Software Defined Networking
and
Dijkstra’s Algorithm

Presented by Prof. Jehn-Ruey Jiang

Department of Computer Science & Information Engineering
National Central University

October 24, 2014
Outline

Software-Defined Networking (SDN)
  - Overview
  - History
  - OpenFlow
  - Google’s WAN B4
  - Languages
  - Simulators and Emulators

Extended Dijkstra’s Algorithm
  - Dijkstra’s Algorithm
  - Extended Dijkstra’s Algorithm
  - Applications
  - Simulations

Summary
Outline

- Software-Defined Networking (SDN)
  - Overview
  - History
  - OpenFlow
  - Google’s WAN B4
  - Languages
  - Simulators and Emulators
- Extended Dijkstra’s Algorithm
  - Dijkstra’s Algorithm
  - Extended Dijkstra’s Algorithm
  - Applications
  - Simulations
- Summary
The Megatrend in the Computer Industry

Vertically integrated
Closed, proprietary
Slow innovation
Small industry

Specialized Applications
Specialized Operating System
Specialized Hardware

Open Interface
Windows (OS) or Linux or Mac OS

Horizontal
Open interfaces
Rapid innovation
Huge industry

Source: Dr. Jennifer Rexford’s presentation
The Same Trend in the Network Industry?

Vertically integrated
Closed, proprietary
Slow innovation

Horizontal
Open interfaces
Rapid innovation

Source: Dr. Jennifer Rexford's presentation
Software-Defined Networking (SDN)

- App (e.g., network virtualization)
- App (e.g., network management)

Northbound API

Controller (e.g., NOX/POX, Floodlight, etc.)

Southbound API (OpenFlow)

Packet Forwarding

Source: Dr. Shie-Yuan Wang’s Presentation
Outline

- Software-Defined Networking (SDN)
  - Overview
  - History
  - OpenFlow
  - Google’s WAN B4
  - Languages
  - Simulators and Emulators
- Extended Dijkstra’s Algorithm
  - Dijkstra’s Algorithm
  - Extended Dijkstra’s Algorithm
  - Applications
  - Simulations
- Summary
Software Defined Networking History

<table>
<thead>
<tr>
<th>~2004: Research on new management paradigms</th>
</tr>
</thead>
<tbody>
<tr>
<td>RCP, 4D [Princeton, CMU,....]</td>
</tr>
<tr>
<td>SANE, Ethane [Stanford/Berkeley]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2008: Software-Defined Networking (SDN)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOX Network Operating System [Nicira]</td>
</tr>
<tr>
<td>Open Flow switch interface [Stanford/ Nicira]</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>2011: Open Networking Foundation</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Board</strong>: Google, Yahoo, Verizon, DT, Microsoft, Facebook, NTT</td>
</tr>
<tr>
<td><strong>Members</strong>: Cisco, Juniper, HP, Dell, Broadcom, IBM,.....</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>Google is using SDN in its private WAN B4</td>
</tr>
<tr>
<td>Many commercialized SDN products</td>
</tr>
</tbody>
</table>

Source: Dr. Marco Cello’s presentation
Software Defined Networking History

Biography
• Born in Bedford, England on April 7, 1963
• Obtained bachelor's degree from the University of Leeds in 1986
• Obtained master's degree in 1992 and PhD in 1995 from the University of California at Berkeley
• Became faculty director of the Clean Slate Program at Stanford University in 2006
• Casado, McKeown and Shenker co-founded Nicira Networks in 2007
• McKeown and Shenker co-founded the Open Networking Foundation (ONF) in 2011
• Nicira was acquired by VMware for $1.26 billion in July 2012

Source: http://en.wikipedia.org/wiki/Nick_McKeown
The Open Networking Foundation (ONF) is a nonprofit organization, founded by Deutsche Telekom, Facebook, Google, Microsoft, Verizon, and Yahoo! on March 21, 2011. Its purpose is to improve networking through software-defined networking (SDN) and standardizing the OpenFlow protocol and related technologies.

https://www.opennetworking.org/
Open Networking Foundation Member Listing

https://www.opennetworking.org/membership/member-listing
Outline

- Software-Defined Networking (SDN)
  - Overview
  - History
  - OpenFlow
  - Google’s WAN B4
  - Languages
  - Simulators and Emulators
- Extended Dijkstra’s Algorithm
  - Dijkstra’s Algorithm
  - Extended Dijkstra’s Algorithm
  - Applications
  - Simulations
- Summary
SDN Architecture

