University of Nevada, Reno

Multi-Core Parallel Routing

A dissertation submitted in partial fulfillment of the

requirements for the degree of Doctor of Philosophy in

Computer Science and Engineering

by

Ahmet Soran

Dr. Murat Yuksel / Dissertation Advisor

May, 2017

THE GRADUATE SCHOOL

We recommend that the dissertation
prepared under our supervision by

AHMET SORAN

Entitled

Multi-Core Parallel Routing

be accepted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

Murat Yuksel, Ph.D., Advisor

Mehmet Hadi Gunes, Ph.D., Committee Member

Shamik Sengupta, Ph.D., Committee Member

Sergiu Dascalu, Ph.D., Committee Member

Gokhan Pekcan, Ph.D., Graduate School Representative

David W. Zeh, Ph. D., Dean, Graduate School

May, 2017

Abstract

The recent increase in the amount of data (i.e., big data) led to higher data volumes to be transferred and processed over the network. Also, over the last years, the deployment of multi-core routers has grown rapidly. However, such big data transfers are not leveraging the powerful multi-core routers to the extent possible, particularly in the key function of routing. Our main goal is to find a way so we can use these cores more effectively and efficiently in routing the big data transfers. In this dissertation, we propose a novel approach to parallelize data transfers by leveraging the multi-core CPUs in the routers. Legacy routing protocols, e.g. OSPF for intra-domain routing, send data from source to destination on a shortest single path. We describe an end-to-end method to distribute data optimally on flows by using multiple paths. We generate new virtual topology substrates from the underlying router topology and perform shortest path routing on each substrate. With this framework, even though calculating shortest paths could be done with well-known techniques such as OSPF's Dijkstra implementation, finding optimal substrates so as to maximize the aggregate throughput over multiple end-to-end paths is still an NP-hard problem. We focus our efforts on solving the problem and design heuristics for substrate generation from a given router topology. Our heuristics' interim goal is to generate substrates in such a way that the shortest path between a source-destination pair on each substrate minimally overlaps with each other. Once these substrates are determined, we assign each substrate to a core in routers and employ a multi-path transport protocol, like MPTCP, to perform end-to-end parallel transfers.

to Halil Soran and my family

Acknowledgments

I would like to start with the name of the One who gives me everything; GOD. Secondly, I want to thank the most important people in my life who always supported me and as I walked down new paths in my life journey; my family. In the first year of elementary school, my family moved out to a bigger city to ensure a better education for me. To them, I have to say more than 'thank you.' Hopefully, they will consider this dissertation as my gift in return for all that they did. By name they are: Munise Soran, Iffet Soran, Zeliha Soran, Serdar Soran, and Omer Batu Akcal. Also, I would like to thank Halil Soran for not only being my father; but also for being my life-coach. I want to express my gratitude to the dissertation committee and my advisor for their extreme patience in the face of numerous obstacles. Moreover, I would like to thank Cemal Elci for his material and spiritual support, rest in peace. Thanks to the Ministry of National Education of Turkey for their support.

I would like to thank all of my friends in this period of my life: Mehmet Dogan, Murat Demirbuken, Hicabi Bozkaya, and Ethem Coskun. They were always there to give me a hand. I would like to thank Turkish Cultural Association members. I would also like to thank Ibrahim Ethem Bagci for his overseas support and his sincere and relentless support. I am grateful to Esra Torlak, the best psychologist I have ever met, for all of her support, smile and motivation work. I also thank to Carol Souders and Dennis Ciceu for their kindness and hospitality.

I want to personally credit and sincerely acknowledge Dr. Mustafa Omer

Kilavuz for starting me on my journey to success. He convinced me to apply to UNR. And my first and the most influential teacher Zehra Erunsal is also held in high esteem for her initial belief in my potential way back in elementary school. I would like to thank all the staff and teachers Beytepe Elementary School for giving me a chance to build a better life.

And now may I move forward and note a special thank you to my most-admired colleagues of the UNR Computer Networking Lab - with a special mention to Esra Erdin, Mahmudur Kahn, Prasun Dey, Vahid Behzadan, Paulo Regis, Suman Bhunia, Jay Thom and Nate Thom. I would also like to thank TOBB Economics and Technology University for giving me an excellent education to demonstrate my skills and TOBB ETU Alumni for always keeping in touch with me to show their support. Most significantly, I would like to thank Dr. Bulent Tavli for mentoring me and always pushing me to do my best. Further, I want to thank the Guest Services staff of Athletics and Lawlor Event Center for being kind to me, and all others. Also, I want to thank all those people that smiled at me and supported me in this monumental journey.

Ahmet Soran

University of Nevada, Reno

May 2017

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The amount of data, “big data”, to be processed at computers and/or transferred across the Internet is growing 50 percent a year [57]. The concept of big data has been introduced in the early 2000s, especially in genetics research [87]. Big data research has become popular in computer science research more recently. The data needed to be managed has grown up rapidly, and managing or even classifying the big data became challenging. Although there is no particular threshold for classifying a dataset as “big”, largely speaking, big data are the datasets so large and complex that managing and processing (i.e., storage, search, sharing, analysis, transfer) them is legitimately challenging within the current capabilities of computing and networking technologies. Another key pattern that arose with big data is the difficulty of moving such large data around. From the networking perspective, data is “big” if it is so large that it cannot be moved to a centralized place but rather only be processed at different locations.

Statistically, the world’s technological capacity to store data per person is doubled every 40 months, and the telecommunication capacity is doubled every 34 months as well [38]. This storage and transfer capacities facilitate larger and big data to exist, intuitively with increasing amounts in future. Therefore, many techniques and discussions about managing big data have been proposed in different fields of study [41, 59]. According to the MGI studies [61], if management of the big data in the US health-care were to be effective, the sector would yield \$300 billion every year more than what it is today. On the other hand, big data provoked the emergence of new database management systems like parallel [24] and distributed [86] databases, and brought back batch processing systems such as Hadoop [90] with a customized computation phasing, i.e., MapReduce [23]. As big data kept emerging, the desire to centralize the processing of the data and transfer of it in more controlled settings have prevailed. Arguably, these have been the major causes for the advent of cloud computing and data center networking technologies. In parallel with those trends, another problem emerged: *How to effectively and efficiently transfer big data across and within data centers? Legacy networking protocols are not capable of handling transfers with speeds beyond 100Gbps.* New transport and routing protocols able to handle beyond petabytes of data are of crucial importance to furthering our capabilities to manage big data. The key points for cloud-based scalable data management systems are given as *"scalability, elasticity, fault-tolerance, self-manageability and ability to run on commodity hardware"* in [2]. Therefore, new data transfer protocols should not only be as fast as possible but also scalable, versatile, compliant with legacy systems, and flexible to adapt new schemes.

Data centers, arguably, are at the heart of big data management. Their design, placement, organization and structure play key role in how big data is going to be handled in the future [28]. Data centers can be located at various parts of the world, because of geographical issues, economic concerns or potential loss under failures. Accordingly, as geographically distributed data centers are gaining importance, the need for big data intra- and inter-datacenter transfers is more paining. More essentially, these big data transfers are crucial to the operation of the data centers for maintenance and backup [79]. Data centers are mostly built geo-distributed [63] because of cost efficiency, and data replication among two or more data centers is a much-used technique to improve the resilience against failures [1]. Also, most of the end-to-end (e2e) sessions in the Internet traffic are now going through a data center, and thus, the performance of "big data" intra- and inter-datacenter transfers is crucial to the overall Internet experience. Data migration techniques, virtualization methods or providing regional services to balance the workload over data center networks are imperative to reduce operational costs [20, 67]. For instance, transferring a high volume of bulk data causes outrageous workload on links between Yahoo! data centers causing peta-scale data to be moved around [19]. When there are multiple sources that need to send bulk data to another data center, managing these transfers becomes much harder, and scheduling is one of the solutions [49, 99].

The legacy end-to-end transfer techniques are not adequate to reach such speeds at the level of 100s of Gbps because of lack of performance on data connections over long-distance or high-bandwidth networks [8, 43, 64]. Apparently, under the scenario that uses a single path for data transfers, the aggregate end-to-end transfer rate will be limited to the bottleneck(s) on that path [45]. Parallel data transfers can be spread over the network in a non-overlapping manner, and hence improve the aggregate throughput [101]. The downside is that such parallel transfers require multi-path routing capability. Availability of such end-to-end paths allows parallel TCP streams to be fed onto different paths and thereby attain a higher utilization of the underlying network which is not possible by legacy single-path shortest-path routing algorithms [70, 97].

According to recent studies, the multi-path parallel streaming approaches seem to be highly successful in addressing the big data transfers. The key focus of these techniques is to diversify and spread the paths available to the end-to-end transport while satisfying various constraints such as delay or loss. Since the problem is too complex, most multi-path routing work boiled down to pre-computed techniques with heavy computations [62, 70]. Further, they typically involve non-shortest path calculations requiring considerable updates to legacy routers, which are not designed for non-shortest path routing. Yet, to attain multi-path routing, the routing system must be able to provide multiple paths for a source-destination pair. So, to address the big data transfer needs, we continue working on how to attain multiple and potentially non-shortest paths between endpoints in the network. Recently, the multi-path routing techniques proved to be useful for scaling up the end-to-end reliable transfers [66]. Still, a practical routing protocol that can offer multiple (non-)shortest paths while effectively handling network dynamics and failures is missing, since existing solutions are too compute-heavy and incapable of handling dynamism. Even under these circumstances, the paths generated by the multi-path routing methods were statically adopted and TCP sessions were successfully parallelized with practical solutions [9, 22].

One key observation is that these end-to-end transfers and the legacy multi-path routing schemes are still yet to utilize multi-core CPUs available in most routers. Although [95] shows that CPUs reach to almost 90 percent of usage under heavy data transfer scenarios, consideration of multiple cores of router CPUs as a first-class citizen in network layer functions, like routing, has been missing. To improve our multi-path routing service within “parallel routing”, we propose routing protocols that leverage the multi-core CPUs. Our goal is to ease the computational complexities of close-to-optimal multi-path routing algorithms *by dividing the overall multi-path routing problem into smaller parts and lending each part to one separate CPU core.*

We propose a "parallel routing" framework [92] that explicitly considers *multi-core routers* and employs only *shortest-path calculations.* The basic idea is to virtually slice the router topology into "substrate" topologies and assign them to a separate router core, which runs a classical shortest path routing protocol on the assigned substrate. We name this approach as "multi-core parallel routing" since it parallelizes the multi-path routing calculation and provides multiple paths for parallel end-to-end transfers by using multiple CPU cores. Rather than solving the multi-path routing problem all at once, our approach transforms it into two subproblems: (i) slicing out substrates from the router topology so that the collection of the shortest paths on each substrate is diverse and has non-overlapping end-to-end paths, and (ii) calculate shortest paths on each substrate. Since the latter problem is already being handled in legacy routers, our approach can easily be adapted to current routers if the former problem is solvable. In one point of view, our approach transforms the multi-path routing problem into a topology/substrate generation as a graph embedding/virtualization problem.

1.1 Parallel Routing: A Simplified Example

A simplified version of the parallel routing problem is illustrated in Figure 1.1. For this sample scenario, we consider only one end-to-end flow and aim to maximize its throughput. It is possible to generate different substrates, i.e. virtual topologies, by abstracting the underlying network multiple times. In Figure 1.1, three substrate

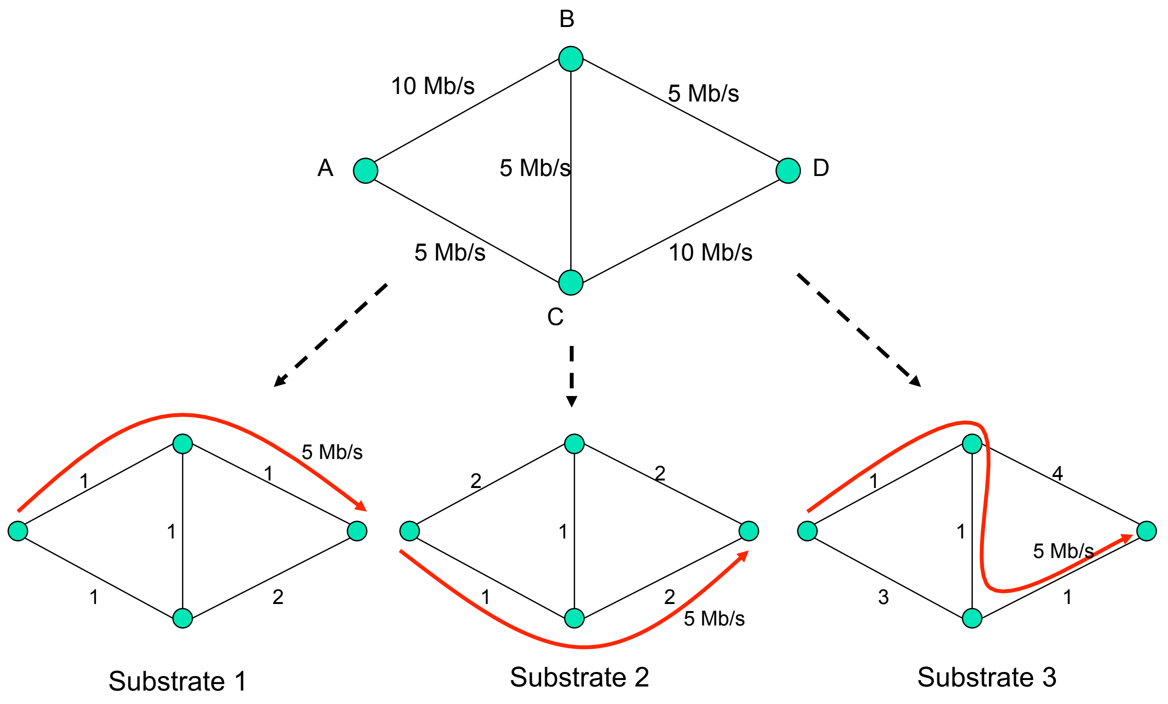


Figure 1.1: Parallel routing over multiple abstractions of the same underlying network.

networks are generated and different link weights are assigned to each substrate. Applying shortest-path routing yields different end-to-end paths from A to D. Each additional substrate adds 5Mb/s to the end-to-end throughput of the flow from A to D. This represents a linear speedup in routing parallelism. However, adding one more substrate would not add any extra throughput for the A-D flow under the same assumptions, showing the criticality of the number of substrates to be generated. Further, selecting the link weights for each substrate is not an easy task and plays a critical role in the efficiency of parallel routing as well.

When we consider a network with multiple flows attempting to maximize their throughput via parallel routing, the problem essentially becomes the well-known multicommodity flow problem [42]. However, our parallel routing approach here is taking the same network and divide it into multiple substrates, and thus allows us to tackle a

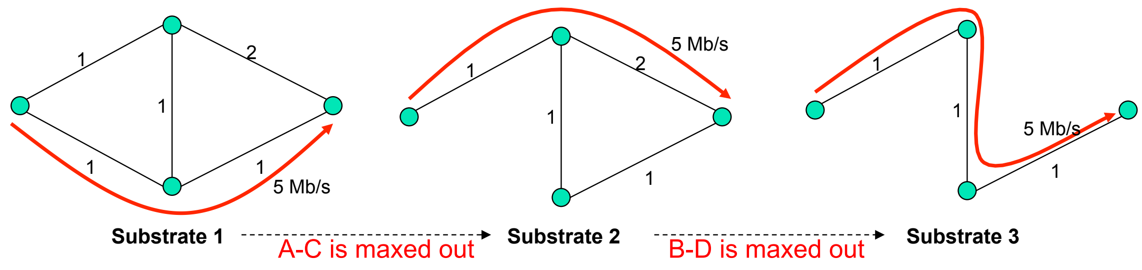


Figure 1.2: A sample heuristic for parallel routing: Remove the link maxed out by current substrates.

smaller version of the multicommodity flow problem. A key issue is how to generate these substrates (such as the number of them, their link weights) so that the end result of solving the multicommodity flow problem in each substrate is still close to the optimum.

Although our parallel routing approach divides the multi-path routing computation into smaller pieces and lends them to multi-core CPUs, it brings a new challenge to tackle: *How to establish overlay substrates so that a close-to-optimal multi-path routing is attained?* To address this issue, we will develop heuristics to construct and dynamically adapt the overlay substrates on the same underlying network. One possible heuristic is to iteratively increase the number of substrates and observe the aggregate throughput of the network. Before generating a new substrate, we can remove the links that are maxed out by the existing substrates. Figure 1.2 illustrates this heuristic for our sample scenario. It is not trivial to extend this heuristic to a general networking environment where cross-traffic and many concurrent flows exist. Substrates might become unnecessary due to changes in background traffic, or more substrates might be needed for the same reason. We will tune various parallelism parameters to establish the substrates, including *the number of substrates* for a given underlying network and *the link weights* in the generated substrates.

1.2 Contributions and Key Insights

In this dissertation, we propose Multi-Core Parallel Routing (MCPR) as a novel approach. The thesis offers the following key contributions and insights:

- Multi-Core Parallel Routing is a divide-and-conquer approach for the general multipath routing problem. It transforms the problem of calculating multiple paths to abstracting the underlying topology into multiple virtual topologies.
- MCPR actively uses the cores in CPUs to improve routing large data flows.
- MCPR is compliant with typical SDN setups in that the substrate generation can be done in the centralized control plane while shortest-path calculations for substrates can be done at the routers where data plane resides.
- MCPR leverages the existing shortest-path routing mechanisms, which are highly-optimized and ubiquitously available in legacy network routers.
- While updating legacy routers is possible, deploying a new multipath routing protocol to the legacy routers is challenging. MCPR is advantageous in this respect. It is easy to deploy with minimal changes to the existing routers which are highly optimized for shortest-path calculations.
- Recalculation of end-to-end paths in MCPR will be much easier in comparison to existing multipath routing schemes which try to solve the entire problem at once. Since MCPR deploys multiple shortest-path routing topologies in parallel, only shortest path recalculation will be necessary when a topological change takes place.

- MCPR will increase the robustness of routing against link or node failures. It will enable multiple simultaneous paths and only a subset of the substrates will recalculate due to failed link or node. This reduces the disruptions to the routing tables and the ongoing high-volume traffic.

- MCPR attains higher throughput for large data transfers and better performance in balancing the load over the network. In particular, during our static analyses of comparing MCPR heuristics against single shortest-path routing in Rocketfuel topologies, we observed 1.6 times speedup in the aggregate throughput when 4-core routers are used. This speedup was minimally affected (i.e., reduced to 1.5) when the count of cores was heterogeneous across the routers.

- MCPR offers robust and low-cost solution for multi-path calculation that can be easily adapted to the current systems and performs well on different topology environments. MCPR also responses well to the network failures.

- MCPR is minimally affected when the network failures, both node and edge failures, occur. In our static analysis, we observed that MCPR's performance may reduce up to 0.4% in the face of failures. Also, MCPR responses better when edge failure occurs. We also observed that edge failures may contribute to MCPR performance, and it might give higher throughput up to 0.2% in some cases.

- MCPR supports scalability when network size is growing because of its adative background.

1.3 Dissertation Organization

In this dissertation, we propose a new end-to-end traffic method, which is exploiting multiple CPU cores of routers to parallelize traffic flows, especially for bulk data transfer such as between data centers. The rest of the dissertation is organized as follows: We start with the background information in Chapter 2 and give details about some of the popular routing protocols. Multi-path routing techniques, Multi-path TCP and some successful applications employing multiple cores are also explained here. We outline the problem of parallel routing and mathematical background for parallel routing in Chapter 3. We present multi-core parallel routing heuristics to solve the substrate generating problem and the results of our simulation experiments in Chapter 4. We give how multi-core parallel routing performs when a failure occurs on the network in Chapter 4.5. We show the performance of parallel routing under the heterogeneous scenarios in Chapter 5. We discuss our proposed framework and future works in Chapter 6.

Chapter 2

Background

Assuming multiple paths are provided between end-systems, the task of *selecting a subset of the available paths* and performing *load balancing of traffic among them* is a highly complicated problem, which is also known as end-to-end traffic engineering. Such end-to-end traffic engineering capability has been a common practice in wire-line networking via protocols like MPLS [85] which provides ISPs a way of managing and throttling their traffic over the network. However, the time-scale of this end-to-end traffic engineering practice has been very large since MPLS requires configuration and management of several switches and routers to realize path establishment and teardown. Further, similar to the source routing in GridFTP, the load balancing of the overall network necessitates pre-computation of MPLS paths (i.e. LSPs) ahead of time as there will be many such end-to-end traffic flows. In general, Layer 2 (link layer) techniques, like MPLS, are external to routing in Layer 3 (i.e., network layer) and require heavy configuration tasks in a large-scale network to be operational. Due to the static and heavy configuration overhead of these Layer 2 techniques, they fall short of addressing network dynamics arising from failures or traffic spikes and coming up with a generalized end-to-end traffic engineering framework. Further, in the face of recent needs for big data transfers, a generalized solution within network layer is becoming a necessity, particularly within or across data centers.

Most of the current routing and transport protocols employ single path, such as in OSPF [65], RIP [60], and TCP [78]. In the early 2000s, discussions on single path routing effectiveness intensified in the context of congestion and throughput achievements needed for emerging big data network transfers. Because of the poor performance of legacy routing and transport protocols on load balancing and congestion control, it has been a consensus that usage of multiple end-to-end paths is needed to attain sustainable transport rates beyond 100Gbps.

Although single path solutions are fast to compute and apply in the real world, one of the important reasons for congestion and data loss has been the single shortest path routing approach of the legacy routing protocols. Therefore, multi-path routing [31, 36, 44, 97] became one of the popular topics as a solution to balance the load over the network. Several techniques, most of which require a-priori computation of paths, have been proposed for multi-path routing. Diversification of paths from source to destination with the additive increase methods has been proposed [69]. Additionally, a path selection method [70] is also worked on to find paths from the set of all possible paths. However, the routing system needs the ability to generate all paths and select a few of them in a short period. Moreover, some researchers tried to compute end-to-end multi-paths like in network flow approach or shortest path with the selection possibility of links [72]. With all that, the real drawbacks of multi-path routing techniques have been (i) the adaptation to the actual systems because of high computation costs and (ii) the un-availability of network infrastructure like routers and switches that allow sending data through more than one path by using current protocols. Eventually, the temporary solution, improving link bandwidths, curtailed the cost of congestion over the network and postponed the problem of multi-path routing until a better and permanent solution for data transfers is found.

Within ten years, some new applications for solving multi-path routing problem have been evaluated with new proposed protocols to support multiple connections between a pair of nodes. However, with growing Internet, some protocols have been standardized, like IP [77] and TCP, and adaptation of new protocols to the current systems has become the biggest problem. For example, changing IPv4 to IPv6 has been taking a much longer time than expected, even though IPv6 has been deployed for about two decades due to lack of IPv4 addresses for new systems. Accordingly, new end-to-end multiple path protocols should be TCP-friendly, and Internet Engineering Task Force (IETF) teams generated Multi-Path TCP (MPTCP) [22, 26, 29], which improves TCP and uses almost the same structure. Briefly, MPTCP is a high-level design which encapsulates sub-flows into TCP packets with a fair congestion control mechanism. MPTCP assumes no knowledge of the underlying topology and focuses on congestion control. Though this is important in realizing a generic multi-path congestion control approach, most of the time situations involving extremely large data transfers are already given a topology of end-systems which can be leveraged for jointly optimizing the selection of end-to-end paths as well as throttling of sending rates on sub-flows.

In the case of failures, one of the fundamental problems for routing protocols is to re-calculate optimized paths while recovering the ongoing data transfer in a timely manner. Calculation and provisioning of multiple paths to the end-systems has been done in various tools. A widely used protocol, as part of the science project Large Hadron Collider [56] for data communications between super computers, is stripped

GridFTP [5], which allows source routing. However, like most of the existing multi-path routing protocols, GridFTP requires all routing calculations to be done a-priori and is not sufficiently adaptive to dynamism and changes in the underlying topology. For example, a link failure or link cost change will trigger recalculation of the complete multi-path routes, which incurs a non-polynomial computational complexity if certain guarantees are desired [32, 97]. Further, this recalculation will have to be performed at a central location so that one can install the new paths to the GridFTP source routing module.

The capability of end-to-end traffic engineering has become essential in *finer time-scales* to achieve real-time load balancing of large inter-data-center traffic flows [52] coupled with the dynamism of the underlying Internet connectivity. Another solution, SPAIN [66], for inter-datacenter traffics is to use pre-computed paths for utilizing redundancy in a network, especially under the high bandwidth data traffic. However, these two techniques fix multiple end-to-end problem with side-channel solutions (i) scheduling or (ii) using offline network controllers. To have adaptive solutions, proposed method should be either scalable or flexible and able to balance workload over the network without big disruptions.

Ultimately, most of the previous solutions for multi-path routing are based on pre-computed techniques with large computation cost. We propose multi-path end-to-end traffic routing method using well-known shortest path calculations. Moreover, our approach exploits CPU cores included in multi-core routers more effectively and efficiently routes big data transfers. Since it uses legacy shortest-path routing as the basic building block, our method is easy to deploy and easily adapts to current routing systems.

The term "network" is used for several descriptions as in social network, biological network, telecommunications network, and neural network. Basically, a network is a set of elements interconnected with each other. In information technology, “network meant the set of serial lines used to attach dumb terminals to mainframe computers” as described by Peterson in “Computer Networks: A Systems Approach” [75]. Different from other kinds of networks like telephone systems, the networks in computer science are built for a general purpose. Therefore, computer networks can carry different types of data and support various applications to make the system more scalable and fast growing. In mathematics, a network is a sub-category or specialized type of a mathematical graph representation of the set of elements and connections between them. Computer networks are directed graphs built by nodes (e.g., computers, routers or other network elements) and arcs/links (i.e.,physical channel between two computers).

According to the term of “network”, we can name the Internet as a “network of networks” built by connected devices. However, physical channel can establish a connection between any pair of elements. Therefore, those elements have to use a common guideline for speaking, namely protocols, to define the general rules of communication steps. Current systems use two different communication approaches: (i) circuit switching and (ii) packet switching. In our work, we focus on “packet switching” approach which forwards data from a node to another node. This method improves the scalability of the network, compared to circuit switching as paths are not reserved for individual flows but shared. In packet switching, however, forwarding incoming packets towards the destination incurs overhead as for each packet routes needs to determine the next hop. Thus, switches are used to forward packets directly to the next hop as fast as possible. However, switches are limited in addressing the routing problem in large networks as they are not scalable that employs routing

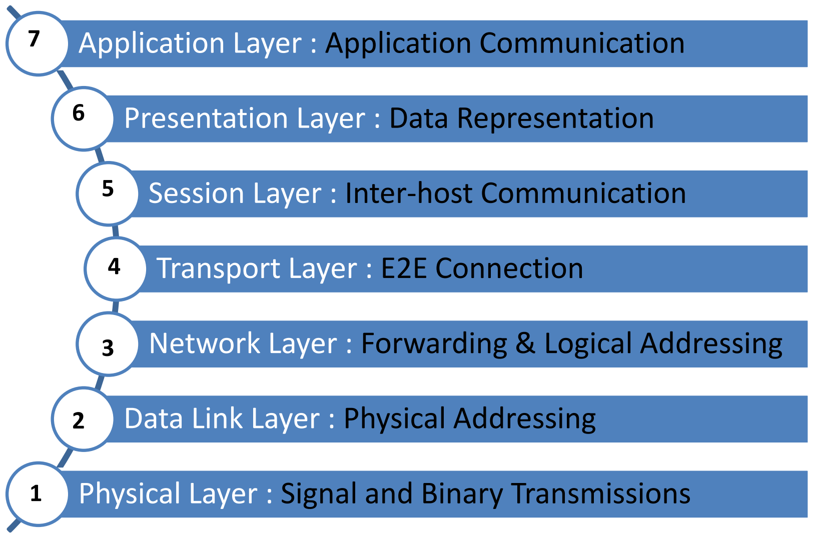

7 Application Layer : Application Communication
6 Presentation Layer : Data Representation
5 Session Layer : Inter-host Communication
4 Transport Layer : E2E Connection
3 Network Layer : Forwarding & Logical Addressing
2 Data Link Layer : Physical Addressing
1 Physical Layer : Signal and Binary Transmissions

Figure 2.1: Open Systems Interconnections conceptual reference model

problem.

2.1 Networking Basics and Protocols

To solve the problem of route generation/calculation, essentially all nodes in the network act as a 'router' and participate in a larger but distributed computation of paths. They calculate the end-to-end paths by sharing information with each other. Once the end-to-end paths are calculated, all routers will have a routing/forwarding table which shows the next node for an incoming packet according to the destination of that data packet. There are two protocol approaches to calculate these routing tables: (i) link-state (e.g., OSPF [65], IS-IS [91]) and (ii) distance vector protocols (e.g., RIP [60], BGP [82]). The basic difference between those two methods is how they make the routers communicate and organize with each other. The link-state protocols make every router exchange information with all other nodes in the network so that each node can have information about the links of the whole network. On the other hand, in the distance vector protocols, the routers collect and store local information about their neighbors and share their knowledge with nearby nodes. While the former approach collects the cost information about the links of the network, the latter gathers the distance to every other node in the network. In this thesis, we focus on link-state protocols, because our proposed system is based on knowing the entire map.

ISO (International Organization for Standardization) standardized a conceptual reference model to separate basic network processes by creating abstract layers over networks in 1994. That OSI (Open Systems Interconnections) [81] model contains seven layers from low-level, physical architecture, to high-level, applications as shown in 2.1. Briefly, physical layer, Layer 1, involves with devices and physical connections between these devices at a machine level. Then, link layer, Layer 2, manages reliability on data-links with controlling packet synchronization. Following Layer 3 is network layer that provides the appropriate environment to perform end-to-end data transfers. Beyond that, transport layer, Layer 4, yields reliable delivery during data transfer sessions. Also, session layer, Layer 5, organizes sessions that allow the communication between the application and transportation layers. Presentation layer, Layer 6, manages data conversions and securities before the application. Lastly, application layer, Layer 6, is the end user layer of OSI reference model. During the data transferring period data processes is shown in Figure 2.2. Every step that data reaches a router, it follows high-level to low-level conversion and while forwarding that data it follows low-level to high-level processes.

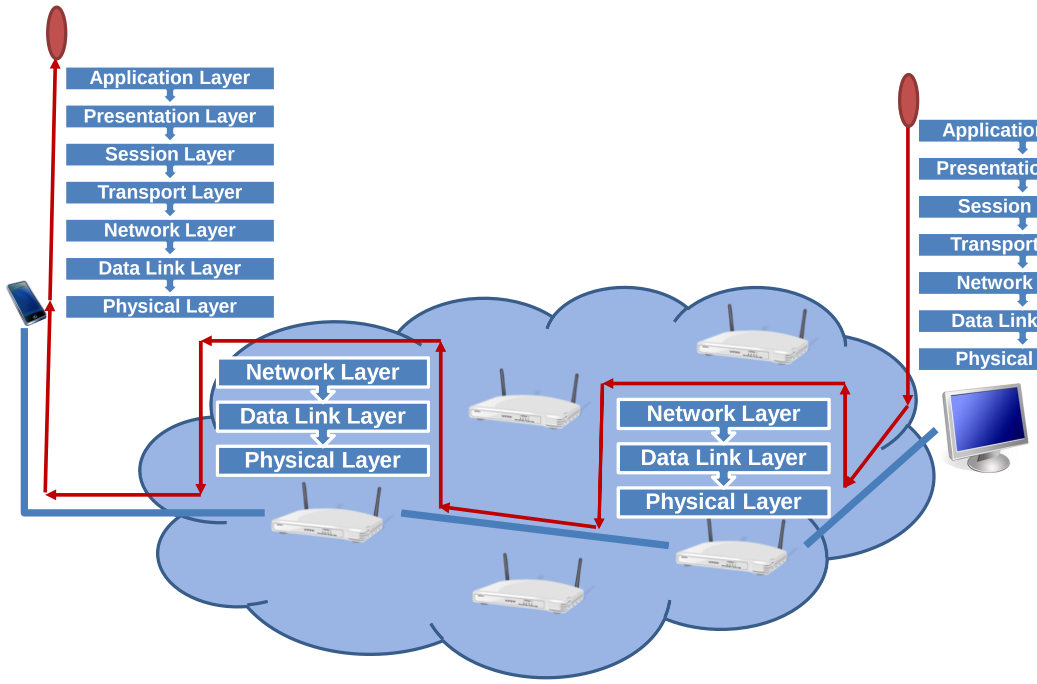


Figure 2.2: Encapsulation during data connection

2.1.1 Link State Routing and Link Layer Support for Traffic Engineering

MPLS (Multi-protocol Label Switching) [84] is a Layer 2 protocol, but it works in between Layer 2 and Layer 3 which controls end-to-end traffic management on switch level by using packet information. It supports not only IP packets but also other network layer protocols such as ATM (Asynchronous Transfer Mode) [54]. However, it extends the current routing protocols' abilities by diversifying end-to-end path selections for connectionless networks. The main purpose of MPLS is to create a virtual end-to-end connection that data can flow through without network layer (i.e., IP level) changes. It is used for two primary purposes: (i) establishing backup paths for each link and (ii) load balancing on an end-to-end basis. By using label switched paths (LSPs), MPLS allows us to configure a backup path for each link so that the traffic is immediately (i.e., in 50 ms) rerouted to a backup path when a link fails. Further, and more relevant to this thesis, LSPs can be used to establish to end-to-end non-shortest-path routes so that traffic can be routed over paths that don't follow typical shortest-path. The purpose of this is to achieve a more load-balanced network. One of the alternative methods to control these label switched paths is that changing OSPF [65] weights as in [15]. Changing Open Shortest Path First (OSPF) weights can be done within the network layer without necessarily requiring a Layer 2 protocol like MPLS. However, one of the well-known problem for changing OSPF link weights is that it may lead to persistent oscillations (i.e., route flaps) between two different end-to-end paths if the links weights are associated with the utilization of the links, which is the usual practice.

OSPF [65] is one of the link-state routing protocols for IP networks that make it possible to compose routing tables to forward datagram packets when it reaches a router. OSPF is mostly used for organizing a network within an autonomous system (AS) or intra-data-center networks. As it is named, OSPF merely provides single path solutions and uses Dijkstra's shortest path algorithm to generate a possible path for a pair of nodes. Therefore, many applications are proposed to change link weights for getting a different path to balance the load on the network. The main advantage of using OSPF is that adapting failure scenarios as fast as possible because of OSPF's collection algorithms of routing information. It detects network breakdowns for link-state routing/link-state update message combinations among the routers and creates a new loop-free path within seconds. Therefore, we propose a new method that maintains the benefits of OSPF for failure scenarios. Our approach uses OSPF as the routing protocol for each substrate running in parallel.

2.1.2 Internet Protocol: End-to-End Design

IP, Internet Protocol, is a network layer protocol designed, in 1981, for packet-switched communication networks which allows to divide data into small packets and organize the addressing of network elements [77]. The key point of IP is to allow a proper environment for inter-connected devices to communicate with each other even if they may be using various kind of different communication protocols among them in a manner that is entirely end-to-end, as discussed in [88].

IP has two principal works: (i) fragmentation and (ii) addressing to point standard rules for interpreting addresses and creating datagrams, which are small packets carrying data. Thus, data can be transferred in a connectionless manner by using datagrams which are independent entities. However, splitting a given data into little packages and reassembling divided packets in a system became another problem. Therefore, various fragmentation methods have been proposed for helping to improve data communications. Although fragmentation process takes time, it improves the system performance during data transfer because of less cost on carrying out a small size data over the network.

