Georg-August-Universität
Göttingen
Fakultät für Mathematik und Informatik

On monitoring and fault management of next generation networks

Dissertation
zur Erlangung des
mathematisch-naturwissenschaftlichen Doktorgrades
“Doctor rerum naturalium”
der Georg-August-Universität Göttingen

Vorgelegt von
Lei Shi
Shanxi,China

Göttingen 2010

Referent: Prof.Dr.Xiaoming Fu
Korreferent: Prof.Dr.Tilman Wolf, Prof.Dr. Dieter Hogrefe

Tag der mündlichen Prüfung: 4 November 2010

Abstract

This thesis investigates monitoring and fault management of next generation networks, in particular in environments where the network nodes within an Autonomous System (AS) are centrally controlled and managed. Existing works on network monitoring and fault management are developed in an isolated and extremely complex manner, where the management protocols, such as SNMP, IPFIX and PSAMP, are separately designed and deployed without an overall consideration. To address this issue, we propose a systematic framework for network monitoring and fault management, which takes account of monitoring protocol, traffic matrix composition and derivation, system rebooting, dynamic fault management and related security issues.

We propose a new network monitoring framework, which exploits the extensibility of the Internet Protocol (IP), especially for IP Version 6 (IPv6), the fundamental building block for next generation networks. This is implemented by defining a new IPv6 hop-by-hop extension header. Messages with such header would be able to carry metrics related to node and links along the path when they traverse the network. This approach is augmented with a path-based intrinsic monitoring protocol, which can effectively associate SNMP-based MIB information to a network path within the AS domain.

To deal with fault management, we propose a novel transient loop avoidance algorithm, which exploits traffic matrix information for updating forwarding tables in an optimal order achieving minimal link overflow. For fault recovery, we present an efficient network rebooting algorithm, which utilizes a priori knowledge of network traffic demand to minimize the rebooting time of all nodes in the entire AS, while ensuring that only a designated portion of traffic volume is affected.

The proposed network monitoring, fault management and recovery schemes are evaluated through extensive analysis and experiments. Results show that our monitoring approach only generates less than 5% of the traffic generated by traceroute, at only around 12% of the time taken by traceroute, to retrieve information for a 16-node network; our fault management method can achieve zero transient loop with minimal link overflow; and our fault recovery scheme can significantly reduce rebooting time, which is 86.78% lower than the traditional approach by rebooting network node one by one. These approaches, although not yet implemented in an

operational network, would provide insights for the future designers and operators of next generation networks.

Acknowledgments

I owe a debt of gratitude to many people for this thesis. My deepest gratitude goes to my advisor Professor Xiaoming Fu, for introducing me to the world of scientific research, and for his support, guidance and conversations. With his intelligence, generosity and kindness, Xiaoming has given invaluable support both at the professional and personal level.

I am grateful to previous and current members of computer networks group in University of Göttingen: Dr. Jun Lei, Dr. Fang-Chun Kuo, Dr. Yang Chen, Florian Tegeler, Niklas Neumann, Mayutan Arumaithurai, Nikunj Modi and John-Patrick Wowra. I appreciate the friendship, discussions and the stimulating working environment.

I am grateful to Miguel Ponce de Leon and Dr. Kevin Quinn at TSSG for hosting me. I appreciate the guidance, support and discussions from all my team members during my time in TSSG. I would like to thank everyone at the EFIPSANS group in TSSG for the nice work atmosphere, in particular Dr. Kevin Quinn, Dr. Alan Davy, Dr. Steven Davy, John Ronan and David Muldowney.

I would like to express special thanks to Dr. Jing Fu. With his intelligence, generosity, and kindness, Jing has given me invaluable support.

I would also like to thank the organizations and entities that have funded my research: the computer networks group of University of Göttingen and the EU FP7 EFIPSANS project.

I am happy to thank my wife Xingyun for her understanding, continuous support and patience. To my parents, I give them my deepest appreciation.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Network Monitoring and Fault Management

The Internet is becoming increasingly relevant and indispensable in our daily lives. The diversity in Internet services includes IP television (IPTV), Video on Demand (VoD), online gaming and voice over IP (VoIP), as well as data services such as HTTP, email and FTP. Over the years, the Internet has become a complex network linking hundreds of millions of nodes, consisting of thousands of interconnected autonomous systems (ASs). An AS is a collection of routers and links operated by an single institution, for example, a firm or an Internet Service Provider (ISP). Within an AS, the routers communicate via intra-domain routing protocols such OSPF (Open Shortest Path First), RIP (Routing Information Protocol) and IS-IS (Intermediate System-Intermediate System). Neighboring ASs use the Border Gateway Protocol (BGP) to exchange information containing details of ways to reach destinations in the Internet, without exposing the specifics of their network topologies and routing policies.

As the communication networks grow in size and complexity, monitoring and fault management of the network have become an increasingly important and challenging task. Due to their practical importance, network monitoring and fault management have been an active area of networking research for many years.

Network monitoring systems play a crucial role in managing communication networks. They collect a wide range of network related metrics so that network managers can carry out essential management functions such as Fault, Configuration, Accounting, Performance and Security (FCAPS) [5]. The network manager has a responsibility to its customers to maintain contractual service level agreement (SLA) in providing of any services over its network. SLAs detail out the contractual obligations between service providers and their customers, wherein performance assurances provided by SLAs range from network path availability to transport-level metrics and application-specific metrics such as web server response or media stream quality. Customers are the main source of revenue for the network operator and failure to meet SLAs can result in loss of revenue. To ensure that SLA performance targets are met with high probability, service providers collect measurements either passively within the network, by injecting measurement

probes into the network, or by using a combination of both. The effectiveness and efficiency of monitoring systems have a direct impact on the quality of network services.

When a network anomaly caused by, for example, software bugs, hardware defects and deny of service (DoS) attacks, is detected by the network monitoring system, fault management could effectively reduce the network downtime and improve the network availability, which is essential for maintaining the operations of the network. Typical fault management tasks include detecting and identifying network problems, determining the root cause and recovering from the faults. Nowadays, the traffic characteristics of real time applications such as VoIP, multimedia and IPTV impose more complications on fault management, especially fault recovery. The fault recovery should minimize the impact on these applications, which are potentially very valuable to the network operators. It is desirable to provide fault management with the traffic demands set in mind, for any large-scale network supporting a lot of users and a diverse set of applications.

In this thesis, several research issues related to network monitoring and fault management were studied. The rest of this chapter is organized as follows. Section 1.2 illustrates the issues investigated in this thesis. First, a new path-based monitoring system in IPv6 networks is discussed. Second, a method of updating forwarding tables in the centrally controlled IP network is detailed upon. Third, we provide the motivation of minimizing the rebooting time of network wide node updates. Section 1.3 presents the thesis contributions whereas Section 1.4 outlines the roadmap of the thesis.

1.2 Motivation and Problems

These days, a primary concern of operators in IP network is ensuring that the network performance on paths to premium customers is within predefined SLA. Ways to efficiently monitor the performance of the path play a crucial role in the network management. The monitored information provides input information to the fault management. Fault management could be used to manage network maintenance, fibre cuts, power outages, node failures, and so on. As the next generation network wants to support traditional data services as well as the real time applications such as VoIP and IPTV, the requirements for the fault management also include the minimum impact on the data traffic, especially on the real time traffic carried on the network. A key concept for network monitoring and fault management throughout this thesis is traffic matrix. A traffic matrix represents the traffic volumes between the origin and the destination in a network, which is provided by monitoring systems and is essential for fault management in next generation networks. Previous

work in [6–8] infers traffic matrix indirectly from observed link loads, which are readily available via the Simple Network Management Protocol (SNMP) as every router maintains a counter of the number of bytes transmitted and received on each of its interfaces.

The path-based monitoring mechanisms for monitoring systems are investigated here. With the traffic matrix derived from the path based monitoring mechanisms as input, we mainly investigate the traffic overflow problem when we update the forwarding tables in a centralized-control network and also ways to minimize the network wide rebooting time considering designated traffic loss.

1.2.1 Path-based Monitoring

As operators are trying to assure the quality of delivery of premium services for their most valued customers, a primary concern is ensuring that the network performance on paths to their premium customers is within a range of pre-defined criteria, according to pre-set SLA. To associate these metrics with particular paths requires offline processing. Furthermore, with conventional monitoring tools it is difficult to get a timely view of a specific path's performance, as the collected data will likely be already outdated due to the dynamic nature of networks.

Minimizing overhead is also critical for monitoring in order to avoid "Heisenberg effects", in which the additional traffic imposed by the network monitors perturbs the network's performance metrics and biases the resulting analysis in unpredictable ways [9]. As a result, an intrinsic path-based monitoring approach that considers efficiently and timely co-relates the performance metrics along the path is an important issue to be addressed in next generation networks.

1.2.2 Loop-Free Forwarding Table Updates with Minimal Link Overflow

The forwarding paths in a network may change due to link/node failures, equipment maintenances and reconfigurations of link weights. When the forwarding paths change, the forwarding tables needs to be updated. During the update process, transient loops might occur since some routers may have updated their forwarding information bases (FIBs) while other have not. Network traffic are delayed and may be lost in the forwarding loop. Applications for interactive multimedia services such as VoIP are sensitive to packet delay and loss. Therefore, transient loops should be avoided whenever possible. Although there have been several approaches to avoid transient loops in a network when forwarding paths change due to link failures, equipment maintenances and reconfigurations of link weights [10, 11], there has been no study considering the possibility of congestion

during the transient phase. For example, in the transient phase, traffic from router A to B may go through link e where the available bandwidth is small, it could be congested in the transient phase. Therefore, a loop-free update order that considers link capacities and traffic demands to avoid congestion is studied.

1.2.3 Minimization of Network Wide Node Update Time

During the maintenance of communication networks, nodes need to be rebooted or brought off line often for software updates, configuration updates or hardware maintenance [12, 13]. ISPs often run the same protocols and use equipment from the same vendor network-wide, increasing the probability that a bug causes simultaneous failures [14, 15]. The extreme case is that all nodes in the network need to be rebooted for the maintenance or upgrade.

