

IEEE/ACM CCGRID 2024

The 24th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing

A Distributed, Asynchronous Algorithm for Large-Scale Internet Network Topology Analysis

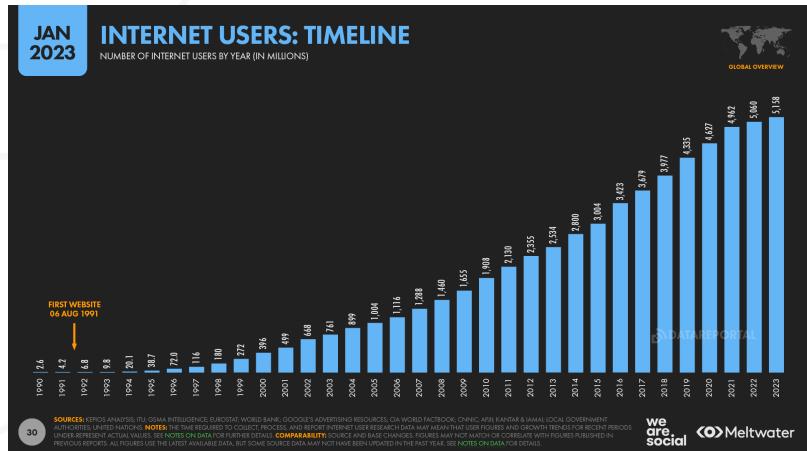
Youssef Elmougy*, Akihiro Hayashi, and Vivek Sarkar

Habanero Extreme Scale Software Research Lab
Georgia Institute of Technology

* Corresponding Author and Presenting Author: yelmougy3@gatech.edu



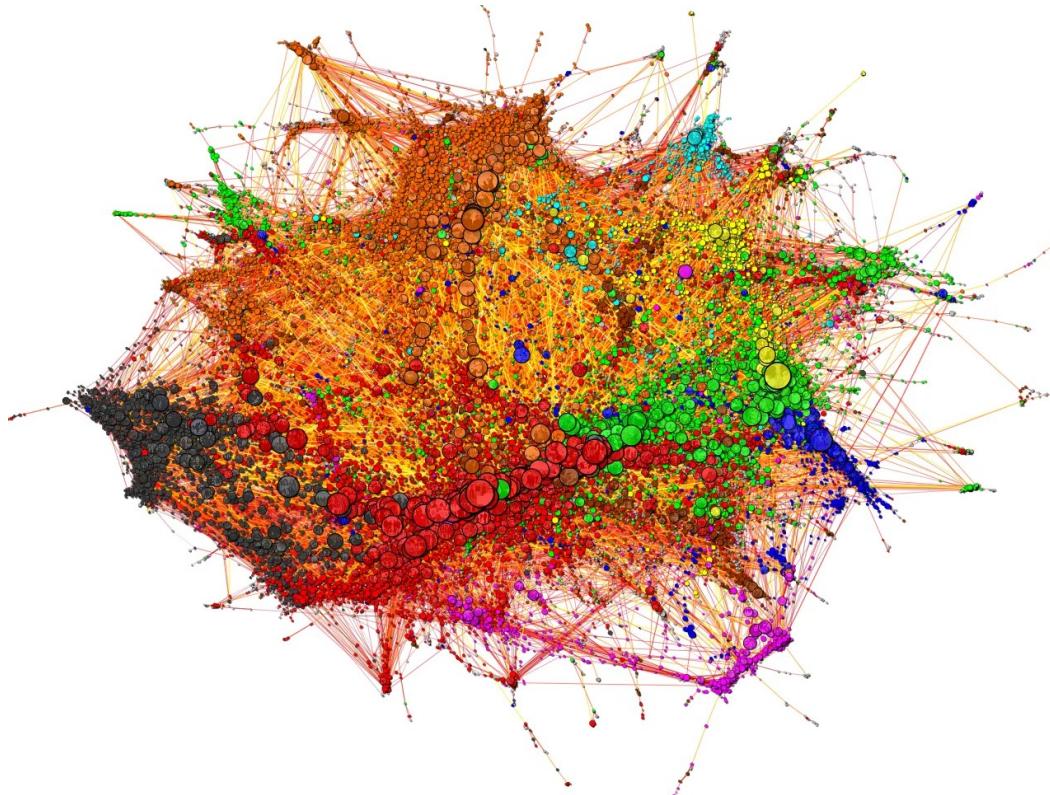
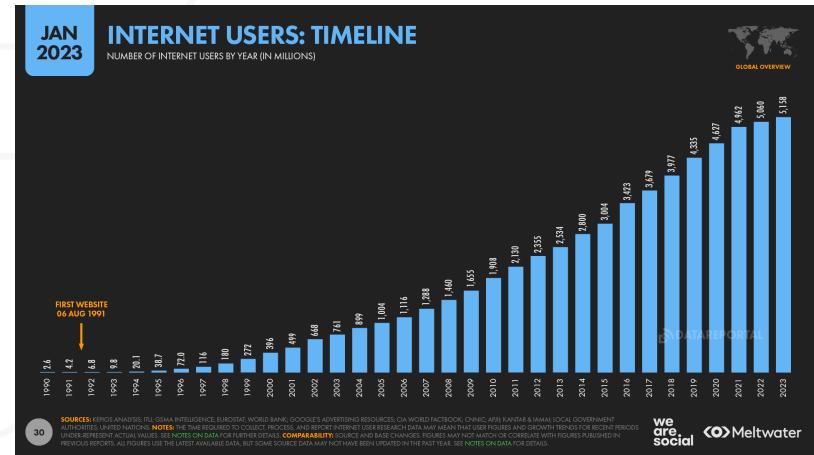
The Growth of the Internet and the increasing complexity of its global network topology



Picture borrowed from:

<https://datareportal.com/reports/digital-2023-global-overview-report>

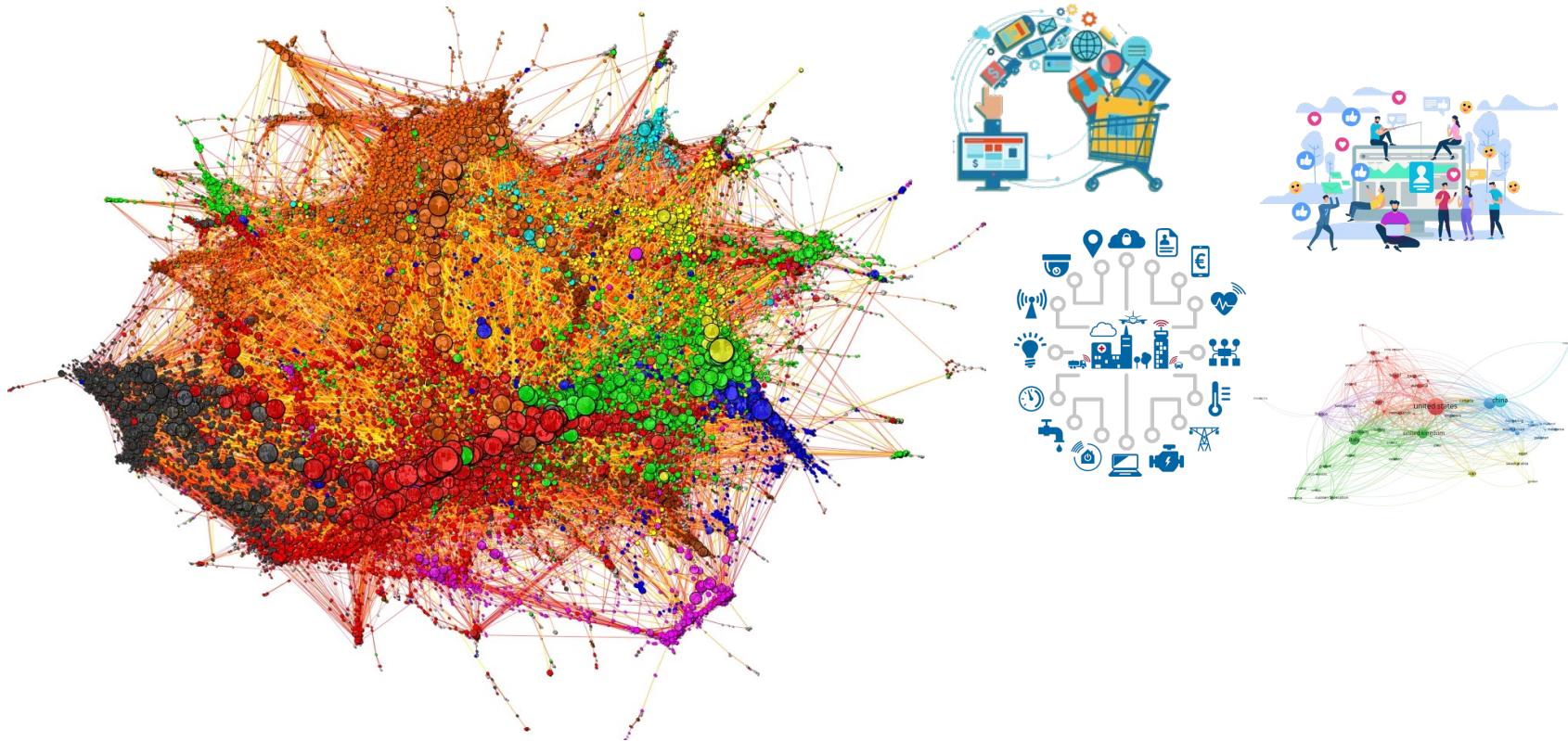
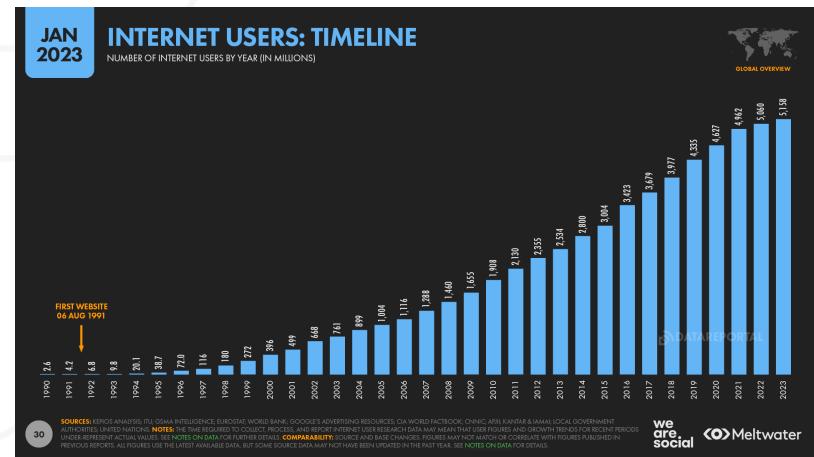
The Growth of the Internet and the increasing complexity of its global network topology



Picture borrowed from:

<https://datareportal.com/reports/digital-2023-global-overview-report>

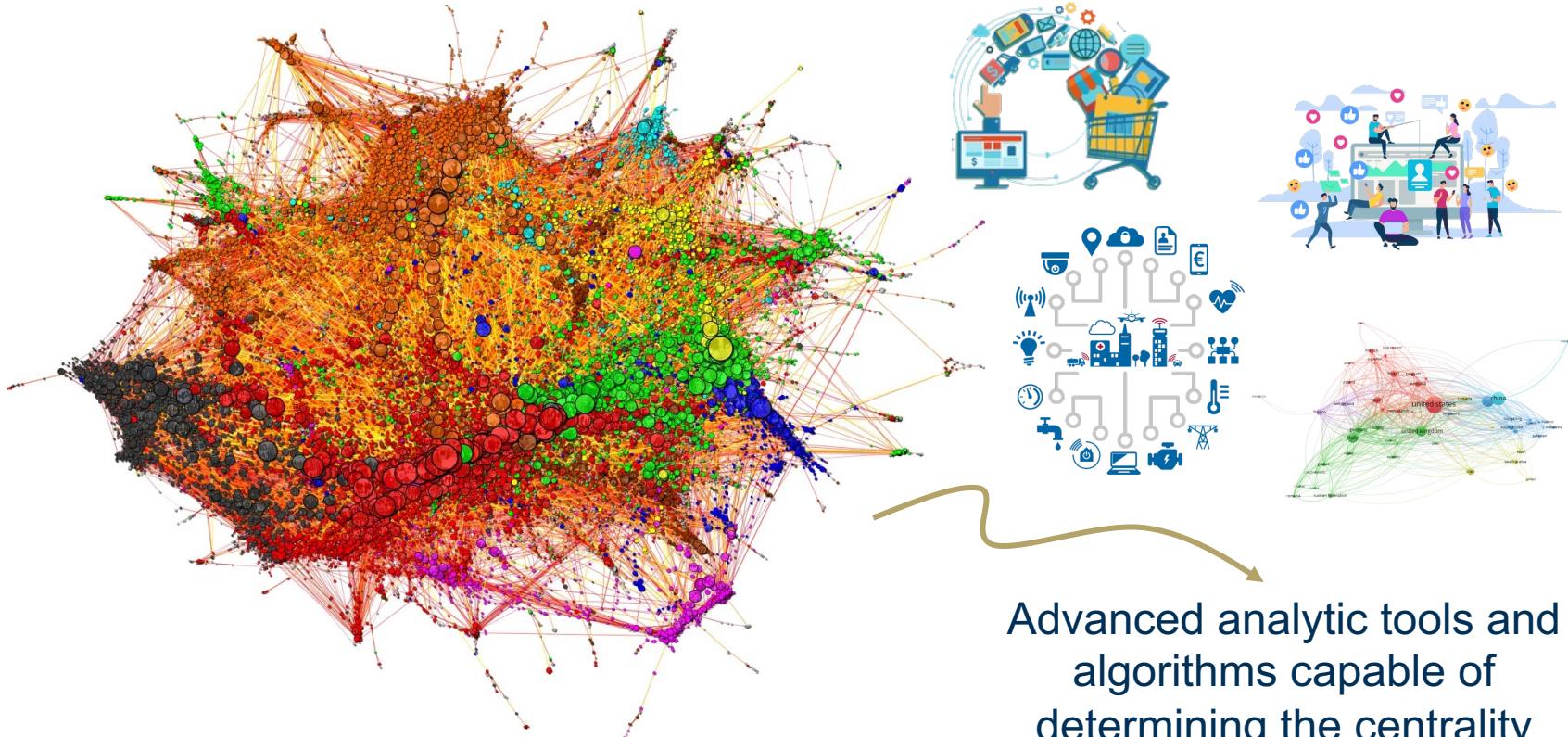
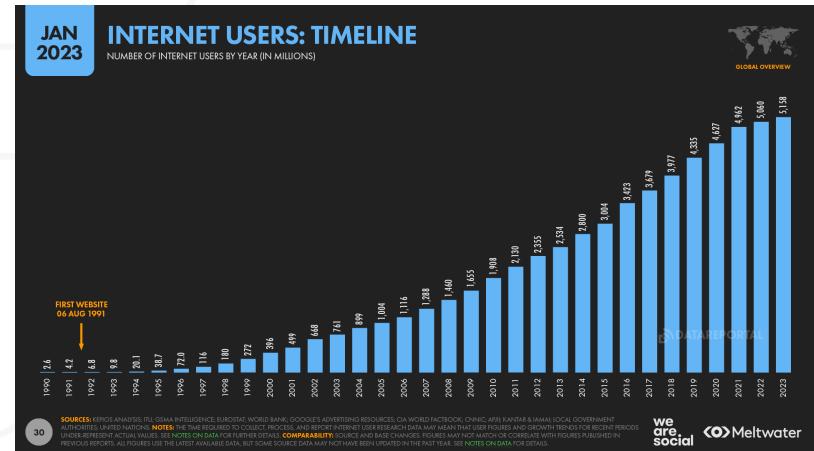
The Growth of the Internet and the increasing complexity of its global network topology



Picture borrowed from:

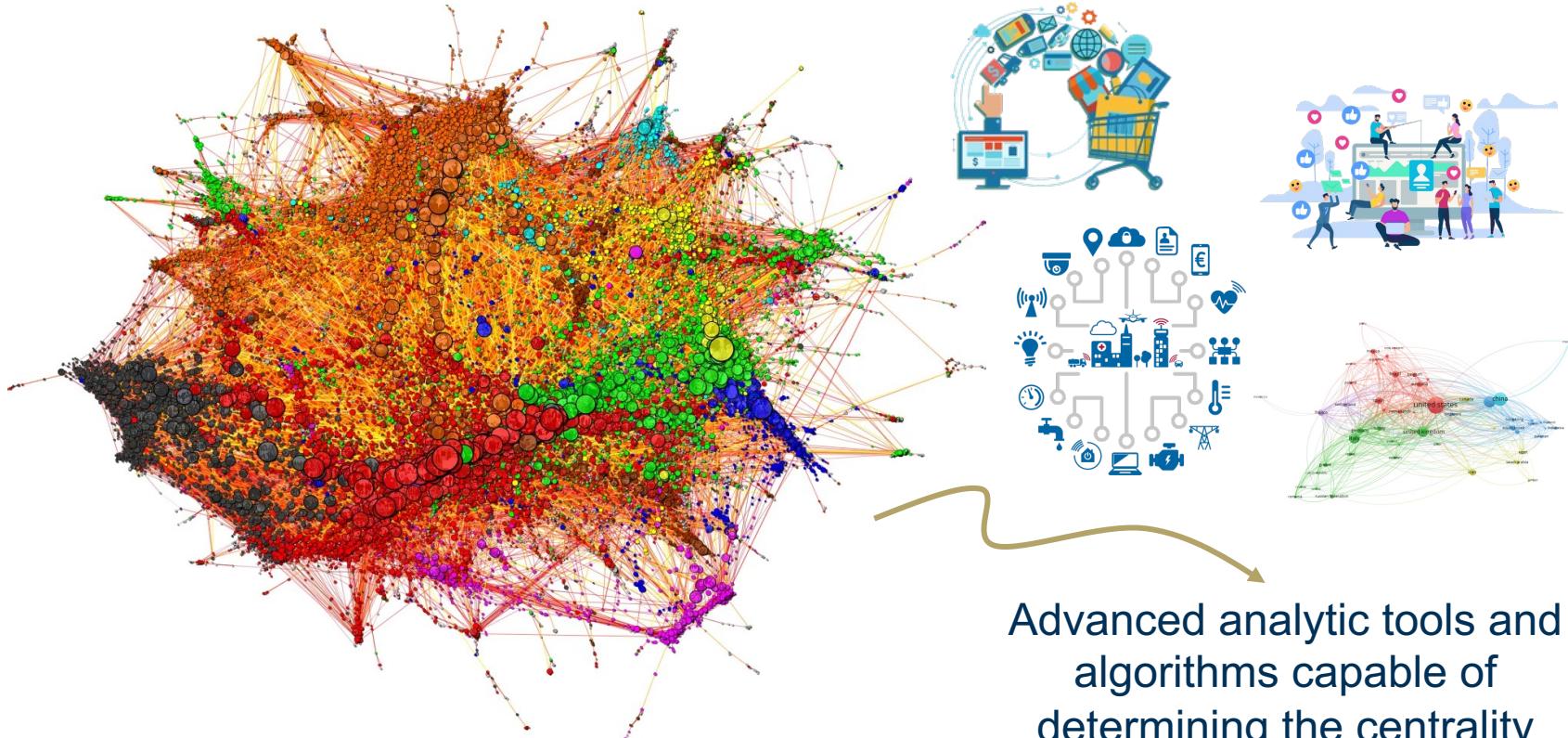
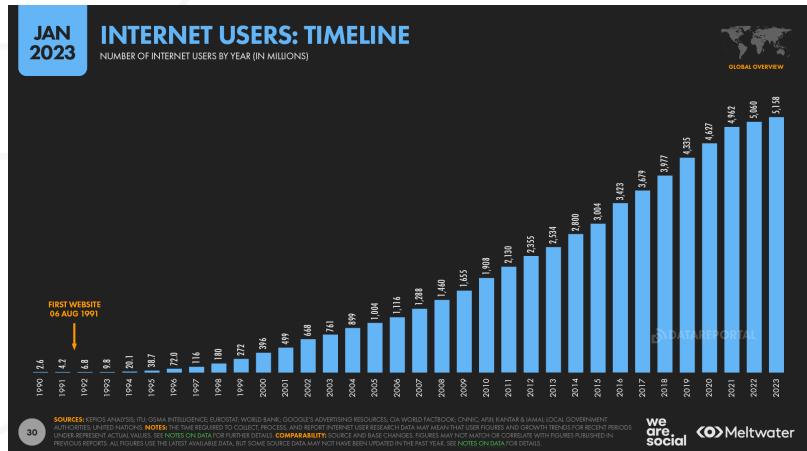
<https://datareportal.com/reports/digital-2023-global-overview-report>

The Growth of the Internet and the increasing complexity of its global network topology



Advanced analytic tools and algorithms capable of determining the centrality and importance of network topology have become increasingly important

The Growth of the Internet and the increasing complexity of its global network topology



Picture borrowed from:
<https://datareportal.com/reports/digital-2023-global-overview-report>

Effectively understanding and analyzing the critical nodes and components within these networks is paramount to:

Network Management

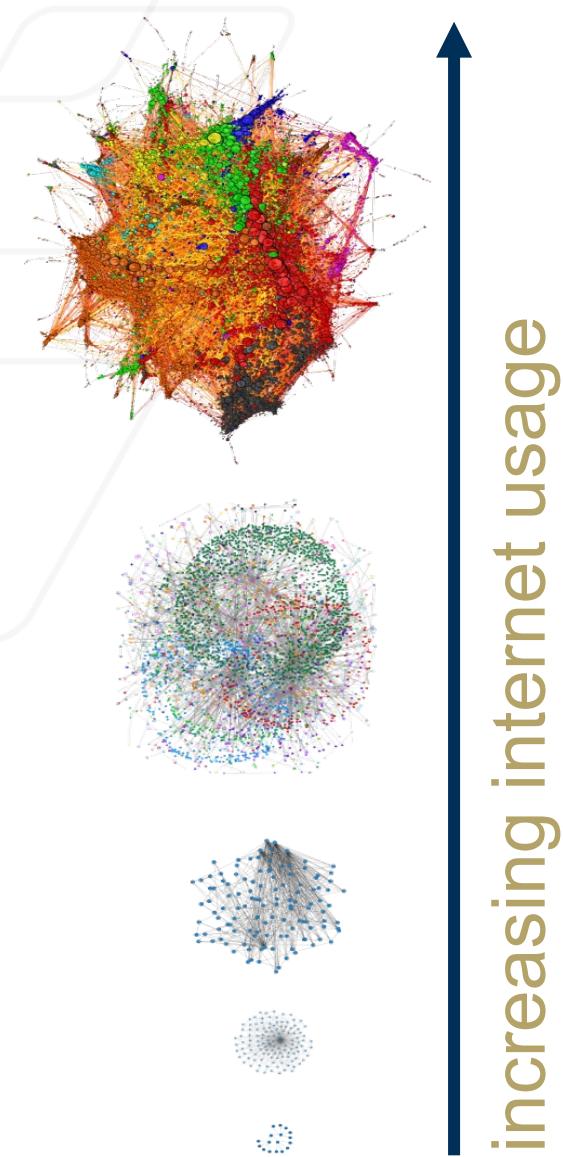
Security

Fault Tolerance and Resilience

Network Stability and Performance

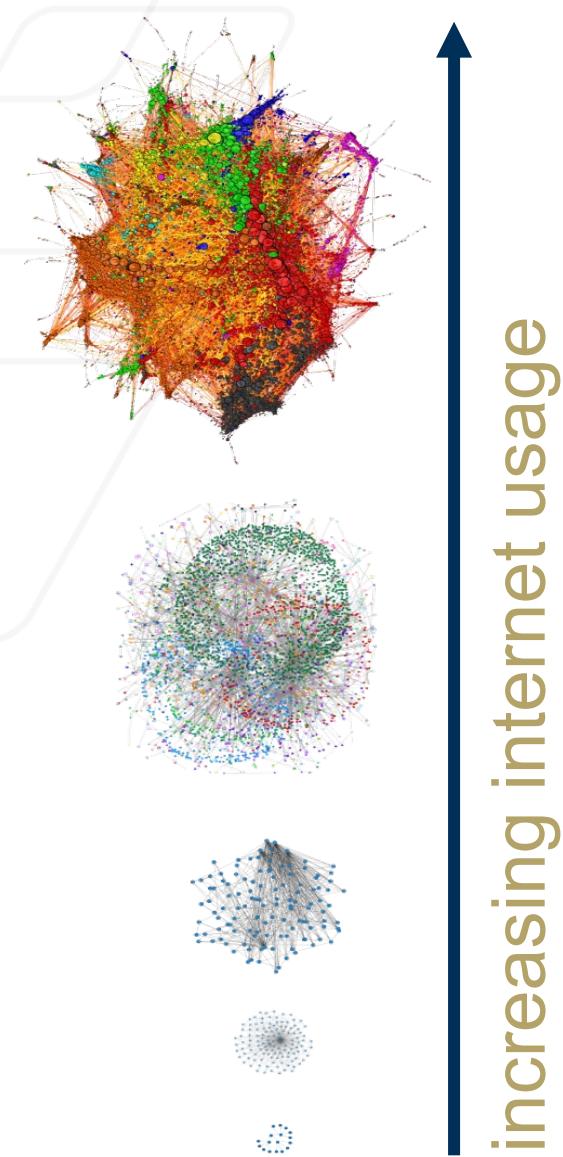
Advanced analytic tools and algorithms capable of determining the centrality and importance of network topology have become increasingly important