On the other hand, to have an end-to-end agreement, nodes should know each other in a uniquely identifiable way which necessitates addresses. So, IP includes destination addresses into every packet after fragmentation procedure and those packets can be analyzed by routers to figure out the final destination of them at each hop. Many network address mapping techniques have been proposed to organize addressing as well. As a result, IP not only creates a systematic environment to manage packet-switched data transfers but also gives the opportunity to generate new transport layer protocols which might be adapted to the current systems because of IP's encapsulation ability.

2.1.3 Transmission Control Protocol

As a connectionless protocol, IP does not have the ability to guarantee reliable communications between source-destination pairs and needs application-specific protocols to manage flow paths among network elements. On the other hand, TCP [78], one of the connection-oriented transport (i.e., Layer 4) protocols, establishes a reliable communication pipe between the source and the destination while supporting packet-switched end-to-end connections. TCP uses sockets [18] to interact with devices and selects a route from the IP layer for data to be sent through. These sockets use specific ports in addition to network address while reaching a device. So, the devices can be used by several connections simultaneously at any time which shows that the devices can also work on reliability issues during data transfers. Therefore, TCP works in a manner similar mail posting by sending and receiving letters indicating the intent of connections. After connection establishes, data transfer process is started on the agreed ports between two the devices.

TCP connection establishment based on three-way handshaking is given in Figure 2.3. There are two nodes one is the server which is the destination, and the other one is sender device called client. The server always in *Listen* mode to gather any connection requests coming from a sender node. Therefore, the client node, to get a connection with the server node, send a special **SYN** packet which shows the intention of communicating. When the server node gets the connection request, it produces a new value based on coming data and replies that request with a specific **SYN-ACK** packet showing the availability for the establishment. Then, when that packet received by the client node, the last step of handshaking is done by sending **ACK** packet which is setting an agreed number that provides the reliability during the data transferring. After connection installation, these two nodes can start

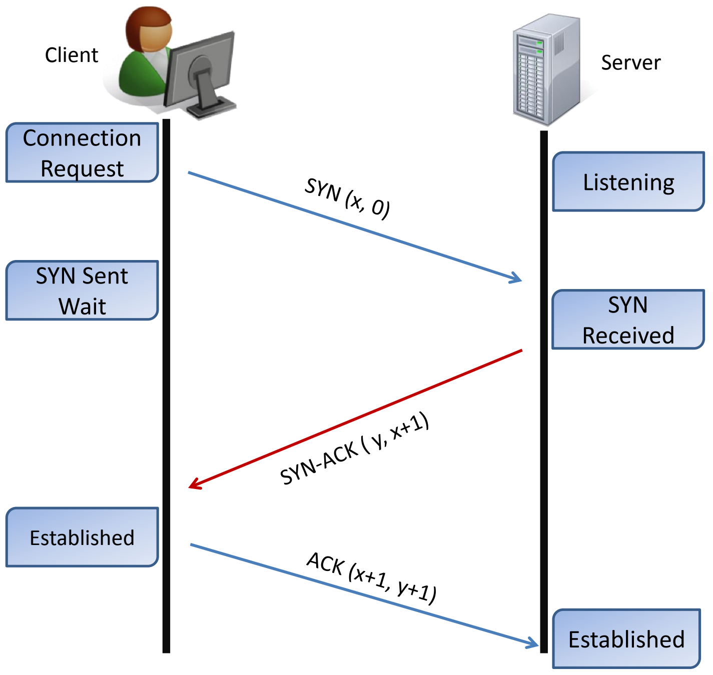


Figure 2.3: TCP 3-way handshaking for connection initialization

to communicate with each other within for each new packets agreed the number is changed based on the agreement. Therefore when a packet is transmitted, the server is able to verify the received data in a manner of reliability by checking that special segment number.

Although that three-way handshaking method provides the connection-oriented reliable end-to-end data connections, in this technique, multi-path solutions are not supported for fair connection between a source/destination pair. Therefore, there are some works on trying to evolve TCP for supporting multi-path communications by creating different TCP sessions among the nodes [7]. However, adding new TCP sessions only improves the data amount which can be transmitted if path diversification is not executed.

2.1.4 Multi-Path TCP

Internet Engineering Task Force (IETF) teams have been working to standardize Multi-Path TCP [29], which is a new TCP-friendly protocol that allows multi-path solutions for end-to-end traffic engineering. As we discussed before, although TCP has a significant performance to solve the end-to-end path connectivity problems, the lack of ability to support for multiple paths is one of the bottlenecks for the current Internet routing. Because of the emerging big data issues, load balancing and congestion control aspects of the Internet need to be improved. Therefore, Multi-Path TCP [22] has been proposed as an approach to perform such real-time practice of multi-path routing, but, it assumes no knowledge of the underlying topology and focuses on congestion control with a little modification on the baseline TCP scheme.

In addition to initial connection setup of TCP settings, Multi-Path TCP also sets up additional sub-flows under the package of the main flow. Each sub-flow is an individual path between a pair of nodes acting as a regular TCP connection as shown in Figure 2.4. In contrary to creating new end-to-end TCP sessions, encapsulating these sub-flows with a new scheme is also optimizing fairness by the congestion control method. Therefore, instead of creating separate TCP flows, the TCP streams are packaged in one big larger Multi-Path TCP connection. However, this congestion control mechanism provokes the increase of control plane traffic over the network.

In parallel with being standardized, new applications based on Multi-Path TCP has been proposed expeditiously in various areas such as wireless networks [73], energy efficiency [76], network management [92], and open flow architectures [96]. Following this trend, we also propose a new technique using Multi-Path TCP architecture to optimize inter/intra -datacenter traffic with leveraging multi-core routers to parallelize path selection mechanism for end-to-end paths. We would like to note, however, that

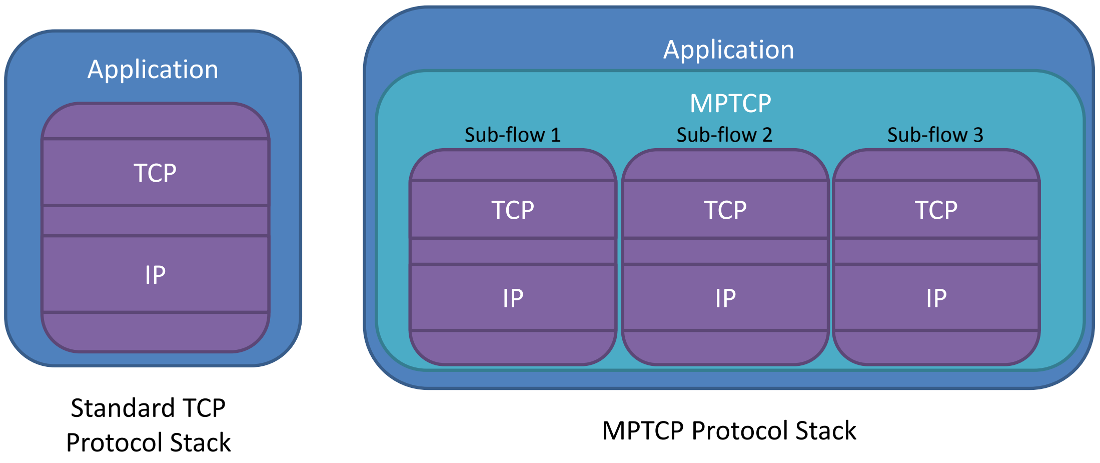


Figure 2.4: Comparison of TCP and MPTCP protocol stacks

our parallel routing approach can work with other multi-path transport protocols as well.

2.2 Multi-path Routing

In addition to MPTCP, in recent years, there has been significant interest in multi-path routing [40, 66, 100, 104]. As we discussed before, multi-path routing is one of the old problems in the networking research community. In the 1990s, an analysis of shortest path routing in dynamic network environments [103] classified routing algorithms based on how adaptive they are as static, quasi-static and dynamic. Then, it showed that single-path solutions limit the maximum flow between the source and the destination nodes. These years, additively increased, a.k.a. incremental, path calculations became one of the popular solutions as in congested-oriented multi-path routing [68], disjoint path computation [71] or loop-free multi-path routing [102]. In the 2000s, multi-path routing was adapted to wireless networks and an alternate path routing scheme [74] was proposed for MANET environments, and 40% improvement was shown in end-to-end traffic delay. Another method [55] proposed to split multi-path routing problem into establishment of more than one paths between pairs in ad hoc networks.

The well-known multi-path routing problem is Maximum Flow/Network Flow problem which aims to send as much data as possible from one source to a destination, without worrying about the end-to-end delay. Extension of this problem to real networks is more complex and includes multiple source-destination pairs for sending data at a given time. This extended version is called Multi-Commodity Flow problem [42]. In a given graph, there are nodes/devices and edges/links as network elements, and each link has a cost if it is used. Basically, shortest path means minimizing the total cost and maximizing data sent between a pair of nodes. Theoretical solutions to the maximum flow problems by mathematicians, but, these solutions are typically not able to be run in real-time due to their intractability. So, instead of finding the best result, researchers aim to figure out a solution which is proximate to the optimum solution and as fast as possible in terms of gaining higher throughput. As a result, the problem is reduced to selecting best paths between a pair of nodes from the all possible path sets [70]. In some solutions, links' weights are changed to diversify paths for end-to-end transfers [84], however, that can cause persistent oscillations between two different end-to-end paths. Therefore, some additional metrics to point costs such as possibility or propagation delay to be chosen are assigned to links in some techniques [51]. But, finding better possibilities and managing the entire links a network, particularly under *network dynamism*, becomes a big problem waiting to be solved for these methods. Another improvement on end-to-end path calculations is finding k-shortest paths as in additively increased methods. However, in that case, the system should find some different paths which are not the shortest paths. So, similarly, some links' weight must be changed, and additional computation costs will be needed. To sum up, there could be two different solution approaches for multi-path routing: completely disjoint sets or some intersections/overlaps between the paths.

As powerful computers have been grown in a parallel with technological developments, finding a new end-to-end multi-path traffic engineering becomes a popular instead of using adaptive techniques to get multiple paths between a pair of nodes. Therefore, such systems, SPAIN [66] and Net-Stitcher [52], have been proposed for improving the throughput performance of inter-datacenter networks while satisfying the reliability of the data transfers. On the other hand, mPath [100] solves multi-path routing problem by using a new TCP-friendly mechanism which finds other possible paths via a set of proxies that give a one-hop detour within the end-to-end paths.

In addition to the key metrics of increasing throughput and balancing load the traffic load on the network in an end-to-end manner, there are other important applications of multi-path routing for reducing delays or controlling congestion. One of the relevant directions is to use multi-path routing in large-scale circuit design. Parallel routing approaches have been proposed in LSI (Large Scale Integration) [94] circuit design. Y. Shintani et.al. [89] use a multi-threaded approach on multi-core processors to find paths for global routing between net-lists which are connections of cells on LSI circuit design. Proposed method creates different blocks for the net-lists and distributes those blocks to the threads, so, each block can be processed in parallel. With their parallel routing method, new technique performs 7.1 times faster than the sequential method.

Multi-path routing is also be a part of a solution to avoid traffic congestion in communication networks such as Software-Defined Networks(SDN) for inter-datacenter networking. Briefly, in SDNs, controllers manages the traffic patterns on the network and configures data-plane communications among the devices. However, control plane messages, communication data between devices for controlling the network, can cause temporary congestion on network elements in software defined networks (e.g., in SD-WANs). Chi-Yao Hong et. al. [40] proposed a new approach which is using multi-path routing to manage capacities on links efficiently for sending these update messages. Then, with this improvement, 60 percent more traffic can be carried than the current system.

2.3 Bulk Data Transfer in Data Center Networking

Bulk data transfers has reached peta scales, and it currently dominates the inter-datacenter traffic. In ACM SIGCOMM 2011 conference, bulk data transfer for data centers was announced as one of the sessions. It shows the importance of the problem as it became very popular in recent years, and multi-path routing can be used to solve the bulk data transfer problem on data center networking applications. In the case of multiple bulk data transfers, the system needs to solve complex problems for deciding paths for sending the bulk data by avoiding congestion. Some of the current solutions include extreme methods like shipping the data in large disks and enlarging or dedicating link capacity between data centers. Therefore, improving the performance of transferring bulk data via the public Internet has been of much interested.

Scheduling techniques that push more data through the Internet for such bulk transfers have attracted interest [53, 99].Scheduling to avoid congestion during multiple bulk data transfers has been of crucial interest too [99]. Wang et al. first observed traffic patterns which they found to be strongly diurnal. The problem, however, is the time differences between data-centers. When one data-center is active due to the morning hours on the East Coast, the other one on the West Coast could be in idle condition because it is not morning the West Coast yet. Thus, the goal is to split the bulk data into blocks and transfer them over multiple multi-hop paths to attain a balanced load across the data centers. Lexicographical minimization, minimizing the traffic of the maximally loaded link and attempts to minimize the traffic of the second maximally loaded link have also been tried in this work.

In addition to scheduling, there have been several other proposals based on improving the underlying infrastructure. For example, in data center networks, shallow-buffered switches are used for decreasing the cost. However, under the congested scenarios, the amount of dropped packets will be higher since the queueing systems at the routers drop packets automatically when the buffer is full. Calder et al. [16] changed paths for those dropped packets instead of losing them. In these approaches, the entire system (which might involve multiple data centers) is under the single administrative control; so, switches can share their buffer conditions with each other. With this type of information sharing, it becomes possible to monitor the whole traffic and detour paths for the packets that would be dropped otherwise. Although this approach could preserve packets, it could cause congestion over the network.

Another work has been done on managing workloads on data centers under the transport fabric [4]. In some applications, as in social networking, short request-response flows which are relevant to each other could be demanded with small response time. In that case, requests are collected with a fraction and responded to the user together. According to the latency metric, one of the methods used is TCP-fabric which is not effective because of waiting time in queues. Therefore, instead of avoiding congestion, pFabric [4] proposed a new scheduling method to balance the load over the network and to improve the performance of fabric transport.

Consequently, the latest research shows that bulk/big data transfers are gaining more importance with the boost of the number and size of data centers on the Internet. Therefore, bulk data transfers is one of the big issues for today and the future Internet and present several specific research challenges to be tackled. In order to solve managing big data transfer issues, all resources will need to be used efficiently; and, in our work, we aim to use multiple cores that already exist in the routers more effectively to offer and improve multi-path routing for the Internet and for the data center networks in particular.

2.4 Multi-Core Protocols

The deployment of multi-core routers has grown rapidly. However, big data transfers are not leveraging the powerful multi-core routers to the extent possible, particularly in the key function of routing. We aim to revise multi-core or multi-threaded solutions for improving data communications of inter-/intra-datacenter traffic. The emerging need for greener energy and greener data centers [] also motivate us to optimize available networking and computing resources. It is now a major challenge to design information technology solutions that are green and use the resources in the best manner possible. Thus, leveraging multi-core devices is gaining importance as well to balance/parallelize works on the machines and the network.

This mindset of using multi-core and multi-threaded routers has recently become prevalent, and there are a few ideas that aimed to use multi-core or multi-threaded techniques in the network protocols. Grover [37] proposed to employ multi-core routers to improve BGP protocols with a multi-threaded PBTS model during the BGP operations. Also, in industry, creating a solution utilizing multi-core routers has been patented by Kingsley et al. in [48], net-list devices are managed in parallel by different regions on the LSI devices. However, in [11], bounding boxes are created on nets, and multiple threads manage each net. Executed threads routed in parallel according to the created boxes.

In addition to usage of hardware-based routers, in the first years of the 2000s, the approach of software-based routers was proposed. A key motivation was that multi-core platforms could be profitable because of their easy customization. Bolla et.al. [14] measured software-based routers' performance and analyzed the networking performance based on throughput and power consumption. They were able to show that the new model proposed was able to save energy in terms of green-energy. Proposed optimization policy provided about %40-%50 of energy savings by using a multi-core structure on software routers.

Multi-core routers are also used for managing router tables in parallel [27]. The leading idea here is to improve software-based routers' performance during forwarding mechanism for a given packet. It is hard to manage the next hop during data transfers because of the growing Internet background traffic. However, by means of smart scheduling, virtualization and parallelization of router's processes across multiple cores, the researchers showed that CPU loads and packet lookup time could be reduced significantly.

Also, multi-core routers are used for lookup process of IPv6 addresses with new proposed approach [30] which uses multi-core background to parallelize store/scan routing tables. IPv4 addresses are 32 bits, and they are becoming insufficient because of the increasing number of Internet-enabled devices. For this reason, a new IP addressing method, IPv6, with 128 bits long addresses is being deployed. This new deployment and augmentation impose two main problems at the routers: bigger size of the routing tables and greater complexity for the packet lookup process. Improving hardware skills and developing new software solutions have been tried to make these processes faster. Another new approach is parallelizing the method of storing and scanning routing tables by means of multi-core designs. In this approach, routing tables are split into smaller ranges organized in a tree-based structure, and different cores process each of them at the same time. Performance is increased 10x times [30] by exploiting multiple cores on the routers for the forwarding problem of IPv6 addressing.

In general, most of the proposed multi-core routing techniques focus on improving the packet lookup time in a router. In our approach, we aim to improve the network-wide routing performance via a multi-core routing design, and, further, introduce a framework to employ multiple cores for composing multiple paths between endpoints.

2.5 Theory - NP Completeness

In the algorithmic analysis, problems' complexity is determined in terms of their computability (i.e., solvable or not) for a given set of input parameters and verifiability of a given possible solution's existence in the set of all possible solutions. Polynomial time bounded (P) problems are the set of problems that can be solved in polynomial time and also verified in polynomial time. On the other hand, Non-Polynomial time bounded problems (NP) can be solved in non-deterministic Turing Machine which is a Turing Machine enhanced with non-deterministic choice function. However, these

NP problems can be verified in polynomial time. The relation between P and NP problems is still one of the famous argue that are not exactly known yet.

NP-Complete problems are NP problems that can be verified in polynomial time and be solved in polynomial time on a non-deterministic Turing Machine. Every NP problem must be reducible/transferable to one of the NP-Complete problems in polynomial time quickly to be stated as an NP-Complete. So, every problem in the NP-Complete set can be reduced to each of them quickly, i.e., in polynomial time. NP-Complete problems are not solvable in realistic time. Further, NP-Hard problems are unique problems that as hard as the most difficult NP-Problem. NP-Complete problems are also NP-Hard, but vice versa is not true for the NP-Hard problems. For example, some decision problems are NP-Hard but not NP-Complete [39]. To show the given problem is NP-Complete/NP-Hard, it is required to confirm that the given problem is in NP-class followed by finding a polynomial time function for reducing the problem to one of the NP-Hard/NP-Complete problems. Some well-known NP-Complete problems [33, 34] are Boolean Satisfiability Problem(SAT), Knapsack Problem, Travelling Salesman Problem (TSP), Subset Sum Problem, Clique Problem, Vertex Cover Program, Independent Set Problem, and Graph Coloring Problem. Also most of combinatorial search problems are considered as NP-Hard [50], and our substrate generation problem is a version of this kind of problems. We now cover some of the most relevant NP-Complete problems relevant to the substrate generation problem for multi-core parallel routing.

2.5.1 Multi-Commodity Flow Problem

Network flow problem is a path selection problem for a flow from one source node to another destination which maximizes the throughput. As a more general version of this problem, multi-commodity flow problem [3] has multiple flow demands among different source-destination pairs. Multi-commodity flow problem finds if all the demands can be satisfied or not by finding a separate path to each flow through the network. In one perspective, the multi-commodity flow problem is the most generic form of the multi-path routing problem we face on the Internet. The problem is NP-complete for feasible integer solutions that allow just integer flow demands. When the fractional flow demand is allowed, the problem can be solved with linear programming in polynomial time.

2.5.2 Subset Sum Problem

Subset Sum Problem is crucial in cryptography and complexity theory. For a given set of integers (S) and a value sum (t), the Subset-Sum Problem determines if there is a subset of S such that the sum of elements in S is equal to the given sum t. The Subset-Sum Problem is a specific version of the Knapsack Problem [47] and the Partition Problem [17], and it is one of Karp's 21 NP-Complete problems [46]. Approximation algorithms can be used to solve this decision problem. It is related to our substrate generation problem as we want to create a set of all possible virtual topologies that maximizes the sum of throughput of each substrate.

2.5.3 Edge Disjoint Path Problem

The Disjoint Set Problem [58] can be both vertex-disjoint or edge-disjoint which are reducible each other. Edge-disjoint path problem tries to find k different non-overlapping paths for a given source-destination pair. Edge-disjoint path problem is one of the graph decomposition problems in graph theory. The disjoint path problem is known NP-Complete, and it is hard to approximate. The disjoint path problem is related to our proposed MCPR technique as its overall goals are similar to our heuristics' goals. In MCPR, we try to generate substrates that include non-overlapping, i.e., most disjoint paths.

Chapter 3

Parallel Routing

Calculating single shortest-path is a well known and spread technique, such as OSPF, for data transfers between two nodes. Shortest-path calculation gives a fast response because of its greedy approach; however, one of the problems with that technique is a limited capability to derive distributed load balancing over the network. In order to solve the workload balancing issue, data transfers can be done through multiple path routing. Yet, dynamic multi-path routing protocols reacting sufficiently fast to the network changes is hard, and typically it takes too much time to generate diverse paths in a feasible time span. The key novelty of our method is to ensure both scalability and load balancing in a practical manner. The main goal of our approach is that many of the current routers have multi-cores, and this parallel processing ability of these routers could be utilized to optimize the data load over the network, instead of jamming all the data into one shortest single path.

Parallel routing aims to provide multiple shortest paths, each of which is generated over different substrates of the topology, as shown in Figure 3.1. The main idea is that different slices (substrates) of the router topology are given to each core,

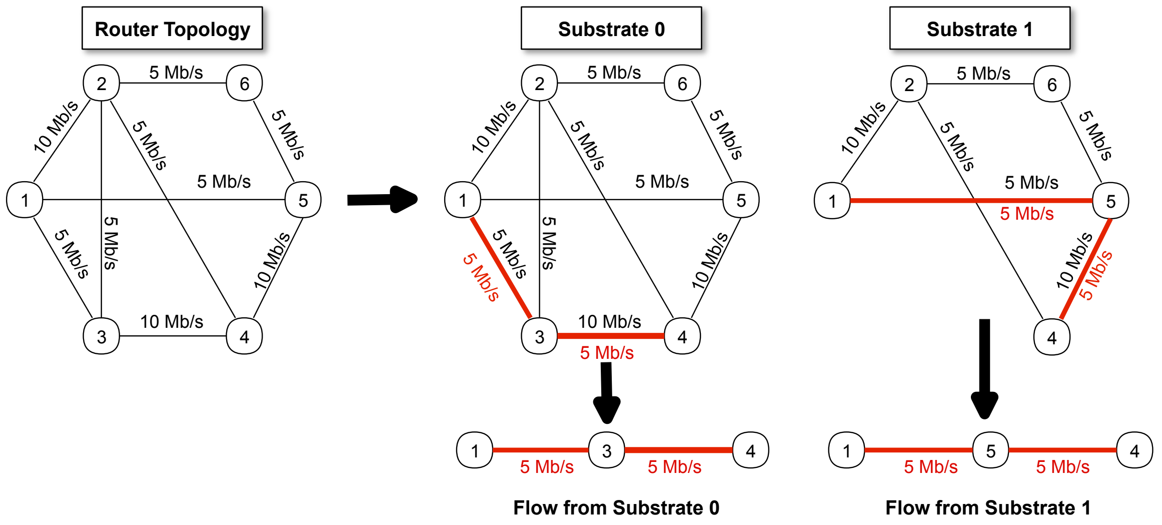


Figure 3.1: Motivating scenario with two cores

and the e2e data transfer is split into the shortest paths calculated on each substrate topology. Once assigned to Substrate i, flow will follow the shortest path calculated by that Substrate i. However, all the flows will be using the same physical topology regardless of which substrate they are assigned to. So parallel routing can lead to a system where parallel routes are produced on virtual substrates over the same physical topology. It is, then, up to the e2e transport protocol's decision to which one of these paths from the substrate topologies to use with what rate.

If (i) different virtual routing topologies based on the actual topology are assigned to a separate core of multi-core routers, (ii) data transfers could be distributed over these virtual topologies, and (iii) the current well-known shortest-path calculation techniques are executed on each core as well; then data load could potentially be better distributed over the network. This thinking is the key inspiration for our design of MCPR. Figure 3.1 illustrates a motivating scenario where two virtual substrates are produced. Substrate 0 is equivalent to the real router topology, whilst Substrate 1 is generated by removing node 3 from Substrate 0. Even though there are many other possible paths available over the network, current systems would carry 5 Mb/s on a single path. But, if these two topologies were given to a two-core router and *current shortest path finding algorithms* were executed in parallel on each core, then each path could transfer 5 Mb/s data. Comparing to the current systems, each substrate's shortest path is different, and the two collectively yield a total of 10Mb/s throughput from node 1 to 4 for *two cores* scenario. While creating a new substrate, in addition to removing node(s), some edge(s) can also be omitted, like the edge connecting node 1 and node 3. In that case, similar to the previous example, there would be two different paths to reach the destination node 4, which are path 1-3-4 and path 1-2-4. Each substrate is able to find the shortest path that can carry 5 Mb/s and a total of 10 Mb/s capacity can be transferred which is again two times of the capacity of current systems that calculate single shortest-path over one virtual topology, a.k.a. OSPF.

Parallel routing, based on shortest path calculations on separate substrate topologies, shows that with little changes on the current systems and by using multi-core property of routers, data transfers can be done over different paths at the same time and those paths can be found in parallel by devising multiple substrate topologies. This way, the multi-path routing problem can be reduced to a heuristic that decides which nodes/edges will be left out when generating the next substrate such that the largest possible aggregate throughput is attained. To reach an optimum solution, the design goal of parallel routing is to *generate substrates that yield the most diverse and non-overlapping shortest paths possible.* For a network with E edges, 2^E different substrates could be generated with no constraints on nodes or connectedness of the network, and we propose various heuristics that select network elements to be removed for creating a new substrate in Chapter 4.

The parallel routing design principles and main features are:

- *Shortest path calculation is abundant and efficient.* Legacy shortest path routing solutions are very much optimized and designed into the fabric of routers. Multi-path calculations utilizing them will be easy to deploy as well.

- *Multi-core CPUs are readily available.* It is possible to execute multiple shortest path routing daemons in parallel on the existing routers with multiple cores. Each core can run a separate instance of legacy routing protocols such as OSPF.

- *Robustness to network dynamics.* A critical challenge of multi-path routing is its brittleness against network dynamics such as failures or demand spikes. Such network changes may require re-calculation of the entire multi-path set, which can be prohibitively costly in routing timescales. Parallel routing delegates the path re-calculation to each substrate and lets the shortest path routing algorithm on each substrate perform the re-calculation.

- *Substrate generation can be centralized.* The dividing part of parallel routing is the most challenging as it requires finding the best set of substrate topologies so that their shortest paths minimally overlap. This is, as we will detail later, a heavy computation task. The advantage is that such heavy computation can be done in software-defined networking (SDN) controllers or other centralized locations with high computation power. Failures or network dynamics do not necessitate the substrate generation to be done within the routers themselves since the re-calculations of shortest paths can be independently done by each core. The substrate generation could be done at larger timescales without any major sub-optimality. Further, centralizing substrate generation allows goals like multi-path traffic engineering which require a global and centralized view of the network.

3.1 Formal Description

We, now, provide a formal definition of the substrate graph generation problem of multi-core parallel routing. Given an underlying network topology as a graph $G = \{V, E\}$ with a set of vertices $|V| = n$ and edges $|E| = m$, multi-core parallel routing's substrate graph generation involves several decision parameters: (i) *number of substrate graphs* to generate and (ii) *edge weights* on each substrate graph. Let $w_{uv} = 0..k-1$ be the weight of the edge from vertex u to v in G, where w_{uv} can be set to k different integer values. Further, let S be the set of all possible substrates of G. Substrate $S_i \in S$, can be expressed as $S_i = \{V_i, E_i\}$ where $V_i \subseteq V$ and $E_i \subseteq E$. Then, the number of possible substrate graphs is $|S| = k^m$.

The overall goal of multi-core parallel routing is to maximize the throughput of the network by using the shortest paths from a subset of the substrates. Let $q \subseteq S$ represent a group of substrates and $P(q)$ be the collection of shortest paths generated from the substrates in q. Further let $T(q)$ be the total throughput attained from the shortest paths $P(q)$. Then, we can formulate multi-core parallel routing's problem of generating a group/set of substrates that maximizes the throughput, MAX_SUBSTRATE_SET, as follows:

$$\max_{q \subseteq S} \quad T(q) \tag{3.1}$$

subject to

$$\exists \;\; (u \to v) \in P(q) \quad \forall \;\; u, v \in V \tag{3.2}$$

where $(u \to v)$ is a path from node u to node v. (3.7) assures the resulting multi-path routing provides at least one path between all source-destination pairs.

Formulation of MAX_SUBSTRATE_SET in (3.1) looks at the problem in a black box manner. It is possible to express the substrate generation problem from the network's point of view in a white box style. In particular, network throughput is maximized when routing calculates paths with minimal overlap. Next, we will express the substrate generation problem in terms of minimizing overlap.

For a graph g=$\{\upsilon, \epsilon\}$, let R_g be the routing vector such that $R_g(l)$ is the number of shortest paths traversing link $l \in \epsilon$. Note that $R_g(l)$ is the edge betweenness centrality of a node. Given a set of substrate graphs $q = \{G_1, G_2, .., G_j\}$ in multi-core parallel routing, we can express the number of shortest paths traversing l as

$$t(l, P) = \sum_{g \in P} R_g(l) \tag{3.3}$$

The substrate generation problem of multi-core parallel routing is, then, to make the usage of each link as even as possible, which also implies a minimal overlap among shortest paths. To factor in the varying number of substrates, we can aim to minimize the difference between the minimum and the maximum $R_g(l)$. Hence, we write multi-core parallel routing's substrate generation problem as a minimization of the unevenness in the usage of links, MIN_SUBSTRATE_SET, in G=$\{V, E\}$:

$$\min_{q \subseteq S} \quad (\max_{l \in V} t(l, q) - \min_{l \in V} t(l, q)) \tag{3.4}$$

subject to

$$\exists \quad (u \to v) \in P(q) \quad \forall \quad u, v \in V. \tag{3.5}$$

MIN_SUBSTRATE_SET provides a clear guidance on how heuristics should be designed for the substrate generation problem. In the next section, we will use this guidance to minimize the maximum load on individual links while trying to maximize the aggregate throughput.

Producing a new substrate dynamically to reach the optimum result is hard to determine in a feasible time cost normally. On each step, we remove one or more network element(s) which could be stuck earlier than others. In multi-core parallel routing method, we defined these elements by using some heuristics, based on network centrality metrics, defined in Chapter 4.

Now, we show MAX_SUBSTRATE_SET problem is NP-Complete by giving a reduction to the Subset Sum problem. First, we formally represent the Subset Sum Problem as SUBSET_SUM(S, t), which is the problem of finding if there is any subset of S such that the sum of its elements is equal to t for a given set of S integer numbers.

SUBSET_SUM(S,t) is one of the Karp's 21 NP-Complete problems [46]. Next, we define one part of the MAX_SUBSTRATE_SET problem, i.e., SUBSTRATE_SET.

Let SUBSTRATE_SET(S,t) be the Substrate Set Problem which finds the substrate set q such that the aggregate throughput obtained from the shortest paths of the substrates in q is equivalent to t. This decision problem can be formulated as follows:

$$\exists q \subset S \mid T(q) = t \tag{3.6}$$

subject to

$$\exists \quad (u \to v) \in P(q) \quad \forall \quad u, v \in V \tag{3.7}$$

Corollary 1: *MAX_SUBSTRATE_SET's search space is* $\mathcal{O}(2^{k^m})$.

Proof: Finding the q that maximizes $T(q)$ requires scanning of all possible qs. Let Q be the set of all possible $q \subseteq S$. Then, the search space size for MAX_SUBSTRATE_SET is the size of Q, which is:

$$\begin{aligned} |Q| &= \sum_{i=1..|S|} \binom{|S|}{i} && (3.8) \\ &= \sum_{i=1..|S|} \frac{|S|!}{i!(|S|-i)!} && (3.9) \\ &= 2^{|S|} - 1 && (3.10) \end{aligned}$$

Substituting $|S| = k^m$ in (3.10), we find the number of possible substrate sets $|Q| = 2^{k^m} - 1$, which is clearly not polynomial in terms of the number of edges m. Since $m \geq n - 1$ in a connected network, it is NP in terms of the number of nodes n as well.

Lemma 1: *SUBSET_SUM(S,t)* $<_P$ *SUBSTRATE_SET(S,t)*

Proof: Calculation of $T(q)$ in (3.1) can be done with $|q|$ all pair shortest path calculations, which is polynomial. Let's assume that a solution substrate set q is one of the combinations of S which is a subset of all possible substrates set. In order to reduce to the SUBSET_SUM(S,t) problem, selected subset will be summed up which is a polynomial process. Similarly, in SUBSTRATE_SET(S,t), each substrate set has a value of total throughput $T(q)$ which can be calculated in polynomial time by using the legacy shortest-path routing algorithms. So, finding a subset of substrates that gives an exact value of a given throughput, can be reduced to SUBSET_SUM(S,t) which is NP-Complete [33].

Lemma 2: *SUBSTRATE_SET(S,t)* $<_P$ *MAX_SUBSTRATE_SET(S)*

Proof: MAX_SUBSTRATE_SET(S) problem is trying to find a solution to the SUBSTRATE_SET(S,t) problem such that the solution maximizes the value of *t*. Assuming that data flows have inifinite demand, the maximum amount of transferable data via the graph G is equal to the sum of capacities of edges/links in G. Let that sum be totaldataflown. So, we can write a polynomial time algorithm that reduces MAX_SUBSTRATE_SET(S) to SUBSTRATE_SET(S,t) by decrementing given value of maximum possible *t*, i.e., totaldataflown, until the SUBSTRATE_SET(S,t) is solved. Assuming that the edge capacities are integer, and hence t is an integer, the complexity of this reduction algorithm will be in the order of sum of all edge capacities, which is clearly polynomial. Algorithm 1 details this reduction algorithm. Instead of decrementing, it is also possible to perform a binary search to find the maximum t value solving SUBSTRATE_SET(S,t). As shown in Algorithm 2, this approach could also work with decimal edge capacities as long as a fixed precision is defined as a stopping condition for SUBSTRATE_SET(S,t). Thus, MAX_SUBSTRATE_SET(S) problem is also NP-Complete with a reduction to SUBSTRATE_SET(S,t).

Theorem 1: *MAX_SUBSTRATE_SET is NP Complete.*

Proof: It follows from Lemma 1 and Lemma 2.

Algorithm 1 Decremental design for Maximum Substrate Set

```
procedure MAX_SUBSTRATE_SET($S$)
    $t = totaldataflown$
    while $!solved$ do
        $solved = SUBSTRATE\_SET(S, t)$
        $t = t - 1$
    end while
end procedure
```

Algorithm 2 Binary search design for Maximum Substrate Set

```
procedure MAX_SUBSTRATE_SET($S$)
    $start = 0$
    $t = totaldataflown$
    $maxt = t$
    while $start <= t$ do
        $mid = (start + t)/2$
        $solved = SUBSTRATE\_SET(S, t)$    ▷ returns true if there is a solution S attaining throughput within a certain precision of t
        if $solved$ is true then
            $start = mid$
        else
            $t = mid$
            $maxt = t$
        end if
    end while
    return $maxt$
end procedure
```

Chapter 4

Parallel Routing Heuristics

Parallel routing creates slices, i.e., *substrates*, from the entire router topology to run those substrates on each core for getting more diversified paths for end to end data transfers. For a given network with E edges, 2^E different substrates could be generated if no constraints are imposed on the substrates being generated. In the entire set of possible substrates, some topologies could be incapable of producing better results, i.e., that could include a partitioned set of nodes in a substrate which causes disconnected routing within a substrate. Considering such constraints, our design goal of parallel routing is to find optimal substrates that yield the most diverse and non-overlapping shortest paths possible. Thus, we develop intuitive heuristics to create new substrates which can improve total aggregate throughput.