A simple and straightforward way is to update one node after another. This approach can achieve the best performance in terms of the affected traffic volumes. However, typically this solution is not optimal as it can take a long time to finish the update and for a large network it is not a scalable solution. One router is assumed to take ten minutes to update, including the configuration time [13]. For a network containing hundreds of nodes, it can easily take a few days to update all the nodes. There is also an overhead associated to rebooting each node that can cause instability in routing.

A time-effective way is to update all the nodes in one update, in particular at a time when there is minimum network traffic in the network. This method is the best in terms of time. However, this method does not consider the traffic that the network carries, which is potentially very valuable to the network operator. In this case, all network traffic will be lost. Method to use the network traffic properties among network nodes to effectively reboot the entire network with the constraint that the network can still accommodate a certain percentage of network load by rebooting nodes in groups, is investigated.

1.3 Thesis Contributions

As network monitoring systems play a crucial role in managing communication networks, an approach that stores node and link related metrics within the IP packet header structure as the packet is processed through the network is provided. In addition, the security considerations for this approach are also presented. To overcome the limitations of the above approach, such as the limited extension header space and the proprietary metric naming structure, it is extended to a path-based intrinsic monitoring protocol that can efficiently associate SNMP MIB information

with a network path within a single administrative domain.

Basic traffic statistics for the entire network can be obtained with the path based monitoring approach to compute the traffic matrix. A traffic matrix represents the traffic volumes between the origin and the destination in a network and could be indirectly inferred from observed traffic statistics, which are available via the SNMP MIB information, since every router maintains a counter of the number of bytes transmitted and received on each of its interfaces.

In the fault management of the IP networks, taking the traffic matrix as input, an approach is proposed for updating forwarding tables of routers in an order to avoid transient loops with minimum link overflow. An algorithm is also presented, which utilizes *a priori* knowledge of network traffic demand to minimize the rebooting time of the entire network, under the constraint that only a designated percentage of the network traffic volume be affected.

The proposed network monitoring, fault management and recovery schemes are evaluated through extensive analysis and experiments. Results show that our monitoring approach only generates less than 5% of the traffic and takes only 12% of the time in comparison with traceroute to retrieve monitoring information for a 16-node network, our fault management method can achieve zero transient loop with minimal link overflow, and our fault recovery scheme can significantly reduce rebooting time (86.78% lower that the traditional approach by rebooting network node one by one).

1.3.1 Path-based Monitoring

An approach that stores nodes and link related metrics within the packet header structure as the packet is processed through the network has been proposed here. The designs are generic enough that they can be applied in both IPv4 and IPv6 networks. For IPv6 networks, through the use of a newly proposed IPv6 hop-by-hop extension header, each intermediate node that supports such a header can use it as a medium to carry node specific metrics. For IPv4 networks, a new IP option could be defined for the same purpose. But due to deployment constraints, it may make more sense to focus on IPv6 networks. This approach enables direct monitoring of both the intermediate nodes that a particular packet traverses and characteristics of the packet's spatial trajectory through the network. The fact that this approach reduces the total overhead associated with collecting the metrics in comparison to the popular traceroute network diagnostics tool, while also reducing the total time taken to retrieve these metrics, has been demonstrated here.

In addition, we have proposed a secure method to collect monitored network information utilizing the IPv6 hop-by-hop extension header. A generic IPv6 hop-by-hop extension header, which can be easily extended to include all the possible

monitored information, has been designed. It is presumed that nodes are deployed with IPsec framework support, and a protocol, such as extended ICMPv6, initiates the key distribution and IPsec security associations (SAs) negotiation between the nodes along the flow transfer path and the destination node. Based on that, a light-weight information encryption and authentication scheme to securely transport collected monitoring information has also been discussed. To the best of our knowledge, it is the first try to offer a secure IPv6 hop-by-hop extension header for all kinds of monitoring purpose, which is a significant step for intrinsic monitoring towards real deployment.

To overcome the limitations of the above approach, such as the limited extension header space and the proprietary metric naming structure, a path-based intrinsic monitoring protocol is described, which can efficiently associate SNMP based MIB information to a network path within a single administrative domain. It provides the network operator with a tool that can be used to focus on monitoring individual paths in the network, where the location of the SNMP agents is not known beforehand. The performance overhead associated with our proposed approach is compared to conventional SNMP get/response message exchanges. The results demonstrate a dramatic reduction in both time and traffic overhead in the collection of node specific metrics. This approach also implicitly relates collected metrics to the network path, thereby making the need to correlate metrics to topology and path information redundant.

1.3.2 Loop-Free Forwarding Table Updates with Minimal Link Overflow

This work illustrates a method that compares the initial and final forwarding paths, and obtains the updatable nodes that do not cause any transient loop or link overflow. However, this goal is not always achievable. Therefore, we propose an algorithm to update the forwarding tables that would refrain the link overflows to a minimal level. The performance study on a real topology with two setups confirms that this approach achieves smaller link overflow than by using a previously proposed approach as described in [4].

1.3.3 Minimization of Network Wide Node Update Time

This work presents an algorithm that utilizes *a priori* knowledge of network traffic demand to minimize the rebooting time of the entire network, under the constraint that only a designated percentage of the network traffic volume is affected. The algorithm uses the fact that closely related routers to be upgraded can be simultaneously done to minimize network wide affected traffic. The term ‘closely related’

here refers to nodes that send traffic to one other more than their other neighboring nodes. A gravity traffic model over a real network topology collected from Rocketfeul measurement has been used to simulate our experiments [16]. Performance results show a significant improvement in terms of overall upgrade time and scalability, which indicates the properties of an efficient solution with respect to minimizing impact on network traffic.

1.4 Roadmap of the Dissertation

This dissertation is organized as follows. Chapter 2 provides the background information and literature review conducted on the research issues regarding the work. Chapter 3 outlines a framework of monitoring and fault management. Chapter 4 presents an approach that stores node and link related metrics within the IPv6 packet header structure as the packet is processed through the network. The security considerations for this approach are also presented. To overcome the limitations of the above approach, such as the limited extension header space and the proprietary metric naming structure, Chapter 5 presents a path-based intrinsic monitoring protocol that can efficiently associate SNMP based MIB information to a network path within a single administrative domain. For the fault management, in Chapter 6, the scheme for the loop-free forwarding table updates with minimal link overflow is explained. Chapter 7 presents an algorithm that utilizes *a priori* knowledge of network traffic demand to minimize the rebooting time of the entire network, under the constraint that only a designated percentage of the network traffic volume is affected. Finally, the conclusion and future work are discussed in Chapter 8.

Chapter 2

Background Information and Literature Review

In this chapter, we provide background information of the work conducted in this thesis. In Section 2.1 we discuss the core IPv6 protocols. The centralized control scheme for IP networks is provided in Section 2.2. Section 2.3 gives an introduction to GANA architecture. Section 2.4 describes the fault management concepts. Section 2.5 presents the related work of path based monitoring. Section 2.7 shows the related work of loop-free updates of forwarding tables of routers. In Section 2.6 we discusses the related work of minimizing network reboot time.

2.1 IPv6 protocols

Internet Protocol version 6 (IPv6) has been standardized within the IETF as a protocol for the replacement of IPv4. The main justification behind this move is the lack of available addresses remaining in the IPv4 address pool. This section will discuss the core IPv6 protocols in the areas of control, routing, security, mobility, QoS and extension headers.

2.1.1 Control

The Internet Control Message Protocol version 6 (ICMPv6) has now been integrated into the IPv6 protocol. With the ICMPv6 protocol, a number of network functions have been developed to improve the management of an IPv6 network. These include neighbor discovery, and address auto-configuration. These functions enable nodes within an IPv6 network to discover each other and set up appropriate addressing schemes to enable communications automatically, without the use of a DHCP server. This also reduces the amount of time the network manager must spend on network configuration.

The ICMPv6 protocol also provides multicast group management by integrating the previously separate protocol IGMP [17, 18]. This enables nodes to use ICMPv6 to create, discover and join multicast groups within a network. This func-

tion is a crucial part of neighbor discovery, as multicast is used to discover nodes on a link.

Within IPv6, packet fragmentation is only carried out at the source node, this optimizes the operation of the IPv6 network as routers do not perform additional processing of the packet to suit the characteristics of the link. To enable this operation, ICMPv6 provides a network function of path maximum transmission unit (MTU) discovery. This function enables source nodes to discover an appropriate MTU for packets using a particular path to destination. Ensuring that the path MTU is the lowest along the path, no fragmentation will be required along the path by the routers.

2.1.2 Routing

There is a wide range of routing protocols suitable for use within an IPv6 enabled network. Routing protocols can be separated into two groups, interior gateway protocols (IGP) are routing protocols that operate within an autonomous system , and exterior gateway protocols, such as the Border Gateway Protocol (BGP) [19].

IGPs include protocols such as Routing Information Protocol new generation (RIPng) and Open Shortest Path First (OSPF) [20, 21]. These protocols can be further divided into to link-state and distance-vector routing protocols based on the methods used to manage routing information within the network. For link-state routing protocols such as OSPF and IS-IS, each routing node possesses information on the complete network topology. Each node then independently calculates the best next hop from it for every possible destination in the network using local information of the topology. The collection of best next hops forms the routing tables for the node. To discover shortest path, the OSPF protocol uses the Dijkstra's algorithm. For distance-vector routing protocols, each routing node does not possess information about the full network topology. Each node advertises its distances from other routing nodes and receives similar advertisements from other routers. Using these routing advertisements each routing node populates its routing table appropriately. Each node uses the Bellman-Ford algorithm to calculate shortest paths.