Source: Open Networking Foundation What paper, 2012
OpenFlow Controller & Switch

Flow Table Entry

<table>
<thead>
<tr>
<th>Rule</th>
<th>Action</th>
<th>Statistics</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Packet + byte counters</td>
</tr>
</tbody>
</table>

1. Forward packet to port(s)
2. Encapsulate and forward to controller
3. Drop packet
4. Send to normal processing pipeline

<table>
<thead>
<tr>
<th>Switch In Port</th>
<th>MAC src</th>
<th>MAC dst</th>
<th>Eth type</th>
<th>VLAN ID</th>
<th>IP Src</th>
<th>IP Dst</th>
<th>IP Prot</th>
<th>TCP sport</th>
<th>TCP dport</th>
</tr>
</thead>
</table>

Source: Dr. McKeown’s presentation
Flow Table Process Example

A Mei
Flow Table Process Example (cont.)

<table>
<thead>
<tr>
<th>Name</th>
<th>Attribute</th>
<th>Attribute</th>
<th>.......</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Mei</td>
<td>Interorbital Width</td>
<td>Mouth size</td>
<td>...</td>
</tr>
<tr>
<td>Jessica</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Iris</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>XXX·XXX</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>XXX·XX</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### Flow Table Process

<table>
<thead>
<tr>
<th>Name</th>
<th>Attribute</th>
<th>Attribute</th>
<th>……</th>
</tr>
</thead>
<tbody>
<tr>
<td>A Mei</td>
<td>Interorbital Width</td>
<td>Mouth size</td>
<td>…</td>
</tr>
<tr>
<td>Jessica</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>Iris</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>XXX·XXX</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
<tr>
<td>XXX·XX</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Flow Table Process

Packet In
Start at table 0

Match in table n?

Yes

Update counters
Execute instructions:
- update action set
- update packet/match set fields
- update metadata

Goto-Table n?

Yes

No

Execute action set

No

Table-miss flow entry exists?

Yes

No

Drop packet
OpenFlow Versions

- **2009**
  - Published OpenFlow Switch Specification 1.0 – Dec. 31

- **2011**
  - Published OpenFlow Switch Specification 1.1 – Feb. 28
  - Published OpenFlow Switch Specification 1.2 – Dec. 5

- **2012**
  - Published OpenFlow Switch Specification 1.3.0 – Jun. 25
    - The OpenFlow 1.3.X is Long Term Support Vision (LTS Vision)
  - Published OpenFlow Switch Specification 1.3.1 – Sep. 6

- **2013**
  - Published OpenFlow Switch Specification 1.3.2 – Apr. 25
  - Published OpenFlow Switch Specification 1.3.3 – Sep. 27
  - Published OpenFlow Switch Specification 1.4 – Oct. 14

- **2014**
  - Published OpenFlow Switch Specification 1.3.4 – Mar. 27
  - OpenFlow Switch Specification 1.5 to be published at the end of 2014
SDN Controllers
### Comparison Among The Controllers

