

Cell Model of Multidimensional Networks

THESIS

Submitted in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

by

S R K PRASAD TALASILA

Under the Supervision of

Prof. NEENA GOVEAS

and

Co-supervision of

Prof. BHARAT M DESHPANDE



BITS Pilani
Pilani|Dubai|Goa|Hyderabad

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE,

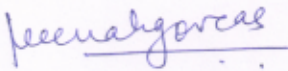
PILANI

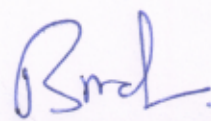
2018

**BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE,
PILANI**

Certificate


This is to certify that the thesis entitled '**Cell Model of Multidimensional Networks**' and submitted by **S R K Prasad Talasila**, ID.No. **2013PHXF0200G** for award of Ph.D. of the Institute embodies original work done by him under our supervision.

Signature of the Supervisor : 
Name in capital letters : **Prof. NEENA GOVEAS**
Designation : **PROFESSOR**
Date : **19/Nov/2018**

Signature of the Co-supervisor : 
Name in capital letters : **Prof. BHARAT M. DESHPANDE**
Designation : **PROFESSOR**
Date : **19/Nov/2018**

Declaration

I, S R K Prasad Talasila, declare that this thesis titled, 'Cell Model of Multidimensional Networks' submitted by me under the supervision of Prof. Neena Goveas and Prof. Bharat M. Deshpande is a bonafide research work. I also declare that it has not been submitted previously in part or in full to this University or any other University or Institution for award of any degree.

Signature of the Student : 

Name of the student : **S R K Prasad Talasila**

ID number of the student : **2013PHXF0200G**

Date : **19/Nov/2018**

Abstract

Multidimensional networks are a useful modeling paradigm for representing the multirelational interactions between entities. Such multirelational interactions frequently occur in the application domains of computer networks, social networks and multi-modal transit networks.

The thesis incorporates a study of multidimensional networks using both the existing and the newly proposed methods. The initial focus is on the connections of a multidimensional network. The thesis starts with an analysis of the measurement data on the broadband connections of the Indian consumers; the analysis is used for understanding the characteristics of network connections. The thesis continues with a demonstration of the multidimensional network modeling in transit networks. The transit timetable of the Indian Railways network is analyzed as a multidimensional network. To demonstrate the advantages of network analysis on the online communities, a structural analysis of network created from the Internet relay chat (IRC) channel logs is presented.

The thesis lists some of the limitations of the existing network models. The existing network models have nodes and connections as passive entities. The existing network models place emphasis on the structural properties of networks; functional and behavioural aspects of networks are not accounted for in the existing network models. To overcome these limitations, the thesis proposes a new network model named the Cell Model. The Cell Model adds functional and behavioural aspects to the existing structural model of a network. The Cell Model makes provisions for the modeling of hierarchical networks. The thesis shows the generality of the Cell Model by showing a representation of the existing network models using the Cell Model.

The utility of the Cell Model has been demonstrated by solving two research problems in the areas of network packet traffic analysis and multi-modal transit scheduling. The Cell Model is used to create a protocol analysis software named Darshini. Darshini has a protocol analysis pipeline that is built using

the Cell Model. Darshini can perform concurrent processing of packets and can perform analysis of selected protocols. In the area of multi-modal transit scheduling, we use the Cell Model to recreate a virtual transit network. In the thesis, multi-modal transit networks of India are modeled as the Cell Model of a network. The Cell Model-based virtual network can process the itinerary search queries and generate a list of feasible itineraries.

The thesis makes the following research contributions.

1. A study of the network conditions experienced by the Indian broadband users. The dataset of the measurement lab's (M-Lab) network diagnostic test (NDT) is used for this study.
2. Representation of the transit timetable of the Indian Railways as multidimensional networks. The case of shared transit vehicles that create multiple entries in the transit timetable is considered in this study. The thesis contains a structural analysis of the resultant multidimensional network.
3. A social network representation and structural analysis for the Internet Relay Chat (IRC) channel logs of Ubuntu and Slackware IRC communities.
4. The Cell Model for the modeling of multidimensional networks. The Cell Model uses the axioms of activeness, message exchange, observational equivalence of FSMs and hierarchy to help model active multidimensional networks. The Cell Model provides flexibility to choose from structural, functional and behavioral views to model the nodes and connections of a network.
5. An implementation of a protocol analysis software using the Cell Model. In the process of developing the implementation, the protocol analysis problem has been redefined as a graph embedding problem. The resulting application, named as Darshini, can perform a selective analysis of the protocols, provides for configurable tradeoffs between memory consumption and execution time, and persists the analysis results into a database.
6. Two implementations of multi-modal journey planner. The first implementation is an extension to the Connection Scan Algorithm (CSA); the proposed extension adds optimization for direct connections between transit stations and for private transport facilities. The second implementation uses the Cell Model for modeling of the multi-modal transit networks of India. Both the implementations provide an interactive journey planner application.

Acknowledgements

I thank my elders and especially my parents, Sri. Satyanarayana and Smt. Devi for encouraging and enabling me to chase my own dreams. I thank my life partner Anusha for bringing joy into my life. I thank our son Roshan for the fond moments we share and for being wise beyond his age. I thank my brother Srikanth and sister-in-law Sudha Rani for their unending support and encouragement.

I thank all my teachers for their lessons and encouragement. Without their guidance, I would have been lost in the wilderness of knowledge.

I thank my doctoral supervisor Prof. Neena Goveas for her mentoring, unending support and patience to read through my half-baked writings. Without her enthusiastic participation and helpful nudges, this work would not have been complete. She truly is a role model for me.

I thank my doctoral co-supervisor Prof. Bharat M Deshpande for his constant support, illuminating advice and hints on research direction. As a head of department during the 2012 to 2017 period, he devoted time from his busy schedule for discussions. He along with Durgesh Samant introduced me to the joy of long distance running. I can't thank both of them enough for introducing me to running that helped me keep the balance during the doctoral studies.

I thank my faculty colleagues Dr. Biju K Raveendran and Dr. Lucy J Gudino for being my doctoral advisory committee members. Both helped set the direction of research. I thank Dr. Ramprasad Joshi for stepping into the shoes of Dr. Lucy J Gudino during the last one year. I thank my faculty colleagues Dr. A Baskar, Dr. Ramprasad S Joshi, Prof. Ashwin Srinivasan, Dr. Rajendra Kumar Roul, Prof. Tarkeshwar Singh, Dr. Debasis Das and Dr. Sreejith Vidhyadharan for the helpful technical discussions during the course of doctoral study. In particular, Dr. A Baskar and Dr. Ramprasad S Joshi have been a constant sounding board of ideas.

I thank my student collaborators Dhruv Shekhawat, Sukanto Guha, Karthik Sathyanarayanan, Sukriti Tiwari, Mihir Kakrambe, Anurag Rai, Sebastin Santy, Aparajita Haldar, Suhas S Pai and Rohan Goel for being research oriented and being the first ones whom I could teach the ideas explained in the thesis.

I thank Anshuli Patil, Sourabh Gawande, Amrita Suresh, PV Srinidhi, Achyudh Ram, Amit Gaiki, Nittin Agarwal, Abhishek Vajarekar, Aman Ahuja, Rohit Kaushik, Krishna Acharya, Kausam Bhat, Bhuvan Gupta and Kapil Kalra for being enthusiastic in their project work and in the exploration and implementation of interesting project ideas.

I thank Rajat Agarwal, Ankshit Jain and Ashu Sharma for help rendered in the conduct of the course work. This enabled me to focus more on enhancing the content delivery during the last three years.

I thank Prof. SD Manjare and Prof. MK Deshmukh for their unconditional encouragement, belief in me and helpful advice. Their encouragement worked like a talisman during the moments of despair.

I thank the faculty colleagues of the department of CS & IS for a memorable six years. The department has truly been an envy of the other units of the BITS, Pilani - KK Birla Goa Campus because of each one of you.

I thank Prof. KE Raman and Prof. G Raghurama for being my teachers and directors. Without their encouragement and support, I wouldn't have made much progress. I thank Prof. JV Rao, Prof. Sasikumar Punnekkat and late Prof. Sanjeev K Aggarwal for their professional advice during the last five years. I thank in-charges of the research funding for sanctioning travel grants to all the conferences that I presented my work in.

I thank all my colleagues at BITS, Pilani - KK Birla Goa Campus for their encouragement and support.

- Prasad Talasila

To

Angulimala, Siddhartha, Virat and Buddha

Contents

Contents	viii
List of Figures	xii
List of Tables	xv
Glossary	xviii
Acronyms	xxi
List of symbols	xxiv
1 Introduction	1
1.1 Multidimensional Networks	1
1.1.1 Social Networks	1
1.1.2 Transit Networks	2
1.1.3 Computer Networks	3
1.1.4 Multidimensional or Multilayer Networks	4
1.2 Models for Multidimensional Networks	5
1.3 Solution Approach	7
1.4 Thesis Outline	7
2 Literature Survey	10
2.1 Basic Definitions	10
2.2 Structural Transformations	12
2.3 Computer Networks	12
2.3.1 Network Connection Characteristics	13
2.3.2 Protocol Analysis	13
2.3.2.1 Parse Graph	14
2.3.2.2 Packet Parsers	15
2.3.2.3 System Architecture	16
2.4 Transit Networks	16
2.4.1 Graph Algorithms	16
2.4.2 Non-graph Algorithms	17
2.5 Social Networks	19
2.6 Structural-Functional-Behavioural (SFB) Views	19

2.6.1	A Combination of SFB Views	20
2.7	Gaps and Objectives	21
3	Structural Analysis of Networks	23
3.1	Objectives	23
3.2	Characteristics of Indian Broadband Connections	23
3.2.1	Motivation	23
3.2.2	Dataset	24
3.2.3	Throughput Analysis	25
3.2.4	Quality of Service (QoS)	28
3.3	Transit Timetables as Multilayer Networks	30
3.3.1	Dataset	30
3.3.2	Definition of Multilayer Transit Networks	31
3.3.3	Algorithms for Creating Multilayer Networks	35
3.3.4	Network Analysis	39
3.4	Structural Analysis on Social Networks	39
3.4.1	Dataset	39
3.4.2	Kinds of Networks	40
3.4.3	Degree Distribution	41
3.4.4	User Communities	42
3.4.5	Experimental Results	44
3.5	Summary	51
4	The Cell Model	52
4.1	Motivation	52
4.2	Notation	53
4.3	Views on Network	55
4.3.1	Structural View	55
4.3.2	Functional View	56
4.3.3	Behavioural View	57
4.4	Axioms of the Cell Model	58
4.4.1	Messages Between Participants	58
4.4.2	Active Participants	59
4.4.3	Equivalence	60
4.4.4	Hierarchical Networks	62
4.5	Generality of the Cell Model	63
4.5.1	Undirected Connections	64
4.5.2	Directed Connections	64
4.5.3	Multiplex Connections	65
4.5.4	Multidimensional Networks	66
4.6	Mapping of the Cell Model to Applications	66
4.6.1	Network Measurements	67
4.6.2	Transit Networks	68
4.7	Results	71

4.7.1	Network Measurements	71
4.7.2	Transit Networks	71
4.8	Implementation Choices	72
4.9	Summary	72
5	Darshini - Protocol Analyzer	73
5.1	Motivation and Problem Definition	73
5.2	Packet Analysis as Graph Embedding	74
5.2.1	Motivation	74
5.2.2	Pipeline as Graph	76
5.3	Graph Embedding: Parse Graph to Pipeline Mapping	77
5.3.1	Motivation for Heuristic Solution	81
5.4	Darshini Architecture	82
5.4.1	Input Data Formats	82
5.4.2	System Overview	84
5.4.3	Analyzer Pipeline	84
5.5	Generic Analyzer Cell (GAC)	85
5.6	Implementation	86
5.6.1	Collaborative Analysis	88
5.6.2	User-Defined Protocol Analysis	88
5.6.3	Measurement Workbench	89
5.7	Experimental Results	89
5.7.1	Dataset	89
5.7.2	User-defined Protocol Analysis	90
5.7.3	Memory Management	91
5.7.4	Throughput	91
5.8	Summary	93
6	Multi-Modal Transit Scheduler	96
6.1	Motivation	96
6.1.1	Itinerary Search	97
6.1.2	Operator Control	97
6.1.3	Solution Approaches	98
6.2	Definitions	98
6.2.1	Connections	99
6.2.2	Connection Array	101
6.2.3	Feasible Itinerary	101
6.2.4	Transfer Times	102
6.2.5	Footpaths and Other Means	103
6.2.6	Direct Connections	104
6.3	Connection Scan Algorithm (CSA)	104
6.3.1	Complexity Analysis	106
6.4	t-CSA Algorithm	106
6.4.1	Complexity Analysis	109

6.5	Cell Model-based Solution Approach	110
6.5.1	Transit Station	110
6.5.2	Transit Connection	110
6.5.3	Path Computation	111
6.5.4	Bridge Functions	113
6.5.5	Itinerary Creation	113
6.6	Cell Model Implementation	115
6.6.1	Architecture	115
6.6.2	Station as Cell (Actor)	116
6.6.3	User Preference	117
6.7	t-CSA Experimental Results	117
6.7.1	Dataset	117
6.7.2	Itinerary Generation Times	118
6.8	Cell Model Experimental Results	118
6.8.1	Implementation Platform	118
6.8.2	Itinerary Generation Times	118
6.9	Summary	120
7	Conclusion	122
7.1	Structural View of Network	122
7.2	The Cell Model	123
7.3	Applications	124
7.4	Summary	125
7.5	Future Work	125
	Bibliography	127
	List of Publications	142
	Biography of the Candidate	144
	Biography of the Supervisor	145
	Biography of the Co-Supervisor	146

List of Figures

1.1	Examples of multidimensional networks.	2
1.2	Different types of multilayer networks.	4
2.1	Sample protocol parse graph with edge weights.	14
3.1	Representativeness of NDT dataset for the calendar year 2012. X-axis enumerates the IP addresses of the clients. Y-axis counts the number of appearances of a client IP address in the dataset for the whole year (2012).	24
3.2	Average throughput of the Indian broadband users for the month of September during the years 2009 - 2014.	26
3.3	Creation of three layers from one example train time table. The timetable considered is a modified version of timetable for the train 11063 MS Salem Express. A tabular representation of the timetable is shown in Table 3.5.	33
3.4	Node degree distributions of multi-layer networks of Indian Railways. The x-axis indicates the degree of a node and the y-axis indicates the fraction of nodes with a given degree. The plots are done on the log-log scale.	37
3.5	Edge weight distributions of multi-layer networks of Indian Railways. The x-axis indicates the sum of weights of all edges of a node and the y-axis indicates the fraction of nodes with a given degree. The plots are done on the log-log scale.	38
3.6	Node degree distributions for the CU and UU presence social networks constructed on the IRC chat channels under study. The data is for January, 2013 for Ubuntu IRC communities.	44
3.7	Node degree distributions for the message exchange social network constructed on the IRC chat channels under study. The data is for January, 2013 for Ubuntu IRC communities and Slackware.	45
3.8	Communities detected on the presence and co-presence social networks of Ubuntu IRC ecosystem for January, 2013. The number of communities has been reduced from 160 to top-30 communities by activity and the number of users have been reduced from more than 15,000 to top-100. These two reductions have been done to better illustrate the community networks.	47

3.9	Communities among top-30 Ubuntu Channels and top-100 users. User nodes are shown in circular shape whereas channel nodes are illustrated in rectangular shape.	48
3.10	Communities in the message exchange social network of chat channels. Each community is given a different color. Cutoff numbers refers to the minimum edge weight threshold considered for community detection purposes. The cutoff numbers were chosen purely for visualization purpose and do not limit the generality of the conclusions.	49
3.11	Communities detected in multi-channel analysis on Ubuntu-devel, KUbuntu and KUbuntu-devel communities for the year 2013. The message exchange graph generated from the three mentioned communities is subjected to an edge weight cutoff of 10. Each node in the given graphs corresponds to one user and the color of a node corresponds to one chat channel. A user is mapped to one chat channel in which (s)he has most presence in the selected time window.	50
4.1	Representation of a connection in a regular graph and in the Cell Model.	54
4.2	A graphical representation of a network using the Cell Model. . .	54
4.3	A typical scenario of message exchange. A message is sent from node v_1 to its connection e_1 . In response, e_1 may send a message to node v_2	58
4.4	Operations on messages performed by a participant of a network. Each keyword denotes the kind of operation performed on a message. The participants are nodes and connections.	59
4.5	Illustration of the axiom of hierarchy. Here v'_2 is a representative for the sub-network consisting of $\{v_1, v_2, v_3, v_4, e_1, e_2, e_3\}$	62
4.6	Elementary operations needed to create hierarchical FSMs. . . .	63
4.7	Modeling of the directed connections using functional view of a connection.	65
4.8	Modeling of multiplex connections using functional view of a connection.	65
4.9	Multidimensional connections and their representation in the Cell Model.	66
4.10	Application of Cell Model to protocol analysis. The hierarchical nature of the Cell Model can be seen in (4.10b) where each node can embed multiple nodes of the graph given in (4.10a).	67
4.11	Generic analyzer cell (GAC) that implements one stage of a pipeline. The same GAC is configured differently for different pipeline stages. 68	68
4.12	A behavioural view for a multi-modal transit station. The station constraints, bridge functions, and generic edges together constitute the state, and behaviour of a station process.	69
4.13	Architecture of the Transport Scheduler application.	69

5.1	Completely connected (K_N graph with self-loops) Pipeline. Here N indicates the number of vertices / pipeline stages. In this figure, we illustrate K_4	75
5.2	Sample protocol parse graph with edge weights.	80
5.3	A sample protocol parse graph specification selecting packets containing only Ethernet-IP-TCP protocols.	80
5.4	Ethernet header specification in P4 language.	81
5.5	IPv4 header specification in P4 language.	81
5.6	System architecture of Darshini.	83
5.7	Protocol analyzer pipeline. Each stage of the pipeline consists of one GAC which is illustrated in part-(b).	83
5.8	Implementation details of the system architecture for Darshini. ES is an acronym for Elastic Search database.	85
5.9	Sequence diagram illustrating collaboration inside BITS Darshini.	87
5.10	A comparison of execution time and packet throughput parameters for different packet parsing tools.	92
6.1	Sample trips $trip_1$ and $trip_2$ undertaken by vehicle identified as ID_1 . The trips $trip_1$ and $trip_2$ together form a loop trip.	99
6.2	A sample query and the generated itinerary.	102
6.3	A behavioural model for a multi-modal transit station. The station constraints, bridge functions, and connections together constitute the state and behaviour of a station process.	110
6.4	The Cell-view of network. Each station is represented by a cell, and the cells are connected as per the underlying connectivity among the stations.	113
6.5	Architecture of the Transit Scheduler application.	114
6.6	Box plot on itinerary search times for two trials. In each trial, we execute 100 random queries.	119
6.7	Query processing times for all the itineraries of the itinerary search results set.	120

List of Tables

2.1	Summary of the literature on the structural view of the multidimensional networks.	18
2.2	Utilization of structural-functional-behavioural modelling approaches in previous work. Pure structural modeling approaches have been covered in Table 2.1.	20
3.1	Annual average throughput for the years 2009 - 2014.	27
3.2	Statistical summary of the four QoS parameters for the month of November, 2010.	28
3.3	Correlation coefficients between different QoS parameters for the month of November, 2010.	29
3.4	Sample entries of train schedules with two of them having two slip-trains. The examples 2 and 3 given here illustrate two different kinds of slip-trains.	31
3.5	A modified version of train schedule for the train 11063 MS Salem Express.	32
3.6	Network characteristics of different layers created from the timetable of the Indian Railway network (IRN).	36
3.7	Description of the datasets under consideration.	40
3.8	Justification for the numerical heuristics used in the analysis framework.	42
3.9	Network statistics for presence and message exchange networks of the IRC communities. The nomenclature is as follows: CC corresponds to Community-Community network; CU corresponds to Community-User network; UU corresponds to User-User network; a prefix of r implies the corresponding network has been reduced to top-30 channels and / or top-100 users.	43
3.10	Curve fit parameters for degree distribution plots of social networks generated from Ubuntu IRC communities. The presence networks are undirected networks, hence only the degree distribution is shown. The message exchange networks are directed networks, hence both the in-degree and out-degree distributions are shown. The given parameters are for the line $\log N_d = -\gamma \log d + K$	43
3.11	Code length for the presence social networks of the Ubuntu IRC communities. The code length has been calculated using InfoMaps community detection algorithm. The results are for the month of Jan, 2013.	46

3.12	Code length for the message exchange social networks of the two communities. The code length has been calculated using InfoMaps community detection algorithm.	48
4.1	The four operations performed by the network participants in terms of properties on a function used under function view.	56
4.2	Operations on messages performed by the participants of network in the Cell Model.	59
4.3	Representation of undirected connections in the Cell Model. . . .	64
4.4	The modeling of different participants using the Cell Model. . . .	67
4.5	The views adopted for different participants of the Cell Model. . .	67
4.6	Connection constraints, path constraints and their compatibility .	70
5.1	API end points for Darshini. The base URL of host and ES URLs are not shown.	86
5.2	Support for measurement strategies in Darshini.	89
5.3	User-defined protocol analysis on a pcap file with 1,508,352 packets. The size of pcap file is 955MB.	90
5.4	Throughput numbers achievable by Darshini.	91
5.5	A comparison of Darshini with other packet processing tools (as of software versions released up to October, 2017.)	95
6.1	Footpaths table.	103
6.2	Other means table.	103
6.3	Algorithmic complexity analysis of CSA vis-à-vis t-CSA algorithms.	109
6.4	The cost function of a connection which is dependent on on time and mode of transport environment variables.	112
6.5	Connection constraints, path constraints and their compatibility .	112
6.6	Query API URL strings	115
6.7	Summary of timetables obtained from the Indian public transport networks.	117

Glossary

Bisimulation Equivalent Two processes (for us, FSMs) are bisimilar, or bisimulation equivalent, if, roughly, they may evolve together in such a way that whenever the first process performs a certain action, the second process is able to respond by performing a matching action, and vice versa. (Hirschfeld et al.).

Bridge Function A function that considers station constraints while extending a path with an edge having edge constraints.

Code Length Size of variable-length code required to enumerate the communities in a given network; typically specified in fraction of bits.

Connection Array A sorted sequence of connections stored in an array. The connections are sorted in the ascending order of their departure time.

Edge An unweighted and undirected binary relation between a node and a connection of the cell model.

Edge Constraints Constraints imposed on an edge.

Inter-layer connections The connections that span more than one layer of a multilayer network.

Itinerary A sequence of connections that satisfy a user query.

Jitter Uncertainty in the arrival time of the packets, often times measured as variation in round trip time.

Latency Time taken by a packet to travel from source to destination, often measured as round trip time.

Multidimensional Network Network with nodes having multiple relations.

Multilayer Network A synonym for multidimensional network.

Multi-modal Transit Network A transit network that operates more than one mode of transport.

Observational Equivalence Also known as weak bisimulation. In observational equivalence, two FSMs are able to perform input-output matching without striving to match the internal states.

Packet Loss Percentage of packets lost in the transmission.

Participant Node or Connection of a network which is modeled as a cell in the cell model of a network.

Path Constraints Constraints imposed on a computed path.

Persistence The act of saving / retaining, used in the context of saving protocol analysis results to a database.

Protocol graph A network created out of protocols (as nodes) and their packet delivery service as provider - user relations (as connections).

Protocol parse graph A protocol graph used for protocol analysis.

Quality of Service Defined by four parameters, namely throughput, latency, jitter and reliability.

Reactive System The systems that react to input with no stop condition.

Route (Computer Networks) Path from source to destination via intermediate nodes.

Route (Transit Networks) A set of all trips that cover the exact same sequence of stations and satisfy FIFO property.

Space of Changes Graph of transit stations where an edge exists between two stations if both the stations are serviced by at least one common train.

Space of Stations Graph of transit stations where an edge exists between two stations if the respective stations are physically connected by a train track.

Space of Stops Graph of transit stations where an edge exists between two stations if the respective stations are adjacent in the timetable of any train..

Station The well-known locations where public transport vehicles stop (like a bus station, railway station or an airport).

Station Constraints Constraints imposed on a station.

Structural Analysis The study of structural properties of networks such as the diameter of the network, betweenness centrality measures, clustering coefficients etc..

Throughput Rate (bits per second) at which data is transferred from source to destination.

Timetable A tuple $(\pi, S, Trips, R, F)$ indicating the scheduled vehicle departures and arrivals in a transit network. π denotes time period of a timetable. S is the set of all stations, $Trips$ is the set of all trips, R is the set of all routes and F is a footpath table containing walking times between nearby stations..

Transfer Time The time required for alighting a connection of one trip and boarding connection of another trip.

Transit network The transport vehicles, stations and timetable of an operator together are referred to as *transit network*.

Union Parse Graph A protocol parse graph containing all the supported protocols. Union parse graph is used by almost all the protocol analyzers.

Acronyms

API	Application Programming Interface.
ASIC	Application Specific Integrated Circuit.
BPF	Berkeley Packet Filter.
CA	Connections Array.
CAIDA	Center for Applied Internet Data Analysis.
CC	Community-Community.
CRAWDAD	Community Resource for Archiving Wireless Data At Dartmouth.
CSA	Connection Scan Algorithm.
CU	Community-User.
DataCat	Internet Measurement Data Catalog.
DB	DataBase.
DCTable	Direct Connections Table.
ES	Elastic Search.
FBS	Frequency-Based Search.
FIFO	First-in-First-Out.
FPGA	Field Programmable Gate Array.
FSM	Finite State Machine.
GAC	Generic Analyzer Cell.

GBR	Guide Book Routing.
HTML	Hyper Text Markup Language.
IC	Incoming Connections Array.
IM	Identity Mapping.
IP	Input Parser / Internet Protocol.
IRC	Internet Relay Chat.
IRN	Indian Railways Network.
ISP	Internet Service Provider.
JVM	Java Virtual Machine.
M-Lab	Measurement Lab.
NC	Network Constructor and Controller.
NDT	Network Diagnostic Test.
OMTable	Other Means Table.
OTP	Erlang's Open Telecom Platform.
P4 Language	Language for expressing packet processing instructions.
PPS	Packets Per Second.
Pub-Sub	Publish-Subscribe.
QC	Query Collector.
QoS	Quality of Service.
RA	Reactive Animation.
RAPTOR	Round-Based Public Transit Routing.
REST	Representational State Transfer.
RIPE	Réseaux IP Européens.

RRD	Round Robin Database.
SFB	Structure Function Behaviour.
TCAM	Ternary Content-Addressable Memory.
TCP/IP	Transmission Control Protocol / Internet Protocol.
TD+-Graph	Expanded Time-Dependent Graph.
TD-Graph	Time-Dependent Graph.
TE-Graph	Time-Expanded Graph.
TNR	Transit Node Routing.
UC	Update Controller.
UML	Unified Modeling Language.
UQC	User Query Controller.
URL	Uniform Resource Locator.
UU	User-User.
VNE	Virtual Network Embedding.
WSGD	WireShark Generic Dissector.

List of symbols

- A_{cc} Adjacency matrix for community-community co-presence network.
- A_{cu} Adjacency matrix for community-user co-presence network.
- A_i Environment variables in the Cell Model.
- A_{uu} Adjacency matrix for user-user co-presence network.
- C Connection – a building block for transit timetables.
- $cost(i)$ Cost function for a connection or a path.
- $CountRTT$ Number of round trip time samples.
- d Node degree.
- E_c Edges of a network which is using the Cell Model.
- $E_{changes}$ Set of all links indicating availability of a train between two stations.
- e_{ij} Weighted, simple edge between vertices.
- $E_{stations}$ Set of all physical links connecting adjacent stations.
- E_{stops} Set of all links formed between consecutive stops of a train.
- γ Exponent for node degree d .
- G_{md} Multidimensional network defined by Kivela et al..
- G_{MLN} Multilayer network defined by Magnani et al..
- G_2 Graph representing protocol analysis pipeline stages.
- G_c Network which is using the Cell Model.

G_{stops} Space of stations graph.

G_{stops} Space of stops graph.

$HCDataOctetsOut$ Number of bytes transmitted including retransmitted bytes.

I A transit itinerary consisting of a sequence of feasible connections.

IM_{ij} Identity mapping of vertices from one layer to another.

J Set of stops at which slip-train(s) separate from the carrier train.

K Exponent for coefficient (scaling factor).

L Elementary layer which when composed with other elementary layers forms a layer of multilayer network..

M Mode of transport.

N Number of stages in the protocol analysis pipeline.

N_d Number of nodes with node degree d .

$OctetsRetrans$ Number of bytes retransmitted.

P Stop sequence prefix for junctions.

p_i i^{th} stage of a protocol analysis pipeline.

π Timetable of a transit network.

r A route which is a set of trips covering the same sequence of stations.

$RTTvar$ Variance in round trip time, also known as jitter.

S Set of states for a finite state machine.

S_{arr} Arrival station of a connection.

S_{dep} Departure station of a connection.

Σ Function input in the functional view of a node or a connection.

s^{in} Initial state of a finite state machine.

SS Stop sequence of a train.

$SumRTT$ Sum of all round trip time samples.

T Indian railways timetable.

t Timetable of a single train.

T_{arr} Arrival time of a connection at S_{arr} .

τ Silent action of FSM.

T_{cwnd} Congestion limited transitions in a NDT test.

T_{day} Time elapsed in a day.

T_{dep} Departure time of a connection at S_{dep} .

Ti Time as an independent variable for the Cell Model of a network.

tr Unique trip identifier in a transit timetable.

T_{recv} Receiver limited transitions in a NDT test.

$trip$ A sequence of feasible connections.

T_{snd} Sender limited transitions in a NDT test.

t_{tr} Transfer time between two connections.

V_1^i All protocols that are assigned to i^{th} pipeline stage.

V_c Nodes of the network which using the Cell Model.

$V_{stations}$ Set of all stations from timetable (same as $V_{changes}$ and V_{stops}).

V_{stops} Set of all stations from timetable (same as $V_{changes}$ and $V_{stations}$).

w_{ij} weight of a connection between two pipeline stages.

X Input messages in the Cell Model.

Y Output messages in the Cell Model.

Chapter 1

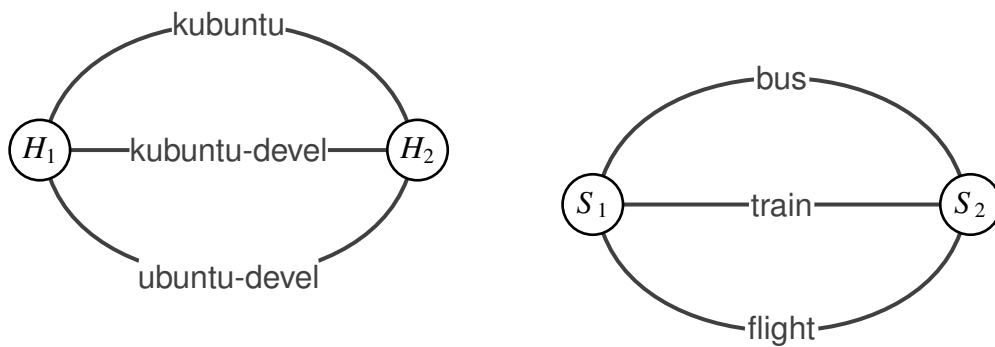
Introduction

1.1 Multidimensional Networks

We live in a connected world. Both humans and technical systems survive and thrive in a web of diverse interactions. Social networks, transportation networks and computer networks are a few examples of such interactions.

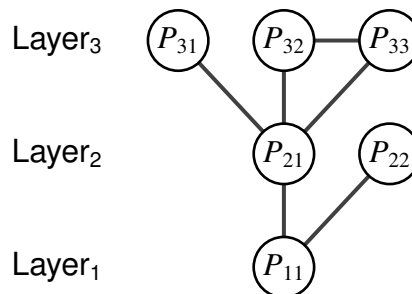
1.1.1 Social Networks

Social networks are a representation of the relations between human beings. Two human beings may be related in more than one way. For example, two humans from the same family may be avid chess players and also work for the same company. Thus these two humans share three different kinds of relations – family, chess and work. This is an example of a network where participating nodes (humans) form multiple relations. Another instance of multiple relations is when two human beings interact with each other via two or more online communities. Facebook, Internet Relay Chat (IRC) channels and mail groups are examples of social networks facilitated by technology platforms; such networks are referred to as *online social networks*. Networks with nodes having multiple relations are referred to as *multidimensional networks*; Here each dimension refers to one kind of relation. Some researchers refer to networks nodes having multiple relations as *multilayer networks* [1]. Multidimensional online social networks are examples of diverse interactions between humans facilitated by



(A) Social network between IRC channel users depicting multiple relations. The text on the connections refers to the IRC channel on which both the users are present.

(B) Transit network between stations depicting multiple relations. The text on the connections refers to the vehicles using a particular mode of transport to service the transit stations.



(C) Protocol graph between protocols depicting use relations. A connection between two protocol nodes refers to a relation from a higher layer protocol to a lower layer protocol.

FIGURE 1.1: Examples of multidimensional networks.

technological systems. Figure 1.1a illustrates the scenario of two human beings interacting with each other on three IRC channels. The depicted IRC channels are kubuntu, kubuntu-devel and ubuntu-devel.

1.1.2 Transit Networks

Transportation networks are structures that convey ... matter (people) or information from one point to another [2]. We consider transport networks that convey passengers using vehicles. For example, India has the long tradition of common transportation providers such as Indian Railways, state transport corporations and Air India etc. These transport service providers ferry passengers between pre-defined locations as per widely published timetables. Such

transport providers who operate services at regular time intervals are known as *transit operators*. The pre-defined locations where public transport vehicles stop (like a bus station, railway station or an airport) are referred to as *stations*. The transport vehicles, stations and timetable of an operator together are referred to as *transit network*. Transport networks have different modes of transport such as bus, train, flight and taxi. A transit network that incorporates more than one mode of transport is called a *multi-modal transit network*. Multi-modal transit networks are examples of multidimensional networks with one mode of transport forming a dimension of the relations between transit stations.

In a multi-modal transit network, we may have many interesting scenarios, like:

1. The same transit network operator uses multiple modes of transport targeting the same or different group of transit stations.
2. Two or more competing transit network operators use multiple modes of transport targeting the same or different group of transit stations.
3. Two or more co-operating transit network operators use multiple modes of transport targeting the same or different groups of transit stations.

In all the three scenarios mentioned above, two transit stations may be serviced by one or more operators using potentially different modes of transport. Figure 1.1b illustrates the scenario of two transit stations being serviced by bus, train and flight. All of these multiple kinds of relations help form multidimensional transit networks. In multidimensional transit networks, a passenger may end up undertaking a journey on two or more networks of different modes.

1.1.3 Computer Networks

A *computer network* is a network consisting of computing devices and their interconnections. A computer network transfers packets from one computing node to another. The bytes are transferred on the network using containers known as *packets*. Network engineers have used layered architecture to achieve scalability in computer networks with the Internet being a prime example of a scalable computer network. The Internet uses layered TCP/IP architecture with each layer containing multiple protocols that help with the packet

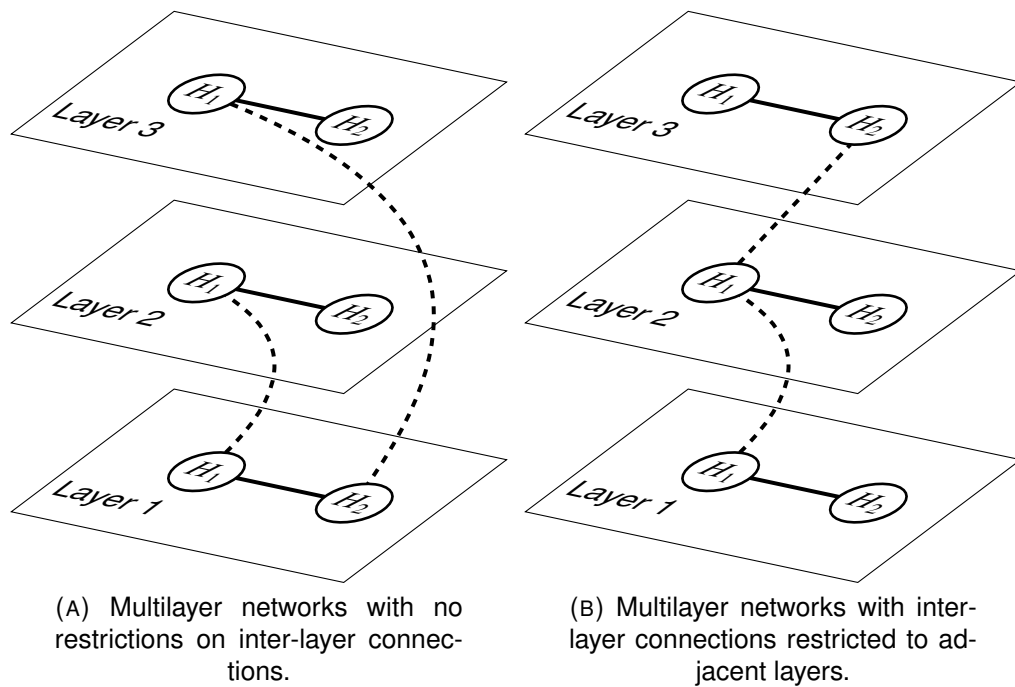


FIGURE 1.2: Different types of multilayer networks.

exchange. Protocols in layer L_i provide packet delivery service to protocols in layer L_i and L_{i+1} . A network created out of protocols (as nodes) and their packet delivery service as provider - user relations (as connections) is called *protocol graph*. Protocol graphs have multiple uses. Protocol graphs provide means of linking protocols operating within a computing node. Protocol graphs also provide means of analyzing network packets inside a protocol analysis software. Protocol analysis software packages are often referred to as *protocol analyzers* and the protocol graphs used by the protocol analyzers are called *protocol parse graphs*. An example protocol parse graph is shown in Figure 1.1c. Since protocol graphs span multiple layers, we can refer to them as multilayer networks. Thus we encounter another instance of multidimensional networks in the form of protocol parse graphs.