High level of activity diagram for substrate generation is given in Figure 4.1. According to the substrate generation method and number of cores, all substrates will be created by the system. Then, the generated substrates will be assigned to different cores of routers, and each of them will continue to do the same thing as in their current shortest path policies without any modifications. Therefore, the proposed

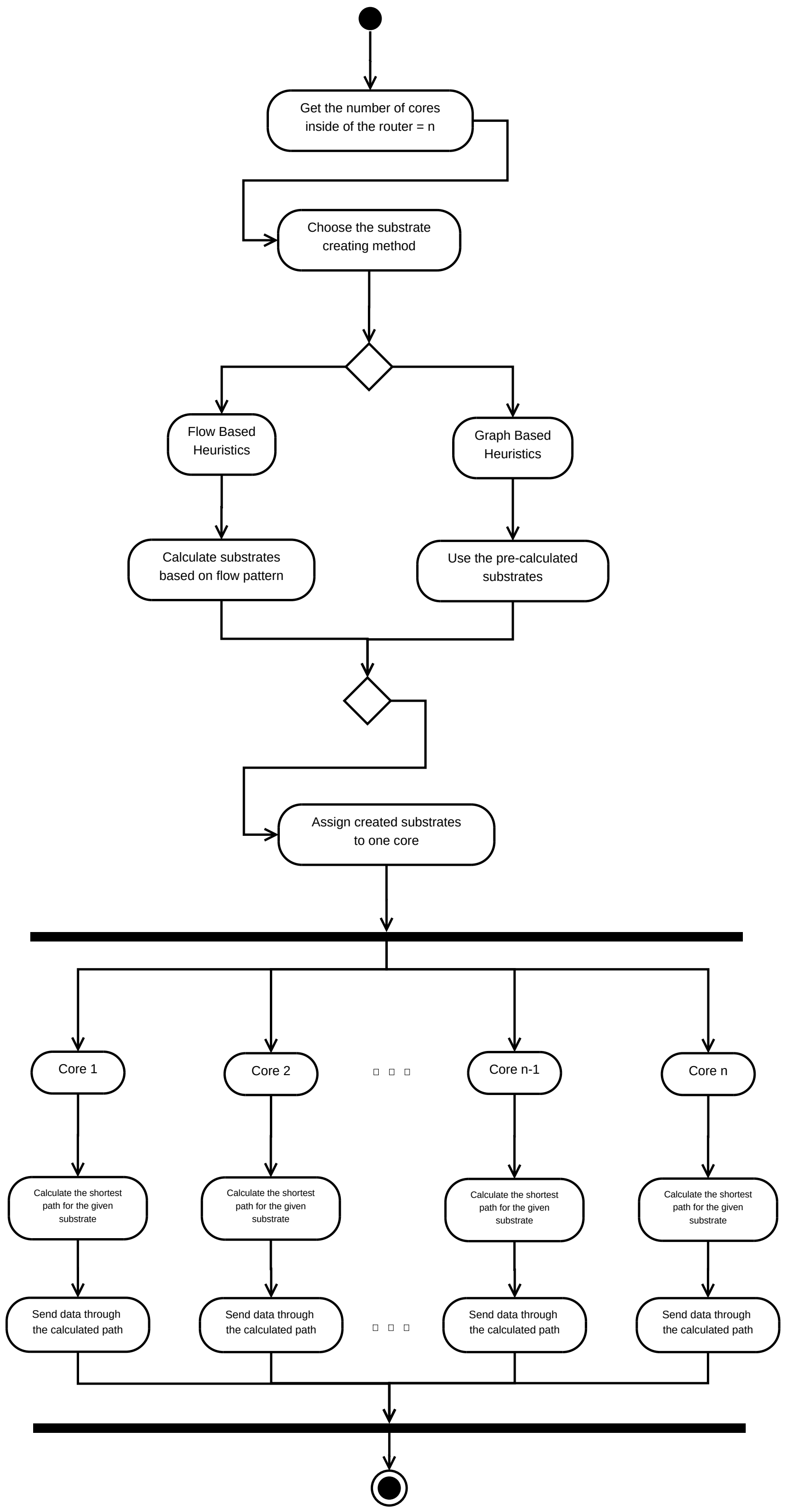

Get the number of cores inside of the router = n
Choose the substrate creating method
Flow Based Heuristics
Graph Based Heuristics
Calculate substrates based on flow pattern
Use the pre-calculated substrates
Assign created substrates to one core
Core 1
Core 2
Core n-1
Core n
Calculate the shortest path for the given substrate
Send data through the calculated path

Figure 4.1: Activity diagram for parallel routing with n cores

parallel routing method is able to (i) give a fast reaction to network dynamics since, in the event of a topology change (e.g., due to failures or demand spikes), each core will be automatically calculating the new shortest paths within their substrates, and (ii) adapt to existing systems easily with little adaptation cost.

When generating the substrates, a crucial challenge is to assure all-to-all connectivity. To address this issue, we define **Substrate 0** as the actual given topology. When generating the subsequent substrates, however, some nodes or edges are going to be omitted. A simple heuristic step could be to omit the nodes/edges that are being used the most by the shortest paths in Substrate 0. We try two approaches which are explained in later in this chapter.

While generating subsequent substrates, some network elements, such as nodes or edges, will be omitted from the given real topology. Our heuristics analyze the given topology to predict which routers/edges could be maxed out earlier than others. These routers/edges will be removed from some of the subsequent substrates to balance the load in the underlying topology. Thus, the primary design parameter is to decide which nodes/edges are going to be omitted on the generated substrate. Our first step of heuristics could be to max out the most utilized nodes or edges that might be more centralized, and as a result, these elements can be used more than the others by being on the most of shortest paths. There could be two different criteria where we can focus on to have improvements in the long term: Graph-based solutions and Flow-pattern-based solutions. In the graph-based technique, the whole topology will be analyzed and all the edges connected to the central elements are going to be expelled on the next generated substrate. Similarly, in the flow-based method, the traffic pattern at the particular time is going to be inspected, and selected edges with highest utilization will not be on the new substrate. Therefore, our heuristics are concentrated on figuring out the possible behavior of the network elements and network dynamics in a feasible way and finding the network elements that can be omitted to generate new substrates.

As Substrate 0 will be the original topology to guarantee the connectivity, for other substrates, the generating process could be done in two ways of selecting the base topology that the new substrate will be generated from: Cumulative approach or Independent approach. In the first method, all substrates are forked from the previous substrate that was already produced. This process ensures that over-capacitated network elements, which are already removed from the previously generated substrates, cannot be on the newly created substrate. For instance, in order to calculate the 4th substrate for a 4 core router, 2nd and 3rd substrates should have already been calculated for that router. If the node was removed in the 2nd substrate, it would not be on the 3rd and 4th substrate as well. Whereas, in the second method, all substrates are calculated from the given real topology, and so all of the different substrates can be generated at the same time. We also observe different amount of node or edge removals and analyze the effect of removal percentages. In each step of substrate generation, we remove elements and calculate the number of removed edges until we reach the amount of the given removal percentage of the edges.

4.1 Graph-Based Heuristics

Graph-based heuristics are purely based on the graph properties of the given real topology. In graph-based heuristics, network centrality metrics are used to find the most 'central' nodes as they are most likely the nodes to be maxed out by the shortest paths. The main purpose of those methods is to increase the number of non-

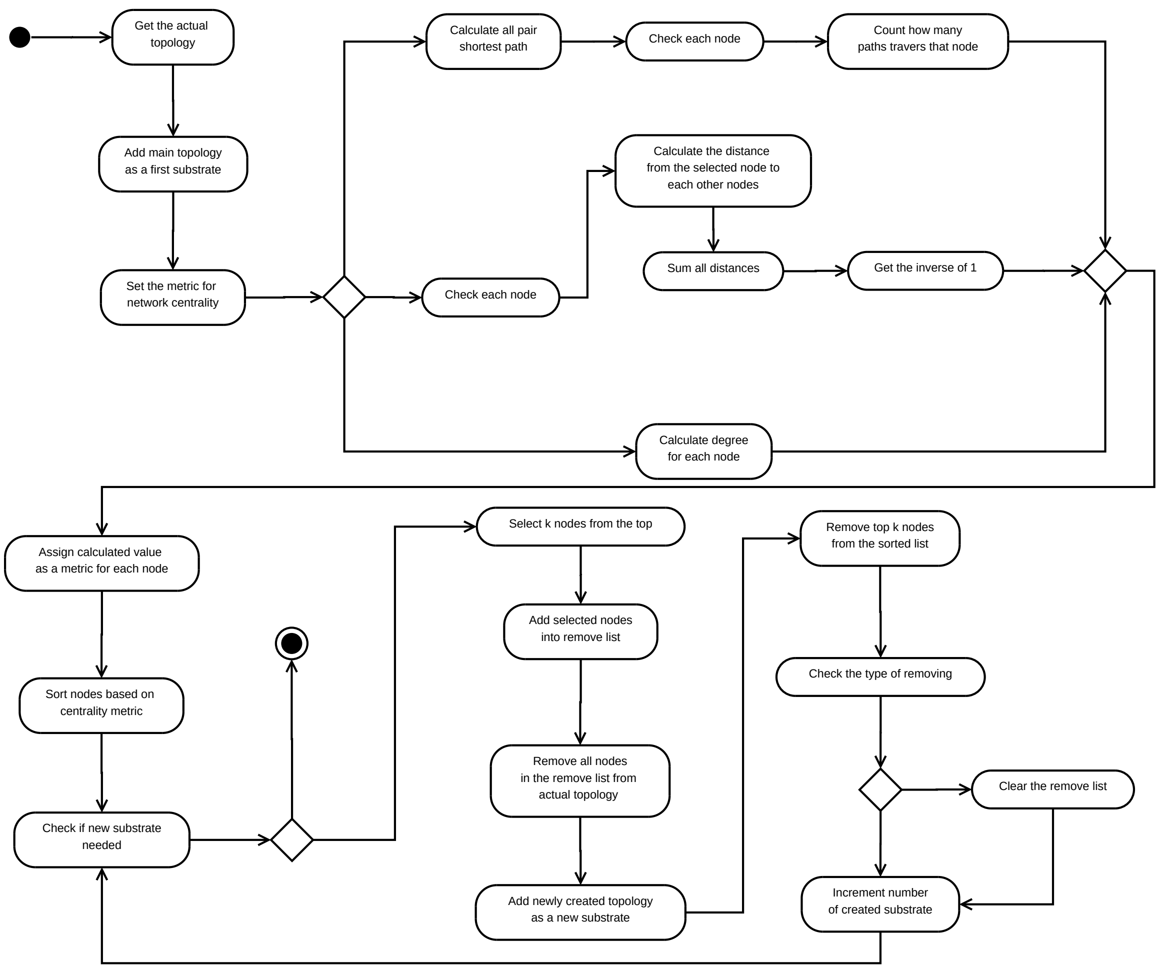

Get the actual topology
Add main topology as a first substrate
Set the metric for network centrality
Calculate all pair shortest path
Check each node
Count how many paths travers that node
Calculate the distance from the selected node to each other nodes
Sum all distances
Get the inverse of 1
Check each node
Calculate degree for each node
Assign calculated value as a metric for each node
Sort nodes based on centrality metric
Check if new substrate needed
Select k nodes from the top
Add selected nodes into remove list
Remove all nodes in the remove list from actual topology
Add newly created topology as a new substrate
Remove top k nodes from the sorted list
Check the type of removing
Clear the remove list
Increment number of created substrate

Figure 4.2: Activity diagram for simulator

overlapping shortest paths for the newly generated substrate. For the graph-based approaches, *the only consideration will be the topology information, regardless of the traffic pattern, to determine the order of node(s) to be removed.* Therefore, these selected nodes for removal can be pre-calculated and remain unchanged until somehow the topology changes. On the other hand, the graph-based methods might not be robust in the dynamic and changing traffic flow patterns because they determine the congestion area only by using the topology information. Also in these methods, substrates are generated for being able to perform well under every possible condition

Algorithm 3 Graph-Based Remove

```
procedure GRAPH-BASED REMOVE($amount, cores$)
    $substrateList.Add(mainTopology)$
    $metric \leftarrow [Betweenness, Closeness, Degree, etc.]$
    $sortedList \leftarrow nodes.Sort(metric)$
    $numOfSubstrates \leftarrow 1$
    while $numOfSubstrates < cores$ do
        $selectedNodes \leftarrow sortedList.Top(amount)$
        $generatedGraph \leftarrow topology.Remove(selectedNodes)$
        $substrateList.Add(generatedGraph)$
        $numOfSubstrate \leftarrow numOfSubstrate + 1$
    end while
end procedure
```

instead of the specific solutions for the given network dynamics. Therefore, spreading data load over the network may not temporarily be well as in flow-based solutions. Algorithm 3 shows the steps for graph-based heuristics.

Our graph-based heuristics are based on the centrality characteristics of the nodes in the network topology. Central nodes are calculated and selected as potential congestion areas without detailed analysis of the actual network flow. They can be calculated once and recomputed solely when the topology changes. We utilized the following centrality metrics to determine which nodes to remove:

Node Degree Centrality (NDC): NDC is based on *degree centrality metric* which measures the number of one step in/out connections between the other nodes. It gives the number of neighbors which also shows the possibility of using a node. If the node has a higher degree value, that node iintuitively should be more central to the overall topology and used (i.e., traversed by shortest paths) more than others. Thus, in the NDC heuristic, higher degree nodes will be eliminated first. For a given topology with n nodes and m edges, time complexity for NDC is O($n + m$).

Node Betweenness Centrality (NBC): NBC uses *betweenness centrality metric* to find the central node(s) to estimate the congested nodes statically. Betweenness centrality is a graph metric that measures the number of all pairs shortest paths traversing through a particular node. Since the current systems use the shortest path for data transfers between the source-destination pairs, NBC reduces possible intersection of the nodes that most commonly used on shortest paths and helps load balancing. NBC, in essence, captures the very basic notion of overlapping shortest paths. Thus, eliminating the nodes with high NBC is in direct intuition with the goal of graph-based heuristics, i.e., to increase non-overlapping shortest paths in the subsequent substrates.

NBC needs to calculate all shortest paths between each possible pairs to find how many shortest paths will pass through a given node, Algorithm 4. For a given topology with n nodes, the time complexity of calculating betweenness centrality metric for the entire topology will be O($n^2 \log n$).

Algorithm 4 Betweenness Centrality score calculation

```
procedure BETWEENNESS($nodeList$)
    for each node $t \in nodeList$ do
        $t.metric \leftarrow 0$                              ▷ initialized to 0
    end for
    for each node $source \in nodeList$ do
        for each node $dest \in nodeList$ do
            $pathList \leftarrow$ SHORTESTPATH($source, dest$)
            for each node $t \in pathList$ do
                $t.metric \leftarrow t.metric + 1$
            end for
        end for
    end for
end procedure
```

Edge Betweenness Centrality (EBC): EBC is the edge version of the Node Betweenness Centrality. Instead of counting the number of shortest paths using a given node, EBC calculates how many shortest paths use a given edge. Thus, EBC is an edge-centric centralization metric that addresses the most central edges over the topology. Removing the most used edges of the central node can be helpful to reach higher aggregate throughput than losing the node itself. Similar to NBC, the complexity time for EBC will be O($n^2 \log n$).

Node Closeness Centrality (NCC): The primary goal of multi-core parallel routing is trying to find the most central node to exclude from the subsequent substrates, and NCC uses *closeness centrality metric* which is another graph metric measuring the distance from one of the nodes to all others. NCC selects the nodes for removal when they have the average shortest distance to all other nodes, with the intuition that such a node would be on more of the shortest paths. NCC is a normalized parameter which all the values are between [0..1]. The time complexity for NCC will be O($n^2 \log n$) for a topology has n nodes, and pseudo code is given in Algorithm 5.

Edge Closeness Centrality (ECC): In comparison to node removal, multi-core parallel routing performs better when the edges are removed to generate a new substrate. So, similar to EBC, we calculate the shortest distance from a given edge to other edges instead of measuring the shortest distance other nodes as in EBC. Small ECC values give us the edges closest to the all other edges on the topology. For a topology with n nodes, the time complexity will be O($n^2 \log n$) as NCC.

Algorithm 5 Closeness Centrality score calculation

```
procedure Closeness($nodeList$)
    for each node $t \in nodeList$ do
        $t.metric \leftarrow 0$                                    ▷ initialized to 0
    end for
    for each node $source \in nodeList$ do
        for each node $dest \in nodeList$ do
            $distance \leftarrow$ ShortestPath($source, dest$)
            $source.metric \leftarrow source.metric + distance$
        end for
        $source.metric \leftarrow 1/source.metric$
    end for
end procedure
```

Eigen Vector Centrality (EVC): Eigen Vector Centrality is another metric in graph theory that gives the relative scores to all nodes. The continuous calculations are performed to find the score of a node. High-scored neighbor nodes will contribute more than low-scored neighbors to score a given node. We used a network analyzing tool Gephi [10, 21] for calculate the EVC. In our calculations, we performed 100 iterations.

Page Rank Centrality (PRC): Page Rank Centrality is a modified version of EVC. It is based on traversing nodes like EVC, additionally with a scaling factor. We used Gephi to calculate PRC scores. We chose probability as 0.85, and 0.001 for the epsilon in our calculations.

Harmonic Closeness Centrality (HCC): HCC is a different version of the Node Closeness Centrality. NCC is the inverted version of the sum of distances. However, HCC is the sum of the inverted distances. We try to see the effect of this little change on multi-core parallel routing.

Multiplication Centrality: Centrality metrics are used to determine the most central nodes in the given graphs. However, in multi-core parallel routing, removing nodes will likely cause significant disruption on the later substrates. So, we decided to add another step for choosing some edges of the central nodes to remove. Instead of scoring node centralities, we try to find the most central edges like EBC, a metric to find how many times shortest paths are traversing through the edge. To merge both the node and edge centrality measures, we multiply the centrality metrics of the two nodes of a given edge to calculate the centrality score of that edge. Thus, the edge will inherit the centrality information of both of the nodes it touches during this decomposition step. In some sense, we change the scoring formula from node space to edge space. For example, when edge connects two central nodes, that means this edge has higher centrality score. However, central nodes can have a connection between nodes with lower centrality scores. In that case, multiplication centrality will give us a lower score than the other edges. As a result, we will not lose all the central edges of the central node in the next generated substrate(s), but the congestion on the hot spot edges will be removed.

We test three node centralities as a multiplication centrality. Since we are using the node centrality metrics to calculate the multiplication centralities, the complexity of the multiplication centralities will be the same as node centrality calculations.

Multiplication Betweenness Centrality (ENB): uses NBC, a.k.a Node Betweenness Centrality, to calculate the centralities of the nodes.

Multiplication Closeness Centrality (ENC): uses NCC, a.k.a Node Closeness Centrality, to find centrality scores.

Multiplication Degree Centrality (END): uses NDC, a.k.a Node Degree Centrality, for the node centrality value.

Random Node Removal (RN): Multi-core parallel routing tries to select removable nodes wisely for generating the next substrate. We also experiment with random node removal to observe the performance of an uninformed node removal process. We perform 20 times uniformly distributed random substrate generations and make static analysis to observe the average performance of multi-core parallel routing.

4.2 Flow-Based Heuristics

Most of current systems use the shortest path which is well-known and widespread technique for deciding a path from source to destination. In that case, some edges can be shared by multiple paths, and such overlapping of end-to-end paths causes congested spots on the network. Intuitively, the most used edges are going to be maxed out earlier than the other edges. In multi-core parallel routing, each substrate will calculate its own shortest paths independently that could increase the load on the shared edges of different substrates. As we remove the most utilized edges from the previous substrate(s), we try to obtain short paths that avoid congestion spots. Thus, our main goal in flow-based heuristics is predicting which edges carry more data flows than other edges, and use that prediction to balance the load across all edges on the topology. According to multi-core parallel routing, those heavily used edges will not be placed on the new substrate and data will be sent through the longer alternative paths.

The graph-based properties might not have an ability to capture dynamism in network traffic. Thus, designing heuristics that consider the current utilization of edges would be favorable to adapt the substrate generation process under traffic

Algorithm 6 Flow-based removal

```
procedure FLOW-BASED REMOVE($removalAmount, cores$)
    $substrateList.Add(mainTopology)$
    for each edge $t \in edgeList$ do
        $e.metric \leftarrow 0$                                    ▷ initialized to 0
    end for
    $numOfSubstrates \leftarrow 1$
    while $numOfSubstrates < cores$ do
        $nodeList \leftarrow SubstrateList.Last().nodeList$
        for each node $source \in nodeList$ do
            for each node $dest \in nodeList$ do
                $pathList \leftarrow$ SHORTESTPATH($source, dest$)
                for each edge $e \in pathList$ do
                    $e.metric \leftarrow e.metric + 1$
                end for
            end for
        end for
        $sortedList \leftarrow edges.Sort(metric)$
        $selectedEdges \leftarrow sortedList.Top(removalAmount)$
        $generatedGraph \leftarrow topology.Remove(selectedEdges)$
        $substrateList.Add(generatedGraph)$
        $numOfSubstrate \leftarrow numOfSubstrate + 1$
    end while
end procedure
```

dynamics. We propose flow-based heuristics as a short-term approach, dependent on the traffic patterns in addition to topology information. So, the generated substrates might change with the estimated/predicted data traffic flows that can emerge later on the network. Worse, for each flow set, a new substrate set must be generated periodically, and that might increase computational complexity. Therefore, flow-based heuristics can be computationally intensive but adaptive to the traffic dynamics.

Highest Flow (HF): To generate a new substrate, we count existing flows, traverse each edge and omit the most used edges in substrate i to produce substrate $i + 1$, which leads us to Highest Flow (HF), given in Algorithm 6. In each step of

Table 4.1: High level comparison of heuristics

Graph-Based	**Flow-Based**
Depends on Topology	Depends on Flows
Pre-Computed	Dynamic
Complex	Simple
Less Speedup	More Speedup
Coarse Granularity	Fine Granularity

substrate generating, we remove the number of edges according to the given removal percentage. If s represents the number of substrates, and n shows the number of nodes, the time complexity for HF is shown in $\mathrm{O}(sn^2 \log n)$. Although that method is faster than some of the graph-based heuristics, HF must be performed in regular periods when the traffic pattern changes. On the other hand, note that the graph-based heuristics must be run when the topology changes, e.g., due to node or edge failures.

Random Edge Removal (RE): In addition to Random Node Removal, multi-core parallel routing is also experimented with random edge removal. We perform multi-core parallel routing with 20 different random seeds to avoid noise.

4.3 Comparison of Heuristics

Graph-based heuristics are based on the topologies and infer which nodes would be congested first by analyzing network centrality metrics, and then remove selected nodes from next substrates. On the other hand, the flow-based heuristics are based on data flow patterns and try to predict which edges will be over-capacitated first. Therefore, graph-based techniques are just performed if the topology is changed such as insertion or deletion of edges or nodes. So, there are no additional computational

Table 4.2: Low level comparison of heuristics

-	**NBC**	**NCC**	**NDC**	**HF**
Metric	Betweenness	Closeness	Degree	Traffic
Complexity	O($n^2 \log n$)	O($n^2 \log n$)	O(n^2)	O($n \log n * s$)

costs to find priority of nodes for removal when the flow patterns change. Yet, flow-based methods adapt to current flows with some additional computational cost at each time when a new flow is generated.

A nice property of the graph-based heuristics is that the re-computation of the shortest paths could be performed if and only if there is a change in the topology, e.g., due to a node or link failure. Thus, flow level changes and traffic trends could be ignored in the graph-based approach, while the flow-based approach may have to recompute its substrates and their shortest paths against such dynamism. Of course, this dynamism brings more computational overhead for the flow-based approach, in return for more speedup possibilities in the aggregate throughput. Table 4.1 summarizes these tradeoffs between the two heuristic approaches.

Although graph-based heuristics are pre-computed techniques, generating a new substrate has higher computation cost. Contrary, the flow-based approach solves basic problems, but it should be held in real time. Explicitly, the flow-based technique will have better performance because of focusing on to spread possible paths by centering in the intensive traffic pattern over the network. Thus, flow-based heuristics achieve higher aggregate throughput with a better load balancing performance. Also, comparison of heuristics based on complexity is given in Table 4.2.

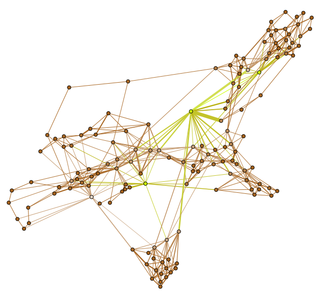
To sum up, flow level changes and traffic trends are ignored in the graph-based heuristics, while the flow-based approach needs to recompute its substrate graphs with such dynamism. The flow-based heuristic typically would achieve higher aggregate throughput with a better load balancing performance.

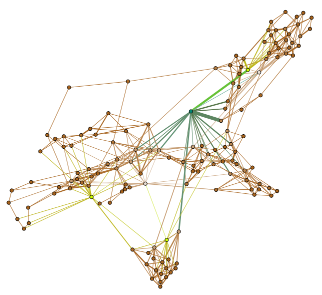
4.4 Removal Methods

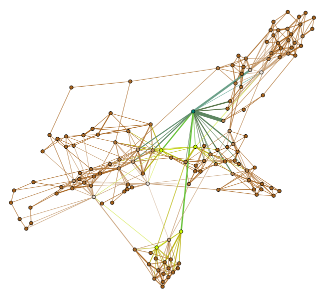
We explained how to score the node or edge centralities for choosing the removal of elements for the next generated substrates in 4.1 and 4.2. Multi-core parallel routing has different selection methods to remove elements from the ordered list. Flow-based heuristics periodically adapt to the flow patterns. They are also very sensitive and adaptive during the new substrate generation process by calculating shortest paths in each step. So, Highest Flow, HF, gives an upper bound for multi-core parallel routing heuristics using the centrality measures. Although HF is the upper bound, it should be calculated each time when flow pattern changes. That means, HF needs to change dynamically and will increase control plane communication. Graph-based heuristics, however, give pre-computed and statically generated substrates for multi-core parallel routing.

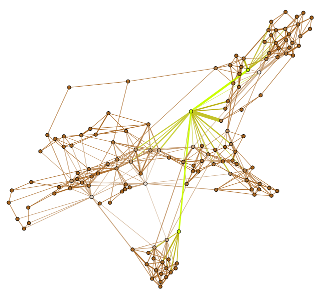
Thus, the main idea is that finding the best-chosen nodes to generate a new substrate from the ordered node list by centrality scores as close as possible to the HF. On the other hand, we compare these heuristics with Random Node Removal as a baseline which gives us a greedy solution without a computation cost.

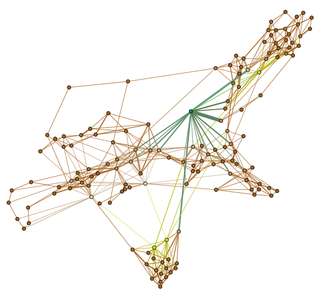
After calculating all the centralities, we perform four different removal methods for different amount of removal percentages. We also experiment with both independent and cumulative methods of the substrate generation techniques in all the removal methods. We now describe the removal methods.

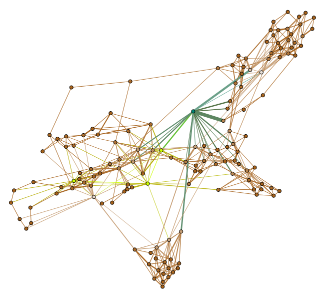

(a) Block Substrate 2

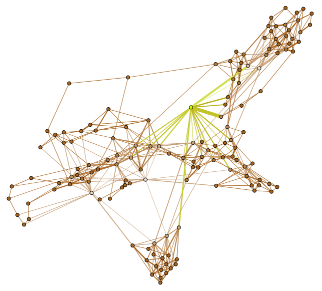

(b) Block Substrate 3

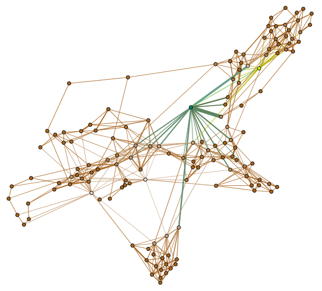

(c) Block Substrate 4

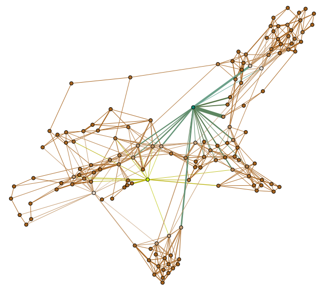

(d) Bucket Substrate 2

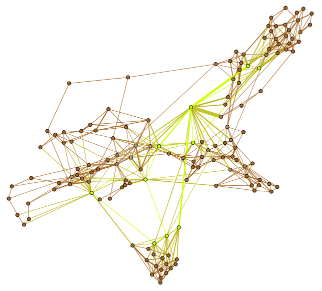

(e) Bucket Substrate 3

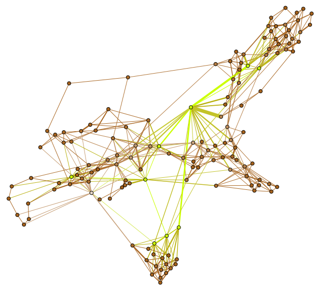

(f) Bucket Substrate 4


(g) Individual Substrate 2


(h) Individual Substrate 3


(i) Individual Substrate 4


(j) Block Cumulative


(k) Bucket Cumulative

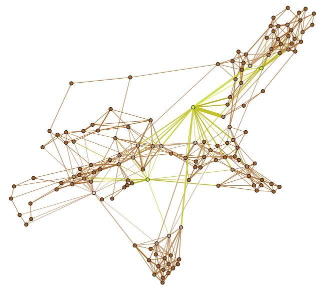
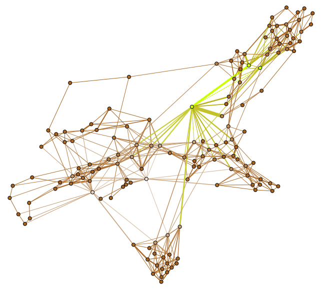
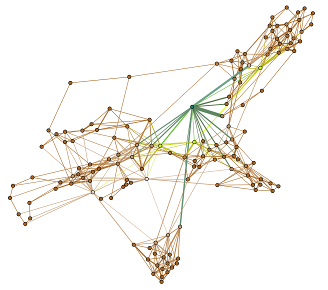

(l) Individual Cumulative

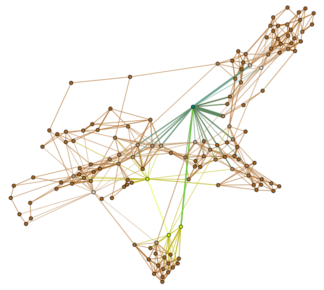
Figure 4.3: Visualization of removal methods on AboveNet with 8% removal

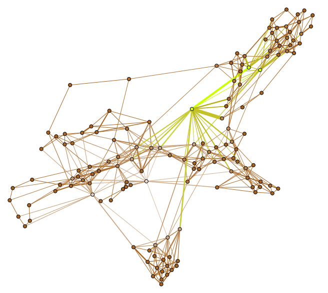
Block Remove : For a given list of elements (nodes or edges) ordered by their centrality, we choose elements as a block of some nodes or edges to remove. In the first substrate, we remove the first top block, and in the second one, elements in the second top block are removed, so on. That method removes most central nodes at the same time, and tries to offload the load to the least central nodes in the subsequent substrates.

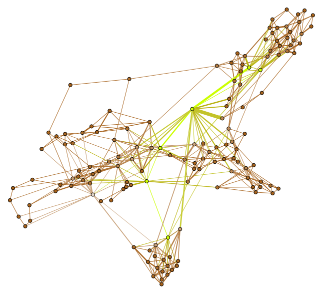
Bucket Remove : We create different clusters of the nodes on the ordered list as blocks and choose one item from each block into a bucket. Thus, in each substrate, we do not clean the top central nodes, and keep nodes in each cluster that causes small changes on the paths as in the adaptive iterative shortest path methods [70].


(a) Region Independent Substrate 2


(b) Region Independent Substrate 3


(c) Region Independent Substrate 4


(d) Region Cumulative Substrate 2


(e) Region Cumulative Substrate 3

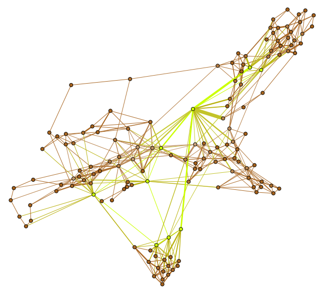

(f) Region Cumulative Substrate 4

Figure 4.4: Visualization of region removal on AboveNet with 8 % removal

Region Remove : We perform Breadth First Search [13] to discover the neighbors of the most central node and remove the central node along with its BFS tree. In each step, we choose a central node and try to clear the neighboring region of the node in the topology. So, if the congestion is coming from the usage of central nodes and it is regional, then we would be able to deal with congestion by clearing the whole congested spot. We do not have the depth limit while creating the BFS tree that discovers all the items until we reach the removal amount. Once, we reach the target count of nodes/edges to remove, we stop the BFS tree discovery process and remove the selected elements.

Individual Remove : In every step, we just remove one element from the ordered list to try to find how we are performing on other methods.

Comparison of The Removal Methods: We analyze different removal methods after scoring all the elements with various metrics. We, now, discuss how these methods affect generated substrates. We show which nodes are removed on the first four substrates of AboveNet, when NBC is used as a metric, and eight percent of the nodes are removed, in Figures 4.3 and 4.4. Remember that the first substrate, i.e., Substrate 0, is the actual topology, so we just show the rest of generated substrates. Green-ish nodes have higher scores whereas dark browns are less central nodes, and yellow nodes represent eliminated nodes. The last row, Figure 4.3(j)(k)(l) respectively Block, Bucket, and Individual remove methods, shows which nodes are deleted for a four core scenario when they are created cumulatively. Note that, all removed nodes on the fourth substrate are also removed on the previous substrates. However, in the Region removal method, there is no correlation between the independent and cumulative methods because of the dynamic discovery of which region to remove based on BFS tree.

In the Individual removal method, in each step, we select the top element and remove this element. Thus, we remove the highest top three nodes one by one as shown in Figure 4.3(g)(h)(i). So, we start to discover the topology from these nodes in the Region removal method. Also, note that, in Block, given in Figure 4.3(a)(b)(c), these three nodes are excluded on the first substrate whereas Bucket selects different nodes with smaller scores. Each of those three nodes is discarded on separate substrates as shown in Figure 4.3(d)(e)(f). When we analyze the cumulative substrate generation, in Figure 4.3(j)(k), we observe that the massive graph disruptions can reveal small changes on the later substrates that causes the similarities. Although Bucket gives us a little bit diversification, it is not better to clean the entire hot spot(s) than Block removal.

On the other hand, in Region remove which is given in Figure 4.4, independent, and cumulative substrate generations have a different pattern because of regional discovery of which nodes to remove. In later substrates, when a node is excluded from previous substrates, we also skip this node to start to discover it again. Thus, intuitively, the Region removal method will give better throughput in the case of cumulative substrate generation. Although there is no pattern for the Region, apparently, it is clearing the neighborhood of the top next element that could provide better throughput.

