

YATES: Rapid Prototyping for Traffic Engineering Systems

Praveen Kumar (Cornell)

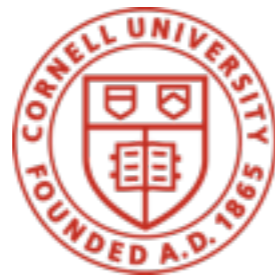
Yang Yuan (Cornell)

Chris Yu (CMU)

Nate Foster (Cornell)

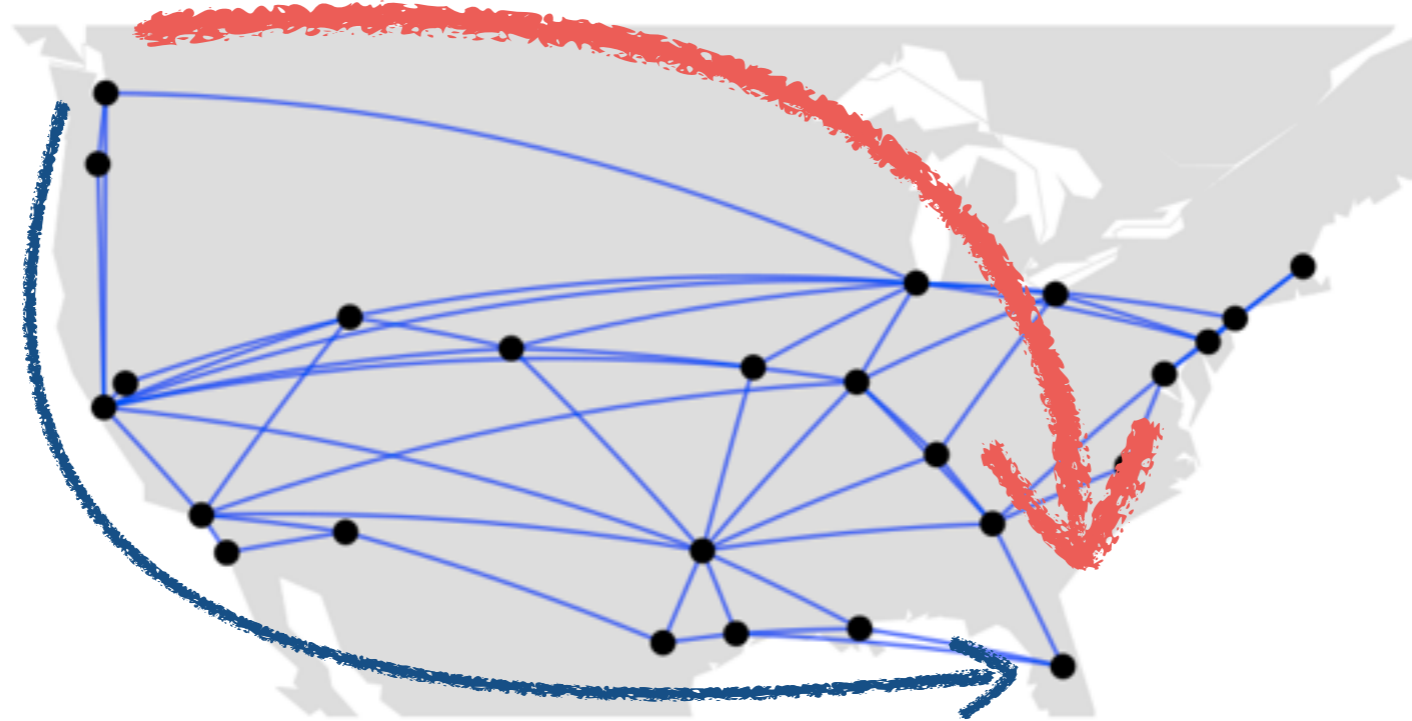
Robert Kleinberg (Cornell)

Robert Soulé (USI Lugano)

