CS434/534: Topics in Network Systems

High-Level Programming for Programmable Networks: Compiler and Network OS

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
- Datapath programming
  - Reactive, high level logic driven user programming
  - Reactive, high level automatic programming
  - Proactive, high level automatic programming (aka compiler)
- Controller software framework (network OS) supporting programmable networks
  - architecture
  - data model and operations: OpenDaylight as an example
  - distributed data store
    - overview
    - basic Paxos
Recap: Basic Ideas of Automatic Datapath Generation

- **Insight**
  - Although the decision function $f$ does **not** specify how flow tables are configured, if for a given decision (e.g., drop), one can know the dependency on the input ($pkt$ attributes) of the decision, one can construct the flow table.

- **Requirement**
  - Program $f$ uses a library to access $pkt$ attributes and environment state variables, to track dependency access.

- **Basic data structures and algorithms**
  - Trace tree: tracking dependency on $pkt$ attributes
  - Dependency table: tracking dependency on environment states

- **Problems**
  - Single table; latency
Recap: Proactive Compiler

- Proactive, fully generate multi-table datapath pipeline
- Basic idea: A function $f$ consists of a sequence of instructions $I_1, I_2, \ldots, I_n$, and one can (conceptually) consider each instruction in the dataflow as a table

```java
Map<MAC, Switch> hostTable;
0. Route onPacketIn(Packet p) {
1.   Switch srcSw = hostTable.get(p.ethSrc());
2.   Switch dstSw = hostTable.get(p.ethDst());
3.   Route aRoutes = AllPairRoutes();
4.   return aRoutes.get(srcSw, dstSw);
}
```

Ex: instruction data flow?
Recap: Map Each Instruction to a Flow Table

Map<MAC, Switch> hostTable; // {11->sw1, 22->sw1, 33->sw2,others swu}
Map<MAC, Cond> condTable; // {11->S, 22->C, 33->C, others UK}
Set openPorts; // {22, 53, 80, 8000, 8080, 9090}

Route onPacketIn(Packet p) {
L1    if (p.dstPort < 1025) { // privileged port, SW_FW
L2:    dstSw = SW_FW;
L3:    dstCond = V;
L4:    } else if (openPorts.contains(p.dstPort)) { // spec user
L5:    dstSw = hostTable[p.dstMac];
L6:    dstCond = condTable[p.dstMac];
L7:    } else {
L8:        return Drop; }
L9:    // All pairs paths for all dstConds
L10:   allPaths = AllPairsCond.execute( ... );
L11:   srcSw = hostTable[p.srcMac];
L12:   return allPaths[dstCond].get(srcSw,dstSw); }

Recap: High-Level Pipeline

L1: p.dstPort < 1025

- Y: L2: dstSw
  - Y: L3: dstCond
    - Y: L9: allPairPaths
      - L11: egress
    - N: L10: srcSw
  - N: L5: dstSw
    - N: L5: dstCond
    - N: L8: Drop

- N: L4: openPort.contains [ p.dstPort ]
  - Y: L5: dstSw
    - N: L5: dstCond
Outline

- Admin and recap
- Datapath programming
  - Reactive, high level logic driven user programming
  - Reactive, high level automatic programming
  - Proactive, high level automatic programming (aka compiler)
    - instruction analysis (symbolic analysis)
### Instruction => Table

#### Naïve enumeration

<table>
<thead>
<tr>
<th>p.dstPort</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reg(_g1) = true jump L2</td>
</tr>
<tr>
<td>2</td>
<td>Reg(_g1) = false jump L2</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>2(^{16}-1)</td>
<td>Reg(_g1) = false jump L2</td>
</tr>
</tbody>
</table>

#### Compact encoding

<table>
<thead>
<tr>
<th>Pri</th>
<th>p.dstPort</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0000 00xx</td>
<td>reg(_g1)=true jump L2</td>
</tr>
<tr>
<td></td>
<td>xxxx xxxx</td>
<td>// 0-1023</td>
</tr>
<tr>
<td>2</td>
<td>0000 0100</td>
<td>reg(_g1)=true jump L2</td>
</tr>
<tr>
<td></td>
<td>0000 0000</td>
<td>// 1024</td>
</tr>
<tr>
<td>1</td>
<td>xxxx xxxx</td>
<td>reg(_g1)=false jump L2</td>
</tr>
<tr>
<td></td>
<td>xxxx xxxx</td>
<td></td>
</tr>
</tbody>
</table>