1.1.4 Multidimensional or Multilayer Networks

Multidimensional networks and multilayer networks have been well studied in the existing literature [1]. We start our explanation with an illustration of the multidimensional networks described in Sections 1.1.1 to 1.1.3. Figure 1.1 summarizes the kinds of diverse relationships explained in this section. Figures 1.1a

and 1.1b show multiple relations between two nodes. The existing literature on network analysis [1] refers to such networks as multidimensional or multilayer networks.

It is common in multilayer networks to have connections within each layer or across different layers. The connections that span more than one layer are called *inter-layer connections*. One possibility is to allow inter-layer connections between all layers. This is the general case. Another possibility is to place layers in a strict sequence and only allow inter-layer connections between adjacent layers. Figure 1.2 illustrates the difference in these two kinds of multilayer networks. While the social networks and the transport networks allow inter-layer connections between all the layers, computer networks like the Internet only allow connections between adjacent layers.

In the thesis, we use the phrases *multidimensional networks* and *multilayer networks* as synonyms.

1.2 Models for Multidimensional Networks

Network models help us create the abstractions for the study of interactions in networked systems. Suitable notation and semantics for nodes and connections are necessary for any valid network model. Researchers have extended the regular graph models to represent multidimensional networks. The notational emphasis of the existing models has the following highlights.

- A node may be present in all the dimensions. Thus effective set of nodes in a multidimensional network is a Cartesian product of nodes and network dimensions. Let this set be represented as V^d .
- Connections may be any subset of $V^d \times V^d$.
- Some network models attribute functions to connections in order to model weighted, dynamic graphs.

Kivelä et al. [1] provide a review of the existing multidimensional models. A common theme running through many of the existing models is the focus on the connections. *Network structure* is defined by the connections among the

nodes. Existing literature on multidimensional networks focuses on structural representation and analysis¹ [1].

Because of emphasis only on the structural view of networks, the existing network models have the following limitations.

- Nodes and connections are passive entities with no ability to compute or respond.
- All the real-life networked systems have cascade events happening in the system and messages flowing through the system. This requires process oriented models for networks. The existing network models do not handle the cascade events.
- Multiple relations between two nodes get represented as seemingly independent connections in different dimensions. The mutual dependence among different relations of any two nodes is not considered.
- There have been studies on the networks of networks [3] for their structural properties and resilience. But the notion of hierarchy to replace a sub-network with an equivalent node is not considered.
- There is a heavy emphasis on the structural view of the network with little attention paid to functional and behavioural views of network nodes and connections.
- The semantics of the interactions between the participants cannot be clearly specified.

The limitations of the existing models is due to their chosen semantics.

¹The study of structural properties of networks such as the diameter of the network, betweenness centrality measures, clustering coefficients etc. is part of the *structural analysis* of a network.

1.3 Solution Approach

Before we provide details of our approach, we list the requirements for an ideal multidimensional network model. One of the desirable properties of a multidimensional network model is to have nodes and connections that are responsive, i.e., they react to interaction. This reaction is necessary to consider the effect of the environment on the network. Another desirable property is to have the network model support selection from competing representations for nodes and connections. Finally, any new network model needs to support hierarchical representation for sub-networks.

We propose a network model with active (responsive) participants; here we refer to nodes and connections of a network as *participants* in a network. These active participants can compute and provide a process orientation to network phenomenon. We model the interactions between the network participants as message exchanges. In real networks, participants do exhibit the ability to effect the messages / passengers / packets passing through them. This ability to effect the messages passing through the participants is transformational in nature. Thus activeness allows the proposed network model to represent the transformational capability of the participants. Our network model allows a node to be a representative placeholder for a sub-network. This ability of a node to represent a sub-network brings in the notion of hierarchy into our proposed network model.

1.4 Thesis Outline

The thesis is organized as follows.

Chapter 2: Literature Survey In this chapter we provide a survey of relevant work on multidimensional networks. We review the related work from the following fields: structural analysis of networks, multidimensional social networks, multi-modal transit networks and network measurements. After that, we provide an overview of the related work on the Cell Model – our proposed model for the multidimensional networks. We provide a brief

overview of the related work on the structural-functional-behavioural (SFB) model for the network participants.

Chapter 3: Structural Analysis of Networks We use the techniques surveyed in the previous chapter to analyze the properties of network connections in computer networks. We generate the multidimensional networks from the datasets belonging to the domains of transit networks and online social networks. We use the logs of Internet Relay Chat (IRC) channels belonging to Ubuntu IRC ecosystem to generate the multidimensional networks. The generated networks are then analyzed for structural properties such as node degree distribution and clustering coefficient. This chapter contains the study of the datasets from all the three chosen application domains.

Chapter 4: The Cell Model This core chapter propounds the Cell Model – our proposed model for multidimensional networks. At first, we motivate the need for the Cell Model with an example. We then proceed to define the Cell Model of a network. Using the definition, we create three different views of a participant (either a node or a connection). The views we create are structural, behavioral and functional in nature. Each view provides a certain computational model of a network participant.

We also discuss the four axioms used for constructing the Cell Model of a network. The axioms emphasize the activeness of the participants, the message exchange between the participants, the equivalence between two models of a participant and the application of equivalence axiom to create hierarchy in network. We proceed to show the generality of the Cell Model of a network by representing regular, weighted, multiplex and dynamic networks as special cases of the Cell Model of a network. We conclude the chapter with a summary of applications developed using the Cell Model.

In the next two chapters of the thesis, we demonstrate the applications of the Cell Model.

Chapter 5: Darshini - Protocol Analyzer The first application of the Cell Model is in the area of network measurements. We develop a protocol analyzer software named Darshini based on the Cell Model. We first define the problem of protocol analysis as a graph embedding problem.

We then choose a heuristic embedding of the protocol parse graph and proceed to create a concurrent protocol analysis software. We are able to demonstrate the following extra features in protocol analyzers built using the Cell Model: concurrent protocol analysis, selective analysis of protocols, speed vs memory tradeoff and optional persistence. Darshini has an advantage over tshark [4] in terms of selective analysis of protocols and memory consumption.

Chapter 6: Multi-modal Transit Scheduler This chapter provides details on the multi-modal transit scheduler application developed using the Cell Model. The chapter contains a mathematical definition for journey planning application on the multi-modal transit networks. We report two research contributions in this chapter. First, we extend the Connection Scan Algorithm (CSA) [5] to support the direct connections and the private transport facilities. Second, we use the Cell Model to create a concurrent journey planner. In this process, we create the Cell Model compliant representations for transit stations and their connections. Both the applications are able to compute the on-demand itinerary generation for the public transit networks of India.

Chapter 7: Conclusion The final chapter provides a summary of the thesis. We emphasize the deficiencies of the existing models of the multidimensional networks. We provide a brief overview of our efforts to perform structural analysis on computer networks, transit networks and social networks. We then highlight the novelty of the Cell Model. We end the thesis by highlighting the applicability of the Cell Model to problems in the computer networks and transit networks.

Chapter 2

Literature Survey

We present a summary of the related work for the topics discussed in the thesis. In Section 2.1, we review some of the existing definitions for multidimensional networks. In Section 2.2, we show the related work done by earlier researchers on the structural operations of the multidimensional networks. In Sections 2.3 to 2.5, we present the relevant literature on multidimensional networks in the application domains of computer networks, multi-modal transit networks and social networks. Section 2.6 contains a review of literature on the structure, function, behaviour representations of systems. We end the chapter with the identification of the gaps in existing the literature and the listing of our research objectives.

2.1 Basic Definitions

Multidimensional networks have been studied by sociologists, physicists and computer scientists since 1939 [1]. Over the years, researchers have come up with different terminology such as multiplex networks, multilayer networks and multiaspect graphs to describe the multidimensional networks. Recently, Kurant et al. [6] use layered complex networks to describe the transit networks. In their model, the node set of the multilayered network is common across the layers; their work uses weighted and undirected connections. The assumption of drawing the nodes of multiple layers / dimensions from the same node set is prevalent in most of the existing models of the multidimensional networks.

Berlingerio et al. [7] and Nicosia et al. [8] use similar notion of the same node set and multiple relations to represent multilayer social networks. Magnani et al. [9] enhances the notion of multilayer networks by relaxing the condition of the same node set in all the layers. Their work uses the notion of multiple nodes in one layer mapping to a single node in another layer. The definition of multilayer network given by Magnani et al. [9] is as follows.

Definition 2.1. A Multilayer network is a tuple $G_{MLN} = (G_1, \dots, G_n, IM)$ where $G_i = (V_i, E_i), i \in 1, \dots, n$ are network layers and IM (Identity Mapping) is an $n \times n$ matrix of node mappings, with $IM_{ij} : V_i \times V_j \rightarrow \{0, 1\}$.

The identity mapping function is responsible for mapping one or more nodes of one layer to a single node in another layer. The network models proposed by Kurant et al. [6] and Berlingerio et al. [7] can be derived from Definition 2.1 by imposing an additional restriction of $IM_{ij} : V_i \times V_j \rightarrow \{0\}$ for $i \neq j$.

The models discussed so far do not consider cross-layer connections. Boccaletti et al. [10] adds cross-layer edges to the multilayer network model and uses the same node set in all the layers.

Wehmuth et al. [11] present a model for time-varying graphs using the notion of *aspect*. They use aspect to generate multiple elementary layers with time being one aspect in the time-varying graphs. Kivelä et al. [1] and Wehmuth et al. [12] provision for multiple aspects in their network model. Each one of the multiple aspects can generate multiple elementary layers. The multilayers / multidimensions of the network are formed by a Cartesian product on all the elementary layers. Mathematically, we can express the network model as follows [11].

Definition 2.2. A multidimensional network $G_{md} = (V_{md}, E_{md}, V, L)$ with V being the set of vertices and L being the set of layers. The set of layers are generated from the Cartesian product of elementary layers, i.e., $L \subseteq L_1 \times L_2 \times \dots \times L_k$ with each $|L_i|$ having a finite value. Each L_i comes from one aspect of the graph. The node set and the connection set are defined as $V_{md} \subseteq V \times L_1 \times L_2 \times \dots \times L_k$ and $E_{md} \subseteq V_{md} \times V_{md}$ respectively.

Our proposed Cell Model does not require the notion of different node sets for different layers. We work with a single node set and a single connection set. We use the term *environment variable* to represent one aspect of a multilayer graph.

2.2 Structural Transformations

Structural transformations of multidimensional networks has not received much attention from the researchers. One of the earliest attempts to perform structural transformation on multi-modal transit networks was time expanded graphs (TE-graphs). Schulz et al. [13] represent time varying networks as time expanded networks which are static networks. The resulting networks are used for the computation of transit itineraries.

Another useful structural transformation is the hierarchical aggregation. In this transformation, one network is reduced to a smaller network of representative nodes. The shortest path computations done on the reduced smaller networks approximate the intensive computations done on the complete network. The hierarchical aggregation technique is used extensively in hierarchical node routing, contraction hierarchies and transit node routing schemes [14].

Researchers have recognized the redundancy in some of the dimensions of a multidimensional network. De Domenico et al. [15] propose a measure for structural reducibility of a multidimensional networks where by the redundant layers are removed from the network.

Wenlei Xie et al. [16] propose three different graph models – aggregate model, time window model and edge decay model – for study of online social networks. We use the aggregate graph model for the study of the social networks.

We represent the connections using function and behaviour. We also replace a sub-network using a node. These actions have an effect of inducing the required structural transformations.

2.3 Computer Networks

Our work in the computer networks area corresponds to network measurements topic. We accomplish two research tasks in network measurements. First we perform the statistical analysis of the Indian broadband connections. Then we create a protocol analyzer software named Darshini. The related work on both these research tasks is presented in this section.

2.3.1 Network Connection Characteristics

We rely on distributed Internet measurements for obtaining data on the connection characteristics of the Indian broadband users. Such distributed Internet measurements are performed by prominent organizations such as Measurement Lab (*M-Lab*), *CAIDA*, *WAND*, *Route Views* and *RIPE*. Researchers take either an active or a passive approach to Internet measurements [17]. Route Views predominantly performs passive measurements where as WAND, RIPE and CAIDA perform both. Lehr et al. [18] lists popular platforms for broadband speed measurements and the discrepancies among the competing platforms. M-Lab specializes in active Internet measurements. M-Lab provides the largest collection of the Internet measurement and performance data. The M-Lab data sets are available through either the cloud storage in raw format or the Google cloud platform via BigQuery interface.

We select the network diagnostic test (NDT) tool for our network measurement and analysis task. NDT relies on the Kernel Instrument Set (KIS) which was developed as a part of the Web100 project [19]. The Web100 project enables a passive per-connection monitoring of the TCP state [20]. Much of the NDT test data set comes from the KIS probes inserted into the Linux kernel.

Apart from NDT, other tools like *iperf*, *Speedtest.net* and *grenouille.com* can also perform throughput measurement. Attempts have also been made to perform customized client-side measurements with devices such as *SamKnows* and *BISMark* [21]. Among all the other tools, *Netalyzr* comes closest to NDT in functionality [22]. We use the NDT dataset to look at the ground reality of broadband customer's QoS experience.

2.3.2 Protocol Analysis

In Darshini, we utilize the graph embedding for creating an implementation of the protocol parse graph. Graph embedding is a familiar concept in the context of virtual networks [23]. In virtual networks, user networks are embedded in the substrate network offered by the network service providers. The problem of virtual network embedding (VNE) has been proven to be an \mathcal{NP} -hard [23]. However, researchers have produced heuristics-based algorithms for solving

the VNE problem in real-time [24]. We use a similar approach to formulate the graph embedding problem for packet parsers. Even though there are significant overlaps between VNE and graph embedding problem for protocol analyzers, there are quite a few differences as well. One major difference is in the mapping of links. In VNE, the links of user networks are mapped onto the physical paths of provider networks. In packet parsers implemented in a machine / cluster, an edge of protocol parse graph gets mapped to one of the edges of the provider graph. In protocol analyzers, the edges of the provider graph are rarely physical links.

Darshini also requires as input the specifications of the following elements: protocol headers, protocol parse graph, protocol analysis pipeline and persistence module. The rest of this section describes literature on each of the above mentioned sub-areas.

2.3.2.1 Parse Graph

The idea of protocol parse graph is very old [25, 26]. A typical protocol parse graph of the network traffic is shown in Figure 2.1. A protocol parse graph can be implemented in hardware, software or a mix of both hardware and software. One popular form of hardware implementation is the synthesis of the parse graph state machine onto ASICs with TCAM [27] and onto the commercially available FPGA architectures [28]. A survey of the hardware implementations

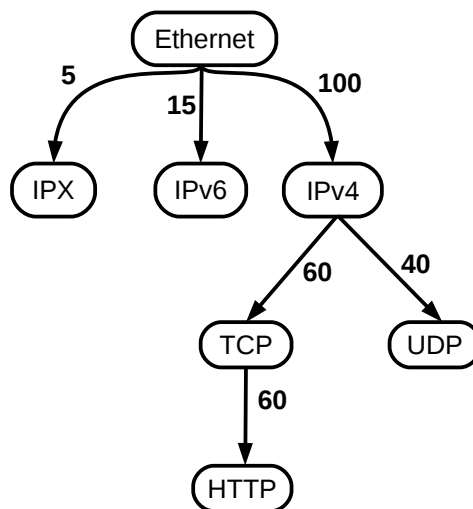


FIGURE 2.1: Sample protocol parse graph with edge weights.

of the protocol parse graph techniques is available [29]. Examples of software implementation for the parse graphs are Ntop[30], Wireshark [31] and tcpdump [32].

Software / hardware implementations of the protocol parse graphs can either be a fixed or a programmable kind. A fixed parse graph can only parse the protocol sequences that are part of the given parse graph. On the other hand, a programmable parse graph can dynamically select a parse graph at the run time. A variation of the programmable protocol parse graph called Berkeley Packet Filter (BPF) is used in the tcpdump [32] and Linux socket filter [33]. In these instances, the parse graph is typically used to select packets of interest. Unlike previous implementations which use fixed parse graph for packet analysis, Darshini uses a programmable parse graph for packet analysis.

Darshini parses the incoming packets in order to analyze the protocol stack of the packet. P4 language [34, 35] presents a parse graph notation that is suitable for both hardware and software implementations. We use P4 language to represent the protocol parse graph.

2.3.2.2 Packet Parsers

There have been attempts to implement a pipeline protocol parse graph to enhance the protocol analysis throughput. The architectural solutions proposed by [27, 28, 36, 37] are some of the pure hardware implementations of the packet parsers.

We adopt the Generic Protocol Parser Interface (GPPI) of Benáček et al. [36, 38] for our design of generic analyzer cell (GAC). The high frequency extractor M2 and the GPPI discussed by Benáček et al. [36, 38] together form a serially connected pipeline implemented in hardware. Wireshark [31] implements parse graph in software without pipelines. Our analyzer pipeline is a complete software implementation with support for concurrency. In addition, we introduce a logical bus connectivity to this pipeline by using feed forward / feedback line between all stages of the pipeline. We support multiple protocols per pipeline stage. There will always be a definite speed gap between the header extraction work of the parser and the table lookup / persistence section where the headers may either be interpreted or stored. The speed mismatch is often compensated

by the use of buffers [27]. The problem has also been explained in terms of the fast-path and slow-path of network processing algorithms [39]. We adopt the fast and slow path separation approach to solve the speed mismatch between protocol analyzers (header extraction) and persistence module.

2.3.2.3 System Architecture

Packet capture and protocol analysis software tcpdump, tshark and Wireshark have been developed as stand alone packet processing utilities. Ntop [30, 40] is a web application with dynamic plug-in system for customization of persistence, analysis and view. Darshini has also been designed as a web application with support for adding new protocols. One major difference between Ntop and Darshini is the user-defined protocol analysis. Darshini allows users to specify parse graph for protocol analysis while Ntop performs basic protocol analysis for the full suite of its supported protocols.

2.4 Transit Networks

In this section, we review the related work on the multi-modal transit networks.

2.4.1 Graph Algorithms

Algorithm researchers have made noteworthy contributions to improve real-time transit query performance on public transport networks. Initial research efforts focused on adoption of best algorithms developed for routing on road networks to the public transport networks [14, 41]. Over the years, researchers have discovered significant differences between the two problems, namely routing on road networks and best itinerary on public transport networks. Bast et al. provides an overview of these differences and a summary of query speed-up techniques for public transport (timetable) networks [42]. Bast et al. also provide a detailed survey of algorithms used on road networks and public transport networks [14].

Transit scheduling algorithms fall into either graph based or non-graph based approaches. Over the years, researchers converged onto time-expanded (TE) graphs and time-dependent (TD) graphs as two appropriate graph-based models for representing public transport networks (also known as timetable networks) [43, 44]. In TE graphs, each arrival and departure event is represented as a vertex and the travel time as a weighted edge [13]. TE graph model was originally proposed by Schultz et al. and then extended by Müller-Hannemann et al. [13, 45]. The widely used General Transit Feed Specification (GTFS) is a TE graph based representation of timetable [46].

Delling et al. contract the arrival and departure nodes of TE graphs completely [47]. They also parameterize the contraction of transfer nodes¹. Bast et al. modify the edges of TE graph model to represent periodic trips. The resulting transfer patterns algorithm requires costly preprocessing but provides query responses in a few milliseconds [48].

Orda et al. propose time dependent (TD) graph model and give shortest path algorithm [49, 50]. Brodal et al. use TD graphs for executing fast itinerary search queries [51]. In basic TD graph model, vertices represent stations and edges represent trips, routes. In expanded TD graph model (TD+), even routes have station-specific vertices; such expanded TD models have been referred to as train route graphs [43]. Geisberger et al. proposes station graph model which is an application of contraction hierarchies preprocessing technique to TD graph [52].

2.4.2 Non-graph Algorithms

Connection Scan Algorithm (CSA), RAPTOR, GBR and FBS are the non-graph based approaches to transit routing [5, 53, 54, 55]. CSA represents timetables using connections; Strasser et al. shows equivalence between running CSA and computing shortest path queries on TE-graphs. Connection scan algorithm (CSA) searches timetables in linear time [56]. *RAPTOR*, short for Round-based Public Transit Routing, utilizes trips and routes for itinerary computation and is

¹Station at which passengers change vehicles

equivalent to queries on TD-graphs [53, 55]. Of the CSA and RAPTOR approaches, CSA uses simpler data structures and exploits cache locality properties of modern microprocessors [5, 57, 58]. *GBR*, short for Guide Book Routing, utilizes network flow-based approach to generate time independent itineraries for a given timetable [54]. *FBS*, short for Frequency-Based Search, shows a way to extract and represent frequent-trips from timetables [55].

CSA uses connection as a fundamental unit for decomposing timetables and searching these decomposed timetables. Such a clean representation allows quick update of searchable timetables for possible delays, cancellations and special trips on busy occasions. An update of timetables only requires changing effected connections of CSA. CSA has been proven to be faster than RAPTOR, TE and TD-graph based algorithms for one-to-one queries [5, 14]. Only transfer patterns is faster than CSA, but transfer patterns technique requires costly preprocessing [14, 48] and is not easily amenable to changes.

Our work adopts techniques outlined in CSA and transfer patterns papers [5, 48]. We propose an extension to CSA using DCTable. DCTable approach is similar to direct connections of transfer patterns [48]. Such a direct connection table approach has given rise to transit node routing (TNR) algorithm – the best performing algorithm for road networks in terms of query response time [62]. The OMTTable extends footpath table of CSA to incorporate frequent trips. We include private transport networks in OMTTable; Consequently, connection array

TABLE 2.1: Summary of the literature on the structural view of the multidimensional networks.

Topic	Related Literature
Network models	Review by Kivela et al. [1], protocol parse graphs [26], CSA [5], TE-graphs [13], TD-graphs [43]
Structural transformations	TE-graphs [13], hierarchical aggregation [14], aggregate graphs [16]
Community detection	Infomaps [59]
Connection characteristics	NDT [19], Web100 [20]
Hierarchy	Nested networks of ecological systems theory (EST)[60], recursive network architecture [61]

gets augmented with private transport networks. Similar approaches such as GBR and FBS affirm the need to concisely represent frequent connections of a timetable. Our other means table (OMTable) can incorporate private transit facilities available to users, thus bridging the gap between public and private transport networks.

2.5 Social Networks

We take up the study of social networks using the Internet relay chat (IRC) channels as an example. Vladimir Gligorijević et al. [63] and Tanmay Sinha et al. [64] study the structural properties of Ubuntu IRC communities. Paul Mutton et al. [65] studies the aggregate network structure and properties of the IRC communities. The existing research work focuses on the structural analysis of a single IRC channel while we extend the work to multidimensional structural analysis.

Community detection has also received attention from social network researchers [66]. Informally, a cluster or community can be considered as a set of entities that are closer to each other [67]. Among the existing community detection algorithms, Infomaps [59] is a community detection algorithm that works natively on directed and weighted graphs. Lee et al. [68] perform community detection on multilayer networks. We perform the community analysis on multiple IRC channels using the InfoMaps algorithm. Unlike previous works, we create presence / co-presence networks and perform community analysis on these networks.

2.6 Structural-Functional-Behavioural (SFB) Views

We define a view as a way of looking at a network. Since a network is modeling the real world interactions, the abstract network model can contain as much detail as we want and nothing more. Where needed, it is advantageous to use functional and behavioural views along with the structural views for the participants of a network.

TABLE 2.2: Utilization of structural-functional-behavioural modelling approaches in previous work. Pure structural modeling approaches have been covered in Table 2.1.

Approach	Prior Work Utilizing the Approach
Function	Review [69, 70]
Behaviour	Review [71], reactive animation (RA) [72]
Structure + Function	signal flow graphs [73]
Function + Behaviour	GemCell [74] and BioCharts [75]
Structure + Behaviour	node-centric programming [76]

All the network models we have reviewed so far take a structural view of the networks. Table 2.1 We present a concise summary review of the literature review given in Sections 2.1 to 2.4.

There has been extensive work on the behavioural modeling of systems (node is an abstraction for an entity or system). Behavioural models are very powerful and help describe network participants that continuously receive and react to external input; The systems that react to input with no stop condition are called *reactive systems*. Behavioural models are sufficient for describing reactive systems [71, 74]. Harel et al. use a hierarchical behavioural modeling for creating reactive animation (RA) [72] which is nothing but a reactive system with an animated output.

2.6.1 A Combination of SFB Views

In creating the Cell Model, we utilize the structure, function and behaviour (SFB) modeling approaches. The existing literature on the use of SFB modeling approaches is summarized in Tables 2.1 and 2.2.

Functional modeling of systems is extensively used in engineering disciplines. For example, the signal flow graphs [73, 77] used by electrical engineers is a prime example of a network model with vertices acting as junctions (signal aggregators) and connections acting as transformers (specified by a transfer function). Here nodes and edges are both acting as transformers of signals passing

through them. Eisenbart et al. [69] provide a review of functional modeling in engineering domains.

Amir-Kroll et al. [74] utilize a combination of function and behavior approaches to model a biological cell; their model is called GemCell. The behavioral part of the system is specified using executable state charts and message sequence charts. Their approach reaffirms the sufficiency of function and behavior views to describe a complex reactive system such as biological cell. From the same research group, Kugler et al. [75] propose BioCharts as another model of biological systems. BioCharts use multiple co-operating state machines within one entity to model biological processes following a mix of structural and behavioral approaches.

John Krogstie [71] provides an overview of eight different modeling perspectives for process modeling. He also concludes that structural view provides a static view of the system, whereas behavioral and functional views provide a dynamic view of the system. He also asserts that a mix of functional and behavioral views can aptly describe most modeling scenarios.

Another approach taken by researchers is to model concurrent computation as node-centric programming, i.e., a computation problem is mapped onto a graph with computation done in nodes and communication dictated by the graph topology. McCune et al. [76] provide a survey of node-centric programming in distributed graph processing.

2.7 Gaps and Objectives

Based on our survey of the existing literature, we have identified the key open research issues as the following:

- Nodes and connections of the existing network models are passive entities with no ability to compute or to respond.
- Real-life networked systems have cascade events happening in the system, messages flowing through the system. This requires process-oriented models for networks. The structural models of networks do not provide the semantics for modeling processes in networks.

- Multiple relations between two nodes get represented as seemingly independent connections in different dimensions. The mutual dependence among different relations of any two nodes is not considered.
- There have been studies on the networks of networks [3] for their structural properties and resilience. But the notion of hierarchy to replace a sub-network with an equivalent node is not considered.

The objectives of our proposed research are to:

1. Create datasets for multidimensional networks from the application domains of transit networks and social networks.
2. Apply the existing network models to perform structural analysis on the multidimensional networks found in social networks and transit networks.
3. Develop a new network model for multidimensional networks. The new network model should be active, must have a process view and account for hierarchy in networks.
4. Demonstrate the proposed model to the research problems in the application domains of protocol analysis and multi-modal transit networks.

Chapter 3

Structural Analysis of Networks

3.1 Objectives

In this chapter we detail our work in the study of multidimensional / multilayer networks. We use the term multilayer networks in this chapter in order to maintain consistency with the published literature. The term multilayer networks has to be understood as multidimensional networks. We start our discussion with the study of connection characteristics (Section 3.2). We then move to the study of multilayer networks in transit networks (Section 3.3) and social networks (Section 3.4).

3.2 Characteristics of Indian Broadband Connections

3.2.1 Motivation

Indian ISP operators form a multilayer network with the backbone ISP forming the common nodes linking the layers. A problem with such networks is the connection quality. The connection quality is defined by a combination of parameters such as high speed, low latency, low packet loss etc.; consumers are also willing to pay a premium for good QoS. The Internet service providers (ISPs) claim good QoS experience on their networks; yet dissatisfied consumers have

contested the QoS claims of the ISPs. The main problem is the lack of objectivity in assessing the QoS parameters of an Internet connection.

To objectively assess the difference between the claimed and the experienced conditions on broadband connections, we utilize the dataset generated by the network diagnostic test (NDT) tool hosted on the measurement lab (M-Lab) platform [78]. NDT is an active measurement tool fashioned in client-server paradigm [79, 80]. NDT helps measure the performance of a network connection. Volunteer users perform NDT test between their computer and an M-Lab server [81]. We use the NDT dataset for the years 2009–2014 to perform QoS analysis [78]. We make inferences on QoS policies deployed on the backbone networks of ISPs.

3.2.2 Dataset

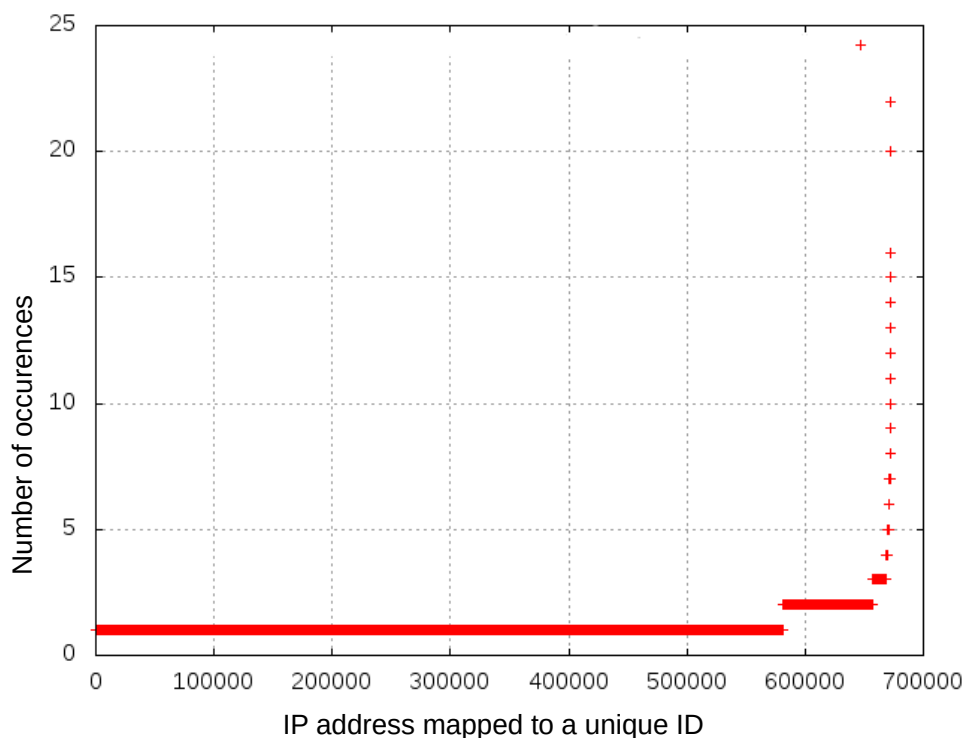


FIGURE 3.1: Representativeness of NDT dataset for the calendar year 2012. X-axis enumerates the IP addresses of the clients. Y-axis counts the number of appearances of a client IP address in the dataset for the whole year (2012).

The NDT dataset contains test results for all the countries [82]. We perform India-centric analysis on NDT dataset for the years 2009-2014. We check the

representativeness of the dataset by checking the number of times a user repeats NDT test in a year. Figure 3.1 shows the repetitiveness of users (IP addresses) in the NDT dataset for the year 2012. For the sake of compact plot, each user IP address from NDT dataset has been mapped to a unique number in an ascending order number series starting at number 1; these numbers are placed along the x-axis. The number of times a client IP address appears in the dataset is indicated on the y-axis. As Figure 3.1 shows, most of the clients appear just once in the NDT dataset over the sample period of one year, thus attesting to the diversity of the NDT dataset.

In the sample space for the calendar year 2012, only 2.2% of the IP addresses appeared more than twice in the entire year, and 88.6% of the values appeared exactly once over the duration of the entire year.

All the NDT tests performed by the M-lab platform are archived in the M-Lab dataset table of the Google BigQuery database [83]. Google BigQuery database itself is hosted on the Google cloud platform. All fields of the M-Lab dataset table can be classified into two categories: connection-specific information and test-specific information. The fields related to connection-specific information record the IP addresses, host names, ISP identity and geolocation information (latitude and longitude) of the client. The test-specific information fields record each of the web100 kernel instrumentation set (KIS) variables as one field in the database table.

3.2.3 Throughput Analysis

The average throughput of a broadband connection is calculated using Equation 3.1 [84].

$$\text{Throughput} = \frac{\text{DataOctetsOut}}{8 * (T_{\text{recv}} + T_{\text{cwnd}} + T_{\text{snd}})} \quad (3.1)$$

Where,

DataOctetsOut = Number of bytes transmitted

T_{recv} = Receiver limited transitions

T_{cwnd} = Congestion limited transitions

T_{snd} = Sender limited transitions

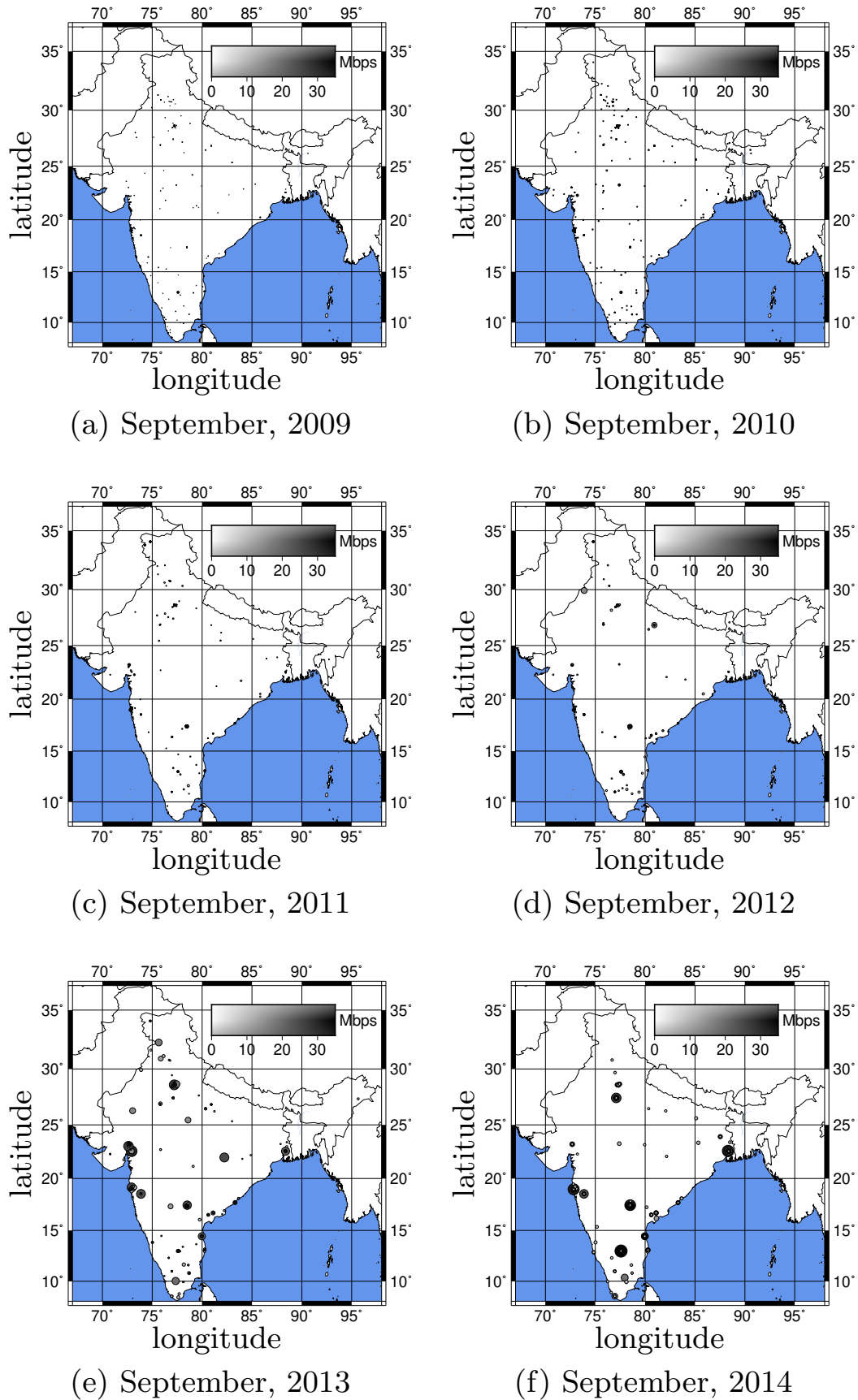


FIGURE 3.2: Average throughput of the Indian broadband users for the month of September during the years 2009 - 2014.

The unit for throughput numbers mentioned in this section is Mbps, unless stated otherwise.

Each of the data points in the NDT dataset contains latitude, longitude, throughput information along with the date of measurement. We use September as the reference month for the annual comparison of the throughput metric.

TABLE 3.1: Annual average throughput for the years 2009 - 2014.