The Scalability Problem



- ❑ Efficient and accurate analysis of network topology becomes increasingly critical as internet usage rapidly increases

The Scalability Problem

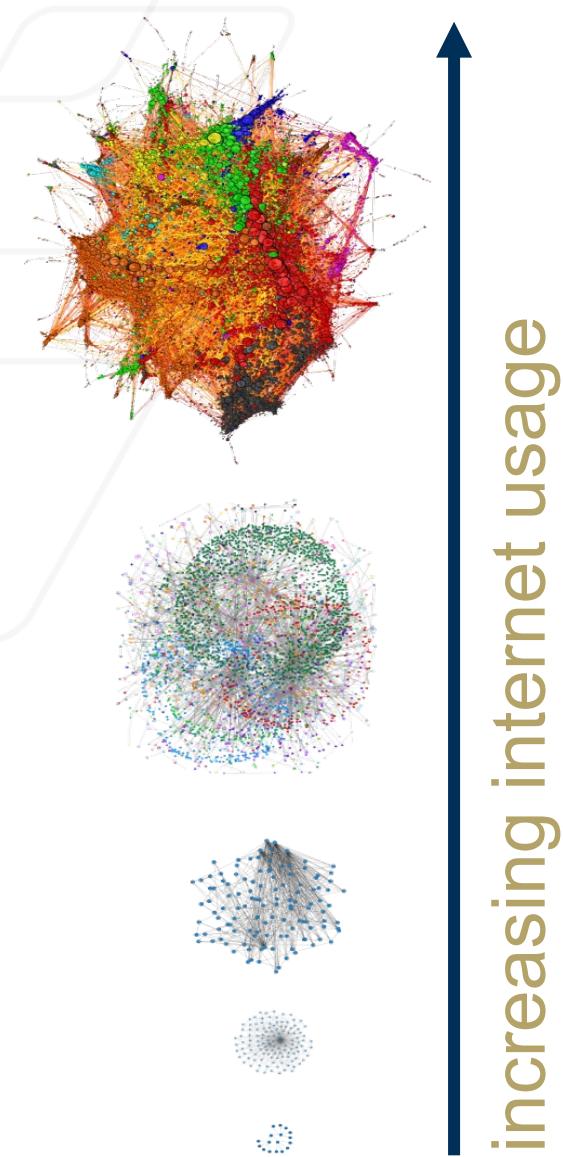


Individual User



- ❑ Efficient and accurate analysis of network topology becomes increasingly critical as internet usage rapidly increases

The Scalability Problem



Individual User



Massive Corporations

- ❑ Efficient and accurate analysis of network topology becomes increasingly critical as internet usage rapidly increases

The Scalability Problem



Individual User



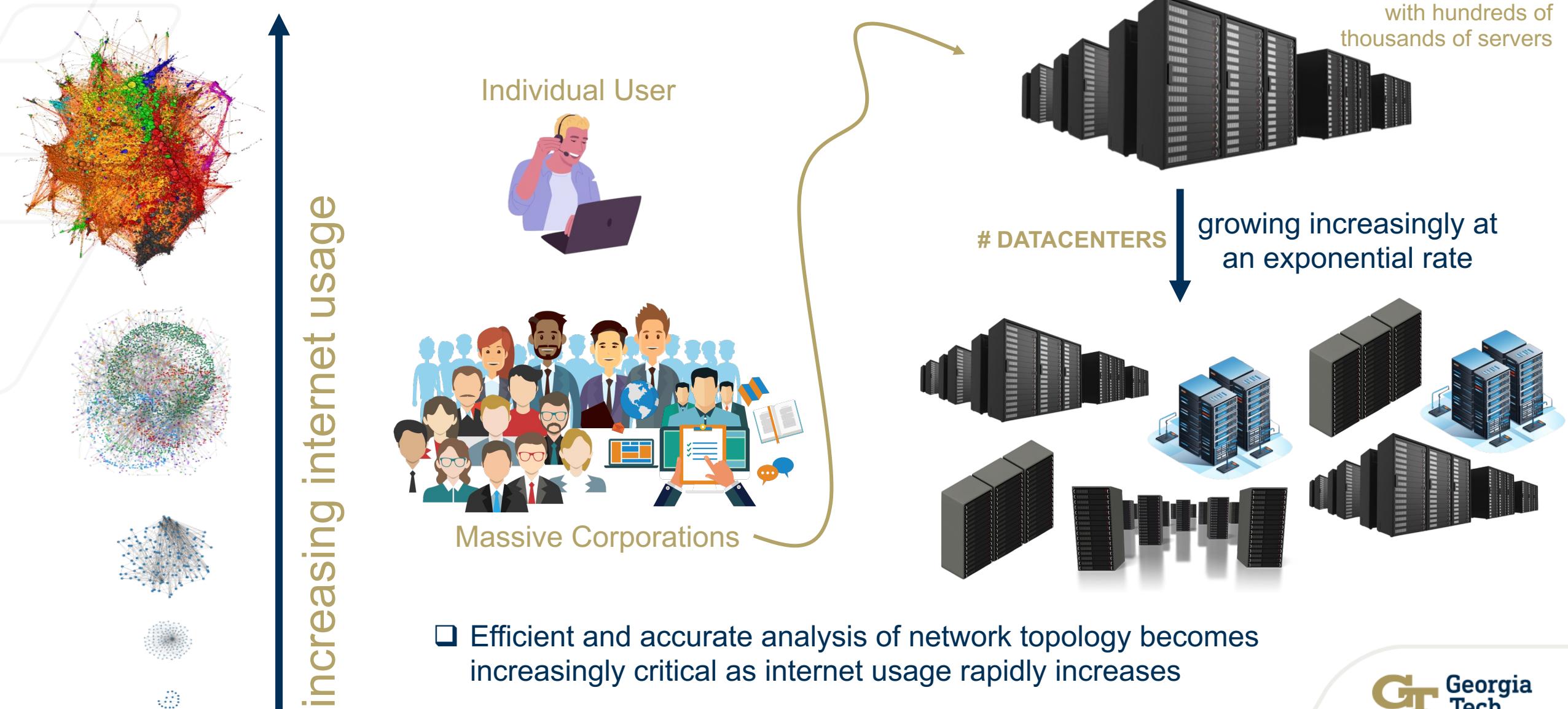
Massive Corporations



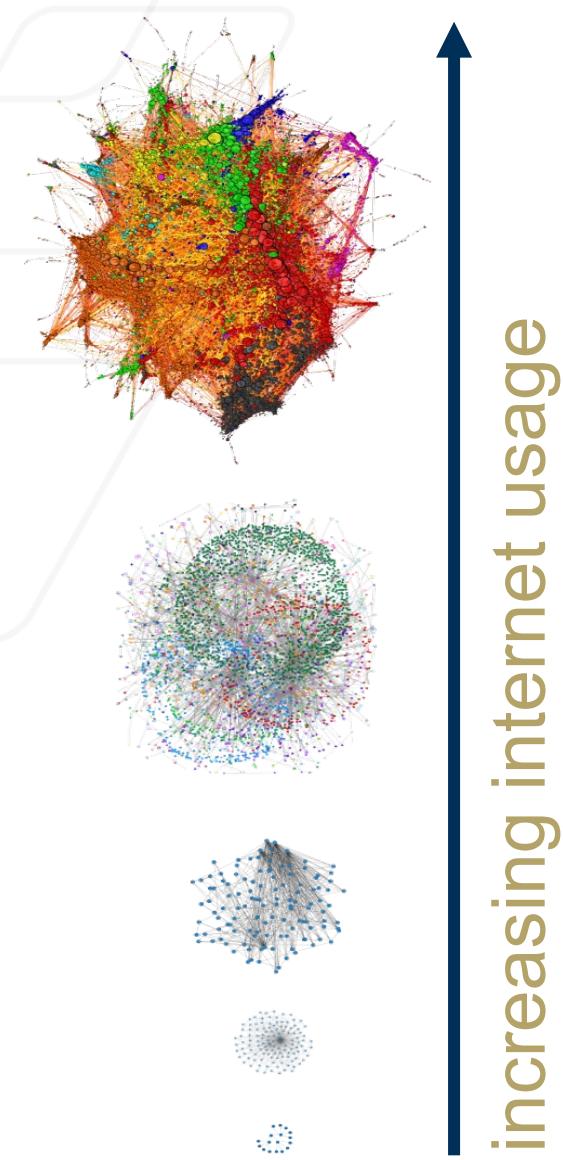
Modern Datacenter
with hundreds of
thousands of servers

- ❑ Efficient and accurate analysis of network topology becomes increasingly critical as internet usage rapidly increases

The Scalability Problem



The Scalability Problem



The Scalability Problem



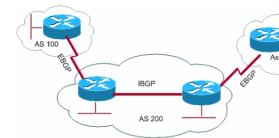
Routers

Bridges and
Switches
127.0.0.1
172.16.0.9
192.0.0.7
IPs



ASes

BILLIONS OF DEVICES



Repeaters
and Hubs



NICs

The Scalability Problem



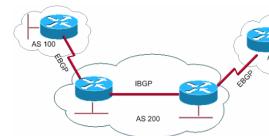
Routers

Bridges and
Switches

127.0.0.1
172.16.0.9
192.0.0.7
IPs



ASes



BILLIONS OF DEVICES

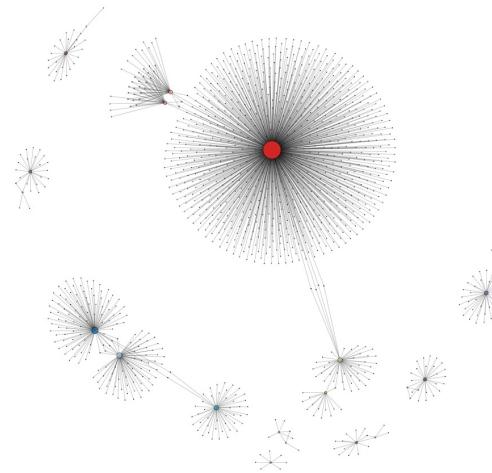


Repeaters
and Hubs

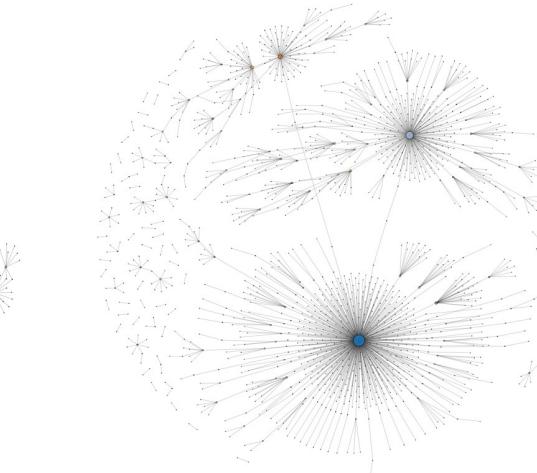


NICs

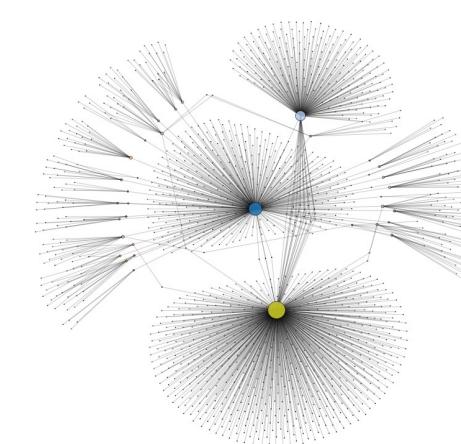
LARGER MAGNITUDE OF CONTINUOUS INTERACTIONS



(a) Exchanges across Routers



(b) Exchanges across IPs

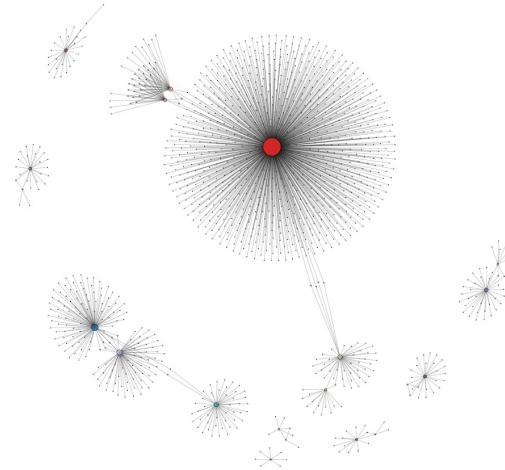


(c) Exchanges across ASes

The Scalability Problem



LARGER MAGNITUDE OF CONTINUOUS INTERACTIONS

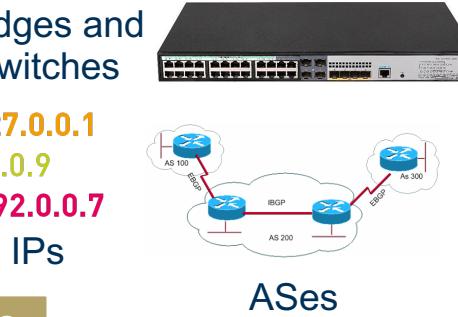


(a) Exchanges across Routers



Routers

Bridges and Switches
127.0.0.1
172.16.0.9
192.0.0.7
IPs



ASes



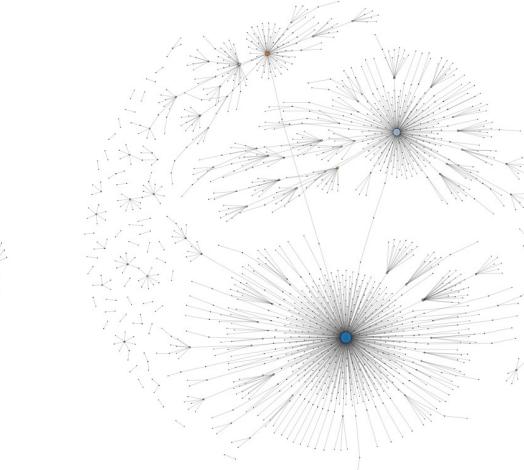
BILLIONS OF DEVICES



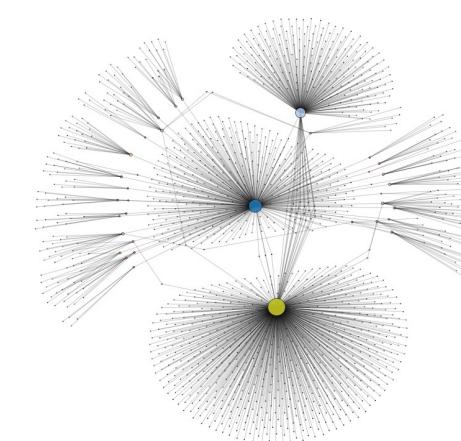
Repeaters and Hubs



NICs



(b) Exchanges across IPs



(c) Exchanges across ASes

- There is a need for parallel computing resources to meet the computational and memory requirements

The Scalability Problem



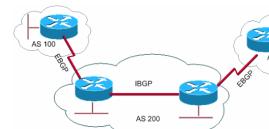
Routers

Bridges and
Switches

127.0.0.1
172.16.0.9
192.0.0.7
IPs



ASes



BILLIONS OF DEVICES

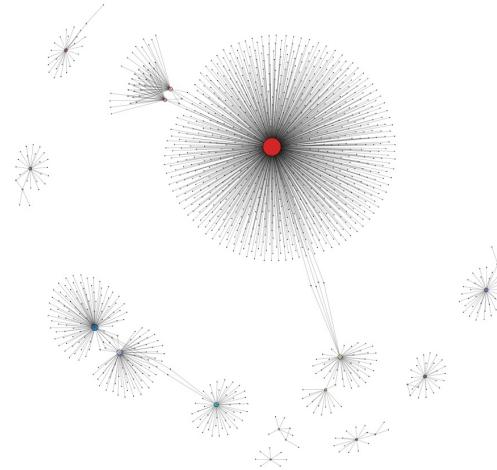


Repeaters
and Hubs

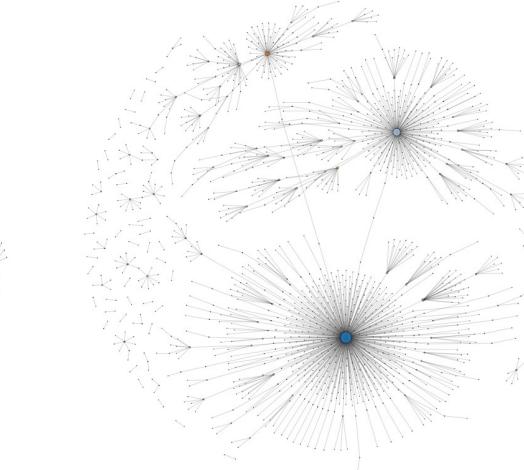


NICs

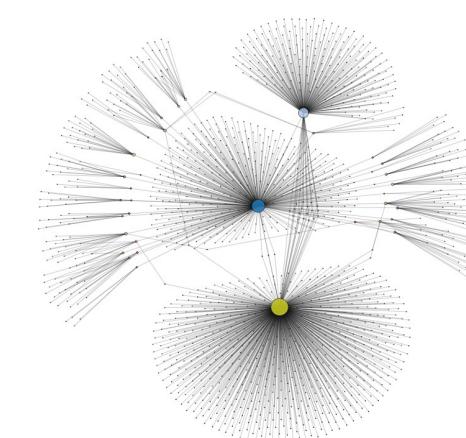
LARGER MAGNITUDE OF CONTINUOUS INTERACTIONS



(a) Exchanges across Routers



(b) Exchanges across IPs



(c) Exchanges across ASes

- There is a need for parallel computing resources to meet the computational and memory requirements

It is important to develop scalable and lightweight systems to efficiently process large-scale network topologies!

The Necessity for Large-Scale Analysis

Criticality of analyzing components within the network topology to identify sources of **vulnerabilities, inefficiencies, and possible breakdowns**, with potential impact on individual and organizational users depending on the internet for day-to-day operations

The Necessity for Large-Scale Analysis

Criticality of analyzing components within the network topology to identify sources of **vulnerabilities, inefficiencies, and possible breakdowns**, with potential impact on individual and organizational users depending on the internet for day-to-day operations

CONSIDER A GLOBAL CLOUD SERVICE PROVIDER

The Necessity for Large-Scale Analysis

Criticality of analyzing components within the network topology to identify sources of **vulnerabilities, inefficiencies, and possible breakdowns**, with potential impact on individual and organizational users depending on the internet for day-to-day operations

CONSIDER A GLOBAL CLOUD SERVICE PROVIDER

e.g.



Microsoft
Azure



1M servers



200 datacenters

The Necessity for Large-Scale Analysis

Criticality of analyzing components within the network topology to identify sources of **vulnerabilities, inefficiencies, and possible breakdowns**, with potential impact on individual and organizational users depending on the internet for day-to-day operations

CONSIDER A GLOBAL CLOUD SERVICE PROVIDER

e.g.



Microsoft
Azure



1M servers



200 datacenters

which manages **diverse arrays of clients**
with unique agreements
and connectivity requirements

The Necessity for Large-Scale Analysis

Criticality of analyzing components within the network topology to identify sources of **vulnerabilities, inefficiencies, and possible breakdowns**, with potential impact on individual and organizational users depending on the internet for day-to-day operations

CONSIDER A GLOBAL CLOUD SERVICE PROVIDER

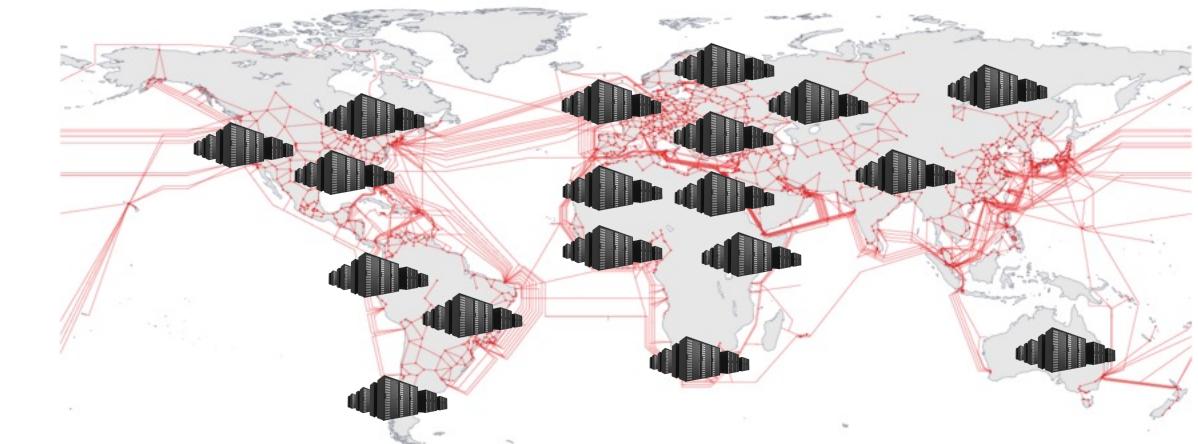
e.g. Microsoft Azure



1M servers 200 datacenters



which manages diverse arrays of clients
with unique agreements
and connectivity requirements



operates across several datacenters in multiple continents
relying on networks of routers, switches, and computing devices

The Necessity for Large-Scale Analysis

Criticality of analyzing components within the network topology to identify sources of **vulnerabilities, inefficiencies, and possible breakdowns**, with potential impact on individual and organizational users depending on the internet for day-to-day operations

CONSIDER A GLOBAL CLOUD SERVICE PROVIDER

e.g. Microsoft Azure



1M servers 200 datacenters



which manages diverse arrays of clients
with unique agreements
and connectivity requirements



operates across several datacenters in multiple continents
relying on networks of routers, switches, and computing devices

DISRUPTIONS 

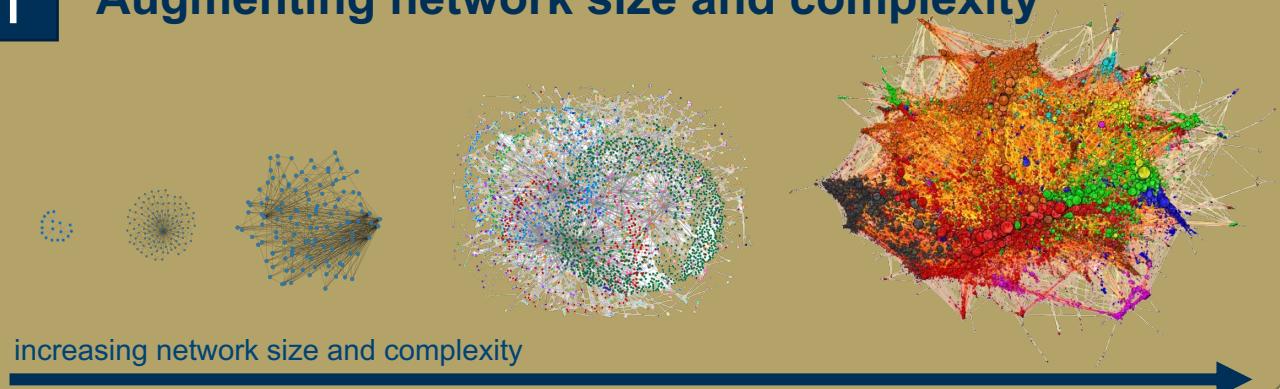
as a result of **performance bottlenecks, component failures, or security breaches**
can have dire consequences



