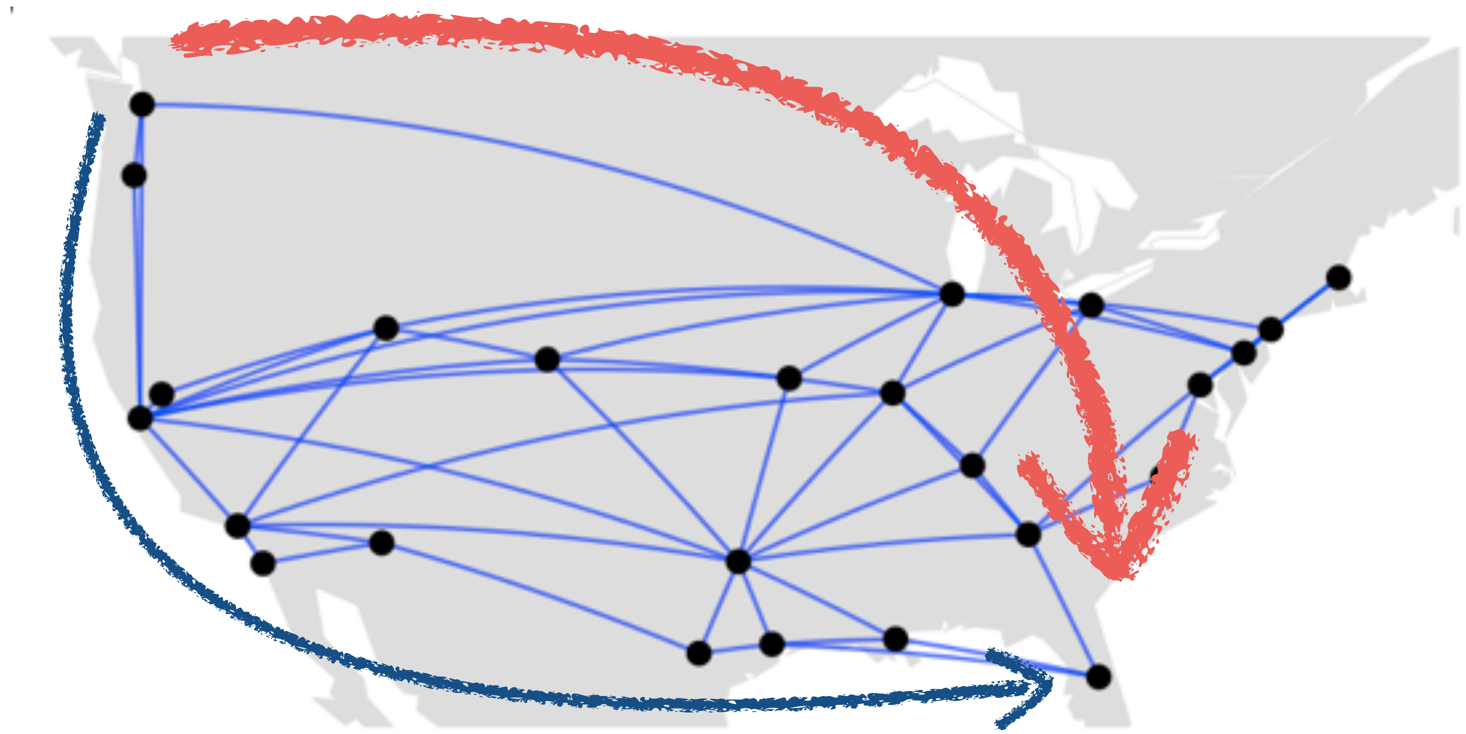
YATES: Rapid Prototyping for Traffic Engineering Systems

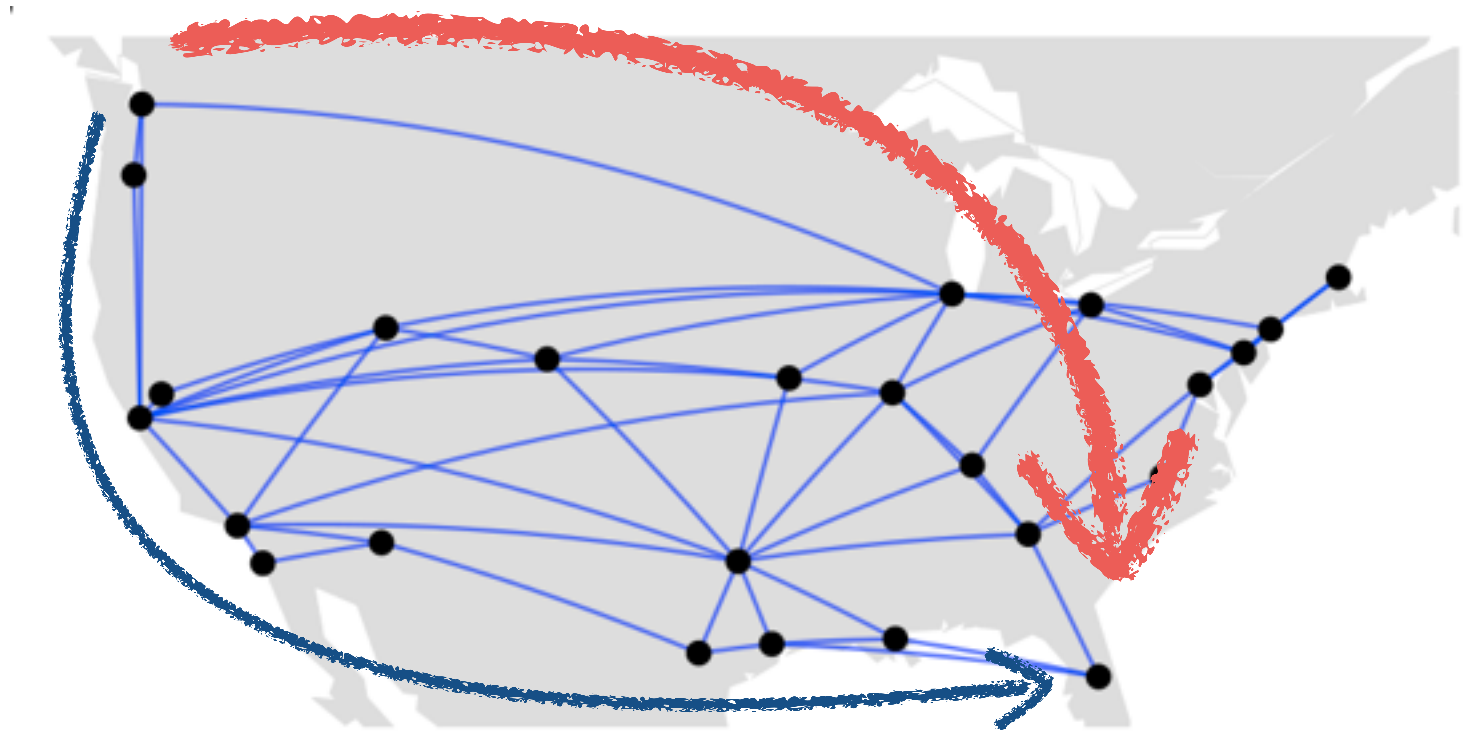
Praveen Kumar (Cornell)
Yang Yuan (Cornell)
Chris Yu (CMU)
Nate Foster (Cornell)
Robert Kleinberg (Cornell)
Robert Soulé (USI Lugano)

ACM SOSR 2018

WAN TE - Example

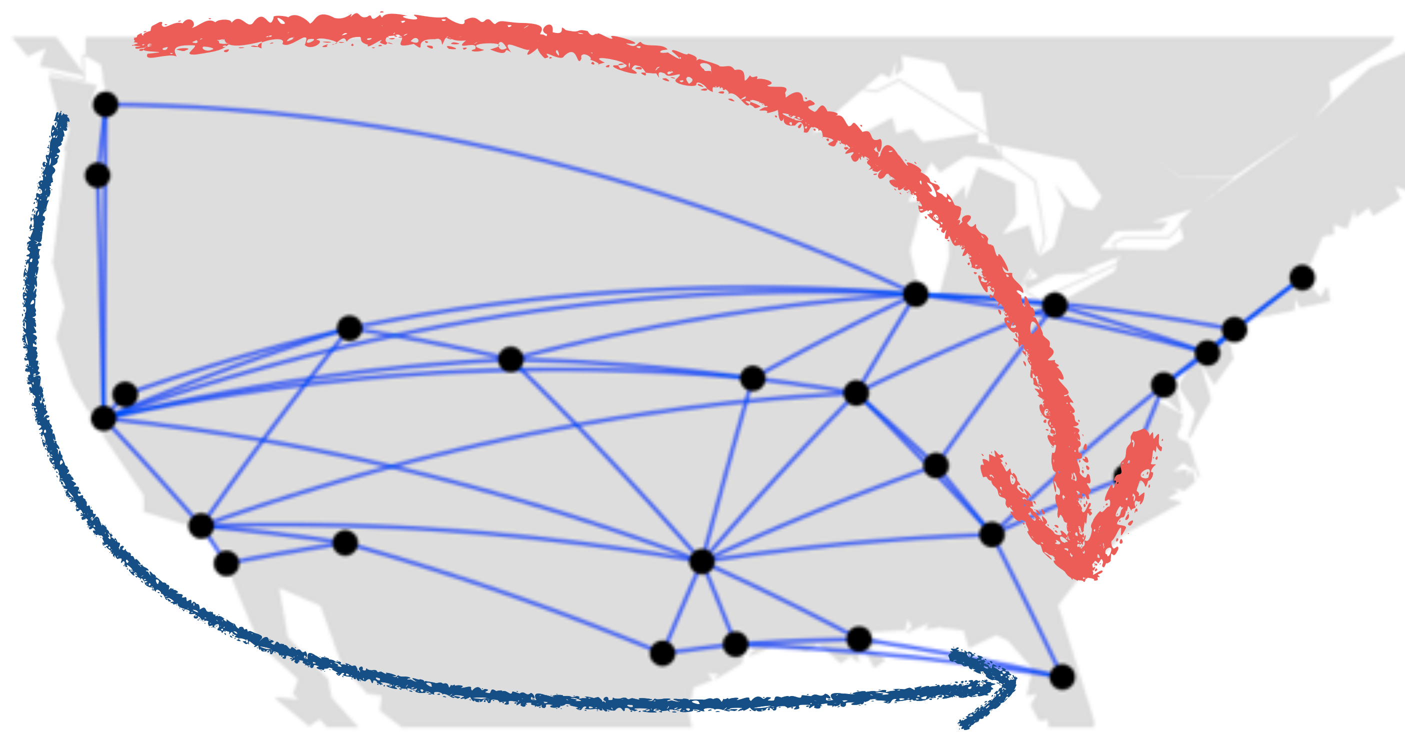


WAN TE - Example



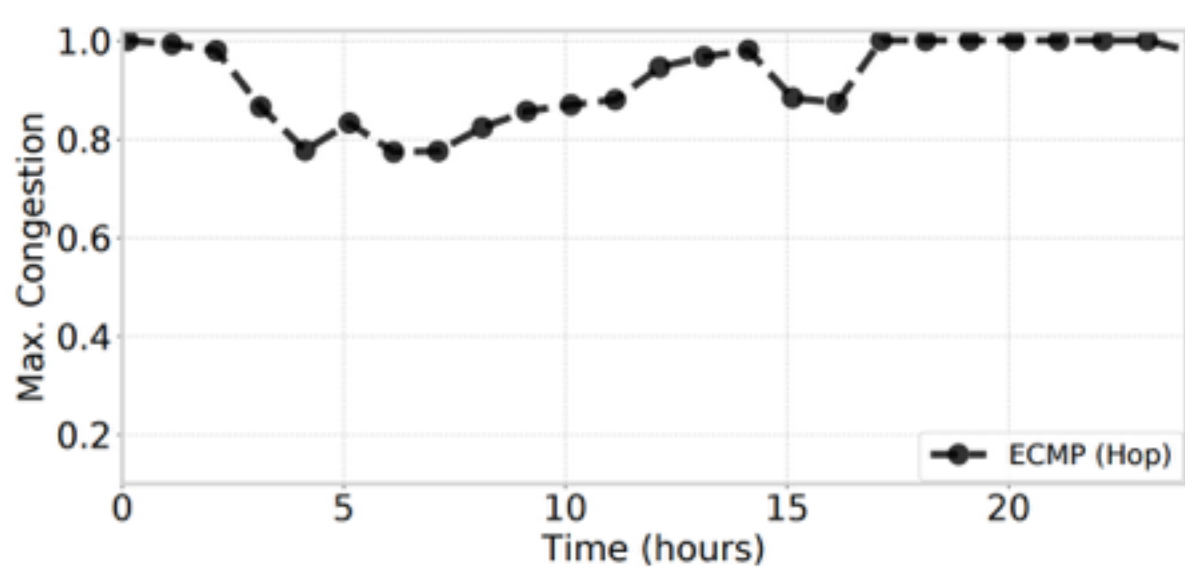
Configure the network to forward traffic using equal-cost multi-path (ECMP)