**Observation:** There are compact-mapable (CM) instruction: Despite large input domain, small # of rules to express.

**L1.** \( g1 = p\.dstPort < 1025; \)
More Example of CM Statements

\[ y = p.\text{ethSrc} == p.\text{ethDst} \]

| \( p.\text{ethSrc} \) & \( p.\text{ethDst} \) & Action |
|-----------------|----------|--------|
| 0               | 0        | y=true |
| 0               | 1        | y=false|
| \( \ldots \)    | \( \ldots \) | \( \ldots \) |
| \( 2^{48} - 1 \) | \( 2^{48} - 1 \) | y=true |

| \( p.\text{ethSrc} \) & \( p.\text{ethDst} \) & Action |
|-----------------|----------|--------|
| \( ***0 \)      | \( ***1 \) | false  |
| \( ***1 \)      | \( ***0 \) | false  |
| \( **0* \)      | \( **1* \) | false  |
| \( **1* \)      | \( **0* \) | false  |
| \( \ldots \)    |          |        |
| \( * \)         |          | true   |

Offline exercise: \( y = p.\text{ethSrc} \neq p.\text{ethDst} \)
Observation: Despite large input domain, system state provides (small) potential values.

Naïve enumeration

<table>
<thead>
<tr>
<th>p.dstPort</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Reg(_{g2}) = false; jump L6</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>22</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>2(^{16}-1)</td>
<td>Reg(_{g1}) = false; jump L6</td>
</tr>
</tbody>
</table>

Compact encoding using system state

<table>
<thead>
<tr>
<th>Pri</th>
<th>p.dstPort</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>22</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>2</td>
<td>53</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>2</td>
<td>8000</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>2</td>
<td>8080</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>2</td>
<td>9090</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>Reg(_{g2}) = false; jump L6</td>
</tr>
</tbody>
</table>

L5. g2 = openPorts.contains(p.dstPort);
   // openPorts = {22, 53, 80, 8000, 8080, 9090};
Q: What are some instructions that are fundamentally hard to express by lookups?
Non-Compact Instruction

- srcMac % n, where n is not a power of 2 has no compact representation.
Map<MAC, Switch> hostTable; // {11->sw1, 22->sw1, 33->sw2,others swu}
Map<MAC, Cond> condTable; // {11->S, 22->C, others UK}
Set openPorts; // {22, 53, 80, 8000, 8080, 9090}

Route onPacketIn(Packet p) {
L1 if (p.dstPort < 1025) { // privileged port, SW_FW
L2: dstSw = SW_FW;
L3: dstCond = V;
L4: } else if (openPorts.contains(p.dstPort)) { // spec user
L5: dstSw = hostTable[p.dstMac];
L6: dstCond = condTable[p.dstMac];
L7: } else {
L8: return Drop; }
// All pairs paths for all dstConds
L9: allPaths = AllPairsCond.execute( ... );
L10: srcSw = hostTable[p.srcMac];
L11: return allPaths[dstCond].get(srcSw,dstSw); }

Ex: State and Issue After Symbolic Analysis+Direct System State Plug in
Summary: Progress & Remaining Problems

Symbolic analysis/direct state-plugin cannot generate tables for some instructions.

Tables computed in isolation can have unnecessary rules.

<table>
<thead>
<tr>
<th>Pri</th>
<th>p.dstPort</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>22</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>2</td>
<td>53</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>2</td>
<td>80</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>2</td>
<td>8000</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>2</td>
<td>8080</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>2</td>
<td>9090</td>
<td>Reg(_{g2}) = true; jump L6</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>Reg(_{g2}) = false; jump L6</td>
</tr>
</tbody>
</table>
Discussion: How to Handle the Problems?

Map<MAC, Switch> hostTable; // {11→sw1, 22→sw1, 33→sw2,others swu}
Map<MAC, Cond> condTable; // {11→S, 22→C, 33→C, others UK}
Set openPorts; // {22, 53, 80, 8000, 8080, 9090}

Route onPacketIn(Packet p) {
    L1    g1 = p.dstPort < 1025;
    L2    if ( g1 ) {
    L3:    dstSw   = SW_FW;
    L4:    dstCond = S;
    }    else {
    L5:    g2 = openPorts.contains(p.dstPort); // not compact
    L6:    if ( g2 ) {
    L7:    dstSw   = hostTable[p.dstMac];
    L8:    dstCond = condTable[p.dstMac];
    }    else {
    L9:    return Drop; }
    }    }
    // All pairs paths for all dstConds
L10:   allPaths = AllPairsCond.execute( ... );
L11:   srcSw   = hostTable[p.srcMac];
L12:   return allPaths[dstCond].get(srcSw,dstSw); // compact inputs?}
Observation