The 4 Challenges in Reliable Internet Provision

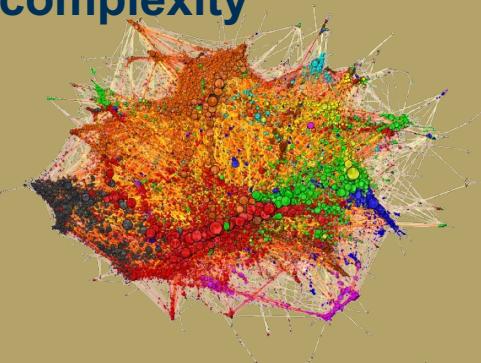
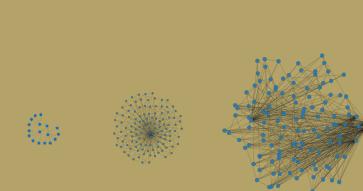
The 4 Challenges in Reliable Internet Provision

1 Augmenting network size and complexity



The 4 Challenges in Reliable Internet Provision

1 Augmenting network size and complexity



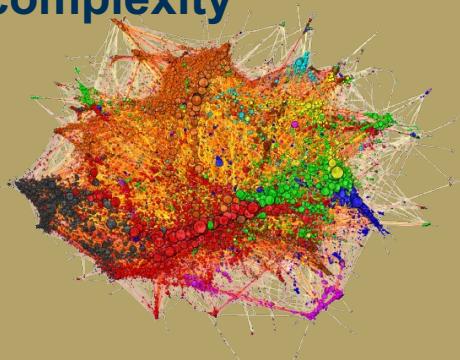
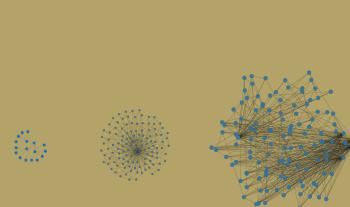
increasing network size and complexity

2 Fault tolerance and security



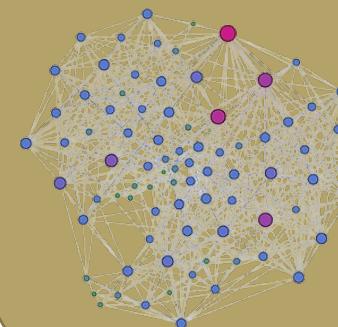
The 4 Challenges in Reliable Internet Provision

1 Augmenting network size and complexity



increasing network size and complexity

3 Real-time decision making

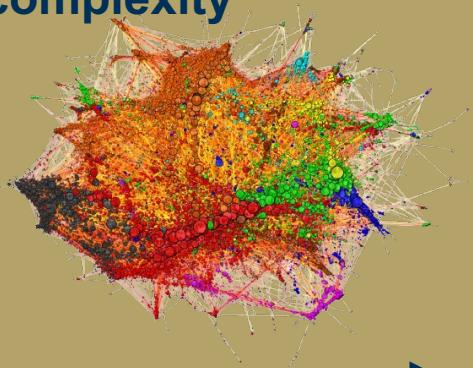
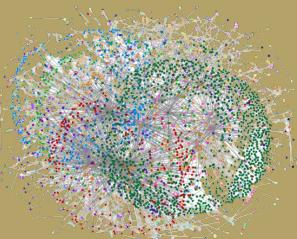
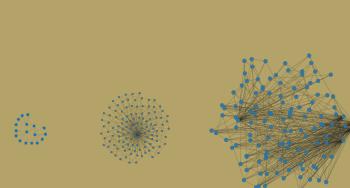


2 Fault tolerance and security



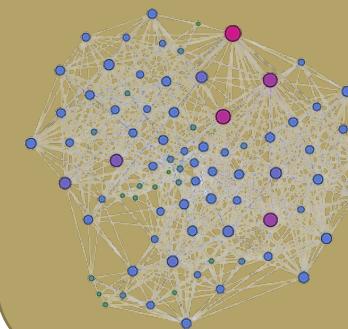
The 4 Challenges in Reliable Internet Provision

1 Augmenting network size and complexity

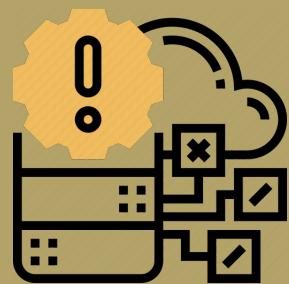


increasing network size and complexity

3 Real-time decision making



2 Fault tolerance and security

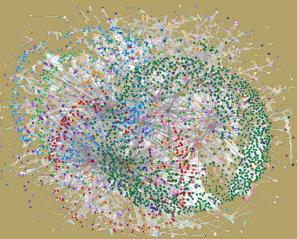
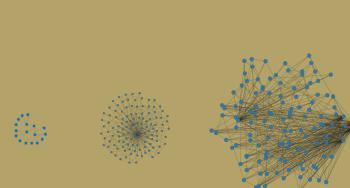


4 Energy considerations



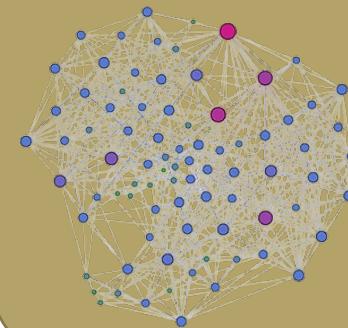
The 4 Challenges in Reliable Internet Provision

1 Augmenting network size and complexity



increasing network size and complexity

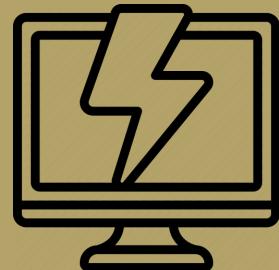
3 Real-time decision making



2 Fault tolerance and security



4 Energy considerations



This paper studies the scalability of our novel algorithm to overcome the inherent challenges of internet provision by performing large-scale network topology analysis!

Current Approaches are *NOT Efficient* for Scalable and Distributed Processing

Current approaches [2],[5],[6],[7],[8],[9] ...

Current Approaches are *NOT Efficient* for Scalable and Distributed Processing

Current approaches [2],[5],[6],[7],[8],[9] ...

are designed and optimized for **smaller scale networks and graphs**

Current Approaches are *NOT Efficient* for Scalable and Distributed Processing

Current approaches [2],[5],[6],[7],[8],[9] ...

are designed and optimized for **smaller scale networks and graphs**

are limited in scalability due to their application of **sequential or inefficient algorithms**

Current Approaches are *NOT Efficient* for Scalable and Distributed Processing

Current approaches [2],[5],[6],[7],[8],[9] ...

are designed and optimized for **smaller scale networks and graphs**

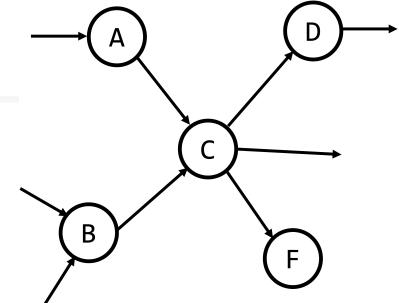
are limited in scalability due to their application of **sequential or inefficient algorithms**

CHALLENGE

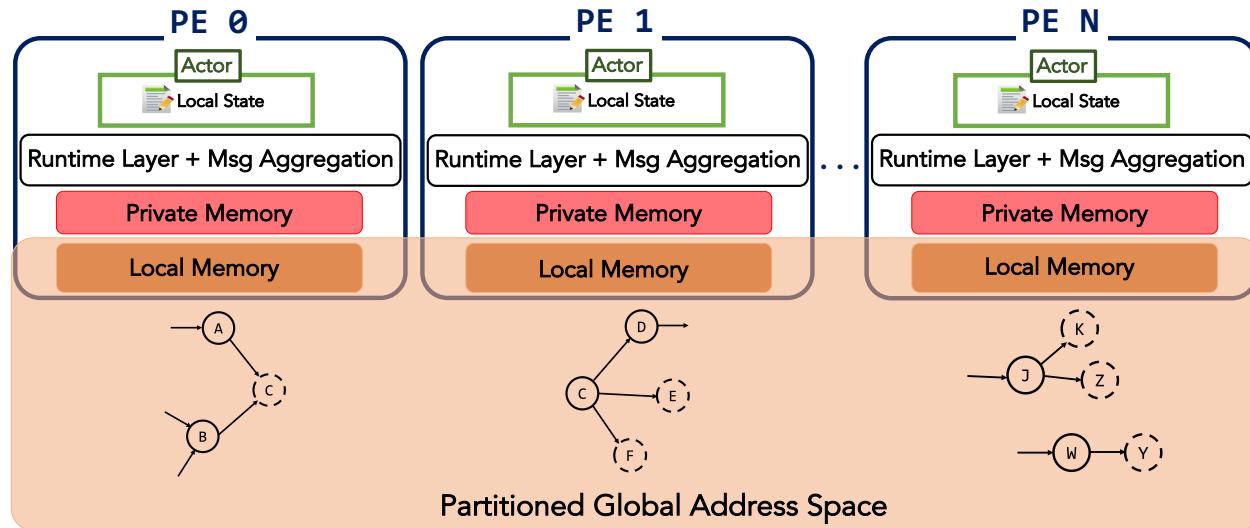
Makes it difficult to process the expansive and intricate dynamic web of interconnected devices constituting the modern internet

Backend Actor-based Scalable Architecture

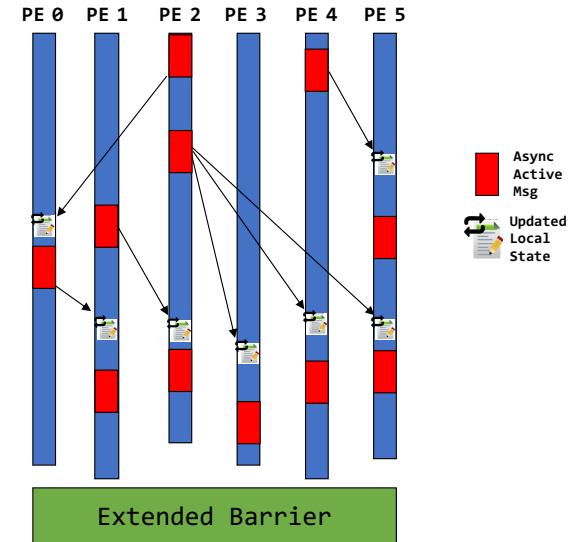
Sample Graph



Distributed Actor Runtime



Execution Model



- Presents a lightweight, asynchronous computation model
- Utilizes fine-grained asynchronous actor messages to express point-to-point remote operations
- Treats actors as primitives of computation, where actors are inherently isolated and share no mutable state
- Actors process messages sequentially within its mailbox, thereby avoiding data races and synchronization

NOTE: “Actor” and “Selector” will be used interchangeably

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

We leverage the inherent formations of triangles to explore the structural importance of individual nodes and the intricate relationships among interconnected entities

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

We leverage the inherent formations of triangles to explore the structural importance of individual nodes and the intricate relationships among interconnected entities

Triangle Centrality:

finds the important (central) nodes within a graph based on the concentration of triangles surrounding each node

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

We leverage the inherent formations of triangles to explore the structural importance of individual nodes and the intricate relationships among interconnected entities

Triangle Centrality:

finds the important (central) nodes within a graph based on the concentration of triangles surrounding each node

Important nodes are at the center of many triangles, thus, being present in many triangles or none

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

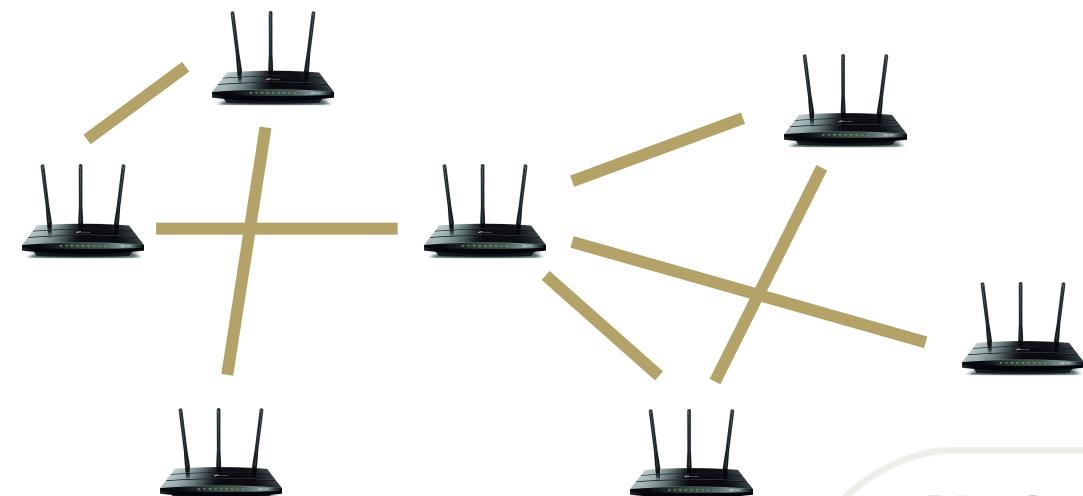
We leverage the inherent formations of triangles to explore the structural importance of individual nodes and the intricate relationships among interconnected entities

Triangle Centrality:

finds the important (central) nodes within a graph based on the concentration of triangles surrounding each node

Important nodes are at the center of many triangles, thus, being present in many triangles or none

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

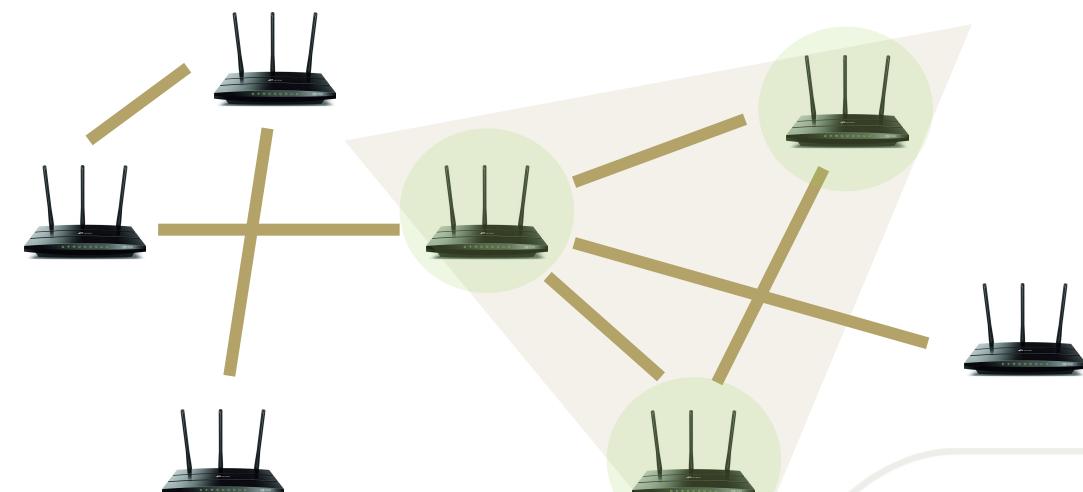
We leverage the inherent formations of triangles to explore the structural importance of individual nodes and the intricate relationships among interconnected entities

Triangle Centrality:

finds the important (central) nodes within a graph based on the concentration of triangles surrounding each node

Important nodes are at the center of many triangles, thus, being present in many triangles or none

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

We leverage the inherent formations of triangles to explore the structural importance of individual nodes and the intricate relationships among interconnected entities

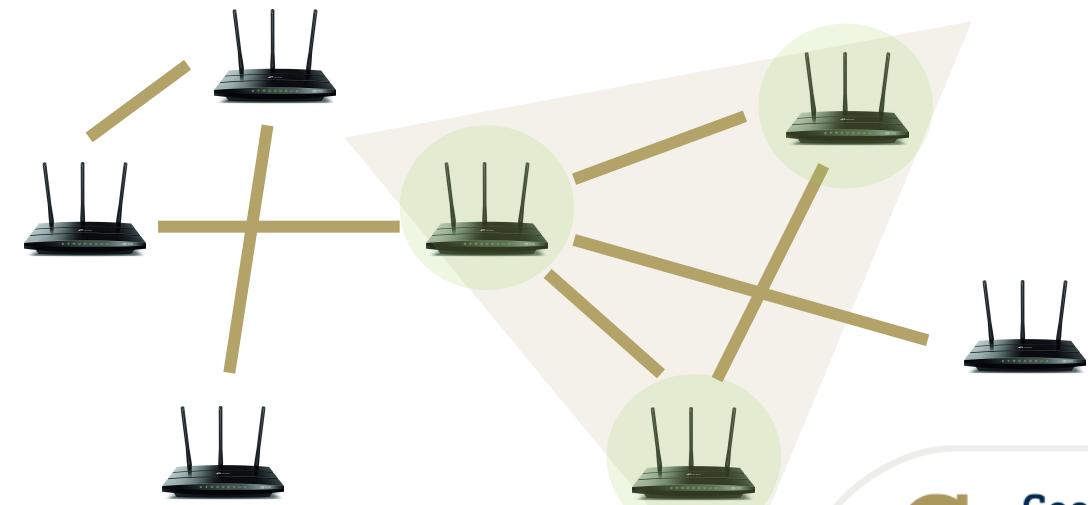
Triangle Centrality:

finds the important (central) nodes within a graph based on the concentration of triangles surrounding each node

Important nodes are at the center of many triangles, thus, being present in many triangles or none

That concentration of triangles is a rep. of augmented network density, permitting the flow and spread of the influence of data more rapidly through the connections

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



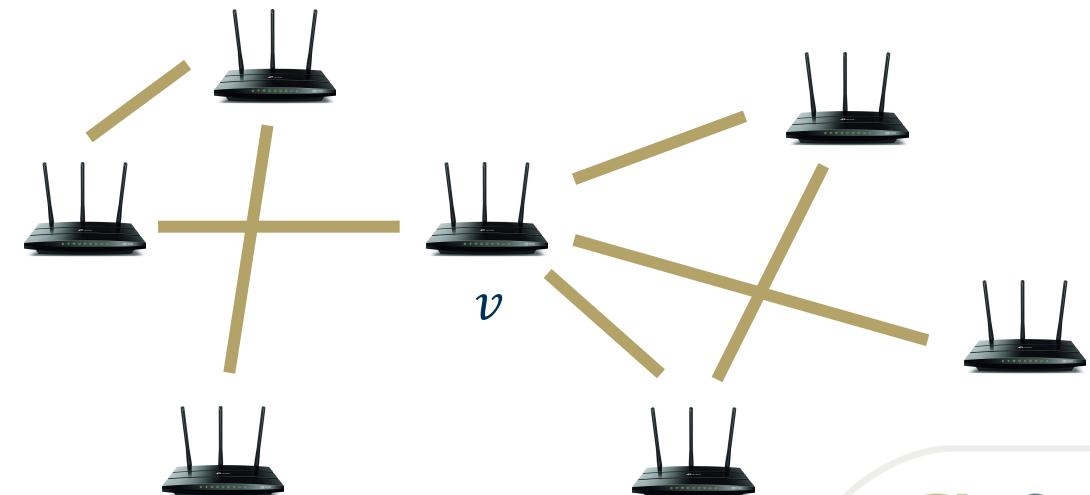
Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

Given an undirected network of entities, a graph $G = (V, E)$ with **$|V|$ vertices** **$|E|$ edges**

where

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

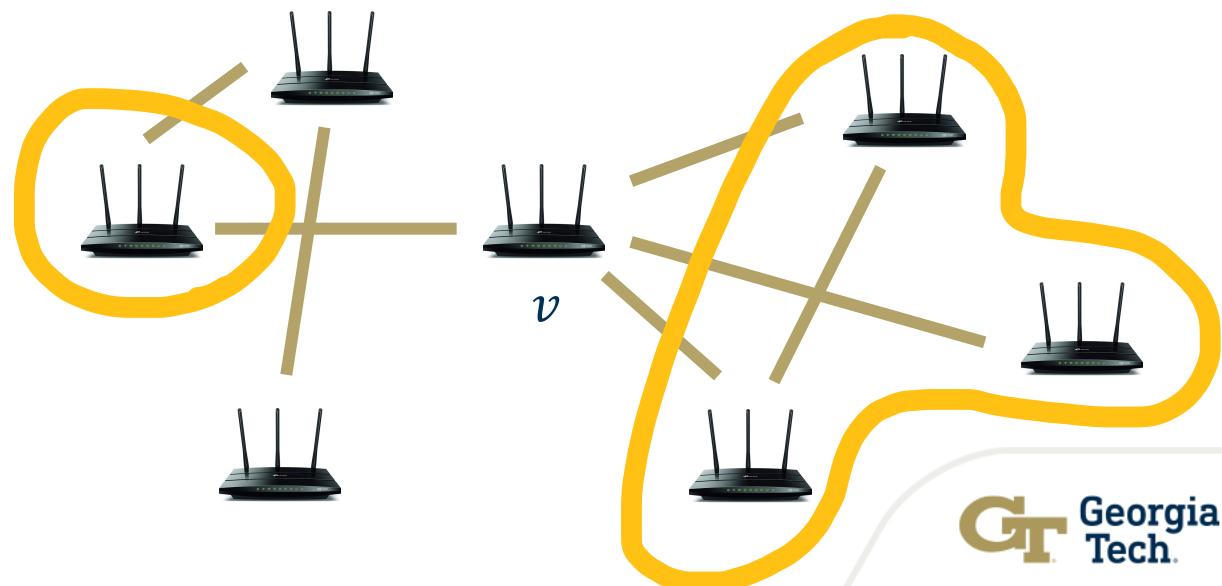
Tailored for internet network topology analysis at a massive scale

Given an undirected network of entities, a graph $G = (V, E)$ with **$|V|$ vertices** **$|E|$ edges**

where

$N(v)$ is the set of neighbors of v

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

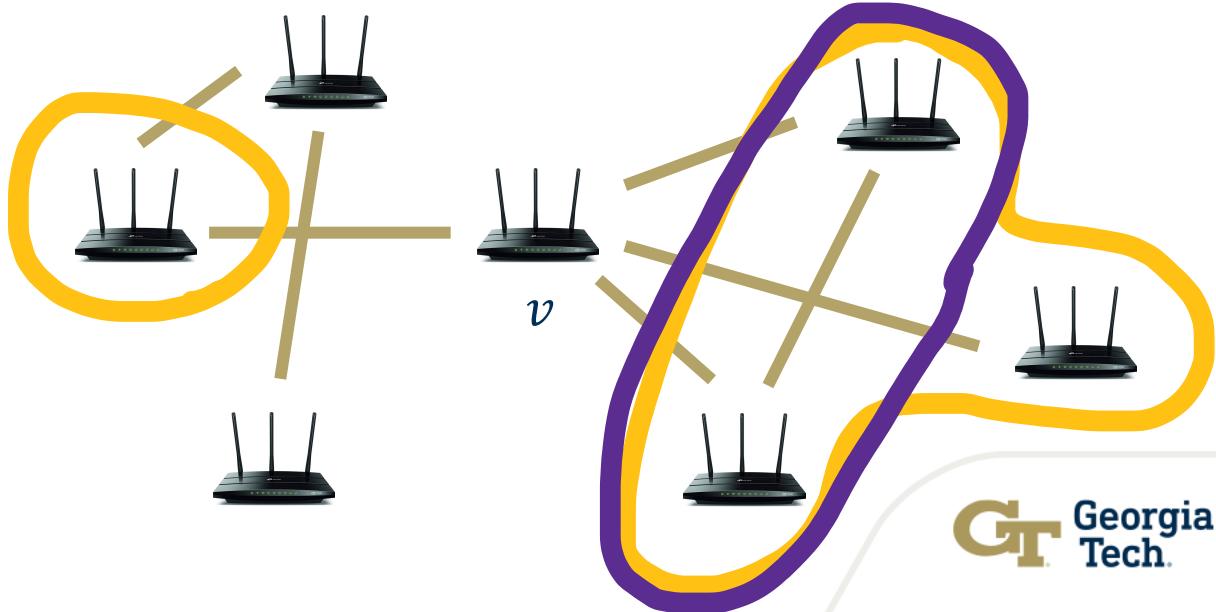
Given an undirected network of entities, a graph $G = (V, E)$ with **$|V|$ vertices** **$|E|$ edges**

where

$N(v)$ is the set of neighbors of v

$N_\Delta(v)$ is the set of neighbors in triangles with v

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

Given an undirected network of entities, a graph $G = (V, E)$ with **$|V|$ vertices** **$|E|$ edges**