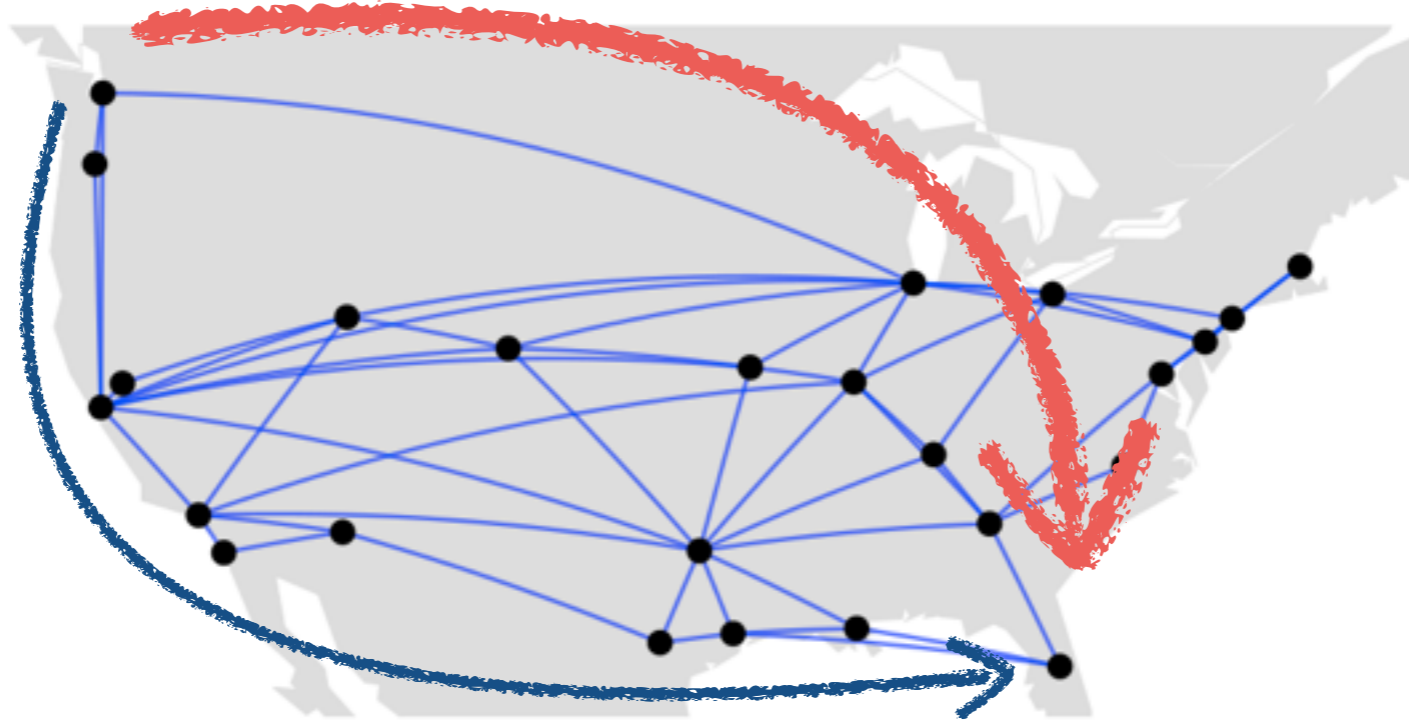


ACM SOSR 2018

WAN TE - Example



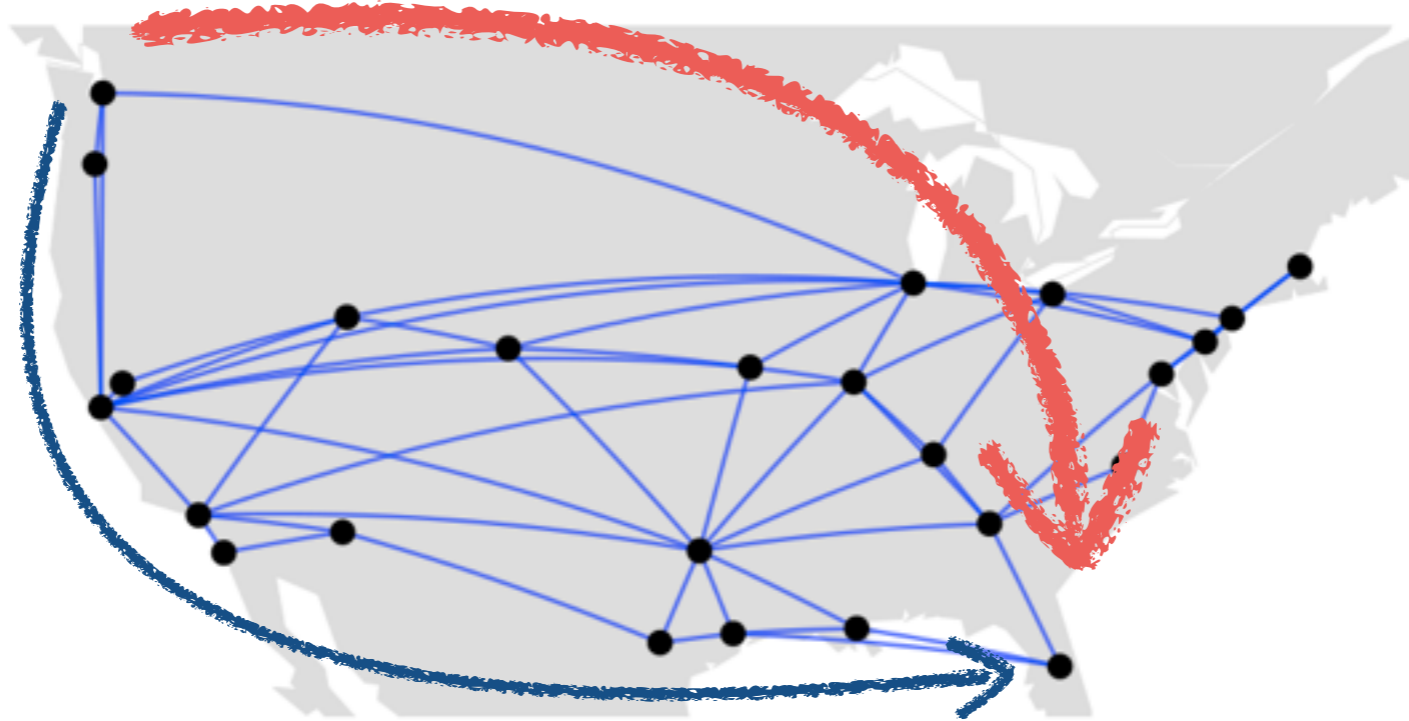
WAN TE - Example



Network Operator

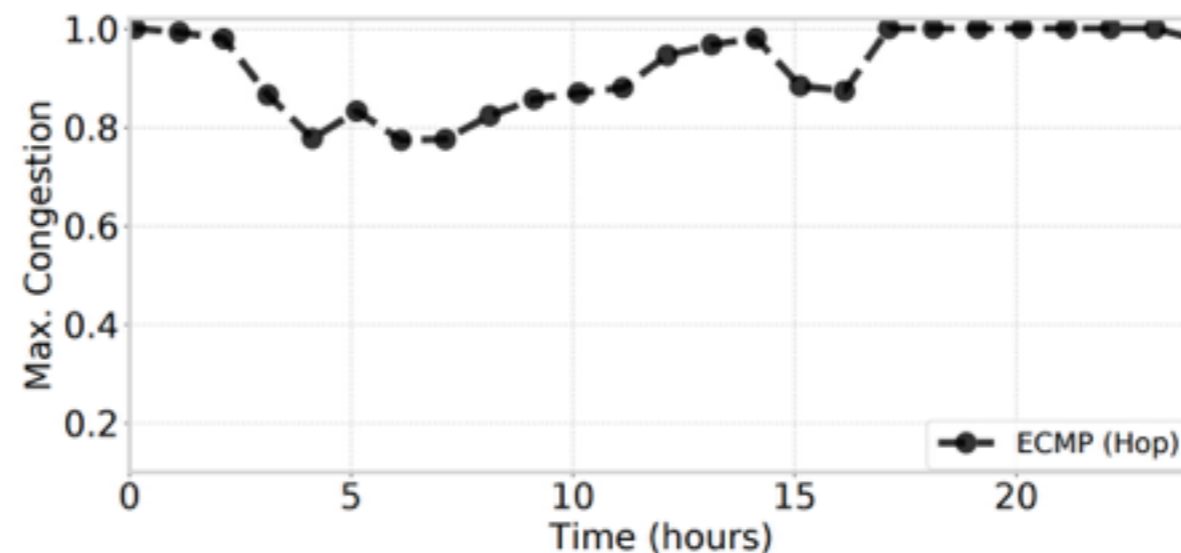
Configure the network to forward traffic using equal-cost multi-path (ECMP)

WAN TE - Example

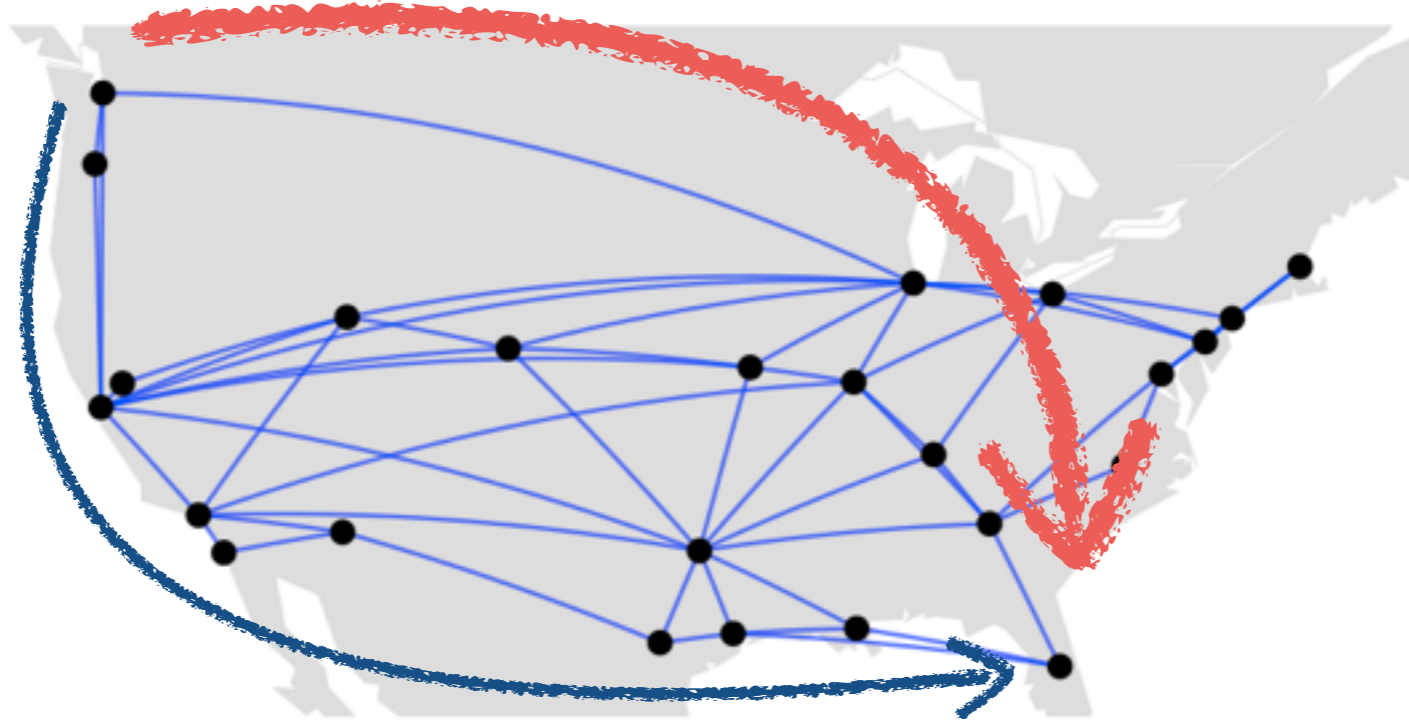


Network Operator

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



WAN TE - Example



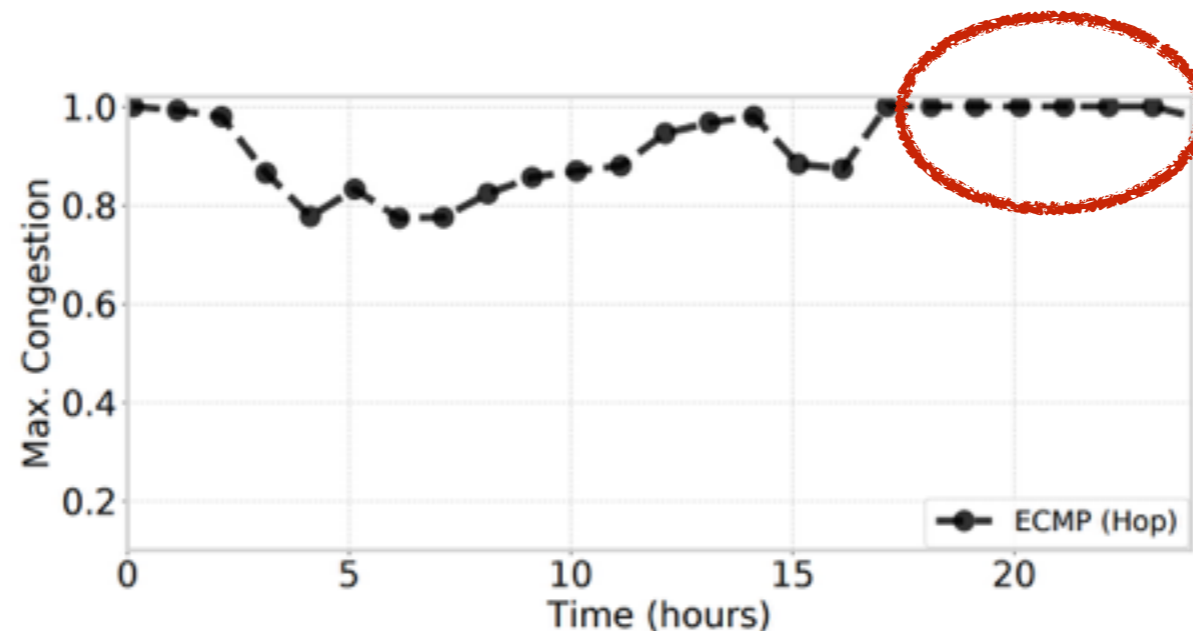
CSPF?



Network Operator

weights?