WAN TE - Example

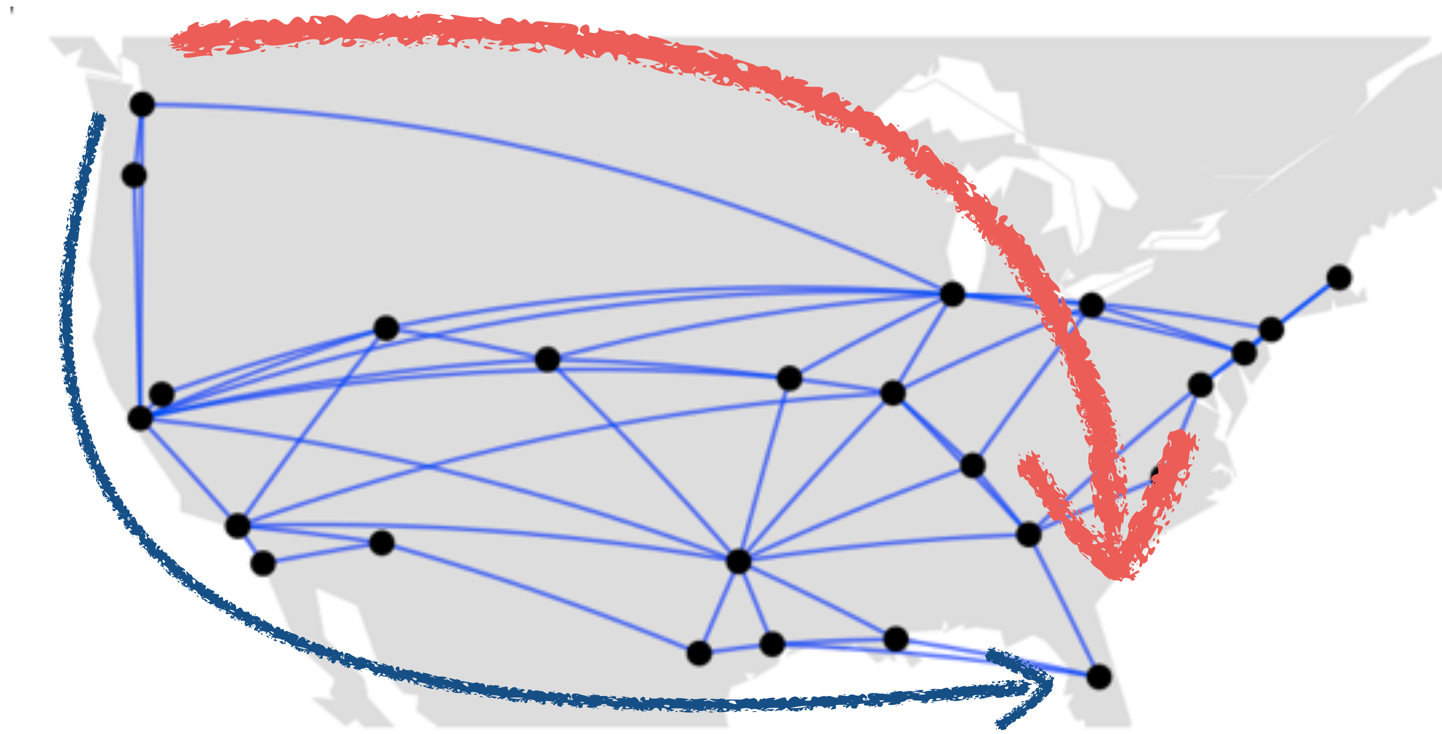



Network Operator

Configure the network to forward traffic using equal-cost multi-path (ECMP)

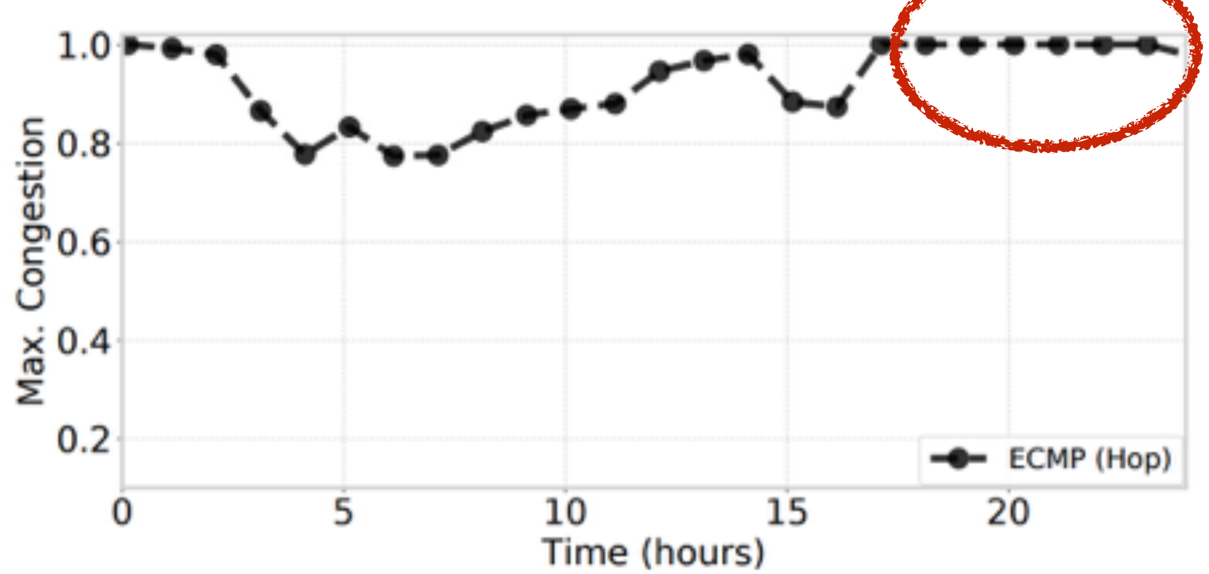


WAN TE - Example



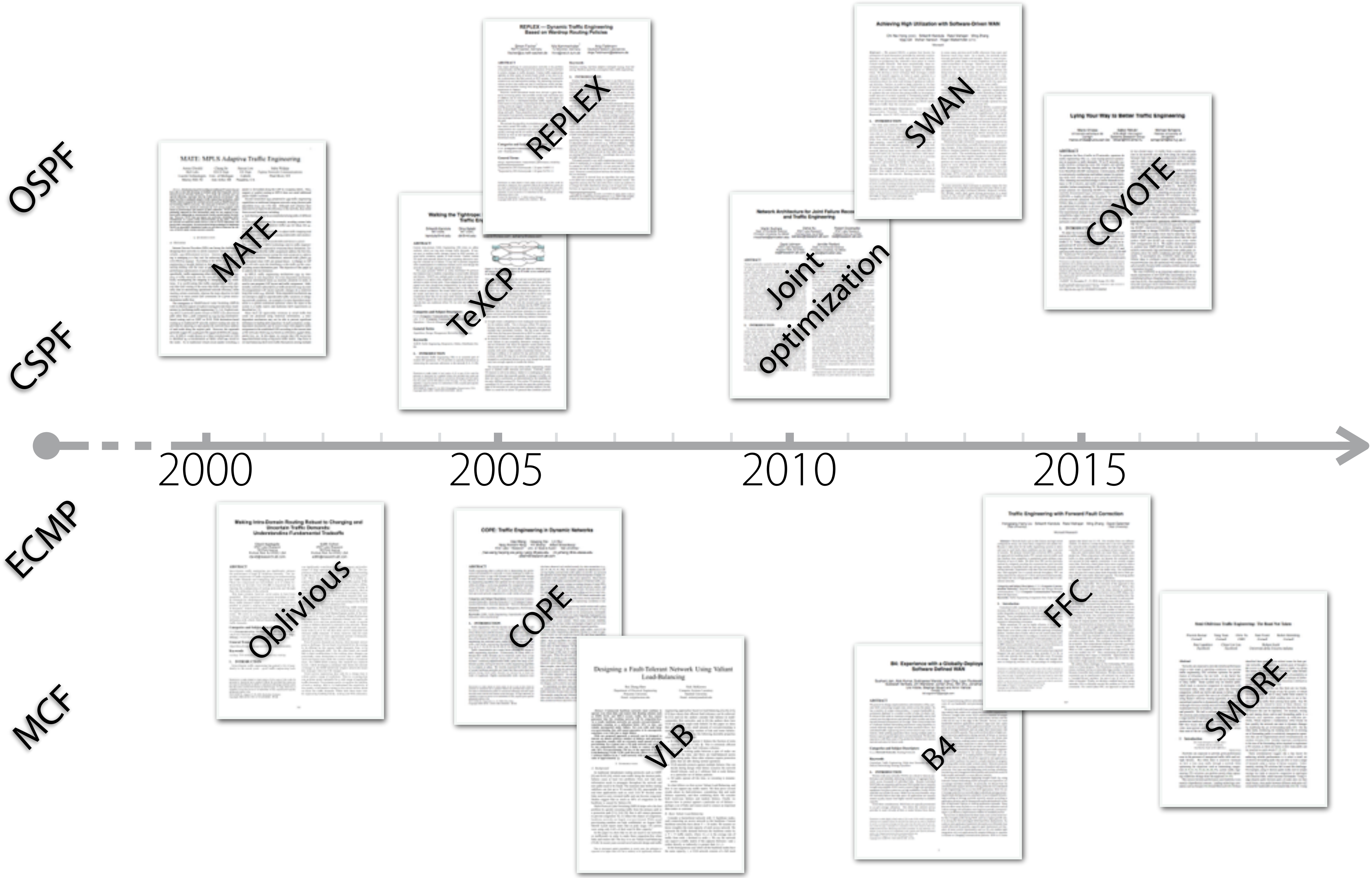
Network Operator

Configure the network to forward traffic using equal-cost multi-path (ECMP)