BGP maintains routing information about network reachability between Autonomous Systems. It is a path vector protocol, which does not use traditional IGP routing metrics but makes decisions based on path and network policies. BGP uses a transport layer protocol such as TCP to communicate routing information between two BGP peers. The BGP 4 protocol has been extended to enable compatibility with IPv6 [22].

2.1.3 Security

The initial IPv4 protocol was not developed with security in mind. As a result, applications are susceptible to a wide range of attacks such as Denial of Service. The IPv6 protocol has integrated into it an IP layer security framework IPsec, which is a flexible method of providing security to a wide number of network protocols and mechanisms. It provides two protocols within the framework namely Authentication Header (AH) protocol and Encapsulated Security Payload (ESP) protocol [23, 24]. AH attempts to guarantee connectionless integrity and the authenticity of origin by protecting the IP payload and all header fields of the IP datagram except for fields that might alter in transit. ESP provides origin authenticity, integrity and confidentiality protection of a packet. Unlike AH, the IP packet header is not protected by ESP. With such a facility available to applications and network functions built into IPv6, a wide range of attacks can be avoided, such as Denial of service, reply attacks, and wiretapping [25].

2.1.4 Mobility

As end nodes become increasingly mobile, the point of attachment to a network can change, possibly resulting in the assignment of a new IP address. The objective of Mobile IPv6 is to ensure this process can take place while maintaining connectivity between communicating nodes [26]. Mobile IPv6 requires that a home agent is always aware of the current care of address of the mobile node. This ensures that any nodes wishing to communicate with the mobile node only need to maintain the node's home address, as packets will be forwarded on to its new location. Route optimization is also possible, where a corresponding node establishes a direct route to the mobile node's care of address.

2.1.5 Extension Headers

Due to the fact that the role of the IPv4 headers is very important in the IP Protocol, this capability was kept in IPv6 as well. On the other hand the processing of some fields of IPv4 headers causes a performance loss by the packet forwarding. This feature was changed in IPv6, so that besides the mandatory main header the so called extension headers were added to provide flexibility and efficiency. These headers are attached to the datagram when needed. Therefore, the size of the main datagram header will be small, containing only the header fields that are necessary.

When extension headers are included in an IPv6 datagram, they appear one after the other following the main header. Each extension header type has its own internal structure of fields. More clear illustration is shown in Fig. 2.1 and Fig. 2.2.

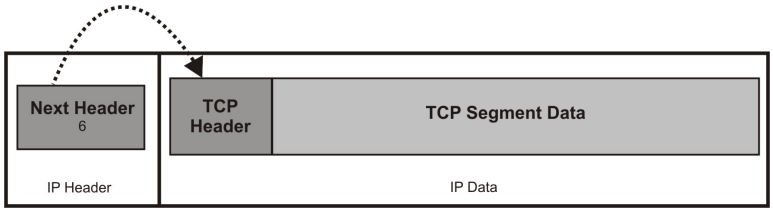


Figure 2.1: IPv6 datagram with no extension headers carrying TCP segment.

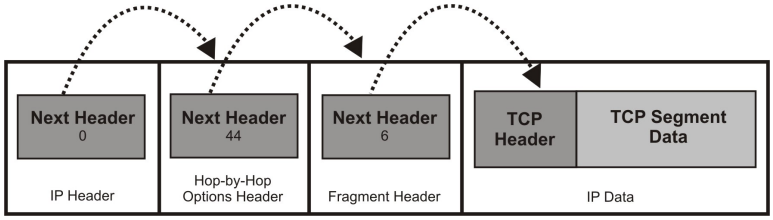


Figure 2.2: IPv6 datagram with two extension headers carrying TCP segment.

The maximum number of predefined extension headers is limited to 255, as the `Next Header` field contains 8 bits and one value is reserved for "no next header" information. If a node receives an unknown header, this header will be discarded. Each extension header is carrying options concerning different purposes. Headers may have different structure, but each of them must begin with `Next Header` and `Header Length` field. The placement of extension headers in the IP packet must be maintained according to a defined order. The types of extension headers and their orders are listed as follows:

- Hop-by-hop Option Header - used to carry information that must be examined by each node along a packet's delivery path.
- Routing Header - used to list one or more intermediate nodes through which packet is to be transmitted to its destination.
- Fragment Header - used by IPv6 source to send a packet whose size exceeds path MTU (fragmenting original packet).
- Destination Options Header - used to carry information to be examined by destination node(s) only.
- Authentication Header - is a security mechanism used to provide connectionless integrity, data origin authentication and protection against replays.

- Encapsulating Security Payload Header - provides confidentiality, data origin authentication, ensures connectionless integrity.

IPv6 Router Alert is a kind of hop-by-hop header, which is defined in RFC2711 [27]. A router that receives a packet with an active Router Alert header, can pass the payload of this packet into pre-defined application logic for processing. Router Alert is used in RSVP [28], where a router alert packet can be used to reserve bandwidth along a path.

2.2 Centralized Control in Next Generation Networks

In today's IP networks, the management of an AS is carried out through centralized network-level decision elements, while the control functions, such as routing, are performed in a decentralized fashion on individual routers. At a high level, network managers set the management policies, including configuring weights of the links, which indirectly control the selection of routing paths. The control of IP networks, however, is performed in a decentralized fashion using link-state routing protocols such as IS-IS and OSPF, and distance-vector protocols such as RIP. In both types of routing protocols, the forwarding path between two routers is the shortest path when considering the sum of the weights for each link along the path. When using link-state routing protocols, each router maintains a database of all links in the network and computes the routing paths using Dijkstra's shortest path first (SPF) algorithm.

This approach leads to a complex system that is difficult to manage and control. Motivated by the complexity of centralized management and distributed control, there have been research initiatives in centralized control recently, which advocate that the control of an AS should also be performed in a centralized fashion with direct control [29–31]: Instead of manipulating high-level management policies such as setting the link weights, a centralized network-level decision element controls the routers' forwarding decisions directly by constructing and distributing the FIBs to the routers. A centralized control scheme has several advantages, for example, the route decision process becomes simpler since a single decision plane replaces the management and control planes. The routing paths can be assigned directly by the network level decision element with more freedom, as opposed to OSPF networks which are more complex to administer and operate. In addition, with shortest-path routing in OSPF, it can be very difficult or even impossible for network administrators to set up desired routing paths by manipulating link weights.

There are also potential concerns, challenges and emerging research issues in the centralized control scheme. For example, the centralized network level decision element is a potential performance bottleneck which could constitute a single point

of failure. However, a logically centralized decision element does not exclude a distributed design. With a logically centralized, but physically distributed decision element, these challenges can be addressed.

2.3 The GANA Reference Model

The EU FP7 project EFIPSANS proposes a Generic Autonomic Network Architecture (GANA) [32] for depicting the network entities and their relationships for network management. In the GANA architecture [32], there are two important components: decision element (DE) and managed entity (ME). A DE is an autonomic management component, which has the similar role of the autonomic manager in the IBM-MAPE model [33]. A ME is the managed resource, which may be of a physical or logical one, for example, a device, memory, protocol or even an automated task. The information provided by the MEs can be used by DEs to manage the associated MEs. The actions taken by the DEs can enforce a policy or trigger the actions on the MEs. The MEs and DEs interact with each other to realize a control loop, where DEs use the supplied information from MEs to control and regulate the associated MEs.

2.3.1 Hierarchy of DEs

The GANA model can be applied to different levels of functionality and abstractions within node and network architectures. Inspired by the 4D architecture [30], GANA defines four levels of abstractions, *protocol*, *function*, *node* and *network* level, for which DE, ME and control loops can be designed.

Protocols Level DE

The concepts of a control loop of self-manageability may be associated with some implementation of a single network protocol. For example, protocols such as OSPF and TCP are known to have the control loop of self-manageability intrinsically implemented. This level is the lowest level of abstraction of functionality that can be associated with the manifestation of control-loops, as depicted in Fig. 2.3.

Functions Level DE

A higher level of abstraction than a single protocol is the network function level. This means the aspect of autonomicity may be addressed on the level of abstracted networking functions, such as routing, forwarding, mobility management, QoS

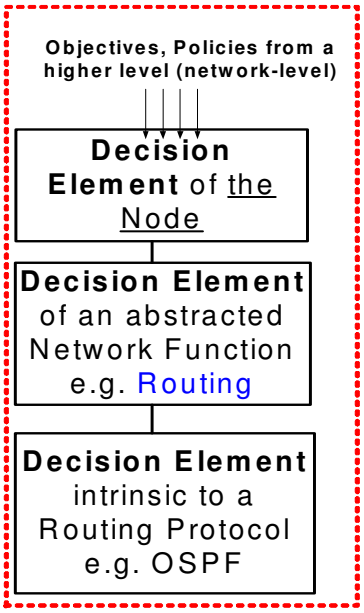


Figure 2.3: The GANA decision element hierarchy architecture [1].

management, etc. At such a level of abstraction, a group of protocols and mechanisms, which are considered to belong to the functionality of the abstracted networking functions and collectively wrapped by a function block, are managed by an assigned DE. For example, all routing protocols and mechanisms of a node are managed by a Routing_Management_DE, which is assigned and designed to manage only these protocols and mechanisms. The DEs operating at this level is called the "Functions-Level" DEs. Fig. 2.4 presents a set of DEs that must operate at the "Functions-Level". A sibling relationship means that the entities are created or managed by the same node level DE. The entities having a sibling relation can form other types of peer relationship within the node or with other entities hosted by other nodes in the network.