Year	Average Throughput (Mbps)
2009	0.437
2010	0.532
2011	0.548
2012	0.644
2013	0.543
2014	0.711

The generated throughput graph sequence shows stark contrast between the initial and final phases of the selected five year period (2009 - 2014), an example of which is displayed in Figure 3.2. The throughput of top-1000 Indian broadband users measured across six years is plotted in Figures 3.2(a) to 3.2(f). The bandwidth of a measurement is represented using a gray shade and the radius of a circle. Higher bandwidth measurements get darker shade of gray and also become larger circles. As we can see in Figure 3.2, the measured throughput see gradual increase in the 2009 - 2014 time period. An interesting observation over the same period is the considerable increase in the number of subscribers in non-metropolitan areas. In addition to that, we observe that the maximum throughput value rises from 2.98Mbps in September 2009 to 21.29Mbps in September 2014.

Furthermore, Table 3.1 shows the average throughput for the years 2009-2014. A steady growth has been observed during the years 2009-2012 and 2013-2014. According to the Telecom Regulatory Authority of India (TRAI), the speed of broadband is largely dependent upon three factors: bandwidth utilization, latency and contention ratio [85]. Except for four months (May-August) in 2013, we observe a month-on-month increase in average throughput during 2009 - 2014 period.

3.2.4 Quality of Service (QoS)

The technical Quality of Service (QoS) is defined by four parameters, namely throughput, latency, jitter and reliability. We assess the throughput values of a broadband connection as per the procedure outlined in Section 3.2.3. The jitter, latency and reliability values are computed from the kernel instrumentation set (KIS) variables of the web100 project [20] using the following equations.

$$\begin{aligned}
 \text{Jitter} &= \text{RTTVar} \\
 \text{Latency} &= \frac{\text{SumRTT}}{\text{CountRTT}} \\
 \text{Reliability} &= \frac{\text{OctetsRetrans}}{\text{HCDataOctetsOut} - \text{OctetsRetrans}}
 \end{aligned} \tag{3.2}$$

Where,

- RTTVar = The round trip time (RTT) variation used in calculation of the RTT.
- SumRTT = The sum of all sampled round trip times.
- CountRTT = The number of timeouts.
- OctetsRetrans = The number of bytes retransmitted in one TCP connection.
- HCDataOctetsOut = The number of bytes transmitted during one TCP connection.

We consider four Quality of Service (QoS) parameters, namely throughput, latency, jitter and reliability of a broadband connection. All the four parameters are measured separately by NDT. NDT records reliability as the percentage of packets dropped during a test, thus a smaller number indicates a more reliable connection. We tabulate the statistical summary of the four aforementioned

TABLE 3.2: Statistical summary of the four QoS parameters for the month of November, 2010.

	Mean	Std. Dev.
Throughput	0.5Mbps	0.3734Mbps
Jitter	57.282ms	49.167ms
Latency	427.18ms	218.22ms
Packet Loss (Reliability)	4.62%	7.37%

TABLE 3.3: Correlation coefficients between different QoS parameters for the month of November, 2010.

	Throughput	Jitter	Latency	Reliability
Throughput	1	-0.296	-0.306	0.012
Jitter		1	0.817	-0.033
Latency			1	-0.024
Reliability				1

QoS parameters in Table 3.2. We calculate the correlations between all possible pairs of QoS parameters using the standard correlation equation of two random variables. The results are shown in Table 3.3.

We observe from Table 3.3 that throughput has a weak negative correlation with all the other parameters. This could be because higher throughput is generally associated with better and more expensive broadband connections, and thus latency and jitter would be low and reliability high. Furthermore, we observe that even the correlation factor is constant (~ -0.3) between throughput and two other parameters, namely jitter and latency.

Another revelation is the strong correlation between latency and jitter. In India, low latency seems to guarantee timely packet arrival (low jitter) and vice versa. Though a correlation between latency and jitter is expected, it is perhaps surprising that only latency and jitter parameters are strongly linked and not reliability.

Putting all of these facts together, we can draw two conclusions. First conclusion is, there is probably a significant deployment of DiffServ across the ISP networks in India. DiffServ traffic classes make the separation of traffic into four classes possible and can also explain the strong correlation between latency and jitter [86]. The high priority packets are serviced first using expedited forwarding of Diffserv [87]; such a packet scheduling algorithm reduces the jitter experienced by high priority (i.e., low latency) packets.

Second conclusion is, throughput determines the DiffServ traffic classes. Throughput (in layman's terms, connection speed) is the most visible and marketed term

in broadband connectivity. Broadband consumers who purchase high throughput connection are placed in higher DiffServ classes. Better treatment for premium customers could explain the negative correlation between throughput and the other three QoS parameters.

Reliability does not have any correlation with either latency or jitter, in fact, it is almost totally independent. In India, reliability seems to be only a weak function of the throughput, otherwise, there is really no way of guaranteeing high reliability.

We conclude our study of the connection characteristics. In the next two sections, we focus on techniques for creation and analysis of multilayer networks.

3.3 Transit Timetables as Multilayer Networks

In this section, we look at a technique for creating the multilayer representation of transit networks. We create a three-layer network from the public transit timetables of Indian Railways. We explain the utility of these distinct layers for railway passengers and Indian Railways. We also perform the structural analysis on the three-layer network.

3.3.1 Dataset

We consider the public timetable of Indian Railways available on their website [88]. The timetable contains schedule of express and mail type of train services. The timetables of 2,018 express and mail trains are available on the website. We consider the timetables for a one week time period.

Indian Railways uses the practice of sharing transit vehicles; the practice is called *slip-train*. Under slip-train practice, some coaches of a given train are separated from it at a pre-determined junction. The coaches thus separated are attached to a different train which takes the coaches toward their destination. The slip-trains are identified in the train timetable by giving different route number to each slip-train. Earlier studies using the Indian Railways dataset did not consider this kind of trains.

TABLE 3.4: Sample entries of train schedules with two of them having two slip-trains. The examples 2 and 3 given here illustrate two different kinds of slip-trains.

Source Station	Junction	Route No.	Destination Station
<u>12621 MAS NDLS Express</u>			
Chennai	N/A	1	New Delhi
<u>12780 Goa Express</u>			
H. Nizamuddin	Londa Junction	1	Vasco Da Gama
		2	Hubballi Junction
<u>17603 Kacheguda - Yesvantpur Express</u>			
Kacheguda	Guntakal Junction	1	Yesvantpur
		2	Hubballi Junction
		3	Vasco Da Gama

Sample entries obtained from the Indian Railways Network (IRN) timetable are shown in Table 3.4. The first train bearing 12621 as train number does not have any slip-train. The second train bearing 12780 as train number requires that a few coaches be detached from the train at the Londa Junction. The detached coaches are then attached to another train which takes them towards the Hubballi station; these coaches are identified with route number 2. Similarly, the third train bearing 17603 as the train number has two slip-trains; both the slip-trains get separated from main train at the Guntakal Junction.

3.3.2 Definition of Multilayer Transit Networks

Timetable of any transit network is a set of timetables of all the vehicles that are part of the transit network. In the case of IRN, the timetable consists of a set of trains each bearing a unique train number and servicing a sequence of stations.

We start with the publicly available IRN timetable to create a three-layer network. The three layers are named as space of stops, space of stations and the space of changes. All the three layers share a common node set; the set of nodes is created by collecting the names of all the train stations available in the IRN timetable. The *space of stops* layer has an edge between two stations if the respective stations are adjacent in the timetable of any train. The *space of stations* layer has an edge between two stations if the respective stations are

physically connected by a train track. The *space of changes* layer has an edge between two stations if both the stations are serviced by at least one common train. Kurant et al. [89] contains a more detailed explanation of these terms.

A summary of the notations used in the rest of the section is as follows.

space of stops graph:

$$G_{stops} = (V_{stops}, E_{stops}) \quad (3.3)$$

V_{stops} = set of all stations from timetable

E_{stops} = set of all links formed between
consecutive stops of a train

space of stations graph:

$$G_{stations} = (V_{stations}, E_{stations}) \quad (3.4)$$

$V_{stations}$ = set of all stations from timetable

$E_{stations}$ = set of all physical links
connecting adjacent stations

TABLE 3.5: A modified version of train schedule for the train 11063 MS Salem Express.

Station Name	Arrival Time	Departure Time	Route No.	Day
Chennai Egmore	source	23:00	1	1
Attur	03:44	03:45	1	2
Salem Junction	05:20	destination	1	2
Salem Junction	source	07:05	2	2
Omalur	07:23	07:25	2	2
Metur Dam	08:25	destination	2	2
Salem Junction	source	06:30	3	2
Erode Junction	09:00	destination	3	2

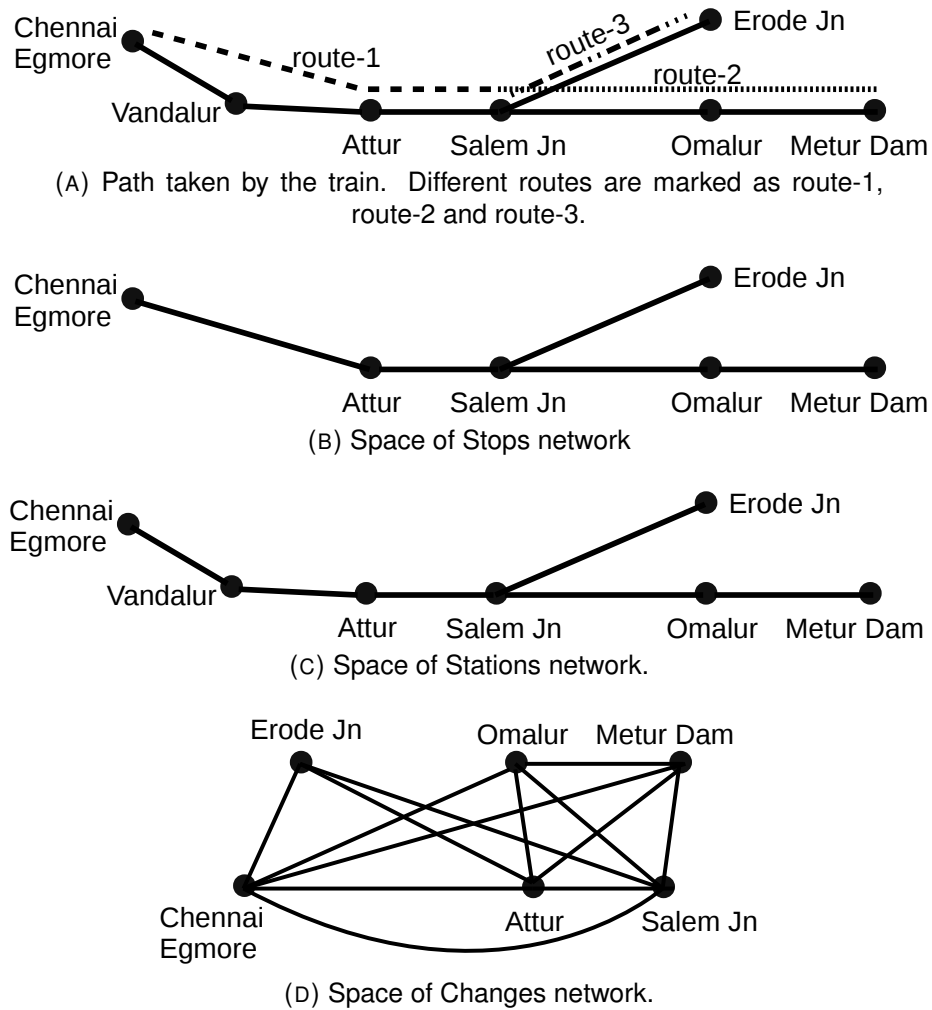


FIGURE 3.3: Creation of three layers from one example train time table. The timetable considered is a modified version of timetable for the train 11063 MS Salem Express. A tabular representation of the timetable is shown in Table 3.5.

space of changes graph:

$$G_{changes} = (V_{changes}, E_{changes}) \quad (3.5)$$

$V_{changes}$ = set of all stations from timetable

$E_{changes}$ = set of all links indicating availability of a train between two stations

T = IRN timetable

t = Timetable of a single train

SS = Stop sequence of a single train

J = Set of stops at which slip-train(s) separate
from the carrier train

P = Stop sequence prefix for junctions

The stop sequence of a train is a sequence of ordered tuples containing (stop, route) elements implied by a given train timetable. For the train timetable given in Table 3.5, the stop sequence (SS) is: (Chennai Egmore, 1), (Attur, 1), (Salem Junction, 1), (Salem Junction, 2), (Omalur, 2), (Mettur Dam, 2), (Salem Junction, 3), (Erode Junction, 3). From SS, we can identify junction stations (J) where slip-trains separate from the carrier train. For the given station sequence, we have Salem Junction as the junction. For each of the junctions, we can identify a stop sequence prefix (P) - sequence of previous stations appearing in the space of stops. In the case of Salem Junction, the sequence of previous stations is: (Chennai Egmore, 1) and (Attur, 1).

The three-layers of the generated network derived from the timetable given in Table 3.5 are shown in Figure 3.3. All the three layers use the same set of stations. Among the three layers, the space of stops has the least number of connections and the space of changes has the most number of connections.

Algorithm 3.1 Algorithm for creating the space of stops network from the IRN timetable having slip-trains.

```

1: procedure CREATESTOPLAYER( $T$ )
2:    $G_{stop} \leftarrow$  empty graph
3:   for all  $t \in T$  do
4:      $SS \leftarrow$  ExtractStopSeq( $t$ )
5:     for all  $s_i \in SS$  do
6:       if IsSameRoute( $s_i, s_{i+1}$ ) then
7:          $G_{stop}$ .AddEdge( $s_i$ .station,  $s_{i+1}$ .station)
8:       end if
9:     end for
10:  end for
11:  return  $G_{stop}$ 
12: end procedure

```

3.3.3 Algorithms for Creating Multilayer Networks

We propose a list-based technique for creating a three-layer network. Our approach takes advantage of the inherent total ordering of a train timetable. Our algorithm also incorporates the case of shared transit vehicles in the form of slip-trains.

Algorithm 3.2 Algorithm for creating the space of stations network layer from the space of stops network layer.

```

1: procedure CREATESTATIONLAYER( $E_{stop}, T$ )
2:    $E_{station} \leftarrow E_{stop}$ 
3:   for all  $e_i \in E_{station}$  do
4:     for all  $t \in T$  do
5:        $SS \leftarrow ExtractStopSeq(t)$ 
6:       if  $Distance(e_i, SS) > 1$  then
7:          $RemoveEdge(e_i, E_{station})$ 
8:         break
9:       end if
10:    end for
11:  end for
12:  return  $E_{station}$ 
13: end procedure

```

First, we discuss generation of the space of stops network from the timetable (Algorithm 3.1). We follow it up with further discussion on the space of stations network (Algorithm 3.2) and the space of changes networks (Algorithm 3.3).

In Algorithm 3.1, we use the IRN timetable (T) as input for constructing the G_{stops} . We initialize the G_{stops} to an empty graph and iterate through timetables of all the vehicles (trains) in the network. In each step of iteration, we extract the stop sequence (SS) from a given train timetable (t). The stop sequence is used to create stations and edges between adjacent stations that belong to the same route. At the end of its work, the algorithm creates a space of stops network.

It is common for express trains to pass through a station and not stop at it; such a scenario leads to addition of shortcut edges in the space of stops network; there is no underlying direct physical railway track that corresponds to a shortcut edge. The space of stations network is created by pruning all the shortcut edges from the space of stops network. Algorithm 3.2 receives E_{stops} and checks if an edge is a real link or a shortcut. We identify an edge as a shortcut if the two stops identified by e_i are non-adjacent in at least one of the train timetables.

Algorithm 3.3 Algorithm for creating the space of changes network from the IRN timetable having slip-trains.

```

1: procedure CREATECHANGESLAYER( $T$ )
2:    $G_{changes} \leftarrow$  empty graph
3:    $P \leftarrow$  empty list
4:   for all  $t \in T$  do
5:      $SS \leftarrow$  ExtractStopSeq( $t$ )
6:      $J \leftarrow$  IdentifyJunctions( $SS$ )
7:     for all  $j \in J$  do
8:        $P.AddPathPrefix(j, SS)$ 
9:     end for
10:     $SSArray \leftarrow$  ExpandSSofSlips( $P, SS$ )
11:    for all  $ss \in SSArray$  do
12:       $G_{changes}.Add(CreateCompleteGraph(ss))$ 
13:    end for
14:  end for
15:  return  $G_{changes}$ 
16: end procedure

```

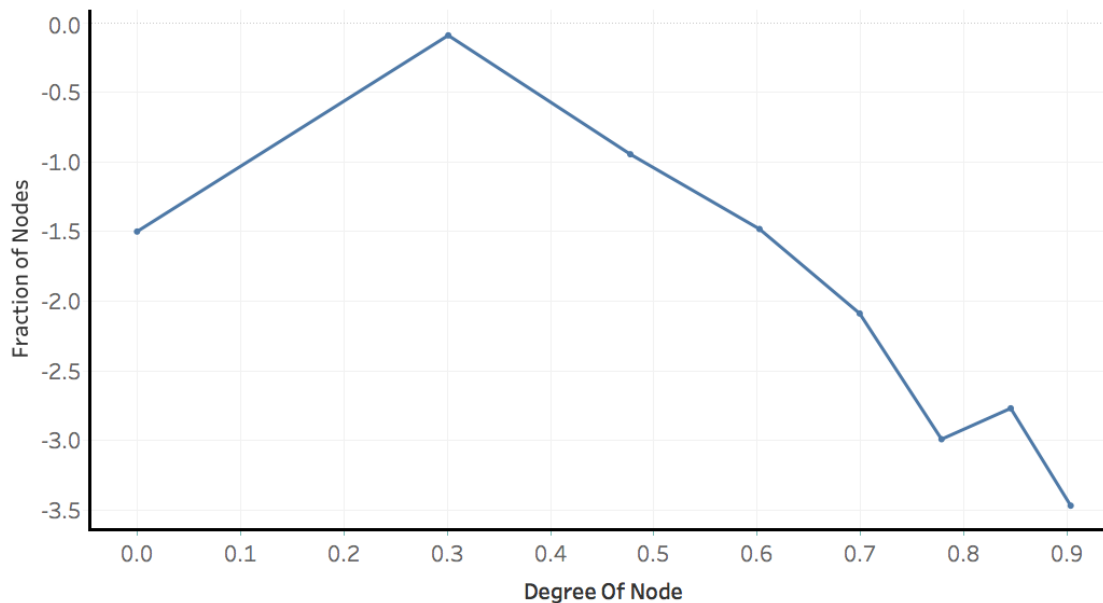
We can utilize the stop sequence to imply the non-adjacent status of two stops identified by e_i . A single scan through a stop sequence is enough to arrive at our conclusion. This list traversal is a linear time operation and is a scalable strategy for identifying shortcuts.

Algorithm 3.3 creates the space of changes network. In Algorithm 3.3, our goal is to connect all the stations that are serviced by a single train. In order to connect all the relevant stations, we elaborate on a given train timetable. Our analysis shows that a slip-train also travels with the carrier train till the junction where decoupling takes place. Hence, it is necessary to prefix junctions of slip-trains with their corresponding stop sequence (P). We effectively create one

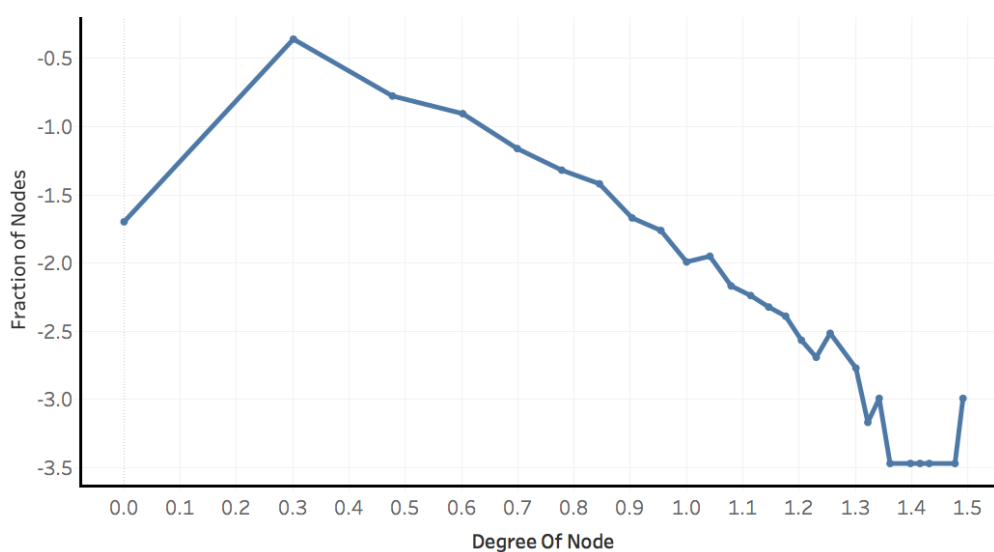
TABLE 3.6: Network characteristics of different layers created from the timetable of the Indian Railway network (IRN).

Name of Parameter	Space of Stations Network	Space of Stops Network
Number of nodes	2942	2942
Number of Edges	3217	5754
Number of Routes	2063	2063
Average Node Degree	2.18	3.91
Diameter of Network	238	43
Clustering Coefficient	0.003	0.449
Average Shortest path	70.78	10.11

stop sequence per route. All the generated stop sequences of a train timetable are stored in $SSArray$. We create one complete graph for each stop sequence and add the generated graph to $G_{changes}$.

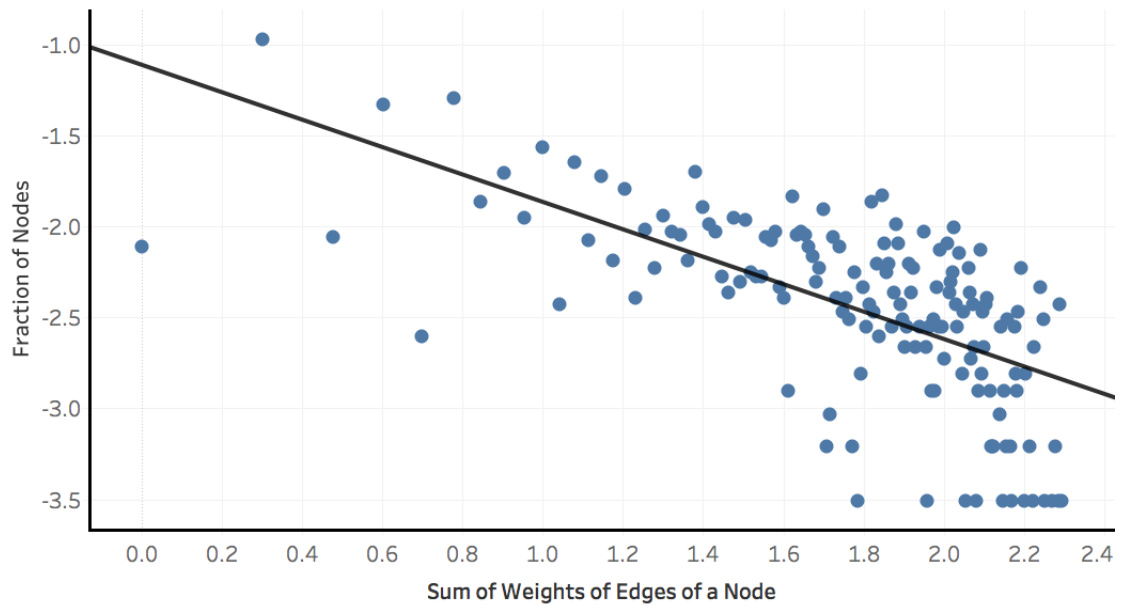


(A) Space of stations network

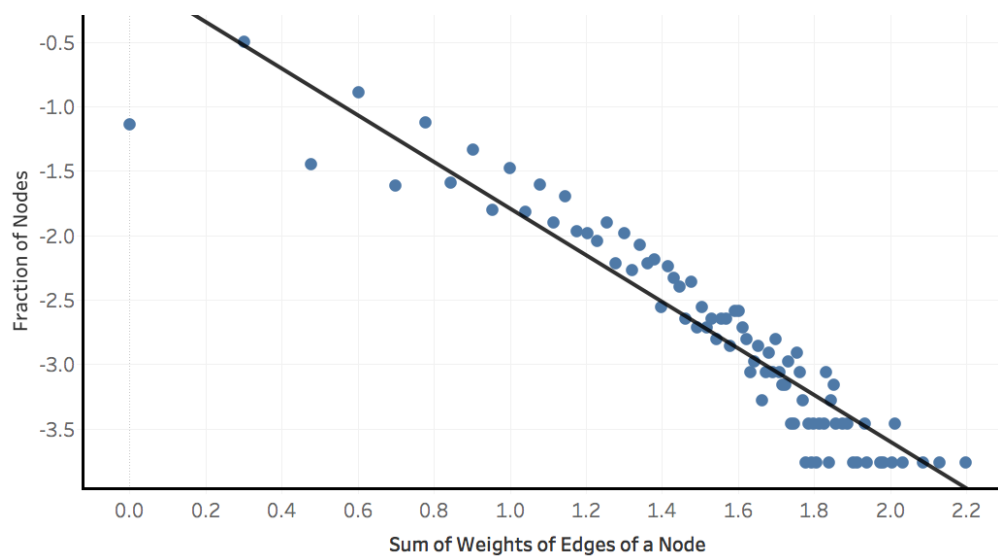


(B) Space of stops network

FIGURE 3.4: Node degree distributions of multi-layer networks of Indian Railways. The x-axis indicates the degree of a node and the y-axis indicates the fraction of nodes with a given degree. The plots are done on the log-log scale.



(A) Space of stations network



(B) Space of stops network

FIGURE 3.5: Edge weight distributions of multi-layer networks of Indian Railways. The x-axis indicates the sum of weights of all edges of a node and the y-axis indicates the fraction of nodes with a given degree. The plots are done on the log-log scale.

3.3.4 Network Analysis

We perform network analysis on the three-layer network generated using the Algorithms discussed in the previous section. The scale of networks in different layers can be seen in Table 3.6. As we move from the space of stations network to space of stops network, we can see the densification of the network in terms of the number of edges and the average node degree. We can also see significant reduction in the diameter - maximum shortest path length - of the network for the space of stops network vis-à-vis the space of stations network. The extent of densification can also be seen by increase in the average clustering coefficient and decrease in the average shortest path.

The node degree distributions of both space of stations and space of stops (shown in Figure 3.4) follow power-law distributions. Prior research on this topic by Sen et al. [90] and Srivastava et al. [91] attest to the power-law network nature of the IRN.

We also consider the load on the network by analyzing the weighted version of the space of stops network. Here, weight of an edge corresponds to the number of trains utilizing that edge. Edge weights can be used to estimate the load on the network. As can be seen from Figure 3.5, the edge weight distributions of the space of stations and the space of stops networks follow power-law distributions.

3.4 Structural Analysis on Social Networks

In this section, we explain the results of our study on multilayer social networks. We consider the Internet relay chat (IRC) communities as example social networks for this study.

3.4.1 Dataset

We have analyzed the IRC log datasets of Slackware operating system (OS) IRC community [92] and Ubuntu IRC ecosystem [93]. Slackware is a classic

TABLE 3.7: Description of the datasets under consideration.

Community	Log availability		Statistics for the chosen time period		
	starting month	ending month	chosen month(s)	users	directed messages
KUbuntu-devel	Apr-2005	May-2017	Jan, 2013	89	3130
Slackware OS	Jan-2005	Dec-2014	Jan, 2013	337	6999

Linux OS distribution. Ubuntu IRC ecosystem is the collection of all IRC communities dedicated to the development of Ubuntu OS and its' derivatives. Ubuntu IRC ecosystem is a flourishing set of IRC communities. Ubuntu IRC ecosystem started off as one community in 2003 and proliferated to 169 communities by January, 2017. We study thirty-six most active communities of Ubuntu IRC ecosystem based on the total number of messages exchanged on the community in January, 2013. We also consider Slackware OS community; this is a single online chat community catering to the online communication needs of Slackware OS developers. A summary of the datasets under consideration is provided in Table 3.7. The users column specifies the number of active users active in a community; directed interactions specifies the number of conversations observed on the community.

3.4.2 Kinds of Networks

Online communities (chat channels) can be modeled as social networks based on two parameters: first is the *co-presence* and second is the *message exchange*. The presence parameter helps us track the presence of a user on a chat channel and the co-presence of a pair of users on a chat channel. The message exchange parameter helps us track the frequency of message exchanges between two users of a chat channel.

A *presence network* can be obtained by tracking the activity of a user on various communities. We constructed the presence social network for Ubuntu IRC communities using the following network construction conventions. All the online communities and users become vertices of the network. The edges themselves are of three kinds:

1. *Community - User (CU) presence network*

The edge between an online community and a user indicates the presence of a user on a community. The edge weight is equal to the number of days a user is active on the community. We represent all the community to user edges in A_{cu} adjacency matrix.

2. *Community - Community (CC) co-presence network*

The edge between any two communities indicates the presence of common users among the communities. The edge weight is equal to the number of common users between the two communities as observed over the selected time period. We represent all the community to community edges in A_{cc} adjacency matrix.

3. *User - User (UU) co-presence network*

The edge between any two users indicates the co-presence of two users on one or more communities. The edge weight is equal to the number of days on which these two users have appeared on a community at the same time. If a pair of users appear on k communities on a day, then the corresponding edge weight gets incremented by k . We represent all the user to user edges in A_{uu} adjacency matrix.

The *message exchange* social network can be constructed based on the direct chat messages exchanged between the users. In this network, users are modeled as nodes; each message adds or increments the weight of the directed edge from the sender to the directed receiver. As a result, we get a directed and weighted social network. The co-presence and message exchange networks are intimately related. The co-presence network is the background network on which the message exchange network manifests.

3.4.3 Degree Distribution

To understand the network characteristics of the online communities, it is imperative to study the properties of their social networks. The node in-degree and out-degree distributions are the starting points in this kind of social network analysis.

TABLE 3.8: Justification for the numerical heuristics used in the analysis framework.

Heuristic	Value used	Justification
Top users	100	These cutoffs were used only for visualization purpose in the chapter. They are not needed for analysis at all. The analysis methods of network profile are capable of working on large graphs with no upper limits on the maximum graph size.
Top channels	30	These cutoffs were used only for visualization purpose in the thesis. They are not needed for analysis at all. The analysis methods of network profile are capable of working on large graphs with no upper limits on the maximum graph size.
Edge weight cutoffs	0 / 10 / 20	These cutoffs were used only for visualization purpose in the thesis. They are not needed for analysis at all.

On community-community (CC) and community-user(CU) presence social networks, two communities with higher edge weight indicates higher potential overlap in user interactions. On user-user (UU) presence social networks, the node degree indicates the potential for interaction between users. On message exchange social networks, the node degree distribution is our primary estimate for determining the expertise/interest and the level of activity of a user on an online community.

3.4.4 User Communities

A social network community is defined as a subset of nodes within the network such that connections between the nodes are denser than connections with the rest of the network [94]. We detect social network sub-communities on the presence and message exchange social networks of an online community using Infomaps community detection algorithm [59]. We use the heuristic cutoffs listed in Table 3.8 for creating the visualizations of communities. These heuristics are only used for visualization purposes, but do not affect the structural analysis of the networks.

TABLE 3.9: Network statistics for presence and message exchange networks of the IRC communities. The nomenclature is as follows: CC corresponds to Community-Community network; CU corresponds to Community-User network; UU corresponds to User-User network; a prefix of r implies the corresponding network has been reduced to top-30 channels and / or top-100 users.

	Presence networks (Ubuntu IRC ecosystem)				Message exchange networks (Select channel(s))					
	CC	CU	UU	UU	rCC	rCU	rUU	Channel-I ^a	Channel-II ^b	Channels-III ^c
Nodes	158	15,667	15,391	15,391	30	130	100	89	337	602
Connections	3,065	20,535	3.3 million	3.3 million	435	509	1,727	392	1,882	2,260

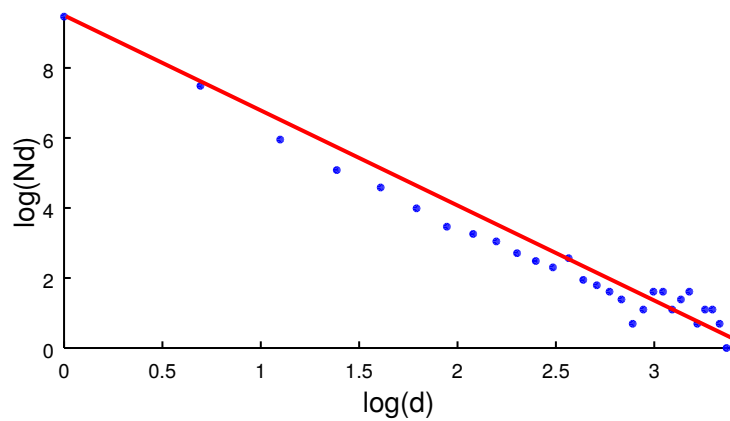
^a Channel-I: KUbuntu-devel, ^b Channel-II: Slackware

^c Channels-III: KUbuntu-devel + Ubuntu-devel + KUbuntu

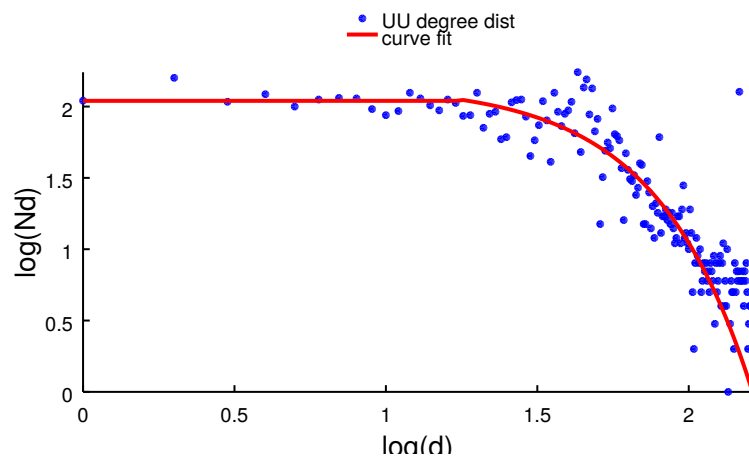
TABLE 3.10: Curve fit parameters for degree distribution plots of social networks generated from Ubuntu IRC communities. The presence networks are undirected networks, hence only the degree distribution is shown. The message exchange networks are directed networks, hence both the in-degree and out-degree distributions are shown. The given parameters are for the line $\log N_d = -\gamma \log d + K$.

	Undirected presence networks (Total degree distribution)		Directed message exchange networks				
	CC	CU	UU	(Out-degree distribution)	(In-degree distribution)	(In-degree distribution)	
	CC	CU	UU	Channel-I ^a	Channel-II ^b	Channel-II ^b	
γ	-0.33	-2.04	-0.38	-0.63	-1.19	-0.69	-1.35
K	1.60	7.3	3.22	2.18	4.32	2.4	4.82
R^2	0.36	0.97	0.39	0.66	0.95	0.67	0.90
MSE	0.17	0.09	0.22	0.16	0.04	0.18	0.09

^a Channel-I: KUbuntu-devel, ^b Channel-II: Slackware



(A) node degree distribution of CU network

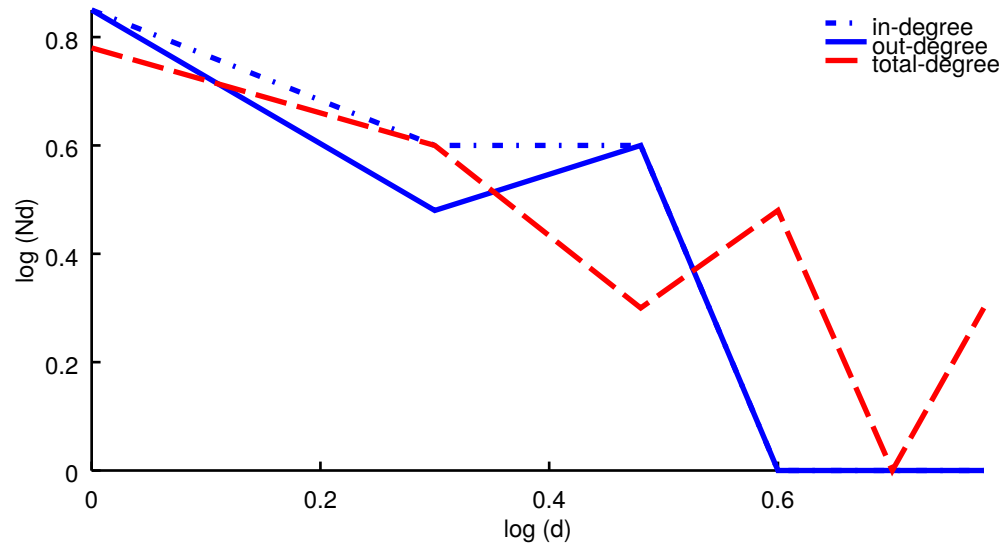


(B) node degree distribution of UU network

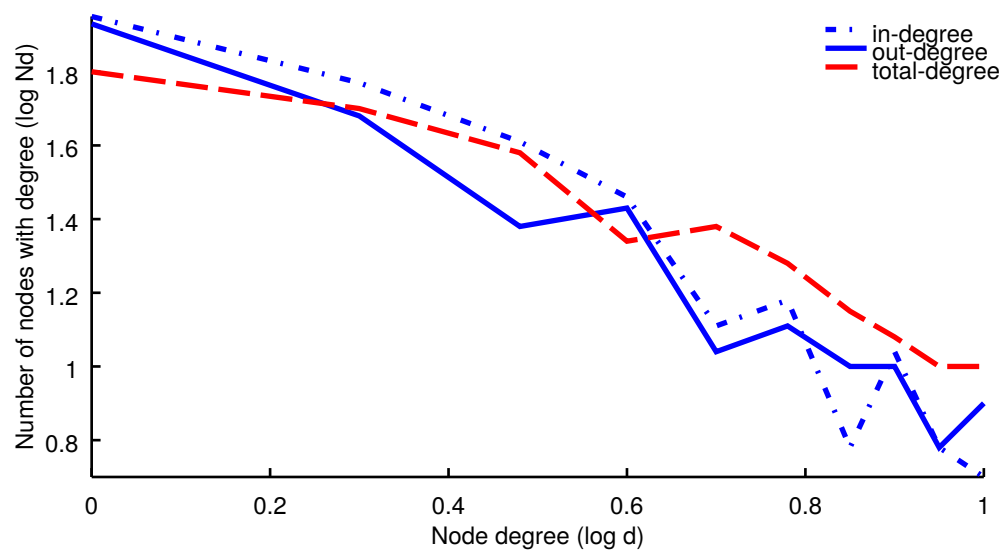
FIGURE 3.6: Node degree distributions for the CU and UU presence social networks constructed on the IRC chat channels under study. The data is for January, 2013 for Ubuntu IRC communities.