<table>
<thead>
<tr>
<th>Feature</th>
<th>POX</th>
<th>Ryu</th>
<th>FloodLight</th>
<th>OpenDaylight</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Interfaces</strong></td>
<td>SB (OpenFlow)</td>
<td>SB (OpenFlow) +SB Management (OVSDB JSON)</td>
<td>SB (OpenFlow) NB (Java &amp; REST)</td>
<td>SB (OpenFlow &amp; Others SB Protocols) NB (REST &amp; Java RPC)</td>
</tr>
<tr>
<td><strong>Virtualization</strong></td>
<td>Mininet &amp; Open vSwitch</td>
<td>Mininet &amp; Open vSwitch</td>
<td>Mininet &amp; Open vSwitch</td>
<td>Mininet &amp; Open vSwitch</td>
</tr>
<tr>
<td><strong>GUI</strong></td>
<td>Yes</td>
<td>Yes (Initial Phase)</td>
<td>Web UI (Using REST)</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>REST API</strong></td>
<td>No</td>
<td>Yes (For SB Interface only)</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Productivity</strong></td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Open Source</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Documentation</strong></td>
<td>Poor</td>
<td>Medium</td>
<td>Good</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Language Support</strong></td>
<td>Python</td>
<td>Python-Specific + Message Passing Reference</td>
<td>Java + Any language that uses REST</td>
<td>Java</td>
</tr>
<tr>
<td><strong>Modularity</strong></td>
<td>Medium</td>
<td>Medium</td>
<td>High</td>
<td>High</td>
</tr>
<tr>
<td><strong>Platform Support</strong></td>
<td>Linux, Mac OS, and Windows</td>
<td>Most Supported on Linux</td>
<td>Linux, Mac &amp; Windows</td>
<td>Linux</td>
</tr>
<tr>
<td><strong>TLS Support</strong></td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Age</strong></td>
<td>1 year</td>
<td>1 year</td>
<td>2 years</td>
<td>2 Month</td>
</tr>
<tr>
<td><strong>OpenFlow Support</strong></td>
<td>OF v1.0</td>
<td>OF v1.0 v2.0 v3.0 &amp; Nicira Extensions</td>
<td>OF v1.0</td>
<td>OF v1.0</td>
</tr>
<tr>
<td><strong>OpenStack Networking</strong></td>
<td>NO</td>
<td>Strong</td>
<td>Medium</td>
<td>Medium</td>
</tr>
</tbody>
</table>

---

Outline

- Software-Defined Networking (SDN)
  - Overview
  - History
  - OpenFlow
  - Google’s WAN B4
- Languages
- Simulators and Emulators
- Extended Dijkstra’s Algorithm
  - Dijkstra’s Algorithm
  - Extended Dijkstra’s Algorithm
- Applications
- Simulations

Summary
Google Networks Scales

- **User base**
  - World population: 6.676 billion people (June’08, US Census est.)
  - Internet Users: 1.463 billion (>20%) (June’08 Nielsen/ITU)
  - Google Search: More than Billion Searches Every Day

- **Geographical Distribution**
  - Google services are worldwide: over 55 countries and 112 languages

- **Data Growth**
  - Web expands/changes: billions of new/modified pages every month
  - Every few hours Google craw/refresh more than entire Library of Congress
  - YouTube gains over 131,518,242,460 hours of video every minute, 4+ billion views every day

- **Latency Challenge**
  - Speed of Light in glass: $2 \times 10^8 \text{ m/s} = 2,000 \text{ km/10ms}$
  - “Blink of an eye response” = 100 ms
ATLAS 2010 Traffic Report

Google as a Percentage of All Internet Traffic

Credit: Arbor Networks
Google’s WAN B4

- Using Software Defined Networking (SDN) principles
- Using OpenFlow
- Supporting standard routing protocols
- Centralized Traffic Engineering (TE)

Google’s WAN B4 (cont.)

- Traditional WAN Routing
  - Treat all bits the same
  - 30% ~ 40% average utilization

- Centralized Traffic Engineering (TE)
  - Drive links to near 100% utilization
  - Fast, global convergence for failures
B4 Sample Utilization

The statistics are from Google company
Google’s WAN B4 History

- Exit testing "opt in" network
- SDN rollout
- SDN fully Deployed
- Central TE Deployed

The statistics are from Google company
Outline

- Software-Defined Networking (SDN)
  - Overview
  - History
  - OpenFlow
  - Google’s WAN B4
- Languages
- Simulators and Emulators
- Extended Dijkstra’s Algorithm
  - Dijkstra’s Algorithm
  - Extended Dijkstra’s Algorithm
- Applications
- Simulations
- Summary
Frenetic

Frenetic is introduced as a high-level language to program the controller to manage switches in the SDN network.
Based on Frenetic and Python, Pyretic is introduced as an SDN programming language or platform that raises the level of abstraction.

Python + Frenetic = Pyretic
Pyretic (cont.)

- It allows programmers to focus on how to specify a network policy at high-level abstraction.

- Pyretic hides low-level details by allowing programmers to express policies as compact, abstract functions that take a packet as input, and return a set of new packets.
Outline

- Software-Defined Networking (SDN)
  - Overview
  - History
  - OpenFlow
  - Google’s WAN B4
  - Languages
  - Simulators and Emulators
- Extended Dijkstra’s Algorithm
  - Dijkstra’s Algorithm
  - Extended Dijkstra’s Algorithm
  - Applications
  - Simulations
- Summary
Mininet

Mininet is an open source network emulator that supports the OpenFlow protocol for the SDN architecture. It is one of the most popular tools used by the SDN research community.
Mininet Features

- **It's fast** - starting up a simple network takes just a few seconds. This means that your run-edit-debug loop can be very quick.