Node Level DE

On a higher level of autonomic networking functionality than the level of "abstracted networking functions", the node level DE is defined. Fig. 2.4 illustrate that the lower level DEs operating on the level of abstracted networking functions become the MEs of the main DE of the node. This means the node's main DE has

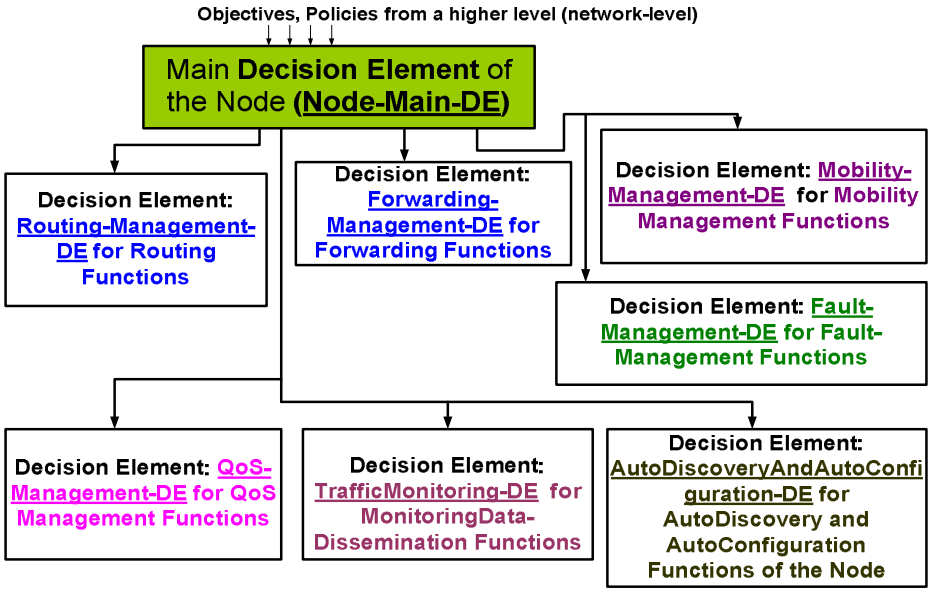


Figure 2.4: The GANA function level decision elements [1].

access to the views exposed by the lower function level DEs, and uses its knowledge of the higher level to influence or enforce the lower level DEs to take certain desired decisions, which may in turn further influence or enforce desired behaviors on their associated MEs, down to the lowest level of individual protocol.

Network Level DE

The next higher level of self-manageability above the "node level" is the "network level". There may exist a logically centralized DE or isolated decision plane that knows the objectives, goals or policies to be enforced by the whole network. The objectives, goals or policies may actually require that the node level DEs of the network export "views" such as events and state information to the centralized DE or decision plane. This is to allow the centralized Network DE or plane to manage the DEs of the nodes that may in turn have an effect of recursive decision changes on the lower level DEs of individual nodes, down to protocol level decisions.

2.3.2 The Functional Planes of GANA

GANA advocates the decomposition of the network control plane into decision, dissemination, discovery and data planes. These functional planes were defined in

4D architecture [30] to abstract the decision logic from the underlying protocols and mechanisms in the network, with the aim of reducing the complexity of today's control and management planes. The purpose of the definition of GANAs functional planes is to map todays protocols to the functional planes, and to assign diverse protocols and mechanisms to concrete DEs, which then play the role of the autonomic managers of the considered protocols and mechanisms.

Decision Plane

The decision plane within GANA, as apposed to 4D, makes all decisions driving the behavior of the node and network under control. Replacing today's management plane, the decision plane operates in real time on appropriate abstracted views of the states (for example, traffic, events, context and context changes, policies, capabilities) of the protocols, devices and network. All DEs at each of the defined hierarchy are part of the decision plane.

In contrast to the centralized DE described within the 4D architecture, GANA states that DEs within the hierarchy can form various relationships to aid decision making and coordination efforts. The relationships are stated as follows.

- A hierarchy relationship between DEs means that a lower level DE is managed by the immediate upper level DE (or viewed as an ME).
- A peering relationship between DEs defines the mechanism to communicate between DEs to exchange views or negotiations pertaining to the control of MEs, or requesting services from each other.
- A sibling relationship means that the entities are managed by the same upper layer DE. This means that sibling DEs can form peer relationships within the autonomic node or with other entities hosted by other nodes within the network.

Dissemination Plane

The dissemination plane consists of protocols and mechanisms that provide a robust and efficient communications substrate that is used to disseminate control information from the decision plane to the data plane, as well as the exchange of abstracted views of network status among DEs within the decision plane.

Discovery Plane

The protocols and mechanisms of the discovery plane are responsible for discovering the entities of the node and representing these logical identifiers consistently.

The functionalities of discovery plane include neighbor, service, capabilities and dependency discovery, and management of the relationship among them. The decision plane uses information obtained by the discovery plane to construct abstracted views of network status.

Data Plane

The data plane consists of protocols and mechanisms that handle individual packets based on the states, i.e., the output by DEs within the decision plane. These states include forwarding tables, packet filters, link-scheduling weights and queue management parameters.

2.4 Fault Management

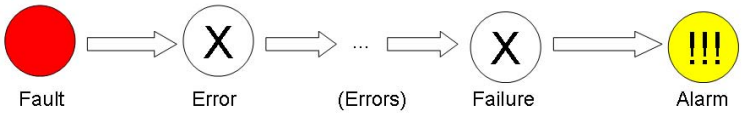


Figure 2.5: Fault-error-failure-alarm propagation [2].

The constantly increasing number of important applications and services makes the communication networks a safety critical infrastructure. It is of paramount importance that the future network has high reliability and is always operational. Therefore, fault management is a key component of the network management system.

Fault management naturally has to deal with incidents (faults/errors/failures) and alarms. As illustrated in Fig. 2.5, we define a fault as the root cause for the malfunctioning of a system, i.e., it is a structural, configurational or algorithmic cause of an erroneous state. Faults get further propagated as one or more errors. An error is a behavioral or computational deviation from correct pre-specified behaviors, values or conditions. An error is either a direct consequence of a fault or of another error, and can propagate further as another error or a failure. A failure is perceived as an error that becomes visible on the level of services, i.e., it is a discrepancy between the expected result of a service and the delivered one. In the following, we refer faults/errors/failures as incidents. Alarms are perceived as notifications concerning detected abnormal conditions. An alarm may or may not represent a fault/error/failure. For example, an alarm might be generated in case a particular threshold is being approached, which does not necessarily mean that a fault has been activated [2].

Fault management consists of fault detection, fault isolation and fault removal. Fault detection constitutes the process of detecting that a fault has been activated. A failure can be detected by the monitoring system. The process of fault detection gives the indication that a fault has occurred in the network. The process of fault isolation is dedicated to correlating the amount of detected incidents and generated alarms, and identifying the fault for the observed malfunctions. The important point is that only faults need to be corrected. Fault removal will resolute all errors and failures resulting from the fault. The combination of all these processes is denoted as fault management - the detection of faults, followed by the fault isolation and fault removal. There should be cooperative mechanisms implemented in nodes and the network to detect incidents, share knowledge about incidents, localize the root cause of the observed incidents, and finally remove the fault, throughout the operation lifetime of a node and the network as a whole.

Fault management provides the ability of the network to recover from some undesirable state and to "provide and maintain an acceptable level of service in the face of various faults and challenges to normal operation" [34]. The concept of fault management can be further divided into reactive and proactive fault management. A node or a network that implements reactive fault management is equipped with a set of components that can react to the presence of a fault by employing some fault-masking and/or remediation techniques [35]. An entity that implements proactive fault management, e.g., a component or a protocol, tries to predict and subsequently avoid future fault activations in the network [35].

2.5 Path-based Monitoring

Nowadays network monitoring systems are crucial to communication networks. They periodically collect network performance metric values, identify performance anomalies, and determine root causes for the problems. Their effectiveness and efficiency largely impact the quality of network services. The most important performance metrics include connectivity, delay, packet loss rate, location of congested network nodes, and bandwidth information.

There is a wide range of network monitoring techniques that can provide the necessary monitoring input to network management processes such as network planning and traffic engineering. In this section, we discuss a number of common network monitoring systems related to the path-based monitoring approach. These network monitoring systems can be classified to active measurement or passive measurement according to their measurement techniques, which are presented in the following sections. We also introduce the traffic matrix estimation by using path-based monitoring approach, and security issues related with intrinsic moni-

toring.

2.5.1 Active Measurement

Path Discovery

Path discovery is an important concept within large scale communications systems, and decisions, such as determining a route to a destination, can be based on such results. A common active based monitoring approach is based on using the traceroute tool [36]. Traceroute is used to measure the IP addresses and round trip time to every node of the path. The tracroute algorithm works by repeated sending UDP probe packets to a given destination with a progressively increasing hop limit (HLIM) and each probe uses a different destination port to distinguish itself.

However, traceroute can fall victim to a common mechanism in traffic routing, which is load balancing. As there are no standard mechanisms for dealing with load balancing over common network paths, traffic destined for a common destination may take an alternate route through a network. Traceroute operates by sending ICMP messages to nodes along the path to the destination node while incrementing the packet TTL field until the packet has reached its destination. As depicted in Fig. 2.6, packets sent across the network may not necessarily follow the same path. Under such circumstances, path discovery using traceroute can lead to inference of an inaccurate network topology, as depicted in Fig. 2.7. This error has been reported in the work of [37].

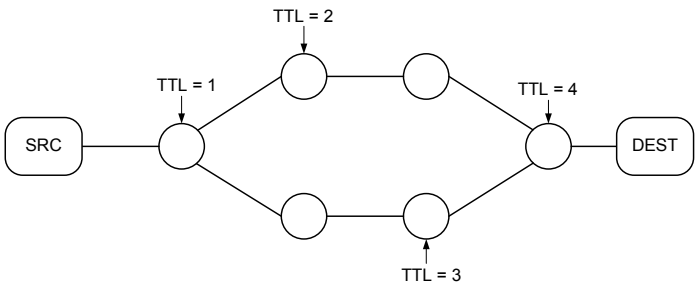


Figure 2.6: Traceroute path inference under load balancing.

Using traceroute has other inherent well-understood limitations. For example, it only shows the forward path and the reverse path itself is completely invisible. The reverse path could be completely different at every hop in the forward path.