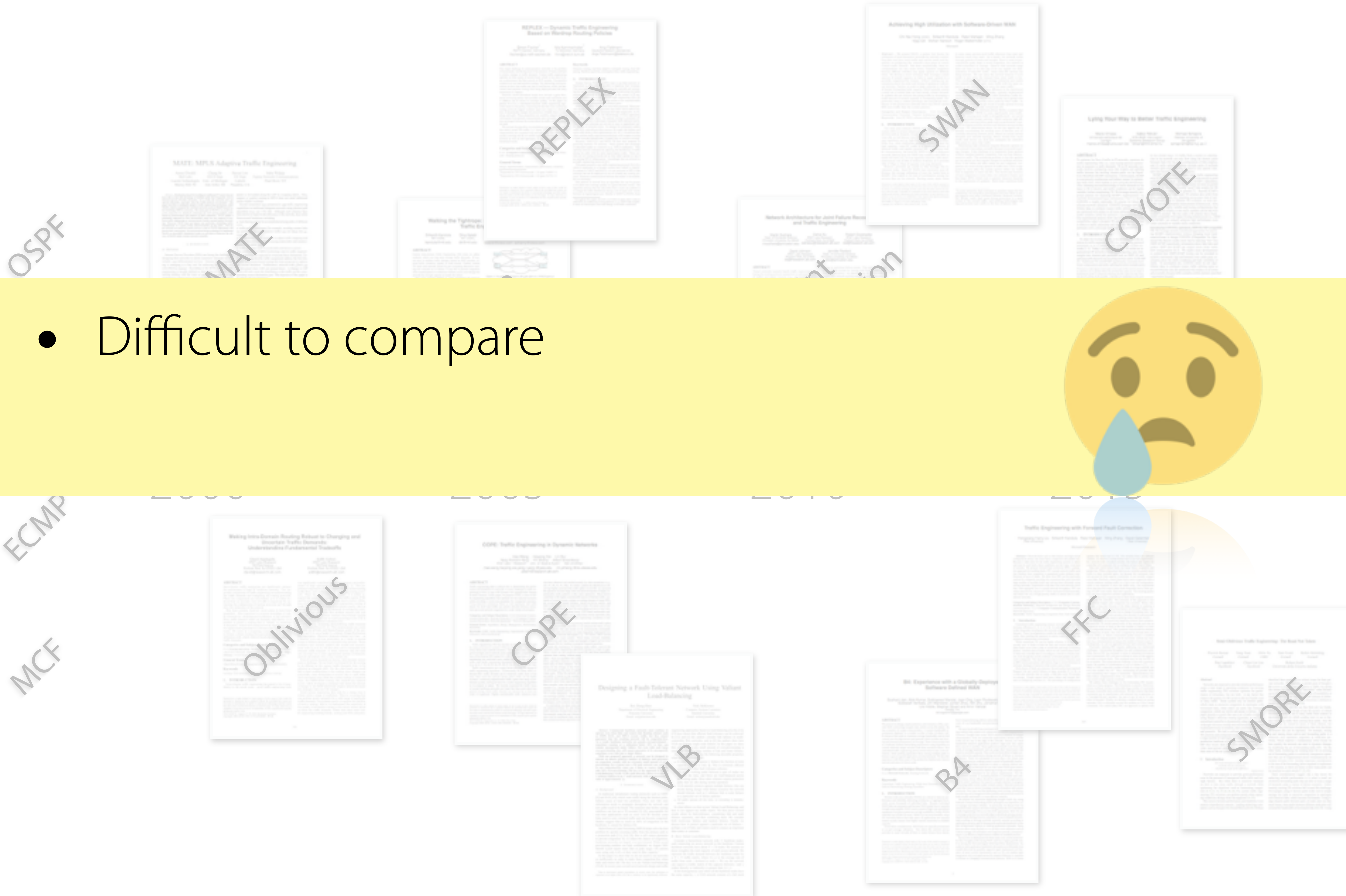


TE Systems

TE Systems


OSPF
CSPF
ECMP
MCF
2000
2005
2010
2015
MATE
TeXCP
REPLEX
Joint optimization
SWAN
COYOTE
Oblivious
COPE
VLB
B4
FFC
SMORE

TE Systems



- Difficult to compare

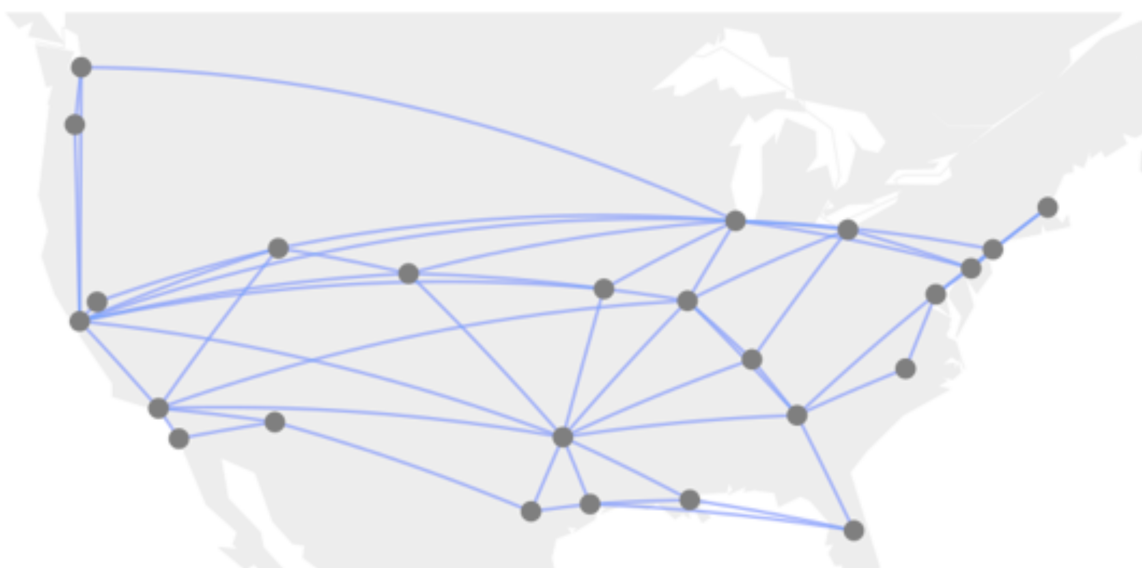
TE Systems

- Difficult to compare
- High cost of evaluation

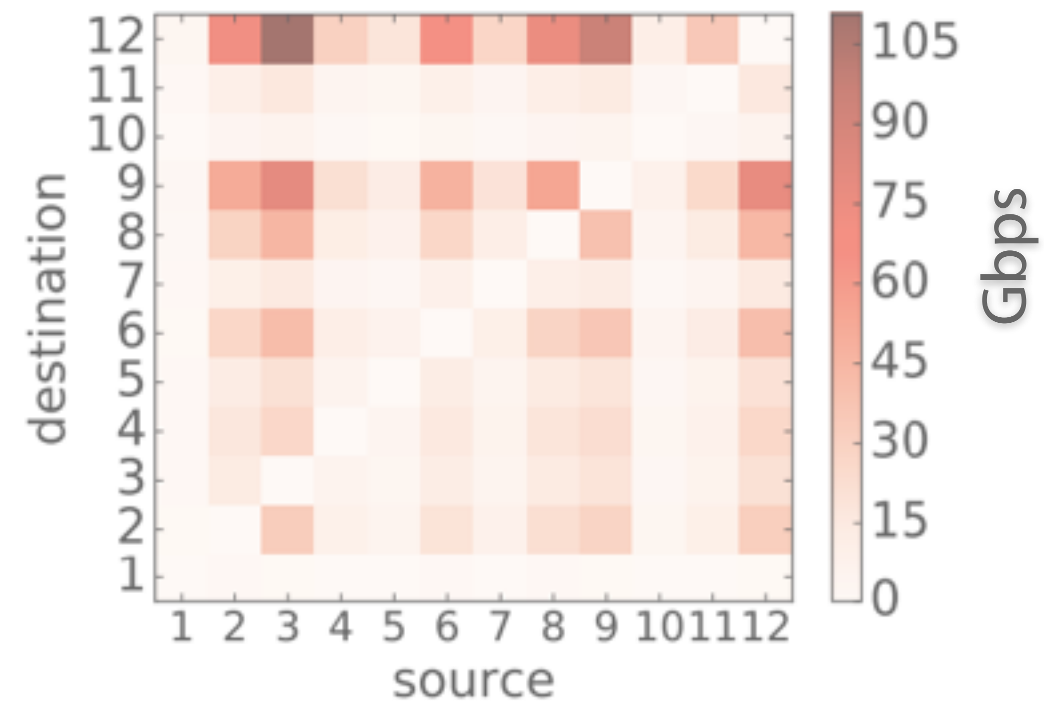
YATES
(Yet Another Traffic Engineering System)

- **Open-source** TE framework
- **High-level abstractions**
- **Modular interface** for implementing TE elements
- **Libraries** and **tools** for
  - generating traffic demands,
  - modeling failures,
  - prediction errors, etc.