4.5 Parallel Routing Under Network Dynamics

As changes to a network topology is reality, we need to analyze MCPR's performance when the network has changed unexpectedly because of traffic spikes and/or sudden failures. Multi-Core Parallel Routing creates new virtual topologies, named substrates, and each topology calculates their own shortest paths. In the case of network dynamics due to failures, end-to-end multi-path routing algorithms will need to recompute their paths, or it is required to pre-compute the paths for the common failure patterns. Recomputation process means extra overhead on both control-plane communication and CPU usage. However, in multi-core parallel routing, only shortest path calculation is enough to modify the paths on each substrate when node or edge failures occur. Yet, after a failure, these modified shortest paths on substrates can have more or less overlap in comparison to the original shortest paths on the substrates. To evaluate the robustness of multi-core parallel routing against such failures, we calculate the total throughput over the network when one node or edge is down, and we compare the average throughput achieved with respect to the original throughput without failures.

4.5.1 Traffic Spikes

Network traffic patterns can be very dynamic and may result in temporary spikes with very different than expected rates. Mail server problems, virus scanners, malicious attacks are the most common reasons for the traffic spikes [98]. In addition to these, remote backups or scheduled backups on the network can cause traffic spikes. When we have a significant amount of data suddenly needed to be transferred among routers, it may cause new hot spots over the network topology. Retuning routing to accommodate these spikes means more calculation that affects CPU and more data on the link or specific paths that fall into congestion. System administrators try to estimate the traffic spikes before they happen to reduce the risk such as scheduling the backups and interfering the path calculations when a spike is detected. Yet, having a routing scheme that is robust to these spikes is critical to the practice.

4.5.2 Failures

In networks, there are two types of failure scenarios: (i) device or node failure, and (ii) link or edge failure. Traffic spikes can increase the workload of the routers which eventually trigger device failures. Also, changing traffic patterns can cause edge failures because of overloading of particular edges. A software update, bugs, traffic spikes, and electrical shut down are the common reasons for the failure of devices. Link failure can occur because of the device failure, and also traffic congestion. Device failures are mostly because of maintenance problems; however, edge failures do not have any specific pattern [35].

End-to-end path calculation is needed whenever network topology changes due to failures. Also, most of the failures are short-term and failed link or device could be back online in a while. In that case, all paths will be recalculated to determine the paths between source and destination pairs, which means increasing the CPU usage and the amount of control-plane data. Existing multi-path routing techniques require partial and complete re-calculation of multi-path routes, which can be too costly in terms of CPU overhead. This process typically is costlier than typical shortest path calculations. On the other hand, Multi-Core Parallel Routing does not require any additional re-calculation overhead because of its reliance on the well-known shortes path calculations. During this calculation, alternative paths are also computed, and it is mostly not needed to recalculation in the case of failure.

4.6 Experimental Setup

We implemented a new static analysis on C++ to create virtual world for ***calculating the amount of data throughput*** over the network under the proposed MCPR heuristics. Based on this simulator, we have the capability to generate substrates, find paths between each flows, and carry all data on flows by using max-min allocation method for competition on edge capacities. The simulator generates substrates according to our MCPR heuristics, and defines the data transfer paths on the substrates generated by the heuristics. Then, all data flows are sent on the substrates, and aggregate throughput is calculated based on the edge capacities of the given topology.

We used C++ in this static analysis because of its both object oriented background and adaptivity on most of the environments. In the object-oriented model [83], we create a set of objects or modules that are interacting with each other. Therefore, we implemented each module (e.g., metric calculator, and substrate generator) separately that improves the performance of the tests with parallelism. Further, we followed the Spiral software development model [12] to reduce the risks, and in each cycle, we implemented one of the modules. In our framework, a class diagram is given in Figure 4.5, we have five modules: Main module, Analysis, Metric Calculator, Removal Selector, and Substrate Generator. We store topology information, data flows, run parameters, centrality metrics, and generated substrates in different folders and files separately.

In this static analysis, we create a scratch folder for each topology and combine all the input files into this temporary folder. Metric Calculator module calculates the scores of the nodes/edges to order them, and stores the scores in a file. Then, removal

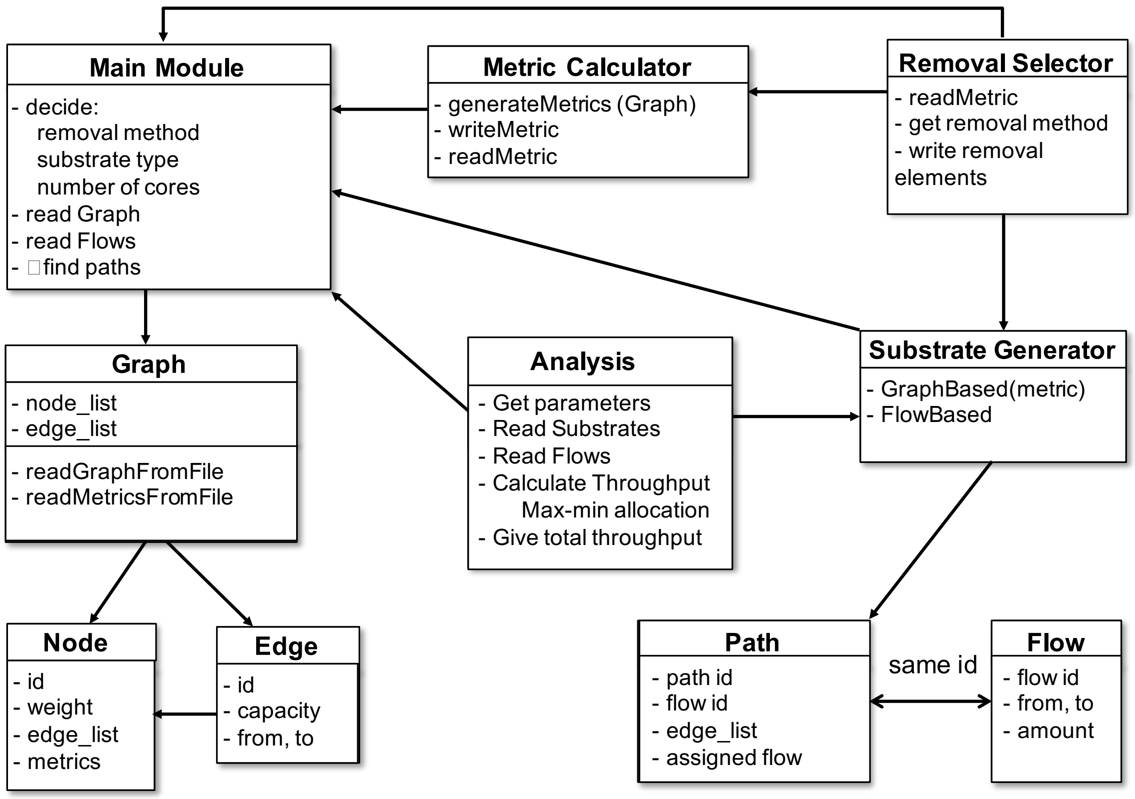


Figure 4.5: Class diagram for substrate generator

selector module defines which nodes/edges will not exist for the next substrate according to the given removal method. Substrate generator module gets these outputs that generated by first two modules and creates substrates cumulatively or independently according to removal percentage. This module stores all substrates in different files. Lastly, the analysis module calculates the total throughput can be sent over the substrates by using max-min allocation when edge sharing occurs. All these modules use the main module that includes the definition of common classes and functions.

We tested our heuristics on six Rocketfuel topologies that have different characteristics [93]. Table 4.3 presents the number of nodes, number of edges, maximum degree, average degree, average path length, clustering coefficient, and assortativity of the network graphs.

In Rocketfuel topologies, link capacities are not provided, but delay-based edge weight is provided. In our analysis, we determine link capacities inversely proportional to the link weights in the Rocketfuel data set. In order to calculate the throughput attained by subgraphs/substrates, we assign data loads incrementally. First, we send maximal data on the first subgraph, re-arrange capacities and use the next subgraph(s) to send the rest of the data. A finer grained model could look for maximum flow in the underlying subgraphs.

We performed a static analysis that calculates the total throughput for a given network. We compared the MCPR against the currently used single shortest path routing to analyze the throughput across the network. We generated network flow between all node pairs based on the gravity model between the point of presences (PoPs) of in the Rocketfuel topologies. In the gravity model [6], flow demands are calculated as the product of populations divided by the square of the geo-distance between two PoP locations. We assumed the link capacities inversely proportional to the link weights that were provided by Rocketfuel. We used max-min allocation to determine the end-to-end rates the flows will attain. We normalized the flow rates based on the smallest flow rate, and, so our throughput results are shown in normalized units.

We tested both *cumulative* and *independent* approaches to generate substrates. In the independent generation, all substrate graphs are generated from the actual topology. In the cumulative substrate generation, each substrate is forked from the previously created one, and hence following graphs have a larger portion of the network removed.

We created a new network model including flow patterns, estimated link ca-

Table 4.3: Characteristics of network topologies

Network	Nodes	Edges	Max Deg	Avrg Deg	Avrg Path Len	Cluster. Coeff.	Assortativity
AboveNet	141	922	40	13.1	3.62	0.269	0.698
Ebone	87	403	51	9.3	3.90	0.299	0.357
Exodus	79	352	24	8.9	3.94	0.286	0.749
SprintLink	315	2333	90	14.8	3.89	0.331	0.387
Telstra	108	368	92	6.8	2.89	0.171	0.006
Tiscali	161	874	406	10.9	2.31	0.072	-0.063

pacities, flow demands of nodes. Instead of using different metrics to calculate total throughput, our model is evaluated based on how many units can be transferred. Thus, we can convert that model to other metric systems.

Topology: Rocketfuel topologies have information about nodes and links. But, link capacities are not measured. In our analysis, we get link capacities inversely proportional to the link weight information coming from Rocketfuel instead of trying to estimate link capacities with actual real capacities.

Flow patterns: In our analysis, gravity-based flow patterns are calculated. In a gravity-based model, we use actual population and geo-location of the cities where nodes are located. Flow demands are calculated based on the gravity model, which is the product of populations divided by the square of the geo-distance between the source and destination locations of the flows. We also normalized the estimated flow rates by the minimum rate attained by any flow. In both models, data flows are full-duplex for each pair of nodes.

Unit equalization: Link capacities are calculated based on weight metrics coming from Rocketfuel and flow demands are given in terms of the minimum flow rate. Thus, unit equalization between the link capacities and the flow rates is needed to calculate the total throughput. We performed this matching process during the max-min allocation when flows share link capacities. Instead of equal distribution, we allocated a link's capacity based on rates/demands of the flows which are crossing through that link. In homogeneous traffic pattern, capacity sharing will be equal. However, in the gravity-based model, each flow will get their portion based on their demands.

Load balancing on substrates: Our proposed method reduces multi-path calculation problem to another problem: *finding optimal subsets of possible substrates to achieve maximum total throughput sent over network.* In this model, it is needed to determine the load on each substrate. In our analysis, we assigned data loads incrementally on the substrates, starting from Substrate 0. First, we try to send all data on a substrate, re-arrange capacities, and use the next substrate to send the rest of data amount. For instance, let us assume that we have two cores and one flow. We try to send the whole data on Substrate 1. If there is still data packets needed to be sent, we will send them on Substrate 2.

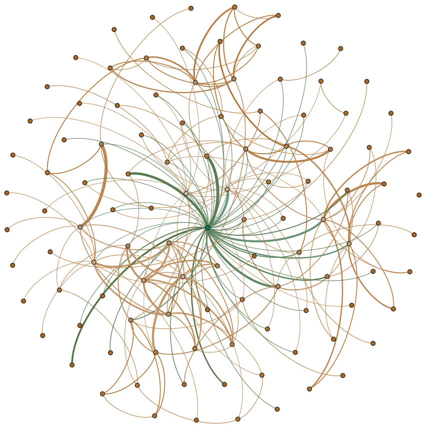
Single Failure Analysis: MCPR is also tested when a failure occurs to show robustness under dynamic networks. Under single failure analysis, nodes are down one by one, and total throughput is calculated for each scenario at the time of failure. Then, the average amount of throughput is computed to understand the performance of MCPR in the case of losing a node temporarily.

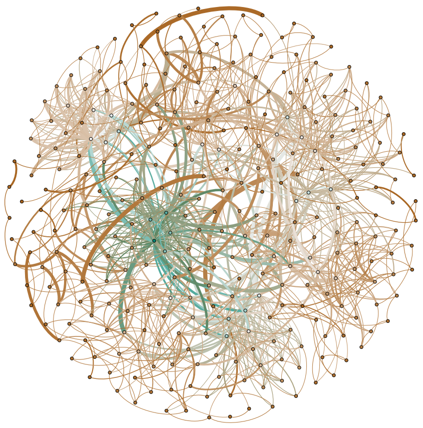
We compare the heuristics against the single shortest path according to the total throughput achieved from the generated substrates. In each scenario, we selected different topologies. We generated gravity distributed model (*inter-datacenter traffic*) flow sets for various topologies. We tested both *cumulative* and *independent* approaches to generate substrates. In the independent generation, all substrate graphs are generated from the actual topology. In the cumulative generation, each substrate is forked from the previously created one and hence later graphs have a larger portion of the network removed. We also test our performance in different percentage of node removals and different removal approaches.

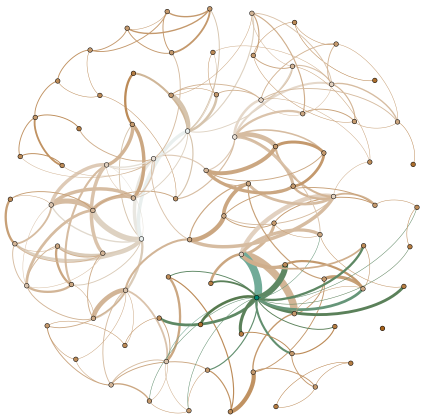
4.7 Evaluation of Heuristics

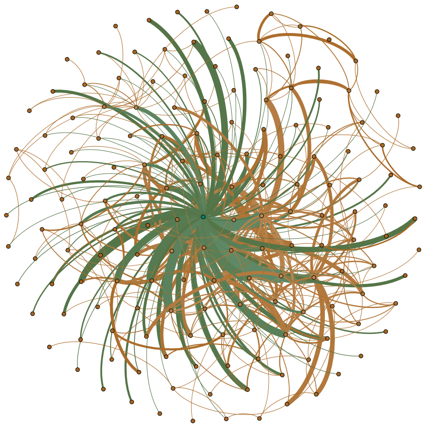
In this section we provide the comparison between multi-core parallel routing heuristics and the shortest path routing. We first provide a summary of key results. Although all techniques are better than the single core, HF (a.k.a. a measure of congestion on the nodes/links) seems to be giving much better results as we expected. HF identifies the edges that are most struck by shortest paths the flows use, and hence attains a better balancing of traffic loads on substrates, each one of which uses shortest-paths for routing. On the other hand, graph-based approaches, although they perform better than the single shortest path, can not attain significant improvements as nodes are being eliminated instead of edges. Note that, graph-based heuristics are pre-computed methods that generate substrates by only using the network topology information, and they do not require recalculation of the substrates until topology changes.

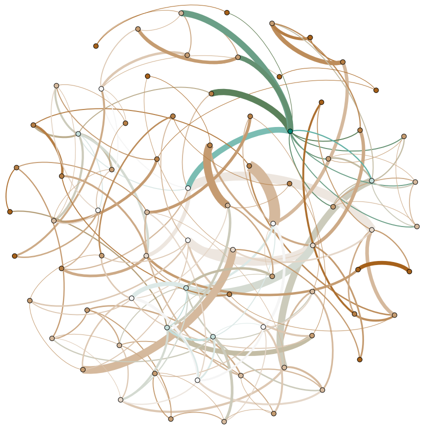
Under the weighted traffic pattern, i.e., inter-data center traffic, *parallel routing heuristics achieve higher total throughput* in comparison to the single core routing. Also, they outperform even further as the number of cores and the offered load (i.e., the number of source-destination flows) increase. *Edge removal clearly outperforms node removal techniques since it works at the finer granularity of edges rather than nodes.* However, as the number of flows and the network gets larger, the flow-based heuristics are computationally harder as they will have to cope with more dynamism


(a) Telstra - 1221


(b) SprintLink - 1239


(a) Ebone - 1755


(b) Tiscali - 3257


(a) Exodus - 3967

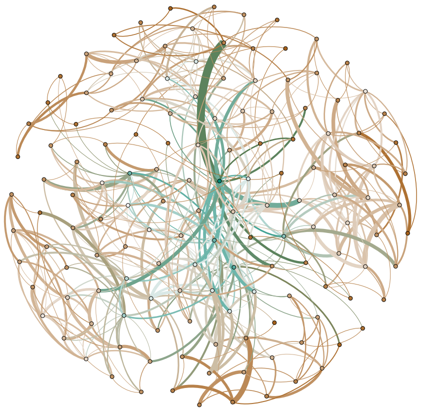

(b) AboveNet - 6461

Figure 4.6: Visualization of Rocketfuel topologies

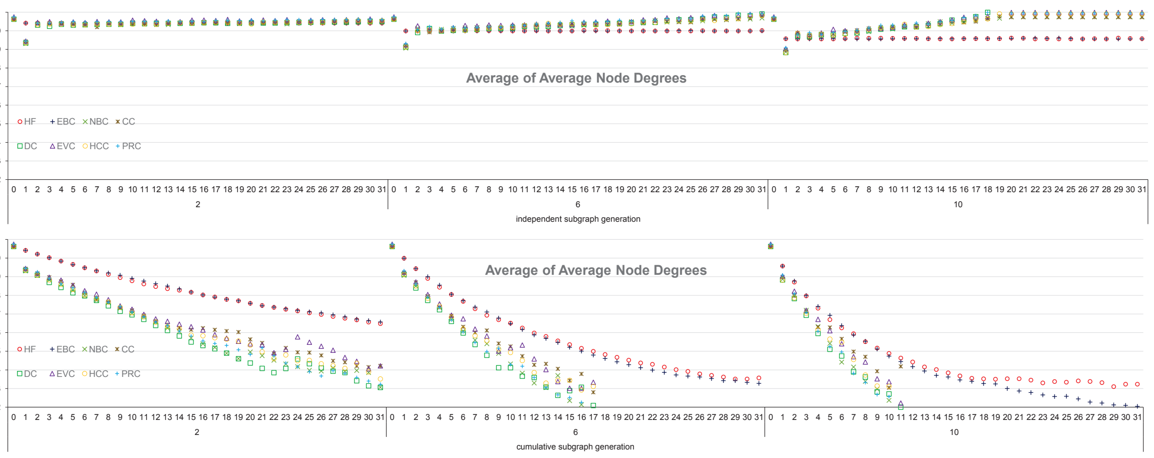


Figure 4.7: Average of average node degree of the substrates for 2%, 6%, and 10% removal

via repeated shortest path calculations on each substrate. Also, in the long term, substrates generation process must be repeated periodically depending on the changes in the flow sets.

4.7.1 Analysis of Substrates

In this section, we perform a detailed analysis of the substrates generated by each of the heuristics. Figure 4.7 presents the average of the average node degree, Figure 4.8 gives the average maximum node degree, and Figure 4.9 shows the average clustering coefficient of all created substrates. We show the average resulsts for the independent as well as the cumulative removal on all Rocketfuel topologies. Cumulative substrates generation significantly changes graph characteristic and sometimes removes connectivity in the substrate. In general, node centrality heuristics disrupt the network in substrates more than edge centrality and HF heuristics. Node centrality heuristics have almost the same effect in terms of topology disruption. As expected, we observe that small removal percentages have small effects on substrate characteristics. How-

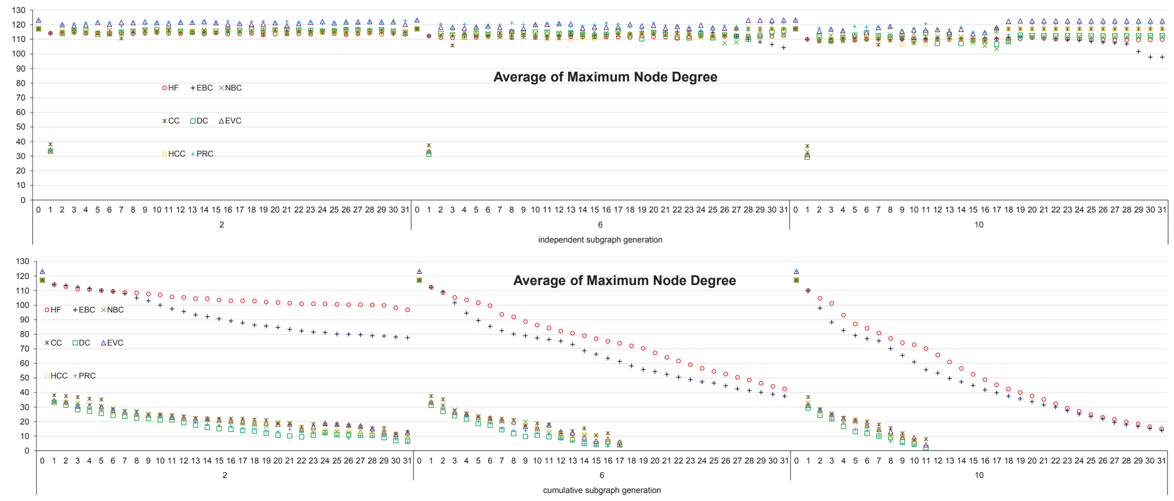


Figure 4.8: Average of maximum node degree of the substrates for 2%, 6%, and 10% removal

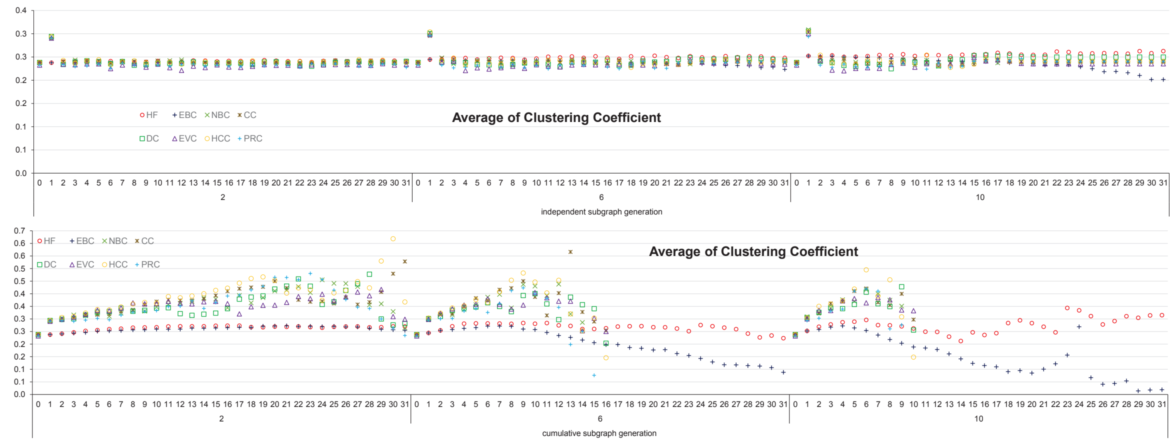


Figure 4.9: Average of clustering coefficient of the substrates for 2%, 6%, and 10% removal

ever, in larger removal percentages, multi-core parallel routing heuristics are not able to generate different substrates, and start to generate too similar substrates after a while. As practical observation, note that the removal percentage should be less than $100/numberOfCores$ as, after this point, there are no more nodes/edges to remove in the substrates.

In node centrality heuristics, graph-based heuristics, the first substrate excludes the most centralized nodes, which causes a significant decrease in the average and maximum node degree. After a point, the heuristics start to remove periphery nodes from the network, and this increases the average node degree. This indicates that later substrates might not improve the performance as removing periphery nodes does not contribute to balance the load. Average node degree, however, is not affected much by the edge removal heuristics. As a result, edge removal heuristics perform better than node centralities (in Figure 4.12). We can observe the same pattern for maximum node degree.

As HF is dynamically generating new substrates, it tends to remove the bridge edges in the network. This causes an increase in the clustering coefficient of new substrates. As EBC, edge betweenness centrality, removes the highest centrality edges in initial substrates, the bridges remain in the subsequent substrates. This leads to a decrease in clustering as central group of nodes are removed in subsequent substrates.

On the other hand, in cumulative approach, average and maximum node degrees reduce significantly. Clustering increases with small removal percentages but, when removal percentage is increased, clustering also plummets. For further results, please see Appendix A.3.

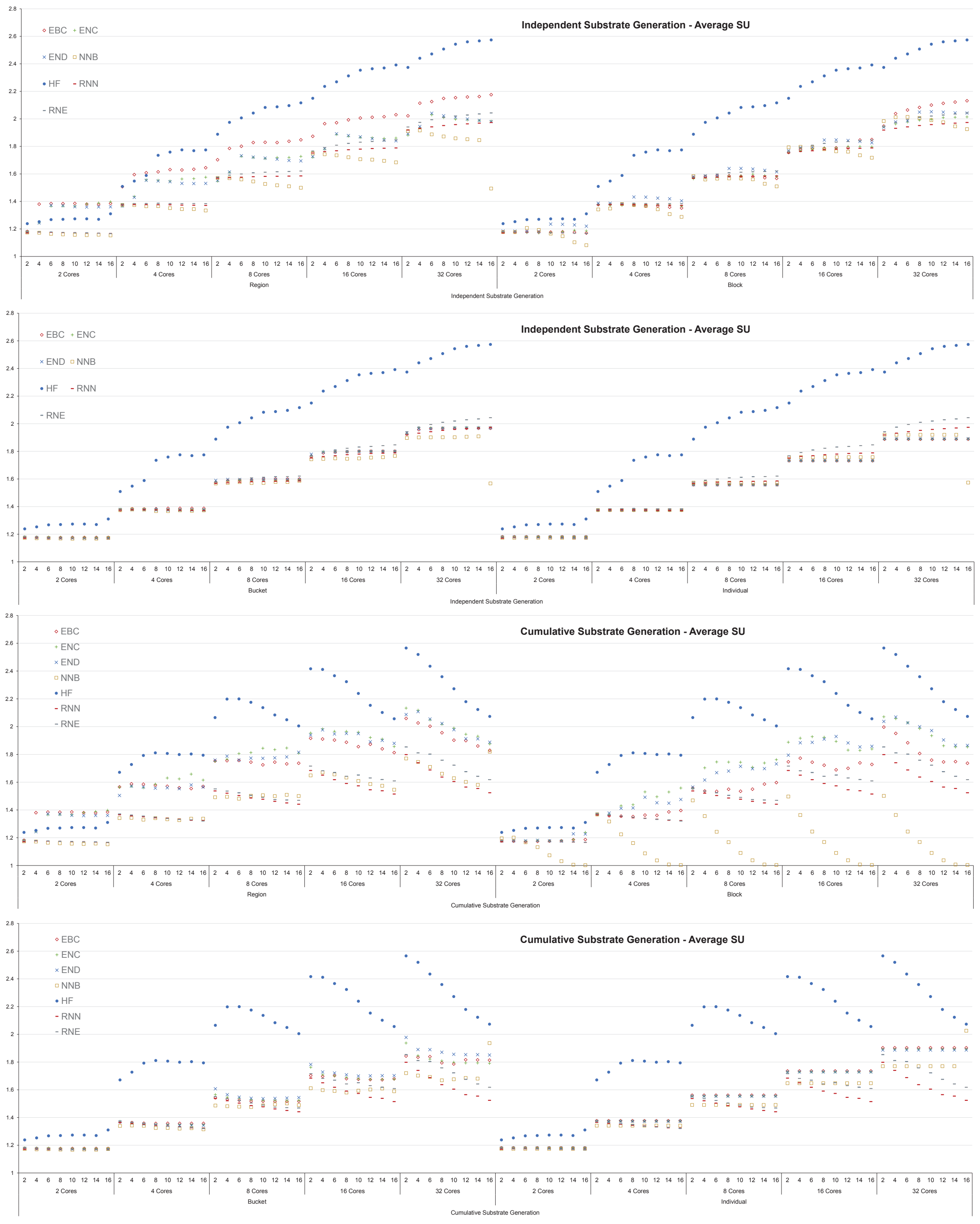


Figure 4.10: Speedups for Removal heuristics (Region, Block, Bucket and Individual approaches) for both independent and cumulative substrate generation

4.7.2 Analysis of Removal Heuristics

In this section, we compare the performance of the removal heuristics for Bucket, Region, Individual, and Block approaches. We give results for a couple of metrics including EBC, ENC, END, NNB, HF, RNN, and RNE. We show HF and RNN to determine upper and lower bounds of multi-core parallel routing performance, as shown in Figure 4.10. As we discussed in the previous section, in cumulative substrate generation, the new substrate are being disrupted rapidly which sets back the total throughput. However, in the Individual method, we remove only one element at each step that lessen deformation on the substrates but improve the performance at a low level. As shown in Figure 4.10, Bucket removal performs better and reduces the topology disruption for the cumulative method, but it does not entirely clear congested spots at the same time. So, it does not have enough contribution to the amount of total throughput. Block removes a block of nodes/edges while creating a new substrate that removes the most important elements in first substrates, and this gives good improvement in throughput. However, in later substrates, it removes low-scored elements which reduces the performance significantly. Finally, Region removal outperforms all other heuristics because of its sense of clearing the neighborhood of the possible hot spots at the same time. On the other hand, in the independent removal method, Region and Block perform better similar to the cumulative process. The Individual approach has moderate improvement, and Bucket does not have significant contribution to the performance. In Bucket, we remove some elements in each block that are not able to clear the whole block and balance the load across the blocks. When we remove the whole block to generate a new substrate, we reduce the load on most used nodes synchronously which improves the performance of load balancing. Finally, Region clears the neighborhood, so the overloaded nodes and the regions around those nodes will not be on the newly generated substrates. When we increase the number of cores, for the later substrates, Region will remove the area of the low-scored nodes, which affects the performance severely. For further results, please see Appendix A.2.

4.7.3 Performance Comparison

In this section, we evaluate the performance of centrality based heuristics to solve substrate generation for multi-core parallel routing with Block Removal method.

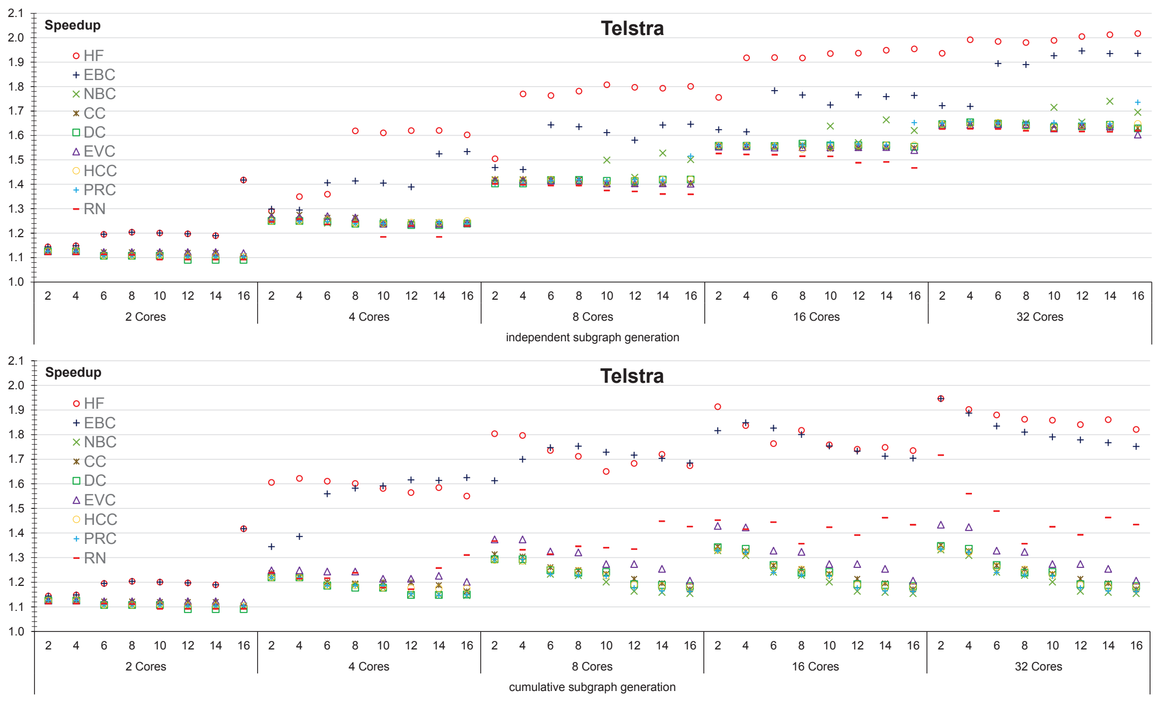


Figure 4.11: Performance of MCPR heuristics for the Telstra topology

To analyze the speedup of heuristics, we show the performance of each against the single shortest path as a baseline. Note that, *highest flow* heuristic is expected to provide the best speedup as it dynamically adopts to the network flow in substrates.

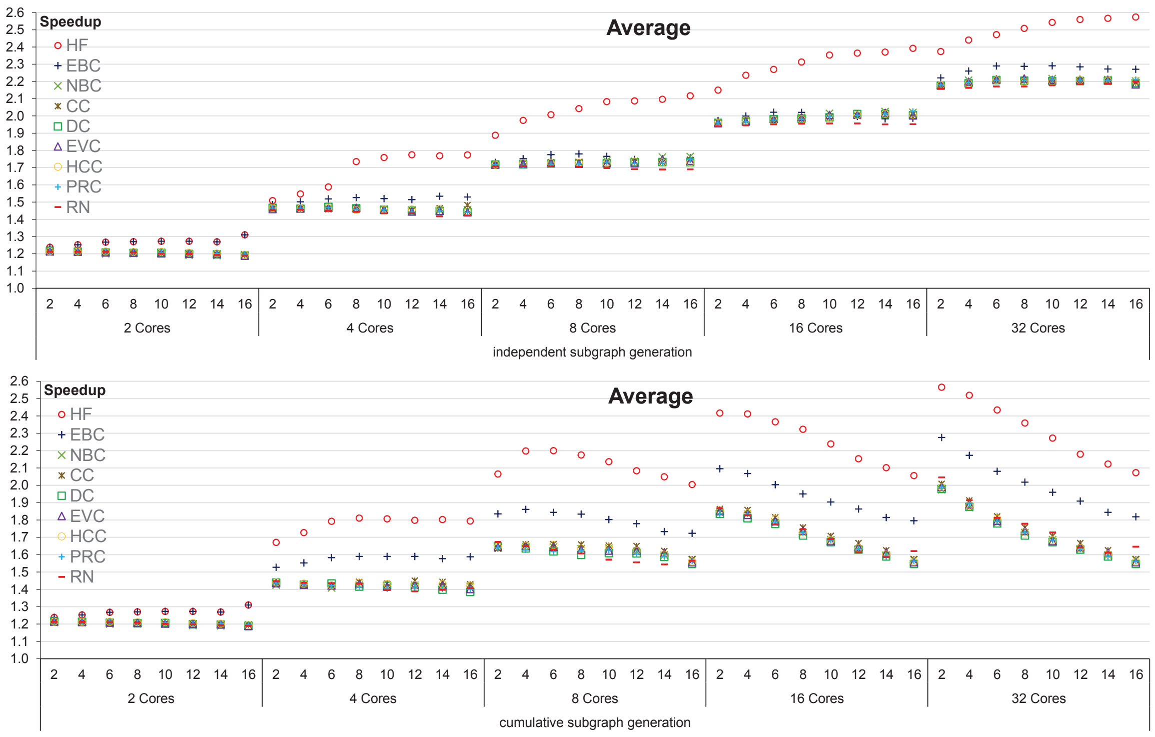


Figure 4.12: Average performance of MCPR heuristics for all topologies

Similarly, *random network* is supposed to provide the lower bound of speed-up as it randomly removes nodes from the substrates.