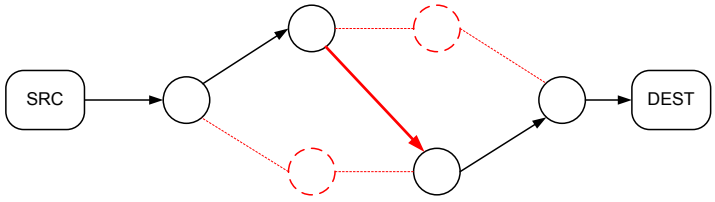


Figure 2.7: Traceroute inferred path with false link.

Pathneck

Pathneck aims to locate network bottlenecks along end-to-end paths on the Internet [40]. Pathneck is based on a probing technique called recursive packet train (RPT). The key idea is to combine measurement packets and load packets in a single probing packet train. Load packets emulate the behavior of regular data traffic, while measurement packets trigger router responses to obtain the measurements. RPT relies on the fact that load packets interleave with competing traffic on the links along the path, thus changing the length of the packet train. By measuring the changes using the measurement packets, the position of congested links can be inferred. The shortage of this work is that it could only locate the network bottlenecks.

PathChirp

PathChirp [41] operates by transmitting a set of packet trains called chirps. Each chirp is a set of packets of uniform size, which are transmitted at an exponentially decreasing time rate. The instantaneous chirp rate increases as the inter packet spacing decreases and when this instantaneous chirp rate reaches the rate of the available bandwidth, each successive packet will experience an increasing transmission delay as the path becomes congested. PathChirp operates on the basis that the instantaneous chirp rate of the first packet to experience increasing delay is equal to the available bandwidth. It improves this estimation by taking the average of multiple chirps to represent the available bandwidth. However, it can only obtain the available bandwidth information about the measured path.

Monitoring Systems Based on IPv6 Extension Headers

The utilization of IPv6 extension headers have been studied in [36, 47–49]. A method is proposed in [48] to accurately and efficiently determine the available bandwidth in IPv6 networks through the use of a IPv6 time stamp hop-by-hop ex-

tension header. In IETF Internet draft [36] the "Record Route for IPv6" option is defined as another new IPv6 hop-by-hop extension header. Based on that, a "Record Route for IPv6" mechanism is also described. In IETF Internet draft [47], IPv6 hop-by-hop extension header is used to record information along a communication path. The collected information includes interface attributes and statistics such as IP address, speed, number of transmitted packets and so on. These IETF Internet drafts were rejected mainly because of the lack of security consideration.

In Chapter 4, we present an intrinsic monitoring system based on a newly introduced hop-by-hop extension header, however, a newly introduced hop-by-hop extension header with a proprietary metric naming structure. This can limit the variety of metrics available for collection within the node. An alternative approach is to leverage existing managed object identifiers by using the SNMP based OID structure, which is presented in Chapter 5.

NSIS Ping Protocol

The IEFT NSIS group [50] is developing a general IP signaling protocol suite. The group is mainly working on signaling, nevertheless, it has done a lot of work related to path-coupled measurements [51–55]. The proposed state ping tool is able to gather information in a path based manner [54]. However, only limited information could be retrieved from the nodes along the path.

Cisco IOS IP SLA

Cisco IOS IP SLAs [56] measures network performance by generating of monitoring traffic in a continuous, reliable and predictable manner. It sends data across the network to measure performance between multiple network locations or across multiple network paths. It simulates network data and IP services, and collects network performance information in real time. The information collected includes response time, one-way latency, jitter, packet loss, voice quality scoring, network resource availability, application performance, and server response time. But this software is commercialized, which limits its widely adoption.

2.5.2 Passive Measurement

IPFIX

IPFIX (IP Flow Information Export) was created from the need for a common, universal standard of export for IP flow information from routers, probes, and other devices that is used by mediation systems, accounting/billing systems, and network management systems to facilitate services such as measurement, accounting and

billing [38]. The IPFIX standard defines how IP flow information is to be formatted and transferred from an exporter to a collector.

A flow is a set of IP packets passing an observation point in the network during a certain time interval. All packets belonging to a particular flow have a set of common properties defined as the result of applying a function to the values of:

- One or more IP header field (such as the destination IP address), transport header field (such as the destination port number), or application header field (e.g., RTP header fields (RTP-HDRF)).
- One or more characteristics of the packet itself (e.g., number of MPLS labels).
- One or more fields derived from packet treatment (e.g., next hop IP address).

Each sender will periodically send IPFIX messages to configured receivers without any interaction by the receiver. The actual makeup of data in IPFIX messages is to a great extent up to the sender. IPFIX introduces the makeup of these messages to the receiver with the help of special templates. A template is an ordered sequence of $\langle type, length \rangle$ pairs, which specify the structure and semantics of a particular set of information. Data records contain values of parameters specified in a template record. The sender is also free to use user-defined data types in its messages, so the protocol is freely extensible and can adapt to different scenarios. IPFIX prefers the stream control transmission protocol (SCTP) as its transport layer protocol [39], but also allows the use of the transmission control protocol (TCP) or user datagram protocol (UDP). Using IPFIX, devices like routers can inform a central monitoring station about their view of a potentially larger network. However, IPFIX only provides a device centric view of the flow information, information correlation is required for the central monitoring station to have a global view of the network status.

SNMP

The Simple Network Management Protocol (SNMP) is an IETF standard for the representation of management information, and the communication of such information between management stations and management agents for monitoring purpose [42–44]. SNMP version 3 as defined by RFC 3411 - RFC 3418 [45] is the current standard version of SNMP. It is widely deployed to monitor, control and configure network elements. It typically polls network devices to collect usage data. For providing resilience against packet loss, management applications retry requests when a response is not received. However, SNMP only provides a device centric view of performance. It cannot be used directly to measure some

performance metrics for a path, such as end-to-end delay, loss, etc. Furthermore, SNMP based monitoring systems generally require a large amount of processing and bandwidth resources due to the polling activities, which are generally seen as inefficient. A general survey of SNMP performance studies can be found in [46].

2.5.3 Demand Matrix Estimation

The traffic matrix of a telecommunications network measures the total amount of traffic entering the network from any ingress point and destined to any egress point. The demand matrix has been shown to be an effective method of representing network wide demand on the network from edge to edge [57–59]. There are a number of different approaches for calculating the demand matrix. The major concern is whether to calculate the demand matrix from direct measurement [58], or to use summarized sampling methods such as trajectory sampling [60]. These approaches generally focus on inference of traffic matrix of the network based on calculated edge to edge traffic demands and routing state models. Traffic demands are estimated at the flow level through monitoring traffic entering and exiting the network at the ingress and egress points respectively. For example, IPFIX [61] protocol has been developed by the IETF for flow based monitoring. Development of this protocol was based on the Cisco Netflow v9 suite [62].

The traffic matrix for the entire network can also be obtained by the path based monitoring approach. Previous work in [6–8] infers traffic matrix indirectly from observed link loads. Link load measurements are readily available via the SNMP, since every router maintains a counter of the number of bytes transmitted and received on each of its interfaces. For path-based monitoring, a set of probes need to be sent to measure the whole network. In [3], an efficient probe-path computation algorithm is proposed for minimizing probe traffic, while still obtaining maximum coverage of the network. A detailed example is given in Fig. 2.8.

2.5.4 Intrinsic Monitoring Security

The intrinsic monitoring method proposed in Chapter 4 attempts to address issues related to collecting and disseminating node specific monitoring data throughout an IPv6 network using the IPv6 extension headers as a carrier medium. The basic idea is that the performance of a data flow can be monitored between a source-destination pair by inserting specific information in the extension header of selected IPv6 packets in the data flow. By initiating an extension header at a source, and updating the extension header at intermediate nodes along the source-destination path, a collector can have a performance evaluation of select nodes in a network based upon the reported data in the IPv6 extension header. The advantages of

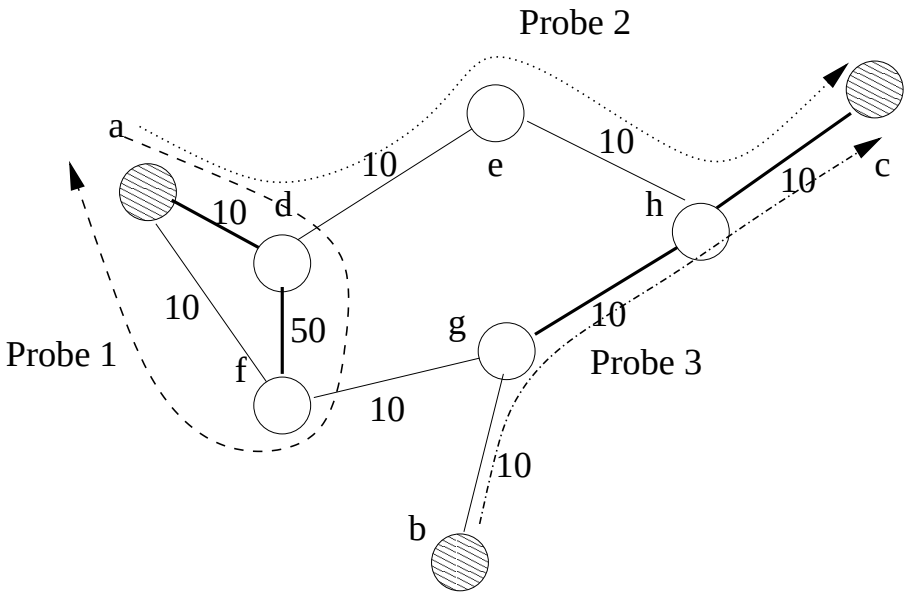


Figure 2.8: A network measurement covered by 3 probes [3].

such an approach can be invaluable to a network operator, offering a wide range of performance and accuracy enhancements over traditional SNMP and IPFIX based collection and dissemination methods. The approach is especially effective in collecting intermediate node specific information associated with the paths of particular monitored flows between source and destination pairs. Such a procedure can incur quite a lot of overhead for traditional approaches, such as the identification of appropriate nodes along the flow path and dissemination of data between each node and the associated collector. With the intrinsic monitoring approach, the packet is used to carry the monitoring data collection request and store the collected data for dissemination to the collector. However, there are associated security concerns regarding intrinsic monitoring with regards to privacy of information.