- Both problems require computing valid inputs to instructions:
  - \( g2 = \text{openPorts.contains}(p.\text{dstPort}); \)
    - If we know valid inputs to dstPort (<1025), we can prune the extra rules (22, 53, 80)
  - \( \text{allPaths}[\text{dstCond}].\text{get}(\text{srcSw},\text{dstSw}); \)
    - If we know the valid values of (dstCond, srcSw, dstSw) reaching this instruction, we can compute a compact flow table for it (not all potential values).
Discuss: How to Compute Valid Inputs (Reachable States) to Each Instruction

Design 1: takes advantage of CM

- Inputs to L4:
  - |dstSw| from L1 x |dstCond| from L2

Q: Is this the best?
Naïve Approach can Compute Non-Valid Inputs

Map<MAC, Switch> hostTable; // {11->sw1, 22->sw2, 33->sw2, others swu}
Map<MAC, Cond> condTable; // {11->S, 22->C, 33->C, others UK}

0. Route onPacketIn(Packet p) {
L1. Switch dstSw = hostTable.get( p.dstMac() );
L2. Cond dstCond = condTable.get( p.dstMac() );
L3. Route aRoutes = AllPairCondRoutes();
L4. return aRoutes.get(dstSw, dstCond);

- L1: \( m_1 \to s_1, m_2 \to s_2, \ldots, m_n \to s_n \)
- L2: \( m_1 \to c_1, m_2 \to c_2, \ldots, m_n \to c_n \)
- L4 sees only \( n \) (sw, cond) combinations, not \( n^2 \) combinations
  - This is called dependent inputs
Basic Approach: The FlowExplore Algorithm

- Computes (I, state) to each instruction, using flow tables as edges
Map<MAC, Switch> hostTable; // {11->sw1, 22->sw2, 33->sw2, others swu}
Map<MAC, Cond> condTable; // {11->S, 22->C, 33->C, others UK}

0. Route onPacketIn(Packet p) {
    L1. Switch dstSw = hostTable.get(p.dstMac());
    L2. Cond dstCond = condTable.get(p.dstMac());
    L3. Route aRoutes = AllPairCondRoutes();
    L4. return aRoutes.get(dstSw, dstCond);
}
0. Route onPacketIn(Packet p) {
    L1. Switch dstSw = hostTable.get(p.dstMac());
    L2. Cond dstCond = condTable.get(p.dstMac());
    L3. Route aRoutes = AllPairCondRoutes();
    L4. return aRoutes.get(dstSw, dstCond);
}

dstMac is already part of state, no need to use flow table, use state propagation
Map<MAC, Switch> hostTable; // {11->sw1, 22->sw2, 33->sw2, others swu}
Map<MAC, Cond> condTable; // {11->S, 22->C, 33->C, others UK}

0. Route onPacketIn(Packet p) {
L1. Switch dstSw = hostTable.get(p.dstMac());
L2. Cond dstCond = condTable.get(p.dstMac());
L3. Route aRoutes = AllPairCondRoutes();
L4. return aRoutes.get(dstSw, dstCond);
}

dstMac is a dead variable and hence should be removed.
0. Route onPacketIn(Packet p) {
L1. Switch dstSw = hostTable.get(p.dstMac());
L2. Cond dstCond = condTable.get(p.dstMac());
L3. Route aRoutes = AllPairCondRoutes();
L4. return aRoutes.get(dstSw, dstCond);
}

Map<MAC, Switch> hostTable; // {11->sw1, 22->sw2, 33->sw2, others swu}
Map<MAC, Cond> condTable; // {11->S, 22->C, 33->C, others UK}

Same state and hence should merge.

<table>
<thead>
<tr>
<th>Pri</th>
<th>p.dstMac</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11</td>
<td>dstSw=sw1</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>dstSw=sw2</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>dstSw=sw2</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>dstSw=swu</td>
</tr>
</tbody>
</table>
0. Route onPacketIn(Packet p) {
L1. Switch dstSw = hostTable.get(p.dstMac());
L2. Cond dstCond = condTable.get(p.dstMac());
L3. Route aRoutes = AllPairCondRoutes();
L4. return aRoutes.get(dstSw, dstCond);
0. Route onPacketIn(Packet p) {
   L1. Switch dstSw = hostTable.get(p.dstMac());
   L2. Cond dstCond = condTable.get(p.dstMac());
   L3. Route aRoutes = AllPairCondRoutes();
   L4. return aRoutes.get(dstSw, dstCond);
}