Figure 4.11 presents the performance gain of substrate generation heuristics compared to the single shortest path for Telstra topology. In the Figure, x-axis has two different indicators. The first line on the top shows the removal percentage of network nodes/links (which is incremented between 2% and 16% by 2). The second line gives the number of cores (i.e., 2, 4, 8, 16, 32 cores) that will generate substrates. The last line indicates the substrate creation approach, i.e., independent and cumulative. Test results are given for gravity-based flow pattern which is proper for inter-data center traffic flows. From Figure 4.11, we observe that multi-core parallel routing performs much better than the single core routing (ranging from 1.2 times with 2 cores to 2.0 times with 32 cores).

Figure 4.12 presents the average performance gain of substrate generation heuristics compared to the single shortest path for all topologies. Each heuristic yields a higher performance than the single shortest path approach indicating even with a poorly chosen heuristic (such as random node removal), multi-core parallel routing can produce better throughput than traditional routing. We observe that multi-core parallel routing heuristics improve with the number of cores.

While in some instances of independent substrate generation (e.g., 16% removal with 32 cores), node centrality heuristics perform slightly worse than random network generation, centrality heuristics overall yield better results than random removals. Additionally, highest flow heuristic produces best throughput speed-up in a majority of the instances as it adopts the substrate generation to network flows.

In both independent and cumulative methods, a specified percent of nodes/edges are removed in the next substrate. As a result, the independent method removes fewer elements in generating the next substrate. In Figure 4.12, we observe that the cumulative method performs worse with higher node/edge removal rates in subsequent substrates. As there are the greater number of nodes/edges ignored in the later substrates, they are not able to yield viable e2e paths. Hence, node centrality metrics occasionally perform worse than random node removal. On the other hand, with independent substrate generation, higher removal rates yield better performance for the HF approach that adjusts the substrates to the networks' flow. Hence, in general with a greater number of cores, one should utilize the independent edge removal in the substrates to obtain viable paths that would provide higher throughput in the network. For further results, please see Appendix A.1.

In the highest flow approach, a small number of removals with a higher number of cores provides better throughput because HF removes hot spot(s) from the subsequent substrates. Note that, however, highest flow approach needs to recalculate substrates after each flow change. Hence, it is viable to use edge betweenness centrality that produces the best among centrality heuristics. Overall, for a large topology, edge centrality heuristics are better with a high number of cores and independent removal percentage within 8% to 10%.

When analyzing individual topologies, we observe that the best overall performance is achieved with the SprintLink and AboveNet, two of the largest networks in the data set. However, the third biggest network, Tiscali shows the most substantial difference between highest flow and other centrality metrics. We believe this is

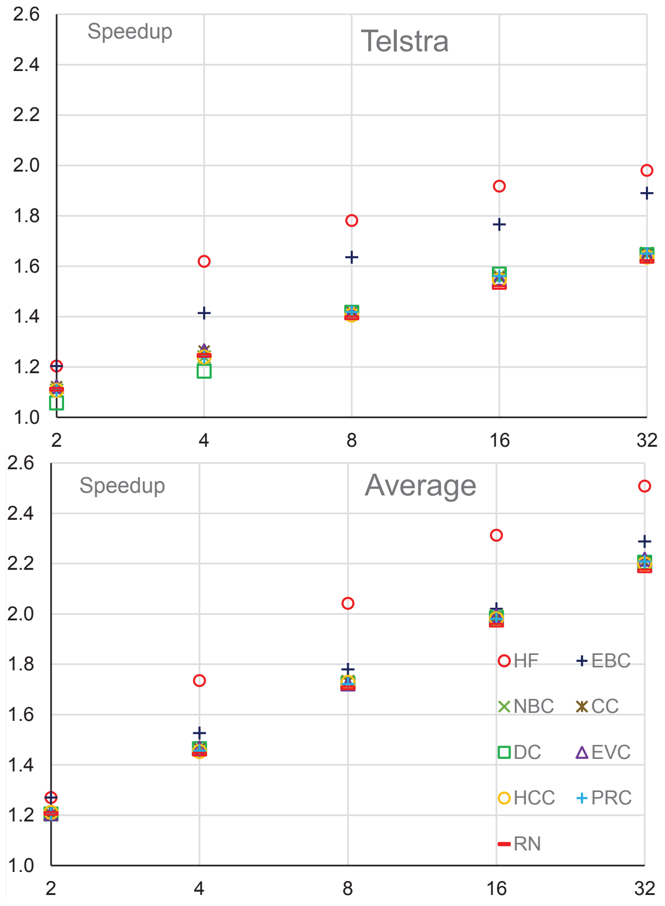


Figure 4.13: Effect of the number of cores with 8% independent removal (log-scale)

in part due to Tiscali graphs' slightly disassortative behavior where highest degree nodes form a core of the network. Hence, centrality based removals are not able to produce viable substrates.

Figure 4.13 presents the effect of the number of cores in the networks with %8 independent edge removal. As observed in both the Telstra and overall average results, the speedup performance improves with the number of cores. However, this improvement is sub-linear as the number of cores increases.

4.7.4 Analysis of Network Dynamics

In this section, we analyze the Multi-Core Parallel Routing heuristics under single node/edge failures. Failure analysis helps to determine the robustness of the approach under dynamic network conditions. Under single failure analysis, nodes or edges are eliminated one by one and the total throughput is calculated for each graph with the failed node/edge. We assume that the substrate graphs stay still even though a node/edge fails. This allows us to see how robust the substrate graphs are against single failures. In our simulation, we fail all the nodes and edges one by one separately and compute the total throughput achieved on the given substrates. We use the same Rocketfuel topologies and the same substrates used in Section 4.7.1. We compare the average results to understand the single node failures and single edge failures.

Figure 4.14 presents the average throughput change when 6% and 8% removal rate is used with closeness centrality and highest flow approaches in the AboveNet topology. Multi-Core Parallel Routing uses only single shortest path calculation, even though when one of the nodes/edges is down. In this case Multi-Core Parallel Routing calculates new shortest paths on respective substrates instead of calculating the whole end-to-end multiple paths. Overall, we observe that performance of multi-core parallel routing is reduced by 0.4% when a node or edge failure occurs. When considering the node failure, both NCC and HF approaches loose 0.7% and 0.6% throughput on average for 6% and 8% removal rates, respectively. When considering the edge failures, NCC looses 0.2% throughput on average while HF looses 0.1% throughput with 6% removal rate. Similarly, with 8% removal rate, the average throughput loss is 0.1%. Note that HF even improves the throughput with edge failures under independent substrate graph generation with 8, 16 and 32 cores. This is due to the fact that HF adjusts to the network condition and, with failure of an edge, it was able to find alternative paths that slightly improved the overall throughput.

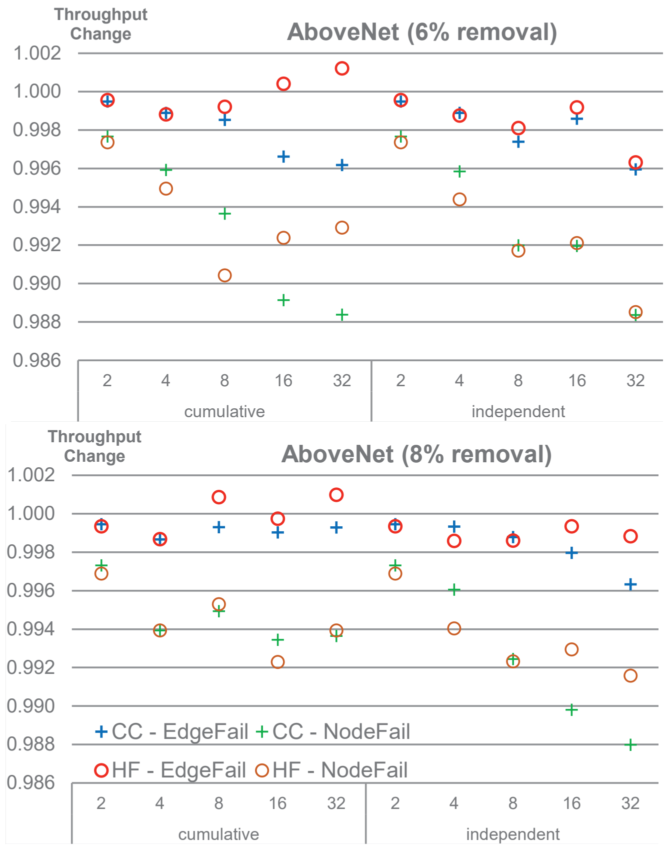


Figure 4.14: Performance of AboveNet with node/edge failures

Chapter 5

Parallel Routing Under Heterogenous Core Distribution

There are different kinds of routers manufactured by various companies with the several numbers of CPU cores. While building a new system, there could be different requirements or limitations such as cost-effectiveness issues or geographical restrictions. Therefore, all devices on the network wouldn't either work or be the same. Since updating most of the systems with new technological devices, is not possible in a short period of time, the homogeneity of the network system will be broken. As a result, we need to find an appropriate MCPR solution that can also work in the heterogeneous environment. MCPR could run on not only homogeneous systems but also on the systems that have routers with various numbers of cores.

Recall that MCPR empowers each core to handle one of the generated substrates to find a feasible manner for multi-path routing. Therefore, each core will be having different topology information and every first core of the routers in the system will have the same topology. Although this attains a basic consistency and connectivity in routing, in the heterogeneous systems, not all substrates may be efficiently hosted by some of the routers if their number of cores is fewer than the count of generated substrates. For example, when four substrates are created, third and fourth substrates cannot be assigned to a separate and dedicated core at a two cored router. The problem with such situations is that, when two cored router has a flow coming from a substrate that is not assigned to a separate core (e.g., third one), the forwarding of the data packets for that flow will have to be handled by one of the two cores that are already assigned to another substrate (e.g., the first or the second substrates). This means that the flows on these unassigned substrates will have to share the cores with other substrates. Such sharing of the cores will reduce the forwarding (and control plane actions) speed and deteriorate the overall performance gains from MCPR.

We will propose two heuristics to accommodate parallel routing with the heterogeneous count of router cores and experiment those solutions with the flow simulator. In our first technique, we just gave up on some routers that cannot manage more substrates during the substrate generation process. The next heuristic is to create an average number of cores with an estimate of additional processing delay due to lack of sufficient number of cores on some of the routers.

5.1 Heterogeneous Heuristics

To design MCPR heuristics that can run over routers with heterogeneous count of cores, we try to estimate the additional load on the routers due to excessive substrates than their core count. The first estimation we have is the router that has a bigger degree value might be carrying more cores since those routers are needed to be more powerful for processing more data than others. So, the distribution of the core count over routers in a given topology is built depending on the degree centrality of each router in our network model. We use the same graph-based heuristics for heterogeneous scenarios with two different additional initial heuristics. These extra features are needed to solve the substrate-core matching problems when all the cores of a router had already been assigned to a substrate and a new substrate is created.

Additional Node Elimination: In this heuristic, after deciding which nodes will be removed to generate a new substrate, some nodes are also maxed out if their core number is fewer than the substrate count. Therefore, those nodes cannot be included on the next substrate generations because they are not able to process the data coming from the excessive substrates. Note that this is a preventive approach, in that it eliminates the possibility of having excessive substrates than the number of cores on a router.

Average Number of Cores - Normal: The second method is to create new substrates up to an average number of cores and consider the additional processing delay at the routers with excessive substrates. Since we did not implement a packet simulator, we decided to simulate the worst case scenario for the data processing delay. In order to evaluate the performance of this heuristic, we drop the data flow speed by some amount if there wasn't enough cores on the router in proportion to how many times the router's interface(s) was used. Following the observations on the performance of MPTCP, we reduce the data flow speeds for the overloaded cores proportionally [80] [25].

To model the data flow speed reduction, we generate three different versions of the flow speed loss based on delay-tolerant and loss-tolerant applications. We assign a (down)scaling factor to each node based on the excessiveness of the substrates, and then, we calculate the scaling factor for a given path. For a flow traversing several routers, we merge the scaling factors of those router to come up with one scaling factor for the end-to-end flow's data flow speed. For loss-tolerant applications, we product all the nodes' scaling factors to calculate how much extra data will be lost (i.e., the reduction in the data flow speed) for a given path. For delay-tolerant applications, we use the arithmetic average of the scaling factors to determine the amount of data flow speed reduction. Finally, we use harmonic average of the scaling factors to understand the heuristic's performance for both delay and loss sensitive applications.

5.2 Simulation Environment

We use a modified version of the implemented simulation environment explained in Chapter 4. We added processing delay for each interface and updated the simulation to calculate the throughput for a delay, loss and mixed tolerant applications. Similar to the homogeneous cases in Chapter 4, we evaluate our flow tests with max-min allocation method to resolve the competition among the flows. But, if there is a data stream coming from an excessive substrate, the router interfaces would be working more to process the data on a core that is not already assigned. So, we added a new feature that computes how many units will be lost because of this processing delay. We reduce the performance of the interface depending on how many additional flows will be processed by that interface. For instance, if there is a two core router trying to handle four substrates, then both cores will be working two times more to process the data flows because of sharing their capacity with other two excessive substrates. In our evaluation we are taking a pessimistic approach, and even if the cores are not used simultaneously, we are downscaling the data flow speeds as a worst case scenario. In a real environment, less data would be lost due to the non-overlapping use of the cores, but the worst case is simulated in our tests.

Similarly, we evaluate our heuristics on all Rocketfuel data set, and our results are shown in normalized unit flow rates. Rocketfuel topologies give the topology information and latencies on the links. However, there is no information about the number of cores on each router. So, it is needed to create a new model simulating the core distribution among the routers. We assume that the most used routers should be stronger than the others to process the data passing through them. We create usage model of the routers with geo-location and population. We use 2, 4, 8, 16, and 32 core routers, and we put an equal amount of each router into the topology. For example, when a topology has 25 routers, we assume that the top 5 most-used, based on their population, routers have 32 cores and following top 5 routers have 16 cores and so on. Our comparison baseline is single shortest path according to the total throughput gained.

5.3 Evaluation Results

We test all topologies for heterogeneous scenarios with a selected set of parameters for core and removal percentages under both independent and cumulative substrate generation methods. However, we just give the performance of EBC, ENB, and NNB to show how multi-core parallel routing performs under heterogeneous core distribution. We also show the average speedup among all Rocketfuel topologies.

We show the average performance of both Eliminated and the additional processing loss (Normal) methods. We indicate the upper and lower bounds with a

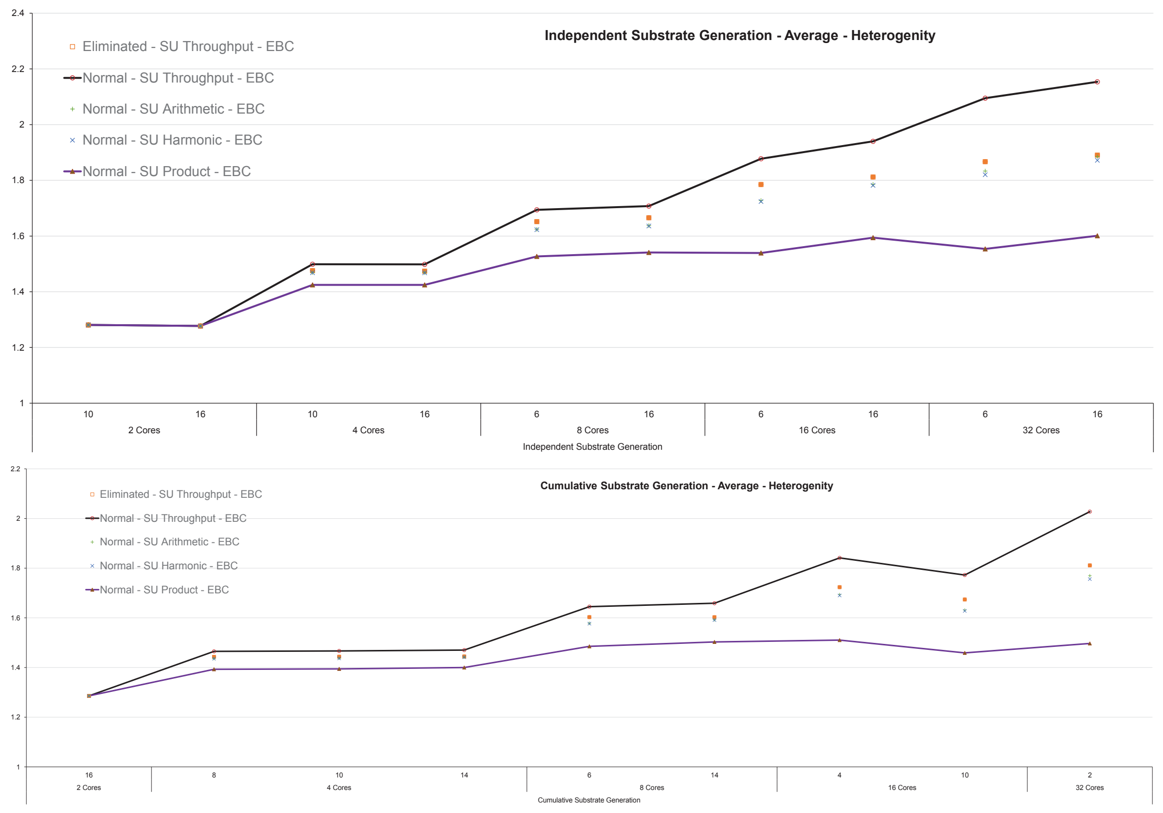


Figure 5.1: Multi-core parallel routing performance with Edge Betweenness Centrality metric under the heterogenous core distribution for both cumulative and independent substrate generation approaches

line. The Normal Method throughput is the upper limit of the metric's performance when all nodes have enough capability to process all of the assigned substrates. The Eliminated Method is a heuristic for the heterogeneous scenarios in such a way that it eliminates the nodes with fewer cores than the number of substrates being assigned to them. Since this elimination is done after the substrate is created, it means that the technique removes additional elements after the substrate is created and this reduces the total throughput. The Normal Method with Arithmetic, Harmonic and Product downscaling factors show the speedup, each with a different type of sensitivity to loss

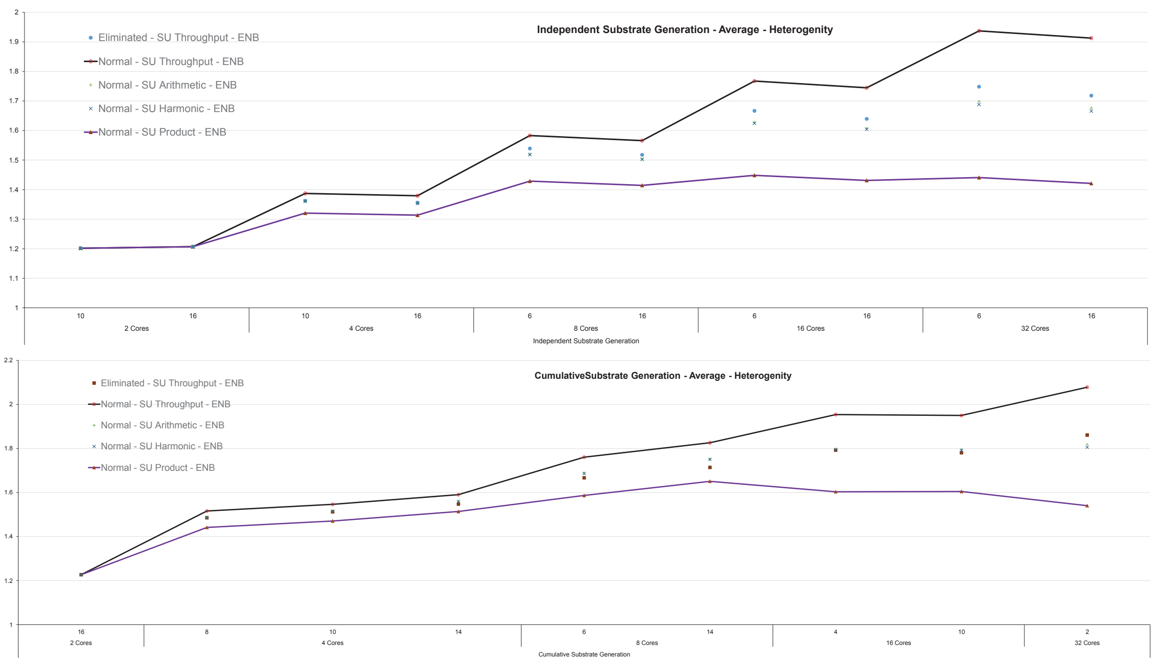


Figure 5.2: Multi-core parallel routing performance with Edge Node Betweenness metric under the heterogenous core distribution for both cumulative and independent substrate generation approaches

and delay. Although the Normal Product is the lower bound of the multi-core parallel routing, it still has an improvement to single shortest path baseline.

Since we got better performance with independent substrate generation technique as shown in Chapter 4, we get a better speed up for independent substrate production for heterogeneous scenarios too. For both of the heuristic techniques, heterogeneous core distribution reduces the performance of multi-core parallel routing, but it still outperforms the current single shortest path routing. In the cumulative scenarios, we already remove most of the nodes from the given topology as shown in Figures 5.1 and 5.2. So, node based heuristics, i.e., NNB as given in Figure 5.3, are not affected too much like other heuristics. For all other scenarios, eliminating additional nodes performs better than other techniques. Note that, in the Normal

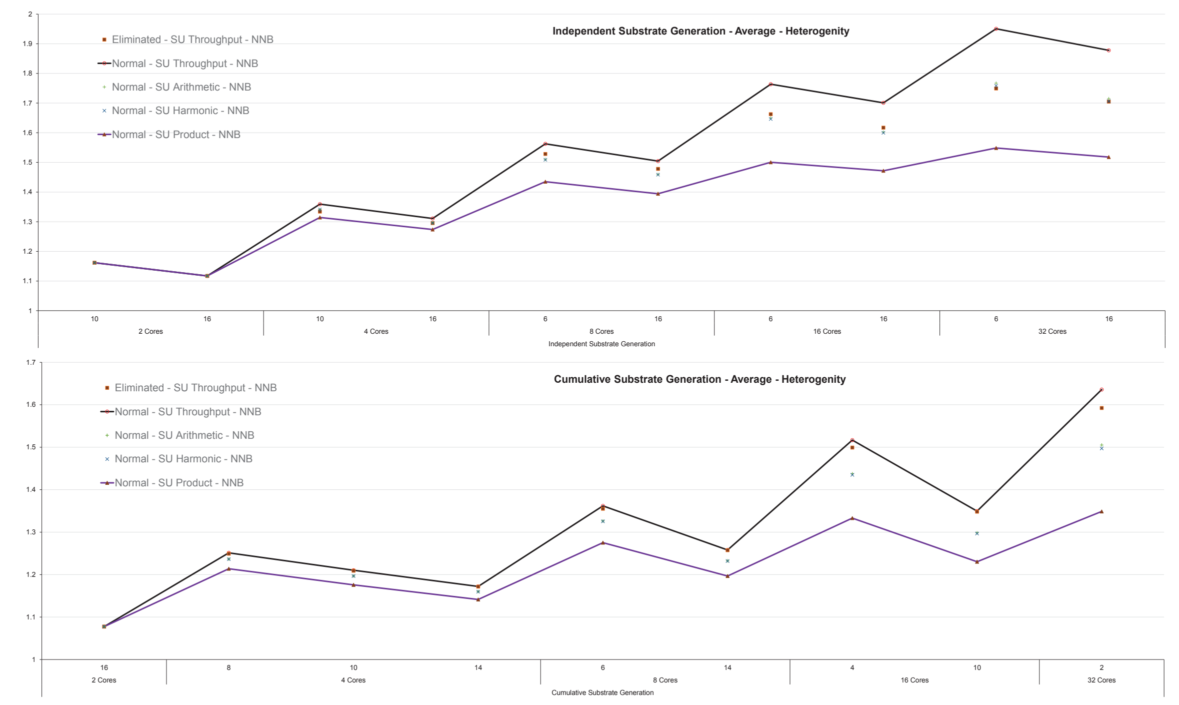


Figure 5.3: Multi-core parallel routing performance with Node Node Betweenness metric under the heterogenous core distribution for both cumulative and independent substrate generation approaches

Method, we add some additional processing delay which causes data loss for those flows going through the nodes with fewer cores than the number of substrates running on them. So, multi-core parallel routing may give better results in real settings.

Chapter 6

Conclusion and Future Work

In this dissertation, we presented a new multi-path routing framework that uses graph abstraction of the network topology and employs network centrality calculations to generate subgraphs for multi-core routers. The basic idea is to virtually slice the router topology into different subgraphs and assign each to a separate router core, which calculates the classical shortest paths on the assigned subgraph. This eases the computational complexity of multi-path routing by dividing the overall problem into smaller ones and lending each subgraph to a separate CPU core with traditional shortest path algorithms. Our evaluations showed that Multi-Core Parallel Routing achieves higher total throughput and performs better with inter-data center networking.

We proposed a new divide and conquer solution for multi-path calculation. In this method, we create new virtual topologies, named as substrates, and calculate shortest paths on each of them, instead of calculating end-to-end multiple paths. After we show the general performance of Multi-Core Parallel Routing under various conditions, we focus on the key point of this divide and conquer method. Normally, multi-path routing algorithms need to calculate end-to-end paths when topology changes including network failures. However, in MCPR, we solely calculate shortest paths which is well-known algorithm already built in most of routers. We, finally, showed how Multi-Core Parallel Routing is robust against single failures. Please note that failures can also improve the performance of MCPR in some cases.

We performed a detailed graph analysis of subgraphs on multiple topologies to determine best centrality heuristics to utilize in Multi-Core Parallel Routing (MCPR). Experimental results show that centrality based heuristics are able to increase overall throughput in the network 2+ times with 8-core routers compared to the current single shortest path approach.

By designing MCPR, we transformed the multiple path routing calculation to a subset selection problem from the set of all possible virtual topologies that can be generated from a given topology. We defined MCPR as a MAX_SUBSTRATE_SET problem and we analyzed its theoretical background. We gave the mathematical definition of the MAX_SUBSTRATE_SET and showed that MCPR is an NP-Complete problem by reducing the MAX_SUBSTRATE_SET to the well-known SUBSET_SUM problem.

We followed two approaches to designing heuristics: graph-based and flow-based. In the former approach, we omitted the most utilized node(s) from Substrate 0 to generate new substrates. In the latter approach, we omitted the most utilized edge for incrementally generating the substrates. Both approaches have advantages and disadvantages. The graph-based approach is less expensive computationally but attains lower aggregate throughput, while the latter achieves higher aggregate throughput with a larger computational cost. We also followed both cumulative and independent approaches to generate later substrates. In the independent approach, we created all the substrates from the underlying topology, however, in the cumulative approach, each substrate is produced from the previous one. We observed that the independent approach gives better throughput with higher removal amount whereas the cumulative approach performs better with small amount of removals as new substrates are generated.

We also analyzed the effects of various selection methods after finding the central nodes to remove for generating the following substrate. We showed that a dynamic method, i.e., discovering the neighborhood in each step, gains better throughput and can cause less disruptions on the later substrates. However, static methods, such as block removing and bucket removing, also outperforms single shortest path baseline, even though they can cause the significant changes on later substrates. Note that, discovering the neighborhoods of nodes dynamically will cause additional cost during the substrate generation process.

We showed the performance of Multi-Core Parallel Routing when routers have different number of cores. We created a model of heterogenous distribution of routers and tested MCPR with this scenario. In this model, some substrates not be assigned a dedicated core on the routers that have fewer cores than the substrate count. We modified our heuristics with both additional node removing and additional processing cost after creating substrates as before. We showed our results for delay-tolerant and loss-tolerant applications. MCPR attains higher throughput against the shortest path baseline under heterogeneous distribution of cores across network routers.

To sum up, in this dissertation, we proposed a new framework having higher throughput during the big data transfers, and better performance to balance the load over the network. Multi-core parallel routing framework solves the multi-path calculation problem with well-known techniques that adapts to the current systems easily. Further, proposed framework is robust and responds well to network failures because of its divide and conquer design.

Possible future work includes to create subgraphs with other search algorithms such as genetic algorithms, and improving heuristics with dynamic and static solutions. Multi-Core Parallel Routing can also be run on a test-bed or real environment to show easy adaptation and real performance. We will experiment our heuristics with end-to-end reliable transport protocols on a larger test base. Lastly, this technique might be enhanced and kernel level implementation can be done as a new protocol. The comparison between Multi-Core Parallel Routing and other multi-path routing algorithms can be investigated. In this dissertation, we focused on the routing aspects to show how Multi-Core Parallel Routing framework performs. As a possible future work, Multi-Core Parallel Routing framework can be analyzed with different aspects of networking such as by exploring its SDN implementations and the performance of Multi-Core Parallel Routing on centralized and de-centralized systems.