- You can **create custom topologies**: a single switch, larger Internet-like topologies, the Stanford backbone, a data center, or anything else.

- You can **run real programs**: anything that runs on Linux is available for you to run, from web servers to TCP window monitoring tools to Wireshark.
Mininet Features (cont.)

- You can **run Mininet on your laptop**, on a server, in a VM, on a native Linux box (Mininet is included with Ubuntu 12.10+!), or in the cloud (e.g. Amazon EC2).

- You can **share and replicate results**: anyone with a computer can run your code once you've packaged it up.

- Mininet is an **open source project**, so you are encouraged to examine its source code on https://github.com/mininet, modify it, fix bugs, file issues/feature requests, and submit patches/pull requests.
EstiNet Simulator/Emulator

- EstiNet combines the advantages of both the simulation and emulation approaches without their respective shortcomings.

- EstiNet uses tunnel network interfaces to automatically intercept the packets exchanged by two real applications and redirect them into the EstiNet simulation engine.
EstiNet Simulator/Emulator (cont.)

- EstiNet’s GUI can be used to easily set up and configure a simulation case and be used to observe the packet playback of a simulation run.
# A Comparison of Network Simulators

<table>
<thead>
<tr>
<th></th>
<th>EstiNet</th>
<th>ns-3</th>
<th>Mininet</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>OpenFlow specification</strong></td>
<td>1.1.0/1.0.0</td>
<td>0.8.9</td>
<td>1.0.0</td>
</tr>
<tr>
<td><strong>Simulation mode</strong></td>
<td>✓</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td><strong>Emulation mode</strong></td>
<td>✓</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Compatible with real-world controllers</strong></td>
<td>✓</td>
<td>—</td>
<td>✓</td>
</tr>
<tr>
<td><strong>Result repeatable</strong></td>
<td>✓</td>
<td>✓</td>
<td>—</td>
</tr>
<tr>
<td><strong>Scalability</strong></td>
<td>High by single process</td>
<td>High by single process</td>
<td>Middle by multiple processes</td>
</tr>
<tr>
<td><strong>Performance result correctness</strong></td>
<td>✓</td>
<td>No Spanning Tree Protocol and no real-world controller</td>
<td>No performance fidelity</td>
</tr>
<tr>
<td><strong>GUI support</strong></td>
<td>✓ Configuration • Observation</td>
<td>✓ Observation only • Using C++ for configuration</td>
<td>✓ Observation only • Using Python for configuration</td>
</tr>
</tbody>
</table>
Outline

- Software-Defined Networking (SDN)
  - Overview
  - History
  - OpenFlow
  - Google’s WAN B4
  - Languages
  - Simulators and Emulators
- Extended Dijkstra’s Algorithm
  - Dijkstra’s Algorithm
  - Extended Dijkstra’s Algorithm
  - Applications
  - Simulations
- Summary
Edsger Wybe Dijkstra

• Dijkstra was born in Rotterdam, Netherlands.
• He received the **1972 Turing Award** for fundamental contributions to developing programming languages.
• In 2002, he received the **ACM PODC Influential Paper Award** in distributed computing for his work on **self-stabilization** of program computation. This annual award was renamed the **Dijkstra Prize** the following year, in his honor.

(1930~2002)
Dijkstra’s shortest path algorithm is a 20-minute invention while drinking coffee

“One morning I was shopping in Amsterdam with my young fiancée, and tired, we sat down on the café terrace to drink a cup of coffee and I was just thinking about whether I could do this, and I then designed the algorithm for the shortest path. As I said, it was a 20-minute invention. In fact, it was published in 1959, three years later.”