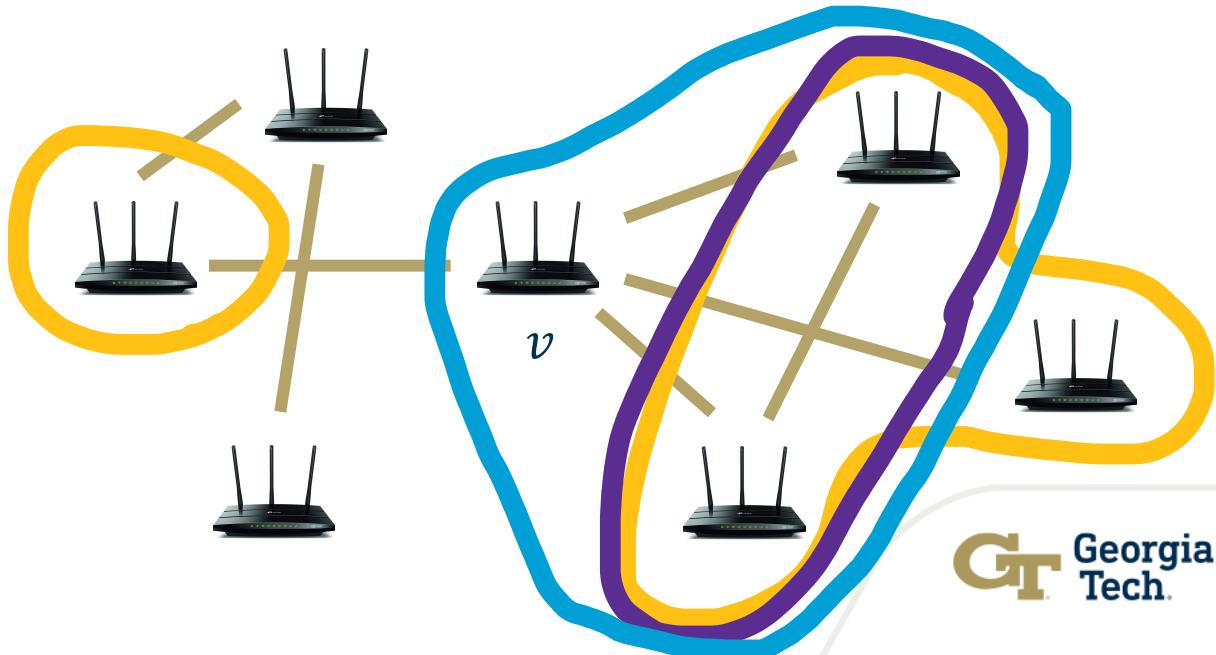
where

$N(v)$ is the set of neighbors of v

$N_\Delta(v)$ is the set of neighbors in triangles with v

$N_\Delta^+(v)$ is the closed set including v

If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

Given an undirected network of entities, a graph $G = (V, E)$ with **$|V|$ vertices** **$|E|$ edges**

where

$N(v)$ is the set of neighbors of v

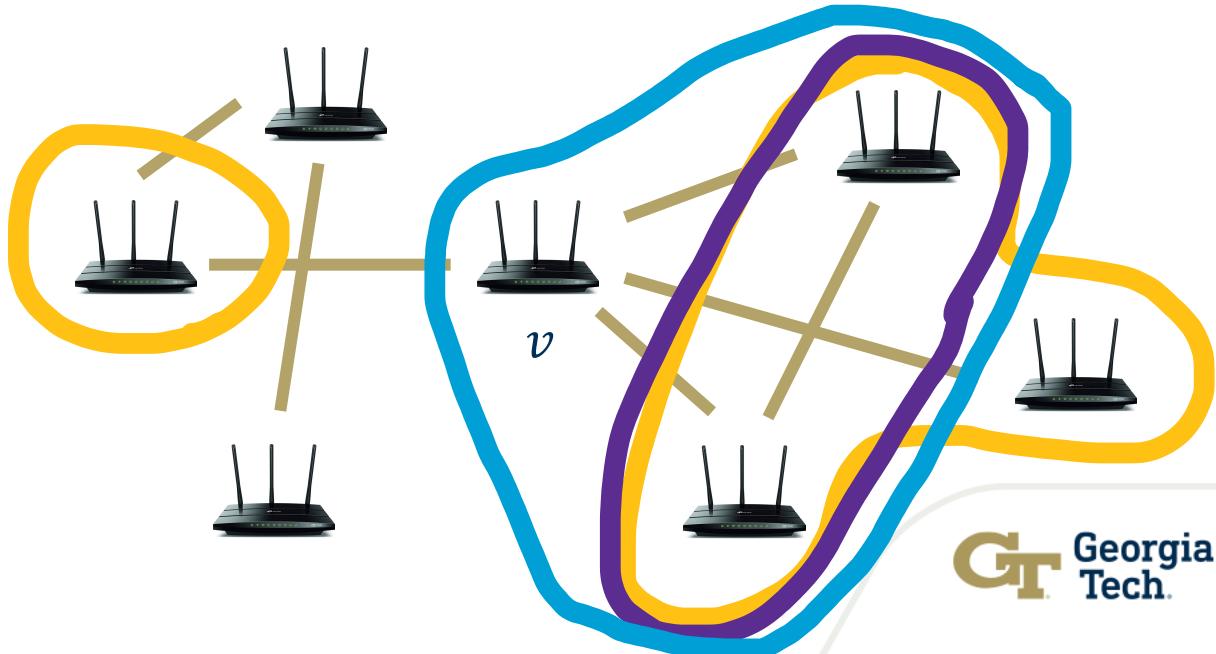
If a HW comp. is connected to two other HW comps., and these two other HW comps. as well are connected, then the trio of comps. is more cohesive

$N_\Delta(v)$ is the set of neighbors in triangles with v

$N_\Delta^+(v)$ is the closed set including v

$\Delta(v)$ is the triangle count of v

$\Delta(G)$ is the triangle count of G



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

Tailored for internet network topology analysis at a massive scale

Given an undirected network of entities, a graph $G = (V, E)$ with **$|V|$ vertices** **$|E|$ edges**

where

$N(v)$ is the set of neighbors of v

$N_\Delta(v)$ is the set of neighbors in triangles with v

$N_\Delta^+(v)$ is the closed set including v

$\Delta(v)$ is the triangle count of v

$\Delta(G)$ is the triangle count of G

$$TC(v) = \frac{\frac{1}{3} \sum_{u \in N_\Delta^+(v)} \Delta(u) + \sum_{w \in \{N(v) \setminus N_\Delta(v)\}} \Delta(w)}{\Delta(G)}$$

sum of Δ for triangle neighbors sum of Δ for non-triangle neighbors

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

```

1: function TRICENTACTOR( $L$ )           ▷ Perform the main computation
2:    $\Delta(G) = \text{TRICOUNTACTOR}(L)$       ▷ Compute global tricount
3:   for  $\{l_{vu} \in \hat{L}\}$  do
4:     if  $\{u \in N_\Delta(v)\}$  then          ▷ Compute  $\frac{\frac{1}{3} \sum_{u \in N_\Delta^+(v)} \Delta(u)}{\Delta(G)}$ 
5:       Send an active message to process1 (non-blocking)
6:        $Actor_p.\text{send}(1, \text{FINDOWNER}(L_u), u, v, \Delta(G), \frac{1}{3})$ 
7:     else                                ▷ Compute  $\frac{\sum_{u \in \{N(v) \setminus N_\Delta(v)\}} \Delta(u)}{\Delta(G)}$ 
8:        $Actor_p.\text{send}(1, \text{FINDOWNER}(L_u), u, v, \Delta(G), 1)$ 
9:      $TC(v) += \frac{1}{3} \Delta(v) / \Delta(G)$ 
10:    WAIT()                         ▷ Wait for the completion of local send/recv
11:    return  $TC$ 
12: function TRICENTACTORPROCESS1( $u, v, \Delta(G), mult$ )
13:   ▷ The process1 message handler
14:    $m_{tc} \leftarrow mult * \Delta(u) / \Delta(G)$ 
15:   Send an active message to process2 (non-blocking)
16:    $Actor_p.\text{send}(2, \text{FINDOWNER}(L_v), v, m_{tc})$ 
17: function TRICENTACTORPROCESS2( $v, m_{tc}$ )
18:   ▷ The process2 message handler
19:    $TC(v) += m_{tc}$ 

```

Notation: Let r_p be the local rank, \hat{L} be the local rows of L owned by r_p , c be the local counter owned by r_p , $Actor_p$ denote the actor instance that is running on r_p , N_Δ be the local array of lists holding set of neighbors in triangles with local rows owned by r_p , TC be the local rows of the triangle centrality matrix owned by r_p , $\Delta(G)$ be the global triangle count. send **Semantics:** $Actor.\text{send}(\text{proc handler}, \text{rank}, \text{packet})$

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

```

1: function TRICENTACTOR( $L$ )
2:    $\Delta(G) = \text{TRICOUNTACTOR}(L)$            ▷ Perform the main computation
3:   for  $\{l_{vu} \in \hat{L}\}$  do                  ▷ Compute global tricount
4:     if  $\{u \in N_\Delta(v)\}$  the             ▷ Send an active message to
5:       Actorp.send(1,FINDOWNER( $L_{wu}$ ), $v, w, u$ )  Actorp.send(1,FINDOWNER( $L_{wu}$ ), $v, w, u$ )
6:     else                                ▷  $Actor_p$ .send(1,FINDOWNER( $L_{wu}$ ), $v, w, u$ )
7:        $Actor_p$ .send(1,FINDOWNER( $L_{wu}$ ), $v, w, u$ )
8:        $TC(v) += \frac{1}{3}\Delta(v)/4$ 
9:   WAIT()
10:  return  $TC$ 
11: function TRICENTACTORPROCESS1( $v, w, u$ )
12:   if  $\{l_{wu} \in \hat{L}\}$  then           ▷ The process1 message handler
13:      $m_{tc} \leftarrow \text{mult} * \Delta(u)/\Delta(w)$ 
14:     Actorp.send(2,FINDOWNER( $L_v$ ), $v, w, u$ )  ▷ Send an active message to process2 (non-blocking)
15:      $TC(v) += m_{tc}$                       ▷  $Actor_p$ .send(2,FINDOWNER( $L_w$ ), $w, v, u$ )
16:   function TRICENTACTORPROCESS2( $i, j, k$ )
17:    $\Delta(i) += 1$ 
18:    $N_\Delta(i) \leftarrow N_\Delta(i) \cup \{j, k\}$ 
19:   function FINDOWNER( $\text{row}$ )           ▷ Returns a rank responsible for row
20:    $\text{row \% } P$                          ▷ 1-D Cyclic distribution
21:   return  $\text{row \% } P$ 

```

Notation: Let r_p be the local rank, \hat{L} be the local rows of L owned by r_p , c be the local counter owned by r_p , $Actor_p$ denote the actor instance that is running on r_p , N_Δ be the local array of lists holding set of neighbors in triangles with local rows owned by r_p , TC be the local rows of the $\Delta(G)$ be the global triangle count. c handler, rank, packet)

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

```

1: function TRICENTACTOR( $L$ )
2:    $\Delta(G) = \text{TRICOUNTACTOR}(L)$            ▷ Perform the main computation
3:   for  $\{l_{vu} \in \hat{L}\}$  do                  ▷ Compute global tricount
4:     if  $\{u \in N_\Delta(v)\}$  then          ▷ Send an active message to
5:        $Actor_p.\text{send}(1,\text{FINDOWNER}(L_{vu}), v, u)$  Actorp.send(1,FINDOWNER(Lvu),v,u)
6:     else                                ▷ Process1 message handler
7:        $Actor_p.\text{send}(1,\text{FINDOWNER}(L_{vu}), v, u)$ 
8:        $TC(v) += \frac{1}{3}\Delta(v)/4$ 
9:   WAIT()                               ▷ Process2 message handler
10:  return  $TC$ 
11: function TRICENTACTORPROCESS1( $v, w, u$ )
12:   if  $\{l_{wu} \in \hat{L}\}$  then          ▷ The process1 message han-
13:      $c += 1$                          ▷ dler, rank, packet)
14:      $Actor_p.\text{send}(2,\text{FINDOWNER}(L_{wu}), w, u)$ 
15:    $TC(v) += m_{tc}$                    ▷ The process2 message han-
16:    $m_{tc} \leftarrow \text{mult} * \Delta(u)/\Delta(v)$  ▷ dler, rank, packet)
17:   function TRICOUNTACTOR( $L$ )
18:     for  $\{l_{vw}, l_{vu} \in \hat{L}, u < w < v\}$  do ▷ Loop through local edges and grab trio of
19:        $Actor_p.\text{send}(1,\text{FINDOWNER}(L_{wu}), v, w, u)$  vertices ( $\{v, w, u\}$ ) to check for a triangle
20:     WAIT()
21:     return ALLREDUCE( $c$ )
22: function TRICOUNTACTORPROCESS1( $v, w, u$ )
23:   if  $\{l_{wu} \in \hat{L}\}$  then
24:      $c += 1$ 
25:      $Actor_p.\text{send}(2,\text{FINDOWNER}(L_v), v, w, u)$ 
26:      $Actor_p.\text{send}(2,\text{FINDOWNER}(L_w), w, v, u)$ 
27:      $Actor_p.\text{send}(2,\text{FINDOWNER}(L_u), u, v, w)$ 
28: function TRICOUNTACTORPROCESS2( $i, j, k$ )
29:    $\Delta(i) += 1$ 
30:    $N_\Delta(i) \leftarrow N_\Delta(i) \cup \{j, k\}$ 
31: function FINDOWNER( $\text{row}$ )           ▷ Returns a rank responsible for row
32:   return  $\text{row \% } P$                 ▷ 1-D Cyclic distribution

```

Notation:
 $\Delta(G)$ be the global triangle count.
 \hat{L} is the set of local edges.
 l_{vw} , l_{vu} are local edges.
 $Actor_p$ is the actor for process p .
 c is the counter for triangles.
 $\text{FINDOWNER}(L_{vw})$ returns the rank responsible for row L_{vw} .
 $\text{ALLREDUCE}(c)$ reduces the counter c across all processes.

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

```

1: function TRICENTACTOR( $L$ )
2:    $\Delta(G) = \text{TRICOUNTACTOR}(L)$            ▷ Perform the main computation
3:   for  $\{l_{vu} \in \hat{L}\}$  do                  ▷ Compute global tricount
4:     if  $\{u \in N_\Delta(v)\}$  then          ▷ Send an active message to
5:        $Actor_p.\text{send}(1,\text{FINDOWNER}(L_{vu}), v, u)$   Actorp.send(1,F
6:     else                                ▷
7:        $Actor_p.\text{send}(1,\text{FINDOWNER}(L_{vu}), v, w, u)$   Actorp.send(1,F
8:        $TC(v) += \frac{1}{3}\Delta(v)/4$            ▷
9:     WAIT()                            ▷
10:    return  $TC$                          ▷
11: function TRICENTACTORPROCESS1( $v, w, u$ )
12:   if  $\{l_{wu} \in \hat{L}\}$  then          ▷ The process1 message han
13:      $m_{tc} \leftarrow \text{mult} * \Delta(u)/\Delta(w)$   dles
14:      $Actor_p.\text{send}(2,\text{FINDOWNER}(L_{wu}), u, v, w)$   ▷ Send an active message to
15:      $TC(v) += m_{tc}$                       ▷ The process2 message han
16:   function TRICENTACTORPROCESS2( $i, j, k$ )
17:      $\Delta(i) += 1$                          ▷
18:      $N_\Delta(i) \leftarrow N_\Delta(i) \cup \{j, k\}$   Returns a rank responsible for row
19:   function FINDOWNER( $\text{row}$ )           ▷ 1-D Cyclic distribution
20:     return  $\text{row \% } P$ 

```

Notation: $\Delta(v)$ be the degree of vertex v , $N_\Delta(v)$ is the neighborhood of vertex v , \hat{L} is the set of local edges, $\Delta(G)$ is the global triangle count.

Loop through local edges and grab trio of vertices ($\{v, w, u\}$) to check for a triangle

Send a non-blocking fine-grained active message to remote vertex owner

MSG HANDLER

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

```

1: function TRICENTACTOR( $L$ )
2:    $\Delta(G) = \text{TRICOUNTACTOR}(L)$            ▷ Perform the main computation
3:   for  $\{l_{vu} \in \hat{L}\}$  do                  ▷ Compute global tricount
4:     if  $\{u \in N_\Delta(v)\}$  then          ▷ Send an active message to
5:        $Actor_p.\text{send}(1,\text{FINDOWNER}(L_{vu}), v, u)$   Actorp.send(1,F
6:     else                                Actorp.send(1,F
7:        $Actor_p.\text{send}(1,\text{FINDOWNER}(L_{vu}), v, u)$           TC(v) +=  $\frac{1}{3}\Delta(v)/4$ 
8:      $TC(v) += \frac{1}{3}\Delta(v)/4$            ▷ Send an active message to
9:     WAIT()                            process1 (non-blocking)
10:    return  $TC$                       Actorp.send(1,F
11: function TRICENTACTORPROCESS1( $v, w, u$ )  ▷ The process1 message han
12:    $m_{tc} \leftarrow \text{mult} * \Delta(u)/\Delta(v)$   dles
13:    $Actor_p.\text{send}(2,\text{FINDOWNER}(L_v), v, w, u)$   ▷ Send an active message to
14:   function TRICENTACTORPROCESS2( $i, j, k$ )  process2 (non-blocking)
15:    $TC(v) += m_{tc}$                       Actorp.send(2,F
16:    $\Delta(i) += 1$                          Actorp.send(2,F
17:    $N_\Delta(i) \leftarrow N_\Delta(i) \cup \{j, k\}$   Actorp.send(2,F
18:   function FINDOWNER( $\text{row}$ )           ▷ Returns a rank responsible for row
19:   return  $\text{row \% } P$                  ▷ 1-D Cyclic distribution

```

Notation: $\Delta(G)$ be the global degree of vertex set G , $\Delta(v)$ is the degree of vertex v , $N_\Delta(v)$ is the neighborhood of vertex v , \hat{L} is the list of local edges, L is the list of global edges, $\Delta(i)$ is the local triangle count of vertex i , $TC(v)$ is the total triangle count of vertex v , m_{tc} is the mult factor of triangle count, P is the number of processes.

MSG HANDLER:

- Loop through local edges and grab trio of vertices ($\{v, w, u\}$) to check for a triangle
- Send a non-blocking fine-grained active message to remote vertex owner
- If triangle present, increment local counter and send non-blocking fine-grained active messages to each of v, w, u to update their vertex triangle count and triangle neighbor lists

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

```

1: function TRICENTACTOR( $L$ )
2:    $\Delta(G) = \text{TRICOUNTACTOR}(L)$            ▷ Perform the main computation
3:   for  $\{l_{vu} \in \hat{L}\}$  do                  ▷ Compute global tricount
4:     if  $\{u \in N_\Delta(v)\}$  then             ▷ Send an active message to
5:        $Actor_p.\text{send}(1,\text{FINDOWNER}(L_{vu}), v, u)$  Actorp.send(1,F
6:     else                                     ▷ Send an active message to remote vertex
7:        $Actor_p.\text{send}(1,\text{FINDOWNER}(L_{vu}), v, w, u)$  owner
8:        $TC(v) += \frac{1}{3}\Delta(v)/m_{tc}$           ▷ Loop through local edges and grab trio of
9:   end for                                ▷ vertices ( $\{v, w, u\}$ ) to check for a triangle
10:  return  $\Delta(G)$                          ▷  $\Delta(G)$  be the global triangles count
11:  Await completion of all local sends and receives
12:  Send an active message to process1 (non-blocking)
13:   $Actor_p.\text{send}(1,\text{FINDOWNER}(L_{vu}), v, w, u)$ 
14: function TRICENTACTORPROCESS1( $v, w, u$ )
15:    $c = 0$                                  ▷ If triangle present, increment local
16:   if  $\{l_{wu} \in \hat{L}\}$  then            ▷ counter and send non-blocking fine-grained active messages to
17:      $c += 1$                             ▷ each of  $v, w, u$  to update their
18:     Send an active message to process2 (non-blocking) vertex triangle count and triangle
19:      $Actor_p.\text{send}(2,\text{FINDOWNER}(L_v), v, w, u)$  neighbor lists
20:      $Actor_p.\text{send}(2,\text{FINDOWNER}(L_w), w, v, u)$ 
21:      $Actor_p.\text{send}(2,\text{FINDOWNER}(L_u), u, v, w)$ 
22:   function TRICOUNTACTORPROCESS2( $i, j, k$ )
23:      $\Delta(i) += 1$                          ▷ Returns a rank responsible for row
24:      $N_\Delta(i) \leftarrow N_\Delta(i) \cup \{j, k\}$  ▷ 1-D Cyclic distribution
25:   function FINDOWNER( $\text{row}$ )
26:     return  $\text{row \% } P$ 
27:   MSG HANDLER
28:   MSG HANDLER

```

Notation
be the
is run
in tri
the glo
count

Send a non-blocking fine-grained active message to remote vertex owner

If triangle present, increment local counter and send non-blocking fine-grained active messages to each of v, w, u to update their vertex triangle count and triangle neighbor lists

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

```

1: function TRICENTACTOR( $L$ )
2:    $\Delta(G) = \text{TRICOUNTACTOR}(L)$            ▷ Perform the main computation
3:   for  $\{l_{vu} \in \hat{L}\}$  do                  ▷ Compute global tricount
4:     if  $\{u \in N_\Delta(v)\}$  then          ▷ Send an active message to
5:        $Actor_p.\text{send}(1,\text{FINDOWNER}(L_{vu}), v, u)$  Actorp.send(1,F
6:     else                                ▷ Send an active message to remote vertex
7:        $Actor_p.\text{send}(1,\text{FINDOWNER}(L_{vu}), v, w, u)$  owner
8:        $TC(v) += \frac{1}{3}\Delta(v)/\Delta(u)$           ▷ Compute global tricount
9:   return  $\Delta(G)$                       ▷ Returns global reduced triangle count
10:  Await completion of all local sends and receives
11:  Send an active message to
12:     $Actor_p.\text{send}(2,\text{FINDOWNER}(L_{vu}), v, w, u)$ 
13:  return  $\Delta(G)$                       ▷ Returns global reduced triangle count
14:  Notation:  $\Delta(v)$  be the degree of vertex  $v$ ,  $N_\Delta(v)$  is the neighborhood of  $v$ ,  $\hat{L}$  is the set of local edges,  $\Delta(G)$  be the global triangles count
15:  function TRICOUNTACTOR( $L$ )
16:    for  $\{l_{vw}, l_{vu} \in \hat{L}, u < w < v\}$  do          ▷ Loop through local edges and grab trio of vertices ( $\{v, w, u\}$ ) to check for a triangle
17:      function TRICOUNTACTORPROCESS1( $v, w, u$ )
18:        if  $\{l_{wu} \in \hat{L}\}$  then          ▷ Send a non-blocking fine-grained active message to remote vertex owner
19:           $c += 1$ 
20:          WAIT()
21:        return ALLREDUCE( $c$ )
22:      function TRICOUNTACTORPROCESS2( $i, j, k$ )
23:        if  $\{l_{wu} \in \hat{L}\}$  then          ▷ If triangle present, increment local counter and send non-blocking fine-grained active messages to each of  $v, w, u$  to update their vertex triangle count and triangle neighbor lists
24:           $c += 1$ 
25:           $N_\Delta(i) \leftarrow N_\Delta(i) \cup \{j, k\}$ 
26:        MSG HANDLER
27:      function FINDOWNER( $row$ )
28:        return  $row \% P$           ▷ Returns a rank responsible for row
29:        ▷ 1-D Cyclic distribution
30:      MSG HANDLER
31:    function FINDOWNER( $row$ )
32:      return  $row \% P$           ▷ Returns a rank responsible for row
33:      ▷ 1-D Cyclic distribution

```

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

```
1: function TRICENTACTOR( $L$ )           ▷ Perform the main computation
2:    $\Delta(G) = \text{TRICOUNTACTOR}(L)$       ▷ Compute global tricount
3:   for  $\{l_{vu} \in \hat{L}\}$  do
4:     if  $\{u \in N_\Delta(v)\}$  then          ▷ Compute  $\frac{\frac{1}{3} \sum_{u \in N_\Delta^+(v)} \Delta(u)}{\Delta(G)}$ 
5:        $Actor_p.\text{send}(1, \text{FINDOWNER}(L_u), u, v, \Delta(G), \frac{1}{3})$ 
6:     else                                ▷ Compute  $\frac{\sum_{u \in \{N(v) \setminus N_\Delta(v)\}} \Delta(u)}{\Delta(G)}$ 
7:        $Actor_p.\text{send}(1, \text{FINDOWNER}(L_u), u, v, \Delta(G), 1)$ 
8:      $TC(v) += \frac{1}{3} \Delta(v) / \Delta(G)$ 
9:   WAIT()                            ▷ Wait for the completion of local send/recv
10:  return  $TC$ 
11: function TRICENTACTORPROCESS1( $u, v, \Delta(G), mult$ )
    ▷ The process1 message handler
12:    $m_{tc} \leftarrow mult * \Delta(u) / \Delta(G)$ 
    ▷ Send an active message to process2 (non-blocking)
13:    $Actor_p.\text{send}(2, \text{FINDOWNER}(L_v), v, m_{tc})$ 
14: function TRICENTACTORPROCESS2( $v, m_{tc}$ )
    ▷ The process2 message handler
15:    $TC(v) += m_{tc}$ 
```