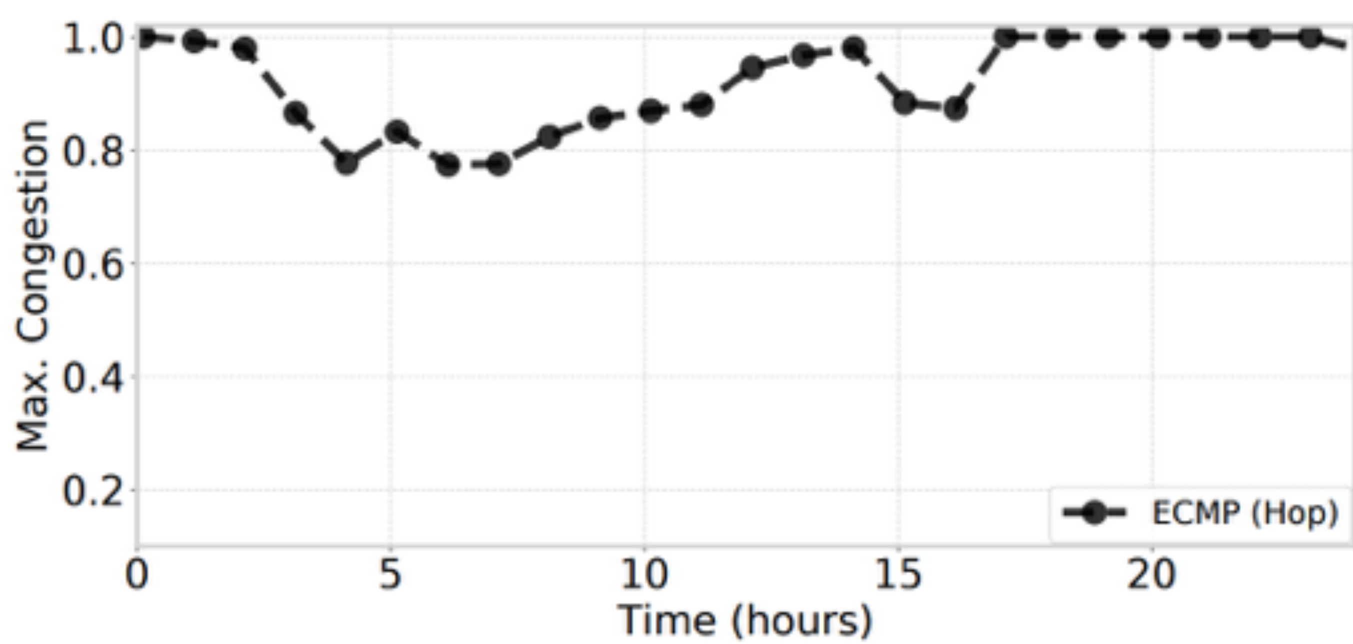
YATES - Example



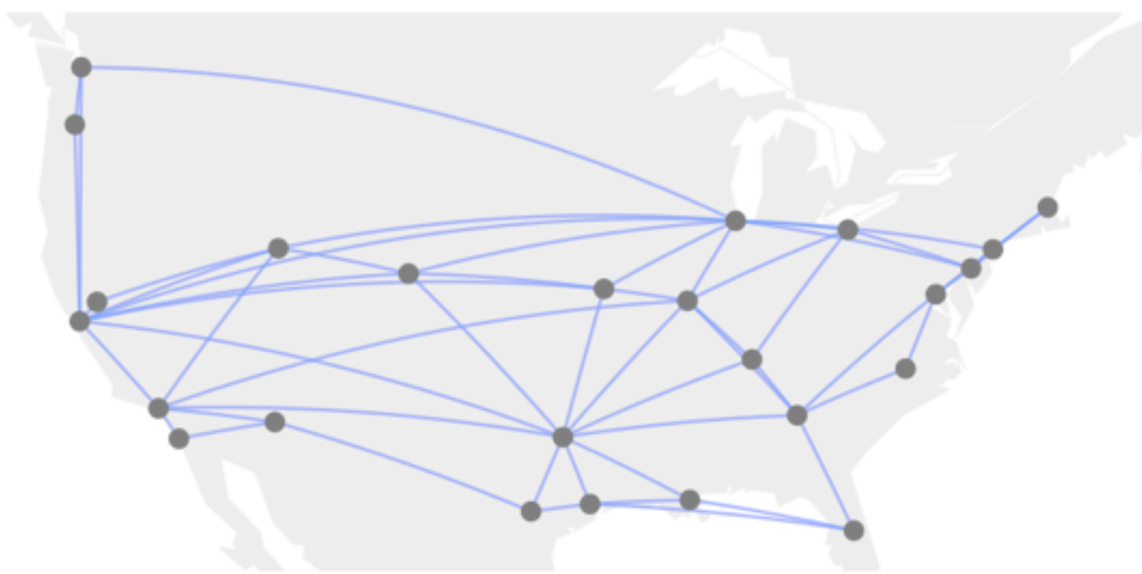
+



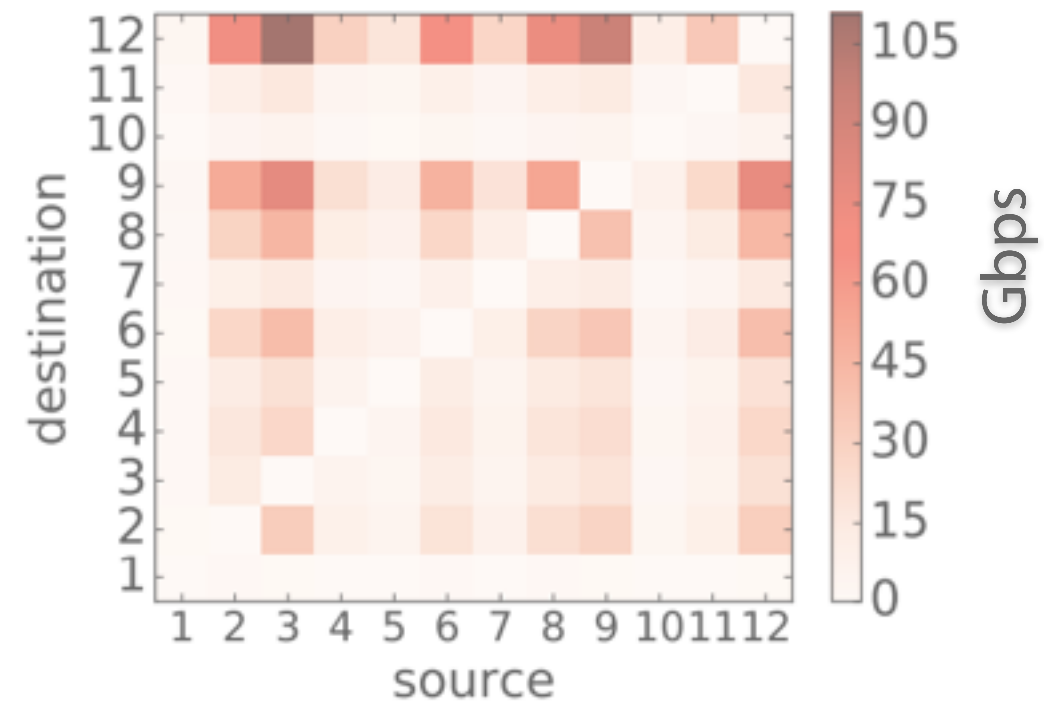
Network Operator



YATES - Example



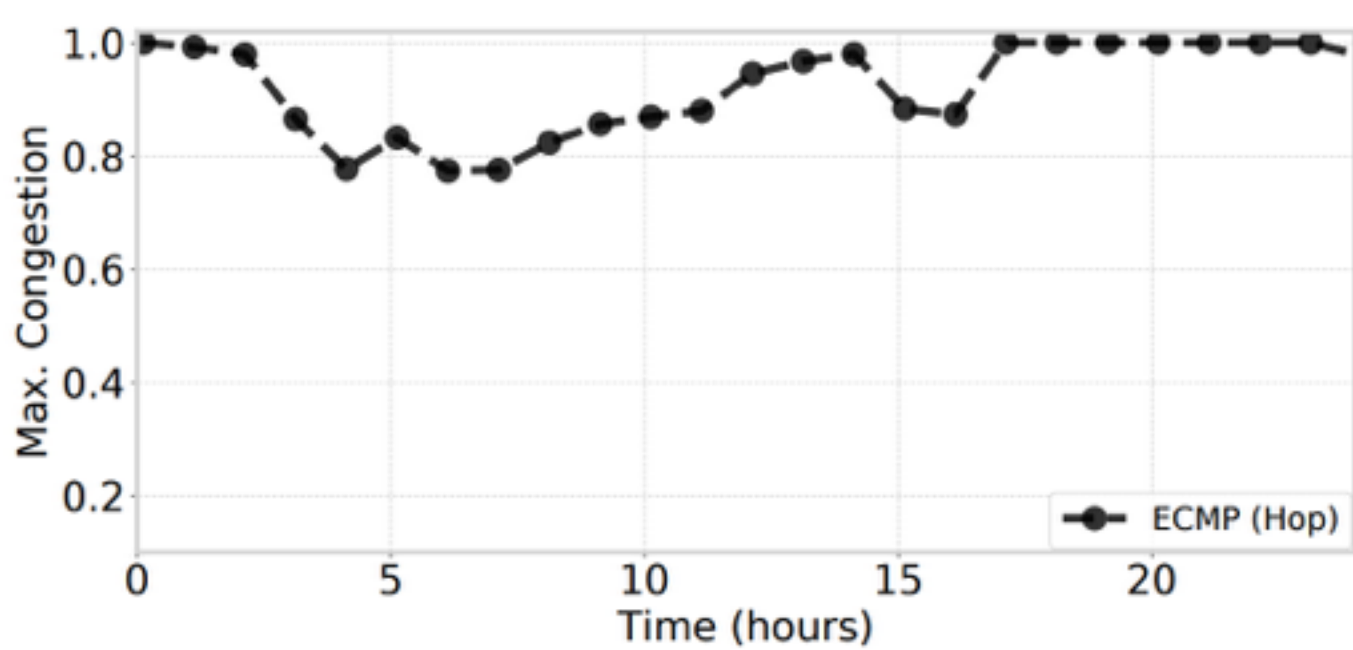
+



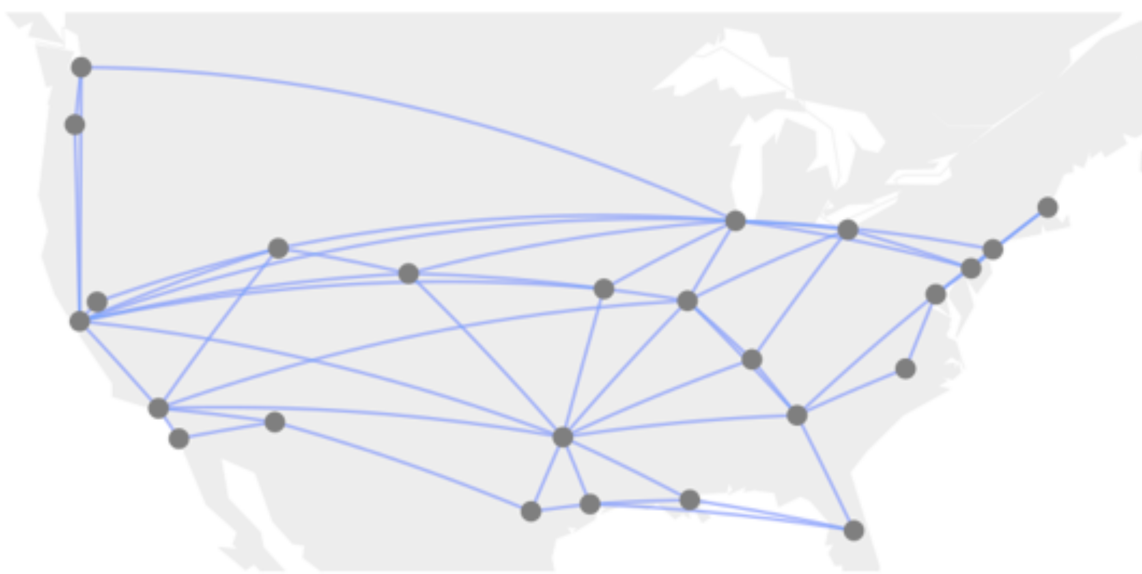
Network Operator

CSPF?

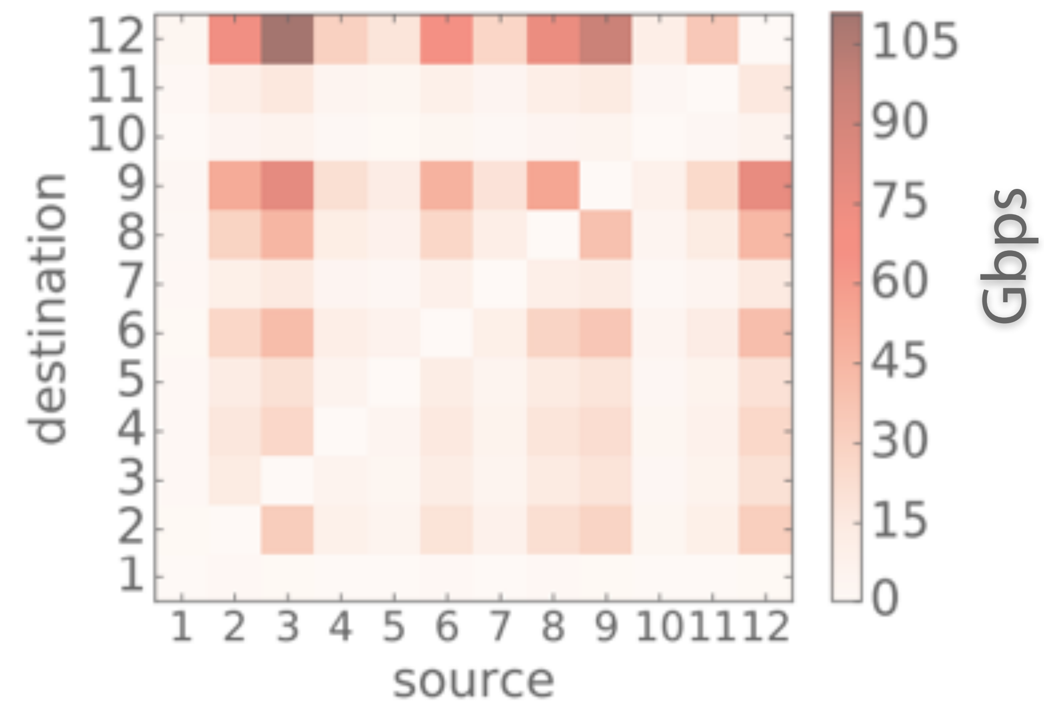
Link weight? (**1** or **RTT**)