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



TE Systems

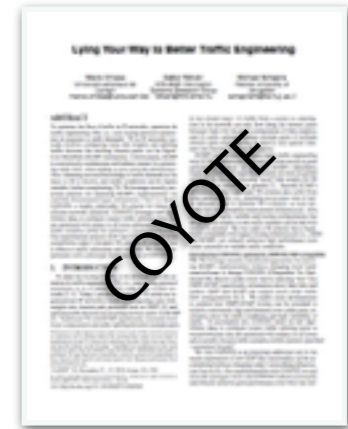
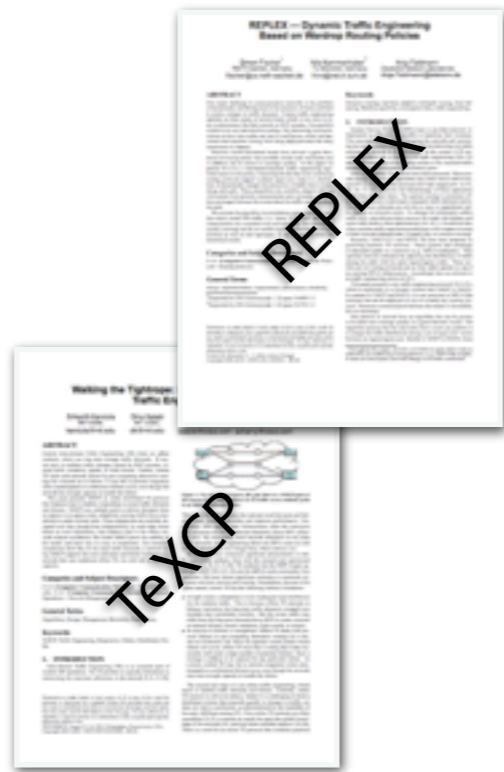
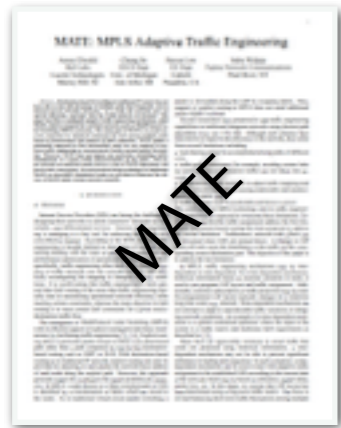
TE Systems

OSPF

CSPF

ECMP

MCF

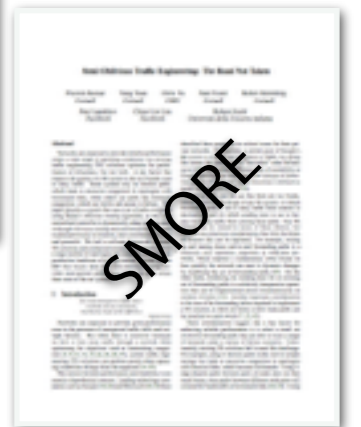
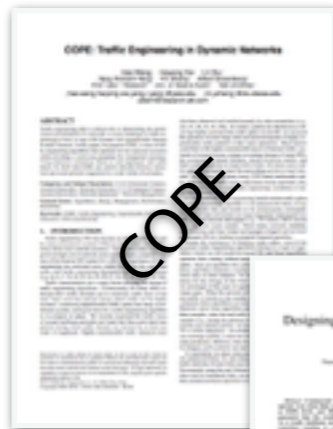


2000

2005

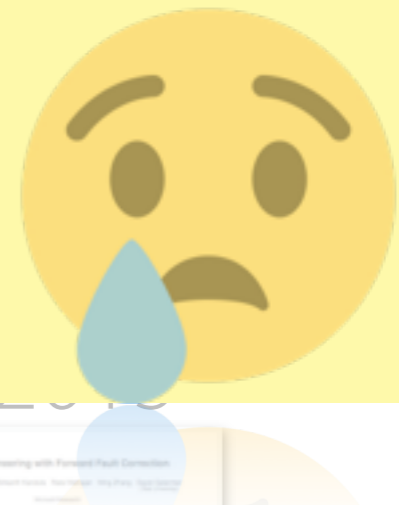
2010

2015



TE Systems

- Difficult to compare



OSPF

MATE

REPLEX

SWAN

COYOTE

ECMP

MCF

Oblivious

COPE

VLB

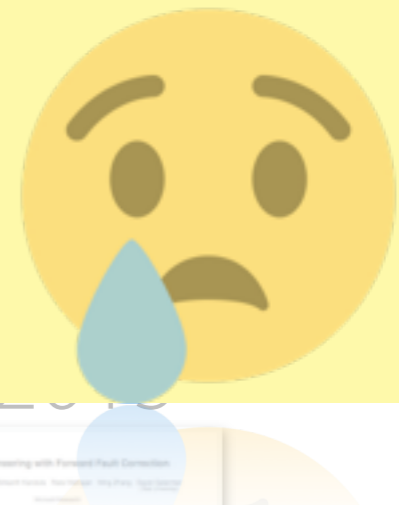
B4

FFC

SMORE

TE Systems

- Difficult to compare
- High cost of evaluation



OSPF

MATE

REPLEX

SWAN

COYOTE

ECMP

MCF

Oblivious

COPE

VLB

B4

FFC

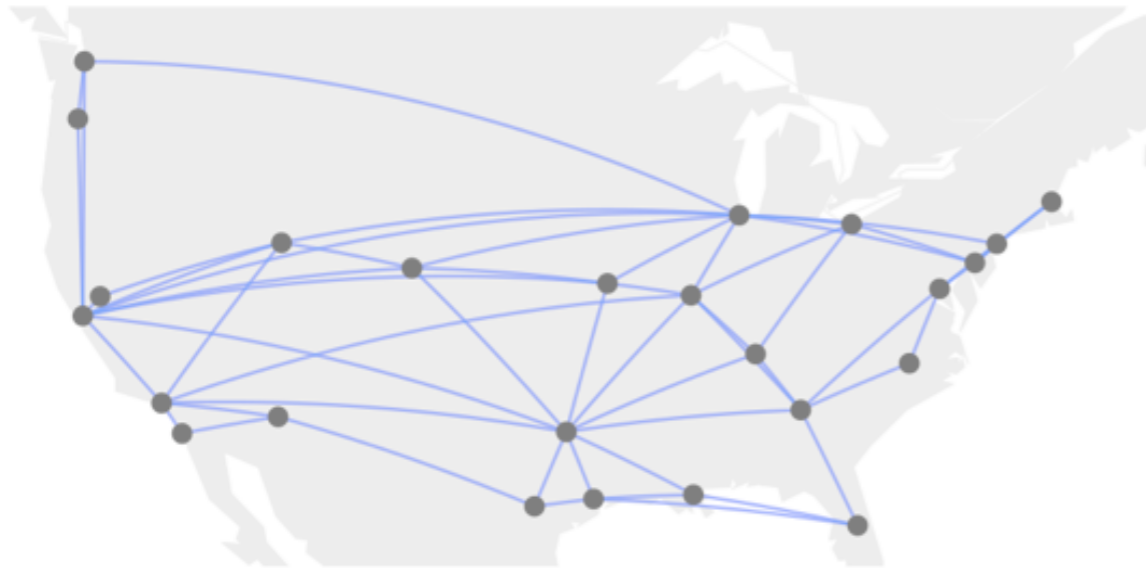
SMORE

YATES

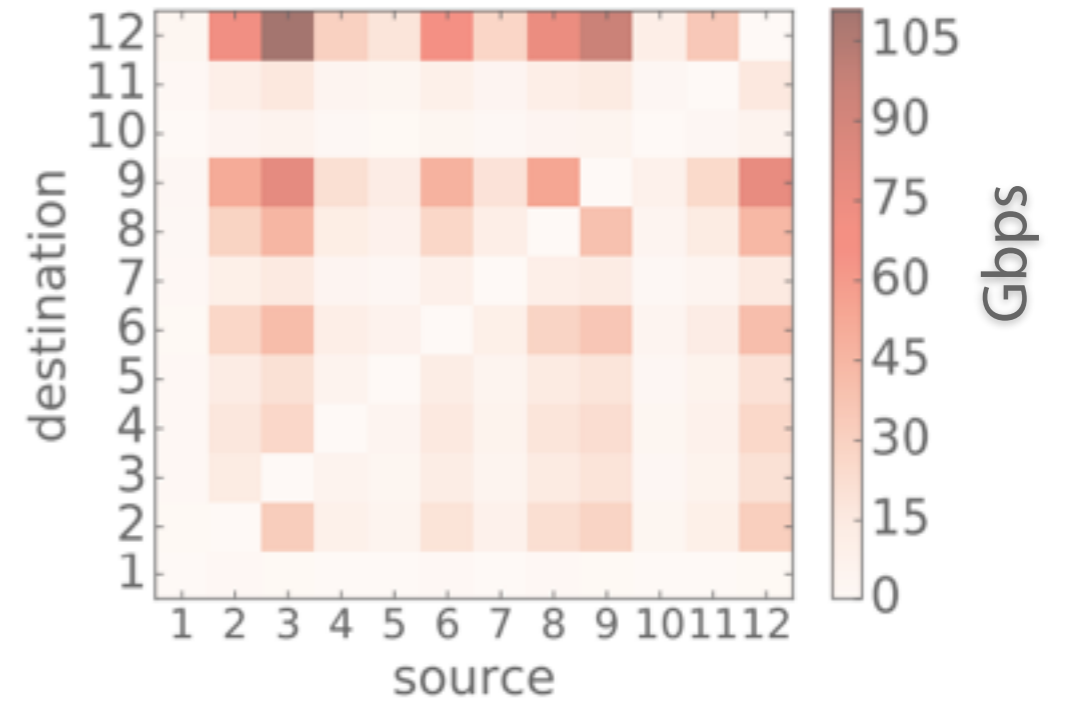
(Yet Another Traffic Engineering System)

- **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.