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

```

1: function TRICENTACTOR( $L$ )
2:    $\Delta(G) = \text{TRICOUNTACTOR}(L)$ 
3:   for  $\{l_{vu} \in \hat{L}\}$  do
4:     if  $\{u \in N_\Delta(v)\}$  then
5:        $\triangleright$  Send an active message to process1 (non-blocking)
6:        $Actor_p.\text{send}(1, \text{FINDOWNER}(L_u), u, v, \Delta(G), \frac{1}{3})$ 
7:     else
8:        $\triangleright$  Compute  $\frac{\sum_{u \in N_\Delta^+(v)} \Delta(u)}{\Delta(G)}$ 
9:        $Actor_p.\text{send}(1, \text{FINDOWNER}(L_u), u, v, \Delta(G), 1)$ 
10:       $TC(v) += \frac{1}{3}\Delta(v)/\Delta(G)$ 
11:      WAIT()  $\triangleright$  Wait for the completion of local send/recv
12:      return  $TC$ 
13: function TRICENTACTORPROCESS1( $u, v, \Delta(G), mult$ )
14:    $\triangleright$  The process1 message handler
15:    $m_{tc} \leftarrow mult * \Delta(u)/\Delta(G)$ 
16:    $\triangleright$  Send an active message to process2 (non-blocking)
17:    $Actor_p.\text{send}(2, \text{FINDOWNER}(L_v), v, m_{tc})$ 
18: function TRICENTACTORPROCESS2( $v, m_{tc}$ )
19:    $\triangleright$  The process2 message handler
20:    $TC(v) += m_{tc}$ 

```

\triangleright Perform the main computation
 \triangleright Compute global tricount
 \triangleright Compute $\frac{\sum_{u \in N_\Delta^+(v)} \Delta(u)}{\Delta(G)}$
 \triangleright Compute $\frac{\sum_{u \in \{N(v) \setminus N_\Delta(v)\}} \Delta(u)}{\Delta(G)}$

Sum of the Δ for the triangle neighbors
 Send non-blocking message to remote vertex owner

Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

```

1: function TRICENTACTOR( $L$ )
2:    $\Delta(G) = \text{TRICOUNTACTOR}(L)$ 
3:   for  $\{l_{vu} \in \hat{L}\}$  do
4:     if  $\{u \in N_\Delta(v)\}$  then
5:        $\triangleright$  Send an active message to process1 (non-blocking)
6:        $Actor_p.\text{send}(1, \text{FINDOWNER}(L_u), u, v, \Delta(G), \frac{1}{3})$ 
7:     else
8:        $\triangleright$  Compute  $\frac{\sum_{u \in N(v) \setminus N_\Delta(v)} \Delta(u)}{\Delta(G)}$ 
9:        $Actor_p.\text{send}(1, \text{FINDOWNER}(L_u), u, v, \Delta(G), 1)$ 
10:       $TC(v) += \frac{1}{3}\Delta(v)/\Delta(G)$ 
11:      WAIT()  $\triangleright$  Wait for the completion of local server
12:      return  $TC$ 
13: function TRICENTACTORPROCESS1( $u, v, \Delta(G), mult$ )
14:    $\triangleright$  The process1 message handler
15:    $m_{tc} \leftarrow mult * \Delta(u)/\Delta(G)$ 
16:    $\triangleright$  Send an active message to process2 (non-blocking)
17:    $Actor_p.\text{send}(2, \text{FINDOWNER}(L_v), v, m_{tc})$ 
18: function TRICENTACTORPROCESS2( $v, m_{tc}$ )
19:    $\triangleright$  The process2 message handler
20:    $TC(v) += m_{tc}$ 

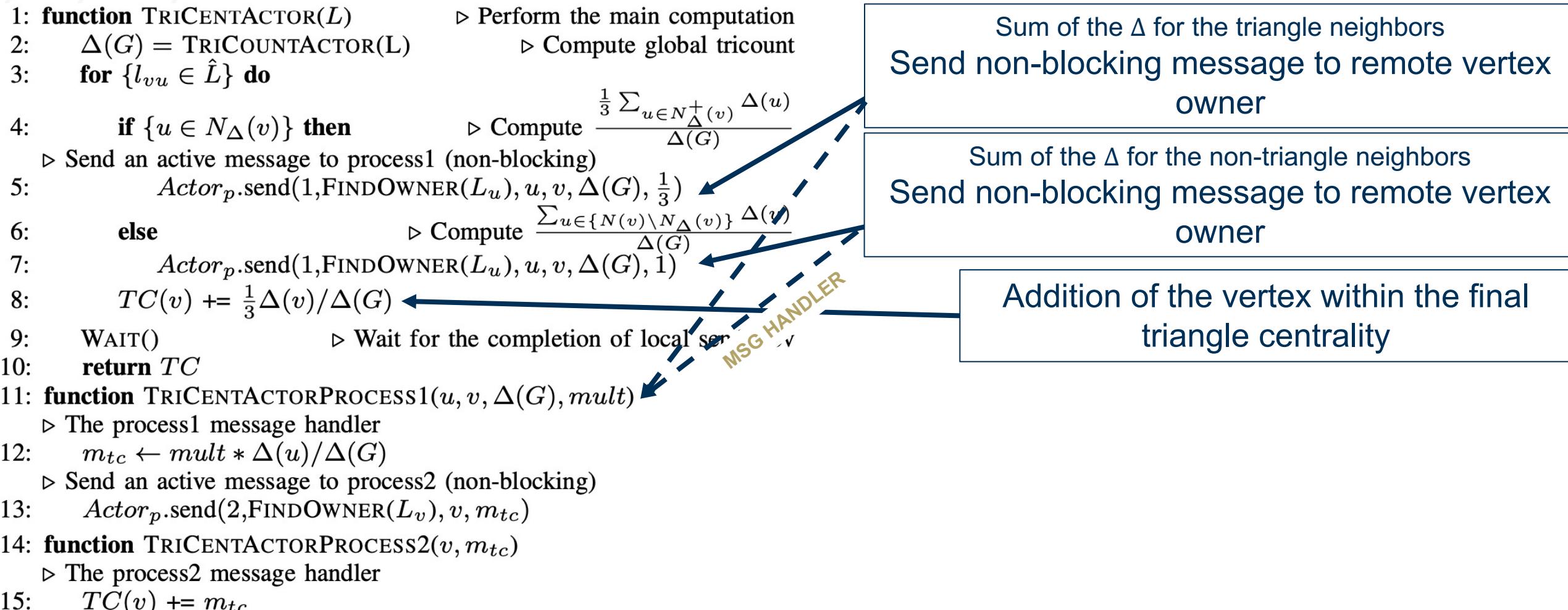
```

▷ Perform the main computation
 ▷ Compute global tricount
 ▷ Compute $\frac{\frac{1}{3} \sum_{u \in N_\Delta(v)} \Delta(u)}{\Delta(G)}$
 ▷ Compute $\frac{\sum_{u \in \{N(v) \setminus N_\Delta(v)\}} \Delta(u)}{\Delta(G)}$

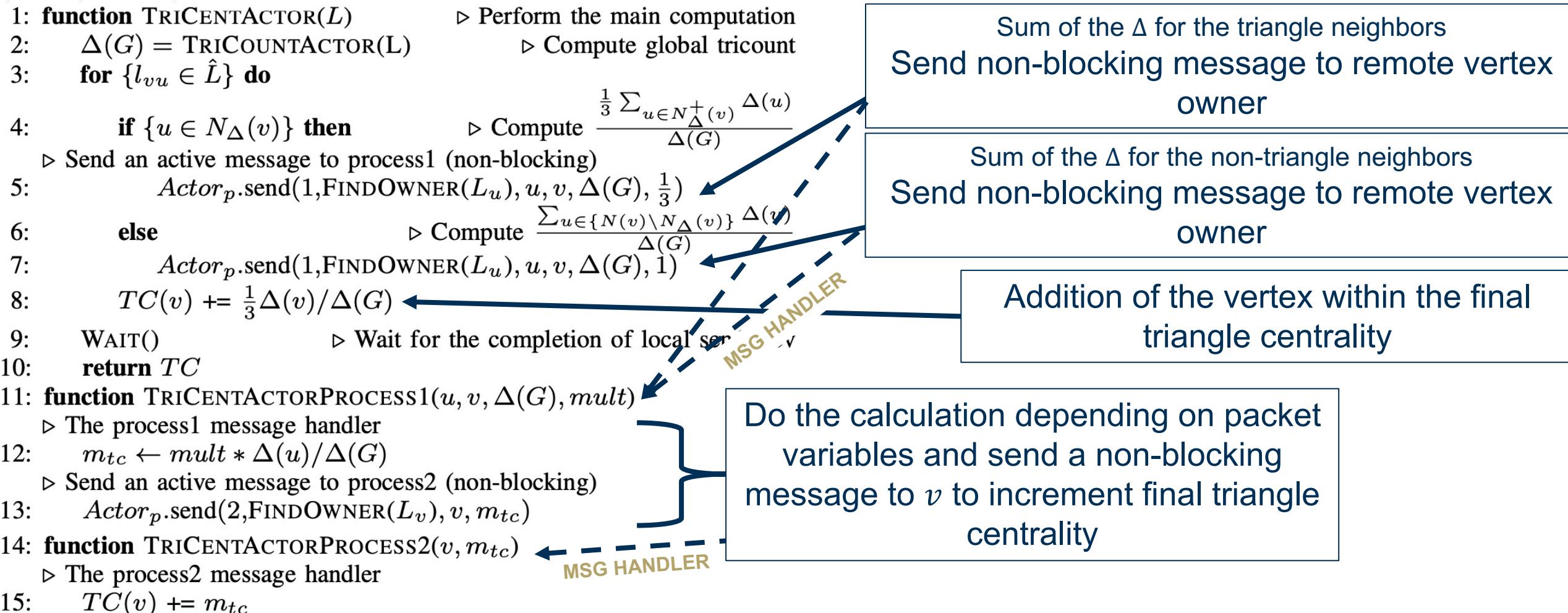
Sum of the Δ for the triangle neighbors
Send non-blocking message to remote vertex owner

Sum of the Δ for the non-triangle neighbors
Send non-blocking message to remote vertex owner

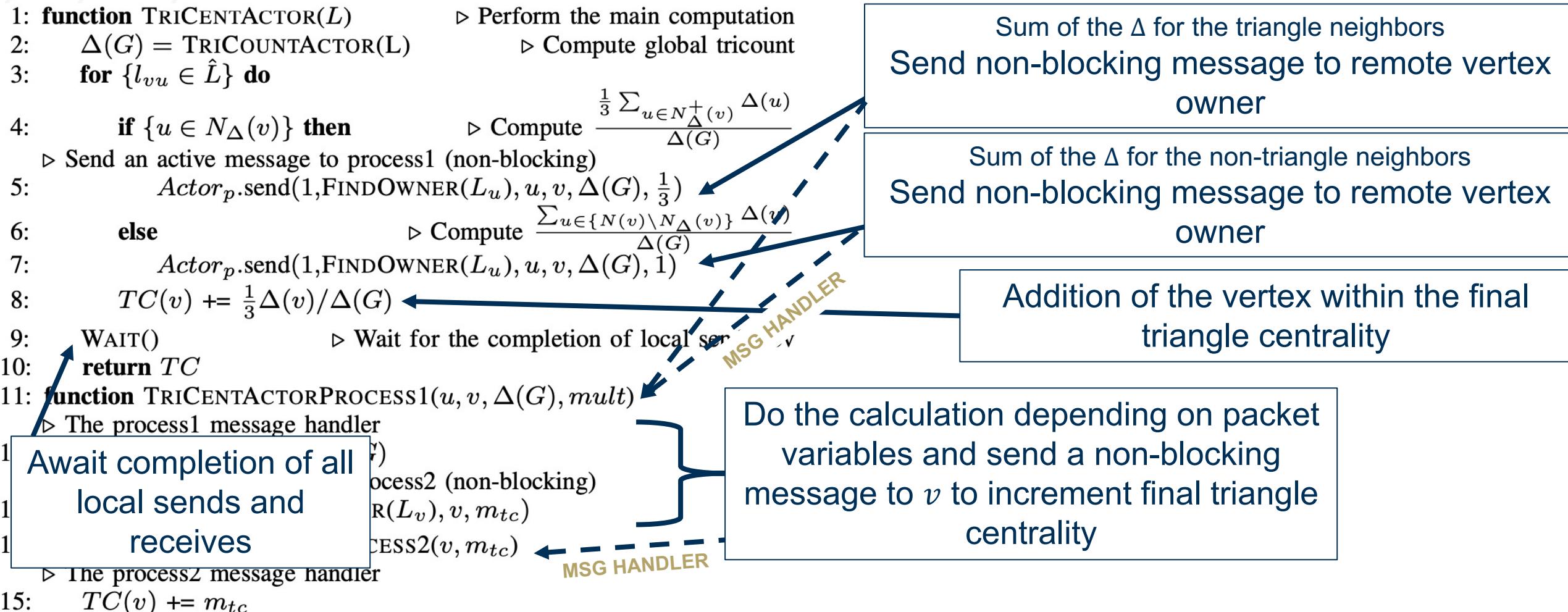
Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis



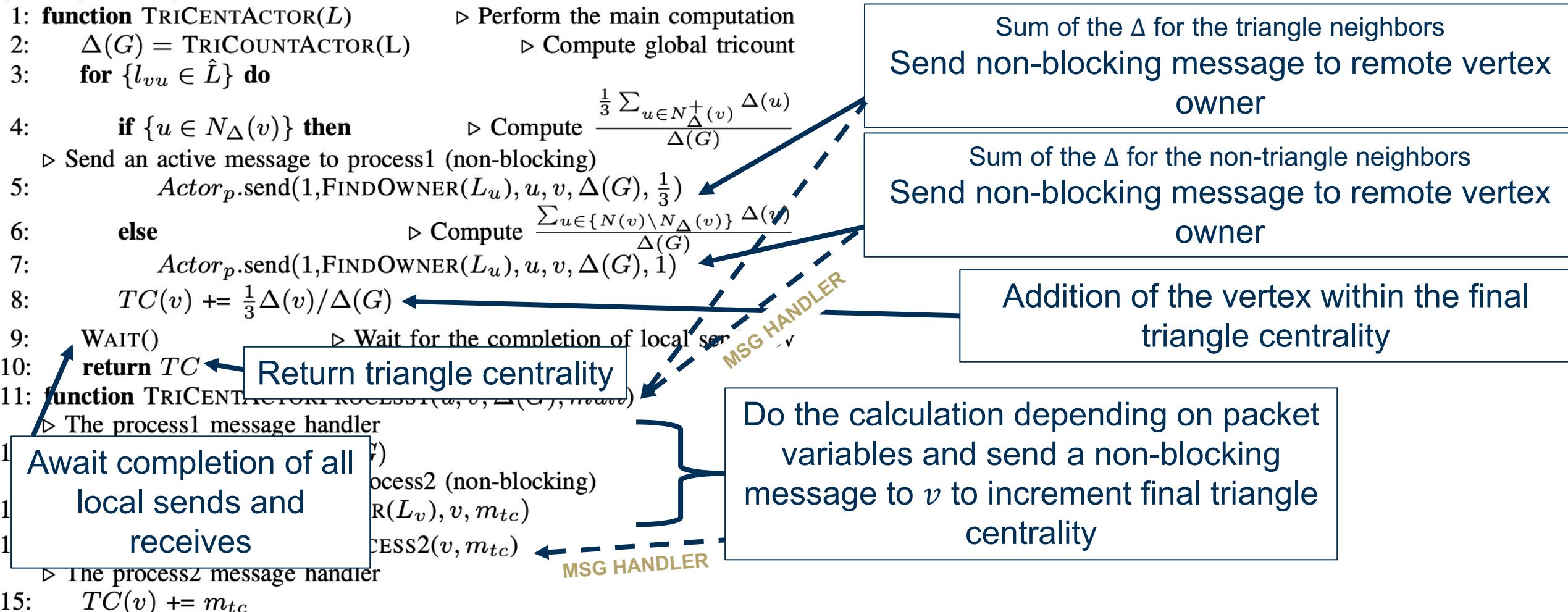
Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis



Distributed, Asynchronous, and Scalable Actor-Based Centrality for Internet Network Topology Analysis

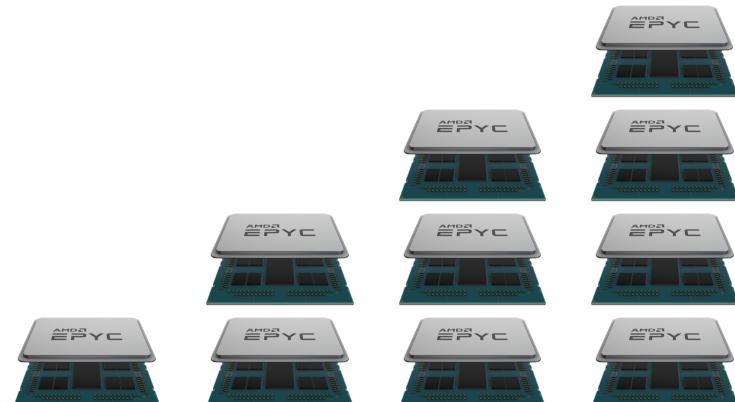


Experimental Setup and Architecture

- Experiments conducted on the CPU nodes of
 - The **Perlmutter Supercomputer** at the National Energy Research Scientific Computing Center (NERSC)
 - Each CPU node: 2x AMD EPYC 7763 (Milan) CPUs, 64 physical cores per CPU, 512 GB memory
 - The **HPC PACE Cluster** at Georgia Tech
 - Each CPU node: Dual Intel Xeon Gold 6226 CPUs, 24 total physical cores, 192 GB memory
- We present results using speedup of execution time compared to single core
- We use four large-scale internet network datasets:
 - A **Routers Network Dataset** ($|V| = 191K, |E| = 608K$)
 - An **IP Addresses Network Dataset** ($|V| = 2.3M, |E| = 21.6M$)
 - An **Autonomous Systems Network Dataset** ($|V| = 1.7M, |E| = 11.1M$)
 - A **Synthetic Network Dataset** (100K rows/core for weak scaling, $|V| = 205M, |E| = 13.9B$ at largest scale)
- Results for **different dimensions of scalability** are presented

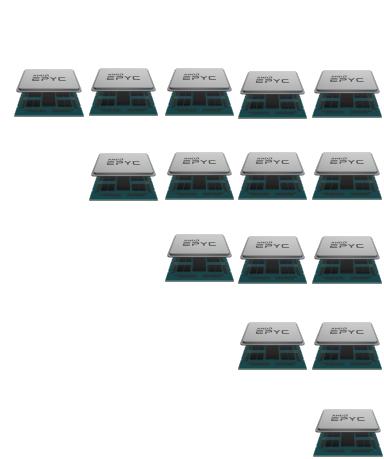
Dimensions of Scalability

(1) SCALE-OUT



increasing #nodes (SCALE-OUT, from 1 to 2,048 cores)

(2) SCALE-UP

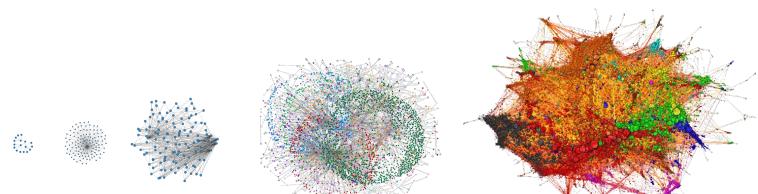


increasing #cores per node
(SCALE-UP, from 1 core/node to 128 cores/node)

HARDWARE SCALABILITY

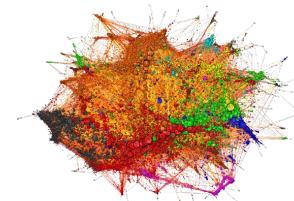
DATASET SCALABILITY

(1) WEAK SCALING



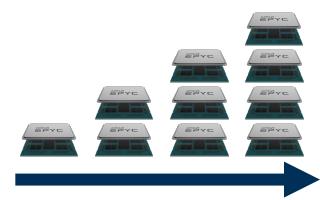
increasing graph size (synthetic dataset, 100K vertices per core)

(2) STRONG SCALING

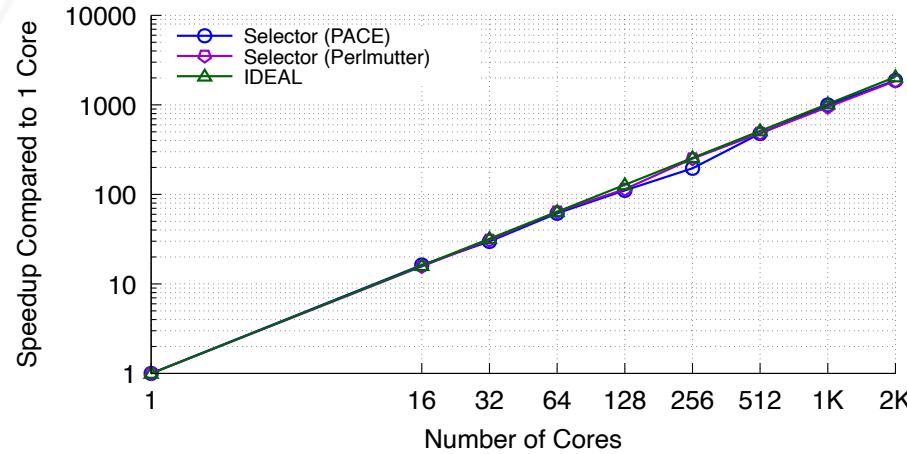


constant graph size
(Routers dataset, IP dataset, ASes dataset)

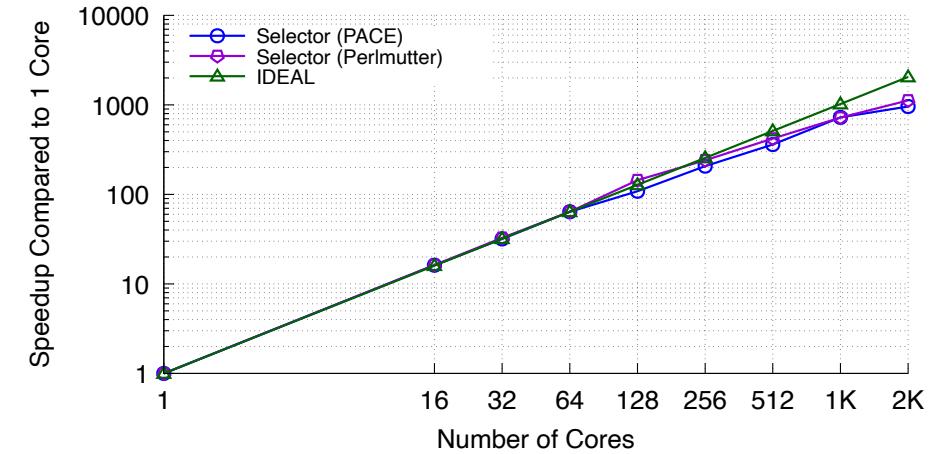
Performance Results: Scalability for SCALE-OUT