YATES - Example



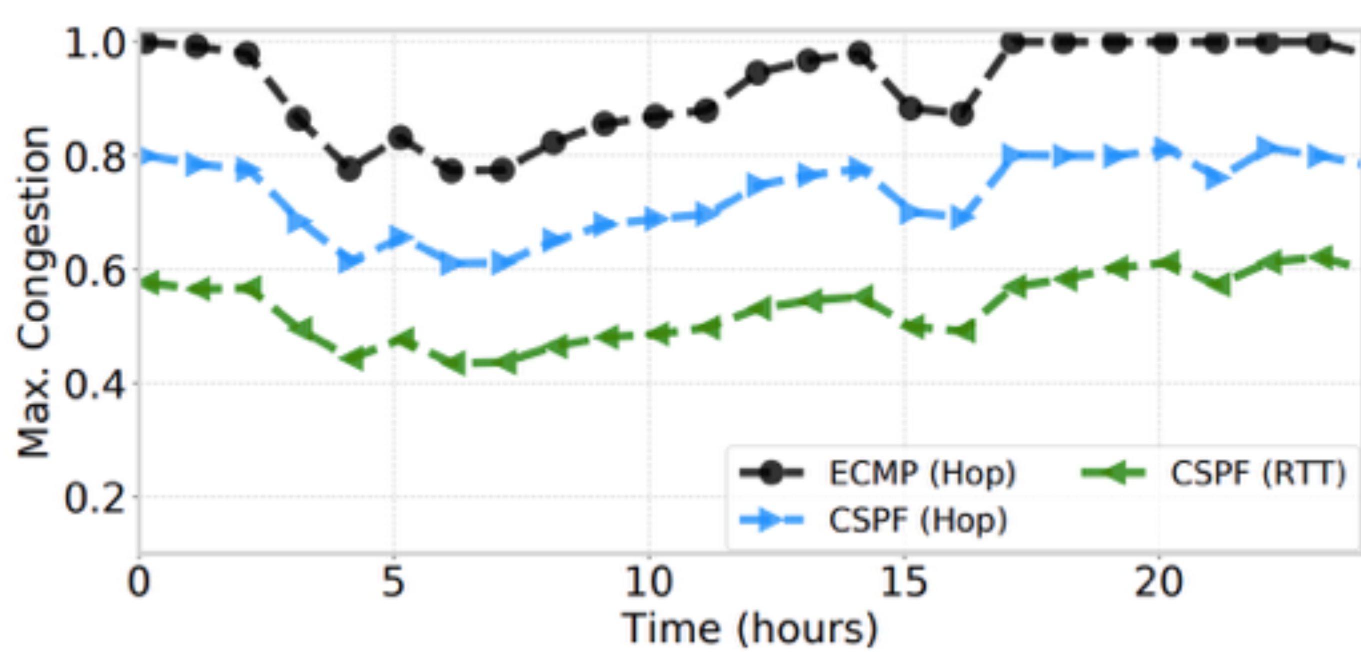
+



Network Operator

CSPF?

Link weight? (**1** or **RTT**)



YATES Interface

High-level and modular interface

YATES Interface

High-level and modular interface

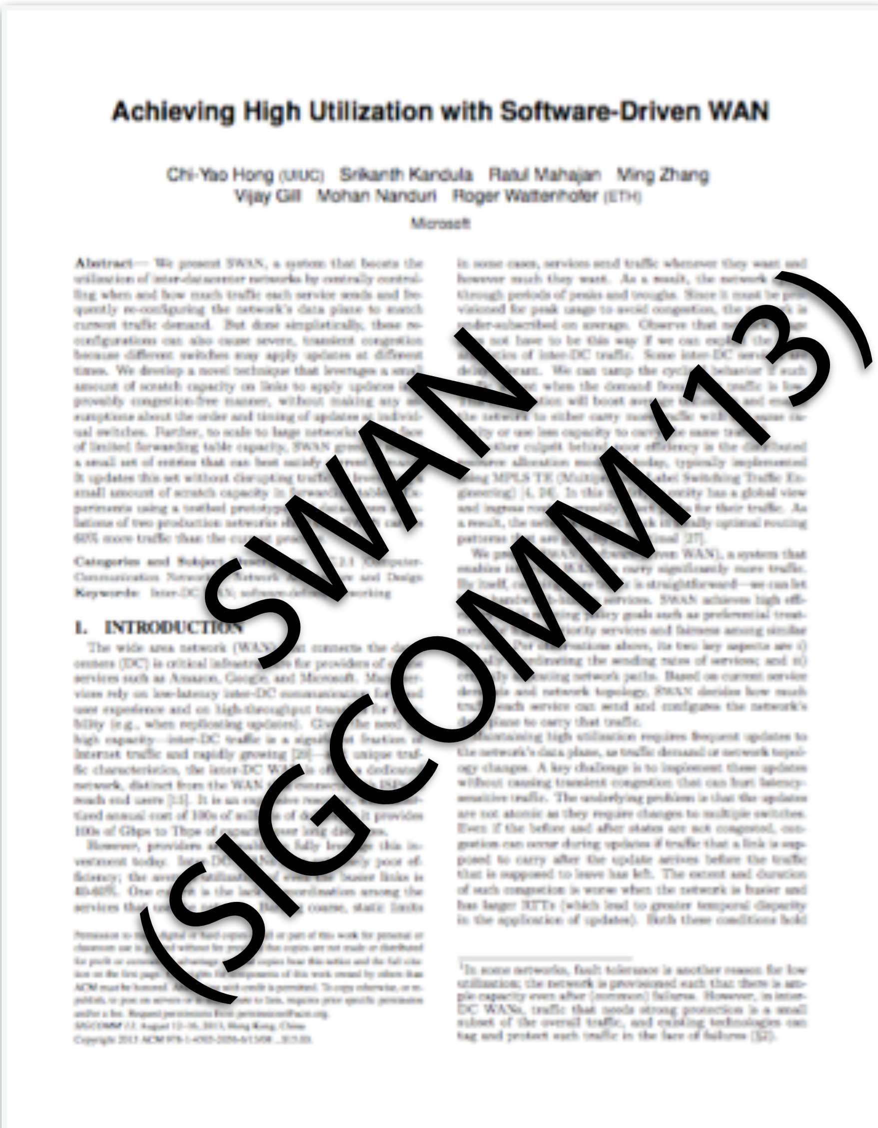
```
(* Map from src-dst pairs to traffic demands *)
type demands = float SrcDstMap.t

(* Map from src-dst pairs to path distributions *)
type scheme = (float PathMap.t) SrcDstMap.t

module type Algorithm = sig
  val initialize : scheme -> unit
  val solve : topology -> demands -> scheme
end
```

YATES Interface

High-level and modular interface

```
(* Map from src-dst pairs to traffic demands *)
type demands = float SrcDstMap.t

(* Map from src-dst pairs to path distributions *)
type scheme = (float PathMap.t) SrcDstMap.t

module type Algorithm = sig
  val initialize : scheme -> unit
  val solve : topology -> demands -> scheme
end
```

Modular Implementation

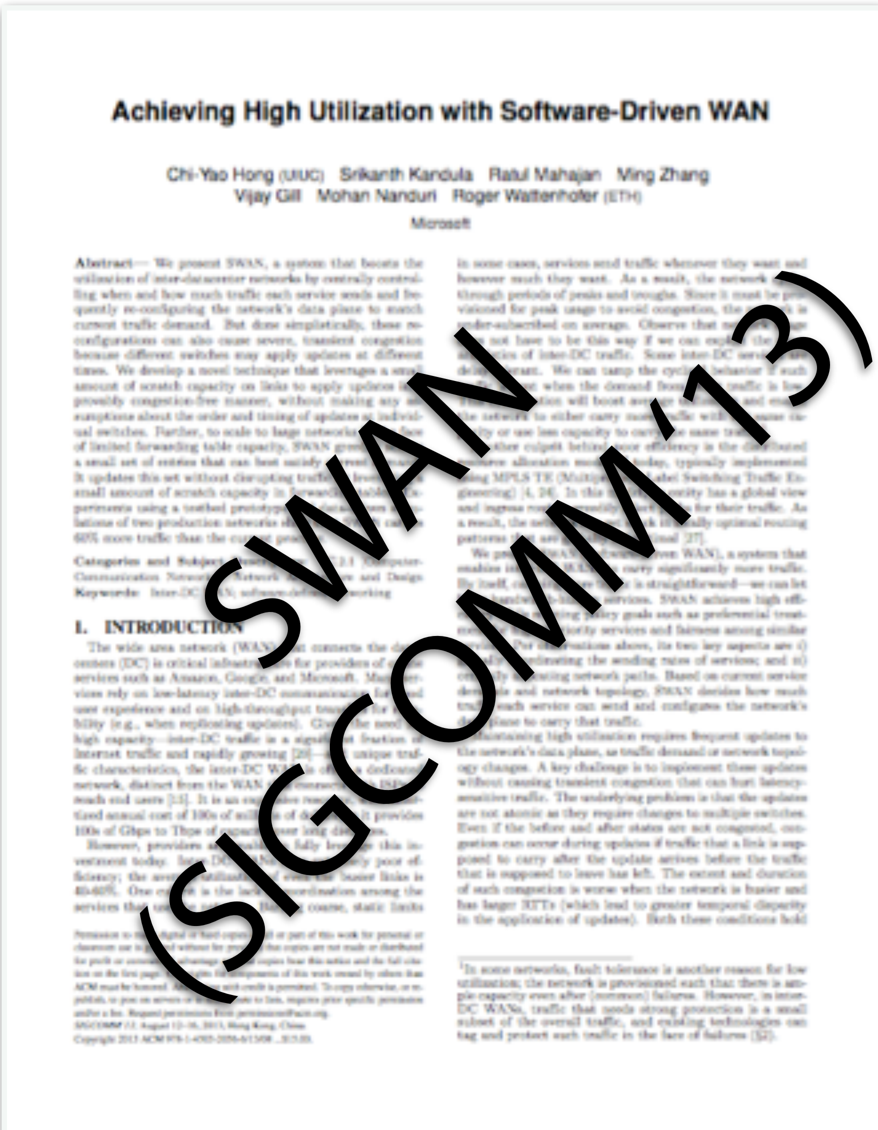


Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UIUC) Srikanth Kandula Ratul Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)
Microsoft

Simplified version: dynamically load-balance (MCF) traffic over k-shortest paths (KSP)