Dijkstra's Algorithm

- Conceived by computer scientist Edsger Dijkstra in 1956, published in 1959, is a graph search algorithm.
- Solves the single-source all-destination shortest path problem for a graph with non-negative edge path costs, producing a shortest path tree.
- This algorithm is often used in routing and as a subroutine in other graph algorithms.

http://en.wikipedia.org/wiki/Dijkstra%27s_algorithm
Outline

- Software-Defined Networking (SDN)
  - Overview
  - History
  - OpenFlow
  - Google’s WAN B4
  - Languages
  - Simulators and Emulators
- Extended Dijkstra’s Algorithm
  - Dijkstra’s Algorithm
  - Extended Dijkstra’s Algorithm
  - Applications
  - Simulations
- Summary
## Extending Dijkstra’s Algorithm

### Extended Dijkstra’s Algorithm

**Input:** $G=(V, E), \ ew, \ nw, \ s$

**Output:** $d[|V|], \ p[|V|]$

1. $d[s] \leftarrow 0; \ d[u] \leftarrow \infty$, for each $u \neq s, \ u \in V$
2. **insert** $u$ with key $d[u]$ into the priority queue $Q$, for each $u \in V$
3. **while** $(Q \neq \emptyset)$
   4. $u \leftarrow \text{Extract-Min}(Q)$
   5. **for** each $v$ adjacent to $u$
   6. \hspace{1em} **if** $d[v] > d[u] + ew[u,v] + nw[u]$ \ **then**
   7. \hspace{2em} $d[v] \leftarrow d[u] + ew[u,v] + nw[u]$
   8. \hspace{1em} $p[v] \leftarrow d[u]$
Node Weight & Edge Weight

- The node weight $nw[v]$ of $v$ is defined according to Eq. (1)
  \[
  nw[v] = \frac{\sum_{f \in Flow(v)} Bits(f)}{Capacity(v)},
  \]
  where $Bits(f)$ stands for the number of $f$’s bits processed by node $v$ per second.

- The edge weight $ew[e]$ of $e$ is defined according to Eq. (2).
  \[
  ew[e] = \frac{\sum_{f \in Flow(e)} Bits(f)}{Bandwidth(e)},
  \]
  where $Bits(f)$ stands for the number of $f$’s bits passing through edge $e$ per second.
Outline

- Software-Defined Networking (SDN)
  - Overview
  - History
  - OpenFlow
  - Google’s WAN B4
  - Languages
  - Simulators and Emulators
- Extended Dijkstra’s Algorithm
  - Dijkstra’s Algorithm
  - Extended Dijkstra’s Algorithm
  - Applications
  - Simulations
- Summary
End-to-End Routing
Multicast
Load Balancing
Outline

- Software-Defined Networking (SDN)
  - Overview
  - History
  - OpenFlow
  - Google’s WAN B4
  - Languages
  - Simulators and Emulators
- Extended Dijkstra’s Algorithm
  - Dijkstra’s Algorithm
  - Extended Dijkstra’s Algorithm
  - Applications
  - Simulations
- Summary
The Abilene Network Core Topology

- The Abilene network is a high-performance backbone network suggested by the Internet2 project in the late 1990s.
- It connects 11 regional sites or nodes across the United States.
- It has 10 Gbps connectivity between neighboring nodes and 100 Mbps connectivity between a host and a node.

Simulation Settings for Routing

- Setting up the Abilene topology in Mininet for routing from the green client to the red client
Simulation Settings for Routing (cont.)

- We use Iperf as a testing tool to generate TCP data streams in our simulation.
- The testing time for every testing case is 1000 seconds.
- The Iperf tool reports the average TCP bandwidth between a client and the server, and we use the packet size of 53 bytes to derive the average end-to-end latency.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth on edges</td>
<td>100Mbps ~ 1Gbps</td>
</tr>
<tr>
<td>Capacity of nodes</td>
<td>3Gbps ~ 7Gbps</td>
</tr>
<tr>
<td>Number of hosts</td>
<td>10</td>
</tr>
<tr>
<td>Number of nodes</td>
<td>11</td>
</tr>
<tr>
<td>Number of edges</td>
<td>25</td>
</tr>
<tr>
<td>Controller</td>
<td>POX 2.0 supporting Pyretic</td>
</tr>
<tr>
<td>Testing tool</td>
<td>Iperf</td>
</tr>
<tr>
<td>Testing time per case</td>
<td>1000 sec</td>
</tr>
</tbody>
</table>
Simulation Results for Routing

The end-to-end latency of an Iperf server and a client
Simulation Setting for Multicasting