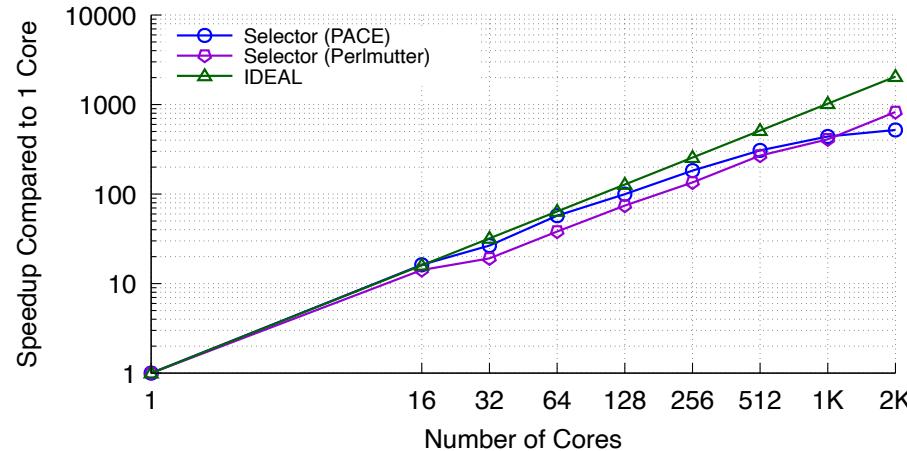
Routers Dataset
Strong Scaling



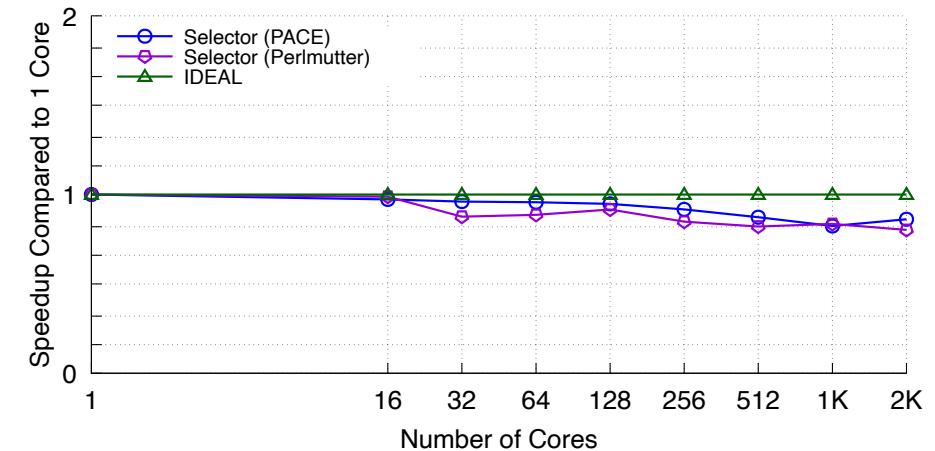
IPs Dataset
Strong Scaling



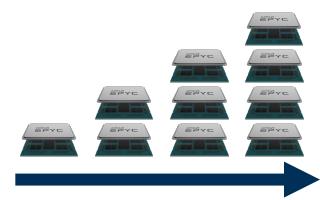
ASes Dataset
Strong Scaling



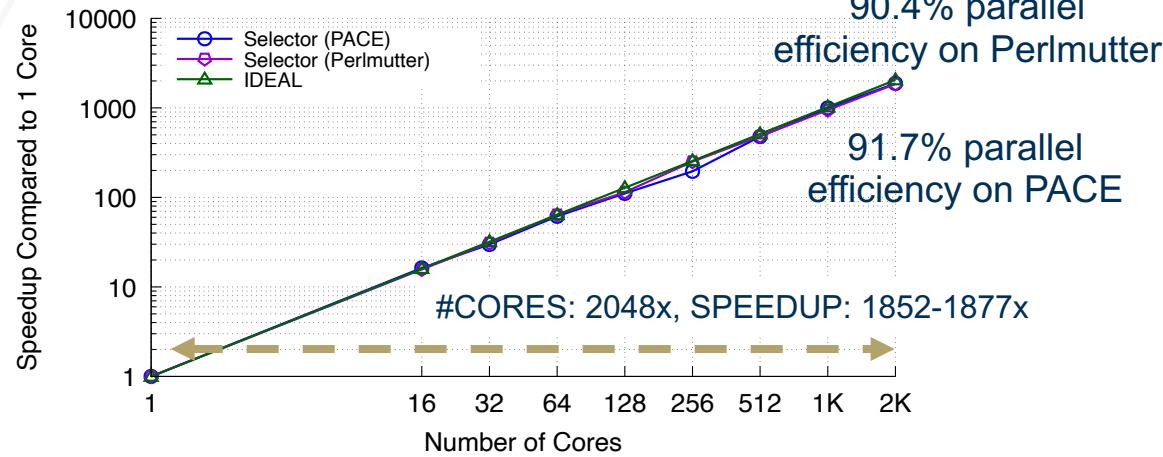
Synthetic Dataset
Weak Scaling



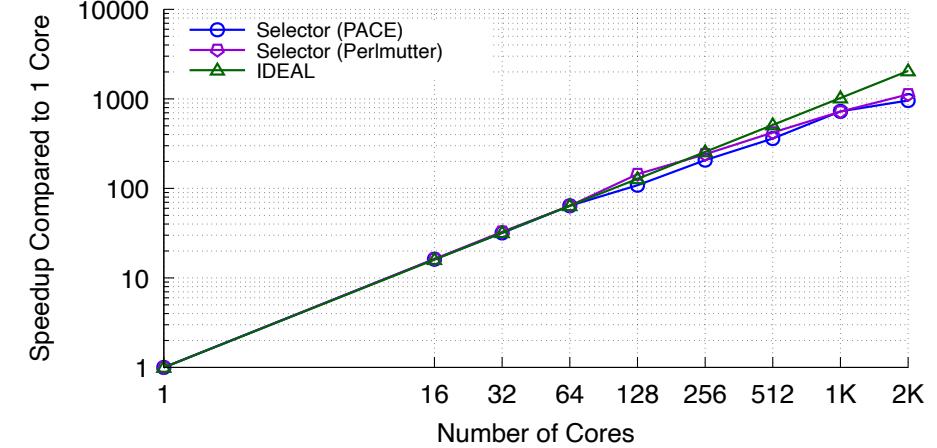
Performance Results: Scalability for SCALE-OUT



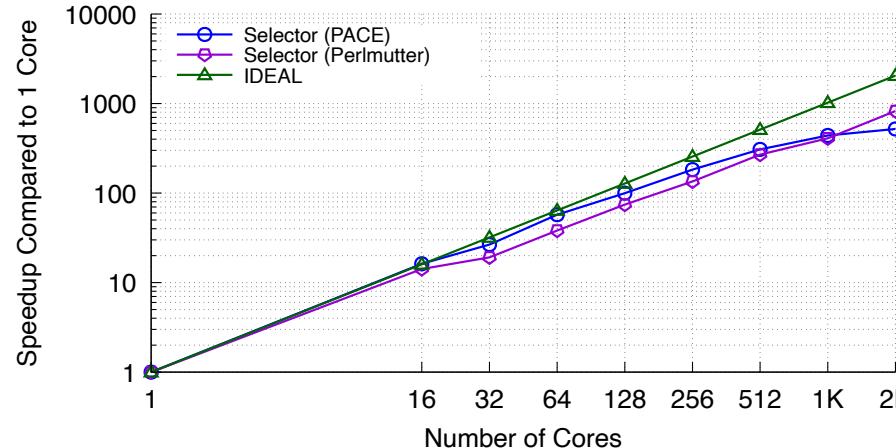
Routers Dataset
Strong Scaling



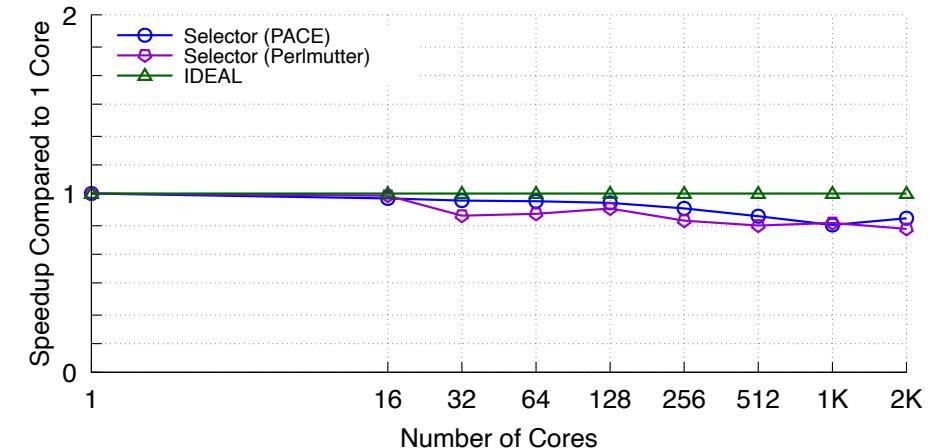
IPs Dataset
Strong Scaling



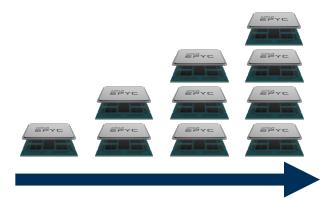
ASes Dataset
Strong Scaling



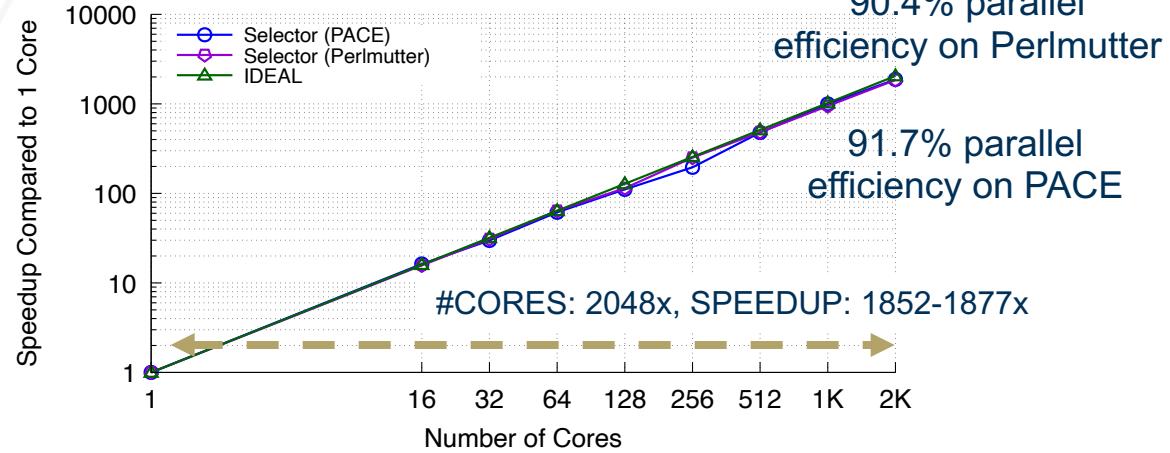
Synthetic Dataset
Weak Scaling



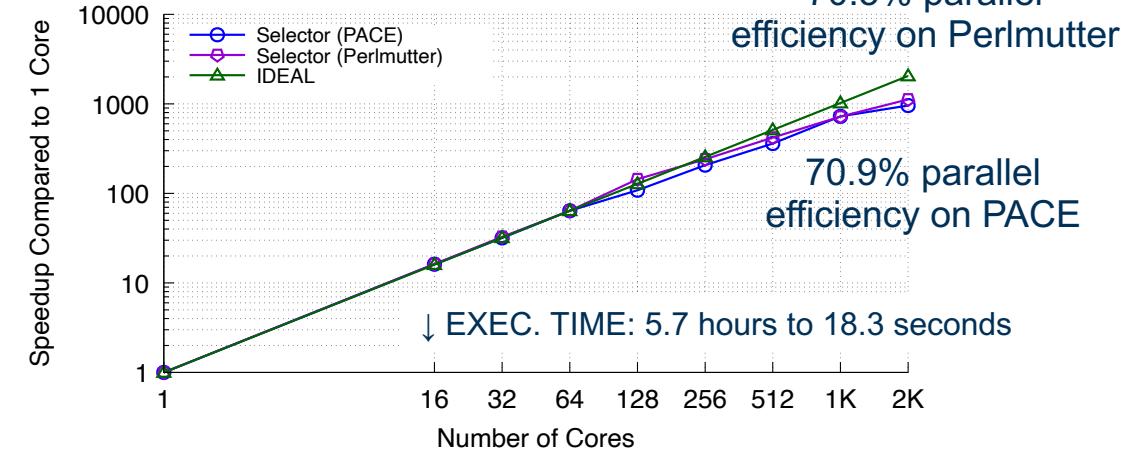
Performance Results: Scalability for SCALE-OUT



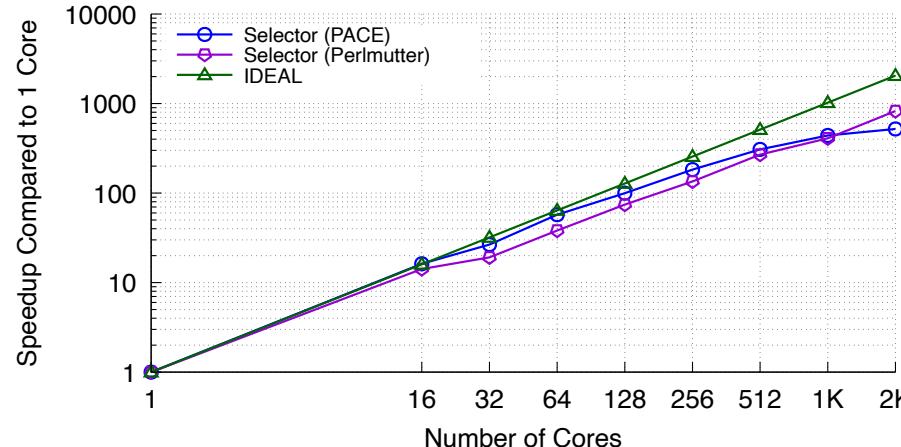
Routers Dataset
Strong Scaling



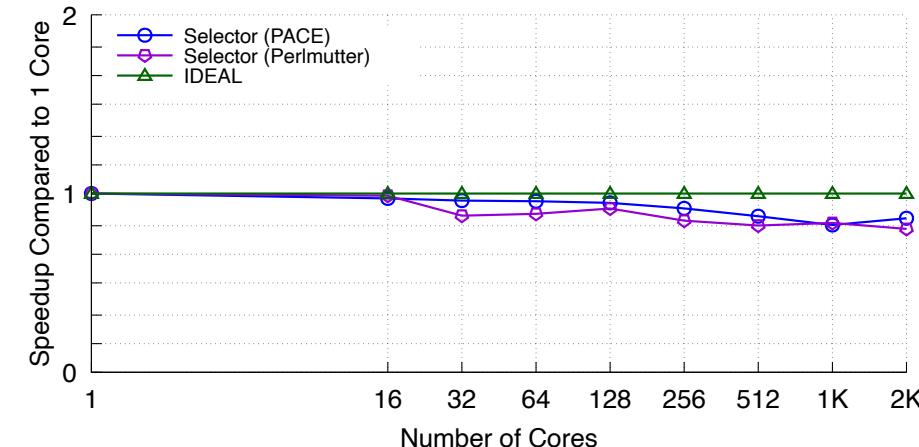
IPs Dataset
Strong Scaling



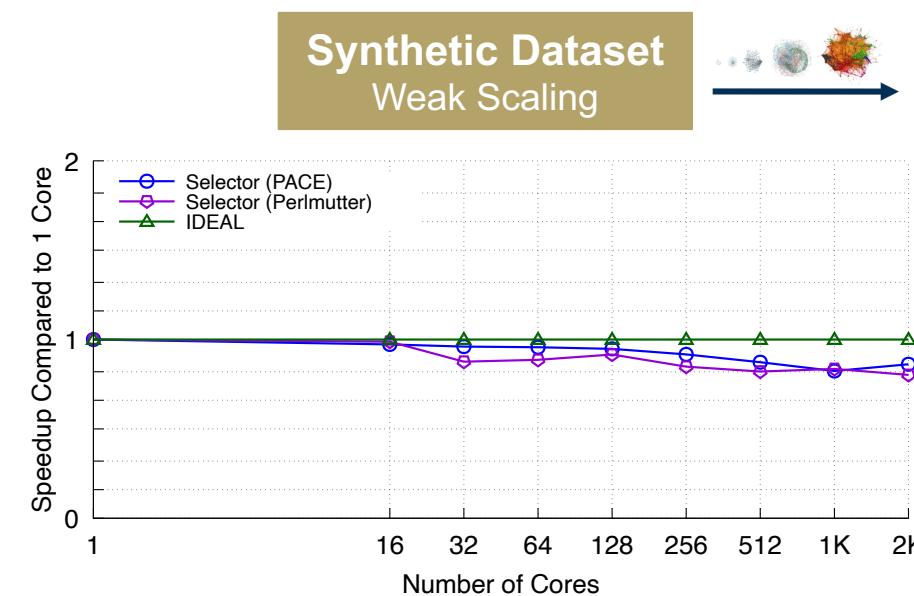
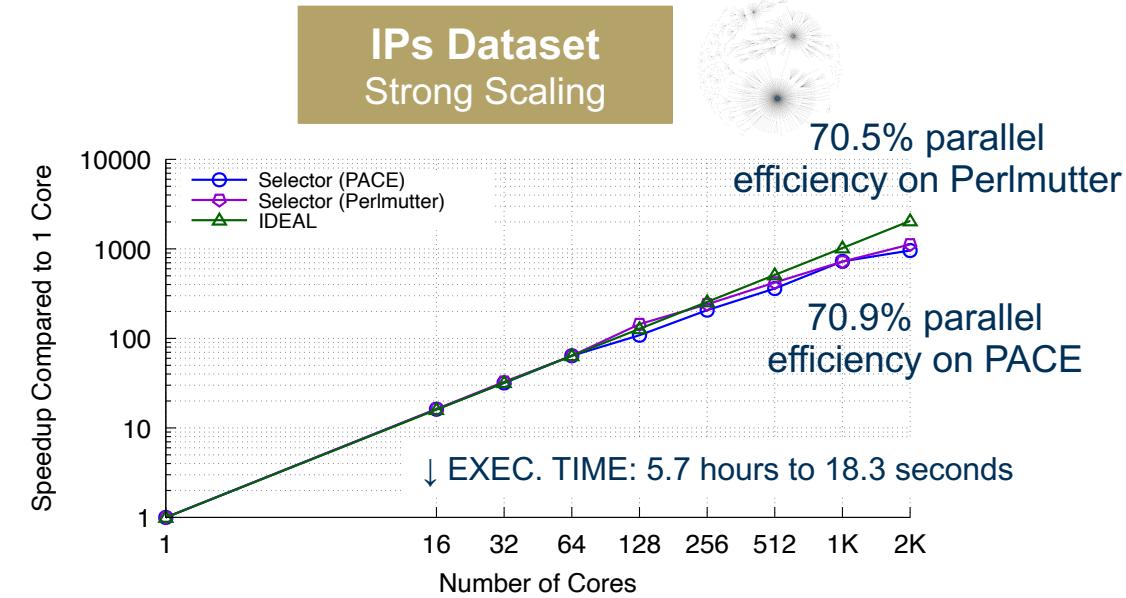
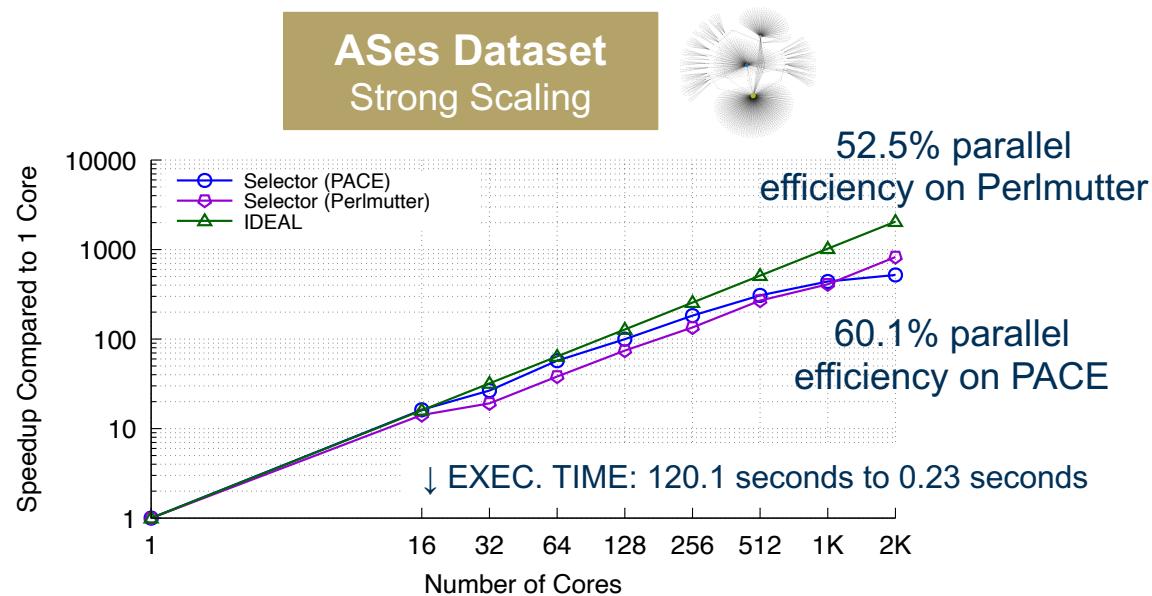
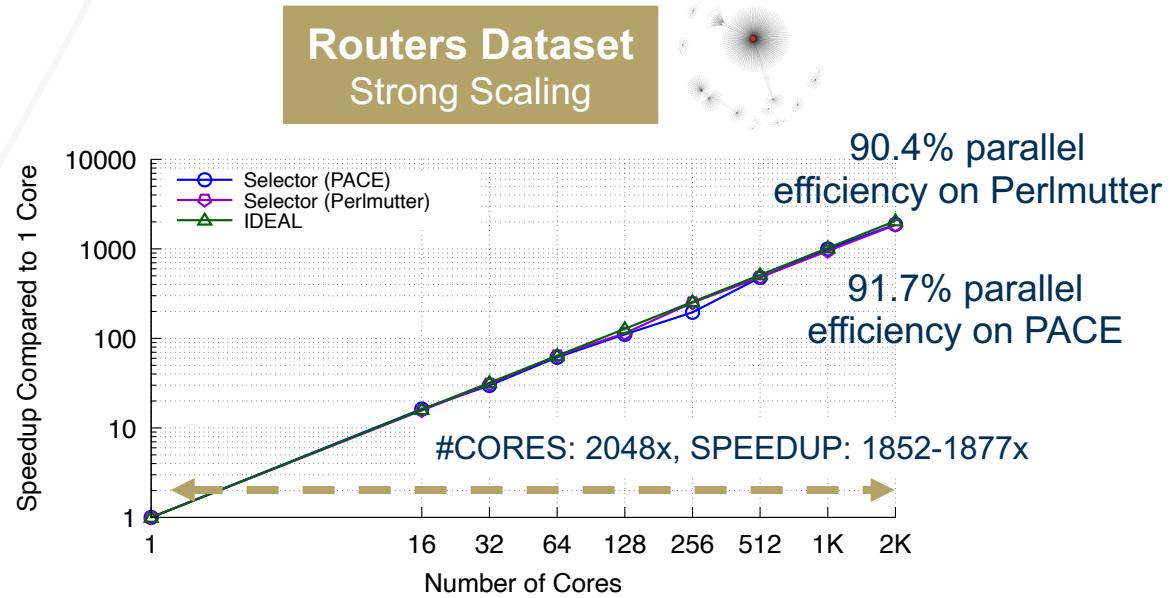
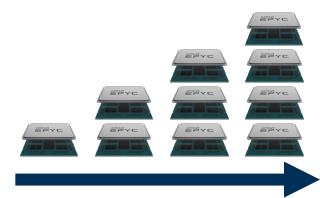
ASes Dataset
Strong Scaling



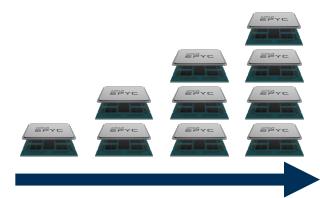
Synthetic Dataset
Weak Scaling



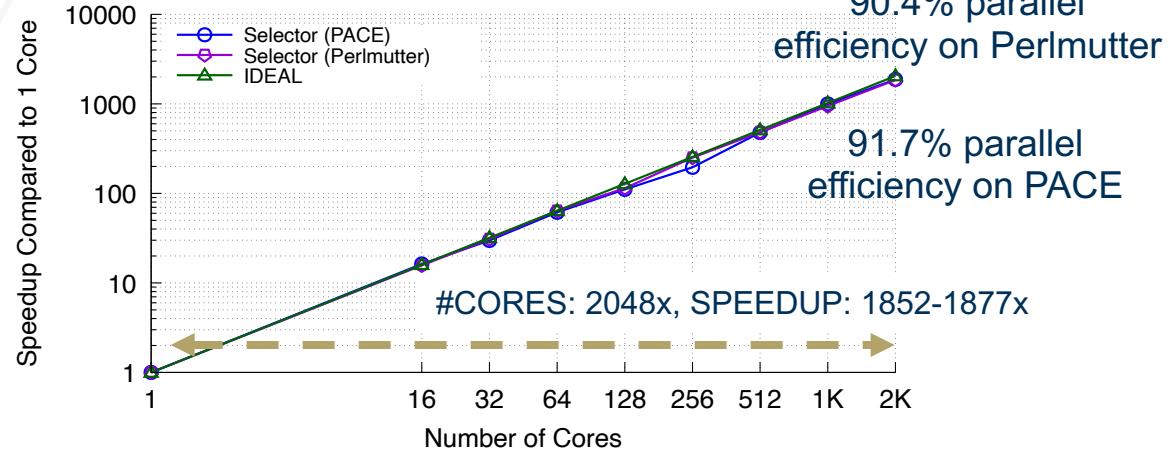
Performance Results: Scalability for SCALE-OUT