Modular Implementation


Achieving High Utilization with Software-Driven WAN

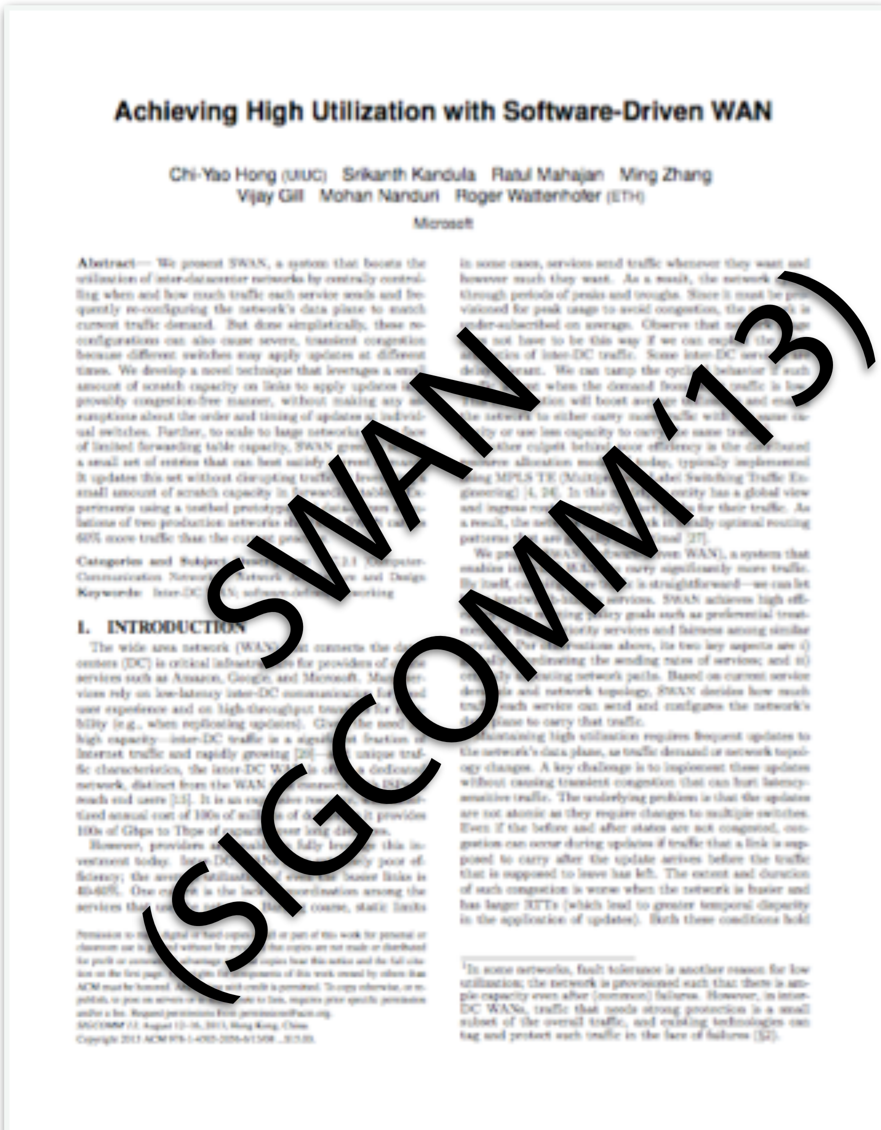
Chi-Yao Hong (UIUC) Srikanth Kandula Ratul Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)
Microsoft

1. INTRODUCTION


```
(* YATES modules *)
module KSP : Algorithm
module SemiMCF : Algorithm

(* Compute base set of k-shortest paths *)
let initial_scheme : scheme =
  KSP.initialize empty_scheme;
  KSP.solve topo empty_demands

(* Initialize SemiMCF with k-shortest paths *)
let () = SemiMCF.initialize initial_scheme in

(* For each traffic matrix, *)
let simulate_step (d:demands) : unit =
  (* Compute updated routing scheme *)
  let (s:scheme) = SemiMCF.solve topo d in
  (* Record performance statistics ... *)
```

Simplified version: dynamically load-balance (MCF) traffic over k-shortest paths (KSP)

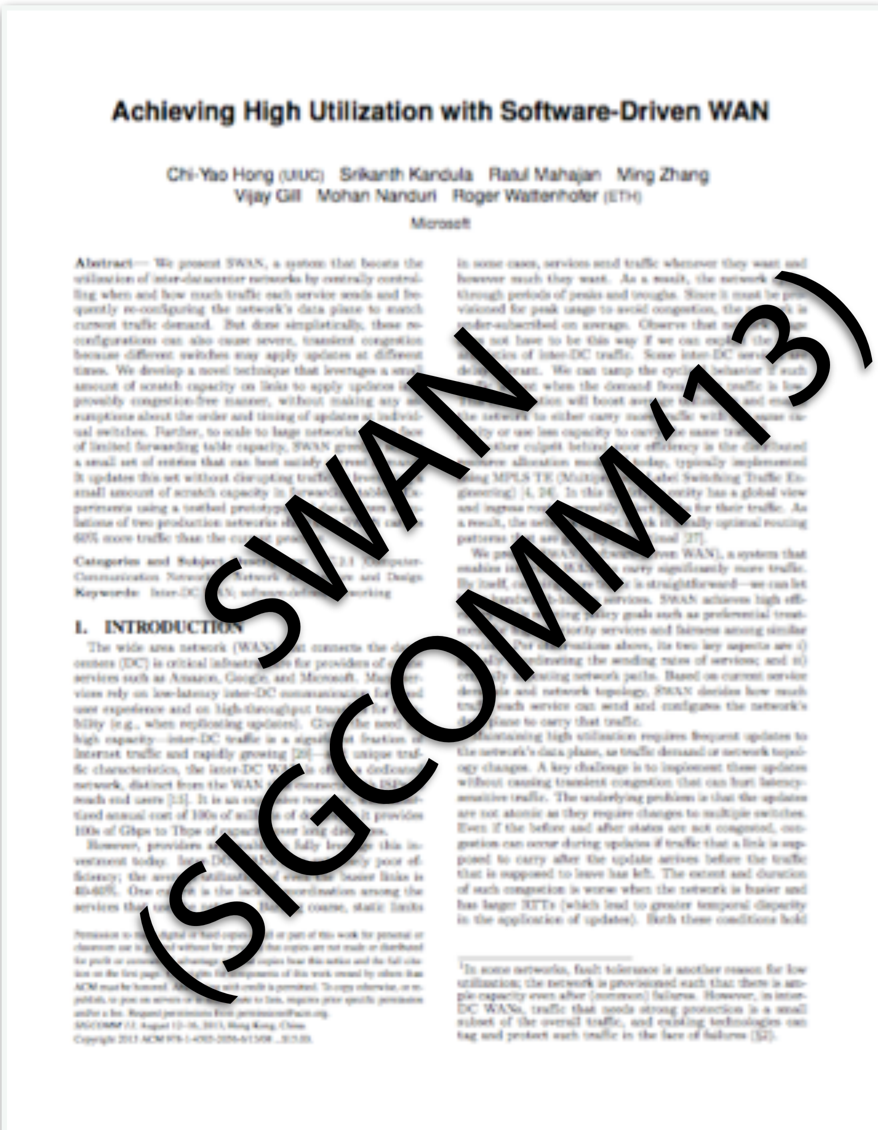
Modular Implementation




```
(* YATES modules *)
module KSP : Algorithm
module SemiMCF : Algorithm

(* Compute base set of k-shortest paths *)
let initial_scheme : scheme =
  KSP.initialize empty_scheme;
  KSP.solve topo empty_demands

(* Initialize SemiMCF with k-shortest paths *)
let () = SemiMCF.initialize initial_scheme in

(* For each traffic matrix, *)
let simulate_step (d:demands) : unit =
  (* Compute updated routing scheme *)
  let (s:scheme) = SemiMCF.solve topo d in
  (* Record performance statistics ... *)
```

Simplified version: dynamically load-balance (MCF) traffic over k-shortest paths (KSP)

Modular Implementation

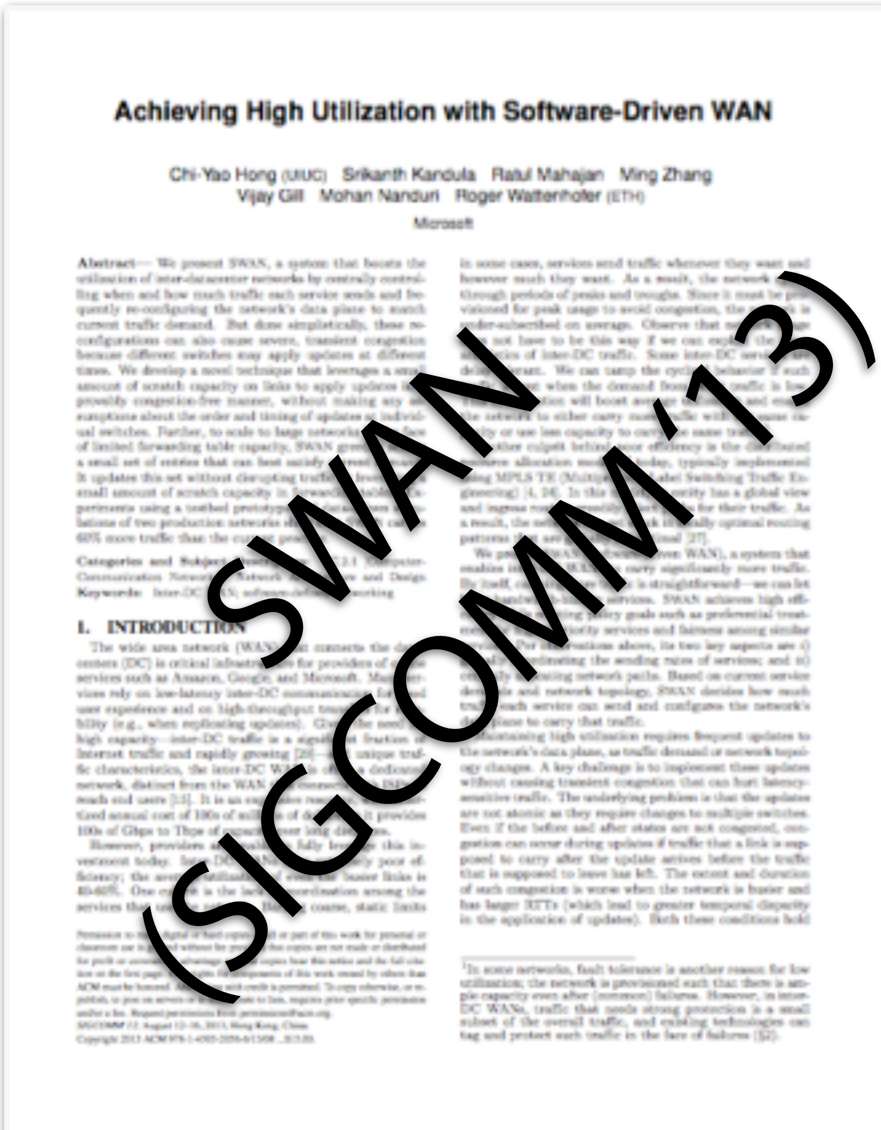

Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UIUC) Srikanth Kandula Ratul Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)
Microsoft


```
(* YATES modules *)
module KSP : Algorithm
module SemiMCF : Algorithm

(* Compute base set of k-shortest paths *)
let initial_scheme : scheme =
  KSP.initialize empty_scheme;
  KSP.solve topo empty_demands

(* Initialize SemiMCF with k-shortest paths *)
let () = SemiMCF.initialize initial_scheme in

(* For each traffic matrix, *)
let simulate_step (d:demands) : unit =
  (* Compute updated routing scheme *)
  let (s:scheme) = SemiMCF.solve topo d in
  (* Record performance statistics ... *)
```

Simplified version: dynamically load-balance (MCF) traffic over k-shortest paths (KSP)