The Internet is operated by thousands of interconnected Internet Service Providers (ISPs), each ISP would like to keep their operational information confidential, such as IP address allocation, packet loss rate, etc. Moreover, monitored information is vulnerable to malicious attacks and data corruption due to the complexity of networks. Therefore, it is important to ensure security properties, such as data confidentiality and integrity of the monitored data carried within the IPv6 monitoring packet, and authentication of origin.

IPv6 has its own IP Security (IPsec) [63] protocol suite to provide security services at the IP layer. It enables communication nodes to establish mutual authentication, negotiate cryptographic keys, select required security protocols and determine the algorithms to use. Two fundamental elements of IPsec are Encapsulated Security Payload (ESP) [23] and Authentication Header (AH) [24]. However, neither ESP nor AH has been designed to encrypt and authenticate the augmentable IPv6 hop-by-hop monitoring packet. As previous work does not consider the implications of the security, deployment of such features has seen little progress. In Chapter 4, we propose an IPv6 hop-by-hop extension header and present a secure method for collecting and disseminating monitored information.

2.6 Loop-Free Forwarding Table Updates with Minimal Link Overflow

There are two types of tables used by routers: Routing Information Base (RIB) for routing and Forwarding Information Base (FIB) for forwarding. RIB is stored in the main memory of a route processor. The route processor receives and processes routing update messages and runs routing protocols such as OSPF and BGP to compute the RIB. Each RIB entry contains the destination IP prefix and associated route information. For example, BGP maintains full AS path and many other attributes for each prefix in RIB. FIB is derived from RIB and router configurations. It is stored in line cards which forward packet as fast as possible. Therefore, FIB usually uses high performance memory, which is more expensive and more difficult to scale. For each destination IP prefix, the FIB has an entry to store the next-hop IP, next-hop MAC address, and outgoing interface for fast data packets forwarding [64].

The possibility of forwarding loops occurrence and how to avoid and eliminate these loops have been well studied in the literature [10, 11]. However, the problem of transient loops during intra-domain routing convergence has only been studied recently [4, 65–69]. Still, there have been a number of studies based on different approaches to avoid transient loops. One approach is to delay the update of FIBs as a function of their distances from the failure [65]. The rule is that the nodes further away from the failure should update their FIBs first, and the nodes close to failure should update their FIBs at last. Based on the timer-based approach, a protocol-based approach has been proposed [66–68]. Instead of relying on timers, it relies on a protocol for routers to notify one another.

Another approach to avoid transient loops is to gradually update open shortest path first (OSPF) link weights [69]. In this approach, a link weight has to be increased a number of times before it can be shut down. At each link weight

update, it may result in FIB updates in all routers. However, this approach does not require modification of the existing routing protocols.

The approaches above have several limitations, as they only work in link-state routing protocols such as OSPF and they only work with single link failure. Therefore, an approach that addresses these limitations has been proposed [4]. The proposed approach analyzes the forwarding tables before and after the updates, and derives a loop-free update order. As it analyzes the resulting forwarding tables instead of studying routing protocols, the approach supports any routing protocol and any type of forwarding paths changes.

Using these approaches, transient loops will never occur and routers can reach one another during the entire transient phase. However, all these studies do not consider possible congestion and link overflow during the transient phase. Although the routers can reach one another, a large amount of traffic may flow over a few set of links during the transient phase, thus causing congestion in the network. The convergence time could take seconds to tens of seconds [4], and realtime traffic may hence still be severely impacted when link overflow occurs. In Chapter 6, we propose an update order that avoids transient routing loops and takes the congestion and link overflow into consideration to deduce the sequences of updatable nodes that cause minimal transient overflow.

2.7 Minimization of Network Wide Node Update Time

Updating networks is critical to ensure their proper operation. There are primarily two categories of reasons for network nodes to be brought offline, namely planned network changes and unplanned network changes. Planned network changes cater for node updates due to upgrades or repairs. Unplanned network changes occur due to node failures or malicious activity. The category we focus on in this work concerns planned network changes. In anticipating router bugs, Keller et al. [12] show that networks can be adversely impacted if router bugs are not dealt with effectively. They introduce a method of dealing with router bugs by adding a distributed voting strategy that can be used to verify router decisions. Affected routers can be detected by examining the results of the voting strategy. However, this approach requires re-implementation of software by the network equipment vendors. Our approach assumes that the network nodes need to be systematically updated.

Elastic Tree [70] is a network-wide optimizer that continuously monitors data traffic conditions and chooses the set of network elements that must stay active to meet performance and fault tolerance goals; then it powers down as many unneeded links and switches as possible. A variety of methods are used to decide which subset of links and switches to stay alive, including greedy bin-packer, topology-

aware heuristic, and other methods. In contrast, we optimize the rebooting time for the whole network where every node has to be rebooted once, find the updatable network nodes subset under the constraint of designated affected network traffic percentage. Further, we detail the tradeoffs associated with our approach, including impact on the number of updatable nodes in each iteration and affected traffic percentage.

2.8 Summary

In this chapter, we presented background information related to the research issues conducted in this thesis. The monitoring framework proposed in this thesis is based on IPv6 protocol suite, which is described in Section 2.1. In particular, IPv6 extension headers and Router Alert are emphasized. The centrally controlled and managed network is presented in Section 2.2, which can take advantage of employing the global view of the network and action synchronization to conduct efficient fault management. In Section 2.3, we presented the GANA network architecture and node architecture accommodating our monitoring and fault management schemes. The GANA architecture takes its influence from a number of clean-slate architectures aimed specifically at reducing the complexity of network management by refactoring the traditional management and control planes within the network. For the fault management, we introduced basic concepts in Section 2.4. The related work of path based monitoring, loop-free updates of forwarding tables of routers and minimizing network reboot time were discussed in Section 2.5, 2.6 and 2.7 respectively. In next chapter, we present our monitoring and fault management framework.

Chapter 3

Framework of the Next Generation Monitoring and Fault Management

To deal with the challenges discussed in Chapter 1, this chapter presents the framework of monitoring and fault management. It is based on the Generic Autonomic Network Architecture (GANA) [32] architecture developed by the FP7 EFIPSANS project, which is introduced in Section 2.3. GANA sets the principles and guidelines which need to be followed by our monitoring and fault management architecture. However, GANA is a generic model, here we demonstrate how it can be instantiated and discuss how monitoring and fault management within the network can be specified based on this model to instill desired behaviors. The monitoring and fault management components of the architecture required inside an network node are described here and their interfaces are also specified. Finally, we give a brief introduction to the research work conducted with respect to path based monitoring and fault management.

3.1 The Monitoring and Fault-Management Architecture

The two major components that constitute the core of the proposed architecture are discussed - the monitoring decision element and the fault management decision element. The basic relationship between these DEs is illustrated in Fig. 3.1. The monitoring DE can supply information about detected incidents or network status for processes of fault management. The fault management DE is responsible for repairing the damaged network infrastructure by diagnosing the underlying root cause and removing it, namely fault isolation and fault removal. Additionally, the fault management DE is the one that implements proactive fault management mechanisms based on estimations about the current status of the network.

Fault management is triggered whenever symptoms indicating the presence of faults in the network are detected. The corresponding process of incident detection is realized by monitoring components inside the nodes. Upon detection of

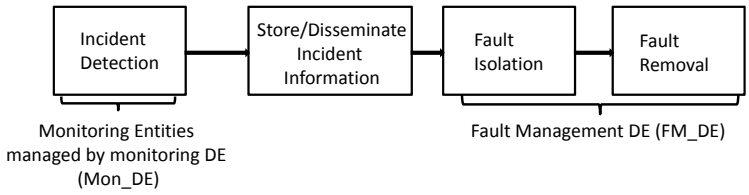


Figure 3.1: The monitoring and fault management architecture.

particular symptoms, the monitoring functions report their observations to a set of common information repositories, which in turn notify the fault management DE about the detected network status. The node repositories for alarms and incidents are considered as the consolidating point for information related to incidents in the node and the network as a whole. After the fault management DE has been notified about the incident, it performs fault isolation by employing the features of its embedded fault isolation functions. These fault isolation functions aim to find the root cause behind the observed symptoms based on fault propagation models [2]. Once the fault isolation is successfully completed, the fault management DE removes the isolated root cause by employing the fault removal function block. The fault removal functions perform actions such as reloading a functional entity, rebooting a node/device or reconfiguring a functional entity. After executing the intended fault removal actions, the fault management DE ensures that the decision taken is adequate and the source of the problem is successfully eliminated, by means of the feedback provided by the monitoring functions.

For the proactive fault management, fault management DEs request the monitoring DE to start different type of monitoring techniques and convey the obtained information back to the requesting DEs. This concept is similar to the on-demand-monitoring presented in [71–73]. Diverse parameters, such as sampling rate, the place of the information repository and the monitored traces, should be provided by the fault management DE during the requesting phase. Based on the monitoring information, fault management DE could generate actions and policies for the future fault management purpose.

The data correlator is responsible for collect information about stored data in node and network level, and makes it available to other DEs or MEs. According to the network topology, monitoring data may be either distributed across the network, or stored in a centralized database. The monitoring DE is aware of the data collection and dissemination scheme that is selected, and acts as an information supplier to other DEs, providing them with guidelines for data acquisition and dissemination.

The data correlator may check the quality of the disseminated context. Various parameters may be used to characterize the quality of information from different perspectives, such as freshness (i.e., time passed since the creation of the content), precision in the estimation, scope (e.g., local, cluster or network level), trust-worthiness, privacy, retrieval time from the content requestor,replication degree, volatility, etc.