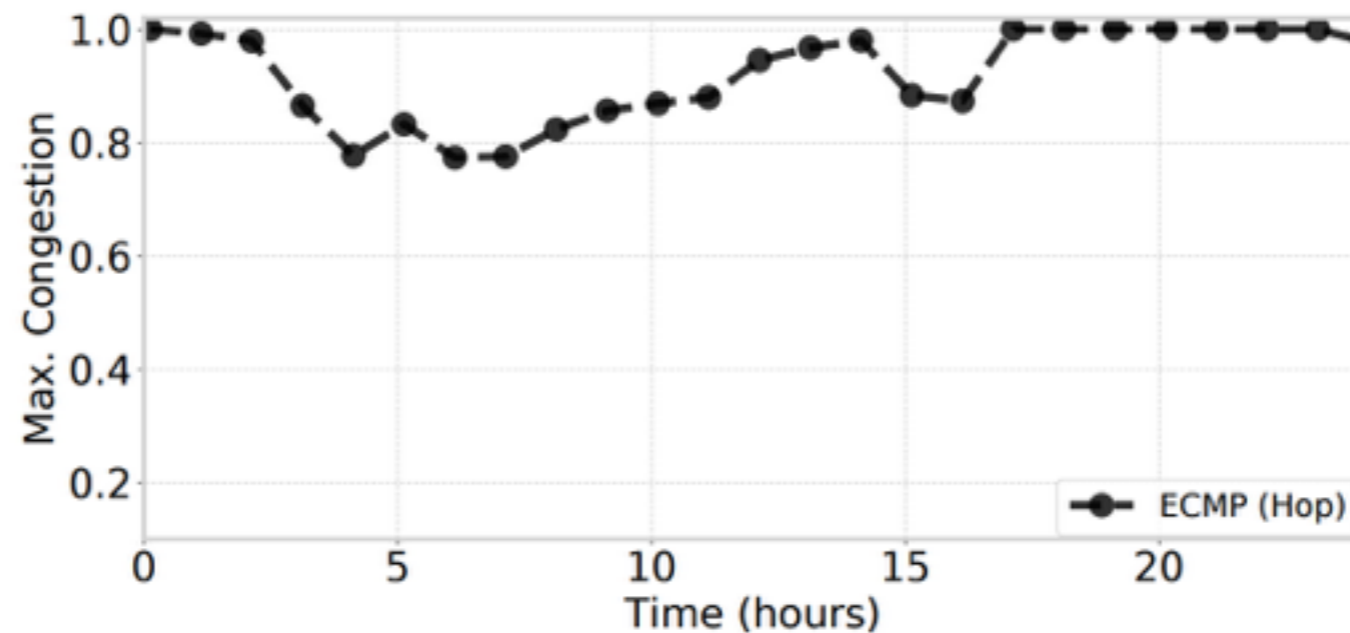
YATES - Example



+



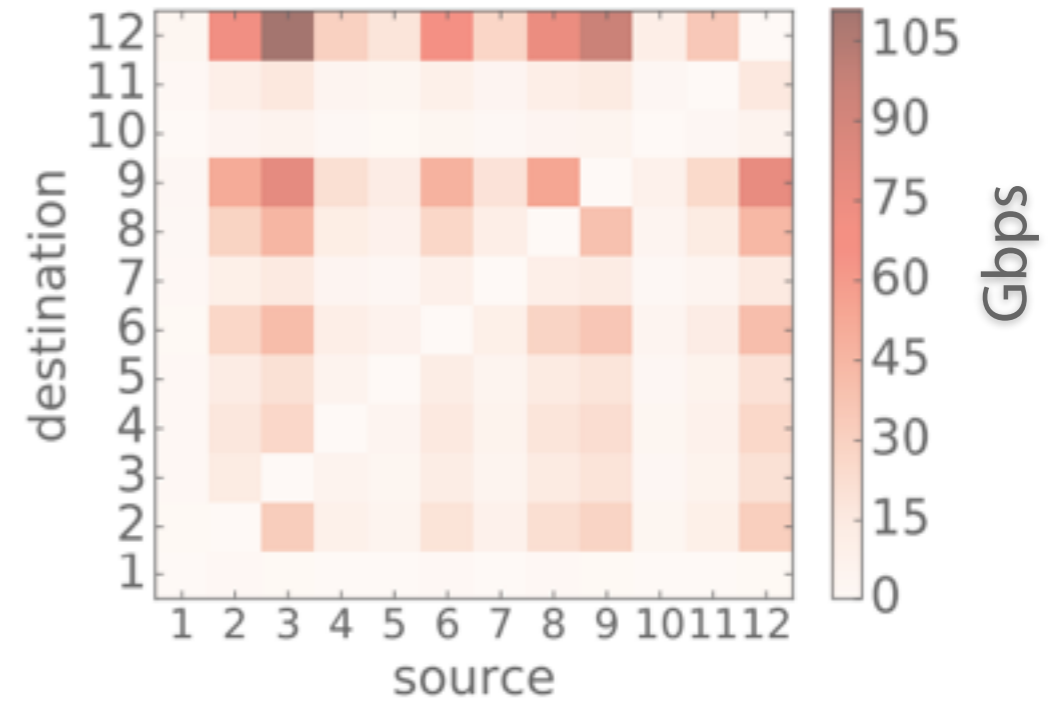
Network Operator



YATES - Example



+

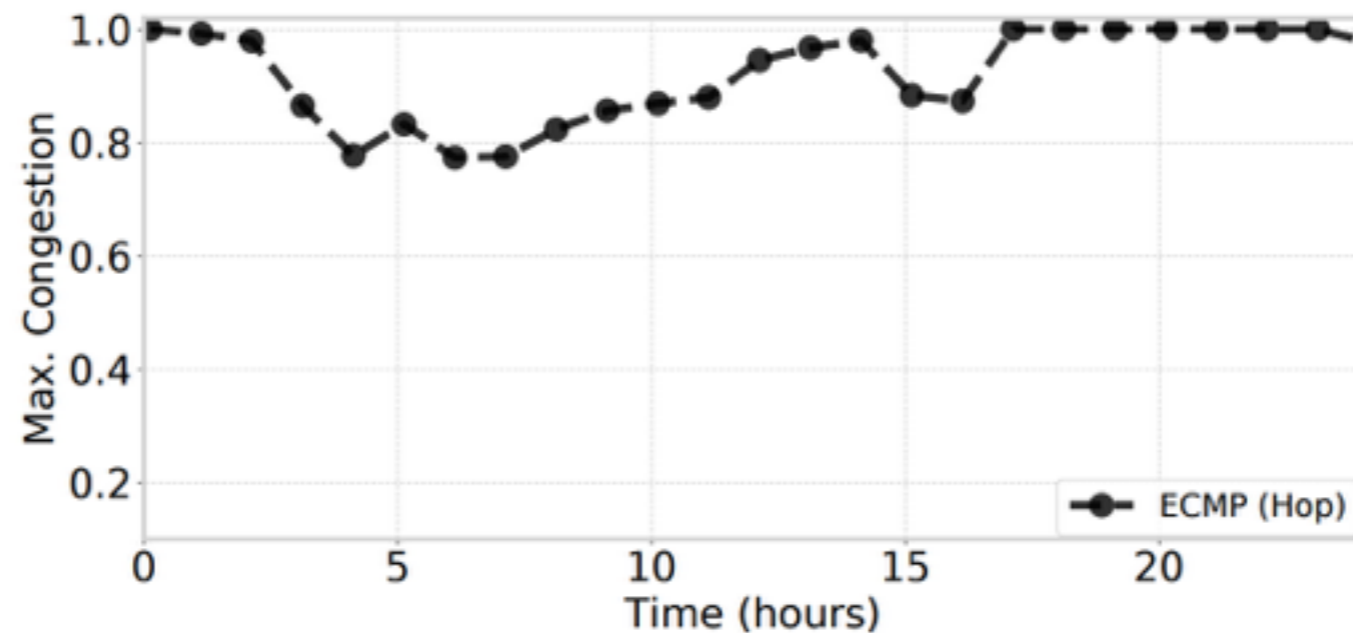


YATES > . . .

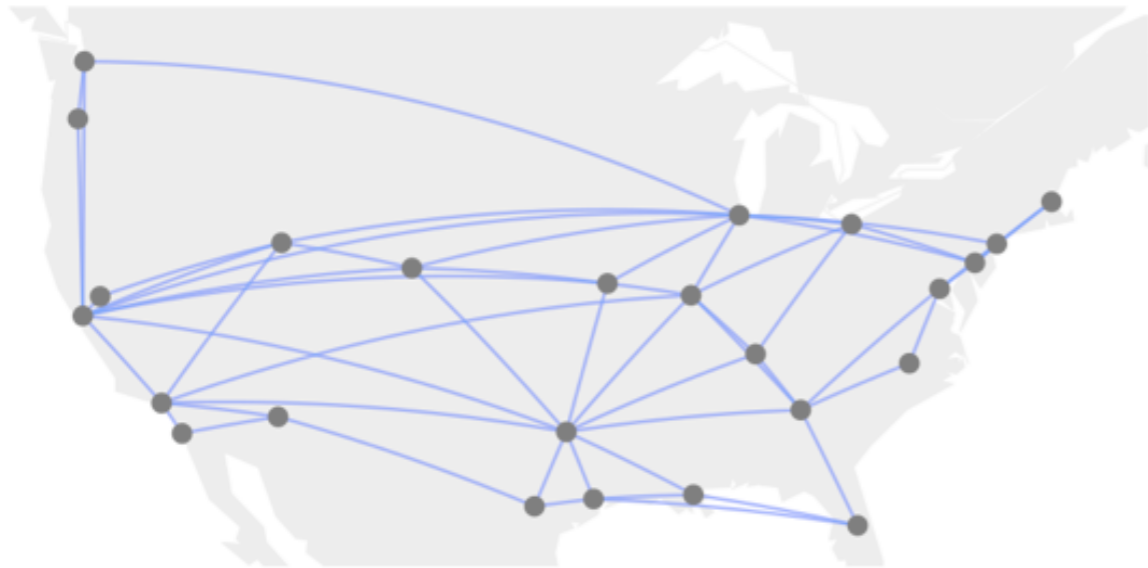
Network Operator

CSPF?