3.4.5 Experimental Results

We construct the adjacency matrices of the two social networks – presence and co-presence network and message exchange social network for Ubuntu communities. We construct message exchange social network for all the three communities. Since presence and co-presence network requires two or more overlapping communities, it is not possible to construct these networks for Slackware community. Table 3.9 shows the network statistics of all the communities analyzed in the thesis.



(A) node degree distribution of message exchange network of KUbuntu-level community.



(B) node degree distribution of message exchange network of Slackware community.

FIGURE 3.7: Node degree distributions for the message exchange social network constructed on the IRC chat channels under study. The data is for January, 2013 for Ubuntu IRC communities and Slackware.

Degree Distribution

We derive the degree distribution analysis of the message exchange social network for all the three communities. The node degrees, both in-degree and out-degree, are highly concentrated. In the case of KUbuntu-devel, we observe that the node degrees range from 1 to 9. The node degree distributions graphs are shown in Figures 3.6 and 3.7. The degree distributions of CU presence, UU presence and message exchange social networks closely fit exponential equation.

We perform curve fit on degree distributions using exponential equations given in Equation 3.6.

$$N_d = e^K d^{-\gamma} \quad (3.6)$$

$$\log N_d = -\gamma \log d + K \quad (3.7)$$

where,

- d = node degree
- N_d = number of nodes with node degree d
- γ = exponent for node degree d
- K = exponent for coefficient (scaling factor)

The exponential curve fit parameters are shown in Table 3.10.

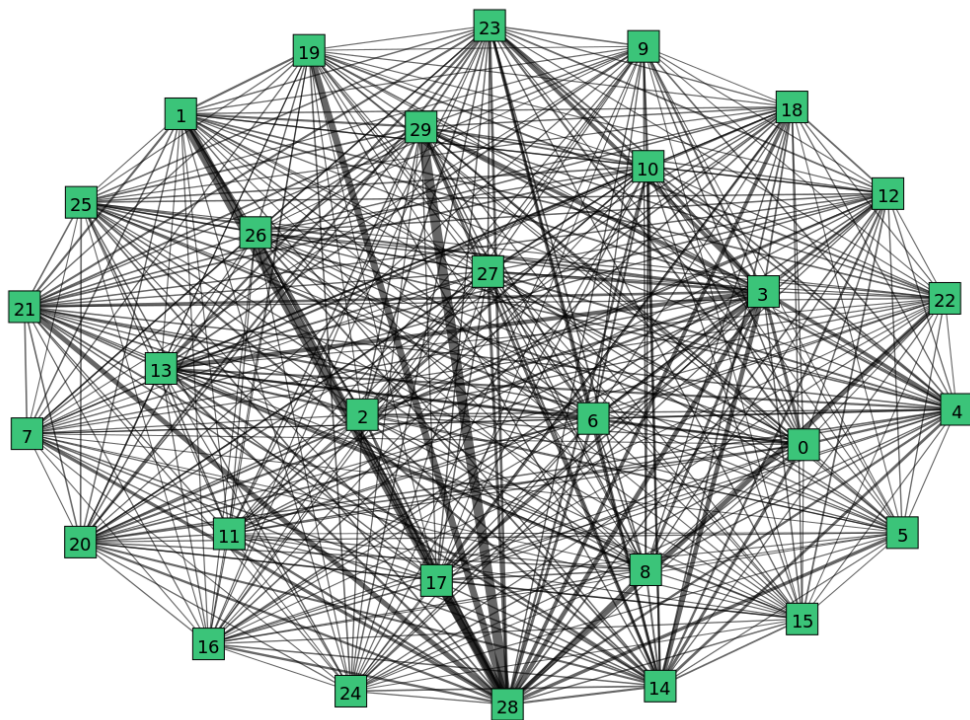
User Communities

We perform community detection on the presence and message exchange social network of the selected IRC communities. We use Infomaps [59] algorithm for community detection.

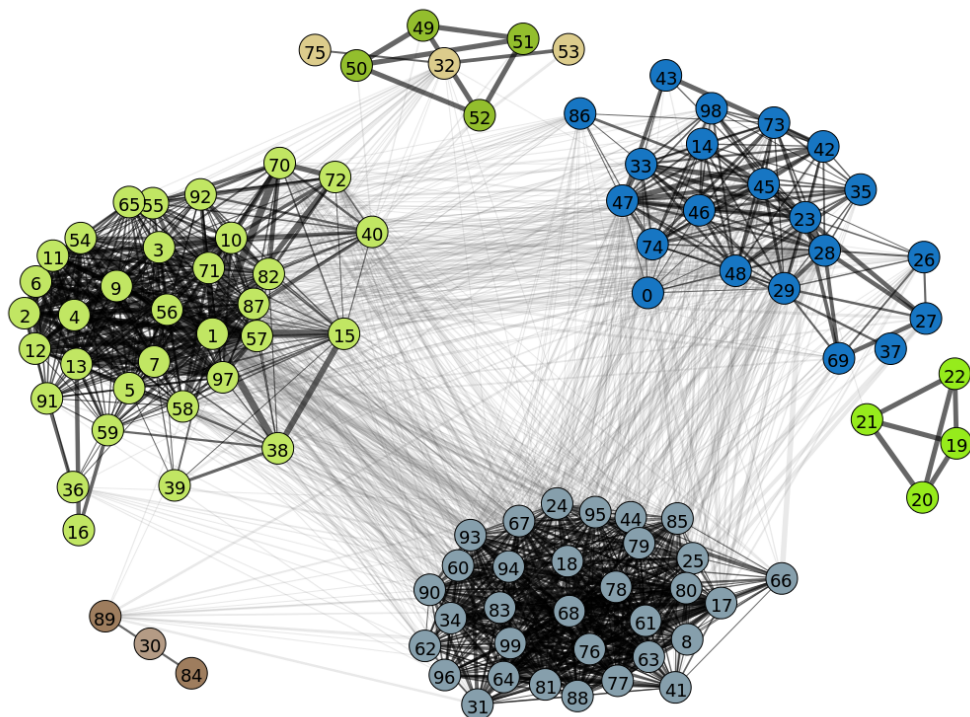
We provide a summary of the presence and co-presence social networks of Ubuntu communities. We prune social networks to the top-30 communities and

TABLE 3.11: Code length for the presence social networks of the Ubuntu IRC communities. The code length has been calculated using InfoMaps community detection algorithm. The results are for the month of Jan, 2013.

CC Network	CU Network	UU Network	Description
6.36	8.15	12.18	complete network
4.74	6.13	6.05	reduced network (top-30 channels and top-100 users)



(A) Communities among top-30 Ubuntu IRC communities.



(B) Communities among top-100 Ubuntu users.

FIGURE 3.8: Communities detected on the presence and co-presence social networks of Ubuntu IRC ecosystem for January, 2013. The number of communities has been reduced from 160 to top-30 communities by activity and the number of users have been reduced from more than 15,000 to top-100. These two reductions have been done to better illustrate the community networks.

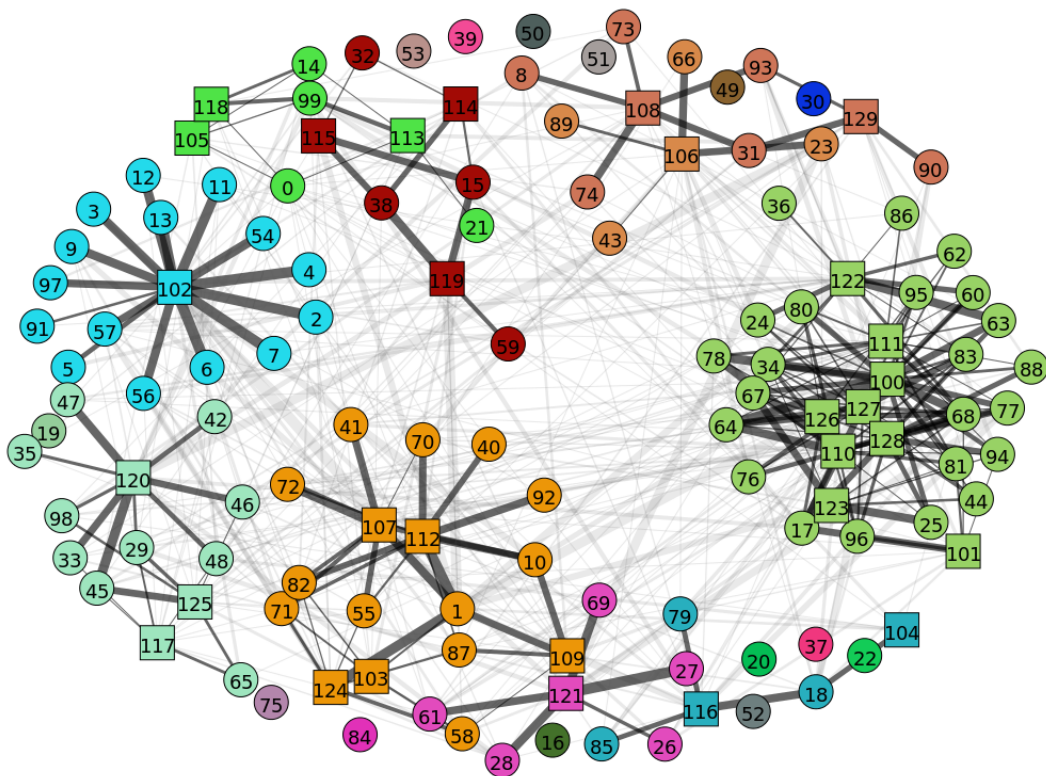


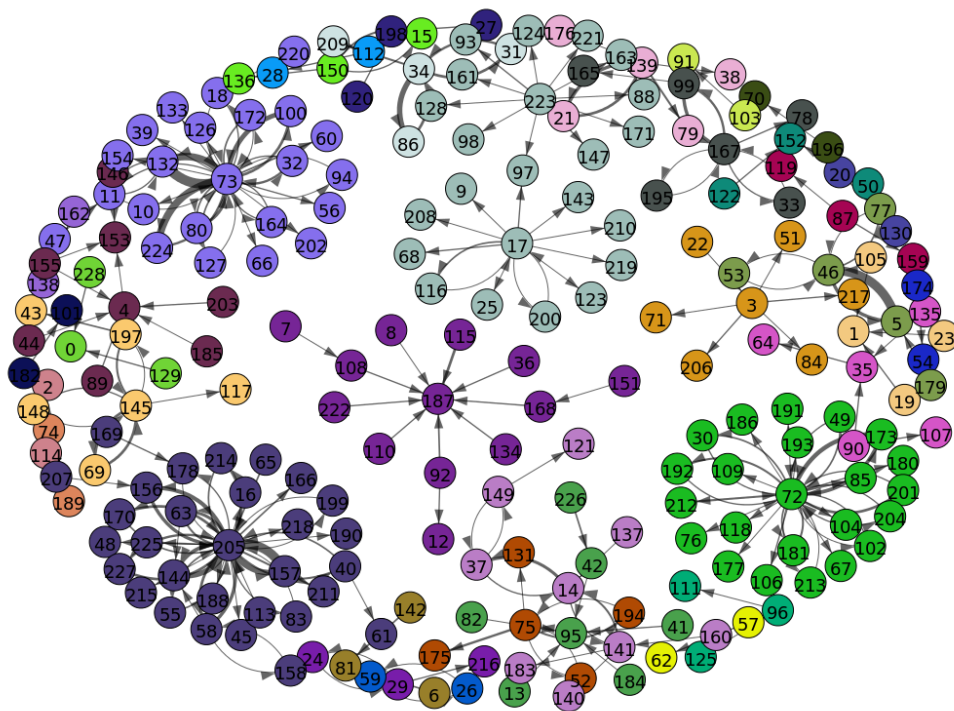
FIGURE 3.9: Communities among top-30 Ubuntu Channels and top-100 users. User nodes are shown in circular shape whereas channel nodes are illustrated in rectangular shape.

the top-100 users from all the communities. The sub-communities detected on the presence social networks of Ubuntu IRC communities are illustrated in Figures 3.8 and 3.9. The code lengths¹ generated by the Infomap community detection algorithm for the three presence social networks are shown in Table 3.11. We find that the average clustering coefficients remain stable within each social network.

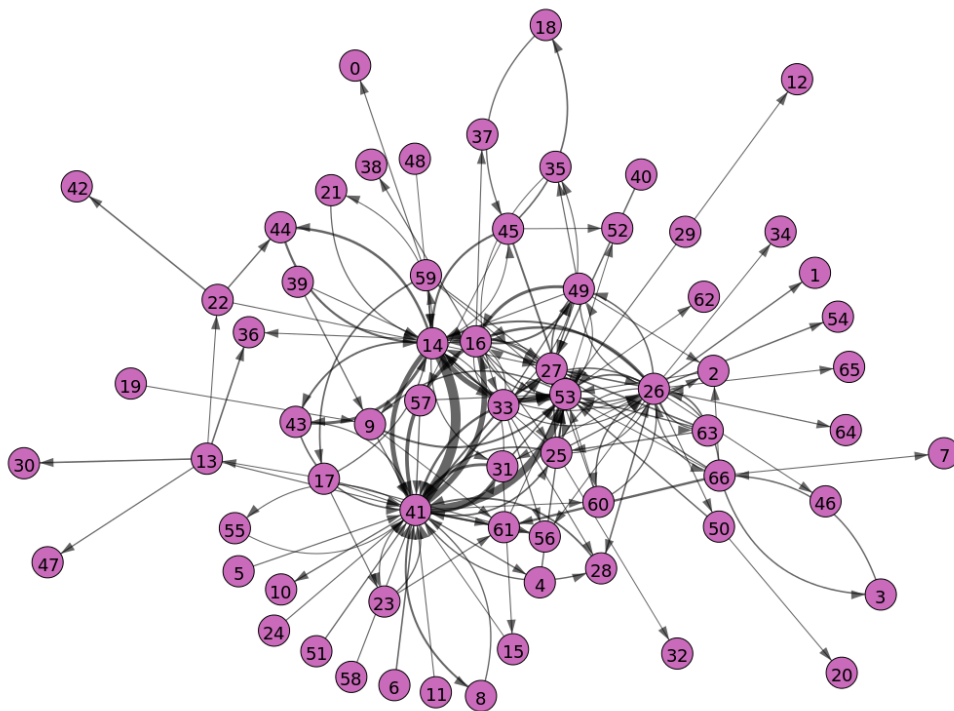
¹Size of variable-length code required to enumerate the communities in a given network; typically specified in fraction of bits.

TABLE 3.12: Code length for the message exchange social networks of the two communities. The code length has been calculated using InfoMaps community detection algorithm.

Month	KUbuntu-level	Slackware
Jan, 2013	4.79	6.93
Jun, 2013	4.97	7.37

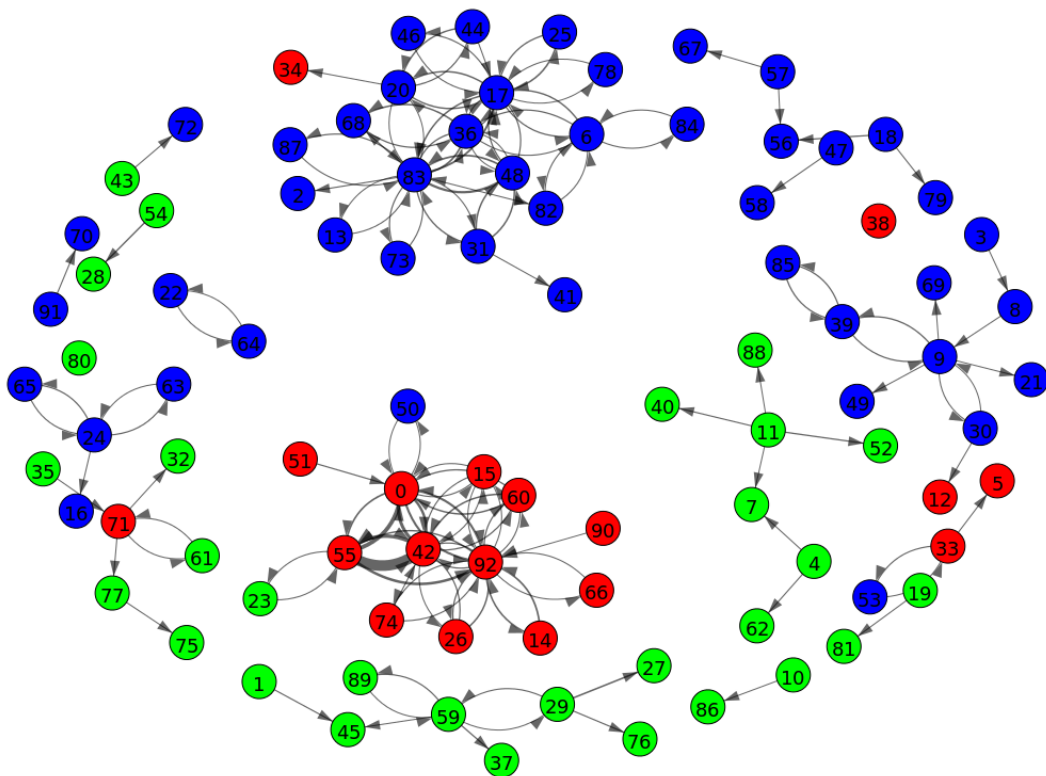


(A) KUbuntu - Jan 2015, cutoff=0



(B) Slackware - Jan 2013, cutoff=10

FIGURE 3.10: Communities in the message exchange social network of chat channels. Each community is given a different color. Cutoff numbers refers to the minimum edge weight threshold considered for community detection purposes. The cutoff numbers were chosen purely for visualization purpose and do not limit the generality of the conclusions.



Color Legend: Red - KUbuntu-devel; Green - KUbuntu; Blue - Ubuntu-devel

FIGURE 3.11: Communities detected in multi-channel analysis on Ubuntu-devel, KUbuntu and KUbuntu-devel communities for the year 2013. The message exchange graph generated from the three mentioned communities is subjected to an edge weight cutoff of 10. Each node in the given graphs corresponds to one user and the color of a node corresponds to one chat channel. A user is mapped to one chat channel in which (s)he has most presence in the selected time window.

The results of a similar community detection analysis on the message exchange social network and the detected communities are shown in Figure 3.10. The thickness of an edge indicates the intensity of interaction between two users. All the nodes of one community are given one color. The code lengths generated by Infomap community detection algorithm for the KUbuntu-devel and Slackware channels are shown in Table 3.12. The average clustering coefficients remain stable within each channel.

We also perform community detection on multiple IRC chat channels. We construct multi-channel message exchange social networks by combining the message history logs of KUbuntu-devel, KUbuntu and Ubuntu-devel communities for the month of January, 2013. On this multi-channel social network, we perform community detection. The resulting communities are illustrated in Figure 3.11.

The message exchange graph generated from the three mentioned communities is subjected to edge filtering , i.e., removal of all edges with edge weight below 10. The edge filtering does not effect the nature of communities detected, but is only used for visual clarity. All the sub-communities predominantly belong to one chat channel. Thus, the images clearly illustrate a clean isolation of social interactions among members of different chat channels.

3.5 Summary

In this chapter, we provide details of our analysis on the structure of multilayer networks in the computer networks, transit networks and social networks areas. In the computer networks area, we choose NDT distributed measurement dataset to perform statistical analysis on the Internet connections of the Indian broadband users. In the transit networks area, we represent the timetables of Indian Railways as a three-layer network. In the social networks area, we use Ubuntu and Slackware IRC channels to perform structural analysis on multilayer social networks.

With the case studies taken up in this chapter, the thesis demonstrates the utility of structural analysis on multidimensional networks found in transport networks and social networks. The networks analyzed in this chapter fit power-law network models. We are able to detect communities in the bipartite CU co-presence network. We are also able to detect communities in the multidimensional networks created from multiple IRC channels.

Chapter 4

The Cell Model

4.1 Motivation

Multidimensional network models have been used for the study of many real-world networks like multi-modal transit networks and for performing network measurements in computer networks. In both the application domains, the participating entities are rarely passive. For example, a transit station of a multi-modal transit network encounters dynamic conditions like rush hour traffic, arrival and departure of vehicles etc. In computer networks, network packets are received, processed, forwarded or possibly discarded by the participating computing nodes. Network measurement engineers actively measure the connection characteristics in order to create realistic models of network connections.

Another interesting feature of transit and computer networks is the causality of events in the network. Events at one node cause a cascade of related events at neighbouring nodes. For example, if an Internet router malfunctions and reports wrong routes¹, then such a corrupt router can effect the regular operations of routers in its' neighbourhood.

The existing structural network models such as random graphs, scale free networks and disease propagation models emphasize on the structural properties

¹A route is a calculated path from source computer to destination computer via intermediate routers

of complex networks. The often studied centrality measures [95] and the network community detection algorithms [67] also emphasize on the structural connections of a network. Some network models such as those proposed for multidimensional networks and disease propagation models use labels for nodes and connections. In the prior literature, the assumptions made in the modeling of complex networks are:

- Nodes of a network are passive entities with no behavioural or functional manifestations.
- Connections are binary relations whose strength may be indicated by a weight or a function.
- All networks are assumed to be flat structures with no hierarchy. Hierarchical representation of network has not been adequately considered.

We propose a network model that considers structural, functional and behavioural aspects of network participants - both nodes and connections. Our network model helps in the study of network phenomenon where participants exchange messages via dynamic connections. We denote nodes and connections of a network as cells and the network model built using this paradigm as the Cell Model.

4.2 Notation

A Cell Model of a network can be extended from the regular network (graph) model. The regular network model can be defined as:

$$\begin{aligned}
 \text{Regular Network: } G &= (V, E) && (4.1) \\
 \text{Nodes: } V &= \{v_1, v_2, \dots, v_m\} \\
 \text{Connections: } E &= \{e_1, e_2, \dots, e_n\} \\
 &= \{e_i : e_i = (v_j, v_k) \in E \text{ for } i = 1 \text{ to } n \text{ and} \\
 &\quad v_j, v_k \in V \text{ for } j, k = 1 \text{ to } m\}
 \end{aligned}$$

An equivalent notation for G (see Equation 4.1) using Cell Model can be constructed as follows.

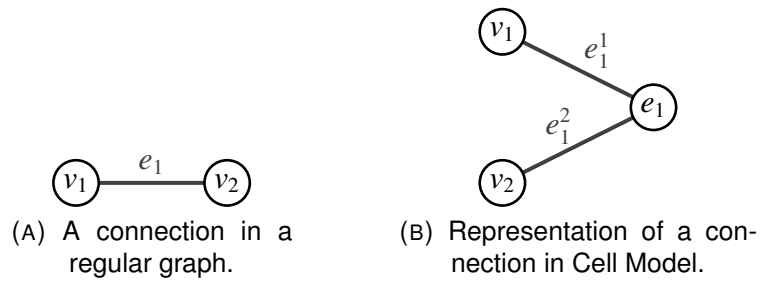


FIGURE 4.1: Representation of a connection in a regular graph and in the Cell Model.

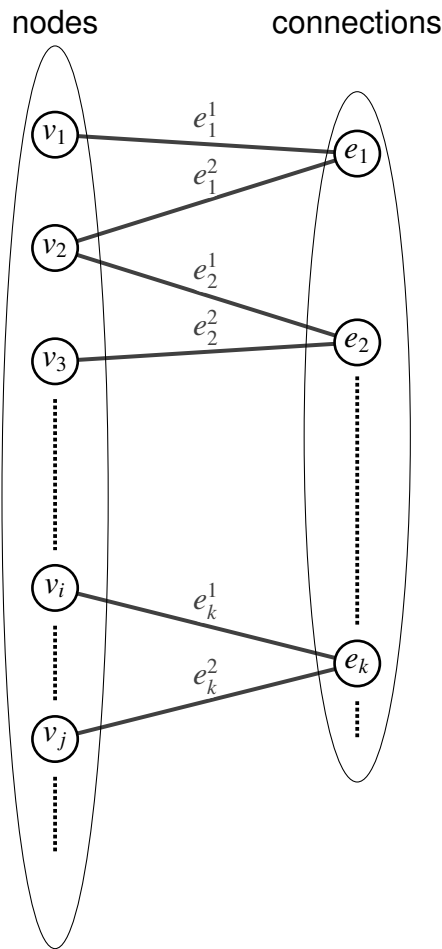


FIGURE 4.2: A graphical representation of a network using the Cell Model.

Network using the Cell Model: $G_c = (V_c, E_c)$ (4.2)

Nodes: $V_c = V \cup E$

Relations (Edges): $E_c = \{e_1^1, e_1^2, e_2^1, e_2^2, \dots, e_n^1, e_n^2\}$
 $= \{e_i^1, e_i^2 : e_i = (v_j, v_k) \in E \text{ and } e_i^1 = (v_j, e_i) \text{ and } e_i^2 = (e_i, v_k) \text{ for } i = 1 \text{ to } n\}$

In the Cell Model, each connection of G (i.e., $e \in E$) is given a node-like prominence in G_c . Each connection $e_i \in E$ becomes a node in V_c , i.e. $E \subset V_c$. The connectivity between vertices indicated by e_i gets transformed into two equivalent edges $e_i^1, e_i^2 \in E_c$. The edges of G_c are unweighted, undirected and unlabeled. The equivalent graphical representation is shown in Figure 4.1. Hence forth, any reference to the word *edge* shall be in the context of the Cell Model; thus the word *edge* infers a binary relation which is undirected, unweighted and unlabeled.

During the creation of a Cell Model, we consider the principal participating entities as well as their relationships as nodes. The edges of Cell Model are placeholders to help preserve the information about the relationships and the participating nodes of each of the relationships.

A graphical representation of the Cell Model of a network is shown in Figure 4.2. The network thus constructed is a bipartite network.

4.3 Views on Network

We can use a combination of structural, functional and behavioural views to provide computable models for all the participants of the network.

4.3.1 Structural View

A structural view represents the topological connections between participants of a network. The Cell Model has the following effect on the structural properties of a network.

- The total degree of a node remains the same. Hence, the total degree distribution of all the nodes belonging to $V \subset V_c$ remains the same.
- The size of a network in Cell Model is equal to $|V|+|E|$ of the corresponding network modeled as a graph.
- All the connections of a network in Cell Model have a degree of two.

In Cell Model, the connections are simple, static edges. All the complexities of relations like weight, functions and labels are hidden in nodes belonging to

E_c . Thus the structural analysis such as determination of centrality measures, communities, path length on a Cell Model of a network is equivalent to structural analysis on a bipartite network.

4.3.2 Functional View

As per Crilly et al. [96], functions have two broad meanings in engineering. One is the role of transforming an input to an output; this definition of function is closer to its mathematical interpretation. The second meaning of function is the utility derived from a system (or the goal / purpose of system as seen) by its users. Thus a functional view can represent the external effect of a network participant as seen by an external observer. A functional model of an active participant is as follows.

$$\begin{aligned}
 \text{Input messages : } X &= \{x_1, x_2, x_3, \dots, x_m\} \\
 \text{Environment variables: } A_1 &= \{a_1^1, a_1^2, \dots, a_1^{k_1}\} \\
 &A_2 = \{a_2^1, a_2^2, \dots, a_2^{k_2}\} \\
 &\dots \\
 &A_n = \{a_n^1, a_n^2, \dots, a_n^{k_n}\} \\
 \text{Function Input: } \Sigma &= X \times A_1 \times A_2 \times \dots \times A_n \\
 \text{Output messages: } Y &= \{y_1, y_2, y_3, \dots, y_p\} \\
 \text{Function: } f : \Sigma &\longrightarrow Y \cup \{null\} \tag{4.3}
 \end{aligned}$$

Here X and Y represent the set of feasible input and output messages respectively. The environment variables are modelled as independent variables each

TABLE 4.1: The four operations performed by the network participants in terms of properties on a function used under function view.

Operation	Properties of Function
Generate	$f(null, a_1, a_2, \dots, a_n) \neq null$
Terminate	$f(x, a_1, a_2, \dots, a_n) = null$
Transform	$f(x, a_1, a_2, \dots, a_n) \neq null$
Forward	$f(x, a_1, a_2, \dots, a_n) = x$

of which take values from $a_i \in A_i$. The set of input messages and environment variables together form the domain of the function while the set of output messages form co-domain of the function.

The functional view of a participant provided in Equation 4.3 does not have memory, i.e. it is stateless. When considering the state, we utilize the behavioural model of a participant. Table 4.1 depicts the functional model to represent the four operations on messages that can be performed by the network participants. A participant can use only one of the four operations mentioned in Table 4.1.

In *generate* operation, an output message is generated even without any input message; the no input message scenario is indicated using *null* message. In *terminate* operation, a participant consumes an incoming message, but does not generate any new message in response to the consumed message. This no output message status is shown as *null* message. In *transform* operation, a matching output message is generated in response to an input message. In *forward* operation, the input message is sent out without any modifications.

4.3.3 Behavioural View

A behavioural view represents internal changes and external responses to stimuli. We use the state machines to provide a behavioural view and restrict the type of state machines to finite state machines (FSM). The notation used for representing the behavioural view of an entity is shown in Equation 4.4.

$$\begin{aligned}
 \text{Finite State Machine } FSM &= (S, \Sigma, Y, f, g, s^{in}) & (4.4) \\
 \text{Set of States: } S &= \{s_1, s_2, s_3, \dots, s_o\} \\
 \text{Input messages: } X &= \{x_1, x_2, x_3, \dots, x_m\} \\
 \text{Environment variables: } A_1 &= \{a_1^1, a_1^2, \dots, a_2^{k_1}\} \\
 &A_2 = \{a_2^1, a_2^2, \dots, a_2^{k_2}\} \\
 &\dots \\
 &A_n = \{a_n^1, a_n^2, \dots, a_n^{k_n}\} \\
 \text{Function Input: } \Sigma &= X \times A_1 \times A_2 \times \dots \times A_n \\
 \text{Output messages: } Y &= \{y_1, y_2, y_3, \dots, y_p\}
 \end{aligned}$$

$$\begin{aligned} \text{Update functions:} \quad & f : S \times \Sigma \longrightarrow S \\ & g : S \times \Sigma \longrightarrow Y \cup \{null\} \\ \text{Initial State:} \quad & s^{in} \end{aligned}$$

The meaning of X, A_i and Y variables remain same as in Equation 4.3. The FSM is represented using a tuple notation with S indicating the set of states and s^{in} being the initial state of the FSM. The update functions f and g indicate the functions for state update and output generation respectively.

4.4 Axioms of the Cell Model

The networks and participants of networks constructed in the Cell Model adhere to four axioms, namely *message exchange*, *activeness*, *equivalence* and *hierarchy*. The rest of this section contains axioms and their interpretation.

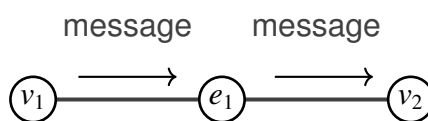


FIGURE 4.3: A typical scenario of message exchange. A message is sent from node v_1 to its connection e_1 . In response, e_1 may send a message to node v_2 .

4.4.1 Messages Between Participants

The participants in the Cell Model of a network, i.e., both nodes and connections, interact with each other by exchanging messages. Thus the axiom of message exchange lays the ground rule for interaction in the network.

Axiom 4.1 (Message Exchange). *All network interactions happen via message exchanges.*

The message transfer between two nodes facilitated by a connection is illustrated in Figure 4.3. In the Cell Model of a network, a connection (for example, e_1 in Figure 4.3) is drawn from one participant of a network to another if they exchange messages. A message is sent from a node to another node via an edge.

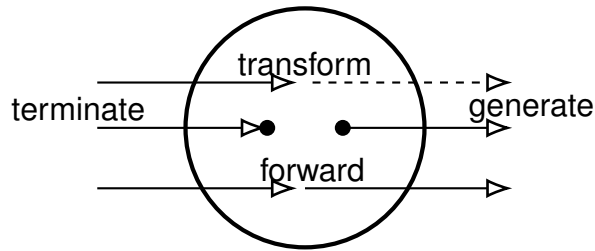


FIGURE 4.4: Operations on messages performed by a participant of a network. Each keyword denotes the kind of operation performed on a message. The participants are nodes and connections.

4.4.2 Active Participants

Both the nodes and the connections of the Cell Model of a network are non-passive entities. The axiom of activeness specifies the kind of operations that these entities can perform.

Axiom 4.2 (Activeness). *All the participants (nodes and connections) are active:*

1. Nodes can generate, terminate, transform or forward messages.
2. Connections can terminate, transform or forward messages.

TABLE 4.2: Operations on messages performed by the participants of network in the Cell Model.

Operation	Illustration	Explanation
Generate		Generate a message and send to neighbours via a connection. Only a node can perform this operation.
Terminate		Terminate a message received. Both node and connection can perform this operation.
Transform		Alter the message and then forward the same. Both node and connection can perform this operation.
Forward		Forward the message with no changes. Both node and connection can perform this operation.

The permitted operations performed on messages by the participants of a network are: generate, terminate, transform or forward. The permitted operations are illustrated in Figure 4.4 and are explained further in Table 4.2. Nodes can perform any of the four operations; Connections cannot perform the generate message operation, but can perform terminate, transform or forward operations. The restriction of generate operation to nodes has been put in place to limit causality of events in the network to only nodes. Only nodes can generate new messages which can trigger further events in the network. Connections only act on the messages sent to them.

There is an influence of environment on all the participants of the network. The environmental influence is modeled using the independent variables. The computational models of network participants are dependent on the environment variables.

There is quite a close relation between the transform and forward operations of a participant. A forward operation is a projection. In general, all the participants have memory (state). A participant can switch between the allowed operations based on the incoming message and state. The next state of a participant is determined by the current state and the incoming message.

4.4.3 Equivalence

We strive for a simplest possible representation of an entity. Thus we often need to replace a very detailed FSM with a more simple, abstract model for FSM. The axiom of equivalence caters to this need for simplicity in representation.

Axiom 4.3 (Equivalence). *Two behavioural views of a participant are equivalent if both have equivalent Finite State Machines (FSM), i.e., both the FSM's are observational equivalent.*

In order to understand the requirements of observational equivalence, we need to understand the bisimulation equivalence.

As per Hirshfeld et al. [97], two processes (for us, FSMs) are bisimilar, or bisimulation equivalent, if, roughly, they may evolve together in such a way that whenever the first process performs a certain action, the second process is able to respond by performing a matching action, and vice versa.

We use the FSM notation defined in Equation 4.4 to mathematically express the bisimulation equivalence between two FSMs. A formal definition of bisimulation for two FSMs is as follows.

Let $FSM_1 = (S_1, \Sigma, Y, f_1, g_1, s_1^{in})$ and $FSM_2 = (S_2, \Sigma, Y, f_2, g_2, s_2^{in})$ with $s_1^i \in S_1$ and $s_2^i \in S_2$. A relation R between S_1 and S_2 is *bisimulation* if the following conditions are satisfied.

1. Start condition:

Both FSMs are in their respective initial states.

$$\Rightarrow (s_1^{in}, s_2^{in}) \in R$$

2. Matching state transitions:

With $s_1^i R s_2^i$, the following three conditions hold true.

(a) for $s_1^i, s_1^{i+1} \in S_1$ and $s_1^{i+1} = f_1(s_1^i, \sigma) \exists s_2^{i+1} \in S_2$ such that $s_2^{i+1} = f_2(s_2^i, \sigma)$

(b) for $s_2^i, s_2^{i+1} \in S_2$ and $s_2^{i+1} = f_2(s_2^i, \sigma) \exists s_1^{i+1} \in S_1$ such that $s_1^{i+1} = f_1(s_1^i, \sigma)$

(c) $s_1^{i+1} R s_2^{i+1}$

3. Matching outputs: $y_1^i = y_2^i$ (OR) $g_1(s_1^i, \sigma) = g_2(s_2^i, \sigma)$

S_1 and S_2 are bisimilar if S_1 and S_2 share bisimilar relation. If S_1 and S_2 are bisimilar, then the corresponding FSM_1 and FSM_2 are bisimilar. A bisimulation equivalency between two FSMs, FSM_1 and FSM_2 is indicated by $FSM_1 \sim FSM_2$.

The notion of bisimulation defined above is a strong bisimulation. The definition of strong bisimulation requires that each transition in the behaviour of one FSM should be matched by a single state transition of the other, regardless of whether that transition is labelled by an observable action or by a silent action represented by τ (adopted from Section 3.4, [98]).