According to the existing conditions in the network, the data correlator may provide guidelines with regards to the need for update or not in the existing data. For example, according to the freshness of data, the more fresh data will be stored in the database; according to the scope of data, it has to be stored locally in the node, or a central or distributed repository; according to the precision requirements, the data correlator may decide the frequency of storing or updating data. Thus, acquired context is complemented with attributes that are useful towards the process of their correlation. According to the attributes' values, data is correlated with existing data.

3.1.1 Monitoring Decision Element

Monitoring information is a fundamental part of a wide number of management activities such as QoS management, routing and mobility. For fault management, the monitoring systems play a crucial role for incident detections. In this section, first the monitoring requirements of monitoring DE and monitoring managed entities (MEs) used to monitor the network are described. Then, the architecture of the monitoring DEs that collect information from their MEs is presented.

Monitoring Requirements

In general, the monitoring DE should provide the possibility to acquire information about diverse metrics in a network. The measurement requirements (MRs) are summarized as follows.

- MR #1: One-way delay.
- MR #2: Round-Trip delay.
- MR #3: Available bandwidth towards some recipients.
- MR #4: CPU utilization (e.g., along a path).
- MR #5: Utilization of the queues in the routers along a path.
- MR #6: Number of dropped packets on different layers, e.g., dropped packets in the TX and RX buffers of a NIC.

- MR #7: Number of packets in a TCP session that were not successfully delivered and must be retransmitted (this can be realized based on counting the duplicate ACKs in a flow).
- MR #8: Number of changes of the congestion control window in a TCP session.
- MR #9: Number of changes of the receiver window in a TCP session.
- MR #10: Dependencies between components within the network.
- MR #11: Hardware status such as CPU/chassis temperature, internal fan status, voltage instability, or hard drive status.
- MR #12: Signal quality (bit error ratio, packet-loss, etc.).
- MR #13: Software status such as system/CPU/memory load.
- MR #14: Anomalies such as memory leaking, file descriptor leaking, unreleased file locks and data corruption.
- MR #15: System and kernel errors.

Monitoring Decision Element Requirements

The configuration and optimization of the monitoring activities may involve high operational overheads, the monitoring decision element is responsible for orchestrating monitoring MEs so that the node-level policies are fulfilled and specific functions are realized. In other words, the monitoring decision element is responsible for the configuration of the monitoring protocols and mechanisms, so that other network function DEs within a node and within the network can guarantee that relevant and sufficiently accurate information is available to conduct their control processes. The monitoring DE should also be able to delegate the control of any MEs implementing specific monitoring mechanisms or actions. In summary, the monitoring DE is responsible for monitoring actions or mechanisms such as:

- Monitoring coordination: Schedule monitoring measurements in such a way that they do not degrade other services (such as QoS or mobility services) or negatively affect collected results.
- Topology discovery: Collect information necessary for creating the topology of the network and publish topology data in an appropriate format to other DEs or MEs.

- Data generation: Activate monitoring measurements in the node in accordance with provided services and network topology.
- Data aggregation and transformation: Process any collected data in order to efficiently store them or produce more valuable results by, for example, data mining.
- Data storage: Store and publish any historical data collected via measurements for later analysis. Monitoring data may either be distributed across the network or stored in a centralized database.
- Data publication and exchange: Make stored data available to other nodes and leverage the search capabilities.
- Service inventory: Identify services supported in the network and select which need to be monitored [2].

Monitoring Decision Element Architecture

The monitoring DE resides in the function level of the GANA architecture and controls all the actions of a network node related to monitoring. It also interacts with the rest of the DEs or the MEs residing in the same or other network nodes, whenever these entities require specific monitoring actions to be taken or monitoring mechanisms to be activated. Specific MEs will implement specific monitoring functions based on decisions made by the monitoring DE. In Fig. 3.2, the monitoring DE architecture and some monitoring MEs associated with specific monitoring functions are shown as follows.

- IPFIX_ME (IP Flow Information Export): The monitoring DE will monitor the collected information to ensure what is being collected is in line with the requirements of the requesting network functions and applications, and to re-configure the IPFIX device appropriately.
- PSAMP_ME (Packet Sampling): PSAMP can be configured based on the requirements of the requesting network functions and applications through the monitoring DE. The monitoring DE will ensure that only minimum amount of information is collected and analyzed.
- PATH_ME (Path based monitoring): The path based monitoring tool is to monitor the network status between two points of a network. As this tool is an active form of network monitoring, the volume of traffic being generated must be controlled.

- EB_ME (Effective bandwidth measurement): This ME is used to accurately measure effective bandwidth of aggregated traffic flows with common QoS targets, such as, packet delay or loss rate. This tool depends on a number of configuration settings that can be modified to suit various traffic conditions. The monitoring DE will manage this tool based on the requirements of the requesting network functions.

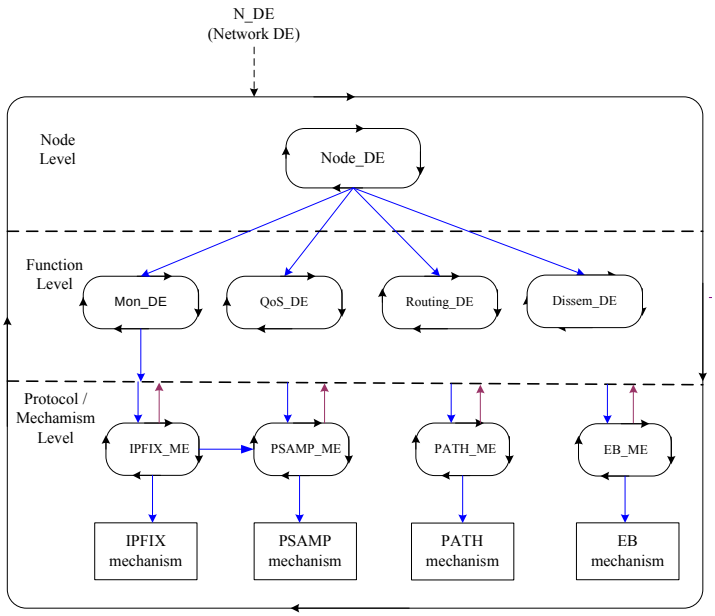


Figure 3.2: Monitoring decision element architecture.

3.1.2 Monitoring Decision Element Interfaces

The interfaces of monitoring DEs are illustrated here. The monitoring DE in a network node may establish a peer relationship with monitoring DEs on other network nodes in order to exchange relevant monitoring information. A peer interface is defined for exchanging information between monitoring DE realized in different nodes, as shown in Fig. 3.3.

All the monitoring MEs at the protocol level are under the control of the monitoring DE. However, when a specific DE at the function level, such as the fault management DE, requires a monitoring ME to be activated, it can request it directly as shown in the Fig. 3.4 (interaction a). On the contrary, the DEs at the

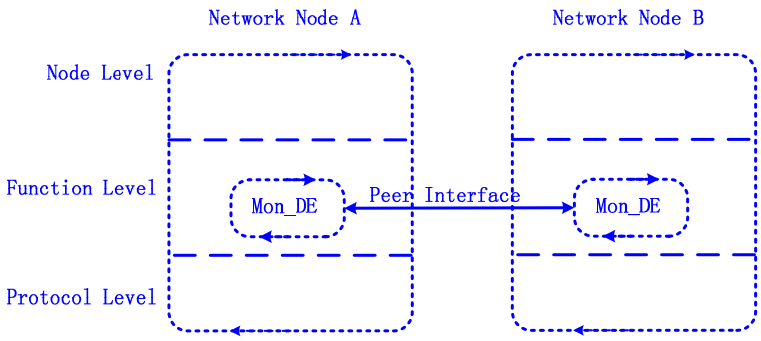


Figure 3.3: Monitoring decision element peer interface.

protocol level can control monitoring MEs only via the monitoring DE and not directly (interaction b). These restrictions are imposed in order to avoid conflicting actions, because monitoring actions consume network or node resources and may also affect other services, for instance, bandwidth aggressive measurements may cause QoS degradation to legitimate traffic.

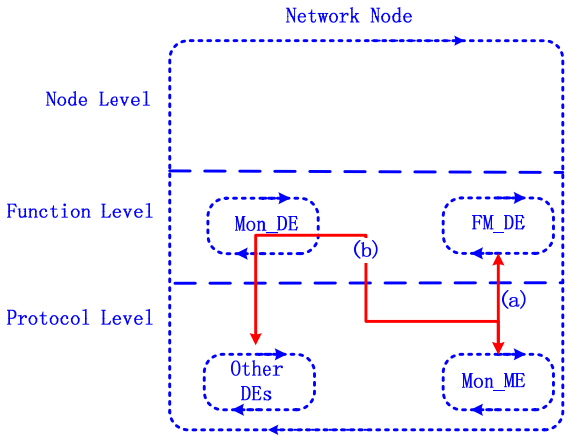


Figure 3.4: Interaction between decision elements and monitoring managed entities.

When a specific DE, such as the fault management DE, requires a monitoring ME to be enabled in another node, it has to request it from its local monitoring DE using the horizontal interface. The local monitoring DE interacts with the remote monitoring DE through its peer interface, as shown in Fig. 3.5. The local and remote monitoring DEs are responsible to resolve any conflicts with other local

behaviors in the node.