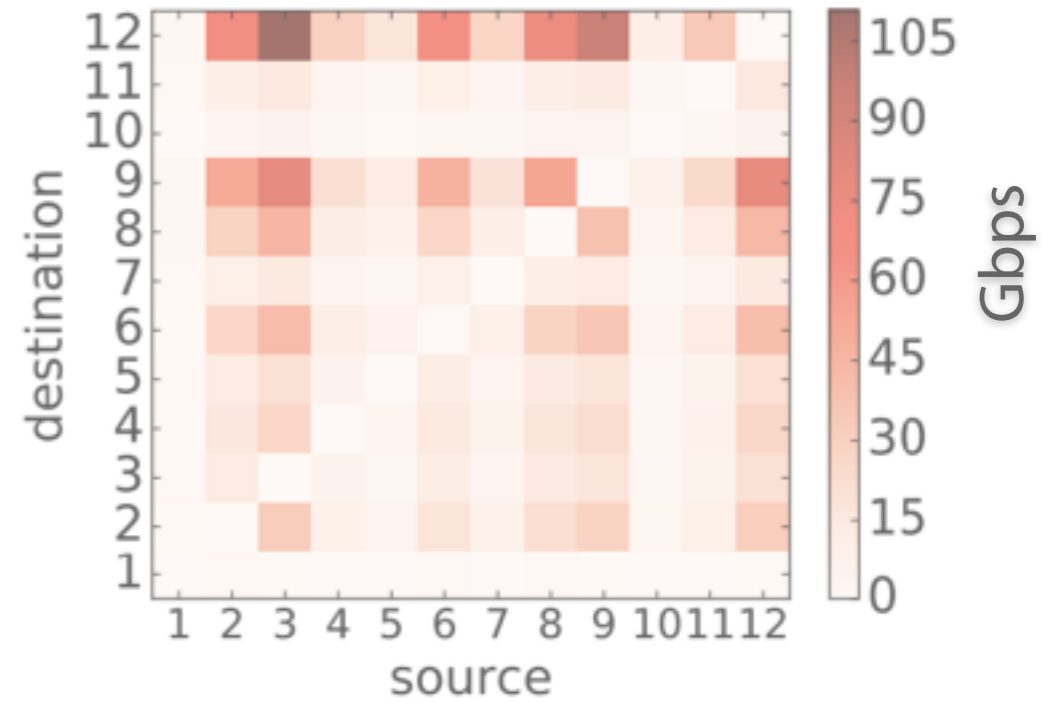
Link weight? (1 or RTT)



YATES - Example



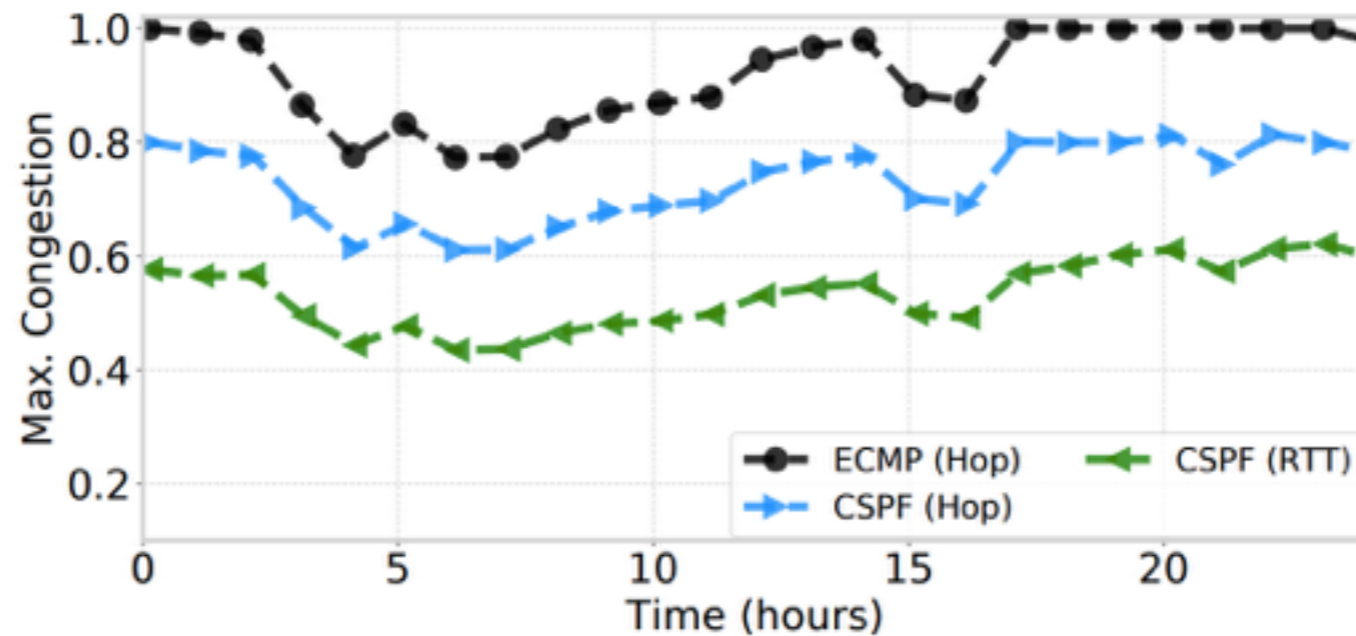
+



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 (UUC) Srikarath Kandula Rajul Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)

Microsoft

Abstract— We present SWAN, a system that boosts the utilization of inter-datacenter networks by centrally controlling when and how much traffic each service sends and frequently re-configuring the network's data plane to match current traffic demand. But done simplistically, these re-configurations can also cause severe, transient congestion because different switches may apply updates at different times. We develop a novel technique that leverages a small amount of scratch capacity on links to apply updates to previously congested links smoothly, without making any assumptions about the order and timing of updates on individual switches. Further, to scale to large networks with large amounts of limited forwarding table capacity, SWAN computes a small set of entries that can best satisfy previous traffic. It updates this set without disrupting traffic by incrementally adding a small amount of scratch capacity in a new update. We demonstrate using a unified prototype, SWAN's performance on two production networks that handle up to 60% more traffic than the current production network.

Categories and Subject Descriptors: C.2.1 Network Communication; Networks; Networks and Design
Keywords: Inter-DC, Software-Driven, Routing

1. INTRODUCTION

The wide area network (WAN) connects the data centers (DC) in critical industries for providers of services such as Amazon, Google, and Microsoft. Many services rely on low-latency inter-DC communication. Latency user experience and on high-throughput transactions (e.g., when replicating updates). Given the need for high capacity inter-DC traffic in a significant fraction of Internet traffic and rapidly growing WAN—unique traffic characteristics, the inter-DC WAN is a distinct network, distinct from the WAN for consumer devices, mobile devices, and users [2]. It is an extremely high capacity network used annual cost of 10s of billions of dollars. It provides 10s of Gbps to Tbps of capacity over long distances.

However, providers are unable to fully leverage this investment today. Inter-DC WANs are currently a poor of efficiency: the average utilization of inter-DC WAN links is 40-60%. One of the main reasons for this low utilization among the services that use inter-DC WANs is, of course, static links

in some cases, services send traffic whenever they want and however much they want. As a result, the network is under-utilized through periods of peaks and troughs. Since it must be provisioned for peak usage to avoid congestion, the network is under-subscribed on average. Observe that this is not how to be this way if we can exploit the dynamics of inter-DC traffic. Some inter-DC services are bursty. We can tame the periodic behavior of such services when the demand for traffic is low. We can also use low capacity to carry the same traffic in some other context behind our efficiency is the distributed nature of traffic allocation mechanisms, typically implemented using MPLS TE (Multi-Protocol Label Switching Traffic Engineering) [4, 26]. In this case, the network has a global view and ingress nodes can dynamically route traffic for their traffic. As a result, the network can dynamically adjust routing patterns to avoid congestion [27].

We present a software-driven WAN, a system that makes it possible to carry significantly more traffic. In fact, it is straightforward—we can let services send traffic whenever they want. SWAN achieves high efficiency by dynamically adjusting the sending rates of services, and by dynamically adjusting network paths. Based on current service demands and network topology, SWAN decides how much traffic each service can send and configures the network's data plane to carry that traffic.

Achieving high utilization requires frequent updates to the network's data plane, as traffic demand or network topology changes. A key challenge is to implement these updates without causing transient congestion that can hurt latency-sensitive traffic. The underlying problem is that the updates are not atomic as they require changes to multiple switches. Even if the before and after states are not congested, congestion can occur during updates if traffic that a link is supposed to carry after the update arrives before the traffic that is supposed to leave has left. The extent and duration of such congestion is worse when the network is busier and has larger RTTs (which lead to greater temporal disparity in the application of updates). Both these conditions hold

¹In some networks, flash tolerance is another reason for low utilization: the network is provisioned such that there is ample capacity even after (common) failures. However, in inter-DC WANs, traffic that needs strong protection is a small subset of the overall traffic, and existing techniques can tag and protect such traffic in the face of failures [32].

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 ... *)
```

Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UUC) Srikar Konda Ravi Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)
Microsoft

Abstract—We present SWAN, a system that boosts the utilization of inter-datacenter networks by centrally controlling when and how much traffic each service sends and frequently re-configuring the network's data plane to match current traffic demand. But done simplistically, these re-configurations can also cause severe, transient congestion because different switches may apply updates at different times. We develop a novel technique that leverages a small amount of scratch capacity on links to apply updates to previously congested links smoothly, without making any assumptions about the order and timing of updates of individual switches. Further, to scale to large networks with large amounts of limited forwarding table capacity, SWAN computes a small set of entries that can best satisfy all traffic demands. It updates this set without disrupting traffic by applying a small amount of scratch capacity in a distributed manner, using a unified protocol to coordinate updates across multiple production networks. Our evaluation shows that SWAN can handle 60% more traffic than the current practice.

Categories and Subject Descriptors: D.1.1 Software Construction; Networks; Network Architecture and Design
Keywords: Inter-DC, Software-Driven WAN

1. INTRODUCTION

The wide area network (WAN) connects the data centers (DC) in critical industries for providers of services such as Amazon, Google, and Microsoft. Many services rely on low-latency inter-DC communication. Low latency means low user experience and on high-throughput throughput (e.g., when replicating updates). Given the need for high capacity inter-DC traffic is a significant fraction of Internet traffic and rapidly growing [21]. Unlike traffic characteristics, the inter-DC WAN is a dedicated network, distinct from the WAN on the Internet. Inter-DC WANs are used for a small number of services, but they provide a small amount of bandwidth for a large number of services. However, providers are not fully leveraging this investment today. Inter-DC WANs are currently a poor of fit; the amount of bandwidth available per link is 40-60%. One of the main reasons for this is the lack of coordination among the services that use the network. In some cases, services send traffic whenever they want and however much they want. As a result, the network is congested during periods of peaks and troughs. Since it must be provisioned for peak usage to avoid congestion, the network is under-utilized on average. Observe that this is not true for the Internet, where the demand for traffic is highly variable. We can tame the cyclic behavior of such traffic by using a small amount of scratch capacity on links to apply updates to previously congested links smoothly, without making any assumptions about the order and timing of updates of individual switches. Further, to scale to large networks with large amounts of limited forwarding table capacity, SWAN computes a small set of entries that can best satisfy all traffic demands. It updates this set without disrupting traffic by applying a small amount of scratch capacity in a distributed manner, using a unified protocol to coordinate updates across multiple production networks. Our evaluation shows that SWAN can handle 60% more traffic than the current practice.

¹In some networks, fault tolerance is another reason for low utilization; the network is provisioned such that there is ample capacity even after (common) failures. However, in inter-DC WANs, traffic that needs strong protection is a small subset of the overall traffic, and existing techniques can tag and protect such traffic in the face of failures [22].

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

Modular Implementation

