKAT Language
- P4 Language
- Advanced Topics

Guest Lectures!
Guest Lecturers

Andrew Ferguson
Google

Chang Kim
Barefoot

Tim Nelson
Brown

Pavol Cerny
Colorado
Themes

• **Language Design**: what abstractions should we use to program networks?

• **Compilation**: how do we map those abstractions down to effect packet-processing devices?

• **Semantics**: what mathematical objects do network programs denote?

• **Verification**: how do we check that a network behaves as expected?

• **Applications**: what innovative features can we implement with programmable networks?