Map<MAC, Switch> hostTable; // {11->sw1, 22->sw2, 33->sw2, others swu}
Map<MAC, Cond> condTable; // {11->S, 22->C, 33->C, others UK}

<table>
<thead>
<tr>
<th>Pri</th>
<th>p.dstMac</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11</td>
<td>dstCond=S</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>dstCond=C</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>dstCond=C</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>dstCond=UK</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pri</th>
<th>p.dstMac</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11</td>
<td>dstSw=sw1</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>dstSw=sw2</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>dstSw=sw2</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>dstSw=swu</td>
</tr>
</tbody>
</table>

dstMac is binded, Is there an ancestor binds to dstMac=11?  Yes. Blocked
Map<MAC, Switch> hostTable; // {11->sw1, 22->sw2, 33->sw2, others swu}
Map<MAC, Cond> condTable; // {11->S, 22->C, 33->C, others UK}

0. Route onPacketIn(Packet p) {
    L1. Switch dstSw = hostTable.get(p.dstMac());
    L2. Cond dstCond = condTable.get(p.dstMac());
    L3. Route aRoutes = AllPairCondRoutes();
    L4. return aRoutes.get(dstSw, dstCond);
}

<table>
<thead>
<tr>
<th>Pri</th>
<th>p.dstMac</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11</td>
<td>dstCond=S</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>dstCond=C</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>dstCond=C</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>dstCond=UK</td>
</tr>
</tbody>
</table>

- L1, EMPTY
- L2, dstSw=sw1, dstMac=11
  - L4, dstSw=sw1, dstCond=S
    - L∞, aRoutes(sw1, S)
- L2, dstSw=sw2, dstMac=22
  - L4, dstSw=sw2, dstCond=C
    - L∞, aRoutes(sw2, C)
- L2, dstSw=sw2, dstMac=33
  - L4, dstSw=sw2, dstCond=C
    - L∞, aRoutes(sw2, C)
- L2, dstSw=swu, dstMac=* 
  - L4, dstSw=swu, dstCond=UK
    - L∞, aRoutes(swu, UK)
Map<MAC, Switch> hostTable; // {11->sw1, 22->sw2, 33->sw2, others swu}
Map<MAC, Cond> condTable; // {11->S, 22->C, 33->C, others UK}

0. Route onPacketIn(Packet p) {
    L1. Switch dstSw = hostTable.get(p.dstMac());
    L2. Cond dstCond = condTable.get(p.dstMac());
    L3. Route aRoutes = AllPairCondRoutes();
    L4. return aRoutes.get(dstSw, dstCond);
}

<table>
<thead>
<tr>
<th>Pri</th>
<th>p.dstMac</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>11</td>
<td>dstCond=S</td>
</tr>
<tr>
<td>2</td>
<td>22</td>
<td>dstCond=C</td>
</tr>
<tr>
<td>2</td>
<td>33</td>
<td>dstCond=C</td>
</tr>
<tr>
<td>1</td>
<td>*</td>
<td>dstCond=UK</td>
</tr>
</tbody>
</table>

Q: How to obtain the flow table of each instruction?
Assignment

- Compute the reachable inputs of the complete example program
Issues of PIT Pipelines

This example needs 10 tables, but a hardware may support only 2.

```java
Map<MAC, Switch> hostTable; // {11->sw1, 22->sw1, 33->sw2, others swu}
Map<MAC, Cond> condTable; // {11->S, 22->C, 33->C, others UK}
Set openPorts; // {22, 53, 80, 8000, 8080, 9090}

Route onPacketIn(Packet p) {
    L1  if (p.dstPort < 1025) { // privileged port, SW_FW
    L2:  dstSw = SW_FW;
    L3:  dstCond = S;
    L4:  } else if (openPorts.contains(p.dstPort)) { // spec user
    L5:  dstSw = hostTable[p.dstMac];
    L6:  dstCond = condTable[p.dstMac];
    L7:  } else {
    L8:   return Drop; }
    // All pairs paths for all dstConds
    L9:  allPaths = AllPairsCond.execute( ... );
    L10: srcSw = hostTable[p.srcMac];
    L11: return allPaths[dstCond].get(srcSw,dstSw); }
```
Algorithm Framework

Assume \( P \) is the set of pipelines eval\'d

\[
p_0 = \text{empty}; \text{resource}_0 = \text{INF}
\]

foreach \( p \) in \( P \)

\[
\text{if } \text{hw\_oracle}(p) \&\& \text{resource}(p) < \text{resource}_0
\]

\[
p_0 = p; \text{resource}_0 = \text{resource}(p)
\]