```
(* YATES modules *)
```

```
module KSP : Algorithm
```

```
module SemiMCF : Algorithm
```

Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UUC) Srikanth Kandula Ravi Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)

Microsoft

Abstract— We present SWAN, a system that boosts the utilization of inter-datacenter networks by centrally controlling when and how much traffic each service sends and frequently re-configuring the network's data plane to match current traffic demand. But done simplistically, these re-configurations can also cause severe, transient congestion because different switches may apply updates at different times. We develop a novel technique that leverages a small amount of scratch capacity on links to apply updates to previously congested links without making any assumptions about the order and timing of updates at individual switches. Further, to scale to large networks with large amounts of limited forwarding table capacity, SWAN computes a small set of entries that can best satisfy the current traffic. It updates this set without disrupting traffic, by using a small amount of scratch capacity in a distributed manner. We present a unified prototype implementation and evaluate two production networks that handle up to 60% more traffic than the current production network.

Categories and Subject Descriptors: D.1.1 Software Construction; Networks; Networks; Networks and Design
Keywords: Inter-DC, Software-Driven WAN

1. INTRODUCTION

The wide area network (WAN) connects the data centers (DC) in critical industries for providers of services such as Amazon, Google, and Microsoft. Many services rely on low-latency inter-DC communication. Latency user experience and on high-throughput throughput (e.g., when replicating updates). Given the need for high capacity—inter-DC traffic is a significant fraction of Internet traffic and rapidly growing [21]—unique traffic characteristics, the inter-DC WAN is a dedicated network, distinct from the WAN for consumer devices, mobile devices and users [23]. It is an expensive, high-speed network with annual cost of 10% of total operating expenses. It provides 100 Gbps of capacity per link and 100 Gbps of bandwidth per link. However, providers are often limited by this investment today. Inter-DC networks are often poorly utilized; the average utilization of inter-DC links is 40-60%. One of the main reasons for this under-utilization is the lack of coordination among the services that use the network. In some cases, static links

In some cases, services send traffic whenever they want and however much they want. As a result, the network is over-utilized through periods of peaks and troughs. Since it must be provisioned for peak usage to avoid congestion, the network is under-utilized on average. Observe that this is not true for all links of inter-DC traffic. Some inter-DC links are not congested. We can ramp the system behavior if such links are not congested. We can ramp the system behavior if such links are not congested. We can ramp the system behavior if such links are not congested.

We present a system, Software-Driven WAN (SWAN), a system that makes it possible to carry significantly more traffic. It is straightforward—we can let the network compute the best routing scheme for each service. SWAN achieves high utilization by dynamically adjusting the routing scheme for each service and latency among similar services. The contributions above, in two key aspects are: (i) dynamically adjusting network paths. Based on current service demands and network topology, SWAN decides how much traffic each service can send and configures the network's data plane to carry that traffic. (ii) managing high utilization requires frequent updates to the network's data plane, as traffic demand or network topology changes. A key challenge is to implement these updates without causing transient congestion that can hurt latency-sensitive traffic. The underlying problem is that the updates are not atomic as they require changes to multiple switches. Even if the before and after states are not congested, congestion can occur during updates if traffic that a link is supposed to carry after the update arrives before the traffic that is supposed to leave has left. The extent and duration of such congestion is worse when the network is busier and has larger RTTs (which lead to greater temporal disparity in the application of updates). Both these conditions hold

¹In some networks, fault tolerance is another reason for low utilization; the network is provisioned such that there is ample capacity even after (common) failures. However, in inter-DC WANs, traffic that needs strong protection is a small subset of the overall traffic, and existing techniques can tag and protect such traffic in the face of failures [32].

(SIGCOMM'13)

```
(* 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 *)

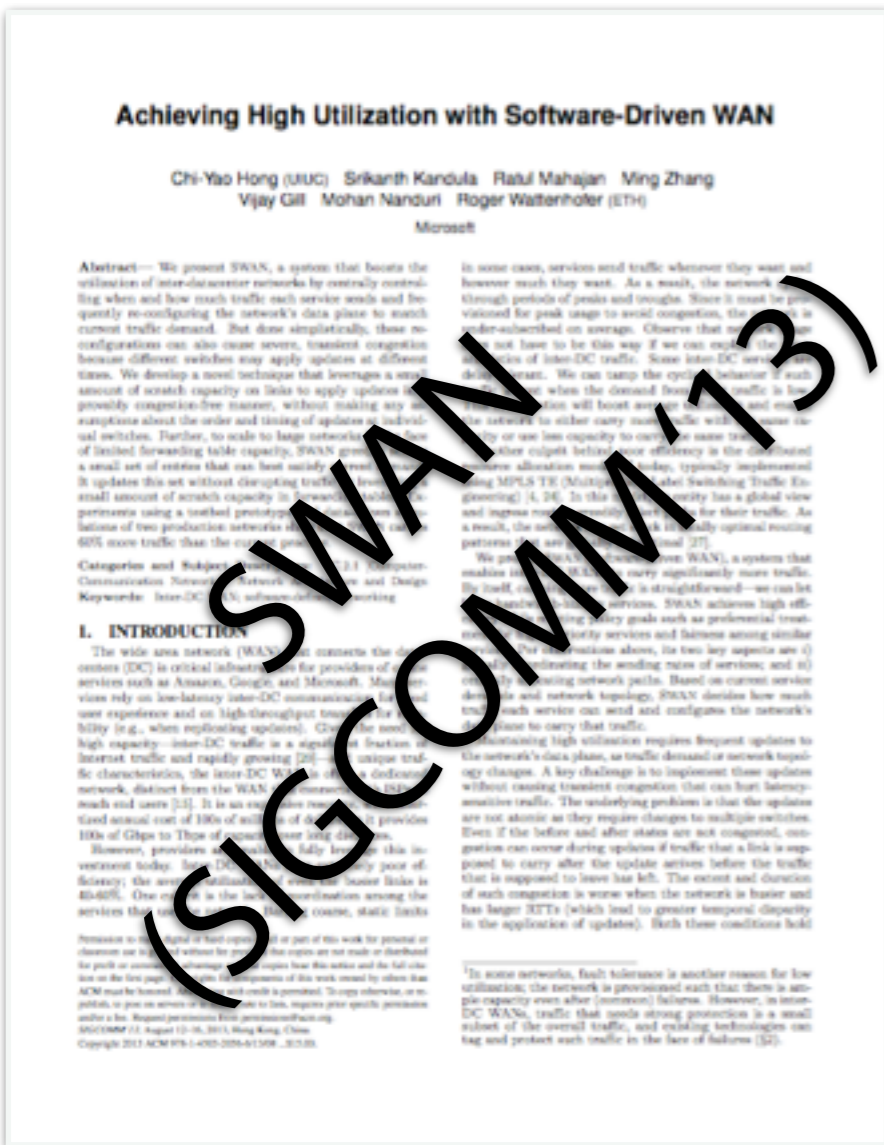
```
let initial_scheme : scheme =  
  KSP.initialize empty_scheme;  
  KSP.solve topo empty_demands
```

(* 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

(* 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 ... *)
```

Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UUC) Srikant Kandula Ravi Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)

Microsoft

Abstract— We present SWAN, a system that boosts the utilization of inter-datacenter networks by centrally controlling when and how much traffic each service sends and frequently re-configuring the network's data plane to match current traffic demand. But done simplistically, these re-configurations can also cause severe, transient congestion because different switches may apply updates at different times. We develop a novel technique that leverages a small amount of scratch capacity on links to apply updates to previously congested links smoothly, without making any assumptions about the order and timing of updates of individual switches. Further, to scale to large networks with large amounts of limited forwarding table capacity, SWAN computes a small set of entries that can be easily updated to reflect the current traffic demand. It updates this set without disrupting traffic by using a small amount of scratch capacity in the network. We demonstrate using a testbed prototype that SWAN can increase the utilization of two production networks by up to 60% more traffic than the current practice.

Categories and Subject Descriptors: C.2.1 [Communication]: Communication Networks; C.2.2 [Communication]: Network Protocols; C.2.3 [Communication]: Network Performance and Design

Keywords: Inter-DC, Software-Driven WAN

1. INTRODUCTION

The wide area network (WAN) connects the data centers (DC) in critical industries for providers of services such as Amazon, Google, and Microsoft. Many services rely on low-latency inter-DC communications. Latency is a key factor in user experience and on high-throughput transactional applications (e.g., when replicating updates). Given the high capacity inter-DC traffic is a significant fraction of Internet traffic and rapidly growing [21].

However, providers are not fully leveraging this investment today. Inter-DC links are often under-utilized. One of the main reasons for this is the lack of coordination among the services that use the network. In some cases, services send traffic whenever they want and

however much they want. As a result, the network is under-utilized through periods of peaks and troughs. Since it must be provisioned for peak usage to avoid congestion, the network is under-subscribed on average. Observe that this is not true for all links. Some links are over-subscribed. We can ramp the system behavior if such links are over-subscribed. We can ramp the system behavior if such links are over-subscribed. We can ramp the system behavior if such links are over-subscribed.

We present SWAN, a system that makes it easy to carry significantly more traffic. In fact, it is straightforward—we can let each service send traffic whenever they want and configure the network to carry that traffic.

To achieve high utilization requires frequent updates to the network's data plane, as traffic demand or network topology changes. A key challenge is to implement these updates without causing transient congestion that can hurt latency-sensitive traffic. The underlying problem is that the updates are not atomic as they require changes to multiple switches. Even if the before and after states are not congested, congestion can occur during updates if traffic that a link is supposed to carry after the update arrives before the traffic that is supposed to leave has left. The extent and duration of such congestion is worse when the network is busier and has larger RTTs (which lead to greater temporal disparity in the application of updates). Both these conditions hold

¹In some networks, fault tolerance is another reason for low utilization: the network is provisioned such that there is ample capacity even after (common) failures. However, in inter-DC WANs, traffic that needs strong protection is a small subset of the overall traffic, and existing techniques can tag and protect such traffic in the face of failures [32].

(SIGCOMM'13)

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 ... *)
```