Modular Implementation



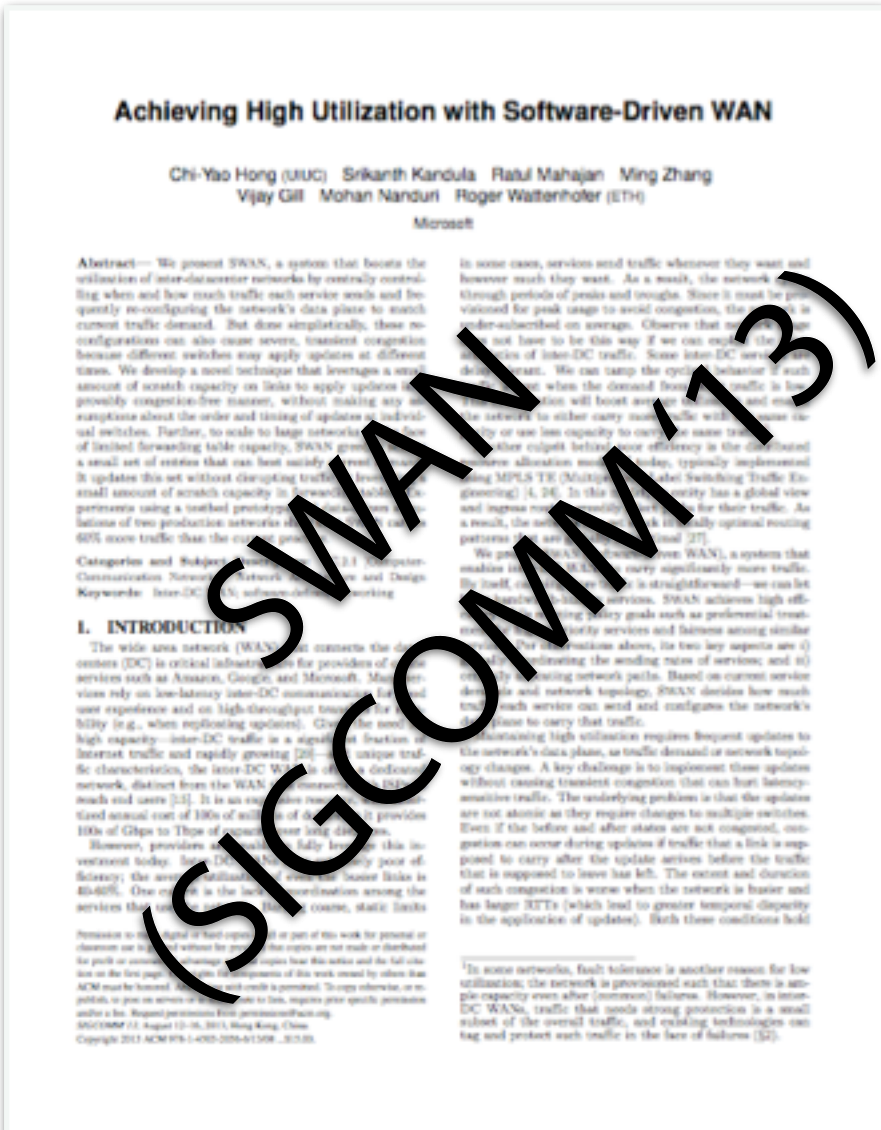
Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UIUC) Srikanth Kandula Ratul Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)
Microsoft

```
(* YATES modules *)
module KSP : Algorithm
module SemiMCF : Algorithm

(* Compute base set of k-shortest paths *)
let initial_scheme : scheme =
  KSP.initialize empty_scheme;
  KSP.solve topo empty_demands

(* Initialize SemiMCF with k-shortest paths *)
let () = SemiMCF.initialize initial_scheme in

(* For each traffic matrix, *)
let simulate_step (d:demands) : unit =
  (* Compute updated routing scheme *)
  let (s:scheme) = SemiMCF.solve topo d in
  (* Record performance statistics ... *)
```

Simplified version: dynamically load-balance (MCF) traffic over k-shortest paths (KSP)

Modular Implementation

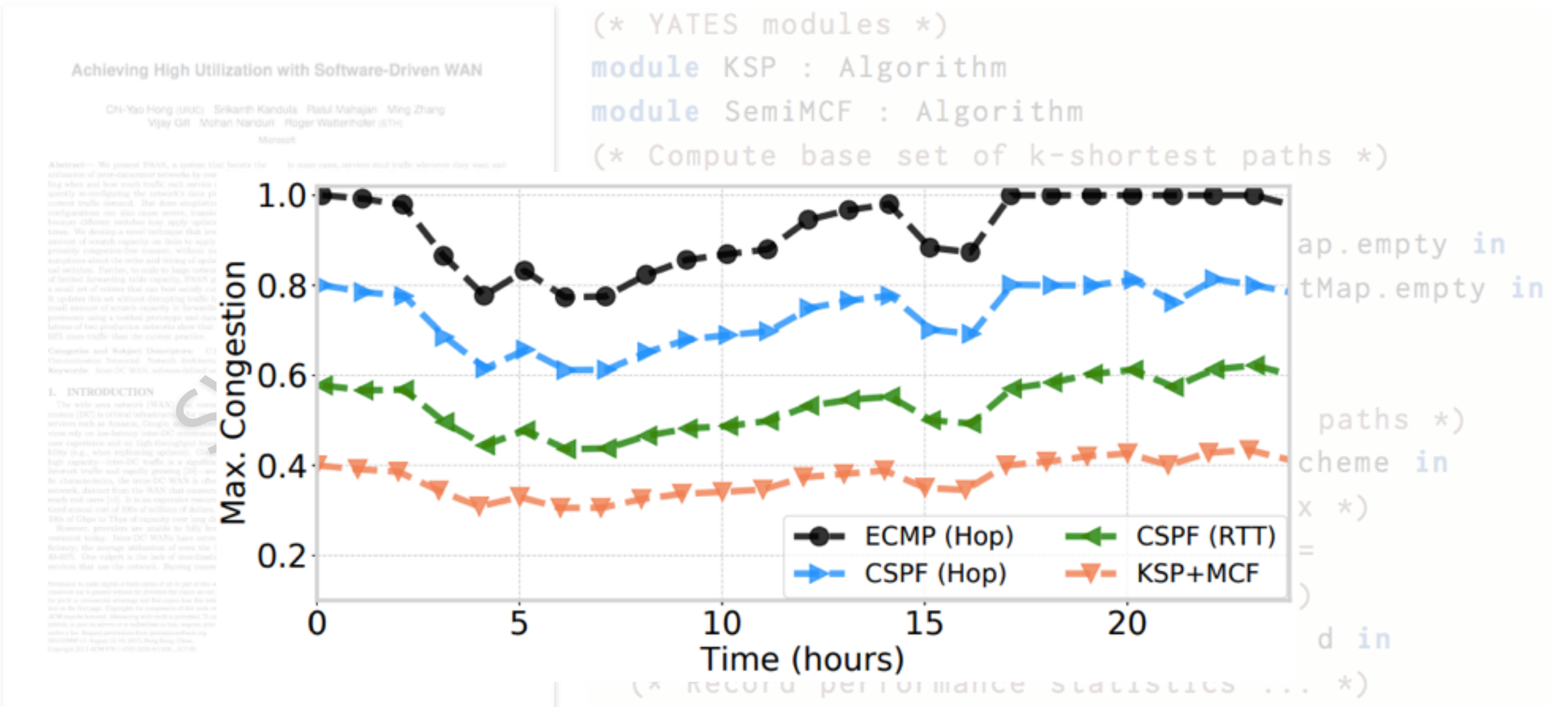

Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UIUC) Srikanth Kandula Ratul Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)
Microsoft


```
(* YATES modules *)
module KSP : Algorithm
module SemiMCF : Algorithm

(* Compute base set of k-shortest paths *)
let initial_scheme : scheme =
  KSP.initialize empty_scheme;
  KSP.solve topo empty_demands

(* Initialize SemiMCF with k-shortest paths *)
let () = SemiMCF.initialize initial_scheme in

(* For each traffic matrix, *)
let simulate_step (d:demands) : unit =
  (* Compute updated routing scheme *)
  let (s:scheme) = SemiMCF.solve topo d in
  (* Record performance statistics ... *)
```

Simplified version: dynamically load-balance (MCF) traffic over k-shortest paths (KSP)

Modular Implementation



Simplified version: dynamically load-balance (MCF) traffic over k-shortest paths (KSP)

YATES Framework

Software Infrastructure

- High-level abstractions
  - Library of 18 different TE approaches
- Tools for modeling operational conditions

Simulator

- Scalable
- Steady-state
- Macroscopic properties

SDN Backend

- Implementations: static paths and source routing

YATES Framework

Software Infrastructure

- High-level abstractions
  - Library of 18 different TE approaches
- Tools for modeling operational conditions