Q: How many pipelines in \( P \) if \( P \) is generated from full enumeration. Assume \( N \) instructions and \( \leq M \) tables?
$g_1 = \text{dstPort} < 1025$

$g_2 = \text{openPorts. contain(dstPort)}$

$\text{dstSw} = \text{SW_FW}$

$\text{dstCond} = V$

$\text{egress} = \text{Drop}$

$\phi(\text{dstSw})$

$\phi(\text{dstCond})$

$\text{srcSw} = \text{hostTable(srcMac)}$

$\text{egress} = f(\text{dstCond}, \text{dstSw}, \text{srcSw})$
Insight: Suppose we already have a table $T$ matching on $(\text{dstPort}, \text{dstMac})$. What should be handled by $T$?
Discussion and Next Step

- The compiler approach
  - Pros?
  - Cons?

- For those who are looking for a project, introduce flow algebra (vs relational algebra) to compute propagation and optimal design (a possible NSDI'20 submission)
Outline

- Admin and recap
- Datapath programming
- Controller framework supporting programmable networks
  - often referred to as network OS (NOS)
Discussion

- What components do we include in a software architecture for network OS (NOS) to support programmable networks?
ONOS System Architecture

Northbound - Application Intent Framework
(policy enforcement, conflict resolution)

Distributed Core
(scalability, availability, performance, persistence)

Southbound
(discover, observe, program, configure)

OpenFlow
(pluggable, extensible)

More API in future release

Source: ON.LAB
ONOS Software Architecture
OpenDaylight System Architecture

OpenDaylight Software Architecture

Network Devices
- Protocol Plugin
- Protocol Plugin
  - Config Subsystem
- NETCONF SERVER

Applications
- RESTCONF
- App/Service Plugin
- App/Service Plugin

Controller
- Model-Driven SAL (MD-SAL)
  - Messaging
  - Data Store
  - Remote Controller Instance
  - Remote Controller Instance
A More Abstract Architecture

Key goal: provide applications with high-level views and make the views highly available (e.g., 99.99%), scalable.

Key component - data store:
- Data model
- Data access model
- Data store availability

logically centralized data store

Network View
Service/Policy
NE Datapath
NE Datapath

Program
Outline

- Admin and recap
- Datapath programming
- Controller framework supporting programmable networks
  - architecture
  - data model and operations: OpenDaylight as an example
Data Model

- Not a fixed model/view, but models described by Yang and loaded into the system by OSGi
- Yang: a modeling language describing
  - data as a tree
  - RPCs and notifications
  - Based on RFC 6020
Yang Data Tree

Data Tree: Details

- Whole data store as a tree
- Two Logical Data Stores
  - config
  - operational
- InstanceIdentifier:
  - pointer to a node
Data Store Operations

- Operations must be included in transactions

```java
ReadWriteTransaction transaction = dataBroker.newReadWriteTransaction();
Optional<Node> nodeOptional;
nodeOptional = transaction.read(
    LogicalDataStore.OPERATIONAL,
    n1InstanceIdentifier);
transaction.put(
    LogicalDataStore.CONFIG,
    n2InstanceIdentifier,
    topologyNodeBuilder.build());
transaction.delete(
    LogicalDataStore.CONFIG,
    n3InstanceIdentifier);
CheckedFuture future;
future = transaction.submit();
```
Outline

- Admin and recap
- Datapath programming
- Controller framework supporting programmable networks
  - architecture
  - data model and operations: OpenDaylight as an example
  - distributed network OS
Discussion: How may a distributed NetOS look like?
A Common Approach: Replicated Log

- Replicated operations log => replicated state machine
  - All servers execute same commands in same order
- Consensus module ensures proper log replication
- System makes progress as long as any majority of servers are up
- Failure model: fail-stop (not Byzantine), delayed/lost messages
Two general approaches to consensus:

- **Symmetric, leader-less:**
  - All servers have equal roles
  - Clients can contact any server
  - We will see Basic Paxos as an example

- **Asymmetric, leader-based:**
  - At any given time, one server is in charge, others accept its decisions
  - Clients communicate with the leader
  - We will see Raft, which uses a leader
  - OpenDaylight is based on Raft
Warning

- Leslie Lamport, 1989
- Nearly synonymous with consensus

“The dirty little secret of the NSDI community is that at most five people really, truly understand every part of Paxos ;-).”
—NSDI reviewer

“There are significant gaps between the description of the Paxos algorithm and the needs of a real-world system...the final system will be based on an unproven protocol.”
—Chubby authors
Prepare for Next Class

- Stare at Paxos and play with examples
- Read Raft
- Start to think about projects