Achieving High Utilization with Software-Driven WAN

Chi-Yao Hong (UUC) Srikant Kandula Ravi Mahajan Ming Zhang
Vijay Gill Mohan Nanduri Roger Wattenhofer (ETH)
Microsoft

Abstract—We present SWAN, a system that boosts the utilization of inter-datacenter networks by centrally controlling when and how much traffic each service sends and frequently re-configuring the network's data plane to match current traffic demand. But done simplistically, these re-configurations can also cause severe, transient congestion because different switches may apply updates at different times. We develop a novel technique that leverages a small amount of scratch capacity on links to apply updates to previously congested links smoothly, without making any assumptions about the order and timing of updates of individual switches. Further, to scale to large networks with large amounts of limited forwarding table capacity, SWAN computes a small set of entries that can best satisfy the current traffic. It updates this set without disrupting traffic by applying a small amount of scratch capacity in a distributed manner, using a modified protocol that is robust to the failure of two production networks. SWAN achieves up to 60% more traffic than the current production network.

Categories and Subject Descriptors: D.1.1 Software Construction; Networks; Networks and Design
Keywords: Inter-DC, Software-Driven, Routing

1. INTRODUCTION

The wide area network (WAN) connects the data centers (DC) in critical industries for providers of services such as Amazon, Google, and Microsoft. Most services rely on low-latency inter-DC communication. Latency user experience and on high-throughput throughput (e.g., when replicating updates). Given the need for high capacity—inter-DC traffic is a significant fraction of Internet traffic and rapidly growing [21]—unique traffic characteristics, the inter-DC WAN is a dedicated network, distinct from the WAN of consumer devices, mobile, and users [23]. It is an environment with high bandwidth and low latency, but with a high cost of ownership (up to 10% of total data center cost). However, providers are not fully leveraging this investment today. In fact, inter-DC links are a poor of capacity; the average utilization of inter-DC links is 40-60%. One of the main reasons for this is the lack of coordination among the services that use the network. In some cases, static links

in some cases, services send traffic whenever they want and however much they want. As a result, the network is through periods of peaks and troughs. Since it must be provisioned for peak usage to avoid congestion, the network is under-utilized on average. Observe that this is not true if we can control the timing of inter-DC traffic. Some inter-DC traffic is scheduled. We can ramp the system behavior if such traffic is scheduled. We can also use the network's capacity to use less capacity to carry some traffic. We can also use the network's capacity to use less capacity to carry some traffic. We can also use the network's capacity to use less capacity to carry some traffic.

We present SWAN, a system that makes it straightforward to carry significantly more traffic. It is straightforward—we can let the network's capacity to use less capacity to carry some traffic. We can also use the network's capacity to use less capacity to carry some traffic. We can also use the network's capacity to use less capacity to carry some traffic.

¹In some networks, fault tolerance is another reason for low utilization: the network is provisioned such that there is ample capacity even after (common) failures. However, in inter-DC WANs, traffic that needs strong protection is a small subset of the overall traffic, and existing techniques can tag and protect such traffic in the face of failures [22].

(SIGCOMM'13)

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

Modular Implementation

Achieving High Utilization with Software-Driven WAN

Chn-Yao Hong (USC), Srikanth Kandula, Rajul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri, Roger Waterholer (ETH), Microsoft

Abstract— We present SWAN, a system that boots the utilization of inter-datacenter networks by one hop when and how much traffic each service (and hence the network's data plane) current traffic demand. This done simplifies configurations can also cause severe, trouble because different switches may apply updates times. We develop a novel technique that in the absence of scratch capacity in links to apply precisely congestion-free updates, without any disruptions about the order and timing of updates and switches. Further, to scale to large volumes of limited forwarding table capacity, SWAN is a small set of entries that can be easily updated. It updates this set without disrupting traffic in small amount of scratch capacity in forwarding tables using a watched prototype and state tables of two production networks show that SWAN runs traffic than the current practice.

Categories and Subject Descriptors: C.2. [Communication Networks] Network Architecture

Keywords: Inter-DC WAN, software-defined

1. INTRODUCTION

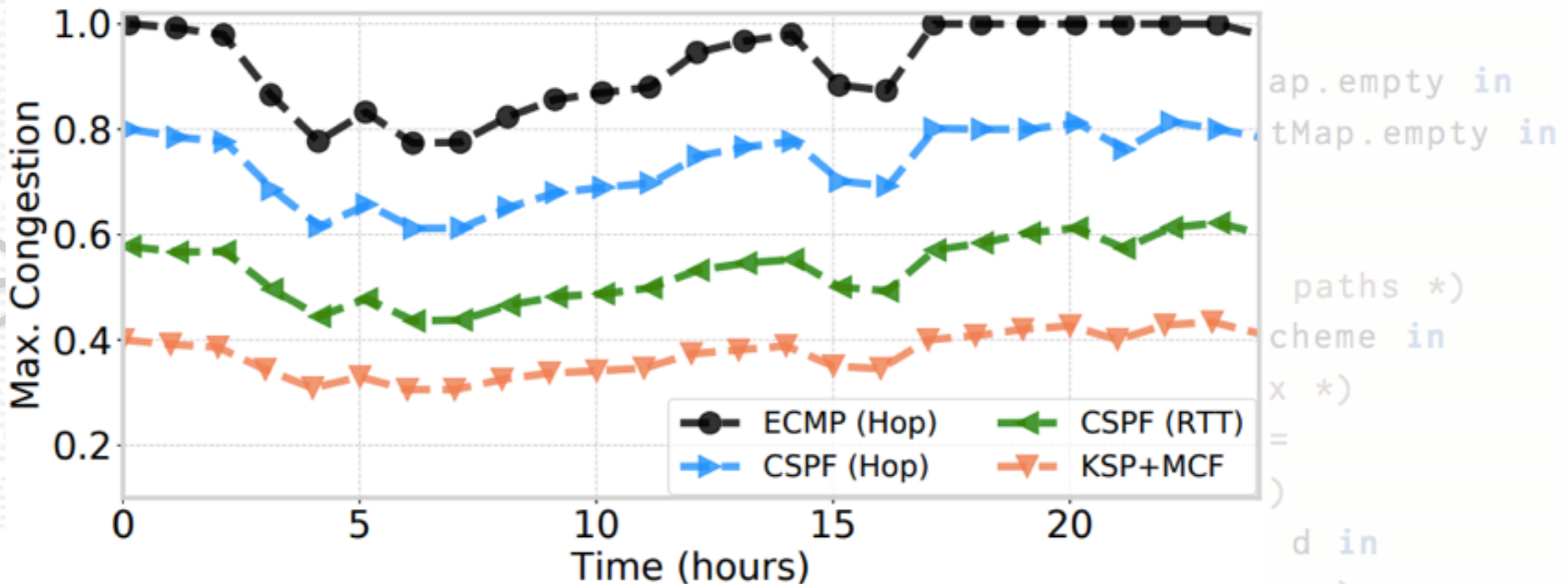
The wide area network (WAN) connecting data centers (DC) is critical infrastructure for services such as Amazon, Google, and Microsoft. Inter-DC traffic is a significant portion of WAN traffic and rapidly growing [26]. Inter-DC traffic is characterized by high capacity, high latency, and high bandwidth. The inter-DC WAN is often a network, distinct from the WAN that connects end users [25]. It is an expensive resource that annual cost of 20% of millions of dollars, 20% of OPEX to 5% of capacity over long its. However, providers are unable to fully use network today. Inter-DC WANs have under capacity: the average utilization of links is 10-15%. One culprit is the lack of coordinated services that use the network. Sharing comes

(* YATES modules *)

module KSP : Algorithm

module SemiMCF : Algorithm

(* Compute base set of k-shortest paths *)