Appendix A

Additional Figures

A.1 Centrality Metrics

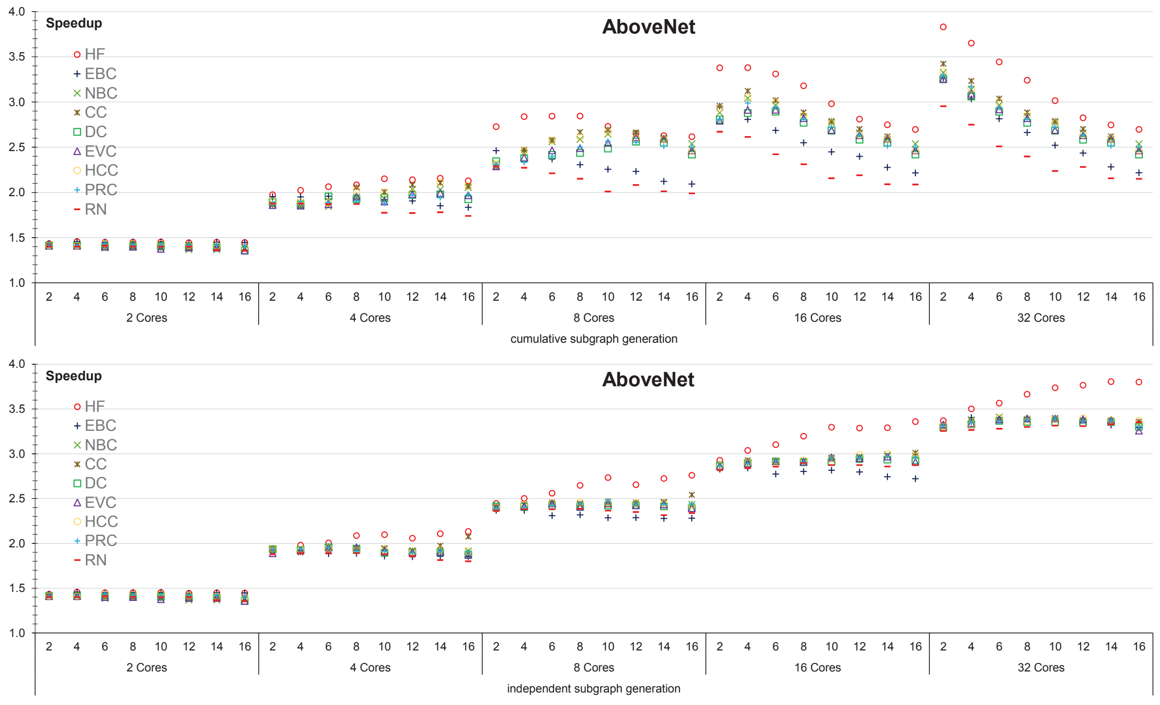


Figure A.1: Centrality results - AboveNet

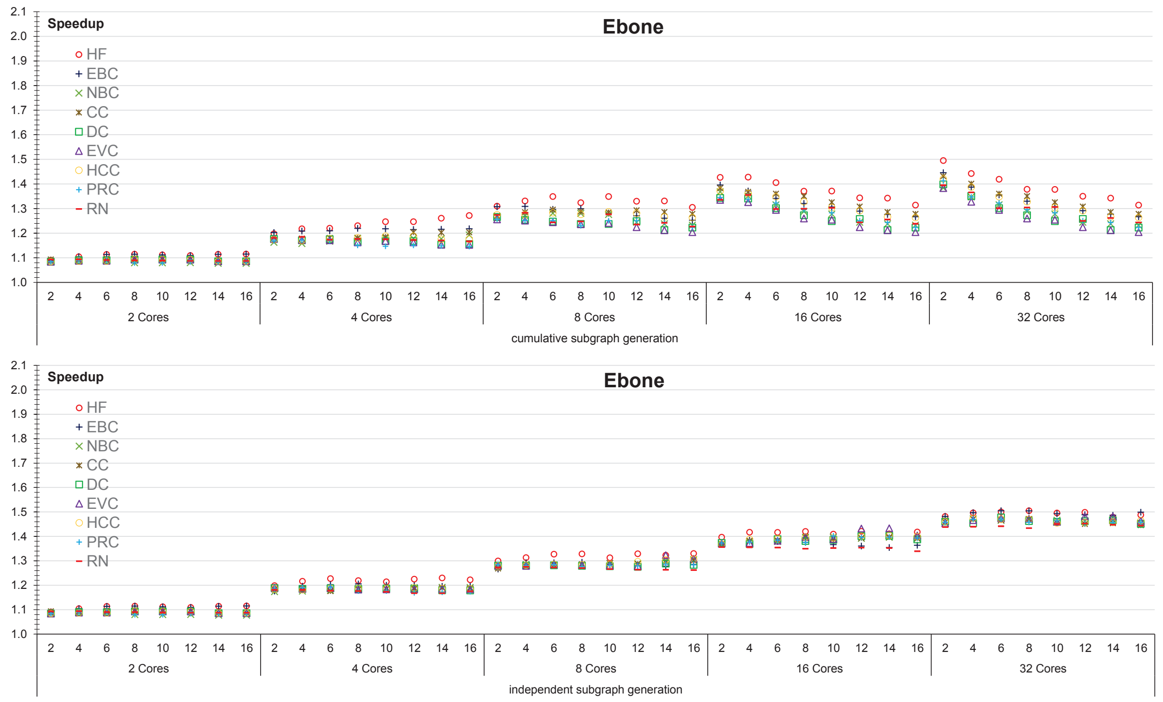


Figure A.2: Centrality results - Ebone

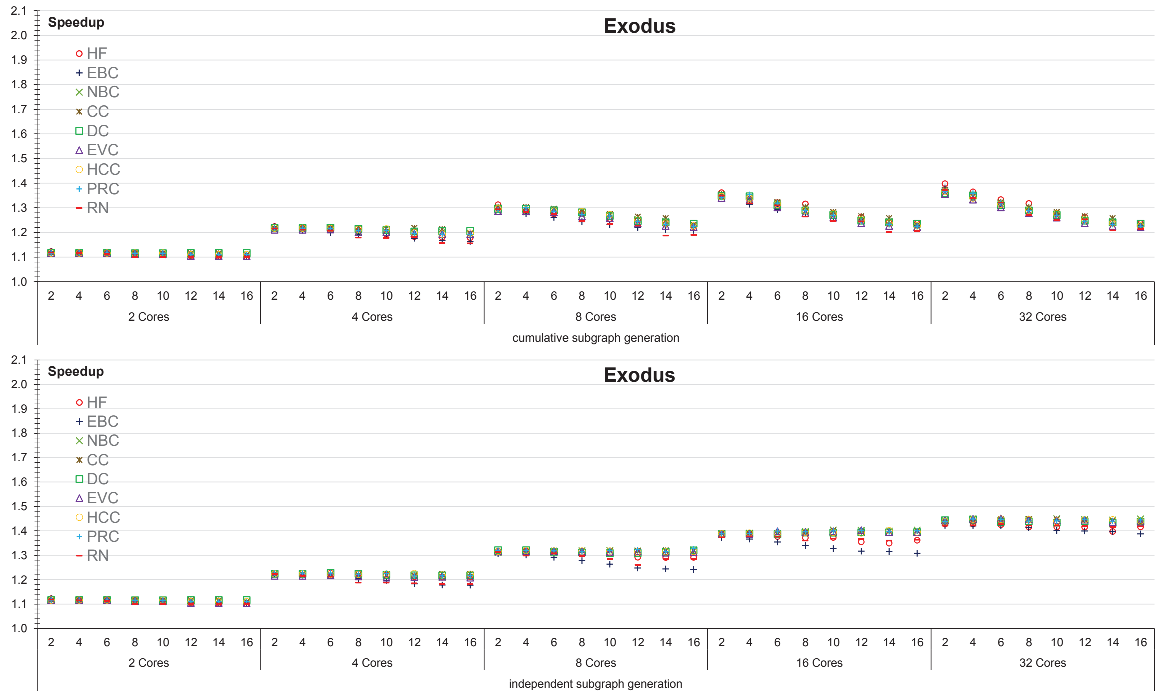


Figure A.3: Centrality results - Exodus

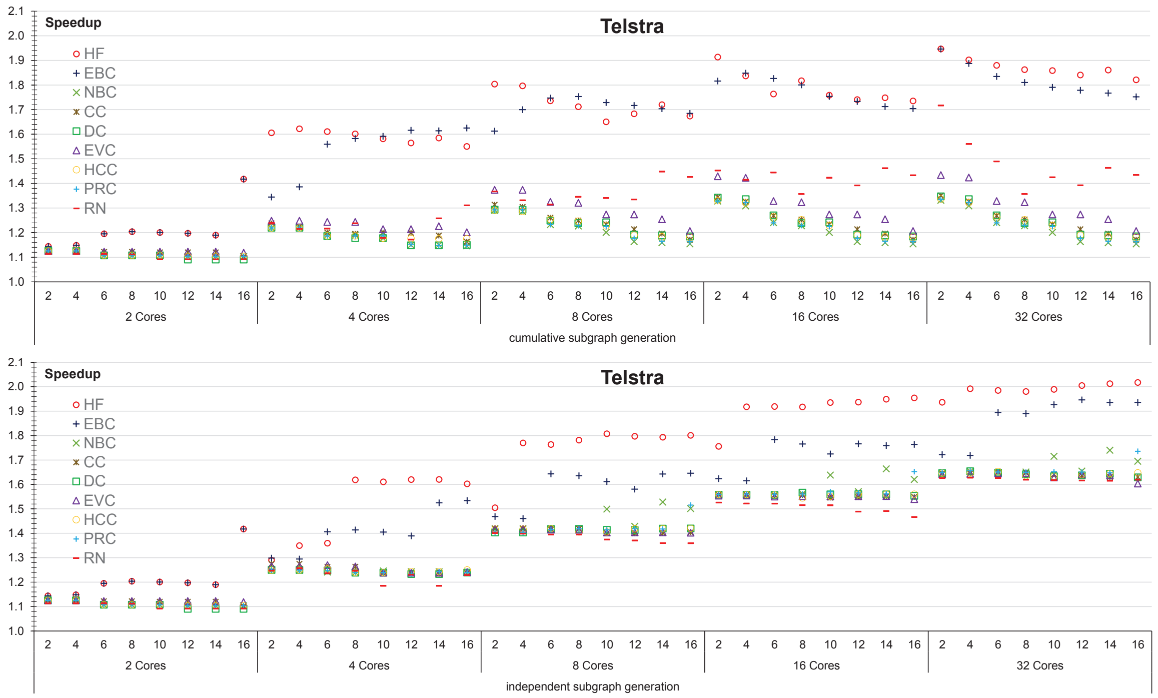


Figure A.4: Centrality results - Telstra

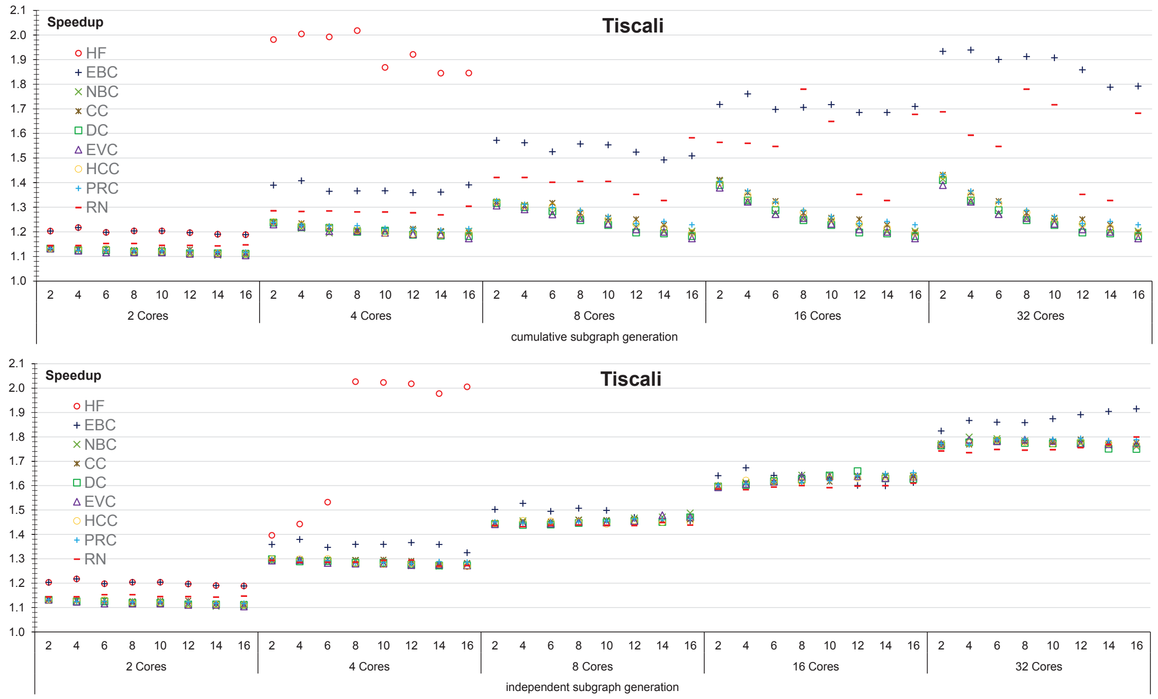


Figure A.5: Centrality results - Tiscali

A.2 Remove Heuristics

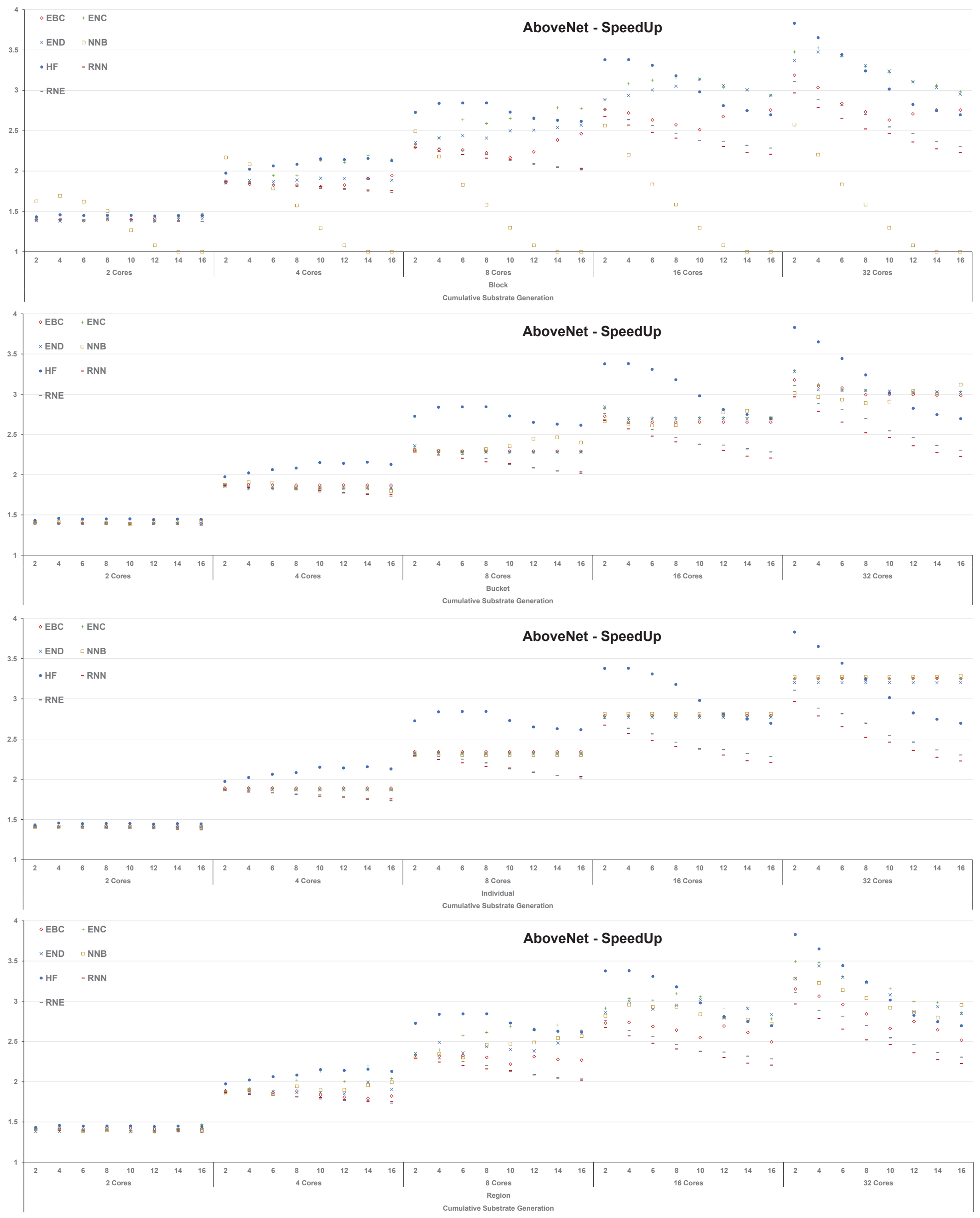


Figure A.6: Removal approaches with cumulative substrate generation for AboveNet

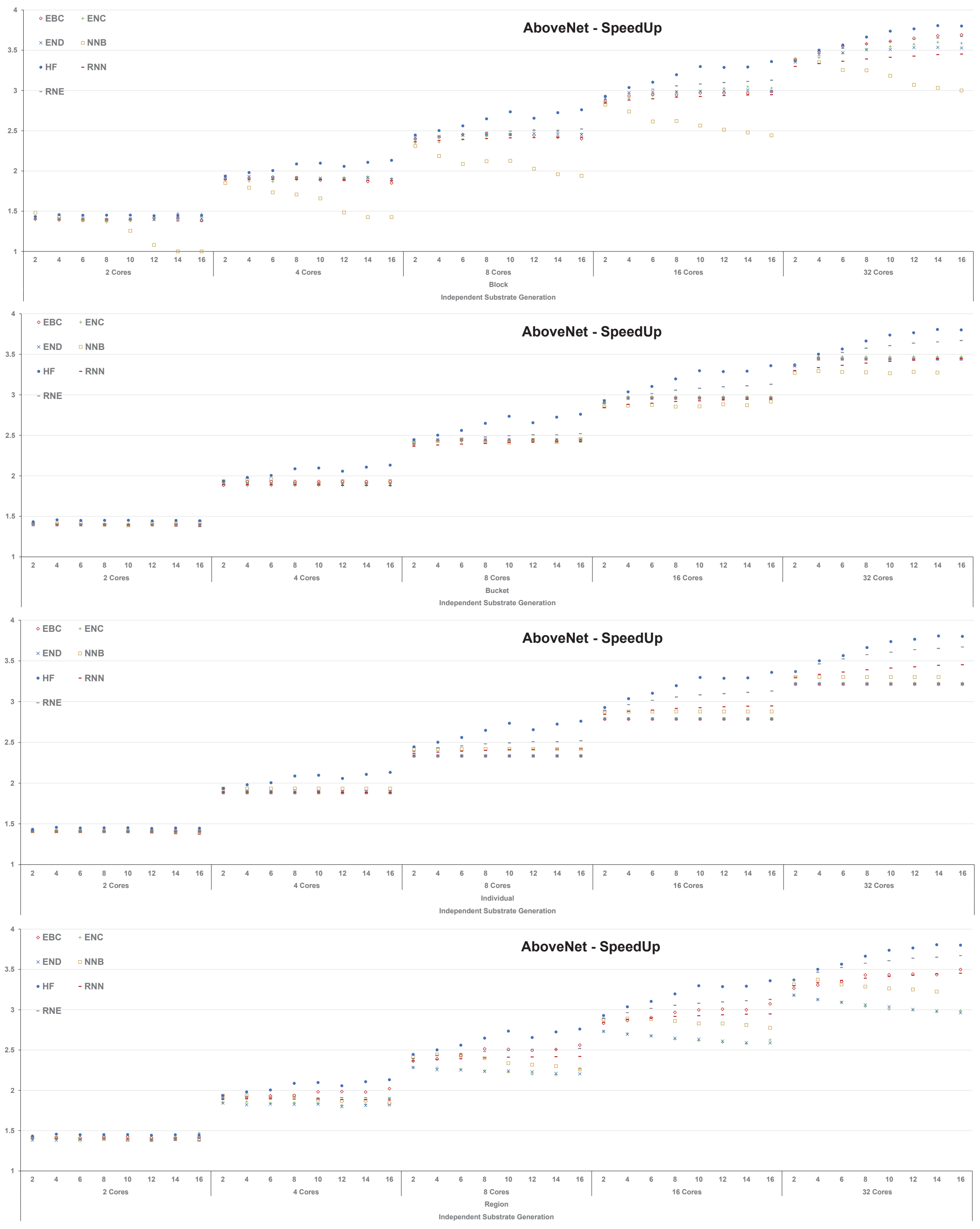


Figure A.7: Removal approaches with independent substrate generation for AboveNet

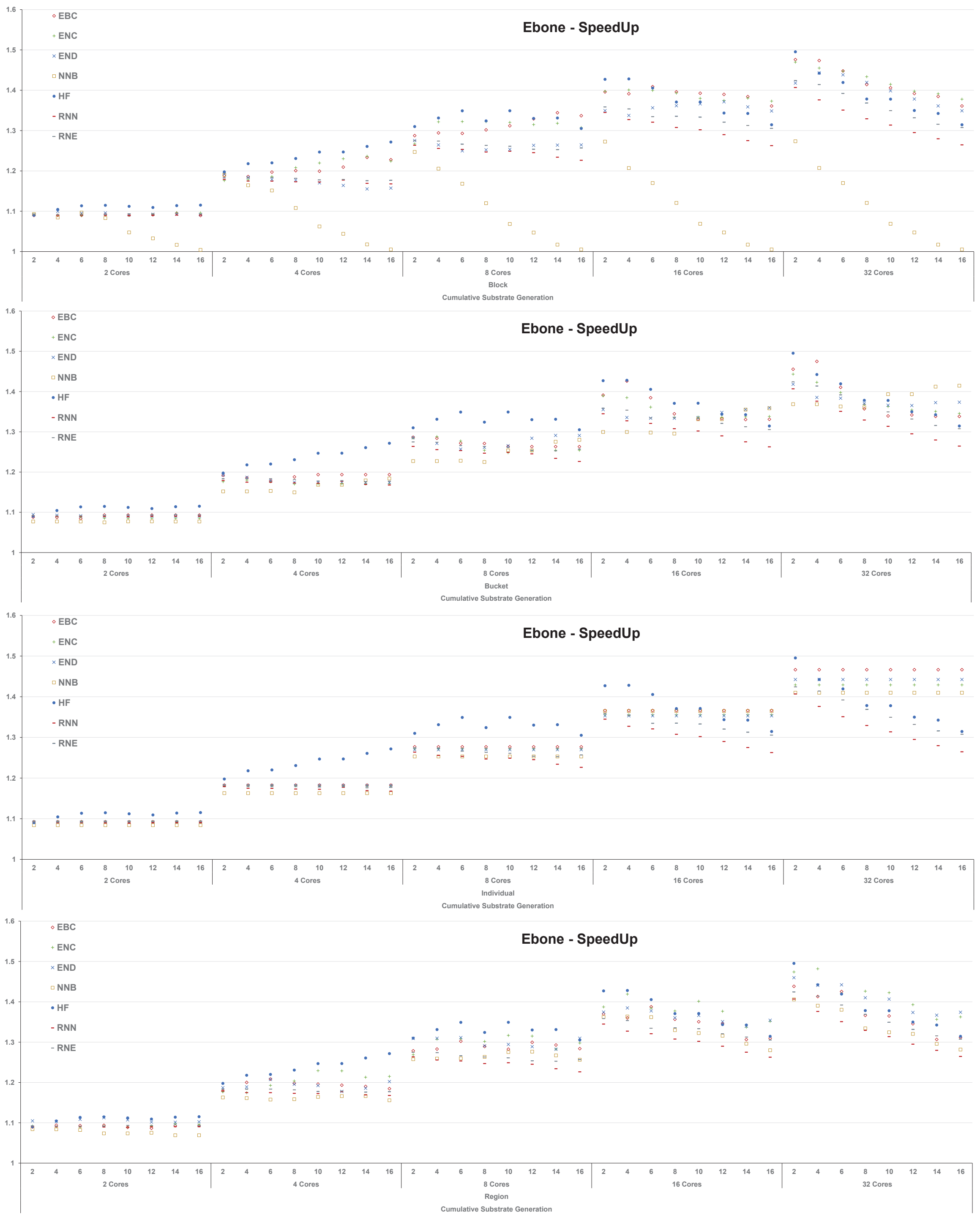
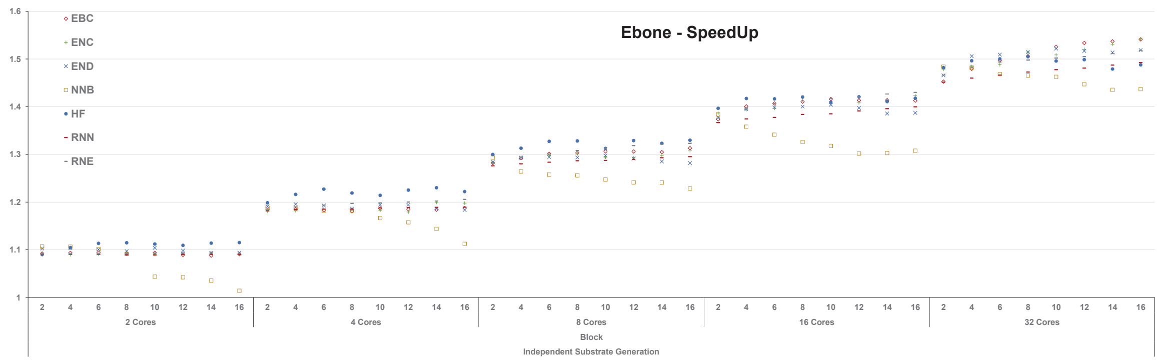
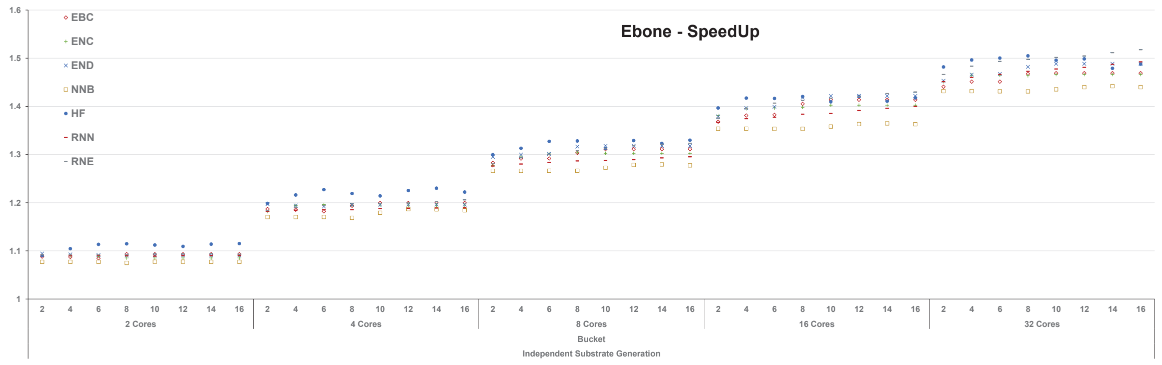
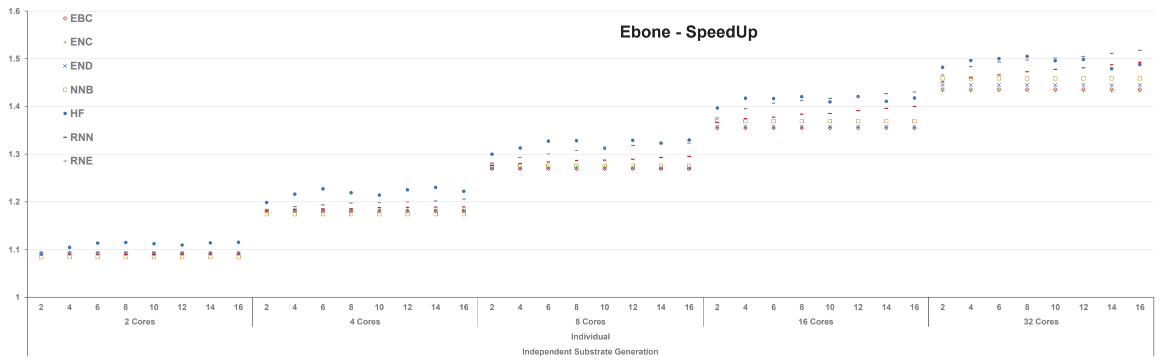


Figure A.8: Removal approaches with cumulative substrate generation for Ebone







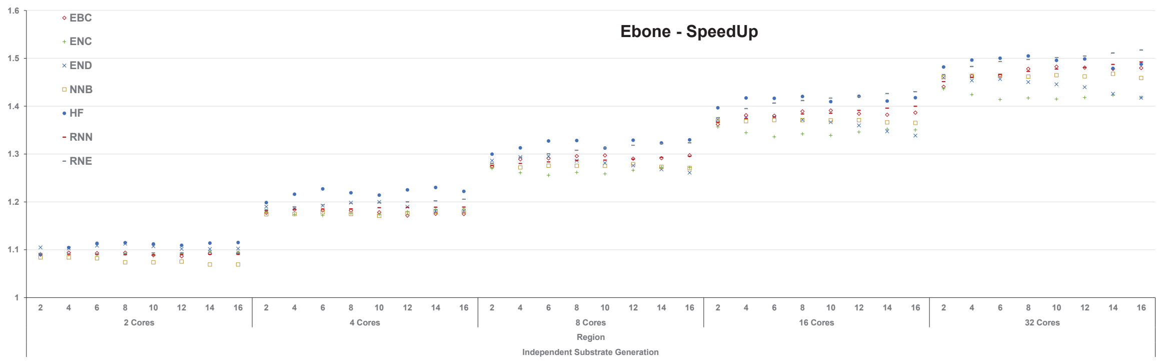
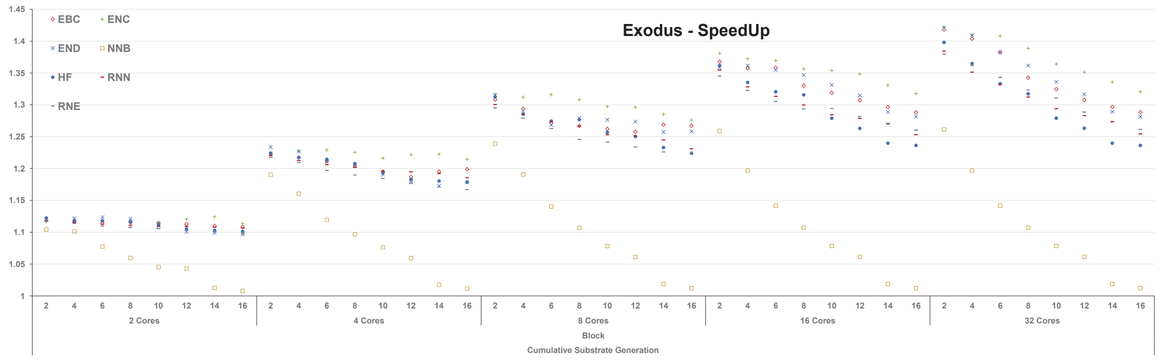
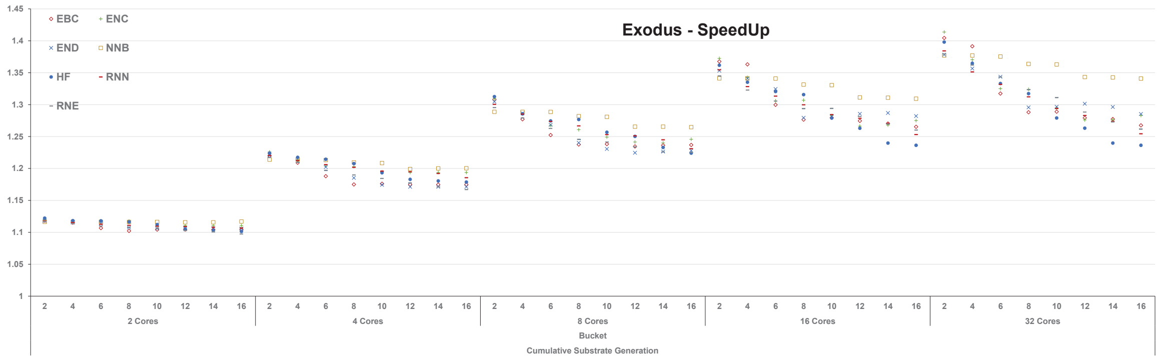
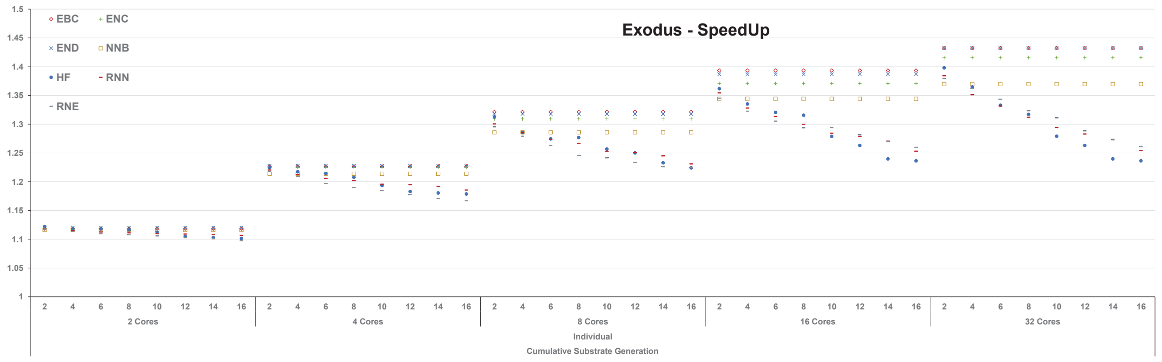


Figure A.9: Removal approaches with independent substrate generation for Ebone







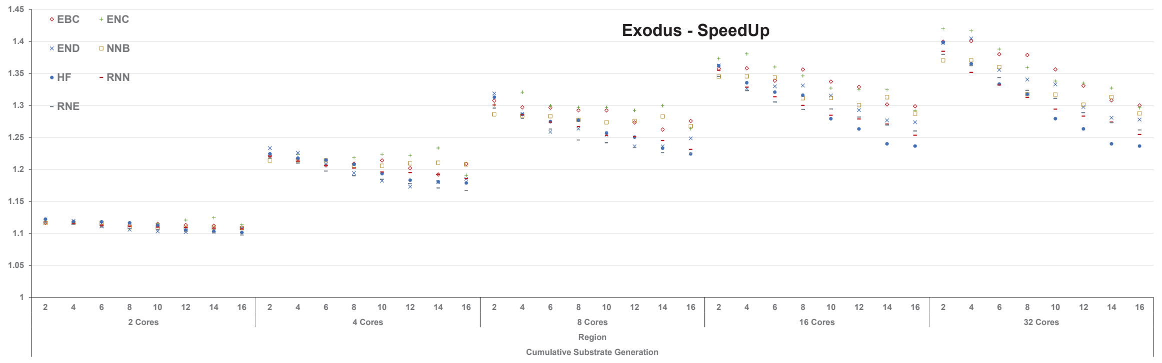
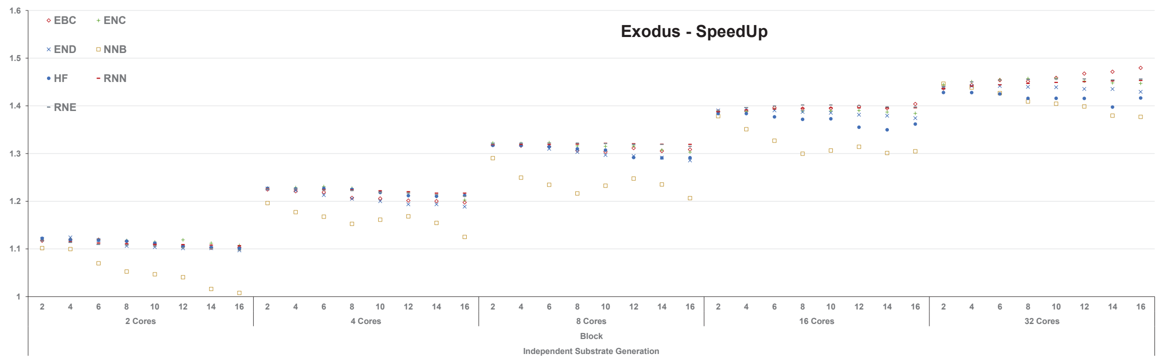
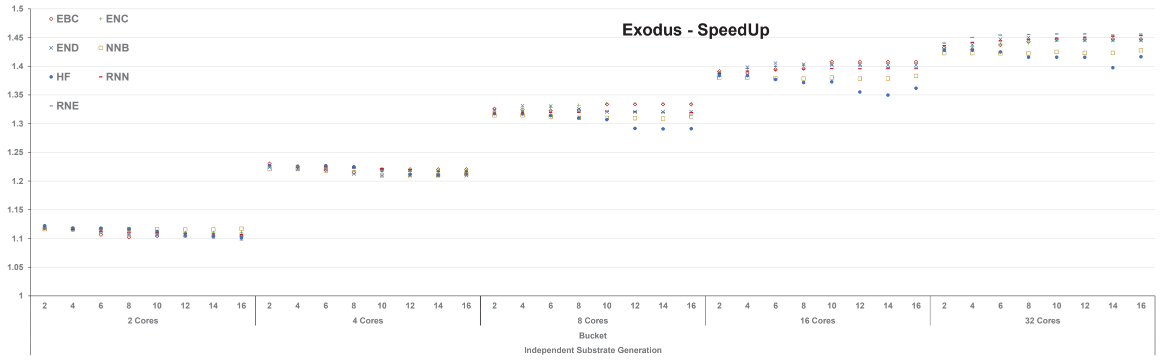
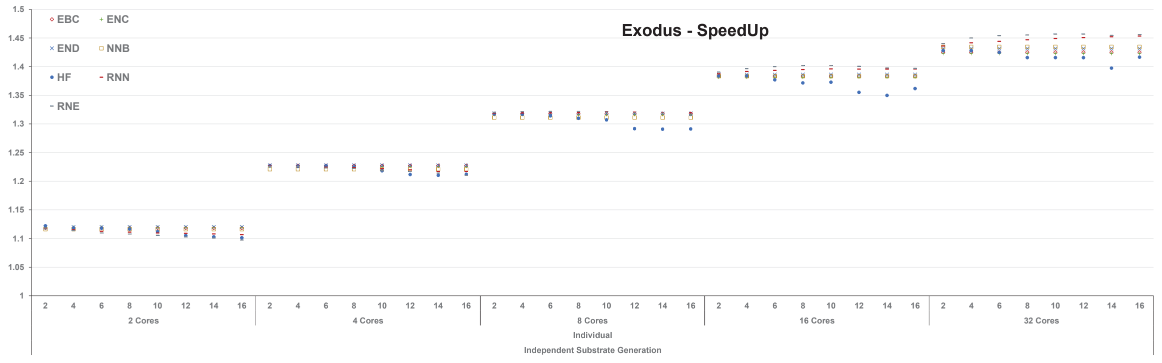