- Topology Setting
Simulation Settings for Multicasting (cont.)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of controller</td>
<td>1</td>
</tr>
<tr>
<td>Number of switches</td>
<td>11</td>
</tr>
<tr>
<td>Number of publisher</td>
<td>1</td>
</tr>
<tr>
<td>Number of subscribers</td>
<td>12</td>
</tr>
<tr>
<td>Number of edges</td>
<td>25</td>
</tr>
<tr>
<td>Controller</td>
<td>POX 2.0 supporting Pyretic</td>
</tr>
<tr>
<td>OpenFlow switch</td>
<td>Open vSwitch 1.0</td>
</tr>
<tr>
<td>Testing tool</td>
<td>Iperf</td>
</tr>
<tr>
<td>Testing time per case</td>
<td>30 sec</td>
</tr>
</tbody>
</table>
Simulation Results for Multicasting

- The BFST

```
parent   4  6  4  5  5  2  11 10  5  6
child    1  2  3  4  6  7  8  9 10 11
```
Simulation Results for Multicasting (cont.)

- The DSPT

![Network Diagram]

<table>
<thead>
<tr>
<th></th>
<th>3</th>
<th>6</th>
<th>4</th>
<th>5</th>
<th>5</th>
<th>8</th>
<th>11</th>
<th>3</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>parent</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>child</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>

Publisher
Simulation Results for Multicasting (cont.)

- The EDSPT

<table>
<thead>
<tr>
<th>parent</th>
<th>4</th>
<th>6</th>
<th>4</th>
<th>5</th>
<th>5</th>
<th>2</th>
<th>11</th>
<th>10</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>child</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
<td>10</td>
<td>11</td>
</tr>
</tbody>
</table>
Simulation Results for Multicasting (cont.)

- **Throughput**

![Graph showing throughput vs. number of subscribers for different protocols.](image-url)
Simulation Results for Multicasting (cont.)

Average Throughput

![Average Throughput Chart]

- BFST
- DSPT
- EDSPT

Average Throughput

Mbp/s

Algorithm

BFST
DSPT
EDSPT
Simulation Settings for Load-Balancing
## Simulation Settings for Load-Balancing (cont.)

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Setting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bandwidth on edges</td>
<td>1 Gbps</td>
</tr>
<tr>
<td>Number of server</td>
<td>2</td>
</tr>
<tr>
<td>Number of switches</td>
<td>11</td>
</tr>
<tr>
<td>Number of edges</td>
<td>25</td>
</tr>
<tr>
<td>Controller</td>
<td>POX 2.0 supporting Pyretic</td>
</tr>
<tr>
<td>Openflow Switch</td>
<td>Open vSwitch 1.0</td>
</tr>
<tr>
<td>Testing tool</td>
<td>Iperf, Netperf</td>
</tr>
<tr>
<td>Testing time</td>
<td>30 sec</td>
</tr>
</tbody>
</table>
Simulation Results for Load-Balancing

End-to-End Latency

![Graph showing End-to-End Latency with different load-balancing methods for varying numbers of clients (4, 8, 12). The x-axis represents the number of clients (4, 8, 12), and the y-axis represents milliseconds. The graph compares three methods: Proposed, Round robin, and Random-based. The Proposed method shows the lowest latency across all client numbers.]
Simulation Results for Load-Balancing (cont.)

Response Time

![Graph showing response time comparison between Proposed, Round robin, and Random-based methods for different numbers of clients (4, 8, and 12). The graph indicates that the Proposed method consistently has the lowest response time across all client numbers.]
Simulation Results for Load-Balancing (cont.)

Total Transactions to Servers

- Proposed
- Round robin
- Random-based

Transactions/second

Number of Clients

4 clients 8 clients 12 clients
Simulation Results for Load-Balancing (cont.)

Variation Loads on Server

- Proposed
- Round robin
- Random-based
Outline

- Software-Defined Networking (SDN)
  - Overview
  - History
  - OpenFlow
  - Google’s WAN B4
  - Languages
  - Simulators and Emulators
- Extended Dijkstra’s Algorithm
  - Dijkstra’s Algorithm
  - Extended Dijkstra’s Algorithm
  - Applications
  - Simulations

Summary
Summary

- We have introduced the SDN concept and its related technologies.

- We have shown how to apply the extended Dijkstra’s algorithm to SDN-based end-to-end routing, multicasting and load-balancing.
Related Publication


Thank You!