There exists a weaker version of bisimulation. In weak bisimulation, an FSM is allowed multiple internal transitions (formally called τ -action) to provide the same output sequence for a given input sequence. The weak bisimulation equality is also known as *observational equivalence*. The observational equivalence abstracts away the internal behaviour (state transitions and silent or uninteresting outputs) and only focuses on the input - output equivalence. Hence, our requirement of behavioural equivalence corresponds to observational equivalence on FSMs. A weak bisimulation equivalency between two FSMs, FSM_1 and FSM_2 is indicated by $FSM_1 \approx FSM_2$.

4.4.4 Hierarchical Networks

Hierarchy is the natural way of building scalable systems. Hierarchical technological systems like Internet and social systems like metropolitan cities are a testament to the power of hierarchy in building scalable systems. We use the same principle in creating the axiom of hierarchy.

Axiom 4.4 (Hierarchy). *A node can represent a sub-network. The representative node must satisfy the equivalence axiom with the sub-network being represented.*

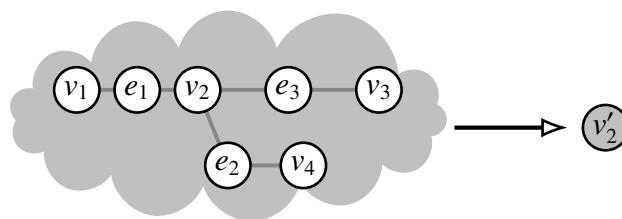


FIGURE 4.5: Illustration of the axiom of hierarchy. Here v'_2 is a representative for the sub-network consisting of $\{v_1, v_2, v_3, v_4, e_1, e_2, e_3\}$.

Figure 4.5 shows a graphical illustration of the axiom of hierarchy. The nodes and connections highlighted by the cloud which form a sub-network of a bigger network have been abstracted out. In its' place, we have a representative node v'_2 . The external connections of the sub-network remain the same and v'_2 provides observational equivalent behaviour to the sub-network it replaces.

We utilize a very restricted form of hierarchy via FSM equivalence. The semantics of our hierarchical approach have been adopted from David Harel's StateCharts [99, 100]. As per the StateCharts approach, the design of a representative FSM involves the following operations.

AND Two FSMs are put side-by-side and both run concurrently. This arrangement gives concurrent processing capability to the hierarchical FSM.

Cascade Two FSMs are connected sequentially so that the output of first FSM becomes the input of the second FSM.

Composite This operation has also been called an *Or*-operation in prior literature [99]. An FSM can be in any on of its valid states. For example, if

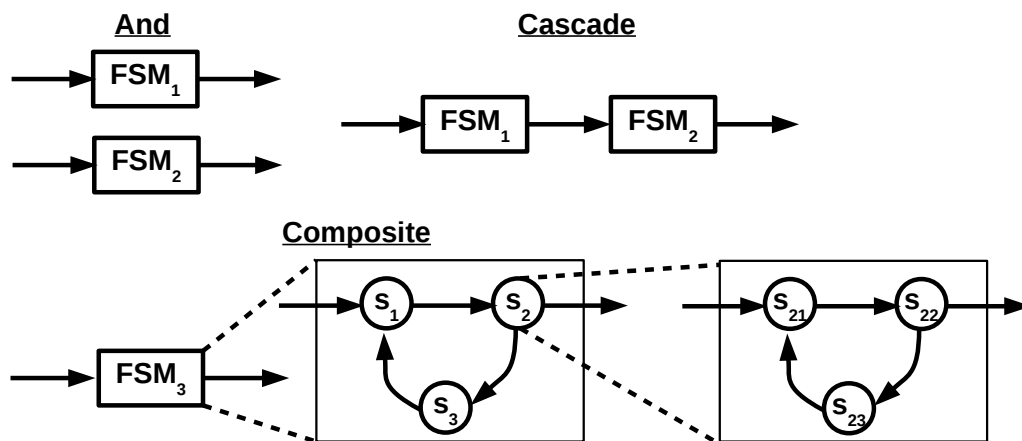


FIGURE 4.6: Elementary operations needed to create hierarchical FSMs.

an *FSM* has only three states s_1 , s_2 and s_3 , then the FSM can be in either s_1 state or s_2 or s_3 state. Hence, the naming of the arrangement as *Or-operation*.

The notion of composition comes from the fact that each state of an FSM can contain within it another FSM. In Figure 4.6, FSM_3 has three states of which s_2 contains another state machine inside it. Thus s_2 is a composite state and FSM_3 is a composite state machine.

These operations are illustrated in Figure 4.6. The AND and OR operations on FSMs are thoroughly described in StateCharts approach [100]. The state machines modeling of Unified Modeling Language (UML) v2.5 is a modern adaptation of StateCharts approach [101]. The semantics of composite operation defined here are in line with the definition of composite state and FSM in UML v2.5 [101]. Modifications on the StateCharts approach have been adopted for modeling of cyber physical systems in Ptolemy II software [102] and for modeling of biological systems using reactive animation [103, 104]. The Cascade operations on FSMs are detailed in Ptolemy approach [102, 105].

4.5 Generality of the Cell Model

In this section, we show the equivalent Cell Model representations for the existing network models. We start with undirected connections, proceed to work

TABLE 4.3: Representation of undirected connections in the Cell Model.

Type of Connection	Function	Operation
Unweighted and Undirected	$y = x$ with $f : X \rightarrow X$	Forward
Static weight	$y = f(x, w)$ with $f : X \times \{w\} \rightarrow Y$, $w \in Z^+$ and f is bijective	Transform
Dynamic connections	$y = f(x, t)$ with $f : X \times T \rightarrow Y$	

with directed connections, multiplex (multiple) connections and conclude with multidimensional connections.

4.5.1 Undirected Connections

The connection illustrated in Figure 4.1 shall be used to explain the representation for undirected connections. The undirected connections may be of three kinds: unweighted, static weights and dynamic connections. All types of undirected connections can be represented using functional view. We utilize the notation for functional view given in Equation 4.3 to provide an equivalent representation in Table 4.3. Dynamic connections are modeled as connections whose weight changes with time. We model time using one independent variable $t = a_1 \in A_1$ given in Equation 4.3.

The unweighted and undirected connections use forward operation of functional view. The forward operation is a projection operation, i.e., $f(x, a_1, \dots, a_n) = x$. The weight of an undirected edge is represented as a function whose domain is X , range is Y and does not depend on the environment variables. The last case of dynamic connection requires consideration of time as well. We model the input of dynamic functions as input messages and time.

4.5.2 Directed Connections

A directed connection has source node and a destination node. We can extend the equivalence presented in Section 4.5.1 for undirected connections with one additional restriction. All the input messages of a connection come only from one node and all the output messages go only to the other node of a connection. The situation is illustrated in Figure 4.7.

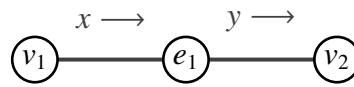


FIGURE 4.7: Modeling of the directed connections using functional view of a connection.

4.5.3 Multiplex Connections

We then consider the case of multiplex (multiple) connections between two nodes. The Cell Model equivalent representation for multiplex connections is given in Equation 4.5. The meaning of the symbols such as Σ, X, A_i is as described in Equation 4.3.

$$\begin{aligned}
 \text{Function Input: } \Sigma &= X \times A_1 \times A_2 \times \dots \times A_n \\
 \sigma &= (x, a_1, a_2, \dots, a_n) \in \Sigma \\
 \text{Output messages: } Y^i &= \{y_1^i, y_2^i, y_3^i, \dots, y_{p_i}^i\} \text{ for } i = 1, 2, \dots, h \\
 \text{Function: } f^i &: \Sigma \longrightarrow Y^i \cup \{null\}
 \end{aligned}
 \tag{4.5}$$

A multiplex connection is a set of h connections between two same nodes. For each function input ($\sigma \in \Sigma$), every multiplex connection sends one transformed output messages ($y^i \in Y^i$). All the transformed messages are received at the destination node. An example scenario is shown in Figure 4.8a. These connections are represented using one single node in Cell Model; the illustration is shown in Figure 4.8b.

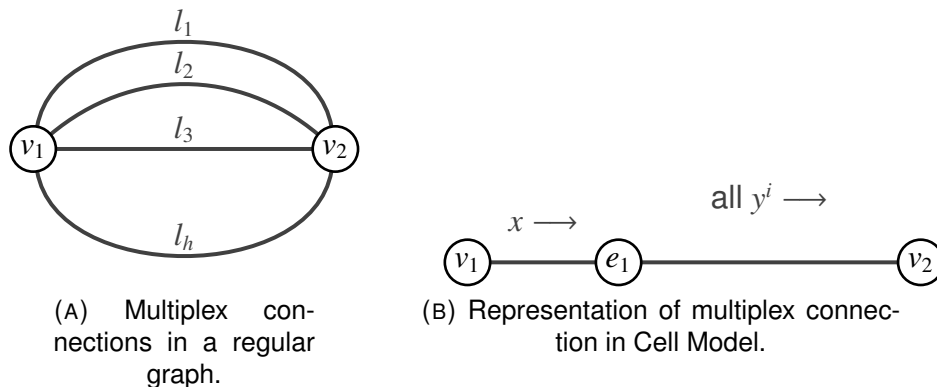
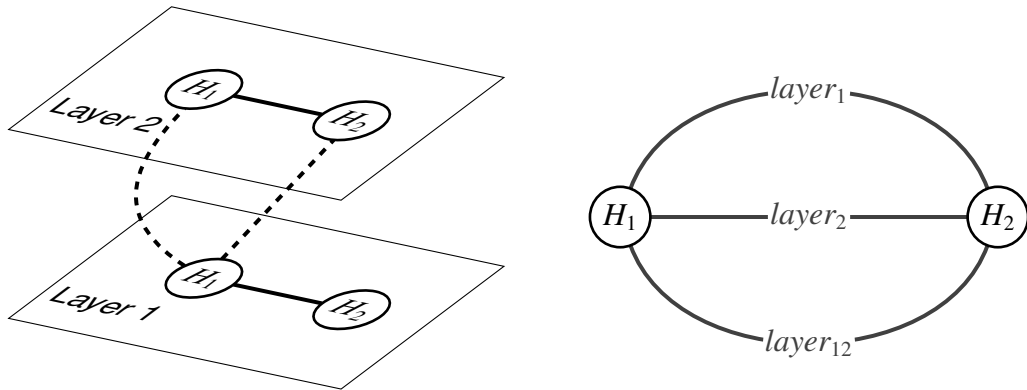


FIGURE 4.8: Modeling of multiplex connections using functional view of a connection.



(A) A multidimensional / multilayer network with two layers illustrated.

(B) An equivalent representation for multidimensional connections.

FIGURE 4.9: Multidimensional connections and their representation in the Cell Model.

4.5.4 Multidimensional Networks

Using the axiom of hierarchy, we can represent all the sub-network induced by all copies of v_i^2 as a single node. Thus all the inter-layer connections between copies of a single node are abstracted away. The resulting multidimensional network will be a labeled multiplex network. We can use the technique discussed in Section 4.5.3 to convert Figure 4.9b into an equivalent Cell Model representation.

4.6 Mapping of the Cell Model to Applications

We apply the Cell Model to two different application domains. The first is a transit scheduler which is responsible for generating travel itineraries from public transit schedules. The second is a packet analyzer which is responsible for protocol analysis of network traffic. The modeling conventions of two different application scenarios using the Cell Model of network are shown in Table 4.4. Different views adopted for nodes and connections of application networks are shown in Table 4.5.

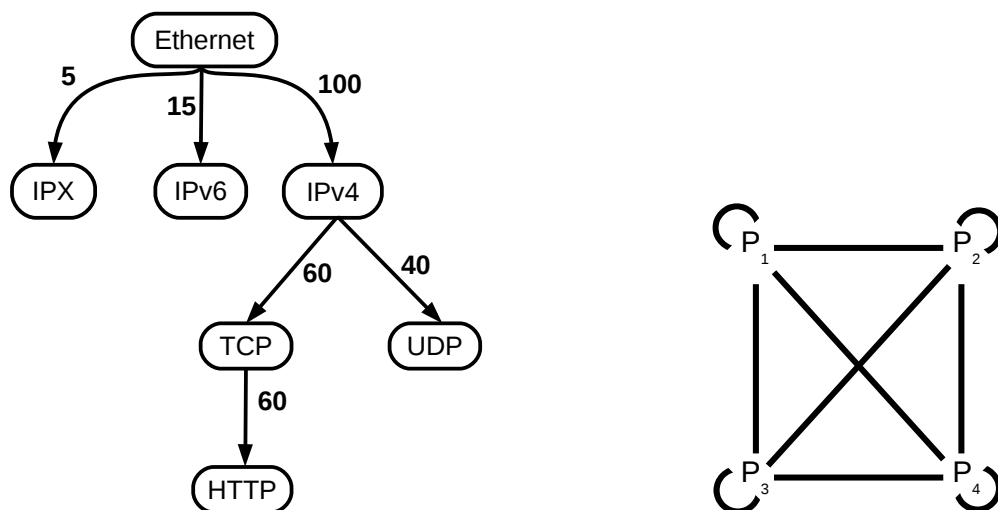
²A copy of v_i may exist in each dimension of a multidimensional network

TABLE 4.4: The modeling of different participants using the Cell Model.

	Transit Scheduler	Packet Analyzer
Network	Transit network	Protocol graph
Nodes	Transit stations	Protocol analyzer
Connections	Road links	Protocol clients
Messages	Transit vehicles	Packets

TABLE 4.5: The views adopted for different participants of the Cell Model.

	Transit Scheduler	Packet Analyzer
Nodes	Behavioural	Behavioural
Connections	Function	Function
Network	Structure	Structure



(A) Sample protocol parse graph with connection weights.

(B) Completely connected (K_N graph with Self-loops) Pipeline. Here N indicates the number of vertices / pipeline stages. In this figure, we illustrate K_4 .

FIGURE 4.10: Application of Cell Model to protocol analysis. The hierarchical nature of the Cell Model can be seen in (4.10b) where each node can embed multiple nodes of the graph given in (4.10a).

4.6.1 Network Measurements

We consider the problem of analyzing the network packet traffic. A typical protocol parse graph of the network traffic is shown in Figure 4.10a. We are interested in concurrent analysis of the protocol parse graph using a multi-threaded implementation of protocol analysis. We create a protocol analysis pipeline as a

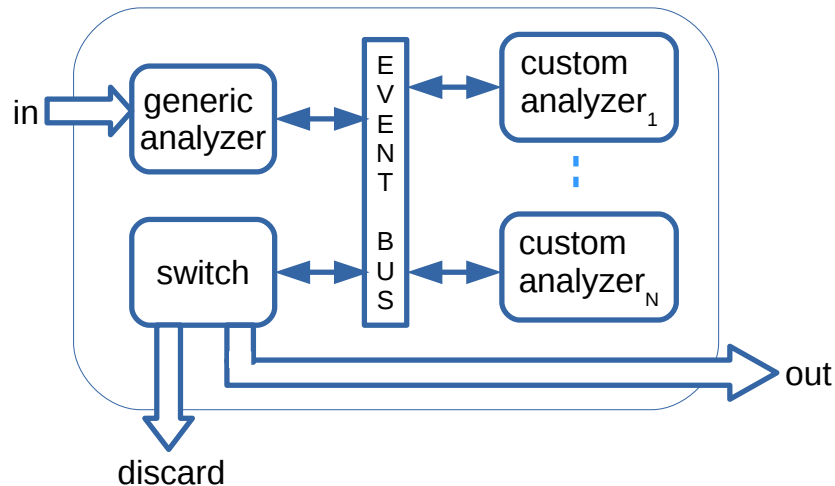


FIGURE 4.11: Generic analyzer cell (GAC) that implements one stage of a pipeline. The same GAC is configured differently for different pipeline stages.

multi-threaded implementation of protocol analysis stages. If we model the implementation stages as nodes and message passing capability between nodes as connections, then we have a complete network formed with self-loops at each node (shown in Figure 4.10b). We embed the parse graph in our analysis pipeline and then perform the protocol analysis.

The analysis pipeline has been built from a symmetric node element named the generic analyzer cell (GAC). Each GAC forms one node of the network shown Figure 4.10b. The architecture of GAC is shown in Figure 4.11. The GAC implements nodes as per the conventions of the Cell Model of network. This application demonstrates the hierarchical nature of Cell Model. Each node of the analysis pipeline is a representative for a sub-network.

4.6.2 Transit Networks

We consider the problem of searching for feasible itineraries in the schedules of public transit networks.

Problem Statement: Create a list of travel itineraries from multi-modal public transit timetables in response to user queries. Transit timetables are expressed as a series of connections [5]. A connection is defined as

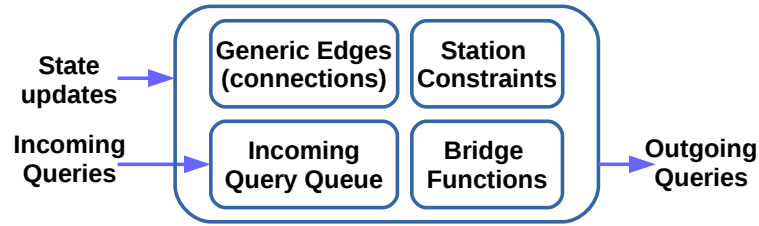
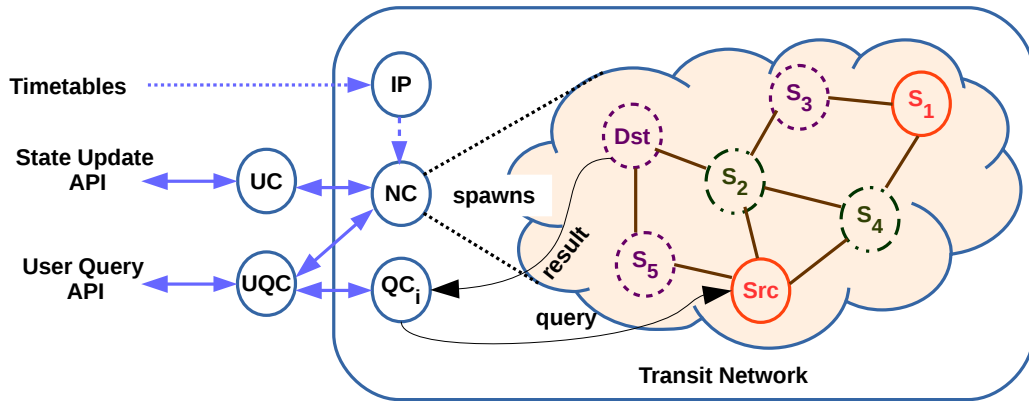


FIGURE 4.12: A behavioural view for a multi-modal transit station. The station constraints, bridge functions, and generic edges together constitute the state, and behaviour of a station process.



Legend

- IP – Input Parser
- NC – Network Constructor and Controller
- UQC – User Query Controller
- UC – Update Controller
- QC_i – Query Collector for ith Query

Operator – Station Mapping

- Operator₁ – Src, S₁
- Operator₂ – S₂, S₄
- Operator₃ – Dst, S₃, S₅

FIGURE 4.13: Architecture of the Transport Scheduler application.

$$C = (S_{dep}, t_{dep}, S_{arr}, t_{arr}, tr)$$

where,

- tr = unique trip identifier
- S_{dep} = departure station of a connection
- t_{dep} = departure time at S_{dep}
- S_{arr} = arrival station of a connection
- t_{arr} = arrival time at S_{arr}

We construct the physical topology of the transit network from timetables. Each transit station is considered as a node in the transit network and is modelled using Cell Model of network. The structure of a transit station is shown in Figure 4.12. The architecture of the transit application is shown in Figure 4.13. We adopt the following approach to model the public transit network.

In the transit station model shown in Figure 4.12, we can see the details of subsumed connections as well. The function of a connection is specified as follows.

$$\begin{aligned} \text{Independent variables: } \quad & \text{Time}(Ti) = \{0, 1, 2, \dots, n\} \\ & \text{Mode}(M) = \{car, bus, flight, train, walk\} \\ \text{Cost function:} \quad & f : \mathbb{R}^+ \times M \longrightarrow Z^+ \end{aligned} \quad (4.6)$$

$$\text{cost(connection)} = \begin{cases} \text{finite} & \text{for } t \in Ti \subset \mathbb{R}^+ \text{ and } m \in M \\ \infty & \text{for all other cases.} \end{cases} \quad (4.7)$$

Equation 4.6 represents the cost of using a connection in a transit network; Here, time and mode of transport are independent variables. In practice, the cost of the connection function is finite only for few $(time, mode) \in Ti \times M$; this insight has been expressed in Equation 4.7. Specific conditions (constraints) are put on the connections; these constraints have to be satisfied by the path in order to use the connection in a path.

We compute travel itineraries as paths in time domain. Let $cost(path_i)$ denote the cost of path from source station to $station_i$. Let e_j be the edge under consideration for travelling from $station_i$ to $station_j$. Let $cost(path_{i+1})$ denote the cost of extending the $path_i$ from $station_i$ to $station_j$ by using the edge e_j . Then the path cost equation for this scenario is given in Equation 4.8.

$$cost(path_{i+1}) = cost(path_i) + cost(e_j) \quad (4.8)$$

The computed path cost for $cost(path_{i+1})$ given in Equation 4.8 becomes finite

TABLE 4.6: Connection constraints, path constraints and their compatibility

Connection Constraints	Path Constraints	Compatibility
flight	any mode	yes
flight	bus	no
train	bus or train arrival before 5PM	yes only if computed arrival time < 5PM
train	bus-flight-bus	no

only if the edge constraints of e_j are compatible with the path constraints of $path_i$. Otherwise, the computed path cost for $cost(path_{i+1})$ becomes infinite. A few sample edge, and path constraints are shown in Table 4.6.

We have implemented the ideas of transit station, generic edges and cost computation in a concurrent application, named transport scheduler. The architecture of transport scheduler is shown in Figure 4.13. The transport scheduler models multi-modal transit network as a network of cells each of which represents a transit station. All the processing in the cells is concurrent, thus making the application itself concurrent. The application supports multi-modal itinerary search queries and schedule updates.

4.7 Results

4.7.1 Network Measurements

Using the concepts developed in Cell Model, we implement Darshini - a modular, concurrent, collaborative and customizable protocol analyzer workbench software package. We are able to provide the following advantages through the Darshini software package.

1. Concurrent protocol analysis.
2. Configurable speed-vs-memory tradeoff.
3. Configurable protocol parse graph.
4. Persistence of analysis.
5. Collaboration

We provide further details of the protocol analyzer in Chapter 5.

4.7.2 Transit Networks

We implement multi-modal transit scheduler for public transit networks of India. We are able to provide the following new features through the transit scheduler software package.

1. Include personal transport facilities in constructing travel itineraries.

2. List of travel itineraries for a query.
3. Operator control over the transit stations.
4. Uniform processing of multi-modal edges using edge constraints.

We provide further details of the multi-modal transit scheduler in Chapter 6.

4.8 Implementation Choices

Our proposed Cell Model is based on message exchange between participants. Thus we need implementation platforms that favour message passing for communication. We demonstrate two implementations of the Cell Model for two different application domains using two entirely different implementation platforms.

For Darshini protocol analyzer, we choose a thread-based implementation with Google Guava's eventbus [106] for distribution of messages between the participants of the Cell Model. The nodes and connections of the protocol analyzer are implemented inside the Java threads.

For the multi-modal transit network implementation, we use an Actor model-based approach. Actor model is an established concurrency model [107]. Implementations of Actor models exist in different programming languages such as Elixir [108], and Java [109].

4.9 Summary

We propose the Cell Model of a network for modeling multidimensional networks. We use three views, namely structural, behavioural and functional views, to represent the participants of a network. The behavioural view helps create process view of a network. All the network models built using the Cell Model obey four axioms: activeness, message exchange, equivalence and hierarchy. We have shown the generality of the Cell Model by representing undirected, directed, multiplex and multidimensional networks using the Cell Model. We have applied our Cell Model on two research problems and our proposed solutions to these problems have been successfully demonstrated on real-world networks.

Chapter 5

Darshini - Protocol Analyzer

5.1 Motivation and Problem Definition

Network packet capture and protocol analysis helps observe the packet traffic flowing through a computer network. Network packet capture and protocol analysis is an integral part of modern network management [30, 110]. A major concern in network measurements community is the lack of emphasis on the application of scientific (repeatable, verifiable and falsifiable) measurement principles [111]. A requirement of researchers and operations engineers is to control / restrict the packet analysis to scenarios of interest as implied by the experimental objectives [112]. Thus user-directed protocol analysis is an important requirement on packet capture and analysis tools. The network measurement community needs *collaboration* and *user-directed protocol analysis* features together in one measurement tool. An ideal measurement tool would also enable *scientific measurements*.

Centralized data repositories such as Cawdad [113] and DataCat [114] maintain useful network measurement datasets created using scientific measurement principles. In the network traffic datasets placed in public domain, the process of creating and documenting experimental design is ad-hoc. Having a packet capture tool that facilitates experimental design, documentation and collaboration would be useful in creating templates for measurement data exchange. In addition, the longitudinal evolution of network traffic mix [115, 116, 117] requires user-directed protocol analysis. We find that the available tools

offer typically two of the required three features – scientific measurements, collaboration and user-defined protocol analysis.

For example, popular protocol analysis tools like Wireshark [31] are developed for the scenario of lone engineer analyzing the captured packet stream on a local machine. Hence the concept of collaborative analysis is not a standard feature in these tools. Persistence is not a standard feature of these tools; thus collaborative analysis becomes repetitive. The experts / reviewers are asked to look at a pcap file without the associated experimental meta-data. Another limitation is the fixed configuration in measurement tools denying users the ability to select / filter protocols of their interest for further analysis.

5.2 Packet Analysis as Graph Embedding

5.2.1 Motivation

We are interested in faster processing of captured packets by increasing the throughput of a packet analyzer. One of the ways of achieving higher throughput is to use concurrency. We can strive for concurrency at the level of protocols. If we have enough processing resources in the form of parallel processors, all the protocols of the protocol parse graph can be processed in a concurrent mode. If not, we strive for maximum possible concurrency. Since the protocol parse graph is a weighted and rooted tree, we architect the concurrent solution to respect the acyclic ordering of protocols implied by the protocol parse graph. The incoming packet traffic need to be processed in the strict ordering of protocols implied by the packet headers. Thus our concurrent solution needs to be a pipeline of protocol parsers. The interconnections among the pipeline stages are controlled by the edges of the protocol parse graph. One advantage of our solution is the concurrent analysis of multiple packets.

We can formulate the mathematical details of a protocol parse graph as follows. Let $G_1 = (V_1, E_1)$ be a rooted, labeled tree. G_1 is used to represent protocol parse graph. We use the following notations to represent different aspects of the protocol parse graph in terms of weighted version of G_1 .

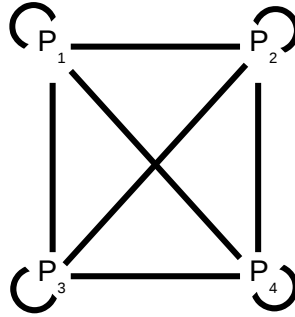


FIGURE 5.1: Completely connected (K_N graph with self-loops) Pipeline. Here N indicates the number of vertices / pipeline stages. In this figure, we illustrate K_4 .

$$G_1 = (V_1, E_1) \quad (5.1)$$

where,

v_a = a vertex in V_1 , $a \in [1, |V_1|]$
 = one protocol in the protocol parse graph

C_{v_a} = weight of the vertex $v_a \in V_1$

e_b = simple, directed and weighted edge between two vertices of V_1
 = possibility of a protocol PDU being embedded as payload of another protocol

w_b = weight of the edge $e_b \in E_1$
 = expected number of user protocol PDUs in the payload of provider protocol PDUs

The weight function f_{w_1} is defined as $f_{w_1} : E_1 \rightarrow \mathbb{N}$. The set intervals are on the set \mathbb{N} .

In a simple sequential execution model, one packet is handled at a time; packet analyzer tools such as tshark, tcpdump take this approach. At the other extreme is the concurrent solution in which one protocol is allocated to one concurrent pipeline stage and the pipeline stages are connected as per the edges of the protocol parse graph. But that would result in too many pipeline stages and is not ideal for software environments where cost of implementing a pipeline stage is high. In order to assign the task of parsing a protocol to a pipeline stage, the processing capacity of the pipeline stage must be greater than or equal to the

requirements of the protocol. The aim is to minimize the number of pipeline stages to complete the work; Within this constraint, we need to minimize the cost of distributing packets between the pipeline stages.

5.2.2 Pipeline as Graph

A representation of a complete graph of all pipeline stages with self-loops for each of the stages is shown in Figure 5.1. We use this graph to represent the pipeline stages.

We implement the communications / connections among the pipeline stages in software using *Pub-Sub* design pattern. In the subsequent discussion, we consider mapping of protocol parse graph to K_N pipeline with self-loops.

The concurrent protocol analysis requires mapping of the protocol parse graph into a completely connected pipeline. Thus we need to map one or more protocols into each pipeline stage. If two protocols mapped to a single pipeline stage have a parent-child relation, then we need to facilitate communication within one stage of a pipeline; such a communication is enabled by self-loops of a pipeline stage. If the same two protocols sharing parent-child relation are mapped to different pipeline stages, then we need to facilitate communication between pipeline stages. We can define a pipeline as follows.

$$G_2 = (P, E_2) \quad (5.2)$$

= Complete graph (K_N) with self-loops
for all vertices

= graph representing all pipeline stages

where,

N = order of the graph $G_2 = |P|$

= number of stages in the pipeline

p_i = a vertex of G_2 , $p_i \in P$

= i^{th} stage of a pipeline, $i \in [1, N]$

C_{p_i} = weight carrying capacity of a vertex $p_i \in P$

= processing capacity of i^{th} pipeline stage.

e_{ij} = weighted, simple edge between vertices

p_i and p_j , $\forall i, j \in [1, |P|]$

w_{ij} = weight of e_{ij}

For the single computer and cluster computer development scenarios, the cost of communicating between any two computers (w_{ij}) tends to be nearly same. Also the cost of communicating within a computer (w_{ii}) tends to be same. Hence, we assume

$$w_{ij} = w_1 \quad \forall i, j \in [1, |E_2|] \text{ and } i \neq j \quad (5.3)$$

$$w_{ii} = w_2 \quad \forall i \in [1, |E_2|] \quad (5.4)$$

The weight function, $f_{w_2} : E_2 \rightarrow \{w_1, w_2\}$, where $\{w_1, w_2\} \subset \mathbb{R}^+$

5.3 Graph Embedding: Parse Graph to Pipeline Mapping

In this section, we represent the graph embedding problem as applicable in parse graph to pipeline mapping. We can represent the details of mapped protocols as follows.

Let v_a^i = vertex $v_a \in V_1$ of G_1 mapped to p_i
 = a protocol assigned to i^{th} pipeline stage

V_1^i = all vertices of V_1 that are mapped to p_i
 = all protocols that are assigned to
 i^{th} pipeline stage

then,

$\sum_{v_a \in V_1^i} C_{v_a}$ = cumulative weight of all vertices of V_1^i
 = total cost of processing for all the
 protocols assigned to i^{th} pipeline stage

For realistic mapping, the cost of processing for all the protocols assigned to one pipeline stage must be less than or equal to the processing capacity of that

pipeline stage.

$$\sum_{\forall v_a \in V_1^i} C_{v_a} \leq C_{p_i} \quad (5.5)$$

The cumulative weight of all vertices assigned to p_i of G_2 must be less than or equal to the weight bearing capacity of p_i . Extending the same reasoning to all the vertices of G_2 , we can say that,

$$\sum_{\forall v_a \in V_1} C_{v_a} \leq \sum_{\forall p_i \in P} C_{p_i} \quad (5.6)$$

If all pipeline stages have the same weight bearing capacity C_p , then N – the order of G_2 – is indicated by,

$$N \geq \left\lceil \frac{\sum_{\forall v_a \in V_1} C_{v_a}}{C_p} \right\rceil$$

The upper limit on the number of pipeline stages is imposed by the order of G_1 , i.e., $N \leq |V_1|$.

Thus the limits on N are:

$$\left\lceil \frac{\sum_{\forall v_a \in V_1} C_{v_a}}{C_p} \right\rceil \leq N \leq |V_1| \quad (5.7)$$

Implementing a parse graph on a pipeline is equivalent to graph embedding of G_1 into G_2 . When we embed G_1 into G_2 , we need to transform the edge weights of G_2 . Let the pipeline graph after edge weight transformation be identified as G'_2 . The only difference between G_2 and G'_2 is their edge weights; in all other aspects, G_2 and G'_2 are identical. Therefore,

$$f_v : V_1 \rightarrow P \quad (5.8)$$

$$f_e : E_1 \rightarrow E'_2 \quad (5.9)$$

$$f'_{w_2} : \{w_b\} \times \{w_1, w_2\} \rightarrow \mathbb{R}^+ \quad \text{where } b \in [1, |E_1|] \quad (5.10)$$

Both f_v and f_e are *onto* functions.

The weight of an edge in E_1 (of G_1) indicate the frequency of using an edge; the weight of an edge in E_2 (of G_2) indicate the cost of using an edge. When we map $e_b \in E_1$ into $e_{ij} \in E'_2$, we indicate the use of e_{ij} exactly w_b times, each use incurring a cost of w_{ij} . For the case of multiple edges of E_1 being mapped into one edge in G'_2 , the effective edge weights get added up. We can express the edge weights of G'_2 as follows.

$$w'_{ij} = w_{ij} \times \sum w_b \quad \forall e_b \in E_1 \cap (V_1^i \times V_1^j) \text{ and} \\ i, j \in [1, |E_2|]$$

In protocol analysis, we wish to minimize the number of pipeline stages and the overall communication cost in the pipeline. On G'_2 , we wish to minimize both the order of the graph (N) and the sum of all edge weights ($\sum_{i,j \in [1, |E_2|]} w_{ij}$). Obviously, any mapping has to satisfy the capacity constraints expressed by the Equation 5.7.

In summary, our graph embedding problem can be formulated as the following optimization problem.

Problem Statement: Embed a rooted, labeled, weighted tree G_1 into a weighted, complete graph with self-loops G'_2 such that

Objectives:

- $\min N = \text{order of } G'_2$
- $\min \sum_{i,j \in [1, |E_2|]} w'_{ij} = \text{sum of edge weights of } G'_2$

Constraints

- capacity limits, $\sum_{v_a \in V_1} C_{v_a} \leq \sum_{p_i \in P} C_{p_i}$
- node mapping, $f_v : V_1 \rightarrow P$, f_v is an *onto* function
- edge mapping, $f_e : E_1 \rightarrow E'_2$, f_e is an *onto* function

We wish to embed G_1 into G'_2 with the aim of G'_2 having the smallest order and least cumulative edge weight.

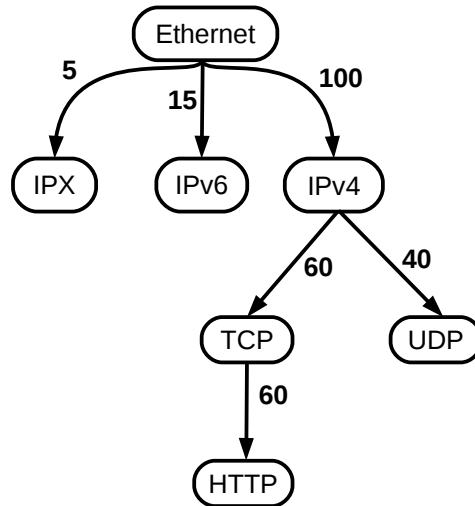


FIGURE 5.2: Sample protocol parse graph with edge weights.

```

graph start {
    ethernet;
}
graph ethernet {
    switch(ethertype) {
        case 0800: ipv4;
    }
}
graph ipv4 {
    switch(protocol) {
        case 06: tcp;
    }
}
graph tcp {
}
graph end {
}
  
```

FIGURE 5.3: A sample protocol parse graph specification selecting packets containing only Ethernet-IP-TCP protocols.

In this chapter, we describe Darshini which demonstrates an implementation of G_1 into G'_2 graph mapping. We allow for modification of protocol parse graph (G_1). Removal of a vertex from G_1 leads to automatic removal of the mapped element from G'_2 . Sections 5.4 to 5.7 describe an implementation G'_2 as a concurrent (multi-threaded) pipeline.

```
header ethernet {
    fields {
        dst_addr : 48;
        src_addr : 48;
        ethertype : 16;
    }
}
```

FIGURE 5.4: Ethernet header specification in P4 language.

```
header ipv4 {
    fields {
        version : 4;
        ihl : 4;
        diffserv : 8;
        totalLen : 16;
        identification : 16;
        flags : 3;
        fragOffset : 13;
        ttl : 8;
        protocol : 8;
        hdrChecksum : 16;
        srcAddr : 32;
        dstAddr : 32;
    }
}
```

FIGURE 5.5: IPv4 header specification in P4 language.

5.3.1 Motivation for Heuristic Solution

The embedding of protocol parse graph into the processing pipeline can be done in two steps. In the first step, a clustering algorithm is run on the protocol parse graph to obtain node clusters. Each cluster can be aggregated to one high-level node. Let us call this aggregated graph as cluster-graph. In the second step, the cluster-graph is embedded into the processing pipeline (a graph). Both the graph clustering and the graph embedding are \mathcal{NP} -hard problems. Thus we require a heuristic solution.

5.4 Darshini Architecture

5.4.1 Input Data Formats

Darshini takes in a stream of packets to operate on. The stream of packets are read from a pcap¹ file. The beautification specification, protocol headers and protocol parse graph are also given as input to Darshini. We use P4 language [34] for specifying these three inputs. The beautification files are used to represent human preferences for representation of protocol header fields (for example, the use of dotted decimal notation for IPv4 addresses).