Figure A.10: Removal approaches with cumulative substrate generation for Exodus







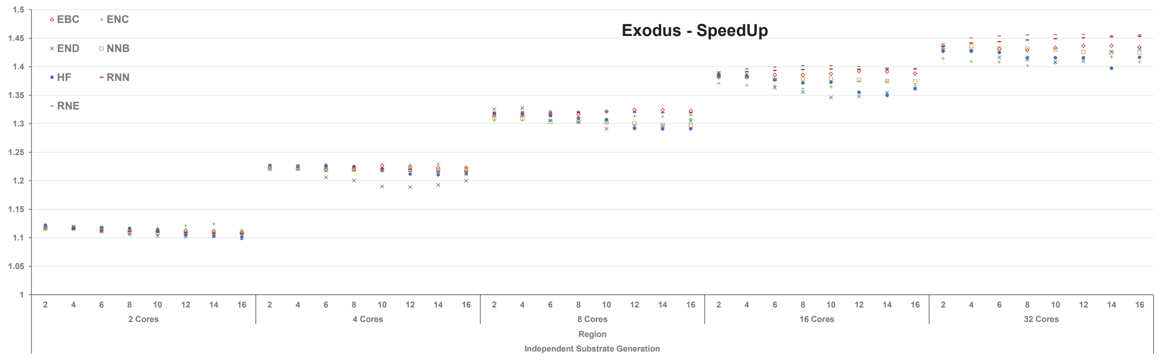


Figure A.11: Removal approaches with independent substrate generation for Exodus

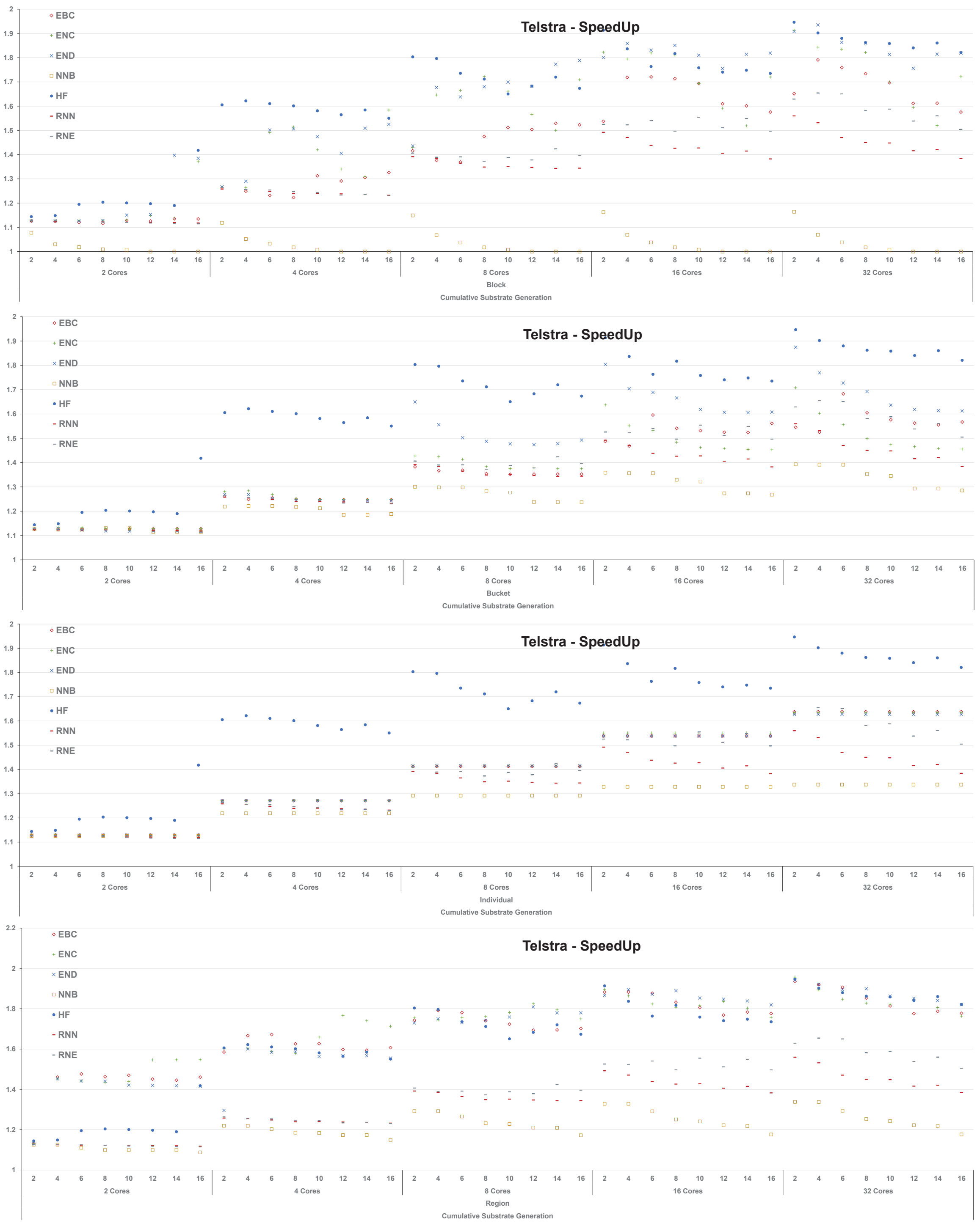
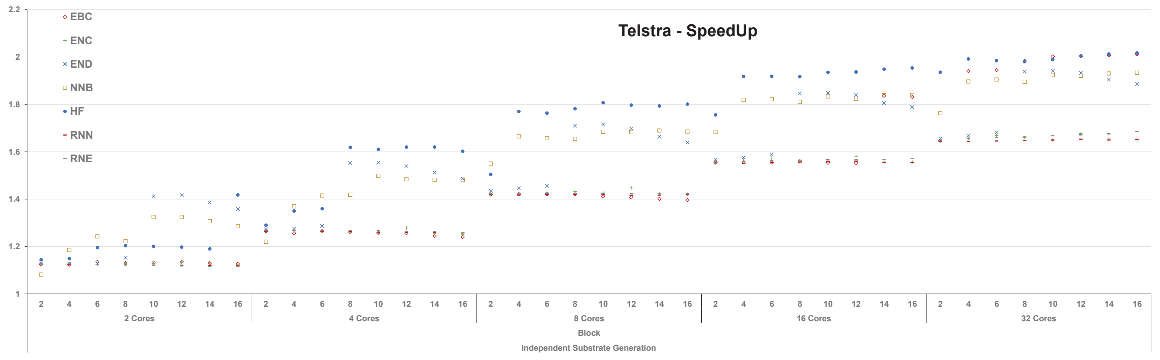
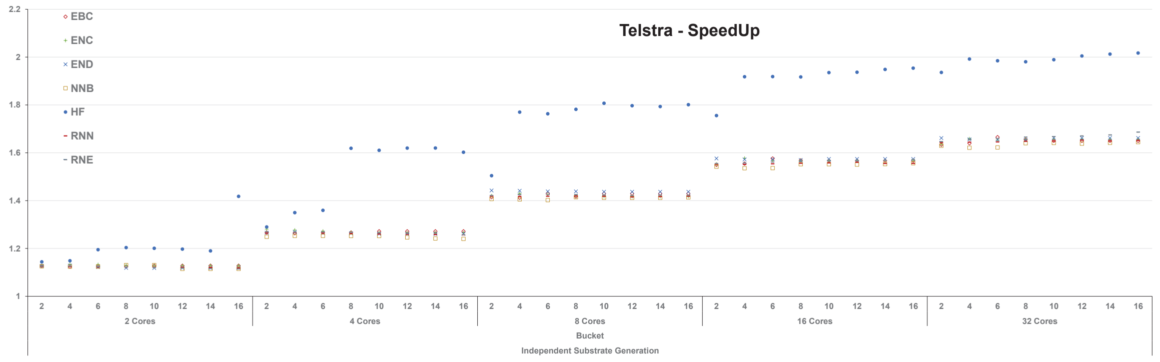
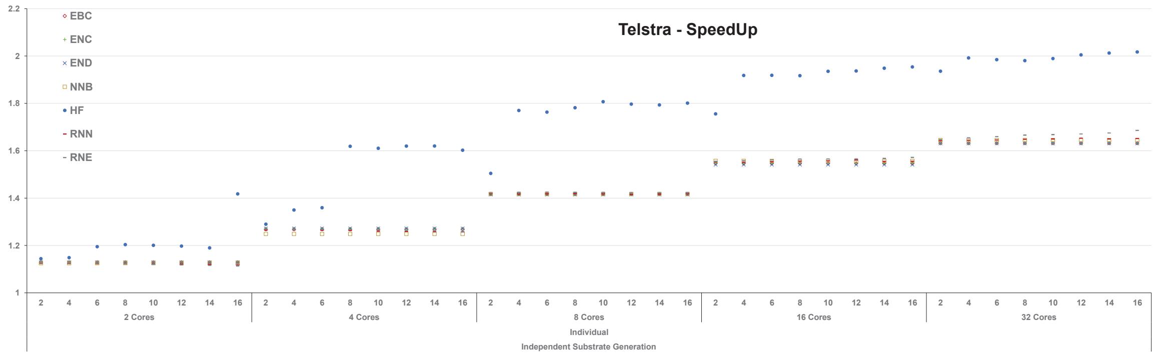


Figure A.12: Removal approaches with cumulative substrate generation for Telstra







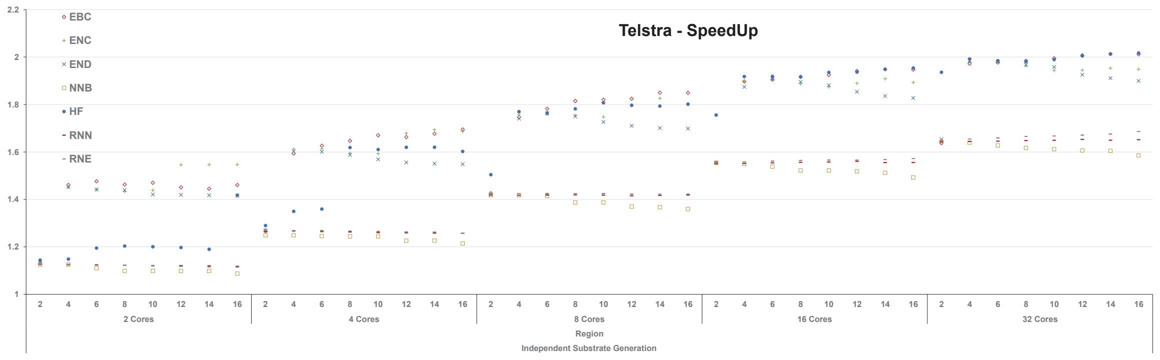
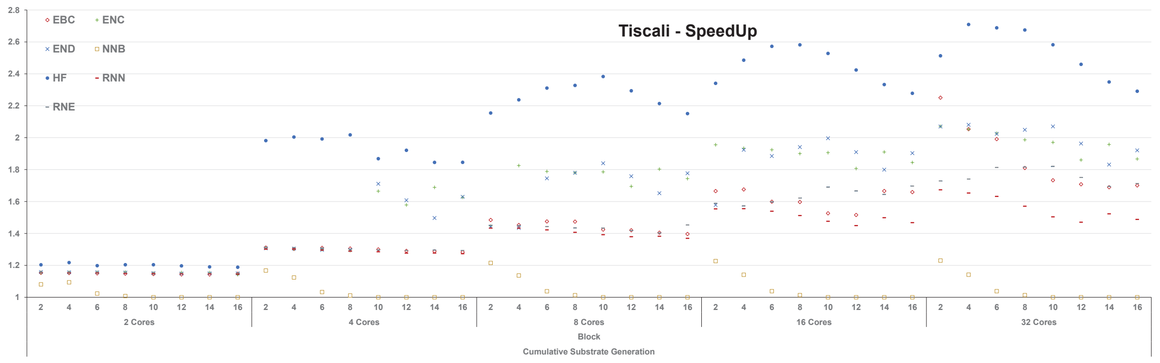
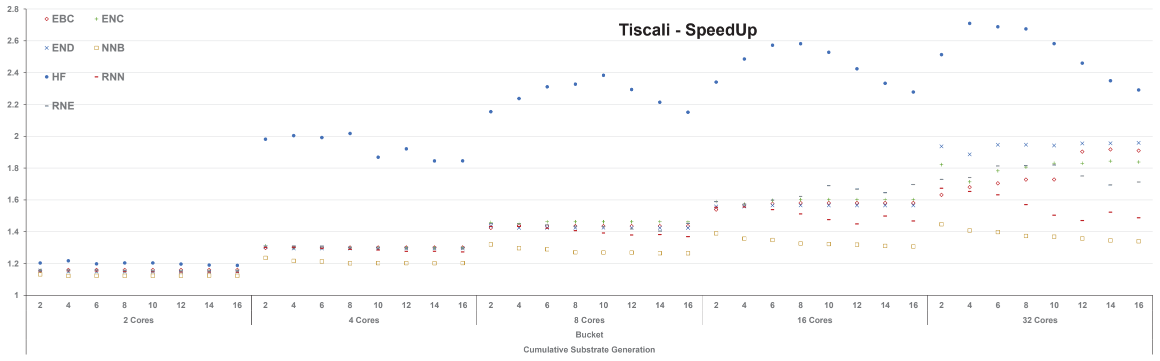
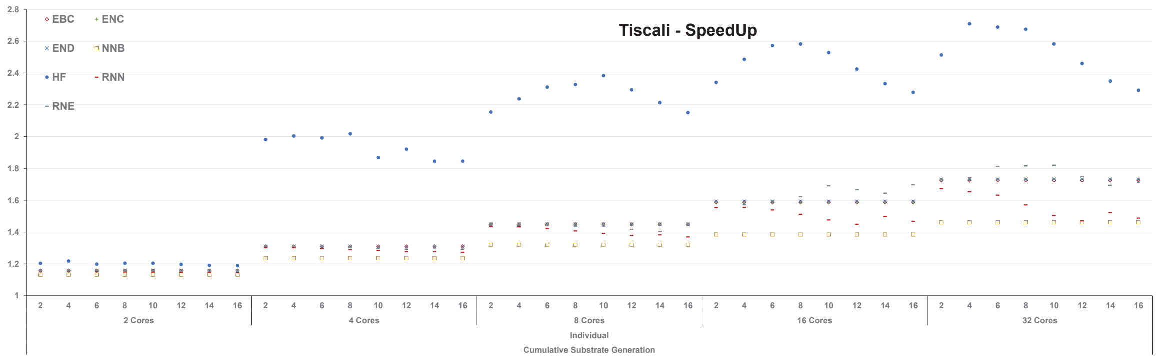


Figure A.13: Removal approaches with independent substrate generation for Telstra







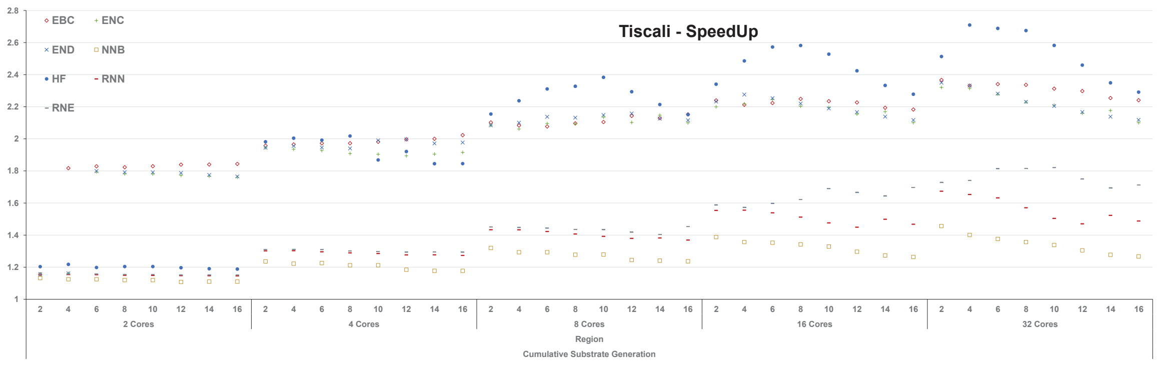
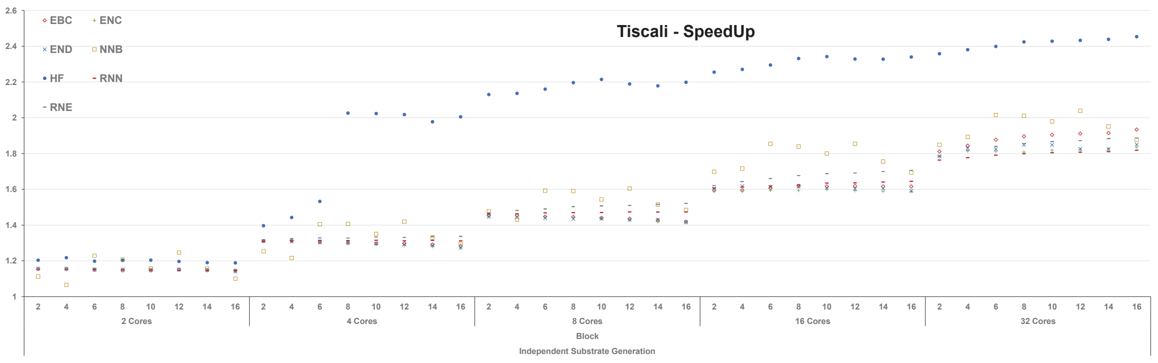
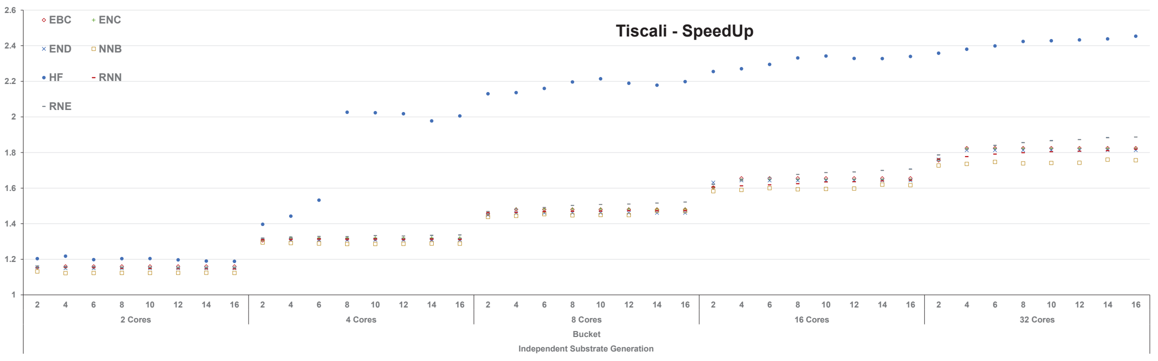
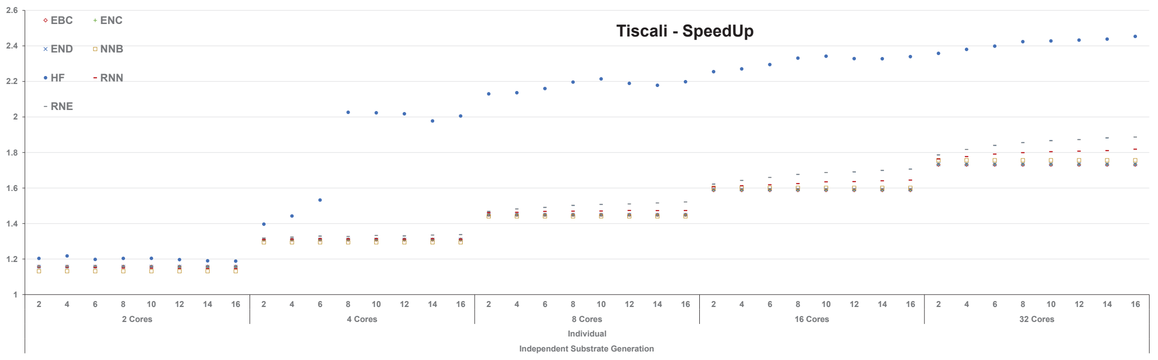


Figure A.14: Removal approaches with cumulative substrate generation for Tiscali







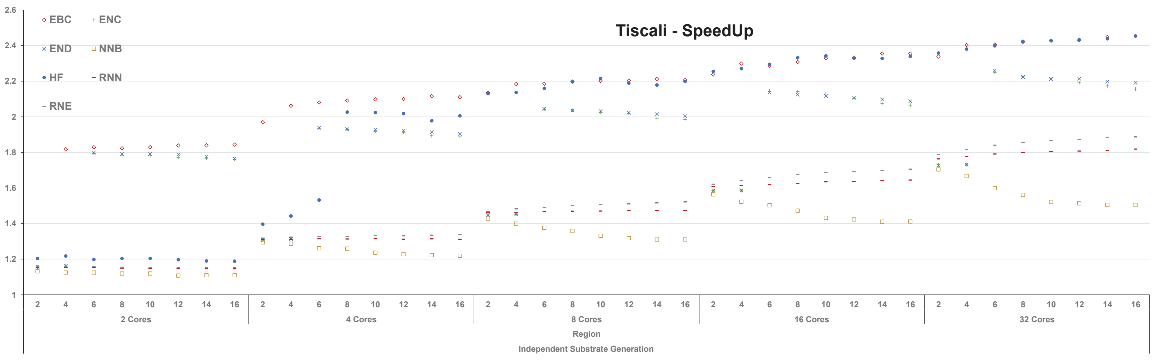


Figure A.15: Removal approaches with independent substrate generation for Tiscali

A.3 Substrate Characteristics

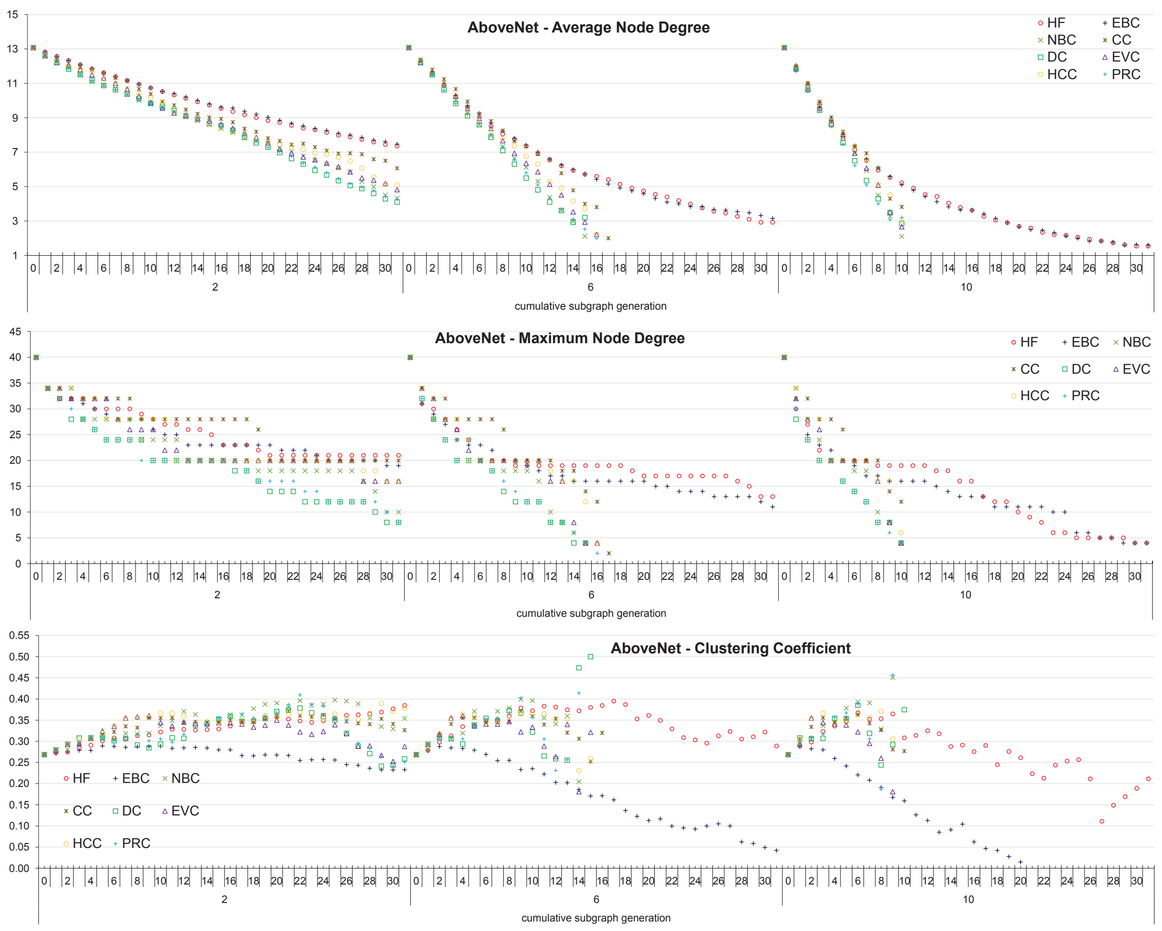


Figure A.16: Substrate changes with cumulative generation for AboveNet

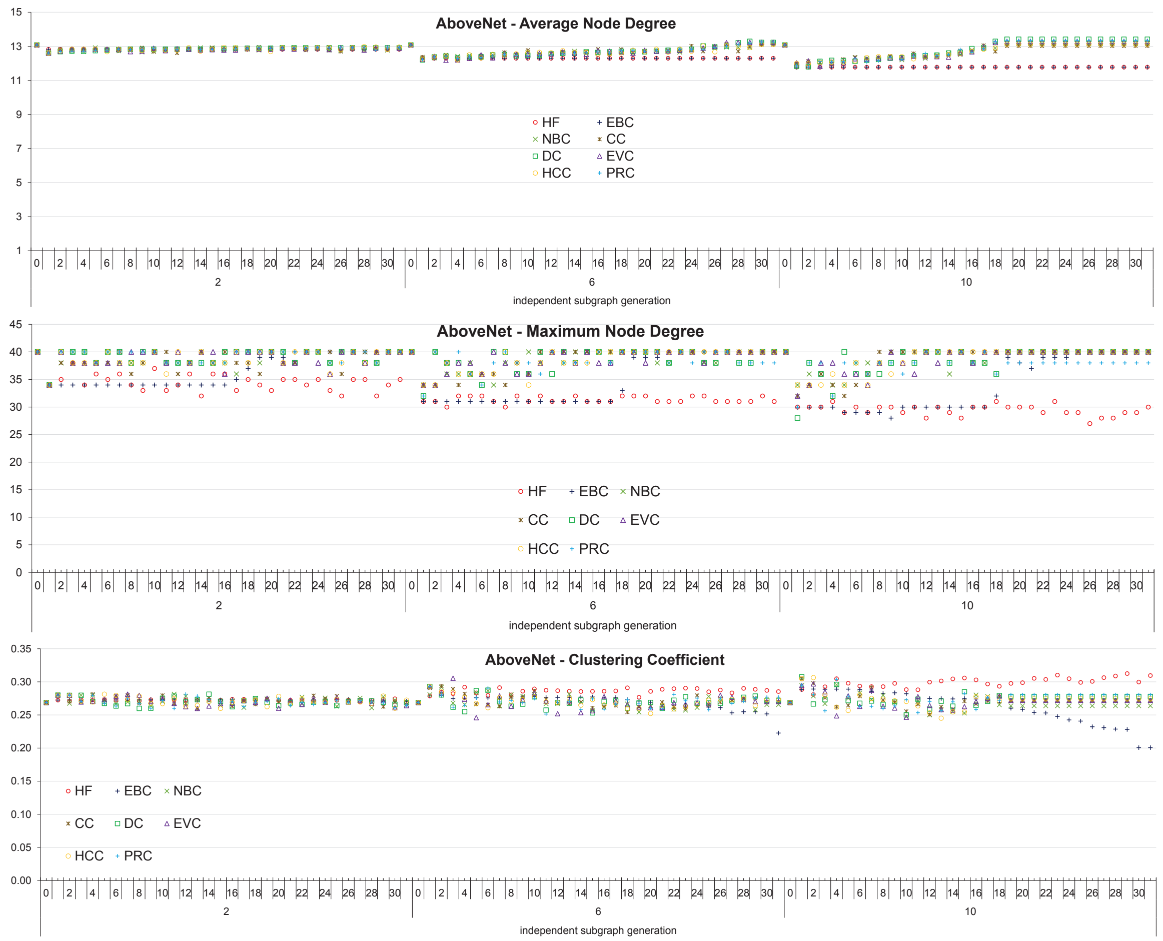


Figure A.17: Substrate changes with independent generation for AboveNet

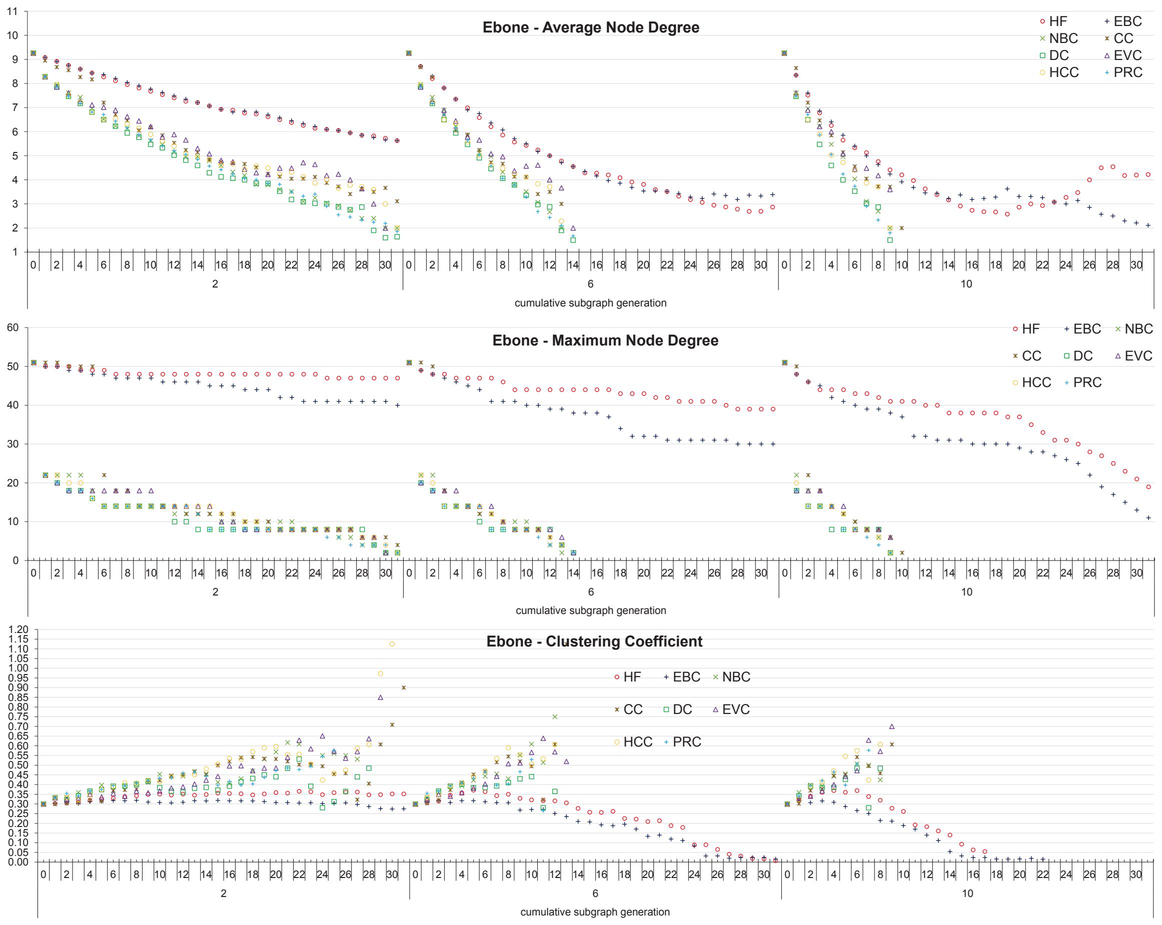


Figure A.18: Substrate changes with cumulative generation for Ebone

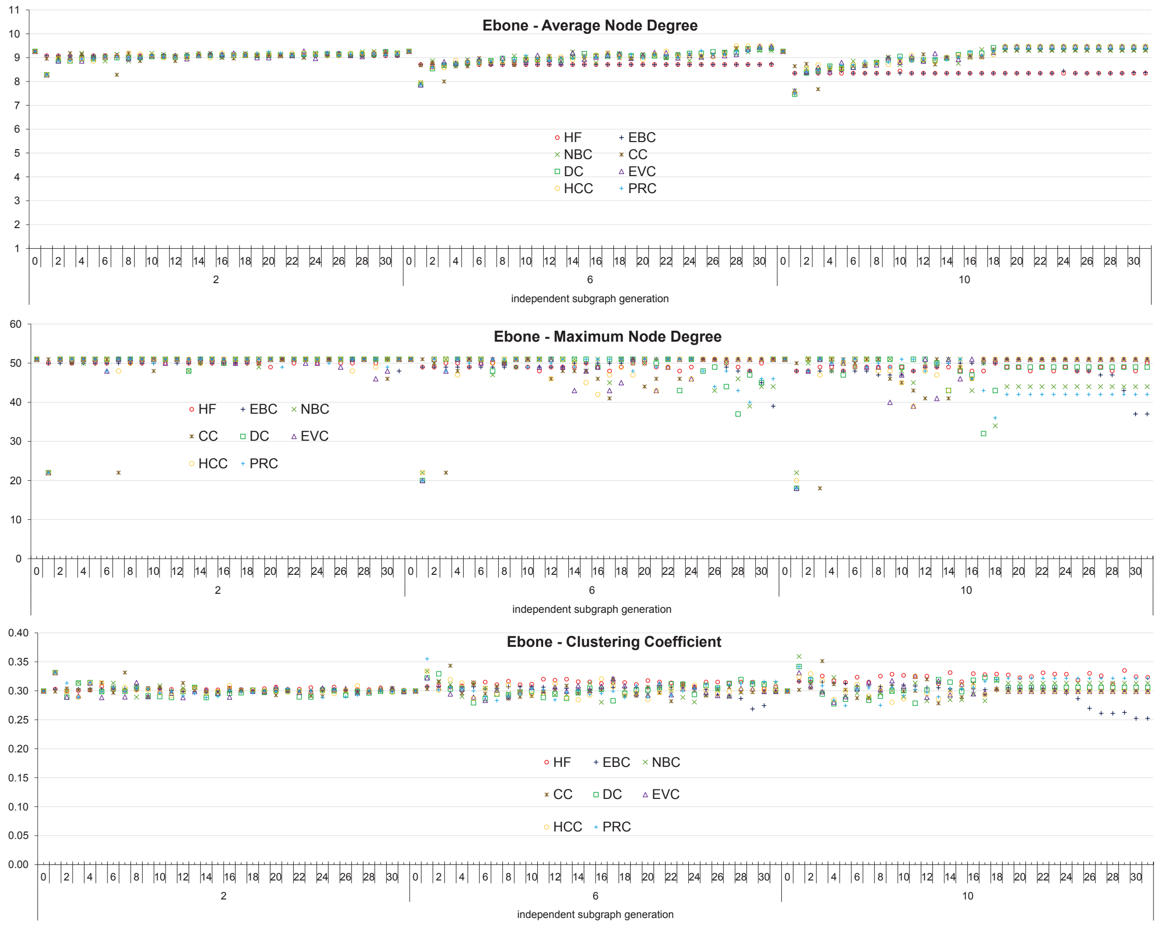
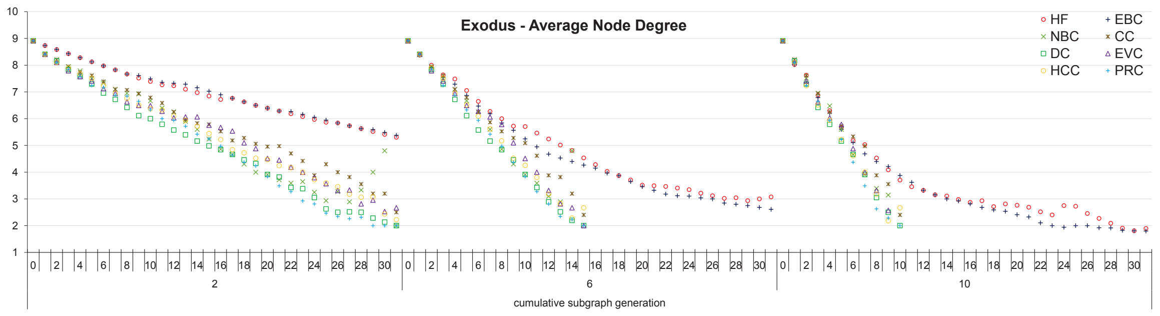
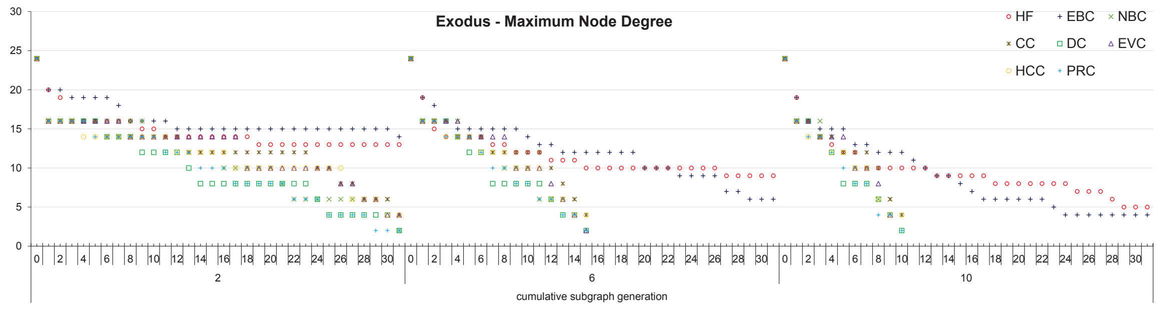