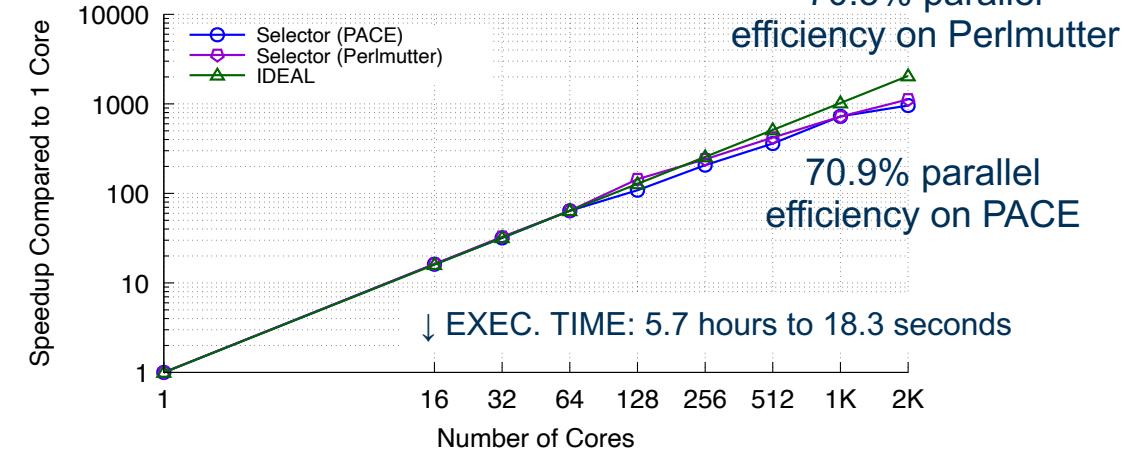
Performance Results: Scalability for SCALE-OUT



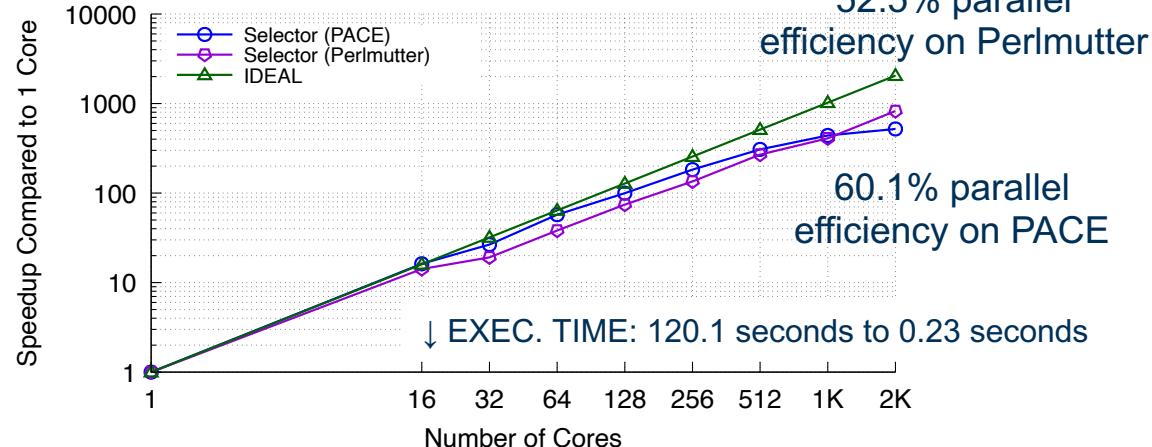
Routers Dataset
Strong Scaling



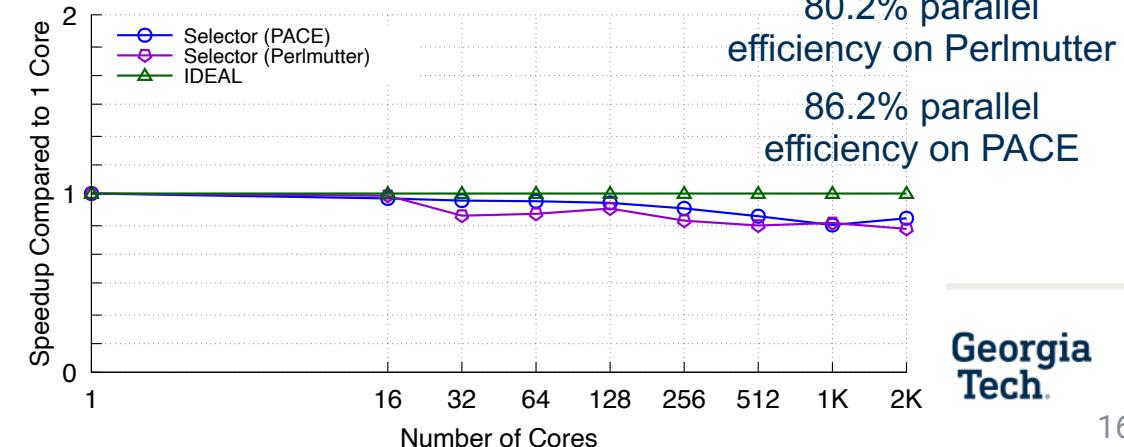
IPs Dataset
Strong Scaling



ASes Dataset
Strong Scaling



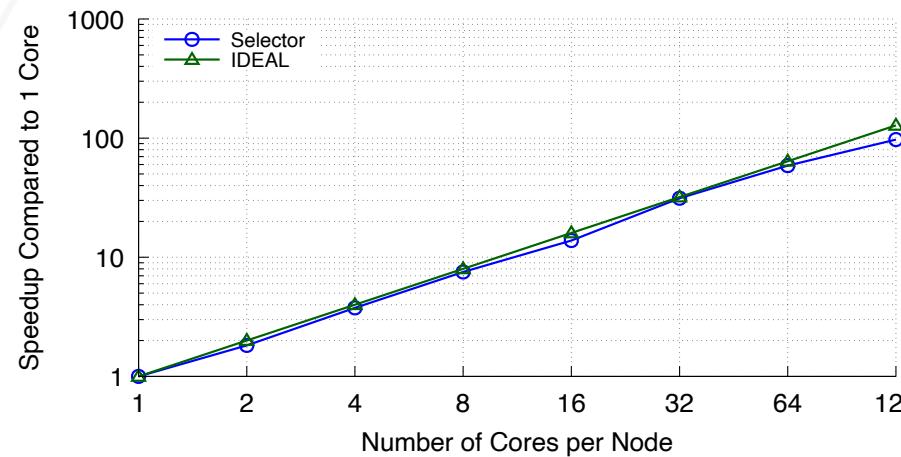
Synthetic Dataset
Weak Scaling



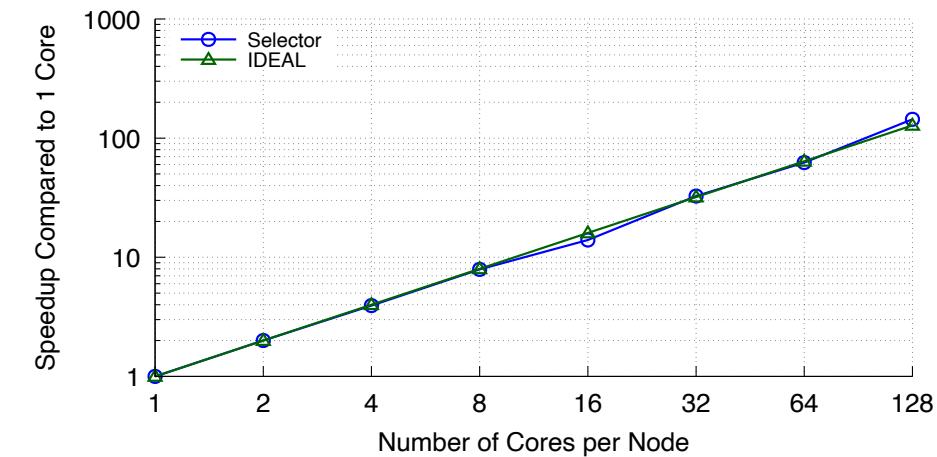


Performance Results: Scalability for SCALE - UP

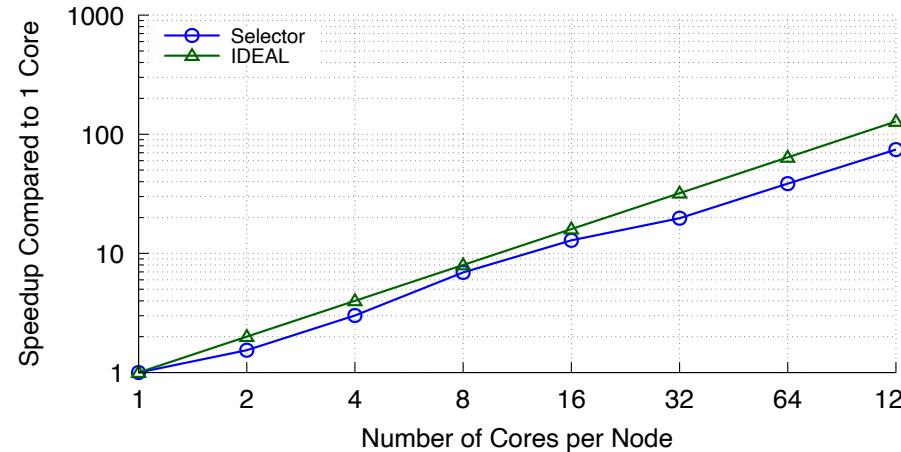
Routers Dataset
Strong Scaling



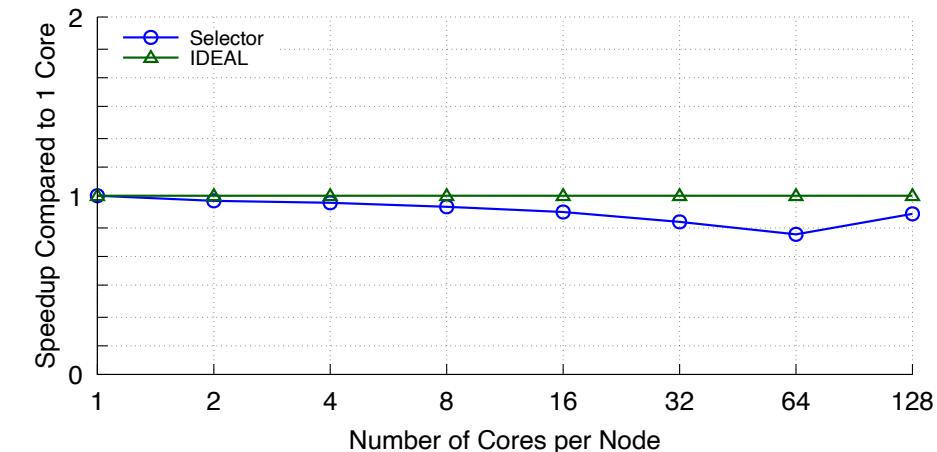
IPs Dataset
Strong Scaling



ASes Dataset
Strong Scaling



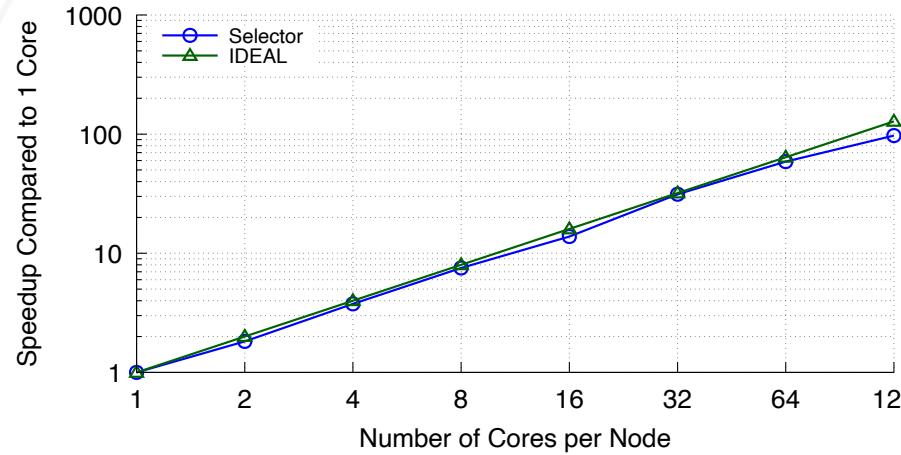
Synthetic Dataset
Weak Scaling



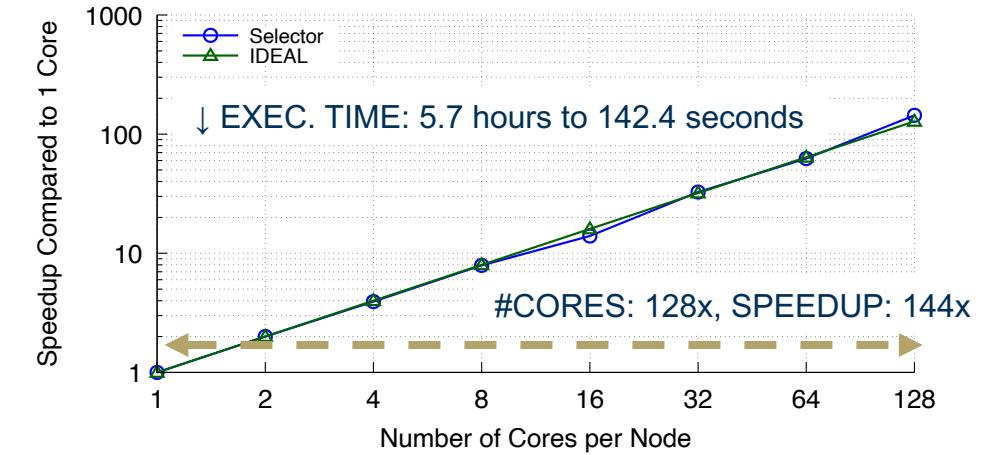


Performance Results: Scalability for SCALE - UP

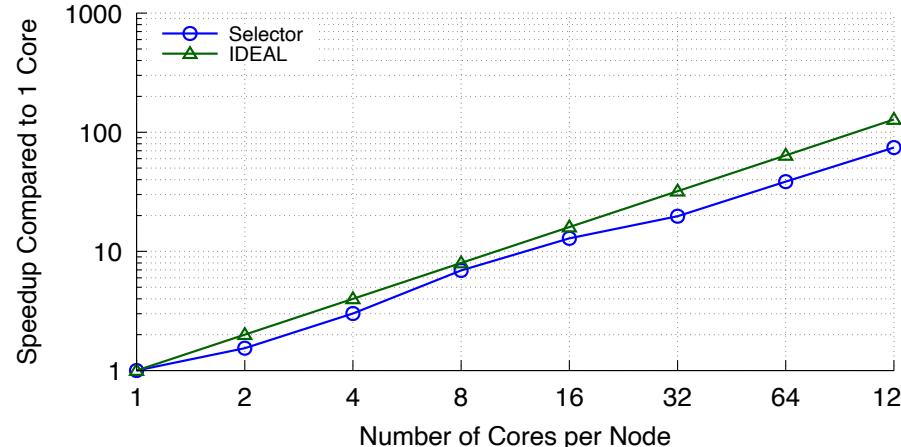
Routers Dataset
Strong Scaling



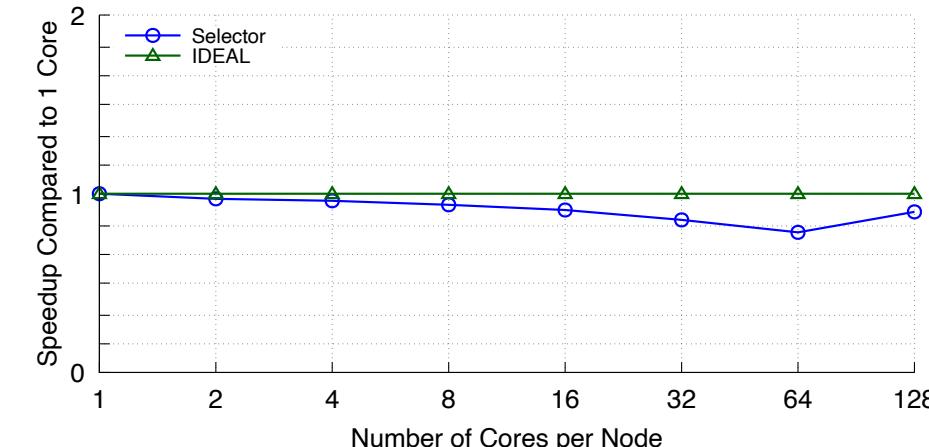
IPs Dataset
Strong Scaling



ASes Dataset
Strong Scaling



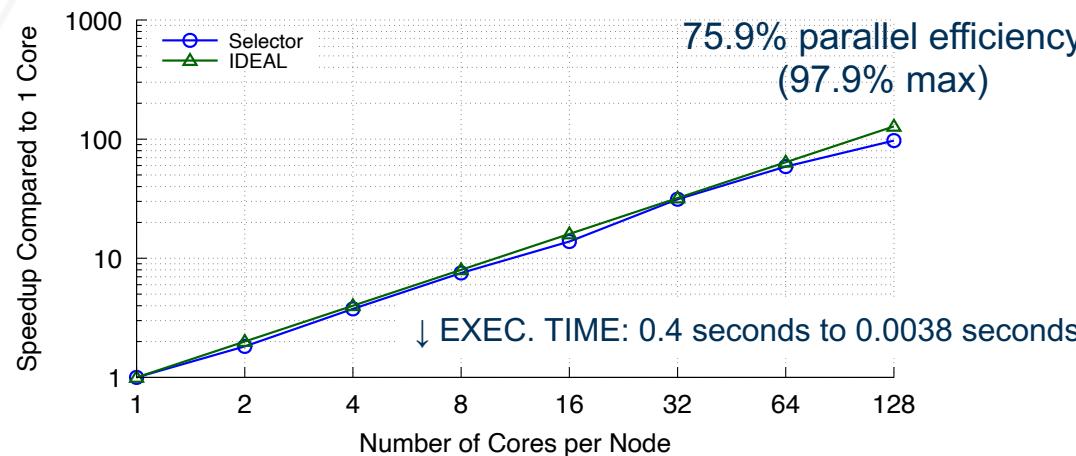
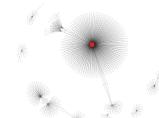
Synthetic Dataset
Weak Scaling



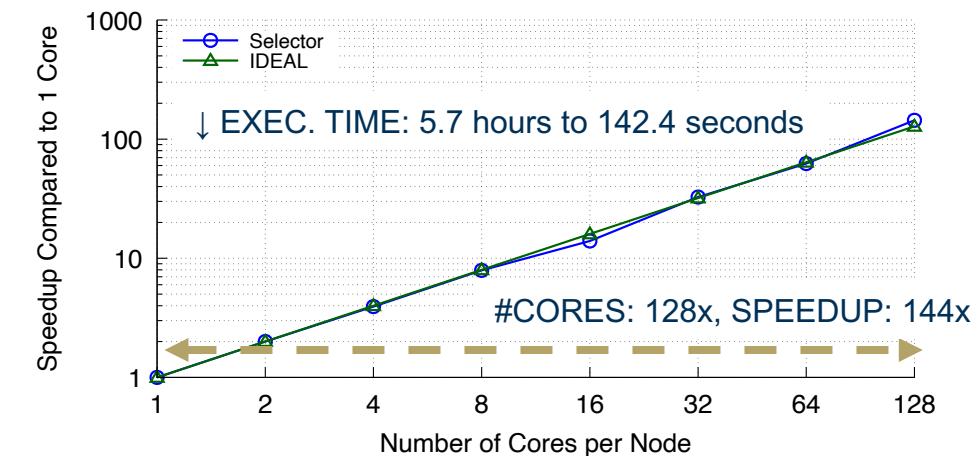


Performance Results: Scalability for SCALE - UP

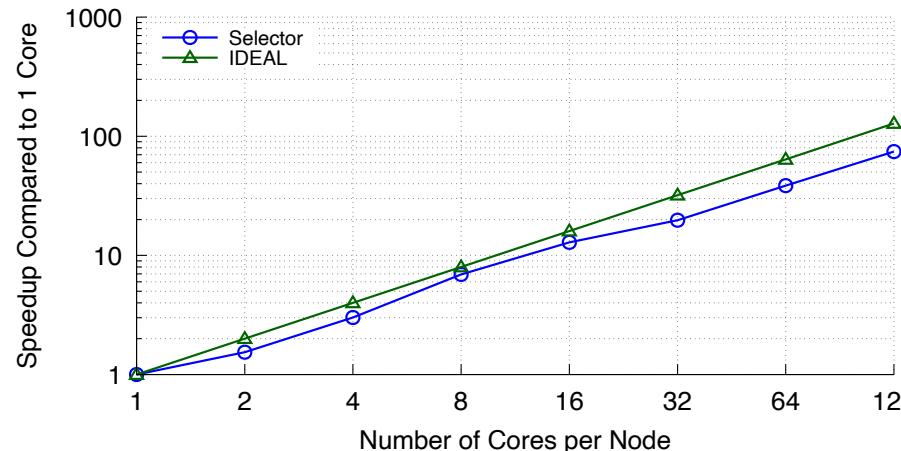
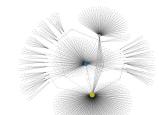
Routers Dataset
Strong Scaling



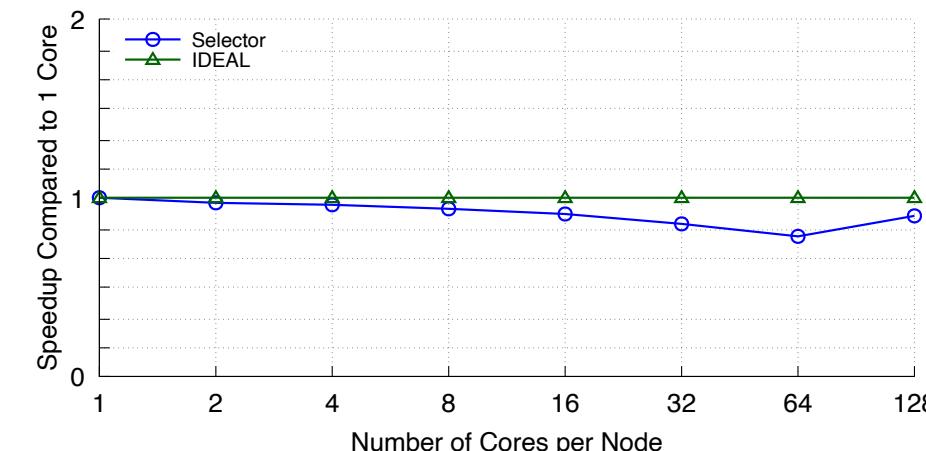
IPs Dataset
Strong Scaling



ASes Dataset
Strong Scaling



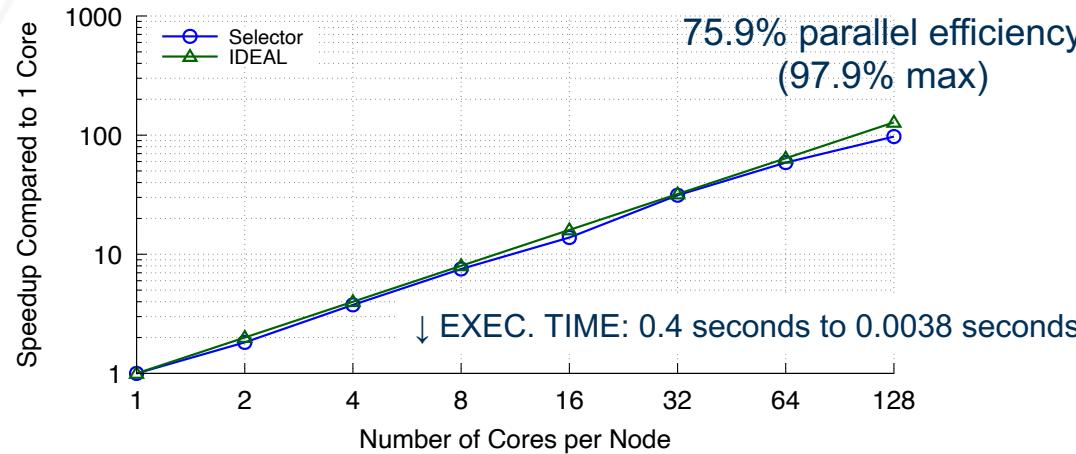
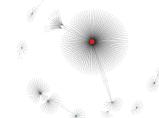
Synthetic Dataset
Weak Scaling