Darshini uses two different parse graphs. One is the parse graph of all supported protocols; prior literature refers to this graph as *union parse graph* [27]. The second parse graph is the user-specified parse graph indicating the protocols of interest to the user. For all practical purposes, user-specified parse graph is a sub-graph of the union parse graph. Both the parse graphs are specified in P4 language. An example protocol parse graph is given in Figure 5.2.

Darshini parses the incoming packets in order to analyze the protocol stack of the packet. A sample protocol parse graph is shown in Figure 5.3. This protocol parse graph only selects packets containing Ethernet-IPv4-TCP protocol headers. We can thus use parse graph input specification to select protocols of interest.

The P4 language specification also allows us to specify the sizes of different headers fields of a protocol. Figure 5.4 shows the Ethernet header specification in P4 language. Ethernet protocol header contains three fields – source address of 48 bits, destination address of 48 bits and a packet type of 16 bits. This protocol-specific information is encoded in the P4 specification of Ethernet header. Similarly, Figure 5.5 shows the specification of IPv4 protocol header in P4 language. Different fields of IPv4 header are specified in strict sequential order.

¹pcap – short for packet capture – is the file format used by packet capture tools like Wireshark to store the captured packets.

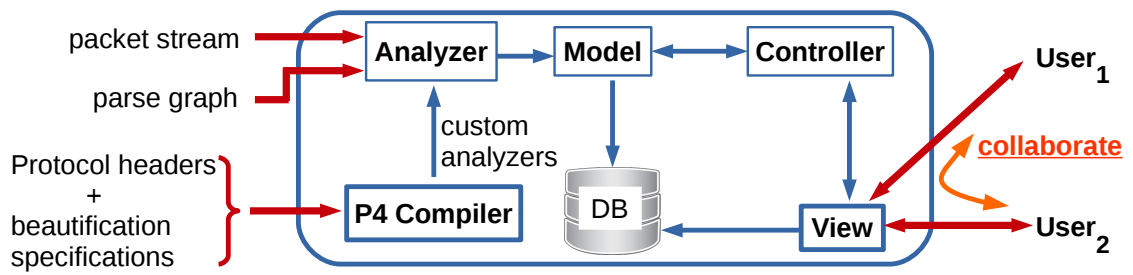
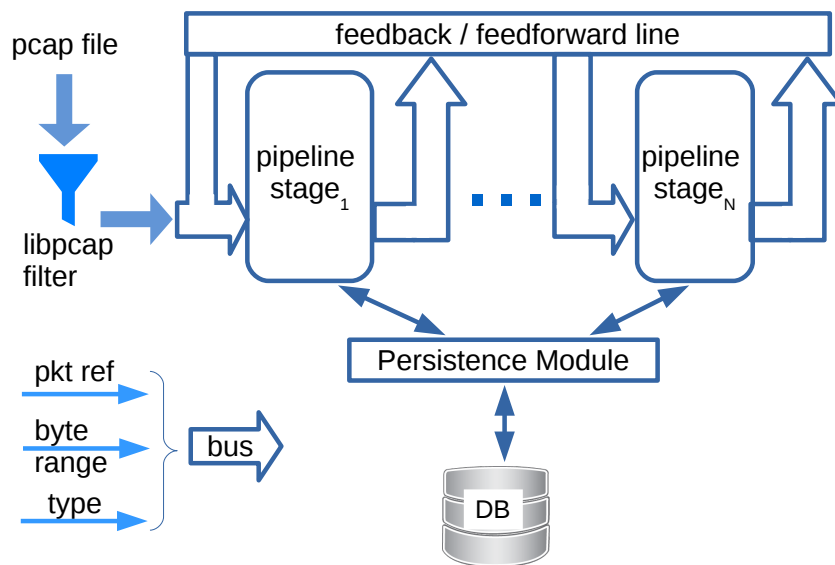
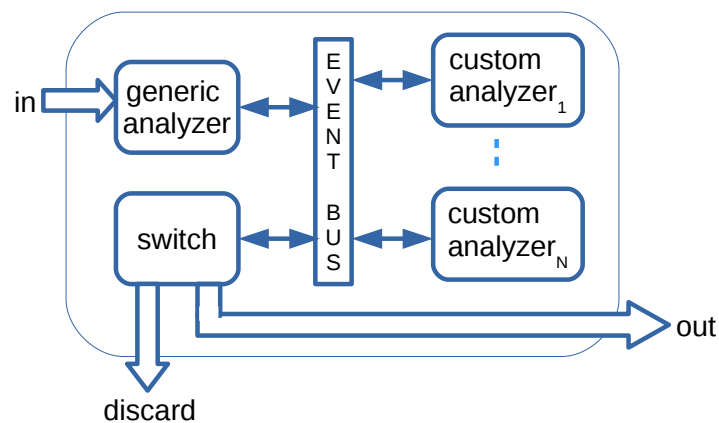


FIGURE 5.6: System architecture of Darshini.



(A) Analyzer module has parser pipeline and it interfaces with persistence module.



(B) Generic analyzer cell (GAC)

FIGURE 5.7: Protocol analyzer pipeline. Each stage of the pipeline consists of one GAC which is illustrated in part-(b).

5.4.2 System Overview

An outline of the system architecture is shown in Figure 5.6. The major building blocks of Darshini are: analyzer pipeline, P4 compiler, database (DB) and MVC components. Our analyzer pipeline receives custom analyzers created by P4 compiler. Analyzer pipeline uses user-defined parse graph and custom analyzers to parse the packet stream. Analyzer pipeline internally follows pipes-and-filters architectural pattern.

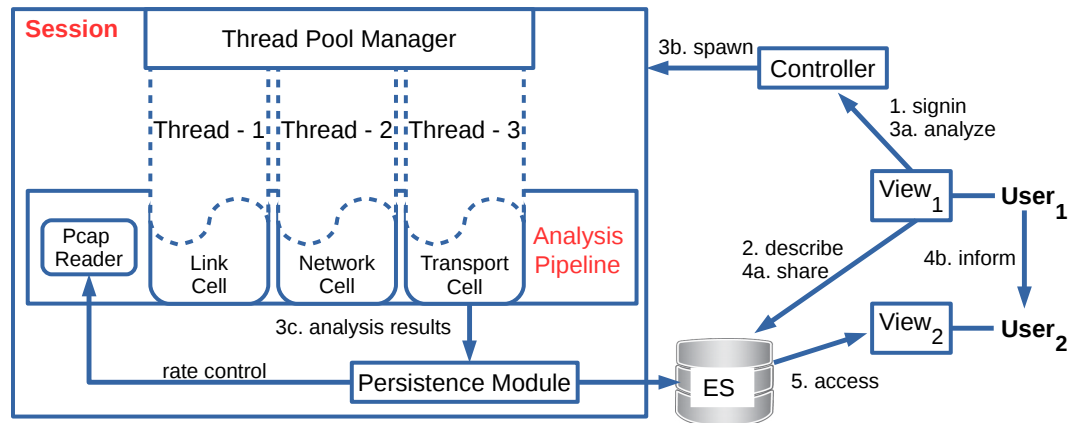
Model module is a part of MVC architecture that stores all the data of analyzers; model interacts with controller and DB. Controller module is also a part of MVC architecture and glues the model with views. Controller is responsible for handling all requests from view module and making method calls to models module. Controller returns data to view module which presents formatted data to user.

View module can directly interact with database over REST API. View module facilitates collaboration between users.

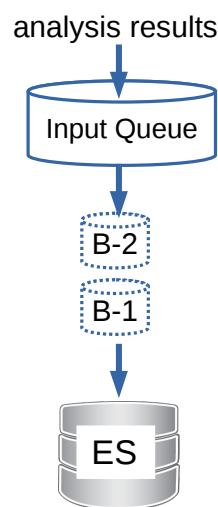
5.4.3 Analyzer Pipeline

Analyzer pipeline forms the backbone of Darshini. Analyzer pipeline is responsible for taking in a filtered stream of packets and analyzing these packets as indicated by the user-specified parse graph. Analyzer pipeline stores the analysis results in DB via the persistence module. The analyzer pipeline architecture is shown in Figure 5.7a.

Each pipeline stage receives a packet and completes analysis for the protocols the stage is responsible for. It then forwards the packet to next stage. A packet is sent to next stage using the feedforward / feedback line. In order to accommodate tunneling scenarios, a feedback line connects a stage to itself or to one of the former pipeline stages. It is possible to encounter packets that do not have PCI header for a protocol layer (ex: raw packets with only TCP header would obviously miss the IP header). Feed forward line enables skipping of a pipeline stage where necessary.



(A) Session creation for one run of protocol analysis.



(B) Persistence queue. B-1, B-2 etc are batched queries.

FIGURE 5.8: Implementation details of the system architecture for Darshini. ES is an acronym for Elastic Search database.

5.5 Generic Analyzer Cell (GAC)

All the pipeline stages are created from the same template named GAC. The block diagram of the GAC is shown in Figure 5.7b.

All the incoming packets of a GAC are received by the generic analyzer. Generic analyzer collects statistical / flow information from the packet for record keeping purposes. Generic analyzer pushes the collected information to the persistence module. After this, the generic analyzer informs all the registered custom analyzers of the analyzer cell about the available packet. An appropriate custom

TABLE 5.1: API end points for Darshini. The base URL of host and ES URLs are not shown.

API URL	Service
/	home page
/signup	user signup
/signin	user login
/session/validate	validate parse graph
/session/analyze	start protocol analysis

analyzer picks up the packet to extract protocol headers. As soon as the protocol header extraction is done in a custom analyzer, next protocol is determined. The processed packet is then forwarded to next analyzer cell; extracted protocol headers are forwarded to the persistence module.

Generic and custom analyzers complete their work much faster than the persistence module (database); the data path from cell input to custom analyzers forms *fast path* of execution. Data path from custom analyzers to persistence module works at much slower rate and is called *slow path*.

Analyzer cells decouple the fast and slow execution paths. As soon as the output of the faster execution path is ready, the processed packet is passed to the next pipeline stage. The slower execution path of a pipeline stage can have long input and output queues to adjust to the packet processing throughput disparity between the fast and slow paths.

5.6 Implementation

Darshini has been implemented as a model - view - controller (MVC) architecture based web application. Figure 5.8 shows the modules of Darshini. The following parallels exist between Darshini and graph G'_2 .

- $p_i \leftrightarrow$ GAC
- $e_{ij} \leftrightarrow$ feedback / feedforward line
- $e_{ii} \leftrightarrow$ Event Bus in GAC

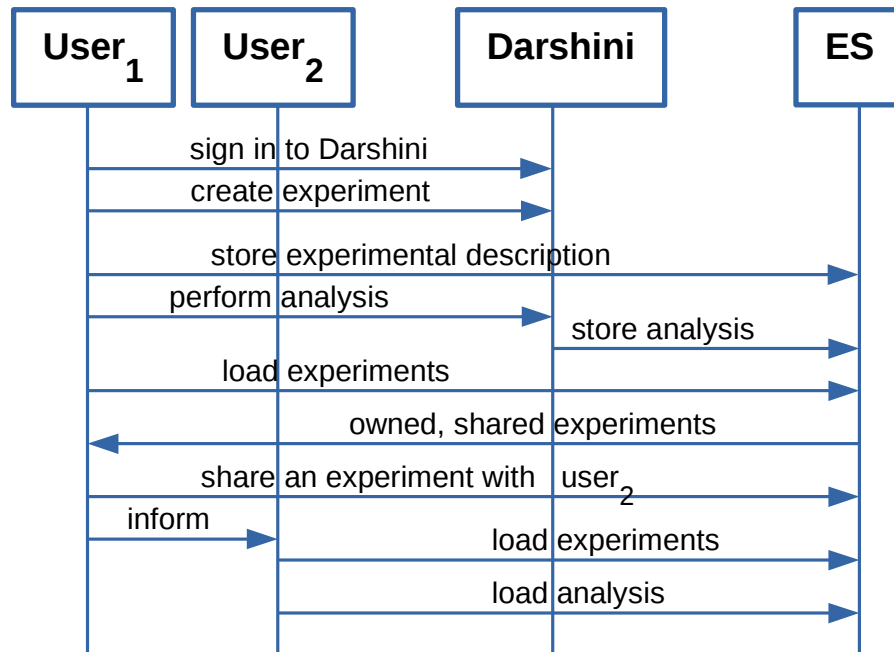


FIGURE 5.9: Sequence diagram illustrating collaboration inside BITS Darshini.

A brief description of different system modules is given below.

Model Consists of all the Java objects representing the data (persistence package) as well as analyzer, protocol and utils packages. All model objects get saved in Elastic Search.

View Consists of client-side (web-browser) code. We use backbone.js Model-View framework to implement client-side functionality for Darshini in the browser. Based on context, view either interacts with server-side controller or with Elastic Search (ES). The API URLs accessed by the client-side code are listed in Table 5.1.

Controller Controller is responsible for authenticating the users. Controller is also responsible for an on-demand launch of a session to manage analyzer pipeline of a packet analysis experiment.

Session Corresponds to one independent protocol analysis. The pipeline protocol analysis itself is performed using Java Threads. Each analyzer cell of an analyzer pipeline is run on a dedicated thread. All custom analyzers persist the protocol analysis results to ES. Details of session and analyzer pipeline are illustrated in Figure 5.8a. The sequence of steps involved in one protocol analysis request are illustrated in Figure 5.8a.

Elastic Search (ES) A plug-and-play module that provides base for persistence of application data, especially the packet analysis data.

Persistence Responsible for managing the speed mismatch between fast analyzer pipeline and the slow ES. The speed mismatch is managed using a two-stage queue as illustrated in Figure 5.8b. Analysis data from custom analyzers is put into batches and handed over to Elastic Search.

5.6.1 Collaborative Analysis

Darshini enables users to share experimental results with other users. A typical sequence of actions taken for collaboration within Darshini are illustrated in Figures 5.8a and 5.9.

User signs into Darshini and creates a new experiment; User is also responsible for supplying experimental description (meta-data). The experimental description gets stored in Elastic Search (ES) database. User then initiates experimental analysis; controller component of Darshini receives user's command and spawns an analysis pipeline to complete protocol analysis. The analysis pipeline persists the results in ES. ES makes the analysis results available to clients via REST API. A user can preview the results of all the analyses done previously. Users have access to two categories of experiments: owned and shared. *Owned* experiments are the experiments created by self; *Shared* experiments are the experiments shared with a user by the other users.

5.6.2 User-Defined Protocol Analysis

Darshini comes pre-configured with a union parse graph that acts as a base graph from which experimenter selects a sub-graph. In this work, we show results for a static mapping from union parse graph to the analyzer pipeline. User has complete freedom to specify any sub-graph of union parse graph for each experiment.

5.6.3 Measurement Workbench

We created a system of measurement workbench where the Internet measurements can go through the measurement cycle (Objective → Strategies → Measure → Analyze → Refine Objective).

Darshini facilitates the measurement cycle (see Table 5.2) using the measurement strategies suggested by Vern Paxson [111].

5.7 Experimental Results

5.7.1 Dataset

We measure the performance of Darshini by using offline pcap files. These offline pcap files contain unfiltered network traffic captured on an edge computer connected to a mid-level enterprise network having approximately 5000 users. Since Darshini is better suited to perform offline protocol analysis on the traffic of

TABLE 5.2: Support for measurement strategies in Darshini.

Measurement Strategy	Implementation
Maintain meta-data	Experimental description page
Error detection	Auto-detection possible with Elastic Search queries
Reproducible analysis	Experiment history
Sub-sample large data	Sub-sampling in protocol domain using parse graph; time domain sub-sampling possible via REST API queries
Periodic analysis	Available as a service
Data reduction scripts	Elastic Search as a service to execute dynamic queries from users
Outlier detection	Supported through REST API interface
Comparing multiple measurements	Supported through REST API interface
Public datasets	Share experiment with other users; avoids sharing pcap files

TABLE 5.3: User-defined protocol analysis on a pcap file with 1,508,352 packets. The size of pcap file is 955MB.

Selected Protocols	Execution time (sec)			Memory consumption (MB)		
	A	B	C	A	B	C
<i>eth, ipv4, tcp</i>	11.3	40.2	394.8	770	128	220
<i>eth</i>	11.3	25.6	115.9	770	128	238

A - tshark

B - Darshini in non-persistent mode

C - Darshini in persistent mode

small to medium-scale networks, mid-level enterprise traffic is a representative test scenario for Darshini.

In this section, we compare the performance of Darshini with tshark tool. Darshini is run in two modes – persistent mode and non-persistent mode. In *persistent mode*, analysis results are saved to Elastic Search. In *non-persistent mode*, analysis results are not saved.

5.7.2 User-defined Protocol Analysis

We consider two protocol parse graphs, namely P_1 and P_2 for demonstrating the user-defined protocol analysis capability of Darshini. P_1 contains protocols *eth*, *ipv4*, *tcp* and P_2 contains just *eth*. We complete the user-defined limited protocol analysis using parse graphs P_1 and P_2 on Darshini. The execution time and run-time memory consumption results of these two experiments are shown in Table 5.3.

Our experiments on the use of Darshini with the available datasets leads to the following three conclusions. First, the run time performance of Darshini is inversely proportional to size of parse graph. Darshini is thus suitable for user-defined limited protocol analysis. Second, the fast path (from input to custom analyzers of generic analyzer cell) is order of magnitude faster than the slow path (from custom analyzers to Elastic Search). The run-time performance of Darshini is limited by the database storage performance of Elastic Search.

Third regarding the memory usage requirement, Darshini uses 128MB for processing a pcap size of 955MB, where as tshark consumes 770MB for processing the same file. The reported number of 128MB includes the memory allocation to Java Virtual Machine (JVM) and Tomcat Servlets. Thus Darshini is more memory efficient when compared with tshark.

5.7.3 Memory Management

We can control batch sizes of queries sent to Elastic Search. Batch size is a configurable parameter for Darshini. With a batch size of 20,000 queries, Darshini processes 1.5 million packets and saves analysis data to Elastic Search in 224 seconds with a maximum memory consumption of 414 MB. With a batch size of 5,000 queries, Darshini processes the same pcap file in 908 seconds with a maximum memory consumption of 235 MB. Thus we can use the batch size to make trade offs between run time vs memory consumption.

5.7.4 Throughput

Figure 5.10 shows the relative performance of packet parsing tools. We compare the tools based on their execution time and memory consumption. Fig 5.10a shows the relative execution times of all the packet parsing tools for different pcap files. Figure 5.10b compares packet processed per second (PPS) metric of Darshini with tshark.

As expected, Darshini performs better in non-persistent mode when compared with persistent mode. Another way of looking at this performance differential is

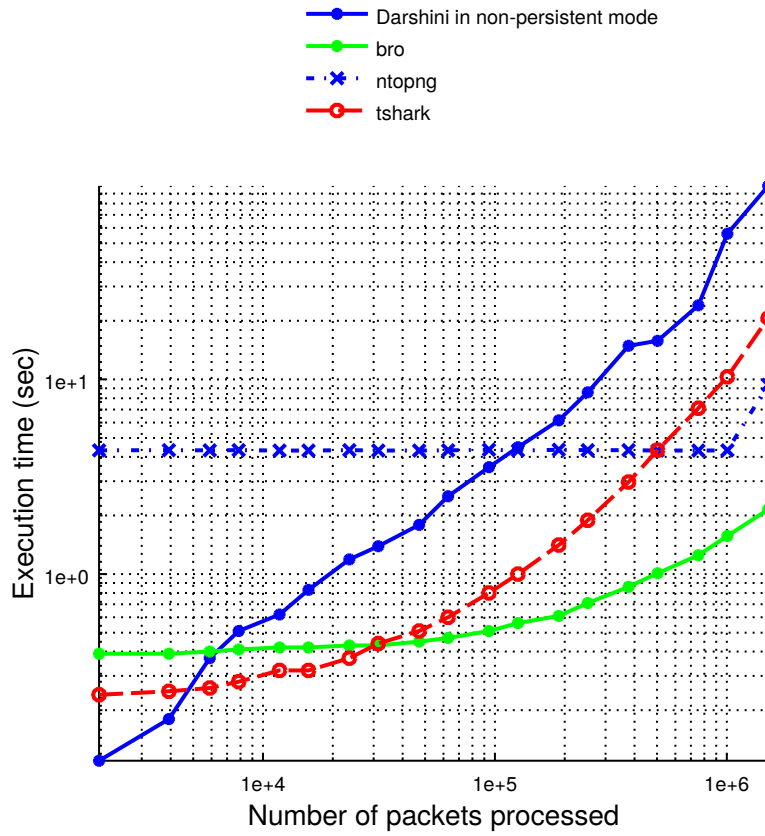
TABLE 5.4: Throughput numbers achievable by Darshini.

frame size	Packets Per Second			Throughput (Mbps)		
	A	B	C	A	B	C
1514 bytes	49,391	51,371	6,638	583	606	78.4
74 bytes	2,30,023	66,157	6,550	189.5	54.5	5.4

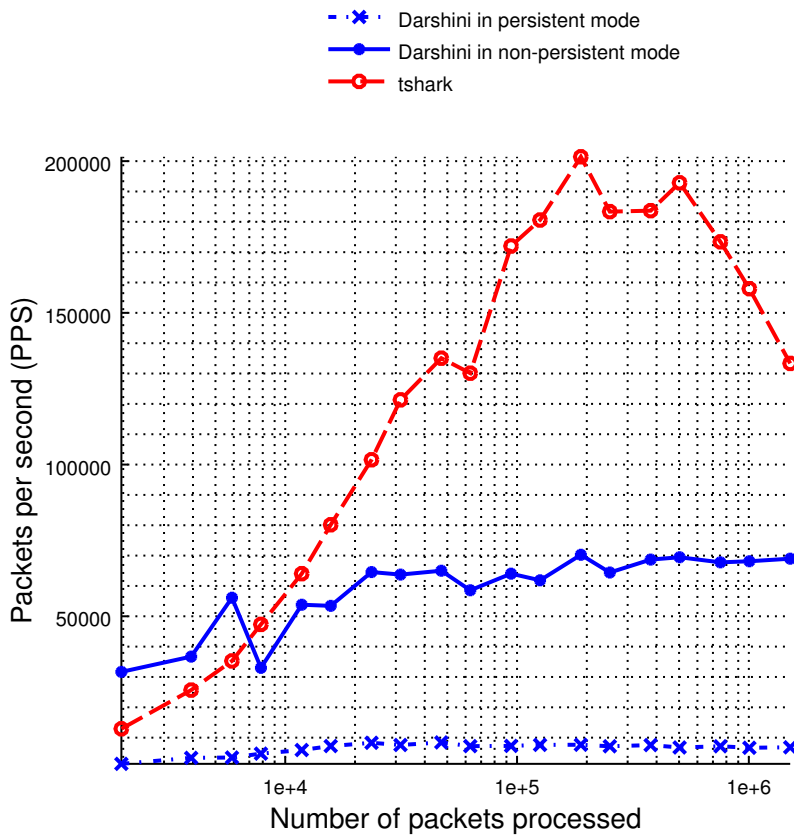
A - tshark

B - Darshini in non-persistent mode

C - Darshini in persistent mode



(A) Execution time



(B) Packet throughput

FIGURE 5.10: A comparison of execution time and packet throughput parameters for different packet parsing tools.

the number of packets processed by the analyzer pipeline vis-à-vis complete application. While analyzer cells consistently process around 51,000 to 66,000 packets per second (PPS) for a range of pcap file sizes, application performance consistently hovers around 6,500 PPS. These performance numbers are independent of packet sizes.

The range of throughput numbers achievable by Darshini are shown in Table 5.4. Since the application and analyzer pipeline performance is packet size invariant, a meaningful performance metric is the packets processed per second (PPS) by Darshini which stands at approximately 66,000 PPS.

5.8 Summary

We propose a concurrent, modular and scalable solution to packet processing. We start with a mathematical formulation of packet processing as a graph embedding problem. We propose a modular, scalable architecture as a heuristic solution to the graph embedding problem. We realize the architecture in our software tool named BITS Darshini with the help of a modular, concurrent architectural element named generic analyzer cell (GAC). The GAC itself has been implemented using software threads to provide the necessary concurrency. Multi-threaded implementation of BITS Darshini is capable of scaling up/down with the availability of processing resources, namely processor and memory resources. A comparison of Darshini with other packet processing tools is available in Table 5.5.

Darshini enables users to select protocols of interest for analysis. The protocols of interest are specified using protocol parse graph. In Darshini, we map the parse graph into an analysis pipeline. Each protocol analysis request from a user launches a custom analyzer pipeline as per the parse graph. Each custom analyzer pipeline is executed in a completely concurrent mode there by taking advantage of the multi-core processor architectures.

The results of protocol analysis are stored in a database (Elastic Search) instance which in turn makes the results data available over REST API service interface. Users can share experiments within Darshini. Darshini facilitates scientific network measurements done in a collaborative manner.

The protocol analyzer pipeline of Darshini is able to perform protocol analysis with a maximum throughput of 606 Mbps. This throughput is sufficient for most offline packet analysis scenarios. In-memory protocol analysis tools have difficulty with analyzing large pcap files; for example, Wireshark has difficulty analyzing pcap files larger than 100MB [31]. Darshini does not have any limitations on the input pcap size; the packet analysis rate of Darshini is independent of the input pcap file size.

TABLE 5.5: A comparison of Darshini with other packet processing tools (as of software versions released up to October, 2017.)

Parameter	BITS Darshini	tshark	Wireshark	ntopng	BROIDS
Maintenance of meta-data	✓	X	X	X	X
Collaboration	share analysis, pcaps den	hid- ←	share pcaps →	share analysis	share logs
Persistence	Elastic Search (ES) DB	X	X	HTML/MySQL/ES	logs
Protocol selectivity for analysis	user-defined parse graph	X	X	only app-layer protocols	BroScript
Concurrency	multi-threaded, multi-core	X	X	X	X
Addition of new protocols	P4 protocol headers	←	←	WSGD ^b / Lua / C	→ C++ and BroScript
Packet filters	BPF ^c for capture filters; ES REST API for display filters	←	←	BPF	→ BroScripts
Parser / Analyzer	parser	both	both	both	both
Live capture	X	✓	✓	✓	✓

^a RRD - Round Robin Database, ^bWSGD - WireShark Generic Dissector, ^cBPF - Berkeley Packet Filter,

Chapter 6

Multi-Modal Transit Scheduler

6.1 Motivation

Public transit networks are the most preferred means of transport in most developing countries. Widespread and efficient usage of public transit networks facilitates cost-effective increase in user mobility. Public transit networks often offer different modes of transport, i.e., flight, train, bus, metro, taxi etc. When a public transit network consists of more than one mode of transport, it is called a *multi-modal transit network*.

Ideally commuters would like to plan their journeys. This journey planning requirement from commuters gives rise to the theoretical problem of searching for a shortest path equivalent on the timetables of the public transport networks. One of the problems plaguing the multi-modal public transit networks is the lack of interactive journey applications which work well in a federated transit system. Existing journey planning applications fall into two categories.

1. Exclusive, operator-centric solutions
2. Third-party, centralized solutions

The first category mostly consists of independent network operators like Emirates airline and Indian Railways [88] who can provide transit through their own network. The second category consists of third-party centralized solutions such

as Euro Railways [118], Google Transit and Expedia. In the centralized solutions, the independent transit network operators would not be able to exercise real-time control on their timetables used in searching for schedules. The timetable changes made by the operators will only be reflected in the search operations after the next round of updates.

6.1.1 Itinerary Search

A passenger using the public transit systems utilizes a series of connections to move from one transit station to another. A strict ordering of the complete set of connections utilized by a passenger during one journey is referred to as an *itinerary*. Passengers would like to have time-efficient, hassle-free itineraries for their journeys. Often, passengers also wish to prefer certain kinds of connections; for example, a passenger can have a preference for train travel over all other modes of transport. Thus any itinerary generated for a passenger must match the preferences indicated by the passenger at the beginning of the itinerary search process.

The job of the interactive journey planning algorithms / applications is to scan through the timetables of potentially multiple multi-modal transit network operators to provide a time-efficient itinerary that matches the user preferences. Interactivity requires itinerary search completion times in a few seconds, preferably less than one second [119].

6.1.2 Operator Control

Independent transit operators often cooperate to provide transit services over wider geographic areas. London transport system's Oyster Card is one such example. Oyster card is accepted by London's bus operators, metro train operators, tram operators and riverbus service operators. The independent operators want to retain control over their transit networks, and timetables, yet cooperate with other transit providers to form federated transit services. Any transit scheduling application designed should be able to provide for operator control over their own transit networks, and timetables.

6.1.3 Solution Approaches

We propose two approaches to solve the problem of interactive journey planner. The first approach is to extend CSA algorithm [5] with two table lookups. The first table lookup accelerates queries for direct connections. Search queries for stations on the same route also get benefit out of the proposed direct connection table (*DCTable*) lookup. The second table lookup connects nearby stations using footpath or any other known means of transport (for example, cabs); this table is named *other means table (OMTable)*. *DCTable* reduces the average response time of the queries, where as *OMTable* adds the multi-modal private transport facilities to known public transport facilities.

In our second approach, we implement a Cell Model-based concurrent solution to the interactive journey planning problem. In the second approach, we propose a decentralized, multi-operator approach to transit network journey planners. In a federated system of transit network operators such as Euro Railways, the independent transit network operators, and passengers would greatly benefit from distributed control over the transit schedules. The independent operators can update their transit schedules as per the local conditions; passengers can then query for feasible travel itineraries as per the current state of the federated transit network. Transit passengers greatly benefit from real-time updates to best possible itineraries on the transit network.

6.2 Definitions

In this section, we define the necessary technical terms for developing the Transit Scheduler application.

Public transport networks contain a set of stations (S) where vehicles stop to pickup or drop-off passengers. These vehicles cover a well-defined sequence of stations; The arrival and departure times of a vehicle at a station is well known. One instance of a vehicle starts at one station, goes through a well-defined sequence of stations and stops at a final station. The vehicle arrives at each station at arrival time (t_{arr}) and departs at the departure time (t_{dep}). The sequence of stations including the first and the last stations along with their

S_1	S_2	\dots	S_n	VehID	
$trip_1$	$(-, t_{dep_1})$	(t_{arr_2}, t_{dep_2})	\dots	$(t_{arr_n}, -)$	ID_1
} Connection (C)					
$trip_2$	$(t_{arr_1}, -)$	(t_{arr_2}, t_{dep_2})	\dots	$(-, t_{dep_n})$	ID_1

FIGURE 6.1: Sample trips $trip_1$ and $trip_2$ undertaken by vehicle identified as ID_1 . The trips $trip_1$ and $trip_2$ together form a loop trip.

respective arrival and departure times is defined as a *trip*. A mathematical formulation of a trip is shown in Figure 6.1.

The arrival and departure times at a station S_i are indicated as (t_{arr_i}, t_{dep_i}) . The arrival time at the starting station and the departure time at the ending station are undefined. The time difference $(t_{dep_i} - t_{arr_i})$ is the waiting time at i^{th} -station [5].

6.2.1 Connections

The timetable of public transit system can be built from a fundamental concept known as a connection. A definition of connection is shown in Equation 6.1.

$$C = (S_{dep}, t_{dep}, S_{arr}, t_{arr}, tr) \quad (6.1)$$

where,

- src = source station of the connection
- dst = destination station of the connection
- t_{dep} = vehicle departure time at the source station
- t_{arr} = vehicle arrival time at the destination station
- tr = a unique identification number

A set of related, concatenating connections form a trip for a vehicle. A set of time-shifted trips form a route. Most public transit systems consist of unique vehicles covering well known routes. Thus we can represent the public transit timetable as consisting of a series of related connections. The theoretical underpinnings of a public transit timetable are detailed in [5] and [120].

In public transport networks vehicles go on a trip, return to the starting station (through another trip) and repeat such a loop many times during one time period (a day / a week / a fortnight); we call such trips as *loop trips*. In such cases, we

consider the onward and return trips as two distinct trips. For a trip covering n stations, we will have $(n-1)$ connections.

$$\begin{aligned} \text{trip} &= C_1 C_2 \dots C_{n-1} & (6.2) \\ \text{where,} & \\ C_i &= (S_i, t_{dep_i}, S_{i+1}, t_{arr_{(i+1)}}, tr) \end{aligned}$$

An interesting variation on loop trips is a *circular trip* where a vehicle starts at station S_1 , goes through a series of stations S_i and terminates the trip at S_1 again. For circular trips, the following timing constraint holds.

$$\begin{aligned} \forall i, j \in [2, n] \text{ and } j > i, \\ t_{dep_1} < t_{arr_i} < t_{dep_i} < t_{arr_j} < t_{dep_j} < t_{arr_1} \end{aligned} \quad (6.3)$$

Two typical events can happen in a public transport system. One, a vehicle makes multiple loop trips covering same sequence of stations with each trip but having different arrival and departure times; Two, between busy stations, multiple vehicles cover the same sequence of stations with each vehicle undertaking multiple trips. *Route* (r) is a set of all trips that cover the exact same sequence of stations and satisfy first-in-first-out (FIFO) property [5].

$$r = \{trip_1, trip_2, \dots, trip_m\}$$

FIFO property guarantees strict time-ordering of all trips belonging to a route. A common sense way of saying this is, "*on a route, waiting never pays off.*"

By definition, all routes of a public transport system are mutually disjoint, i.e., no two routes contain a common trip. Since trips belonging to different routes can have overlapping stations, routes may have a few common stations.

We can define *timetable* for a public transport system as follows [53].

$$\text{Timetable} = (\pi, S, Trips, R, F) \quad (6.4)$$

where,

π = time period of a timetable

S = the set of all stations

$Trips$ = the set of all trips

R = the set of all routes

F = footpath table containing walking times between nearby stations

6.2.2 Connection Array

In the previous section, we have shown a way to transform timetables into elementary connections. We know that each connection has a start time. Based on the start times, we can sort the connections in ascending order and place the resulting connection sequence in an array. This array is called *connection array* (CA). In a connection array, let i and j be the positions of two connections, then $t_{dep}(C_i) \leq t_{dep}(C_j) \forall i < j$.

Timetables of public transport systems tend to be periodic; Often, the time period is a day, a week or a fortnight. We can easily incorporate the periodicity by making the following adjustments to the connection array.

1. Connection array becomes a circular list.
i.e., after last connection in the array, we can pursue the first connection in the connection array to determine the travel itinerary.
2. Arrival and departure times of each connection need to be adjusted based on the number of passes through the CA.

$$t'_{arr} = i \times T_{day} + t_{arr} \quad (6.5)$$

$$t'_{dep} = i \times T_{day} + t_{dep}$$

where, $i = 0, 1, 2, \dots, k$ is the number of passes made through the connection array. T_{day} corresponds to the time elapsed in a day. We can limit k to 3 days even for country-wide timetables.

6.2.3 Feasible Itinerary

An itinerary (I) is made up of a sequence of connections, $I = C_1 C_2 \dots C_h$ with consecutive connections satisfying the condition $S_{arr}(C_i)$ is equal to $S_{dep}(C_{i+1})$. Each of the consecutive connections of an itinerary may belong to the same trip. If two consecutive connections are not part of the same trip, a *transfer* is required at the corresponding station.

Query: $S_1 @ 00 : 10 \rightarrow S_5$

Itinerary:

1. Wait for 5 minutes at S_1 .
2. Take $trip_2 = C_4 C_5 C_6$ from S_1 to S_4 and reach S_4 by 02 : 10.
3. Wait for 30 minutes at S_4 . (transfer from $trip_2$ to $trip_3$)
4. Take C_8 (part of $trip_3$) and reach S_5 at 03 : 15.

FIGURE 6.2: A sample query and the generated itinerary.

A representative query from a user and the itinerary generated using CA are shown in Figure 6.2. The itinerary shown in Figure 6.2 is feasible because the commuter can catch C_8 after C_6 . If C_8 departs anytime before 02 : 10, the itinerary becomes in-feasible.

6.2.4 Transfer Times

In simple connection model, *transfer time* – the time required for alighting a connection of one trip and boarding connection of another trip (*trip transfer*) – has been assumed to be zero. But in reality, transfer times are non-zero. The transfer time is often dependent on connections, station at which transfer occurs and the time of day. The transfer time is zero for a commuter not undertaking the trip transfer at a station. We can represent transfer time as

$$\begin{aligned} t_{tr}(C_i, C_j) &= 0 && \text{if } C_i, C_j \in \text{same trip} \\ &= f(C_i, C_j) && \text{else} \end{aligned} \quad (6.6)$$

We can approximate $f(C_i, C_j)$ as

$$\begin{aligned} f_1(C_i, C_j) &\approx f_2(S_{arr}(C_i), t) \approx f_3(S_{arr}(C_i)) && (6.7) \\ \text{where, } f_1 &: C_i \times C_j \rightarrow T, f_2 : S \times T \rightarrow T, f_3 : S \rightarrow T \end{aligned}$$

Function f_3 is sufficient to model transfer times in most practical scenarios. In the rest of the chapter, we will use function f_3 to model transfer times.

TABLE 6.1: Footpaths table.

S_{start}	S_{end}	t_w
S_i	S_j	t_{w1}
S_j	S_k	t_{w2}
S_i	S_k	$t_{w3} \leq t_{w1} + t_{w2}$

With non-zero transfer times, a transfer C_i to C_j is feasible if and only if

$$\begin{aligned}
 S_{arr}(C_i) &= S_{dep}(C_j) \text{ and} & (6.8) \\
 t_{dep}(C_j) &\geq t_{arr}(C_i) + t_{tr}(C_i, C_j) \\
 &\approx t_{arr}(C_i) + t_{tr}(S_{arr}(C_i))
 \end{aligned}$$

6.2.5 Footpaths and Other Means

In case two public transport stations are nearby, the passengers can alight a connection at one station, walk to the nearby station on footpath and board a new connection. For all the nearby stations of a public transport network, we can maintain a table of walk times. A sample set of entries are shown in Table 6.1. S_{start} denotes starting station, S_{end} denotes ending station and t_w indicates walk time.