Figure A.19: Substrate changes with independent generation for Ebone





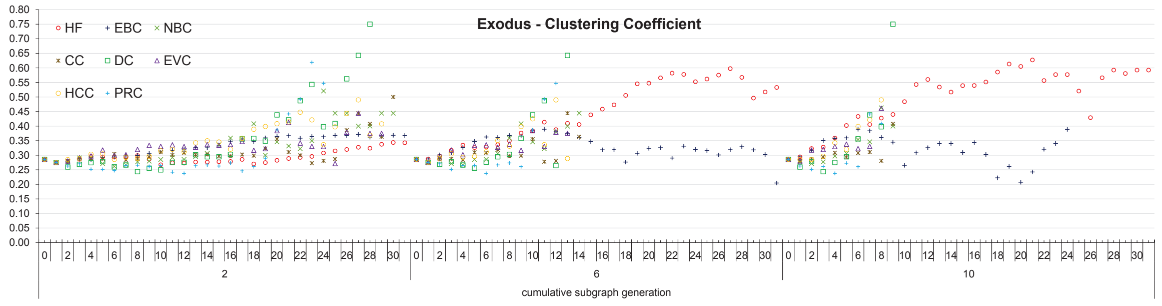


Figure A.20: Substrate changes with cumulative generation for Exodus

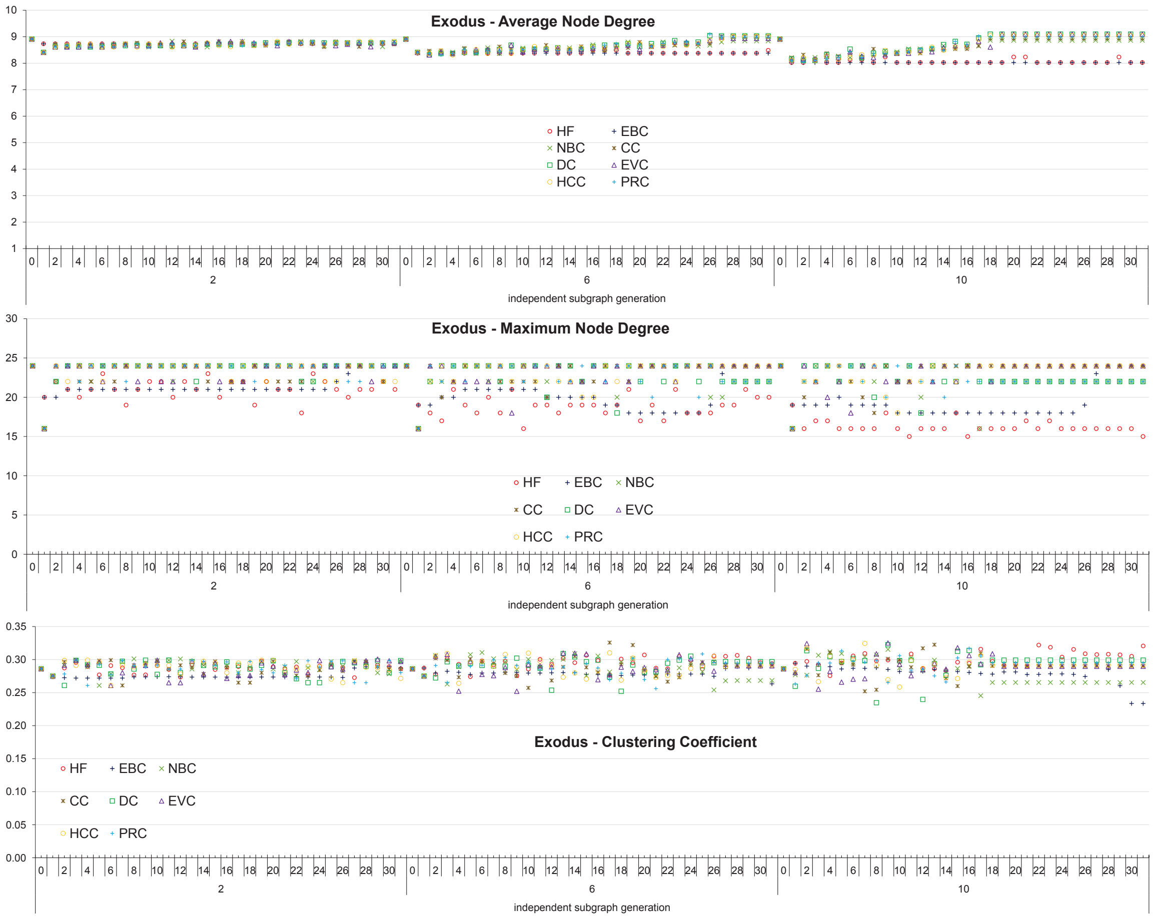


Figure A.21: Substrate changes with independent generation for Exodus

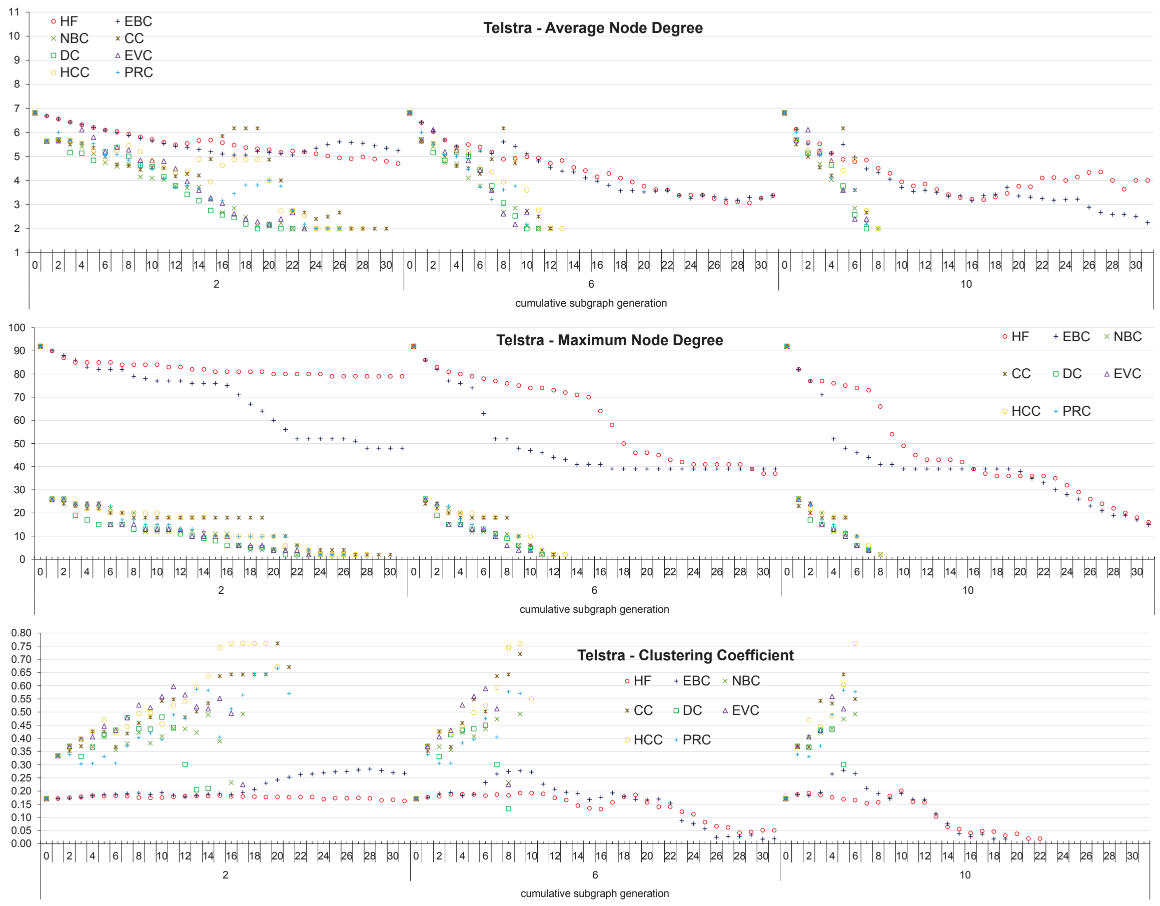
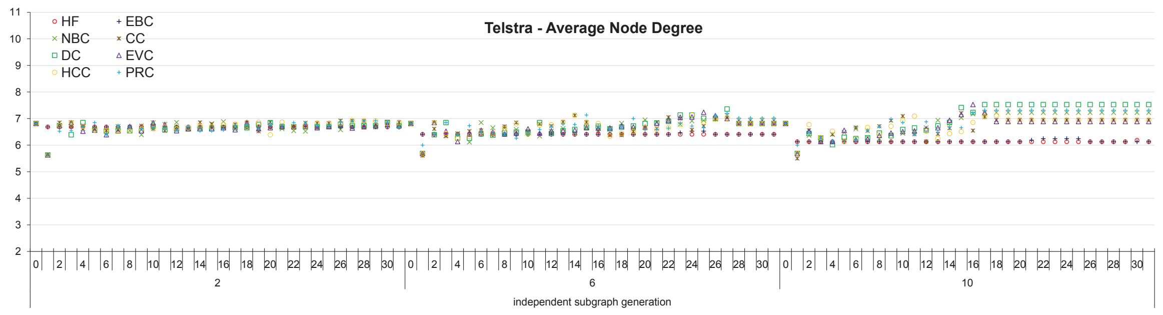
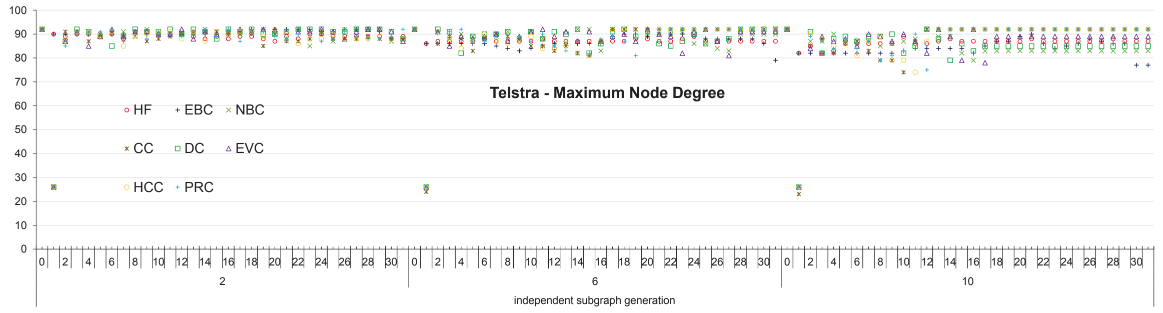


Figure A.22: Substrate changes with cumulative generation for Telstra





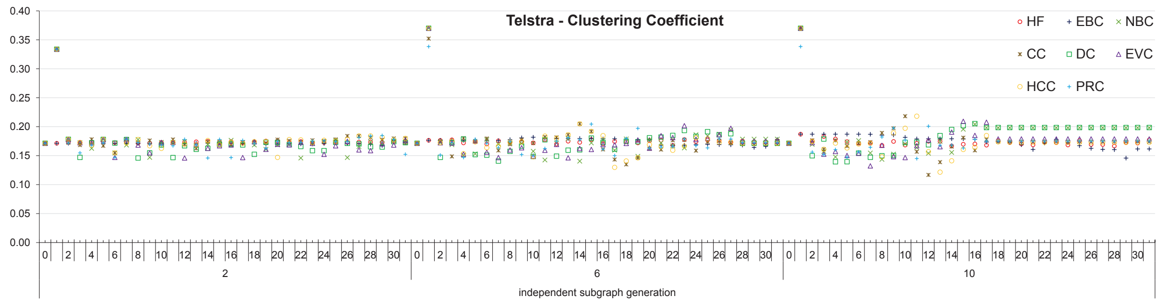


Figure A.23: Substrate changes with independent generation for Telstra

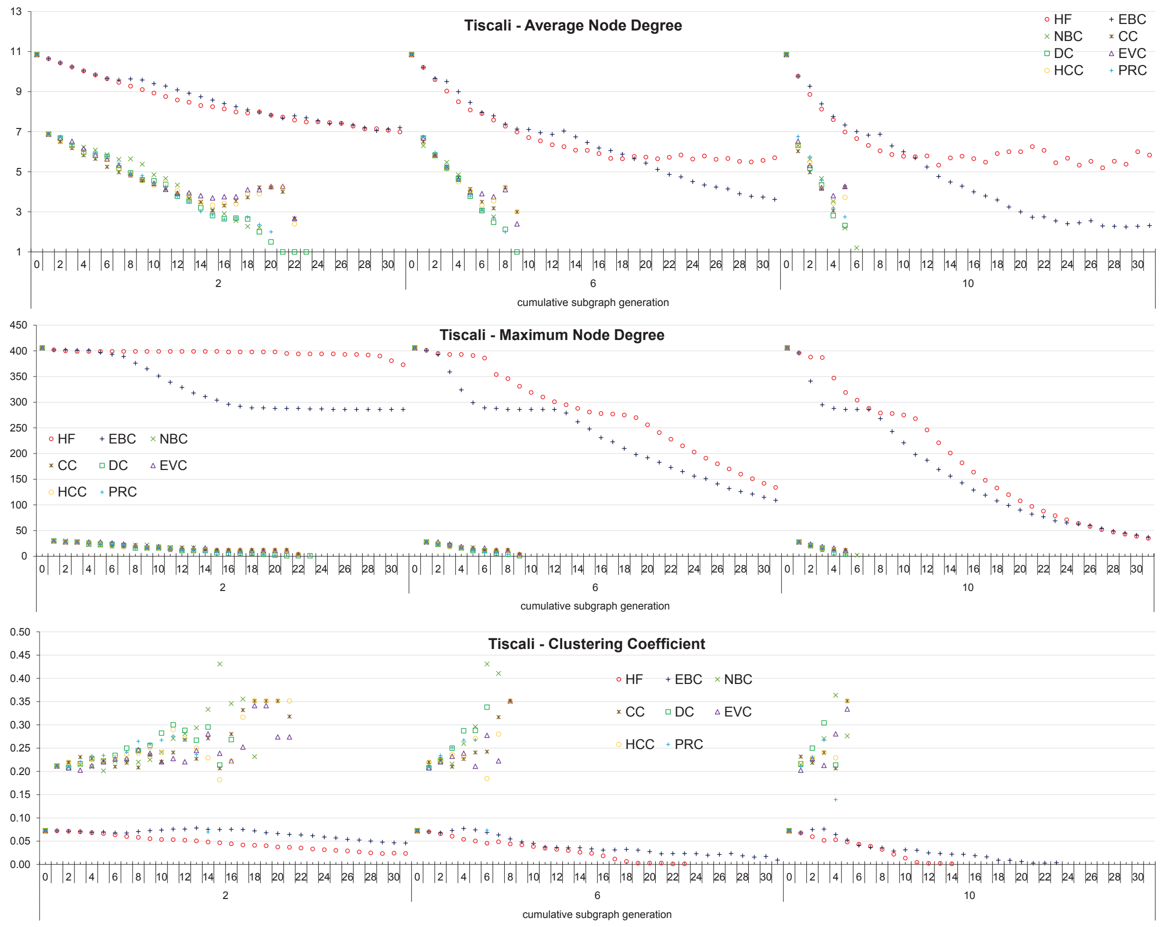
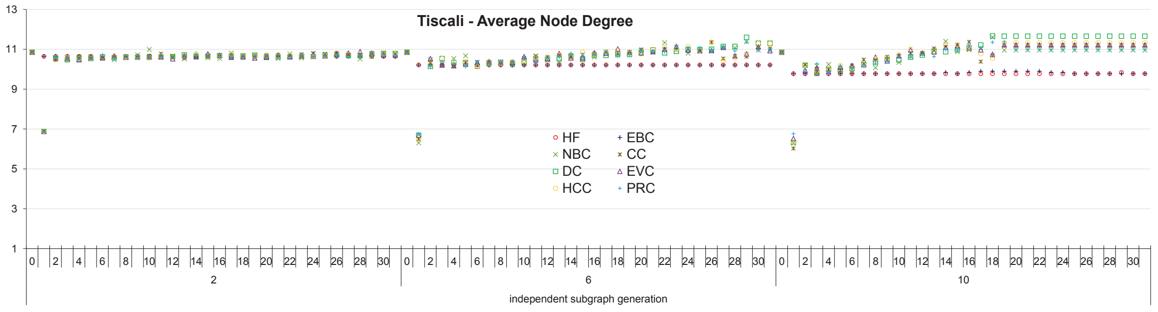
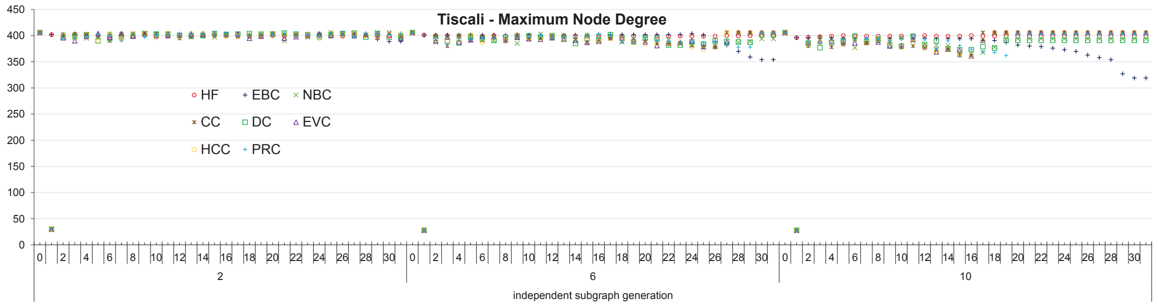


Figure A.24: Substrate changes with cumulative generation for Tiscali





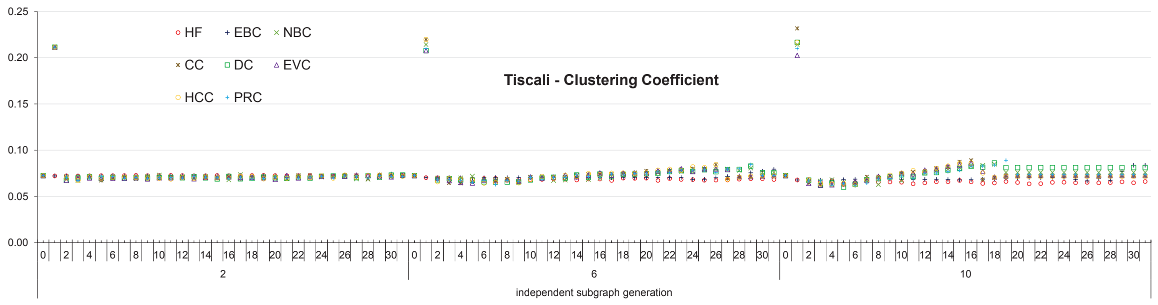


Figure A.25: Substrate changes with independent generation for Tiscali

Bibliography

[1] S. Agarwal, J. Dunagan, N. Jain, S. Saroiu, A. Wolman, and H. Bhogan. Volley: Automated data placement for geo-distributed cloud services. In *NSDI*, pages 17–32, 2010.

[2] D. Agrawal, S. Das, and A. El Abbadi. Big data and cloud computing: Current state and future opportunities. In *Proceedings of the 14th International Conference on Extending Database Technology*, EDBT/ICDT '11, pages 530–533, New York, NY, USA, 2011. ACM.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. Network flows: theory, algorithms, and applications. 1993.

[4] M. Alizadeh, S. Yang, M. Sharif, S. Katti, N. McKeown, B. Prabhakar, and S. Shenker. pfabric: Minimal near-optimal datacenter transport. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 435–446. ACM, 2013.

[5] W. Allcock, J. Bresnahan, R. Kettimuthu, M. Link, C. Dumitrescu, I. Raicu, and I. Foster. The Globus striped GridFTP framework and server. In *Proceedings of ACM/IEEE conference on Supercomputing*, page 54, 2005.

[6] J. E. Anderson. The gravity model. *Annu. Rev. Econ.*, 3(1):133–160, 2011.

[7] M. Balman and T. Kosar. Dynamic adaptation of parallelism level in data transfer scheduling. In *Complex, Intelligent and Software Intensive Systems, 2009. CISIS'09. International Conference on*, pages 872–877. IEEE, 2009.

[8] C. Barakat, E. Altman, and W. Dabbous. On tcp performance in a heterogeneous network: a survey. *Communications Magazine, IEEE*, 38(1):40–46, 2000.

[9] S. Barré, O. Bonaventure, C. Raiciu, and M. Handley. Experimenting with multipath tcp. *ACM SIGCOMM Computer Communication Review*, 41(4):443–444, 2011.

[10] M. Bastian, S. Heymann, M. Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *ICWSM*, 8:361–362, 2009.

[11] V. Betz, J. Swartz, and V. Gouterman. Method and apparatus for performing parallel routing using a multi-threaded routing procedure, Sept. 10 2013. US Patent 8,533,652.

[12] B. W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5):61–72, 1988.

[13] L. Bolc and J. Cytowski. *Search methods for artificial intelligence*. Academic Press, 1992.

[14] R. Bolla, R. Bruschi, and P. Lago. Energy adaptation in multi-core software routers. *Computer Networks*, 65:111–128, 2014.

[15] J. E. Burns, T. J. Ott, A. E. Krzesinski, and K. E. Müller. Path selection and bandwidth allocation in {MPLS} networks. *Performance Evaluation*, 52(2–3):133 – 152, 2003. Internet Performance and Control of Network Systems.

[16] M. Calder, R. Miao, K. Zarifis, E. Katz-Bassett, M. Yu, and J. Padhye. Don't drop, detour! In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 503–504. ACM, 2013.

[17] K. Cameron, E. M. Eschen, C. T. Hoàng, and R. Sritharan. The list partition problem for graphs. In *Proceedings of the fifteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 391–399. Society for Industrial and Applied Mathematics, 2004.

[18] V. Cerf, Y. Dalal, and C. Sunshine. Specification of internet transmission control program rfc 675. 1974.

[19] Y. Chen, S. Jain, V. K. Adhikari, Z.-L. Zhang, and K. Xu. A first look at inter-data center traffic characteristics via yahoo! datasets. In *INFOCOM, 2011 Proceedings IEEE*, pages 1620–1628. IEEE, 2011.

[20] A. CloudFront. Amazon cloudfront. *URL: http://aws. amazon. com/cloudfront*, 2014.

[21] G. Consortium et al. Gephi. *Computer program](version 0.8. 2 Beta) http://gephi. github. io/. Accessed*, 14, 2014.

[22] C.Raiciu, S. Barre, C. Pluntke, A. Greenhalgh, D. Wischik, and M. Handley. Improving datacenter performance and robustness with multipath tcp. In *Proceedings of ACM SIGCOMM*, pages 266–277, 2011.

[23] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[24] D. J. DeWitt, S. Ghandeharizadeh, D. A. Schneider, A. Bricker, H.-I. Hsiao, and R. Rasmussen. The gamma database machine project. *Knowledge and Data Engineering, IEEE Transactions on*, 2(1):44–62, 1990.

[25] J. Duan, Z. Wang, and C. Wu. Responsive multipath tcp in sdn-based datacenters. In *Communications (ICC), 2015 IEEE International Conference on*, pages 5296–5301. IEEE, 2015.

[26] P. Eardley. Survey of mptcp implementations. 2013.

[27] N. Egi, G. Iannaccone, M. Manesh, L. Mathy, and S. Ratnasamy. Improved parallelism and scheduling in multi-core software routers. *The Journal of Supercomputing*, 63(1):294–322, 2013.

[28] N. Farrington and A. Andreyev. Facebook's data center network architecture. In *IEEE Opt. Interconnects Conf*, pages 5–7. Citeseer, 2013.

[29] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. Tcp extensions for multipath operation with multiple addresses. *RFC*, (6824), January 2013.

[30] T. Ganegedara and V. Prasanna. 100+ gbps ipv6 packet forwarding on multicore platforms.

[31] Y. Ganjali and A. Keshavarzian. Load balancing in ad hoc networks: single-path routing vs. multi-path routing. In *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, volume 2, pages 1120–1125. IEEE, 2004.

[32] J. J. Garcia-Luna-Aceves, M. Mosko, and C. E. Perkins. A new approach to on-demand loop-free routing in networks using sequence numbers. *Computer Networks*, 50(10):1599–1615, 2006.

[33] M. R. Garey and D. S. Johnson. *Computers and intractability*, volume 29. wh freeman New York, 2002.

[34] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified np-complete graph problems. *Theoretical computer science*, 1(3):237–267, 1976.

[35] P. Gill, N. Jain, and N. Nagappan. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 350–361. ACM, 2011.

[36] I. Gojmerac, T. Ziegler, F. Ricciato, and P. Reichl. Adaptive multipath routing for dynamic traffic engineering. In *Proceedings of IEEE Global Communications Conference (GLOBECOM)*, San Francisco, CA, November 2003.

[37] S. Grover. *Using Multicore to Accelerate Network Routing Protocols.* North Carolina State University, 2013.

[38] M. Hilbert and P. López. The world's technological capacity to store, communicate, and compute information. *Science*, 332(6025):60–65, 2011.

[39] C. J. Hillar and L.-H. Lim. Most tensor problems are np-hard. *Journal of the ACM (JACM)*, 60(6):45, 2013.

[40] C.-Y. Hong, S. Kandula, R. Mahajan, M. Zhang, V. Gill, M. Nanduri, and R. Wattenhofer. Achieving high utilization with software-driven wan. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, pages 15–26. ACM, 2013.

[41] D. Howe, M. Costanzo, P. Fey, T. Gojobori, L. Hannick, W. Hide, D. P. Hill, R. Kania, M. Schaeffer, S. St Pierre, et al. Big data: The future of biocuration. *Nature*, 455(7209):47–50, 2008.

[42] T. C. Hu. Multi-commodity network flows. *Operations Research*, 11(3):344–360, 1963.

[43] V. Jacobson, R. Braden, D. Borman, M. Satyanarayanan, J. Kistler, L. Mummert, and M. Ebling. Rfc 1323: Tcp extensions for high performance, 1992.

[44] J.Raju and J. Garcia-Luna-Aceves. A new approach to on-demand loop-free multipath routing. In *International Conference on Computer Communications and Networks (ICCCN)*, pages 522–527, 1999.

[45] H. T. Karaoglu, M. Yuksel, and M. H. Gunes. On the scalability of path exploration using opportunistic path-vector routing. In *2011 IEEE International Conference on Communications (ICC)*, pages 1–5. IEEE, 2011.

[46] R. M. Karp. Reducibility among combinatorial problems. pages 85–103, 1972.

[47] H. Kellerer, U. Pferschy, and D. Pisinger. *Introduction to NP-Completeness of knapsack problems.* Springer, 2004.

[48] C. H. Kingsley and G. L. McHugh. Parallel signal routing, Feb. 26 2013. US Patent 8,386,983.

[49] T. Kosar, E. Arslan, B. Ross, and B. Zhang. Storkcloud: Data transfer scheduling and optimization as a service. In *Proceedings of the 4th ACM workshop on Scientific cloud computing*, pages 29–36. ACM, 2013.

[50] D. L. Kreher and D. R. Stinson. *Combinatorial algorithms: generation, enumeration, and search*, volume 7. CRC press, 1998.

[51] L. Lan, L. Li, and C. Jianya. A multipath routing algorithm based on ospf routing protocol. In *Semantics, Knowledge and Grids (SKG), 2012 Eighth International Conference on*, pages 269–272. IEEE, 2012.

[52] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter bulk transfers with netstitcher. In *Proceedings of ACM SIGCOMM*, 2011.

[53] N. Laoutaris, M. Sirivianos, X. Yang, and P. Rodriguez. Inter-datacenter bulk transfers with netstitcher. *ACM SIGCOMM Computer Communication Review*, 41(4):74–85, 2011.

[54] M. Laubach and J. Halpern. Rfc 2225: Classical ip and arp over atm. *Newbridge Networks*, 1998.

[55] S.-J. Lee and M. Gerla. Split multipath routing with maximally disjoint paths in ad hoc networks. In *Communications, 2001. ICC 2001. IEEE International Conference on*, volume 10, pages 3201–3205. IEEE, 2001.

[56] T. LHC Study Group et al. The large hadron collider, conceptual design. Technical report, CERN/AC/95-05 (LHC) Geneva, 1995.

[57] S. Lohr. The age of big data. *New York Times*, 11, 2012.

[58] L. Lovász. *Combinatorial problems and exercises*, volume 361. American Mathematical Soc., 1993.

[59] C. Lynch. Big data: How do your data grow? *Nature*, 455(7209):28–29, 2008.

[60] G. Malkin. Rfc 2453: Rip version 2. *Request for Comments*, 2453, 1998.

[61] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers. Big data: The next frontier for innovation, competition, and productivity. 2011.

[62] M. K. Marina and S. R. Das. On-demand multipath distance vector routing in ad hoc networks. In *Network Protocols, 2001. Ninth International Conference on*, pages 14–23. IEEE, 2001.

[63] R. Miller. Google data center faq. *Datacenter Knowledge*, 27, 2008.

[64] S. Molnár, B. Sonkoly, and T. A. Trinh. A comprehensive tcp fairness analysis in high speed networks. *Computer Communications*, 32(13):1460–1484, 2009.

[65] J. Moy. rfc 2328: Ospf version 2, 1998.

[66] J. Mudigonda, P. Yalagandula, M. Al-Fares, and J. C. Mogul. Spain: Cots data-center ethernet for multipathing over arbitrary topologies. In *NSDI*, pages 265–280, 2010.

[67] J. Mudigonda, P. Yalagandula, J. Mogul, B. Stiekes, and Y. Pouffary. Netlord: a scalable multi-tenant network architecture for virtualized datacenters. *ACM SIGCOMM Computer Communication Review*, 41(4):62–73, 2011.

[68] S. Murthy and J. Garcia-Luna-Aceves. Congestion-oriented shortest multipath routing. In *INFOCOM'96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, volume 3, pages 1028–1036. IEEE, 1996.

[69] A. Nasipuri and S. R. Das. On-demand multipath routing for mobile ad hoc networks. In *Computer Communications and Networks, 1999. Proceedings. Eight International Conference on*, pages 64–70. IEEE, 1999.

[70] S. Nelakuditi and Z. Zhang. On selection of paths for multipath routing. In *Proceedings of IEEE/IFIP International Workshop on Quality of Service (IWQoS)*, pages 170–186, 2001.

[71] R. G. Ogier, V. Rutenburg, and N. Shacham. Distributed algorithms for computing shortest pairs of disjoint paths. *Information Theory, IEEE Transactions on*, 39(2):443–455, 1993.

[72] S. Okada. Fuzzy shortest path problems incorporating interactivity among paths. *Fuzzy Sets and Systems*, 142(3):335–357, 2004.

[73] C. Paasch, G. Detal, F. Duchene, C. Raiciu, and O. Bonaventure. Exploring mobile/wifi handover with multipath tcp. In *Proceedings of the 2012 ACM SIGCOMM workshop on Cellular networks: operations, challenges, and future design*, pages 31–36. ACM, 2012.

[74] M. R. Pearlman, Z. J. Haas, P. Sholander, and S. S. Tabrizi. On the impact of alternate path routing for load balancing in mobile ad hoc networks. In *Mobile and Ad Hoc Networking and Computing, 2000. MobiHOC. 2000 First Annual Workshop on*, pages 3–10. IEEE, 2000.

[75] L. L. Peterson and B. S. Davie. *Computer networks: a systems approach.* Elsevier, 2007.

[76] C. Pluntke, L. Eggert, and N. Kiukkonen. Saving mobile device energy with multipath tcp. In *Proceedings of the sixth international workshop on MobiArch*, pages 1–6. ACM, 2011.

[77] J. Postel. Rfc 791: Internet protocol. 1981.

[78] J. Postel. Transmission control protocol (tcp)-rfc 793, 1981.

[79] R. Potharaju and N. Jain. When the network crumbles: An empirical study of cloud network failures and their impact on services. In *Proceedings of the 4th annual Symposium on Cloud Computing*, page 15. ACM, 2013.

[80] C. Raiciu, C. Paasch, S. Barre, A. Ford, M. Honda, F. Duchene, O. Bonaventure, and M. Handley. How hard can it be? designing and implementing a deployable multipath tcp. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 29–29. USENIX Association, 2012.

[81] I. Recommendation. 200 (1994)— iso/iec 7498-1: 1994. *Information technology–Open Systems Interconnection–Basic Reference Model: The basic model.*

[82] Y. Rekhter and T. Li. A border gateway protocol 4 (bgp-4). 1995.

[83] T. Rentsch. Object oriented programming. *ACM Sigplan Notices*, 17(9):51–57, 1982.

[84] E. Rosen et al. Rfc 3031: Mpls architecture. *IETF Request of Comments*, 2001.

[85] E. Rosen, A. Viswananthan, and R. Callon. Multiprotocol label switching architecture. *IETF RFC 3031*, February 2001.

[86] J. B. Rothnie Jr, P. A. Bernstein, S. Fox, N. Goodman, M. Hammer, T. A. Landers, C. Reeve, D. W. Shipman, and E. Wong. Introduction to a system for distributed databases (sdd-1). *ACM Transactions on Database Systems (TODS)*, 5(1):1–17, 1980.

[87] J. Rozas, J. Sanchez-Delbarrio, X. Messeguer, and R. Rozas. Dnasp, dna polymorphism analyses by the coalescent and other methods. *Bioinformatics*, 19:2496–2497, 2003.

[88] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems (TOCS)*, 2(4):277–288, 1984.

[89] Y. Shintani, M. Inagi, S. Nagayama, and S. Wakabayashi. A multithreaded parallel global routing method with overlapped routing regions. In *Digital System Design (DSD), 2013 Euromicro Conference on*, pages 591–597. IEEE, 2013.

[90] K. Shvachko, H. Kuang, S. Radia, and R. Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[91] H. Smit and T. Li. Is-is extensions for traffic engineering. 2008.

[92] A. Soran, F. M. Akdemir, and M. Yuksel. Parallel routing on multi-core routers for big data transfers. In *Proceedings of the 2013 workshop on Student workhop*, pages 35–38. ACM, 2013.

[93] N. Spring, R. Mahajan, and D. Wetherall. Measuring isp topologies with rocketfuel. In *Proc. of ACM SIGCOMM*, pages 133–145, 2002.

[94] T. Thorsen, S. J. Maerkl, and S. R. Quake. Microfluidic large-scale integration. *Science*, 298(5593):580–584, 2002.

[95] B. Tierney, E. Kissel, M. Swany, and E. Pouyoul. Efficient data transfer protocols for big data. In *E-Science (e-Science), 2012 IEEE 8th International Conference on*, pages 1–9. IEEE, 2012.

[96] R. van der Pol, M. Bredel, A. Barczyk, B. Overeinder, N. van Adrichem, and F. Kuipers. Experiences with mptcp in an intercontinental multipathed openflow network. In *Proceedings of the 29th Trans European Research and Education Networking Conference, D. Foster, Ed. TERENA*, 2013.

[97] S. Vutukury and J. Garcia-Luna-Aceves. Mdva: A distance-vector multipath routing protocol. In *Proceedings of IEEE INFOCOM*, 2001.

[98] H. Wang, H. Xie, L. Qiu, Y. R. Yang, Y. Zhang, and A. Greenberg. Cope: traffic engineering in dynamic networks. In *ACM SIGCOMM Computer Communication Review*, volume 36, pages 99–110. ACM, 2006.

[99] Y. Wang, S. Su, A. X. Liu, and Z. Zhang. Multiple bulk data transfers scheduling among datacenters. *Computer Networks*, 2014.

[100] Y. Xu, B. Leong, D. Seah, and A. Razeen. mpath: High-bandwidth data transfers with massively multipath source routing. *Parallel and Distributed Systems, IEEE Transactions on*, 24(10):2046–2059, 2013.

[101] E. Yildirim, E. Arslan, J. Kim, and T. Kosar. Application-level optimization of big data transfers through pipelining, parallelism and concurrency. *IEEE Transactions on Cloud Computing*, 4(1):63–75, 2016.

[102] W. T. Zaumen and J. Garcia-Luna-Aceves. Loop-free multipath routing using generalized diffusing computations. In *INFOCOM'98. Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1408–1417. IEEE, 1998.

[103] J. C. Zheng Wang. Analysis of shortest-path routing algorithms in a dynamic network environment. *Computer Communication Review*, 22(2):63–71, 1992.

[104] Z. Zhu, W. Lu, L. Zhang, and N. Ansari. Dynamic service provisioning in elastic optical networks with hybrid single-/multi-path routing. *Lightwave Technology, Journal of*, 31(1):15–22, 2013.