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

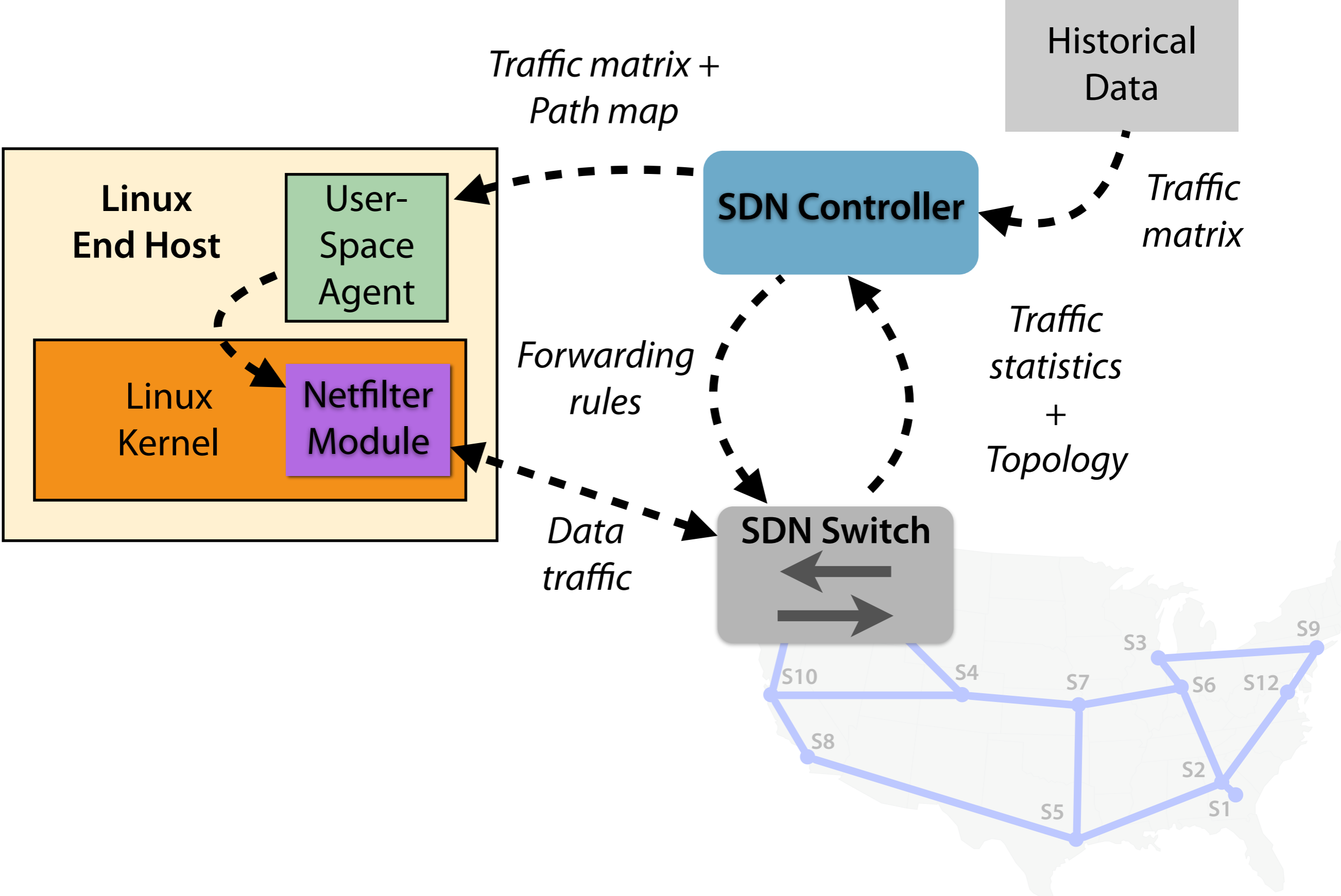
Simulator

- Scalable
- Steady-state
- Macroscopic properties

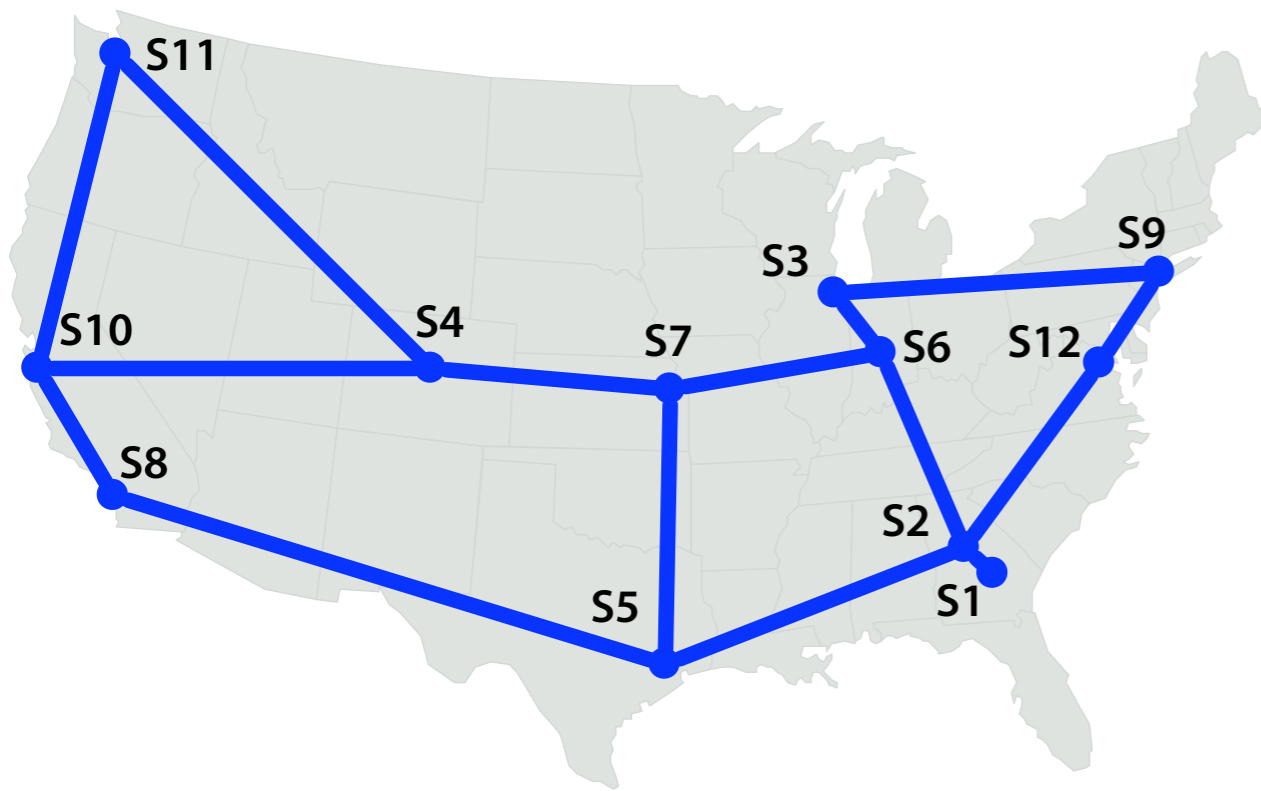
SDN Backend

- Implementations: static paths and source routing

SDN Backend



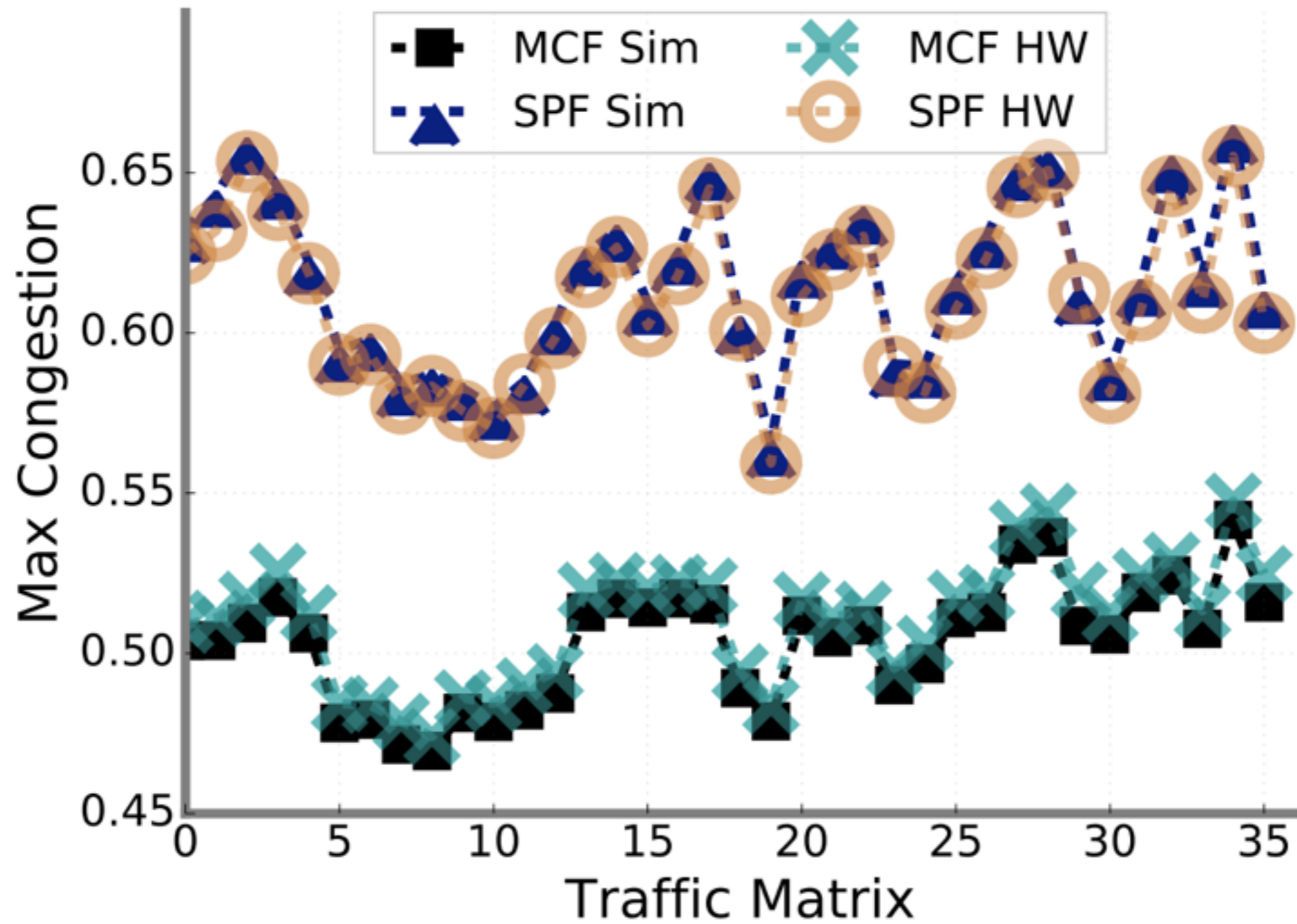
Hardware Testbed



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>