All entries are transitively closed which enables a direct lookup of all possible walks between nearby stations. If the entries are not transitively closed, we will be forced to run shortest path algorithm on the table for each query, an expensive proposition.

No single public transport network provides best any-station to any-station connectivity at all times. Our footpath table is a more effective way of reaching the nearby stations. We can segregate all known means of transport into public transport system with timetable and everything else (*other means*). Popular other means could be footpaths, autos, taxis, car pools or a helpful drop by a

TABLE 6.2: Other means table.

S_{start}	S_{end}	t_{omt}	Details
S_i	S_j	10 min	walk
S_i	S_k	30 min	cab

friend. The transport by other means can be summarized into a table named *other means table* (OMTable). Entries in OMTable will be a tuple of the form $(S_i, S_j, \text{travel time}, \text{details})$. The format of OMTable is given in Table 6.2.

Connection array is a very efficient representation for aperiodic connections. Most timetables have high-frequency trips between popular stations, say a bus every 5 minutes between S_i and S_j . One such timetable entry leads to 288 connections in the connection array. Such routes have been called as guidebook routes or transfer patterns [48, 54, 55]. We can efficiently represent such cases in OMTable.

6.2.6 Direct Connections

CSA gives fast query response even on country-wide multi-mode timetables [14]. We reduce the queries times even further by storing answers to queries of direct connections that can be answered directly from timetable. Such direct connections are the norm in long-distance bus networks. For these scenarios, direct lookup is optimal.

We copy all such direct connections cases from timetables and place them in *DCTable*, short for direct connections table. Entries in the DCTable are stored as connection tuples. Any incoming query is first checked against a match in DCTable; Upon a match in the DCTable, all possible direct connections/itineraries between two stations specified in the query are returned. If an answer is not found, then CSA is run. DCTable's role can be augmented as a cache for popular queries.

6.3 Connection Scan Algorithm (CSA)

In this section, we give an outline of algorithm called a Connection Scan Algorithm (CSA) that operates on a connection array to produce travel itinerary [56]. A basic version of the CSA is given in Algorithm 6.1. Algorithm 6.1 requires connection array (CA), starting station (S_1), start time (t_s), ending station (S_2), and station array (SA) as inputs. If possible, the Algorithm 6.1 produces a feasible itinerary (I). Algorithms 6.1 and 6.2 use a few temporary variables

during their operation. These temporary variables are as follows: IC is an array to hold incoming connections for each station; d is an array to hold the earliest arrival time for each station; C_i is a temporary connection variable to hold the connection under consideration.

Algorithm 6.1 Connection Scan Algorithm (CSA)

Require: CA, S_1, t_s, S_2, SA

Ensure: I

```

1:                                     ▶ Temporary variables:  $IC, d, C_i$ 
2: for all  $s_i \in SA$  do
3:    $IC[s_i] \leftarrow \text{null}$ 
4:    $d[s_i] \leftarrow \infty$ 
5: end for
6:                                     ▶ Compute itinerary
7: for all  $C_i \in CA$  do
8:   if  $t_{dep}(C_i) > d[S_{dep}(C_i)]$  and
      $t_{arr}(C_i) < d[S_{arr}(C_i)]$  then
9:      $d[S_{arr}(C_i)] \leftarrow t_{arr}(C_i)$ 
10:     $IC[S_{arr}(C_i)] \leftarrow C_i$ 
11:   end if
12: end for
13:  $I \leftarrow \text{ITINERARY}(IC)$ 
14: return  $I$ 

```

Algorithm 6.2 Fetch itinerary through backtracking (ITINERARY)

Require: IC

Ensure: I

```

1:                                     ▶ Temporary variable:  $C_i$ 
2:  $I \leftarrow \text{null}$ 
3:  $C_i \leftarrow IC[S_2]$ 
4: while  $C_i$  do                                     ▶ valid connection
5:    $I.\text{PREPEND}(C)$                                    ▶ add at the beginning
6:    $C_i \leftarrow IC[S_{dep}(C_i)]$ 
7: end while
8: return  $I$ 

```

Algorithm 6.1 operates as follows. The arrays IC and d are initialized. The for loop given in the lines 7–12 iterates through each connection and tries to settle the arrival station of a connection. If the relaxation function (d) of arrival station can be modified, the corresponding entries are updated in the incoming connections array (IC). After processing all the connections, Algorithm 6.1 uses Algorithm 6.2 to extract an optimal itinerary from IC array.

6.3.1 Complexity Analysis

The initialization of station array in the lines 2 – 5 of Algorithm 6.1 always requires n_{SA} iterations. The for loop given in the lines 7–12 iterates through each connection of connection array; this for loop always requires n_{CA} iterations. The Algorithm 6.2 has a while loop which at best requires one iteration and at worst requires n_{SA} iterations. Thus CSA has the running time algorithmic complexity bounds of $\Omega(n_{SA} + n_{CA})$ and $O(n_{SA} + n_{CA})$.

6.4 t-CSA Algorithm

The CSA outlined in Algorithm 6.1 have the following limitations.

- The algorithm assumes the zero transfer times.
- The algorithm assumes that an itinerary can be found on or before we reach the end of connection array (CA).
- End of CA is also the stop condition for the loop given in the lines 7 – 12 of Algorithm 6.1.

We can redesign the basic CSA to overcome these three limitations.

An enhanced version of CSA (named, t -CSA) that incorporates modifications is shown in Algorithm 6.3. Algorithm 6.3 builds on Algorithm 6.1. Over and above the inputs required by Algorithm 6.1, Algorithm 6.3 requires four additional inputs. These additional inputs are: the maximum number of travel days (n), station transfer times array (TT), direct connections table ($DCTable$) and other means table ($OMTable$). C_j , C_{last} , itr , n_{ca} , t_{dep}, t_{arr} and $settled$ are the newly introduced temporary variables. C_j denotes the first connection in the connection array with a starting time $t_{dep}(C_j) > t_s$. C_{last} holds the last connection in the connection scan array. itr specifies the number of times the connection array needs to be scanned for creating a feasible itinerary. n_{ca} , t_{arr} and t_{dep} are the temporary variables used to enhance readability. $settled$ indicates a change in earliest arrival estimate at a station.

The first major addition to Algorithm 6.3 over Algorithm 6.1 is the $DCTable$ code given in lines 2 – 5. $DCTable$ is consulted first; only in case the query fails on $DCTable$, will the CSA be run. The function call on line 14 returns C_j . In abstract

Algorithm 6.3 An enhanced CSA (t-CSA)**Require:** $CA, S_1, t_s, S_2, SA, n, TT, DCTable, OMTable$ **Ensure:** I

```

1:   ▶ Temporary variables:  $IC, d, C_i, C_j, C_{last}, itr, n_{ca}, t_{dep}, t_{arr}, settled, C_{temp}$ 
2:   ▶ try direct connections first
3: if  $I \leftarrow \text{SEARCHDCT}(DCTable, S_1, t_s, S_2)$  then
4:   return  $I$ 
5: end if
6:   ▶ CSA implementation
7:   ▶ Compute number of iterations
8:  $C_{last} \leftarrow CA[CA.length - 1]$ 
9:  $n_{ca} \leftarrow t_{dep}(C_{last}) \bmod 1440$ 
10:  $itr \leftarrow n \bmod n_{ca}$ 
11: Initialize  $IC$  and  $d$  as per lines 2-5 of Algorithm-6.1
12:   ▶ Pick first connection with  $t_{dep}(C) > t_s$ 
13:
14:  $C_j \leftarrow \text{CONNSEARCH}(CA, t_s)$ 
15:
16:   ▶ iterate through  $CA$  for sufficient number of days
17: for all  $day \leftarrow 1, 2, \dots, (itr + 1)$  do
18:   if  $day == 2$  then
19:      $C_j \leftarrow C_0$ 
20:   end if
21:   for all  $C_i \in CA \ni C_i \geq C_j$  do
22:      $settled \leftarrow false$ 
23:     ▶ set correct transfer time
24:     if  $tr(C_i) == tr(IC[S_{dep}(C_i)])$  then
25:        $t_{tr} \leftarrow 0$ 
26:     else
27:        $t_{tr} \leftarrow TT[S_{dep}(C_i)]$ 
28:     end if
29:     ▶ offset times and settle a station
30:      $t_{dep} \leftarrow t_{dep}(C_i) + day \times 1440$ 
31:      $t_{arr} \leftarrow t_{arr}(C_i) + day \times 1440$ 
32:     if  $t_{dep} > d[S_{dep}(C_i)] + t_{tr}$  and
        $t_{arr} < d[S_{arr}(C_i)]$  then
33:        $settled \leftarrow true$ 
34:        $d[S_{arr}(C_i)] \leftarrow t_{arr}(C_i)$ 
35:        $IC[S_{arr}(C_i)] \leftarrow C_i$ 
36:     end if
37:   ▶ continued...

```

Algorithm 6.3 An enhanced CSA (t-CSA) continued...

```

38:                                     ▶ use OMTable to settle nearby stations
39:     if settled then
40:         for all  $S_{arr}(C_i), S_i \in OMTable$  do
41:             if  $d[S_j] > d[S_{arr}(C_i)] + t_{omt}[S_{arr}(C_i), S_j]$  then
42:                  $d[S_j] \leftarrow d[S_{arr}(C_i)] + t_{omt}[S_{arr}(C_i), S_j]$ 
43:                  $C_{temp} \leftarrow (S_{arr}(C_i), t_{arr}(C_i), S_j, t_{arr}(C_i) + t_{omt}[S_{arr}(C_i), S_j], om)$ 
44:                  $IC[S_j] \leftarrow C_{temp}$ 
45:             end if
46:         end for
47:     end if
48:     if  $t_{dep} > d[S_2]$  then
49:         break search
50:     end if
51: end for
52: end for
53:  $I \leftarrow ITINERARY(IC)$ 
54: return  $I$ 

```

terms, CONNSEARCH performs a binary search on connection array to find the connection C_j that satisfies the condition.

$$\min t_{dep}(C_j) - t_s$$

With Constraints:

$$t_{dep}(C_j) - t_s > 0$$

$$C_j \in CA$$

The code block in lines 17 – 52 iterates through connection array required number of times, examines all eligible connections and tries to settle stations whenever relaxation is possible. The code in lines 24 – 28 considers realistic station transfer times. Another feature addition is *settling* the nearby stations of a recently *settled* station by using the entries of other means table (*OMTable*). The corresponding code is in lines 39 – 47 of Algorithm 6.3. Early termination condition on line 48 helps us discard all connections whose departure times are greater than the current estimate of the earliest arrival time at the destination station.

TABLE 6.3: Algorithmic complexity analysis of CSA vis-à-vis t-CSA algorithms.

Complexity Bound	CSA	t-CSA
Lower Bound	$\Omega(n_{SA} + n_{CA})$	$\Omega(1)$
Upper Bound	$O(n_{SA} + n_{CA})$	$O(n_{SA} + \log_2 n_{DCT} + n_{CA} \times n_{OMTable})$

6.4.1 Complexity Analysis

The t-CSA algorithm adds direct connections table (DCT), connection array and other means table ($OMTable$). We search the DCT using binary search which has the running time algorithmic complexity bounds of $\Omega(1)$ and $O(\log_2 n_{DCT})$. Initialization of IC on the line 11 is similar to the basic CSA algorithm. Similarly, the connection search mentioned on the line 14 is done using binary search which has the running time algorithmic complexity bounds of $\Omega(1)$ and $O(\log_2 n_{CA})$. The nested loops between lines 17 – 52 of Algorithm 6.3 has the running time algorithmic complexity bounds of $\Omega(1)$ and $O((days \times n_{CA} - \log_2 n_{CA}) \times n_{OMTable})$. Thus t-CSA has the running time algorithmic complexity bounds of $\Omega(1)$ and $O(n_{SA} + \log_2 n_{DCT} + \log_2 n_{CA} + (days \times n_{CA} - \log_2 n_{CA}) \times n_{OMTable})$.

A fair comparison of the algorithmic complexity of CSA and t-CSA requires that we remove the optimization for the connection search in CA . We also need to remove the iteration over multiple days of schedule. If we remove these two additions, then we can see the modification of run time complexity due to addition of DCT and $OMTable$. The modified run time algorithmic complexity bounds of t-CSA are: $\Omega(1)$ and $O(n_{SA} + \log_2 n_{DCT} + n_{CA} \times n_{OMTable})$. A comparison of the algorithmic complexity bounds of CSA vis-à-vis t-CSA is shown in Table 6.3. The enhanced CSA algorithm improves the lower bound but adds $\log_2 n_{DCT}$ to the upper bound. The change is desirable because of the added benefit from drastic reduction in search completion times experienced by queries satisfied from the direct connections table. The enhanced CSA algorithm also includes other means of transport (using $OMTable$) along the public transit schedules used by the CSA algorithm. With the inclusion of $OMTable$, we increase the feasible number of itineraries for a given itinerary search request.

6.5 Cell Model-based Solution Approach

In this section, we adopt the Cell Model for developing the Transit Scheduler application.

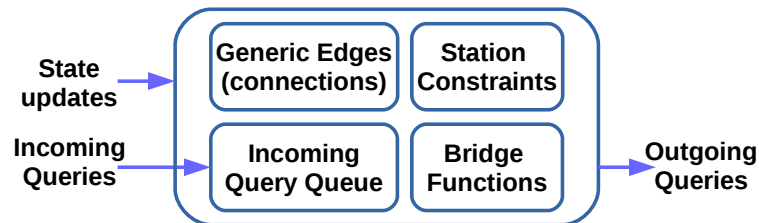


FIGURE 6.3: A behavioural model for a multi-modal transit station. The station constraints, bridge functions, and connections together constitute the state and behaviour of a station process.

6.5.1 Transit Station

In our approach, each transport station of the transit network exists independently as a stand alone entity with internal state and behaviours. We use behavioural view to represent the transit stations. We model all the stations as independent nodes that help search the timetables to answer user queries on itineraries. We find that all the stations need to go through the same algorithmic steps in answering user queries. The only difference is in terms of the station constraints, and the connections that are available to each of the stations. Thus we can represent a station using the abstract schematic shown in Figure 6.3.

Here, each station entity has internal state that can be updated in real-time in a distributed manner by a transit operator. Transit operator controls the state of a station node in order to restrict the kind of itineraries that can be formed using that particular station as a participating node.

6.5.2 Transit Connection

Most of the vehicle routing, and transit routing algorithms use graph theory notions of weighted, and directed graphs for network representation, and query processing [14]. In these approaches, a node is a passive junction that connects multiple edges. Connections themselves may be weighted, directed or

both. Some transit routing algorithms such as CSA [5], transfer patterns [121] use time-varying connections to model the transit networks. In CSA, transit edges are represented as connections which are nothing but an equivalent form of time-varying edge representation. In transfer pattern approach, the connections are stored as a compressed data structure. Both the approaches are based on the availability of a time-varying connection at certain (periodic) times as described by Equation 6.9.

$$cost(edge) = \begin{cases} C_i & \text{at } t = t_i, \\ \infty & \forall t \neq t_i. \end{cases} \quad (6.9)$$

In Equation 6.9, C_i represents the cost of using the connection at time t_i . In practice, C_i value depends on the connection available at that time. Thus we can represent the cost of using a connection as:

$$cost(C) = \begin{cases} t_{arr} - t_{dep} & \text{when } t = t_{dep}, \\ \infty & \forall t \neq t_{dep}. \end{cases} \quad (6.10)$$

In Equation 6.9, we can understand the condition $t = t_i$ as a *constraint* on the connection. If the constraint is not satisfied, the connection become unusable. Similarly in Equation 6.10, $t = t_{dep}$ is a constraint on the connection; if that constraint is satisfied, the cost of using the connection is $t_{arr} - t_{dep}$.

We generalize the constraint on a connection to be dependent on many environment variables. Without loss of generality, we can say that all the connections of a network are dependent on two environment variables *time* (t) and *mode of transport* (m). Our definition of the cost function for a connection is shown in Table 6.4. The cost function for a connection has also been explained in Equations 4.6 and 4.7.

6.5.3 Path Computation

In transit scheduling, we compute itineraries for a given user query. Since each itinerary is an ordered set of connections, an itinerary is equivalent to a path on a network. For multimodal transit search queries, it is common to specify

TABLE 6.4: The cost function of a connection which is dependent on on time and mode of transport environment variables.

time	mode	cost
t_1	m_1	$f(t_1, m_1)$
t_2	m_1	$f(t_1, m_1)$
...		
t_i	m_j	$f(t_i, m_j)$
all other cases		∞

TABLE 6.5: Connection constraints, path constraints and their compatibility

Connection Constraints	Path Constraints	Compatibility
flight	any mode	yes
flight	bus	no
train	bus or train arrival before 5PM	yes only if computed arrival time < 5PM
train	bus-flight-bus	no

the conditions that an itinerary needs to satisfy. The conditions imposed on itineraries effectively become conditions on the network paths. Conditions or constraints imposed on the computed paths are called *path constraints*. Effectively, each user query comes with a set of path constraints. For example, the paths computed might only have to use air connections. Such conditions put on path computations can be understood as path constraints. We incorporate user preferences into itinerary search as path constraints. A path can be extended using a connection only if the constraints of the selected connection match with the existing constraints on the path.

Let there be two neighbouring nodes (v_a, v_b) connected by a connection e_j . Let $path_i$ be the path computed from *source* to v_a ; the cost of $path_i$ is denoted by $cost(path_i)$. Let $path_{i+1}$ be the the extension of $path_i$ with the connection e_j . Let us denote the cost of $path_{i+1}$ as $cost(path_{i+1})$. Then the path cost computation can be represented as:

$$cost(path_{i+1}) = cost(path_i) + cost(e_j) \quad (6.11)$$

The computed path cost for $cost(path_{i+1})$ given in Equation 6.11 becomes finite only if the connection constraints of e_j are compatible with the path constraints

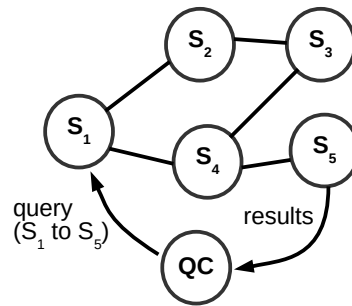


FIGURE 6.4: The Cell-view of network. Each station is represented by a cell, and the cells are connected as per the underlying connectivity among the stations.

of $path_i$. Otherwise, the computed path cost for $cost(path_{i+1})$ becomes infinite. A few sample connection and path constraints are shown in Table 6.5.

6.5.4 Bridge Functions

In transit scheduling, passengers can change vehicles at intermediate stations. Thus we need to consider time required to alight one vehicle, and board another vehicle at a station (called *transfer time*). It is impractical to schedule a connecting journey if the passenger does not get enough time to transfer vehicles. The transfer time is a function of transit station. We propose a generic approach to consider such station-specific constraints. We propose a *bridge function* that considers the suitability of a connection to extend a given path by utilizing station-specific local constraints (*station constraints*). Bridge functions give station manager ability to control traffic flow via their own station. Bridge functions are similar in nature to cost functions of a decorated edge. Even bridge functions have to satisfy the constraint compatibility requirements during path cost computations.

6.5.5 Itinerary Creation

In the previous subsections, we have looked at the adoption of the Cell Model to the itinerary search problem. In this subsection, we connect the co-operating transit stations to form a virtual transit network. Two stations are deemed to be

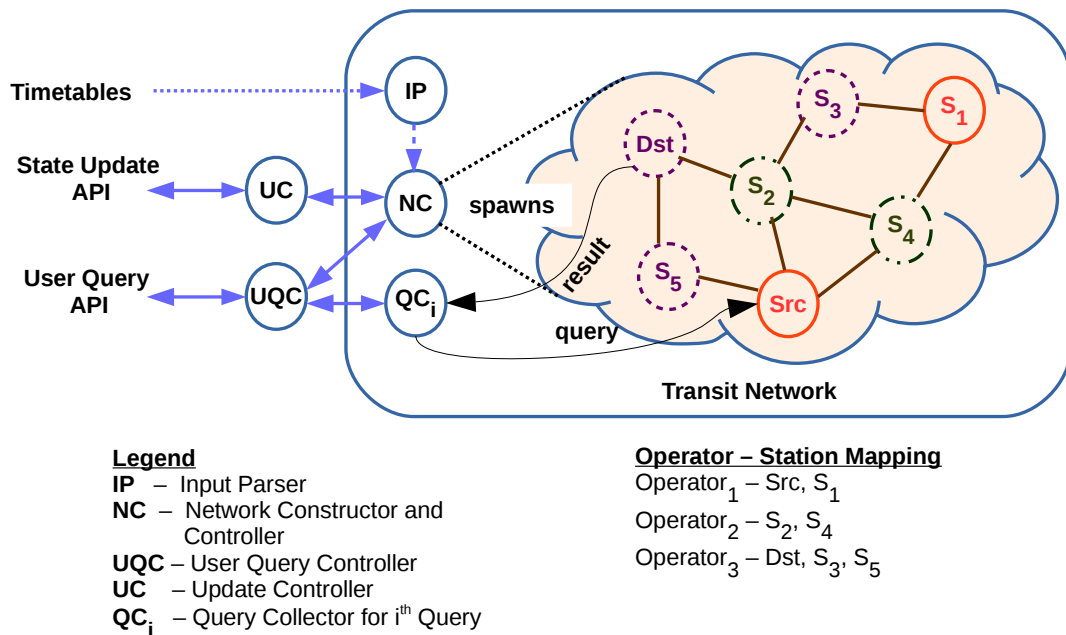


FIGURE 6.5: Architecture of the Transit Scheduler application.

cooperating if both the stations participate in a connection. The connections between co-operating stations help propagate the itinerary search queries across the transit network.

A representative 5-node network with a process for each station is shown in Figure 6.4. Apart from the transit nodes, an auxiliary node named query collector (QC) exists. The QC is responsible for passing an incoming query to the source station node, and collating all the itineraries coming out of the transit network. In Figure 6.4, a sample query from station S_1 to station S_5 is shown. As can be seen from the example given, a query always starts at the source node, propagates through the network, and exits the transit network at the destination node.

Since the query explores all the possible shortest paths to the destination, the query results also contain the shortest path results from source station to all other stations in the network. Thus when we query the transit network, we get one source to all destinations kind of profile answer at no extra cost.

6.6 Cell Model Implementation

6.6.1 Architecture

Our Transit Scheduler application is implemented using Elixir - a functional, concurrent programming language [108]. Elixir + Erlang Open Telecom Platform (OTP) has given us the advantage of implementing station entities using Actor concurrency model. The architecture of transit scheduler application is shown in Figure 6.5.

The Elixir modules that form the base for our application are:

Input Parser (IP) that reads station-specific local variable values, and connection information from dataset files.

Network Constructor, and Controller (NC) that spawns a separate Station process for each station, and initializes the spawned stations with data; NC also maintains a registry of process identifiers for all transit station processes.

Station process is a node of the Cell Model that uses behavioural view. A station process maintains its state using a finite state machine (FSM) based on local variable values, and stores its outgoing connections data. Each station process uses an internal data structure to represent all the local information, and also selects from a set of predefined functions for manipulating any dependent local variables.

Update Controller (UC) that performs create, read, update, and delete (CRUD) operations on a station state. For example, an authorized transport manager may change the congestion level at a station which is reflected in transfer times,

TABLE 6.6: Query API URL strings

Module	Query Type	Query String
UQC	itinerary search	baseURL/api/search
UC	station schedule (fetch / update)	baseURL/api/station/schedule
UC	station create	baseURL/api/station/create
UC	station state (fetch / update)	baseURL/api/station/state

or mark it as having disturbance which renders the station unavailable for a period of time. UC helps in incorporating the effects of accident / incident related events into transit stations.

Query Collector (QC) that initiates query processing at source transit station, and collects query results from destination station. One dedicated query collector exists for each user query. QC also filters the itinerary result set based on user preferences.

User Query Controller (UQC) that receives user queries for itinerary search. UQC spawns a new QC for each incoming query, and hands over the job of query processing to the newly spawned QC.

Three kinds of queries are supported by the system. They are: itinerary search, station schedule - both fetch, and update, and station update - create a new station, update a station, fetch details of a station. The system is accessible via REST API URLs shown in the Table 6.6.

6.6.2 Station as Cell (Actor)

Elixir implements Actor model of concurrency, and refers to each instantiated Actor as a process. Each station process utilizes three distinct inputs. The first input is a set of outgoing connections represented in the format (vehicleID, source station, destination station, departure time, arrival time, mode of transport). The second input is a set of other means table (OMTable) connections available due to footpath, and private transport connections between stations. These connections in OMTable are described in [120]. The third input is a set of local variables, collectively called *station state*, that control the station behaviour. We use three station-specific local variables, namely, *congestion*, *delay*, and *disturbance*. The congestion is a three-valued variable holding low, medium, and high values; congestion variable indicates the level of congestion at the local station. Delay indicates the necessary transfer time to change vehicles at a station; As expected in a real-life scenario, we make the effective delay a function of the congestion level. Disturbance is a binary variable which indicates any local disruptions that stop the flow of vehicles via a transport station. If the local disturbance is set to yes, then all the incoming queries of a station are discarded. Effectively, the station becomes unavailable.

TABLE 6.7: Summary of timetables obtained from the Indian public transport networks.

Name of Network	Mode of Transport	#Stations	#Connections
Indian Railways (IR)	train	2136	44876
Indian Flights (IFlights)	flight	80	2501
Long Distance Buses (LDB)	bus	147	12219
Complete Network (India)	multi-mode	2264	59555

6.6.3 User Preference

User queries for itinerary search can contain transit mode specification. We utilize the user preference to filter the itinerary result set to discard undesirable results. The itineraries are filtered at QC. An itinerary result becomes undesirable if it contains any transit mode not preferred by the user. For example, a user choosing flight mode will have all the itineraries containing non-airline connections filtered out.

6.7 t-CSA Experimental Results

6.7.1 Dataset

We utilize transport data available from Indian public transport networks. The characteristics of the networks selected are summarized in Table 6.7. The #Stations and #Connections columns indicate the number of stations and connections respectively. The train data has been obtained from Indian Railways timetable [88]. This railway timetable network is the largest Indian public transport network utilized by us. The flight data has been generated from the domestic flight schedules released by Directorate General of Civil Aviation (DGCA), India [122]. The bus data has been obtained from numerous public and private bus transport operators (for one such example, namely GSRTC, see [123]). Apart from 59,555 regular transit connections, the dataset contains 151 other means (i.e., footpath, and private transport) connections. The dataset has been derived from the transit network schedules of public transport network operators in India.

6.7.2 Itinerary Generation Times

Algorithms 6.1, 6.2 and 6.3 have been coded in Python2.7 programming language and tested on a commodity laptop. The laptop has 2-core 64-bit Intel i5-2430M processor with 256KB of L2 cache and 3MB of L3 cache; the operating system is 64-bit Ubuntu. We run queries on complete Indian network (has 2264 stations and 59555 connections). We randomly generate 1000 queries with randomly chosen source, destination pairs and a random start time. Our Python2.7 implementation of Algorithm 6.1 gives an average query response time 0.5ms. Algorithm 6.3 gives an average query response time of 0.7ms. The direct connection queries on *DCTable* execute very fast (worst case time: $1\mu s$; average time: $0.5\mu s$). These numbers compare favorably with average response time of 1.8ms for London metropolitan data reported in [5].

6.8 Cell Model Experimental Results

6.8.1 Implementation Platform

The transit scheduler has been implemented using Elixir language v1.4 compiled to run on Erlang Open Telecom Platform (Erlang/OTP) v19.0. All the tests have been run on a computer with Intel core i5 2GHz processor, and 8GB RAM running Ubuntu 16.04 x86_64 operating system. As discussed in Section 6.6, our application incorporates user mode preferences, updates the state of a station using API, and outputs a list of itineraries for user queries.

6.8.2 Itinerary Generation Times

We conducted a set of randomized trials to study the time response characteristics, and concurrent characteristics of Transit Scheduler. Because of the inherent concurrent nature of the architecture as described in Section 6.6.1, the transit scheduler can concurrently propagate itinerary search query across the transit network. We issue one itinerary search query at a time, and measure the response time. Here, response time of a query is defined as the time taken to collect fifteen possible itinerary results from the network. We implement a time

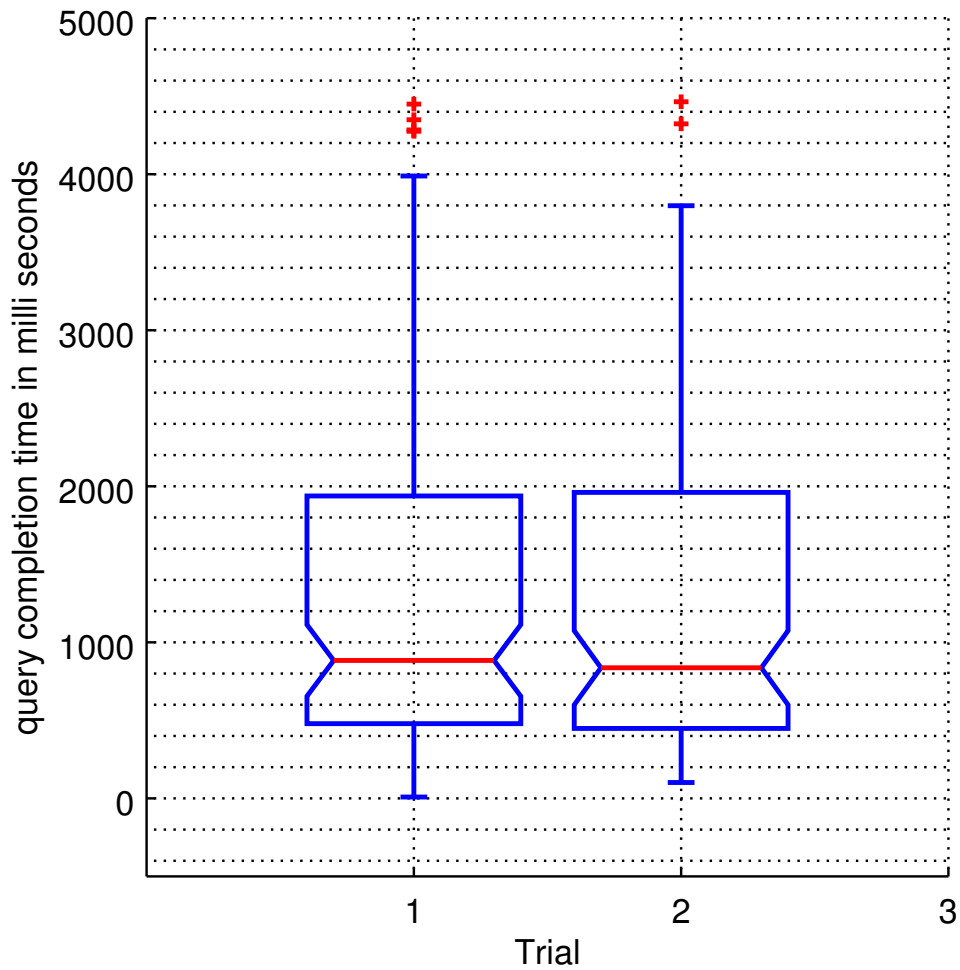


FIGURE 6.6: Box plot on itinerary search times for two trials. In each trial, we execute 100 random queries.

out of 30 seconds to avoid deadlock for queries with less than fifteen feasible answers. A statistical analysis of the query response times for 100 queries results in a query response time distribution with median of 0.9 seconds, standard deviation of 4.1 seconds, skewness of 3.81, and kurtosis of 19.75. We perform multiple randomized itinerary search trials on the transit scheduler application. We show results of two randomized trials in Figure 6.6 using box plot.

For one of the two randomized trials, we also collected the timestamps of all the itineraries in the itinerary result set. We then sorted the queries in the ascending order of their completion time. Figure 6.7 illustrates the time at which different itineraries of the result set have been collected. A dotted line marks the observed median of 823 milli seconds for this instance of randomized trial. Figure 6.7 shows a right tailed distribution with heavy preference for low execution

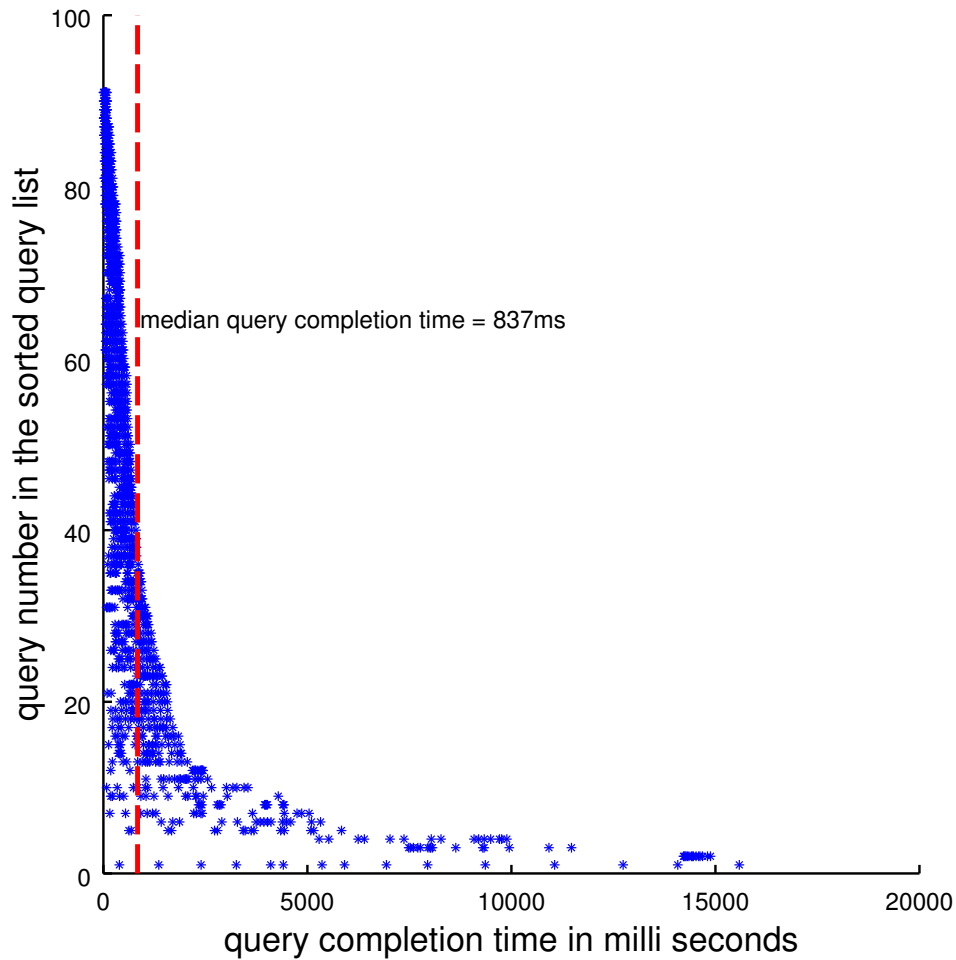


FIGURE 6.7: Query processing times for all the itineraries of the itinerary search results set.

times.

The above response time characteristics makes the system suitable for interactive transit scheduling on the public transport networks such as those of India.

6.9 Summary

We use connection as a building block to model timetables of public transport networks. We propose t-CSA which adds direct connection table (*DCTable*) and other means table (*OMTable*) to connection scan algorithm. *DCTable* stores itinerary details of stations that are directly connected through a trip. *OMTable*

stores details of alternative (footpath and miscellaneous modes of transport) connections between nearby stations. We find that t-CSA extends the capabilities of CSA significantly.

We also demonstrate the implementation of extended CSA on timetables of Indian public transport networks. Our t-CSA eliminates the need to run CSA on direct connection queries. Since direct connection queries are the bulk of queries on timetables, t-CSA significantly accelerates interactive journey planner applications.

We make two original contributions to the modeling of transit networks. Our first contribution is the application of the Cell Model to the itinerary search problem. We utilize the behavioural view of the Cell Model for transit stations and the functional view of the Cell Model for the transit connections. We utilize the concept of constraint to provide a generic conceptual model for station-specific and connection-specific conditions.

Our second contribution is the application of Actor concurrency model to implement the Cell Model-based solution. We develop a Transit Scheduler application that is a native concurrent solution to the itinerary search problem. We represent each station as an Actor of the Actor concurrency model. Actors can encapsulate complex behaviours, and have dynamic, targeted messaging capability. We also implement a real-time state update capability for all the stations. Our approach allows independent transit operators to control the transit stations owned by them.

Our solution allows multiple network operators to control the transit network in real-time. We test our implementation on air, rail, and bus transit networks of India. When run on a laptop computer with Intel core i5 processor, and 8GB RAM, our application enables an interactive concurrent itinerary search on the public transit timetables of India.

The transit scheduler application we built incorporates user preferences in creating itineraries. All the feasible itineraries results are collected at query collector (QC) and are filtered for user preferences. Only the itineraries that satisfy the user preferences are forwarded to the user.

Chapter 7

Conclusion

The last two decades have seen an explosion of interest in the networked systems. Researchers have proposed network models such as random graphs, scale-free networks, power-law networks to understand the statistical properties of networks. These traditional network models represent a single-type of relation between the networked entities. Researchers have extended the network models from single relation to multiple relations. The approach taken by the researchers is to model multiple relations as multiple dimensions with each dimension having a network defined by a single relation.