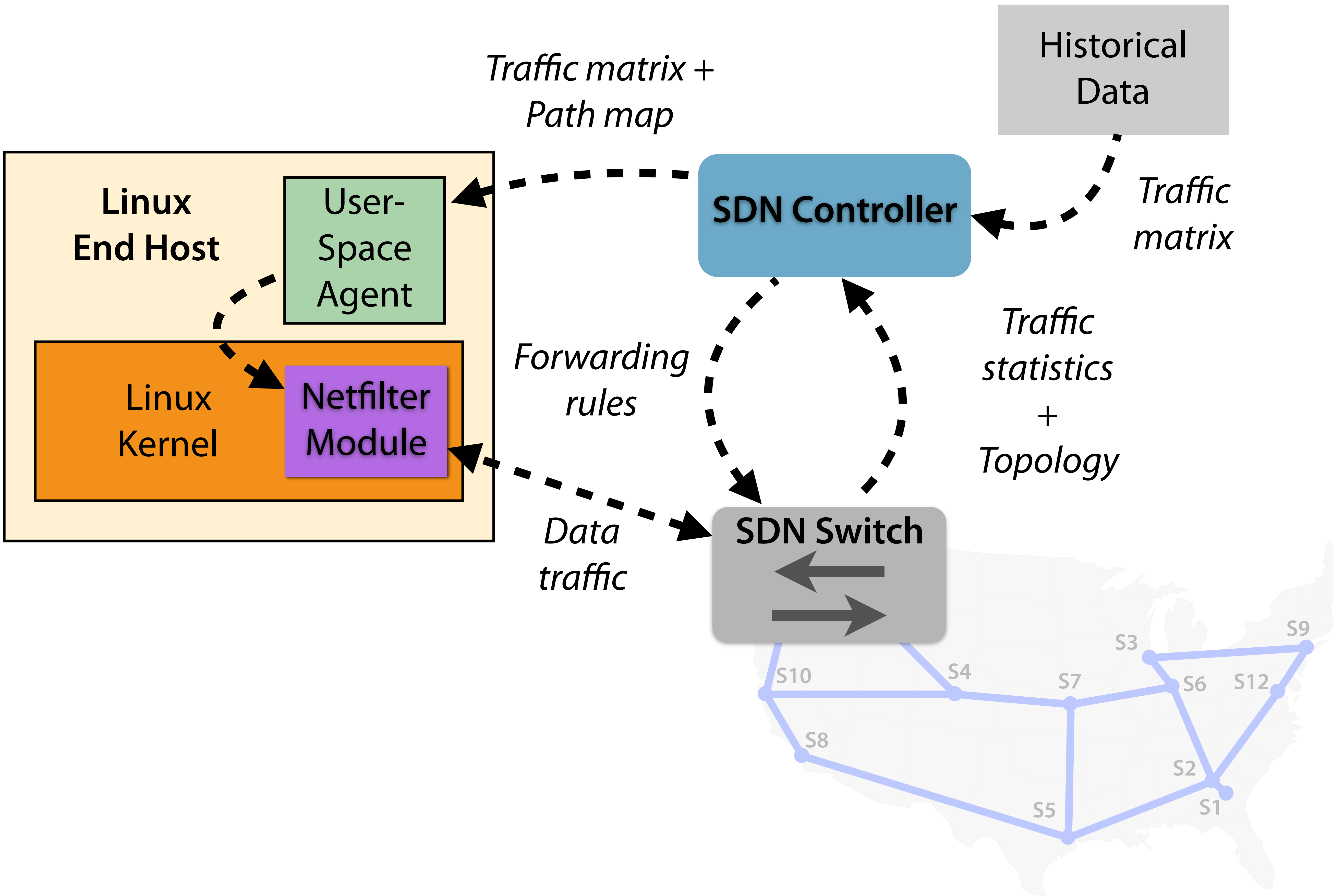
Simulator

- Scalable
- Steady-state
- Macroscopic properties

SDN Backend

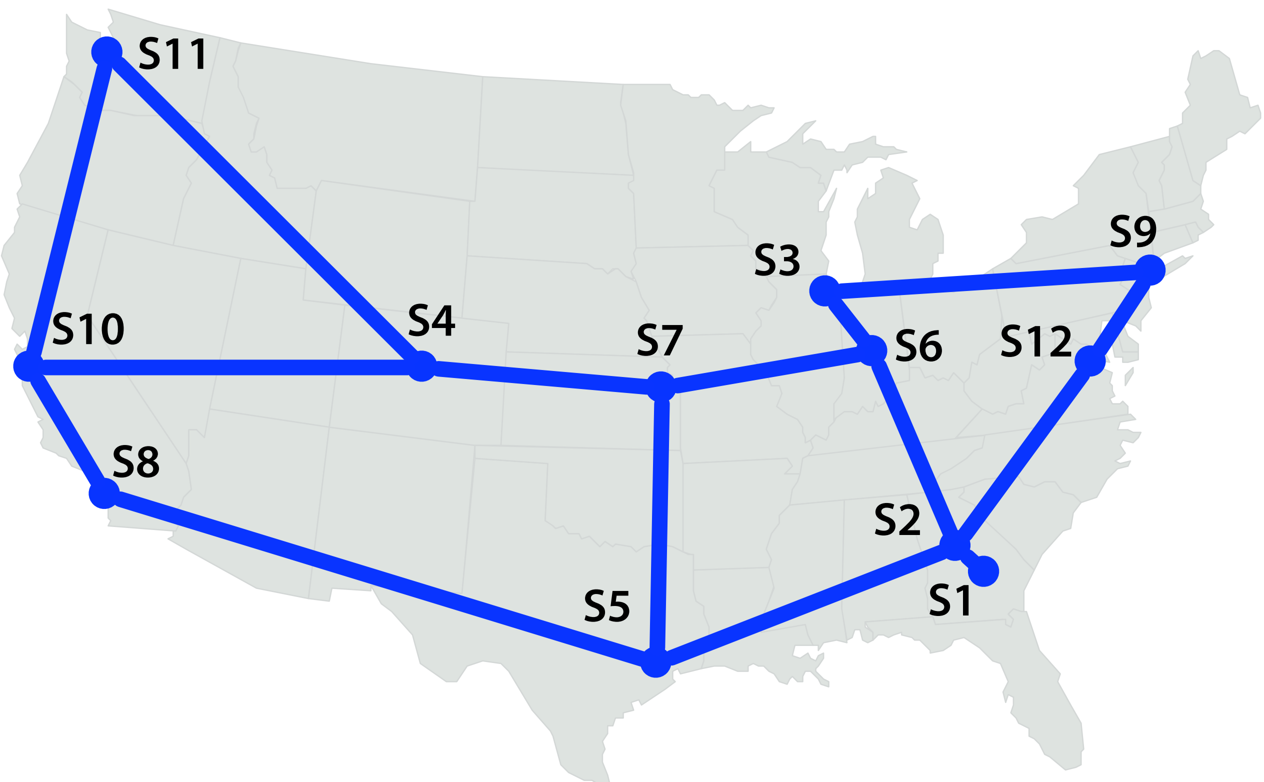
- Implementations: static paths and source routing

SDN Backend

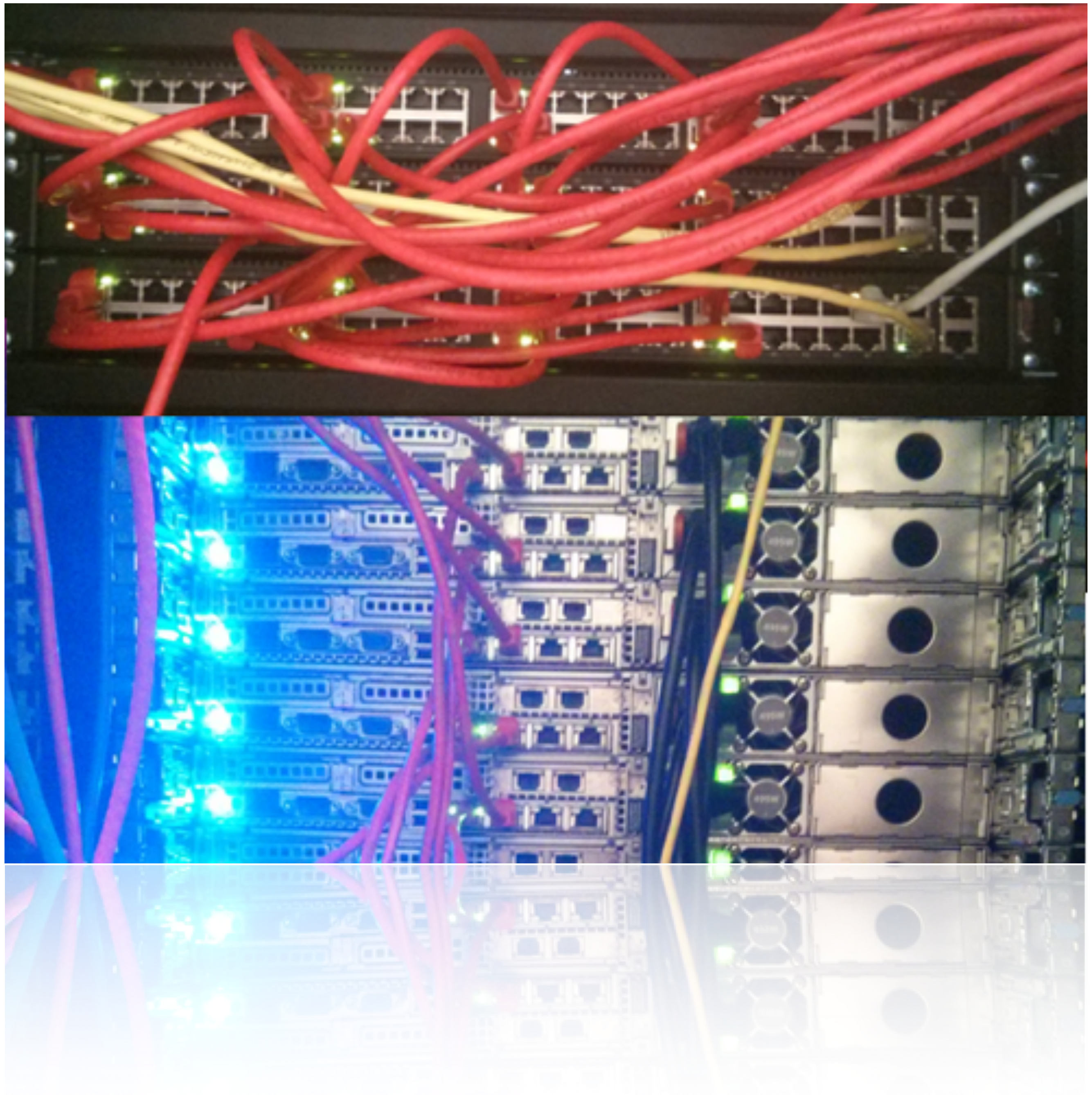


Historical
Data
Traffic matrix +
Path map
Traffic
matrix
Linux
End Host
User-
Space
Agent
SDN Controller
Linux
Kernel
Netfilter
Module
Forwarding
rules
Traffic
statistics
+
Topology
Data
traffic
SDN Switch
S3
S9
S10
S4
S7
S6
S12
S8
S2
S5
S1

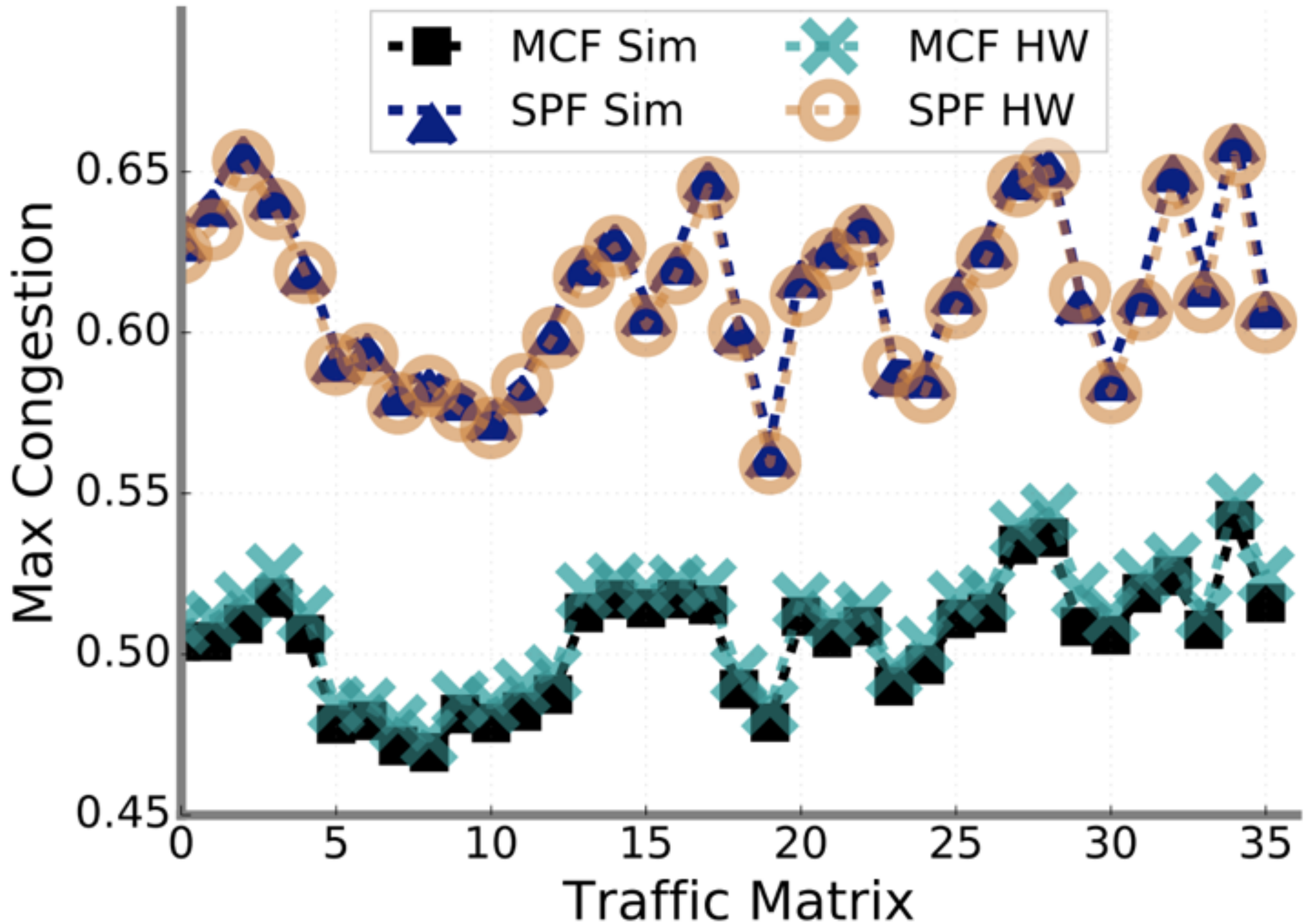
Hardware Testbed



Abilene Backbone Network



Simulator Calibration



PCC = 0.996

Take Aways

- YATES lowers the bar to perform credible TE research
- **Easy to prototype** and evaluate new TE systems
- **Modular design** allows composing pieces together
- Tools to model **operational conditions**
- **Calibrated** with deployments to ensure credible results

Questions?

Try out YATES!

https://github.com/cornell-netlab/yates