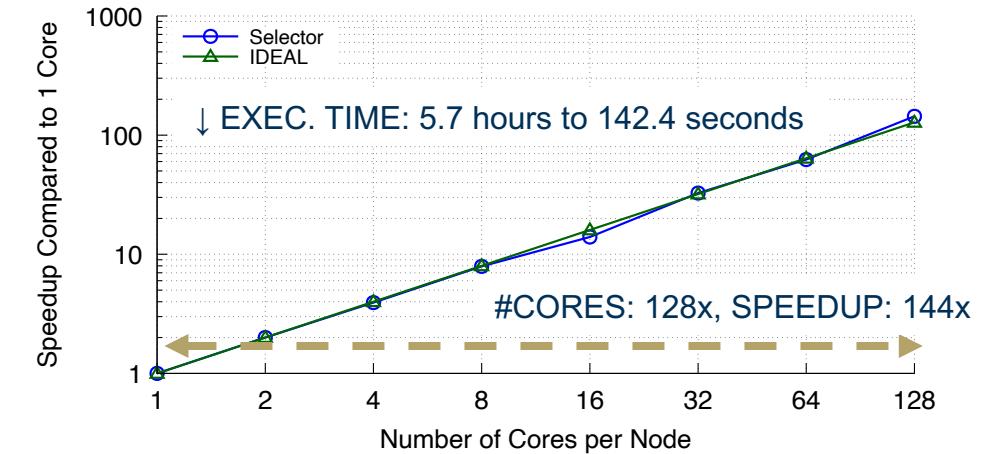
Performance Results: Scalability for SCALE - UP



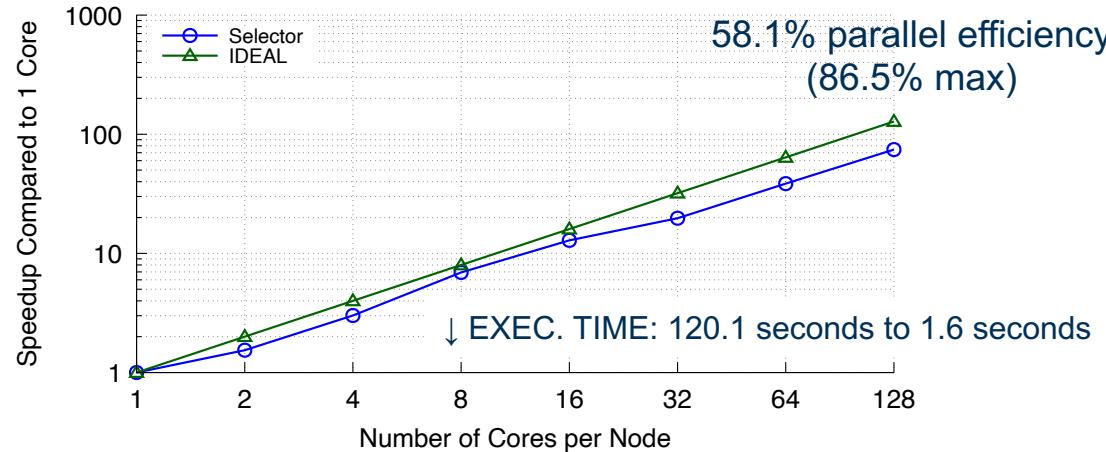
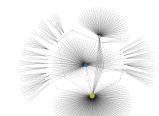
Routers Dataset
Strong Scaling



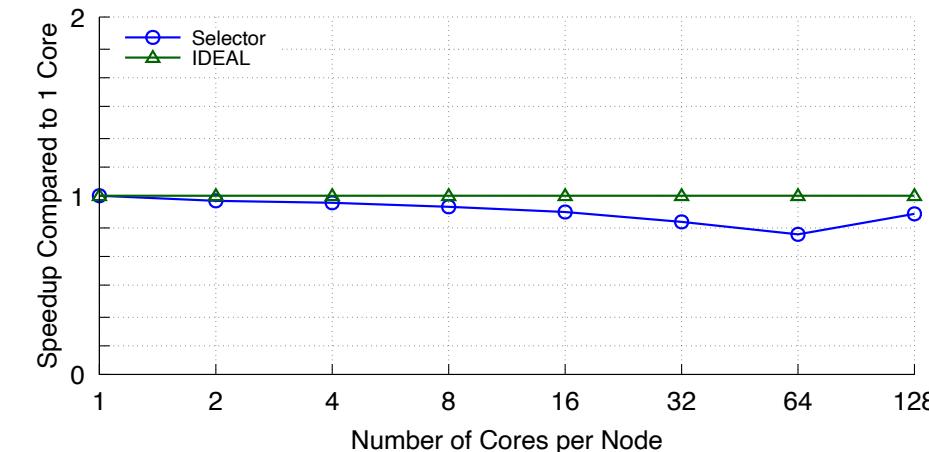
IPs Dataset
Strong Scaling



ASes Dataset
Strong Scaling



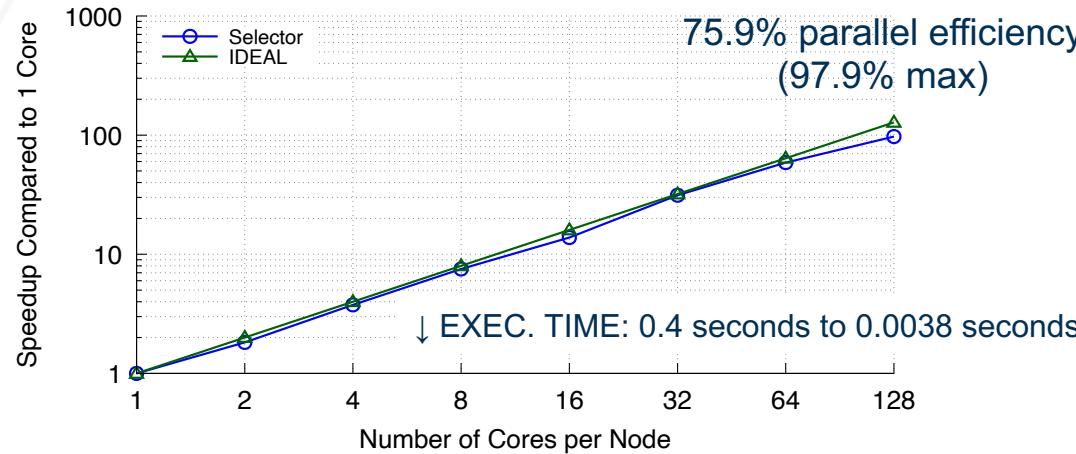
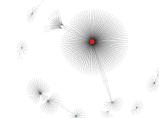
Synthetic Dataset
Weak Scaling



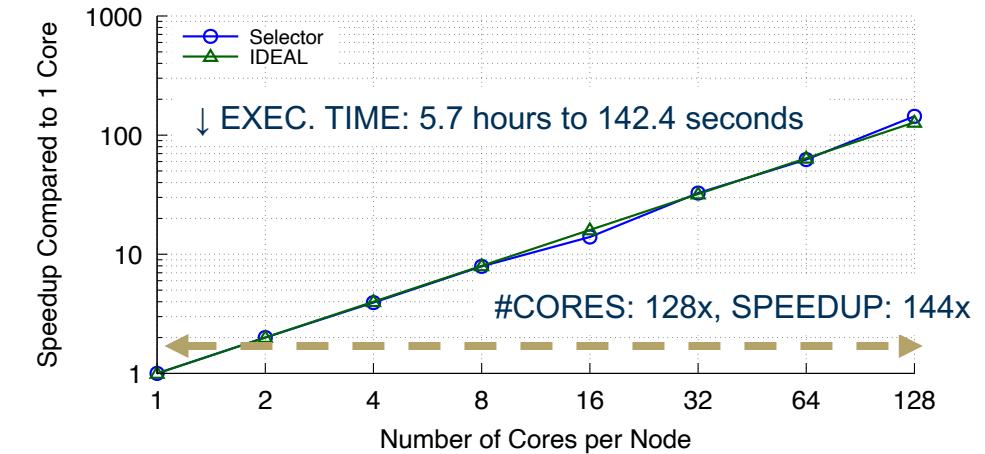
Performance Results: Scalability for SCALE - UP



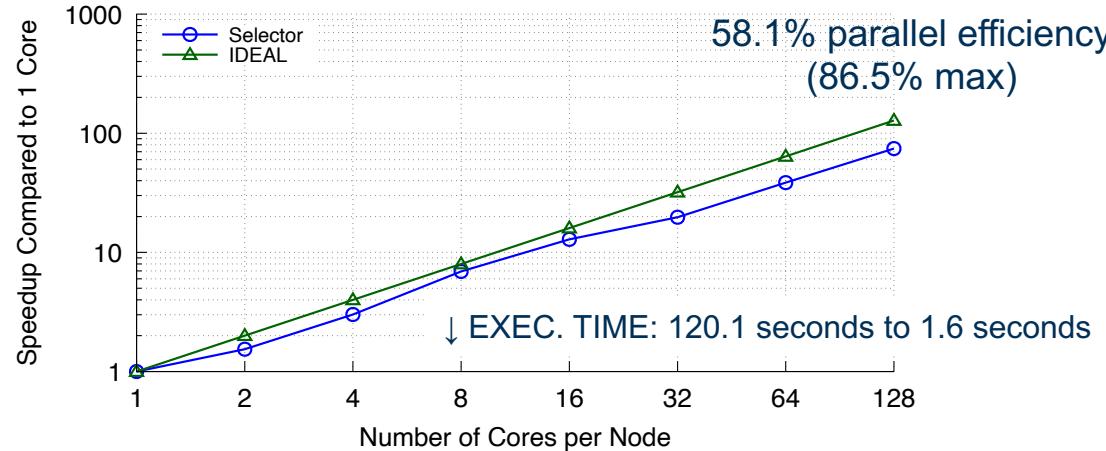
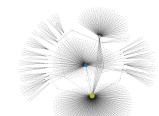
Routers Dataset
Strong Scaling



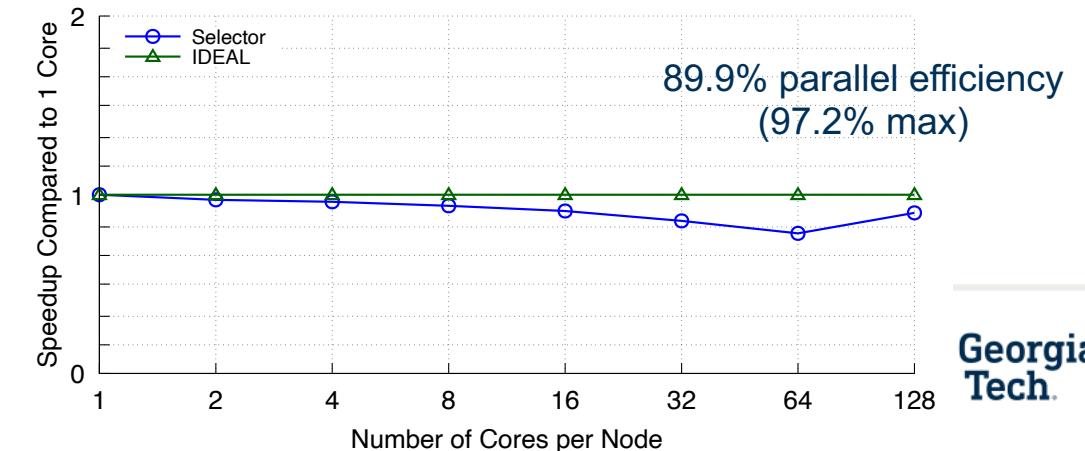
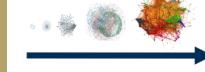
IPs Dataset
Strong Scaling



ASes Dataset
Strong Scaling



Synthetic Dataset
Weak Scaling

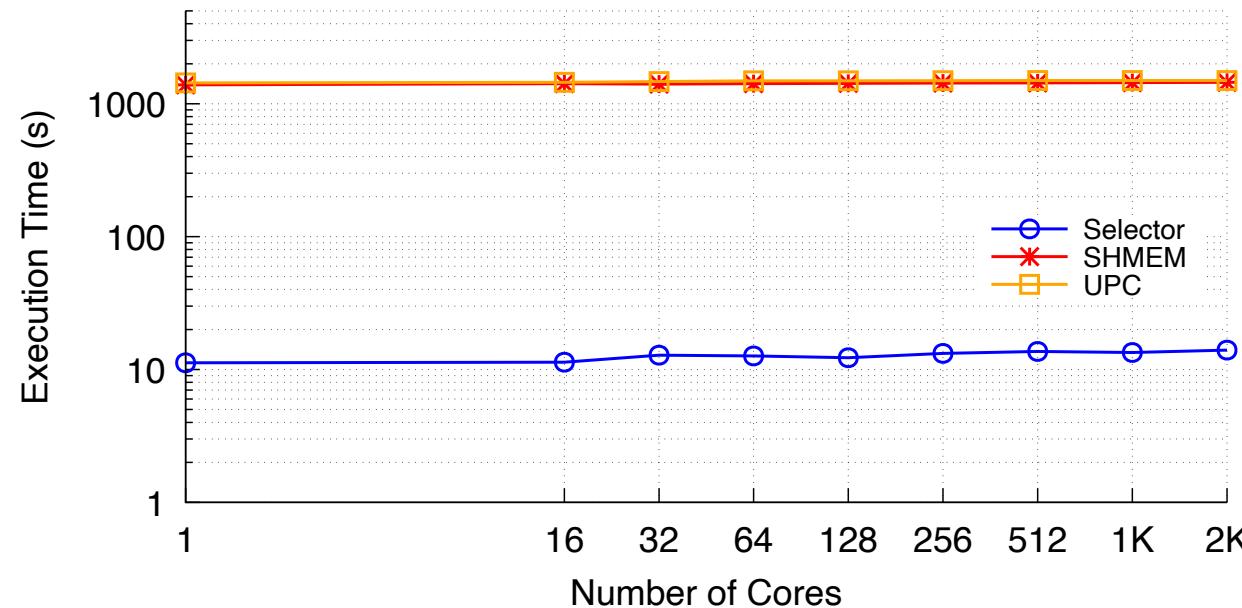


Contrasting to Related Approaches

- We contrast with respect to **performance on the large-scale synthetic dataset (weak scaling) in the SCALE-OUT dimension** on the Perlmutter Supercomputer
- Related PGAS approaches: OpenSHMEM, UPC

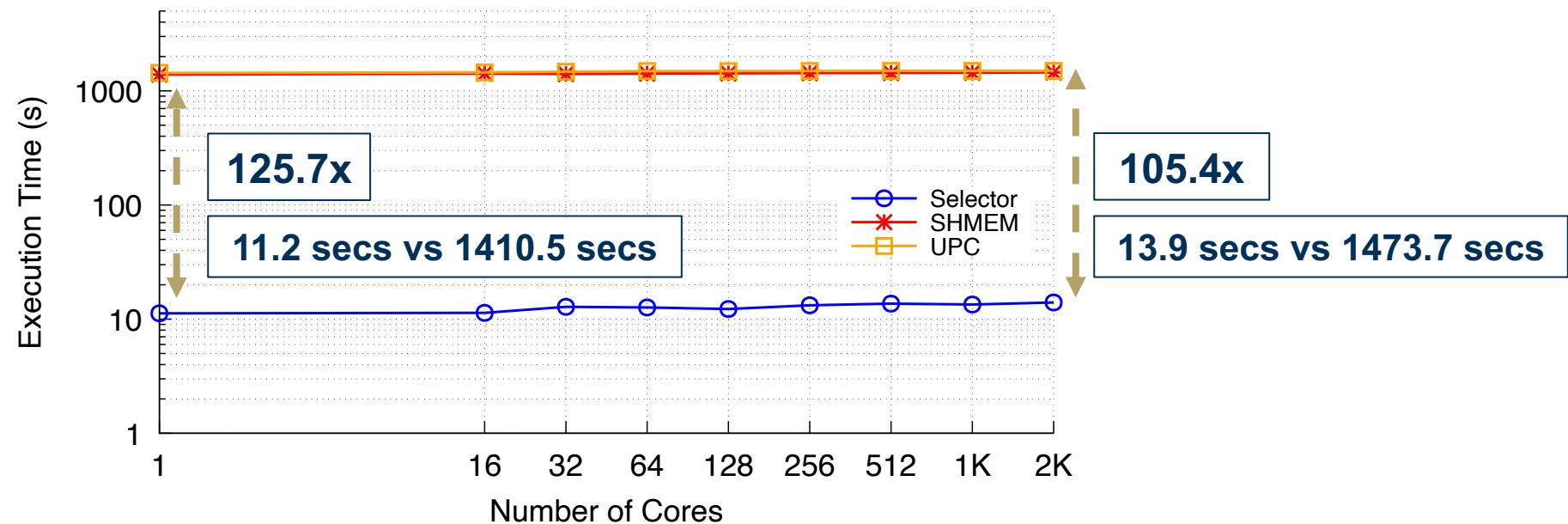
Contrasting to Related Approaches

- We contrast with respect to **performance on the large-scale synthetic dataset (weak scaling) in the SCALE-OUT dimension** on the Perlmutter Supercomputer
- Related PGAS approaches: OpenSHMEM, UPC



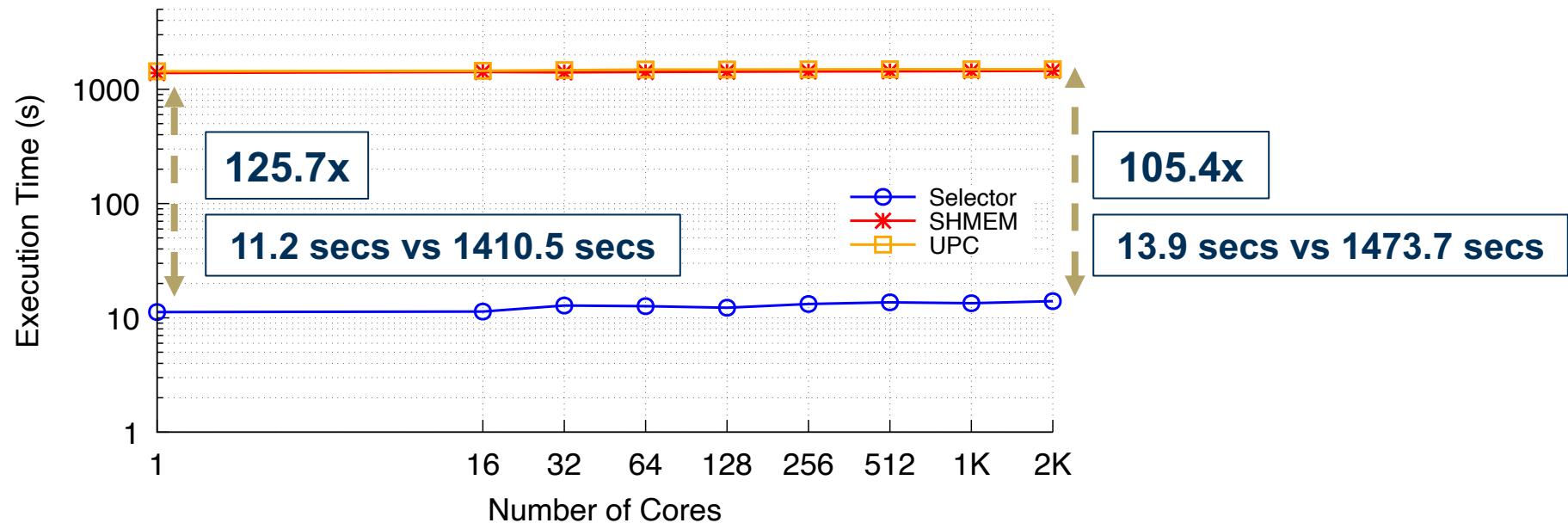
Contrasting to Related Approaches

- We contrast with respect to **performance on the large-scale synthetic dataset (weak scaling) in the SCALE-OUT dimension** on the Perlmutter Supercomputer
- Related PGAS approaches: OpenSHMEM, UPC



Contrasting to Related Approaches

- We contrast with respect to **performance on the large-scale synthetic dataset (weak scaling) in the SCALE-OUT dimension** on the Perlmutter Supercomputer
- Related PGAS approaches: OpenSHMEM, UPC



Attributed to the distributed and async. characteristics of our centrality algorithm as well as the efficient backend runtime

Impact of the Solution

- Our algorithm has shown efficient **scalability** and **performance**
- The extensibility of this algorithm can front five impacts:

Impact of the Solution

- Our algorithm has shown efficient **scalability** and **performance**
- The extensibility of this algorithm can front five impacts:

1

Our algorithm can be applied to network graphs of increasing size and complexity, as well as scaled to larger hardware resources

Impact of the Solution

- Our algorithm has shown efficient **scalability** and **performance**
- The extensibility of this algorithm can front five impacts:

1

Our algorithm can be applied to network graphs of increasing size and complexity, as well as scaled to larger hardware resources

2

Our algorithm allows for effective real-time decision making

Impact of the Solution

- Our algorithm has shown efficient **scalability** and **performance**
- The extensibility of this algorithm can front five impacts:

1

Our algorithm can be applied to network graphs of increasing size and complexity, as well as scaled to larger hardware resources

2

Our algorithm allows for effective real-time decision making

3

Our algorithm reduces network energy consumption due to classification of areas of high-traffic, congestion, and underutilized resources

Impact of the Solution

- Our algorithm has shown efficient **scalability** and **performance**
- The extensibility of this algorithm can front five impacts:

1

Our algorithm can be applied to network graphs of increasing size and complexity, as well as scaled to larger hardware resources

2

Our algorithm allows for effective real-time decision making

3

Our algorithm reduces network energy consumption due to classification of areas of high-traffic, congestion, and underutilized resources

4

Our algorithm can be applied to internet network topologies within global internet infrastructure, datacenters, cloud, and private networks

Impact of the Solution

- Our algorithm has shown efficient **scalability** and **performance**
- The extensibility of this algorithm can front five impacts:

1

Our algorithm can be applied to network graphs of increasing size and complexity, as well as scaled to larger hardware resources

2

Our algorithm allows for effective real-time decision making

3

Our algorithm reduces network energy consumption due to classification of areas of high-traffic, congestion, and underutilized resources

4

Our algorithm can be applied to internet network topologies within global internet infrastructure, datacenters, cloud, and private networks

5

Our algorithm can be extended to other domains

DEMO

A Distributed, Asynchronous Algorithm for Large-Scale Internet Network Topology Analysis

You can try this at home... Just visit hclib-actor.com !

Bulk Synchronous Parallel - HC x +

hclib-actor.com/background/bsp/

HClib-Actor Documentation Search hclib_actor

Table of contents

What is the bulk synchronous parallel model?

Single Program Multiple Data (SPMD) Programming

Further Readings

Bulk Synchronous Parallel

What is the bulk synchronous parallel model?

The Bulk Synchronous Parallel (BSP) model is one of the most popular parallel computation models.

The model consists of:

- A set of processor-memory pairs.
- A communication network that delivers messages in a point-to-point manner.
- Efficient barrier synchronization for all or a subset of the processes.

Diagram illustrating the BSP model:

- Virtual Processors:** PE₀, PE₁, PE₂, ...
- Local Computation:** Each processor performs local computation.
- SUPERSTEP:** A bracket indicates the duration of a superstep.
- Inter-processor Communications:** Arrows show communication between different processors.
- Barrier Synchronization:** A horizontal bar at the bottom represents the barrier synchronization step.

ACKNOWLEDGEMENT

This research is based upon work supported by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), through the Advanced Graphical Intelligence Logical Computing Environment (AGILE) research program, under Army Research Office (ARO) contract number W911NF22C0083. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the ODNI, IARPA, or the U.S. Government.

IEEE/ACM CCGRID 2024

The 24th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing

A Distributed, Asynchronous Algorithm for Large-Scale Internet Network Topology Analysis

Youssef Elmougy, Akihiro Hayashi, and Vivek Sarkar

Habanero Extreme Scale Software Research Lab
Georgia Institute of Technology

Thank you for your attention!