The consensus multidimensional network models have nodes, connections and dimensions/layers as three representative elements to model real-world entities and their multiple-relations. Multidimensional network models are useful for research on social networks, multi-modal transit networks and computer networks.

7.1 Structural View of Network

The existing network models adhere to the notion of vertices and edges proposed in the graph theory. Emphasis of the existing network models is on the analysis of the network structure. The widely accepted network characterization metrics such as degree distributions, betweenness refer to the structure of a network. In the thesis, the representation of the network using the graph

theoretic assumptions and the study of derivative properties is called structural view of a network.

Our initial research contribution is in the structural analysis of multidimensional networks in the domains of social networks, data networks and multi-modal transit networks. We start our study of multidimensional networks with data collection and analysis. We perform statistical analysis of the properties of broadband network connections obtained using a distributed network measurement tool named network diagnostic test (NDT). We collect the transit timetable data from the public transit service providers of India. We represent the transit timetables as multidimensional networks. In the process, we build on the previous work and also consider the shared transit vehicles. We collect the data on the Internet relay chat (IRC) community and study the multidimensional structure of the online chat communities.

Our work on the structural analysis of multidimensional networks made us aware of some of limitations of the structural view of a network. Emphasis on structural view of a network leads to significant limitations in the network models. For example, the existing network models of multidimensional networks do not adequately address the following aspects of network modeling.

- Nodes and connections are modeled as passive entities.
- Hierarchy is an integral part of large scale networks, but this aspect does not receive enough attention.
- There is a heavy emphasis on the structural view of the network with little attention paid to functional and behavioural views.
- The specification and semantics of the interactions between the participants is not clearly specified.

To overcome the above mentioned limitations, we propose the Cell Model – a new network model for multidimensional networks.

7.2 The Cell Model

The Cell Model enriches the network participants with structure-function-behavior (SFB) views; such a combination of three views provides a chance for better modeling of network participants. The interactions between all the participants

of a network are represented in the structural view of a network. The nodes and connections are active, i.e., they can perform computation. The computational models of nodes and connections are expressed using either the functional or the behavioural views.

The Cell Model gives node-like prominence to relations between nodes which are modeled as connections. More importantly, multiple relations between any two nodes are modeled as one single connection, thus leading to a notational consistency. The Cell Model of a network is built using four axioms.

Message Exchange All network interactions happen via message exchanges.

Activeness All the participants (nodes and connections) are active. A participant can generate, terminate, transform or forward a message.

Equivalence Two models of a participant are equivalent if both have equivalent Finite State Machines (FSM), i.e., both the FSM's are observational equivalent.

Hierarchy A node can represent a sub-network. The representative node must satisfy the equivalence axiom with the sub-network being represented.

7.3 Applications

We model two research problems, one from protocol analysis and another from multi-modal public transit networks. We provide solutions to both the research problems using the Cell Model.

We express the protocol analysis as embedding of a protocol parse graph into a protocol analysis pipeline. The problem is \mathcal{NP} -hard; we chose a heuristic embedding scheme in our implementation. We implement the protocol analysis pipeline in a software named Darshini. Darshini performs analysis of the selected protocols, provides for configurable tradeoffs between memory consumption and execution time, and persists the analysis results into a database.

We then present our work on the multi-modal public transit networks. We extend the connection scan algorithm (CSA) for direct connections and personalized transport services; we call our extension as t-CSA algorithm. We redefine

the mathematical representation of the itinerary search problem in terms of the Cell Model. We implement the Cell Model-based solution for the public transit networks of India. The implementations of t-CSA algorithm and the Cell Model-based solution search the timetables of the public transit networks of India; both provide the itinerary search results in less than a second.

7.4 Summary

Based on our research experience in the multidimensional networks area, we draw the following conclusions.

1. Active network models are much better at modeling the process-oriented phenomenon in networks.
2. The Cell Model provides a flexible approach to creating the network models. The computational models of nodes and connections can be as simple as those of regular graph models or as complex as finite state machines.
3. The Cell Model is applicable across multiple domains. The thesis showcases the applicability of the Cell Model to the domains of computer networks and transportation networks.
4. The Cell Model with its message passing paradigm is very close to the message passing method of concurrent computation. Hence, the Cell Model works well for computer clusters (data centers).

7.5 Future Work

One promising area of future work is to generate the domain-specific functional and behavioral views of the Cell Model. The domain-specific views permit computational complexity analysis of the the Cell Model for the chosen domain.

The results in this thesis also lay the ground work for an effective network measurement workbench and a multi-modal transit scheduler. A natural extension

to Darshini is to add more network-centric data types such as socket. The custom data types defined in Bro scripting language [112] are a testament to the convenience of having well-suited data types for a packet processing software.

Ultimately, the goal of multi-modal transit scheduler is to provide itinerary search across distributed and independent operators. Another area for further exploration is the modelling of disaster recovery scenarios. A comprehensive disaster recovery solution requires live updates and dynamic alternative path detection.

The Cell Model proposed in this thesis can address the modelling issues in the areas of biological networks and agent-based systems. The Cell Model can provide a solution to the problem of mutating infectious diseases. The problem of disease propagation on biological networks can potentially be more effectively solved by using the domain-specific functional and behavioral views of the Cell Model. There has been active interest in the application of multirelational network structural analysis techniques to model the energy grid where energy sources are modelled as agents [124]. The prior work like multi-agent simulation environment [125] attempts to combine the techniques of network analysis with agent-based systems to model traffic simulations. The functional and behavioral views allow for a more accurate specification of agents in such agent-based systems.

Bibliography

- [1] M. Kivelä, A. Arenas, M. Barthelemy, J. P. Gleeson, Y. Moreno, and M. A. Porter. Multilayer networks. *Journal of complex networks*, 2(3):203–271, 2014.
- [2] M. Barthélemy. Spatial networks. *Physics Reports*, 499(1-3):1–101, 2011.
- [3] J. Gao, S. V. Buldyrev, H. E. Stanley, X. Xu, and S. Havlin. Percolation of a general network of networks. *Physical Review E*, 88(6):062816, 2013.
- [4] tshark: Terminal-based Wireshark. https://www.wireshark.org/docs/wsug_html_chunked/AppToolstshark.html/. Accessed: 2018-05-08.
- [5] J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner. Intriguingly Simple and Fast Transit Routing. In *Experimental Algorithms*, pages 43–54. Springer, 2013.
- [6] M. Kurant and P. Thiran. Layered complex networks. *Physical review letters*, 96(13):138701, 2006.
- [7] M. Berlingerio, M. Coscia, F. Giannotti, A. Monreale, and D. Pedreschi. Multidimensional networks: foundations of structural analysis. *World Wide Web*, 16(5-6):567–593, 2013.
- [8] V. Nicosia, G. Bianconi, V. Latora, and M. Barthelemy. Growing multiplex networks. *Physical review letters*, 111(5):058701, 2013.
- [9] M. Magnani and L. Rossi. The ml-model for multi-layer social networks. In *Advances in Social Networks Analysis and Mining (ASONAM), 2011 International Conference on*, pages 5–12. IEEE, 2011.
- [10] S. Boccaletti, G. Bianconi, R. Criado, C. I. Del Genio, J. Gómez-Gardenes, M. Romance, I. Sendina-Nadal, Z. Wang, and M. Zanin. The structure and dynamics of multilayer networks. *Physics Reports*, 544(1):1–122, 2014.

- [11] K. Wehmuth, A. Ziviani, and E. Fleury. A unifying model for representing time-varying graphs. In *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*, pages 1–10. IEEE, 2015.
- [12] K. Wehmuth, É. Fleury, and A. Ziviani. Multiaspect graphs: algebraic representation and algorithms. *Algorithms*, 10(1):1, 2016.
- [13] F. Schulz, D. Wagner, and K. Weihe. Dijkstra’s algorithm on-line: an empirical case study from public railroad transport. *ACM Journal of Experimental Algorithmics*, 5, Dec. 2000. ISSN: 1084-6654.
- [14] H. Bast, D. Delling, A. Goldberg, M. Müller-Hannemann, T. Pajor, P. Sanders, D. Wagner, and R. F. Werneck. Route planning in transportation networks. In *Algorithm engineering*, pages 19–80. Springer, 2016.
- [15] M. De Domenico, V. Nicosia, A. Arenas, and V. Latora. Structural reducibility of multilayer networks. *Nature communications*, 6:6864, 2015.
- [16] W. Xie, Y. Tian, Y. Sismanis, A. Balmin, and P. J. Haas. Dynamic Interaction Graphs with Probabilistic Edge Decay. In *2015 IEEE 31st International Conference on Data Engineering*, pages 1143–1154, Apr. 2015.
- [17] M. Crovella and B. Krishnamurthy. *Internet Measurement: Infrastructure, Traffic and Applications*. John Wiley & Sons, Inc., New York, NY, USA, 2006. ISBN: 047001461X.
- [18] W. Lehr, S. Bauer, and D. D. Clark. Measuring Performance when Broadband is the New PSTN. *Journal of Information Policy*, 3:411–441, 2013.
- [19] M. Mathis, J. Heffner, and R. Reddy. Web100: extended TCP instrumentation for research, education and diagnosis. *SIGCOMM Comput. Commun. Rev.*, 33(3):69–79, July 2003. ISSN: 0146-4833.
- [20] M. Mathis, J. Heffner, and R. Raghunarayan. TCP Extended Statistics MIB. RFC 4898 (Proposed Standard), Internet Engineering Task Force, May 2007. URL: <http://www.ietf.org/rfc/rfc4898.txt>.
- [21] S. Sundaresan, W. de Donato, N. Feamster, R. Teixeira, S. Crawford, and A. Pescapè. Broadband internet performance: a view from the gateway. *SIGCOMM Comput. Commun. Rev.*, 41(4):134–145, Aug. 2011. ISSN: 0146-4833.

- [22] C. Kreibich, N. Weaver, B. Nechaev, and V. Paxson. Netalyzr: illuminating the edge network. In *Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement, IMC '10*, pages 246–259, Melbourne, Australia. ACM, 2010. ISBN: 978-1-4503-0483-2.
- [23] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach. Virtual network embedding: a survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, 2013.
- [24] M. Chowdhury, M. R. Rahman, and R. Boutaba. Vineyard: Virtual Network Embedding Algorithms with Coordinated Node and Link Mapping. *IEEE / ACM Transactions on Networking (TON)*, 20(1): 206–219, 2012.
- [25] J. Mogul, R. Rashid, and M. Accetta. The Packet Filter: An Efficient Mechanism for User-level Network Code. *SIGOPS Oper. Syst. Rev.*, 21(5):39–51, Nov. 1987.
- [26] S. McCanne and V. Jacobson. The BSD Packet Filter: A New Architecture for User-level Packet Capture. In *USENIX Winter 1993 Conference, USENIX'93*, pages 2–2, San Diego, California. USENIX Association, 1993.
- [27] G. Gibb, G. Varghese, M. Horowitz, and N. McKeown. Design Principles for Packet Parsers. In *ACM / IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 13–24, Oct. 2013.
- [28] M. Attig and G. Brebner. 400 Gb/s programmable packet parsing on a single FPGA. In *ACM / IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pp. 12–23, Oct. 2011.
- [29] V. A. P. Kumar, V. Thiyagarajan, and N. Ramasubramanian. A Survey of Packet Classification Tools and Techniques. In *Computing Communication Control and Automation (ICCUBEA), 2015 International Conference on*, pages 103–107, Feb. 2015. DOI: [10.1109/ICCUBEA.2015.26](https://doi.org/10.1109/ICCUBEA.2015.26).
- [30] Ntopng – high-speed web-based traffic analysis and flow collection. <http://www.ntop.org/products/traffic-analysis/ntop/>. Accessed: 2017-05-08.
- [31] W. D. Team. Performance – Wireshark Wiki. <https://wiki.wireshark.org/Performance>. Accessed: 2017-05-08.

- [32] TCPDUMP/LIBPCAP public repository. <http://www.tcpdump.org/>. Accessed: 2017-05-08.
- [33] Linux Socket Filtering aka Berkeley Packet Filter (BPF). <https://www.kernel.org/doc/Documentation/networking/filter.txt>. Accessed: 2017-05-08.
- [34] P4. <http://p4.org/>. Accessed: 2017-05-08.
- [35] P. Bosshart, D. Daly, G. Gibb, M. Izzard, N. McKeown, J. Rexford, C. Schlesinger, D. Talayco, A. Vahdat, G. Varghese, and D. Walker. P4: programming protocol-independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95, July 2014. ISSN: 0146-4833. DOI: [10.1145/2656877.2656890](https://doi.org/10.1145/2656877.2656890). URL: <http://doi.acm.org/10.1145/2656877.2656890>.
- [36] P. Benáček, V. Puš, and H. Kubátová. Automatic generation of 100 Gbps packet parsers from P4 description. www.beba-project.eu/papers/CESNET_p4.pdf. Accessed: 2017-05-08.
- [37] C. Kozanitis, J. Huber, S. Singh, and G. Varghese. Leaping Multiple Headers in a Single Bound: Wire-Speed Parsing Using the Kangaroo System. In *IEEE INFOCOM*, pages 1–9, Mar. 2010.
- [38] V. Puš, L. Kekely, and J. Kořenek. Low-Latency Modular Packet Header Parser for FPGA. In *ACM / IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*, pages 77–78. ACM, 2012.
- [39] S. Kumar. *Acceleration of Network Processing Algorithms*. PhD thesis, Department of Computer Science and Engineering, Washington University, St. Louis, May 2008.
- [40] L. Deri and S. Suin. Effective Traffic Measurement using ntop. *IEEE Communications Magazine*, 38(5):138–143, May 2000.
- [41] R. Geisberger, P. Sanders, D. Schultes, and D. Delling. Contraction hierarchies: faster and simpler hierarchical routing in road networks. In *7th International Workshop on Experimental Algorithms (WEA'08)*. Volume 5038, Lecture Notes in Computer Science, pages 319–333. Springer Berlin Heidelberg, 2008.

- [42] H. Bast. Car or public transport – two worlds. In *Efficient Algorithms*. Volume 5760, Lecture Notes in Computer Science, pages 355–367. Springer Berlin Heidelberg, 2009.
- [43] E. Pyrga, F. Schulz, D. Wagner, and C. Zaroliagis. Efficient models for timetable information in public transportation systems. *ACM Journal of Experimental Algorithmics (JEA)*, 12:2–4, 2008.
- [44] M. Müller-Hannemann, F. Schulz, D. Wagner, and C. Zaroliagis. Timetable information: models and algorithms. In *Algorithmic Methods for Railway Optimization*. Volume 4359, Lecture Notes in Computer Science, pages 67–90. Springer Berlin Heidelberg, 2007.
- [45] M. Müller-Hannemann and K. Weihe. Pareto shortest paths is often feasible in practice. In *5th International Workshop on Algorithm Engineering*. Volume 2141, Lecture Notes in Computer Science, pages 185–197. Springer Berlin Heidelberg, 2001.
- [46] General Transit Feed Specification (GTFS). <https://developers.google.com/transit/?hl=en>. Accessed: 2015-10-30.
- [47] D. Delling, T. Pajor, and D. Wagner. Engineering time-expanded graphs for faster timetable information. In *Robust and Online Large-Scale Optimization*. Volume 5868, Lecture Notes in Computer Science, pages 182–206. Springer Berlin Heidelberg, 2009.
- [48] H. Bast, E. Carlsson, A. Eigenwillig, R. Geisberger, C. Harrelson, V. Raychev, and F. Viger. Fast routing in very large public transportation networks using transfer patterns. In *Proceedings of the 18th Annual European Symposium on Algorithms (ESA'10)*. Volume 6346, Lecture Notes in Computer Science, pages 290–301. Springer Berlin Heidelberg, 2010.
- [49] A. Orda and R. Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM (JACM)*, 37(3):607–625, July 1990.
- [50] A. Orda and R. Rom. Minimum weight paths in time-dependent networks. *Networks*, 21(3):295–319, 1991.
- [51] G. S. Brodal and R. Jacob. Time-dependent networks as models to achieve fast exact time-table queries. *Electronic Notes in Theoretical Computer Science*, 92:3–15, 2004.

- [52] R. Geisberger. Contraction of timetable networks with realistic transfers. In *Proceedings of the 9th International Symposium on Experimental Algorithms (SEA'10)*. Volume 6049, Lecture Notes in Computer Science, pages 71–82. Springer Berlin Heidelberg, 2010.
- [53] D. Delling, T. Pajor, and R. F. Werneck. Round-based public transit routing. *Transportation Science*, 49(3):591–604, 2015.
- [54] H. Bast and S. Storandt. Flow-based guidebook routing. In *ALLENEX*, pages 155–165, 2014.
- [55] H. Bast and S. Storandt. Frequency-based search for public transit. In *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 13–22. ACM, 2014.
- [56] B. Strasser. *Delay-Robust Stochastic Routing In Timetable Networks*. Master's thesis, Department of Informatics, Institute of Theoretical Informatics, KIT, Karlsruhe, Germany, July 2012.
- [57] J. Dibbelt, T. Pajor, B. Strasser, and D. Wagner. Connection Scan Algorithm. *arXiv preprint arXiv:1703.05997*, 2017. Accessed: 2017-07-30.
- [58] B. Strasser and D. Wagner. Connection scan accelerated. In *Proceedings of the Meeting on Algorithm Engineering & Experiments*, pages 125–137. Society for Industrial and Applied Mathematics, 2014.
- [59] M. Rosvall and C. T. Bergstrom. Maps of random walks on complex networks reveal community structure. *Proceedings of the National Academy of Sciences*, 105(4):1118–1123, 2008.
- [60] J. W. Neal and Z. P. Neal. Nested or networked? future directions for ecological systems theory. *Social Development*, 22(4):722–737, 2013.
- [61] Y. Wang, I. Matta, F. Esposito, and J. Day. Introducing protorina: a prototype for programming recursive-networking policies. *ACM SIGCOMM Computer Communication Review*, 44(3):129–131, 2014.
- [62] H. Bast, S. Funke, and D. Matijevic. Ultrafast shortest-path queries via transit nodes. In volume 74, pages 175–192. American Mathematical Society Providence, RI, 2009.

- [63] V. Gligorijević, M. Skowron, and B. Tadić. Directed Networks of Online Chats: Content-Based Linking and Social Structure. In *Signal Image Technology and Internet Based Systems (SITIS), 2012 Eighth International Conference on*, pages 725–730, Nov. 2012.
- [64] T. Sinha and I. Rajasingh. Investigating Substructures in Goal Oriented Online Communities: Case study of Ubuntu IRC. In *Advance Computing Conference (IACC), 2014 IEEE International*, pages 916–922, Feb. 2014.
- [65] P. Mutton. Inferring and Visualizing Social Networks on Internet Relay Chat. In *Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on*, pages 35–43, July 2004.
- [66] A. D. J. Leskovec K. J. Lang and M. W. Mahoney. Statistical properties of Community Structure in Large Social and Information Networks. In *IW3C2, 2008 WWW conference on*, pages 695–704, Apr. 2008.
- [67] F. D. Malliaros and M. Vazirgiannis. Clustering and community detection in directed networks: a survey. *Physics Reports*, 533(4):95–142, 2013.
- [68] S. H. Lee, J. M. Magallanes, and M. A. Porter. Time-dependent community structure in legislation cosponsorship networks in the congress of the republic of peru. *Journal of Complex Networks:cnw004*, 2016.
- [69] B. Eisenbart, L. Blessing, K. Gericke, et al. Functional modelling perspectives across disciplines: a literature review. In *Proceedings of 12th international design conference, design*, 2012.
- [70] M. S. Erden, H. Komoto, T. J. van Beek, V. D’Amelio, E. Echavarria, and T. Tomiyama. A review of function modeling: approaches and applications. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 22(02):147–169, 2008.
- [71] J. Krogstie. Perspectives to process modeling. In *Business Process Management*, pages 1–39. Springer Berlin Heidelberg, 2013.
- [72] D. Harel and Y. Setty. Generic reactive animation: realistic modeling of complex natural systems. *Formal Methods in Systems Biology*:1–16, 2008.
- [73] S. J. Mason. Feedback theory-some properties of signal flow graphs. *Proceedings of the IRE*, 41(9):1144–1156, 1953.

- [74] H. Amir-Kroll, A. Sadot, I. R. Cohen, and D. Harel. Gemcell: a generic platform for modeling multi-cellular biological systems. *Theoretical Computer Science*, 391(3):276–290, 2008.
- [75] H. Kugler, A. Larjo, and D. Harel. Biocharts: a visual formalism for complex biological systems. *Journal of The Royal Society Interface*, 2009.
- [76] R. R. McCune, T. Weninger, and G. Madey. Thinking like a vertex: a survey of vertex-centric frameworks for large-scale distributed graph processing. *ACM Computing Surveys (CSUR)*, 48(2):25, 2015.
- [77] S. J. Mason. Feedback theory: further properties of signal flow graphs, 1956.
- [78] The MLab NDT Dataset, January 2009 - December 2014. URL: <http://measurementlab.net/tools/ndt>.
- [79] R. Carlson. Network Diagnostic Tool, 2014. URL: <http://software.internet2.edu/ndt/>.
- [80] R. Carlson. Developing the Web100 based network diagnostic tool (ndt). In *Proceedings of the 4th International Conference on Passive and Active Measurement (PAM)*, 2003.
- [81] Network Diagnostic Test, 2015. URL: <http://www.measurementlab.net/tools/ndt>.
- [82] Google cloud platform M-Lab data set, 2014. URL: <https://cloud.google.com/bigquery/docs/dataset-mlab#schema>.
- [83] M-Lab. M-Lab Dataset - Bigquery Schema, 2014. URL: <https://github.com/m-lab/mlab-wikis/blob/master/BigQueryMLabDataset.md>.
- [84] Documentation of variables for the Web100 TCP Kernel Instrumentation Set (KIS) project, 2014. URL: <http://www.web100.org/download/kernel/tcp-kis.txt>.
- [85] TRAI. Bandwidth required for ISPs for better connectivity and improved quality of service, 2009. URL: <http://www.trai.gov.in/WriteReaddata/ConsultationPaper/Document/cpaper15jan09.pdf>.
- [86] K. Nichols, S. Blake, F. Baker, and D. Black. Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers. RFC 2474 (Proposed Standard), Internet Engineering Task Force, Dec. 1998. URL: <http://www.ietf.org/rfc/rfc2474.txt>. Updated by RFCs 3168, 3260.

- [87] B. Davie, A. Charny, J. Bennet, K. Benson, J. L. Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An Expedited Forwarding PHB (Per-Hop Behavior). RFC 3246 (Proposed Standard), Internet Engineering Task Force, Mar. 2002. URL: <http://www.ietf.org/rfc/rfc3246.txt>.
- [88] Indian Railways. Trains/fare/accommodation availability between important stations. http://www.indianrail.gov.in/between_Imp_Stations.html. Accessed: 2015-10-30.
- [89] M. Kurant and P. Thiran. Extraction and analysis of traffic and topologies of transportation networks. *PHYSICAL REVIEW E*, 74:036114, 2006.
- [90] P. Sen, S. Dasgupta, A. Chatterjee, P. Sreeram, G. Mukherjee, and S. Manna. Small-world properties of the indian railway network. *Physical Review E*, 67:036106, 2003.
- [91] A. Srivastava, B. Mitra, N. Ganguly, and F. Peruani. Correlations in complex networks under attack. *Physical Review E*, 86:036106, 2012.
- [92] Slackware. Slackware OS IRC Logs. <https://phra.gs/logs>, 2017. Accessed: 2017-05-01.
- [93] U. /. Canonical. Ubuntu IRC Logs. <http://irclogs.ubuntu.com/>, 2017. Accessed: 2017-05-01.
- [94] F. Radicchi, C. Castellano, F. Cecconi, V. Loreto, and D. Parisi. Defining and identifying communities in networks. *Proceedings of the National Academy of Sciences of the United States of America*, 101(9):2658–2663, 2004.
- [95] A. Landherr, B. Friedl, and J. Heidemann. A critical review of centrality measures in social networks. *Business & Information Systems Engineering*, 2(6):371–385, 2010.
- [96] N. Crilly. Function propagation through nested systems. *Design Studies*, 34(2):216–242, 2013.
- [97] Y. Hirshfeld and M. Jerrum. Bisimulation equivalence is decidable for normed process algebra. In *International Colloquium on Automata, Languages, and Programming*, pages 412–421. Springer, 1999.
- [98] L. Aceto, A. Ingólfssdóttir, K. G. Larsen, and J. Srba. *Reactive systems: modelling, specification and verification*. cambridge university press, 2007.

- [99] D. Harel and A. Pnueli. Logics and models of concurrent systems. In K. R. Apt, editor, chapter On the Development of Reactive Systems. Springer-Verlag New York, Inc., New York, NY, USA, 1985.
- [100] D. Harel. Statecharts: a visual formalism for complex systems. *Science of computer programming*, 8(3):231–274, 1987.
- [101] Object Modeling Group (OMG). OMG Unified Modeling Language™ (OMG UML). <https://www.omg.org/cgi-bin/doc?formal/15-03-01.pdf>. Accessed: 2018-04-15.
- [102] E. A. Lee and S. Tripakis. Modal models in ptolemy. In *Proceedings of 3rd International Workshop on Equation-Based Object-Oriented Modeling Languages and Tools (EOOLT 2010)*, pages 11–22, Oct. 2010.
- [103] S. Efroni, D. Harel, et al. A theory for complex systems: reactive animation. In *Studies in Multidisciplinarity*. Volume 3, pages 309–324. Elsevier, 2005.
- [104] S. Efroni, D. Harel, and I. R. Cohen. Reactive animation: realistic modeling of complex dynamic systems. *Computer*, 38(1):38–47, 2005.
- [105] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorfer, S. Sachs, and Y. Xiong. Taming heterogeneity-the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, 2003.
- [106] K. Bourrillion and J. Levy. Google Guava. <https://github.com/google/guava>, 2018. Accessed: 2018-04-01.
- [107] G. Agha and C. Hewitt. Concurrent Programming using Actors: Exploiting Large-Scale Parallelism. In *Foundations of Software Technology and Theoretical Computer Science*, pages 19–41. Springer, 1985.
- [108] Plataformatec. Elixir. <http://elixir-lang.org/>. Accessed: 2017-03-14.
- [109] R. K. Karmani, A. Shali, and G. Agha. Actor Frameworks for the JVM Platform: a Comparative Analysis. In *Proceedings of the 7th International Conference on Principles and Practice of Programming in Java*, pages 11–20. ACM, 2009.
- [110] C. Williamson. Internet Traffic Measurement. *Internet Computing, IEEE*, 5(6):70–74, 2001.

- [111] V. Paxson. Strategies for Sound Internet Measurement. In *ACM SIGCOMM Conference on Internet Measurement*, IMC '04, pages 263–271, Taormina, Sicily, Italy. ACM, 2004. ISBN: 1-58113-821-0.
- [112] V. Paxson. Bro: a system for detecting network intruders in real-time. *Computer Networks*, 31(23-24):2435–2463, 1999.
- [113] CRAWDAD: Community Resource for Archiving Wireless Data at Dartmouth. <http://crawdad.org/>. Accessed: 2017-05-07.
- [114] DataCat: Home. <http://imdc.datcat.org/>. Accessed: 2017-05-07.
- [115] A. Arora and S. K. Peddoju. Minimizing Network Traffic Features for Android Mobile Malware Detection. In *International Conference on Distributed Computing and Networking (ICDCN)*, ICDCN '17, 32:1–32:10, Hyderabad, India. ACM, 2017.
- [116] P. Richter, N. Chatzis, G. Smaragdakis, A. Feldmann, and W. Willinger. Distilling the Internet's Application Mix from Packet-Sampled Traffic. In *International Conference on Passive and Active Network Measurement*, pages 179–192. Springer, 2015.
- [117] T. Benson, A. Akella, and D. A. Maltz. Network Traffic Characteristics of Data Centers in the Wild. In *ACM SIGCOMM Conference on Internet Measurement*, pages 267–280. ACM, 2010.
- [118] Timetable: European Train Schedules. <http://www.eurorailways.com/timetable/>. Accessed: 2017-04-20.
- [119] P. Federl and P. Prusinkiewicz. Virtual laboratory: an interactive software environment for computer graphics. In *Computer graphics international*, volume 242, pages 93–100, 1999.
- [120] T. Prasad, K. Sathyanarayanan, S. Tiwari, N. Goveas, and B. Deshpande. t-CSA: A Fast and Flexible CSA Implementation. In *2016 8th International Conference on Communication Systems and Networks (COMSNETS)*, pages 1–6, Jan. 2016.
- [121] H. Bast, J. Sternisko, and S. Storandt. Delay-robustness of transfer patterns in public transportation route planning. In *ATMOS-13th Workshop on Algorithmic Approaches for Transportation Modelling, Optimization, and Systems-2013*, volume 33, pages 42–54. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 2013.

- [122] Directorate General of Civil Aviation. Domestic flight schedule. http://dgca.nic.in/dom_flight_schedule/flight_index.htm. Accessed: 2015-10-30.
- [123] GSRTC. Search & book tickets. <http://www.gsrtc.in/site/>. Accessed: 2015-10-30.
- [124] J. M. G. de Durana, O. Barambones, E. Kremers, and L. Varga. Agent based modeling of energy networks. *Energy Conversion and Management*, 82:308–319, 2014.
- [125] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan. Mason: a multiagent simulation environment. *Simulation*, 81(7):517–527, 2005.

Responses to reviewers' comments

Corrections suggested by Examiner 1: (Prof. Sudip Misra)

1. In Chapter 3, Fig. 3.2 shows the average throughput of users for different months during the year 2009-14. The figure quality is poor and axis label is missing in all the subgraphs.
2. In Chapter 3, Section 3.2.4, the reliability parameter of QoS needs more explanation. It is good to explain the variable names used in the mathematical analysis immediately after the mathematical expression is given (to increase the readability). How the correlation coefficients between different QoS parameters is calculated? Is it directly taken from the source or the author has done it by his own?
3. In Chapter 5, Darshini architecture – is there any specific format for file input to pipeline stage?
4. In Chapter 6, the complexity analysis of proposed t-CSA algorithm should be provided.
5. In Chapter 7, author should add a subsection for future work.

The modifications made in the thesis as per the suggestions of Prof. Sudip Misra are outlined below.

1. In Chapter 3, Fig. 3.2 shows the average throughput of users for different months during the year 2009-14. The figure quality is poor and axis label is missing in all the subgraphs.

Ans: The image quality has been improved and the axes labels are added in all the subgraphs. Please see Figure 3.2 on page 26.

2. In Chapter 3, Section 3.2.4, the reliability parameter of QoS needs more explanation. It is good to explain the variable names used in the mathematical analysis immediately after the mathematical expression is given (to increase the readability). How the correlation coefficients between different QoS parameters is calculated? Is it directly taken from the source or the author has done it by his own?

Ans: The QoS equation (Eq. 3.2) has been corrected. All the variables used in the equation have been explained in the text accompanying the equation (Please see page 28). The QoS parameters explained in Eq. 3.2 are available from the source dataset. The values of Throughput, Jitter, Latency and Reliability parameters are computed as per Eq. 3.2. We calculate the correlations between all possible pairs of QoS parameters using the standard correlation equation of two random variables. The corresponding text on page 29 has been corrected to explain the calculation of correlation equation.

3. In Chapter 5, Darshini architecture – is there any specific format for file input to pipeline stage?

Ans: Sec 5.4.1 **Preliminaries** has been expanded to include explanation for data formats of all the inputs. The section has also been renamed to Sec 5.4.1 **Input Data Formats**. Please see page 82.

4. In Chapter 6, the complexity analysis of proposed t-CSA algorithm should be provided.

Ans: The algorithmic complexity analysis has been added to both CSA and t-CSA algorithms. New Section 6.3.1 discusses the algorithmic complexity of CSA algorithm (Please see page 106). New Section 6.4.1 discusses the algorithmic complexity of t-CSA algorithm. Please see page 109.

5. In Chapter 7, author should add a subsection for future work.

Ans: Section 7.5 **Future Work** has been added. Please see page 125.

Corrections suggested by Examiner 2: (Prof. Anil Maheshwari)

No changes were suggested

Corrections suggested by Examiner 3: (Prof. Neena Goveas)

No changes were suggested

Corrections suggested by Examiner 2: (Prof. Bharat M. Deshpande)

No changes were suggested

List of Publications

Cell Model for Multidimensional Networks:

1. **Prasad Talasila**, Neena Goveas and Bharat Deshpande, Cell Model for Multidimensional Networks. (Communicated)

Multi-modal Transit Networks:

2. **Prasad Talasila**, Aparajita Haldar, Suhas S. Pai, Neena Goveas, and Bharat M. Deshpande, Multi-modal Transit Scheduler: An Actor-based Concurrent Approach, 20th IEEE International Conference on Intelligent Transportation Systems, Yokohama, JAPAN, 16 - 19 October, 2017.
3. **TSRK Prasad**, Kartik Sathyanarayanan, Sukriti Tiwari, Neena Goveas and Bharat Deshpande, t-CSA: A fast and flexible CSA Implementation, 2nd Workshop on Intelligent Transport Systems, 8th International Conference on Communication Systems & Networks, 5 - 9 January, 2016.
4. **Prasad Talasila**, Shaik Asifullah, Neena Goveas and Bharat Deshpande, Transit Timetables as Multi-Layer Networks, 4th Workshop on Intelligent Transport Systems, 8th International Conference on Communication Systems & Networks, 3 - 7 January, 2018.

Network Measurements:

5. **Prasad Talasila**, Mihir Kakrambe, Anurag Rai, Sebastin Santy, Neena Goveas, Bharat M. Deshpande, BITS Darshini: A Modular, Concurrent Protocol Analyzer Workbench, 19th International Conference on Distributed Computing and Networking (ICDCN), Varanasi, 4 - 7 January, 2018.
6. **TSRK Prasad**, Dhruv Shekhawat, Sukanto Guha, Neena Goveas and Bharat Deshpande, Analysis of Impartial Quality Measurements on Indian Broadband Connections, 22nd National Conference on Communications, 4 - 7 March, 2016.

Multidimensional Online Social Networks:

7. **Prasad Talasila**, Rohan Goel, Dhruv Shekhawat, Neena Goveas and Bharat Deshpande, Discovering Patterns in Activities of Online Chat Communities: A Case Study on Internet Relay Chat Channels, Springer Journal of World Wide Web (WWW). (Communicated)

Mapping of publications to Thesis chapters:

Chapter No.	Chapter Name	Publication Number(s)
1.	Introduction	None
2.	Literature Survey	None
3.	Structural Analysis of Networks	4, 6, 7
4.	The Cell Model	1
5.	Darshini - Protocol Analyzer	5
6.	Multi-Modal Transit Scheduler	2, 3
7.	Conclusion	None

Biography of the Candidate

S R K Prasad Talasila, is currently serving as a Lecturer in the Department of Computer Science & Information Systems, BITS Pilani K K Birla Goa Campus, Goa, India. He received his Bachelor's degree in Electronics & Instrumentation Engineering from BITS, Pilani - Pilani Campus, Rajasthan in 2001. He completed his Masters degree in Computer Science and Engineering in 2009 from RVR & JC College of Engineering, under Acharya Nagarjuna University, Guntur, Andhra Pradesh. He graduated with distinction in both the Bachelor's and the Master's degree programs. He is currently pursuing Ph.D from BITS-Pilani, K K Birla Goa Campus, Goa. His research interests include Network Science and its applications to transport, social and computer networks.

Biography of the Supervisor

Neena Goveas is with the Department of Computer Science at BITS Pilani K K Birla Goa campus. Earlier she was with Department of Physics BITS Pilani, Pilani campus.

For her PhD thesis, she worked on "Mean field approaches to thermodynamic properties of magnetic systems" at IIT Bombay, advisor Prof. G. Mukhopadhyay. She worked on INDO-US sponsored project "Development and characterization of materials suitable for magneto-optic Devices" at A. C. R. E., I. I. T. Bombay. She worked as DST-Young Scientist Scheme Project entitled "Study of low dimensional magnetic systems" at IIT Guwahati.

Her main theme of research work is to study magnetic systems. Using various mean field and computational approaches to understand their properties. Recent research work is on Network Science and its applications to transport, social and computer networks; modeling of Cyber Physical Systems and Wireless Sensor Networks.

Biography of the Co-Supervisor

Prof. Bharat Deshpande received his Ph.D. degree from IIT Bombay in the year 1998. After which for a year he received postdoctoral fellowship from Department of Atomic Energy.

Before joining BITS he worked as a Reader for two years at M.S. University Baroda. Dr. Deshpande joined BITS Pilani in year 2001 and moved to BITS Pilani, Goa Campus in the Year 2005.

He was the head of the Department of Computer Science & Information Systems at Goa from 2006 to 2017. Apart from basic courses in Computer Science, he has taught specialized courses like Algorithms, Theory of Computation, Parallel Computing, Artificial Intelligence and a few more. His research interests are in areas of Complexity Theory, Parallel Algorithms, and Data Mining. Over the years he has supervised numerous masters and doctoral students. He has many national and international publications to his credit.

In 2010 he was felicitated as Distinguished faculty at the BITS alumni global meet held in New Delhi. He was also on the Board of Studies of College of Engineering, Pune. Dr. Deshpande was also Vice President of the Goa Chapter of Computer Society of India.

Along with his academic interests, Dr. Deshpande is also a sports enthusiast. He was In-charge of Students sporting activities at BITS-Pilani, Goa. He was a member on the panel Committee of the Goa Football Association.