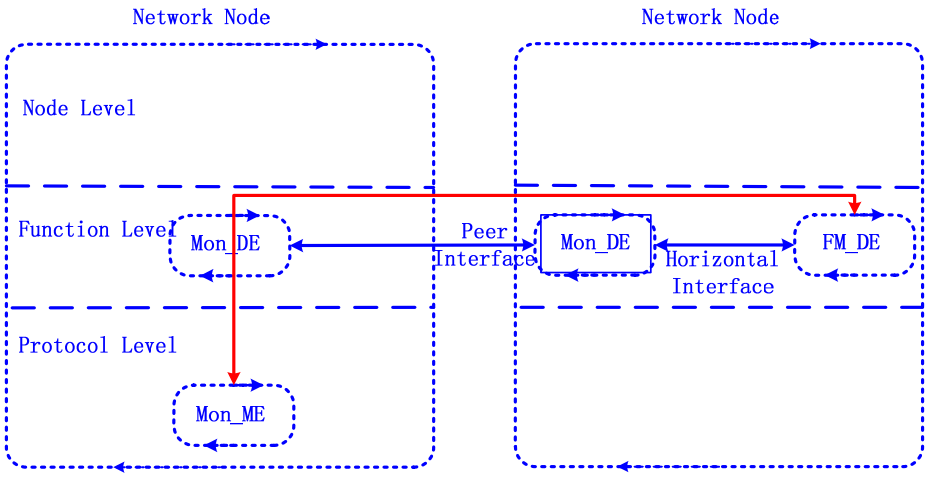


Figure 3.5: Monitoring decision element horizontal interface.

It should be noted that one monitoring ME usually participates in more than one control process under the authority of different DEs. In most cases, the ME operates as an information supplier for all the associated DEs. However, there are cases where a ME can be controlled by two different DEs, which interact with each other in order to avoid conflicting operations and ensure optimal operation of the ME. In some cases, the conflict is resolved at the node level implemented by the node level DE.

3.1.3 Fault Management DE

The fault management DE (fault management DE) is the entity which realizes fault isolation and fault removal during the operation of the network. The fault management DE at the node level inside the GANA reference model is shown in Fig. 3.6.

The distributed collaboration of fault management DEs should strive to converge on having a consistent similar view of the knowledge of the network status. After converging, each and every fault management DE on the nodes proceeds with the process of fault isolation and removes the isolated fault, if it is directly or closely related to the local node that the fault management DE is responsible for.

To solve network problem by employing the global view of the network and synchronize actions, which are beyond the competence of the corresponding node level fault management DE, a centralized network level fault management DE is

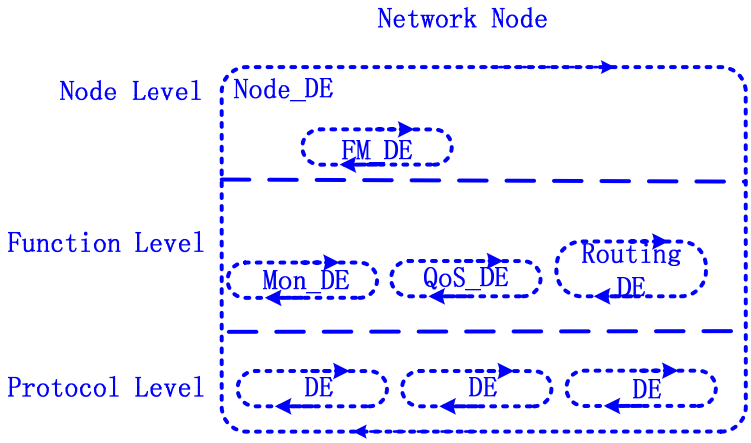


Figure 3.6: The fault management decision element as a part of the node level decision element.

required to act in the best possible way, which implies the constellation of entities as shown in Fig. 3.7. The centralized network level fault management DE should get informed about network status. It is always one of the relevant nodes to which incident descriptions are sent and which keeps dynamic network information such as traffic matrix.

3.1.4 Monitoring Decision Element and Fault Management DE Relationship

Fault management components have close interaction with monitoring entities, which constitutes an internal mechanism of the node leading towards a highly reliable network. The information flow between the fault management DE and the monitoring DE is bi-directional, the different directions of information flow are described below.

- Monitoring DE $\rightarrow$ Fault management DE: The monitoring DE collects information through its managed entities, which is required by the fault management DE as an input to its internal processes. The fault management DE can request the monitoring DE to gain access to data that is already stored in its internal repositories, or to initiate a new data collection process that will be set up by the monitoring DE on behalf of fault management components.
- Monitoring DE$\leftarrow$ Fault management DE: Fault management is expected to communicate with the monitoring DE in order to inform the it about network

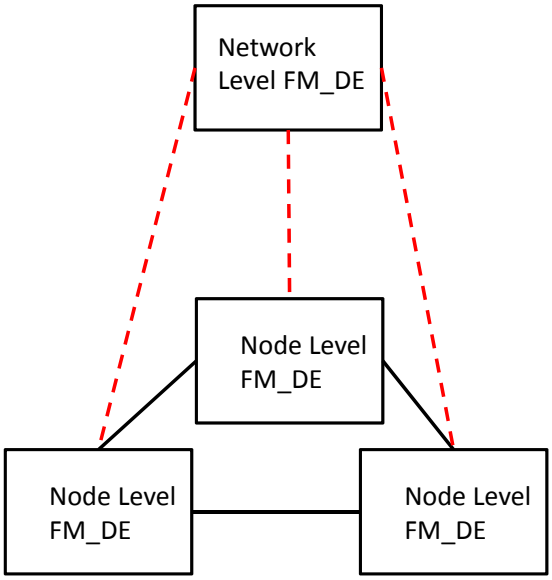


Figure 3.7: A distributed overlay of fault management decision elements and the centralized network level fault management decision element.

information which need to be monitored in the network. Subsequently, after having received this information, the monitoring DE is expected to start appropriate monitoring MEs and extract information to the information repositories.

3.2 Path ME

The constantly growing Internet applications have created a massive increase in demand for bandwidth, performance, predictable QoS and differentiated network services. Network performance is important for many applications. Simultaneously the need has emerged for measurement technology that will support this growth by providing IP network managers with effective tools for monitoring network performance. Knowledge of the up-to-date network status is critical for numerous important network management tasks, including application and user profiling, fault management, proactive and reactive resource management, and traffic engineering, as well as providing and verifying QoS guarantees for end user applications.

Existing network monitoring tools can be divided into two categories. Node-oriented tools collect monitoring information from network devices such as routers,

switches and hosts using SNMP/RMON probes [42] or the Cisco Netflow tool [62]. These are useful for collecting statistical and billing information, and for measuring the performance of individual network devices (e.g., link bandwidth usage). However, these tools cannot monitor network parameters that involve several components, such as link and end-to-end path latency. The second category contains path-oriented tools for connectivity and latency measurement, e.g., ping, traceroute [74], skitter [75], and tools for bandwidth measurement such as Pathneck [40] and PathChirp [41].

A novel network monitoring architecture for collecting and disseminating monitoring information has been developed using the IPv6 hop-by-hop packet header itself as a carrier medium [76], referred as intrinsic monitoring. The technique can collect specific node related information from each node along a path between the source and destination, and report this information to a designated collector. This approach can easily be used to collect information relating to monitoring requirements. The general approach followed is to use a newly defined IPv6 hop-by-hop option to request intermediate nodes to insert requested information, such as ingress IPv6 addresses, CPU utilization or current queue properties, into the header data field. This information can then be collected and analyzed at a collection point. Analysis can explicitly reveal detailed information relating to the path, which the packet traversed. This information can be collected over very short time scales. Analysis can assess whether various suspicious characteristics are being detected and will be reported as incidents.

Path-based monitoring, which is implemented as the path ME managed by the monitoring DE, provides an approach to collect performance metrics on a path within a network between any two router that aid in the assessment of the performance of any individual path. It can be an active monitoring, where the management station continuously measures a set of characteristics of a set of specific paths. It can also be a reactive monitoring, where the management station needs information about the network state in order to react to certain alarm conditions that may develop in the network. Such conditions usually indicate either a fault, which is the root problem of system disorder, or some anomalous behavior, which cause a fault later.

3.3 Traffic Matrix Related Fault Management

Recent years have seen an enormous growth in demand for Internet access, with applications from personal use to commercial operations. Several of these applications are sensitive to a “quality” of packet delivery. For instance, although archiving data transfer can tolerate long delays, VoIP is very sensitive to latency.

When network failures occur, guaranteed services need assurances about reliability in terms of continuity and quality of services maintained by, for example, fast rerouting in MPLS networks. However, the network topology and capacity may be insufficient to meet the current demands. At the same time, there is mounting pressure for ISPs to provide QoS in terms of SLAs with customers. As a sequence, there are significant network management challenges that arise from the combination of rapid growth and increased complexity of the network. All of these issues point to the importance of making use of traffic matrix and existing network resources to meet the demands of prevailing traffic.

After obtaining the traffic matrix according to some approaches suggested in Section 2.5.3, two emerging research issues within the context of fault management were investigated. The aim is to quickly reboot the network nodes and update the network forwarding tables for every node in the event of failures while maintaining QoS requirements. The work on loop-free updates shows how routers can be updated in an order to avoid transient loops and link overflow. In particular, this approach can be applied to any routing protocols and any type of forwarding path changes. Moreover, the study on the network wide reboot provided an algorithm to efficiently reboot the whole network. Performance results show a significant improvement in terms of overall upgrade time and scalability, which indicates the properties of this efficient solution with respect to the minimization of impact on network traffic. These mechanisms are embedded into the fault management DE to conduct effective management. More detailed information is shown in the following chapters.

3.4 Summary

In this chapter we presented the framework of monitoring and fault management conforming to the GANA architecture. A monitoring DE provides monitoring information as input to the fault management DE, the fault management DE is responsible for fault isolation and fault recovery. After fault recovery, fault management DE utilizes the monitoring information provided by monitoring DE to verify if the fault recovery is successful. Monitoring DE, fault management DE, and their relationship were illustrated. Our research work mainly focuses on the path monitoring ME, and fault recovery schemes located in the network level fault management DE, which can take advantage of the global network view and action synchronization to effectively conduct fault recovery. In the next chapter, we introduce our path based monitoring mechanisms.

Chapter 4

Intrinsic Monitoring within an IPv6 Network: Relating Traffic Flows to Network Paths

Network monitoring is fast becoming a critical component of network management systems, in particular, management processes such as network planning and traffic engineering. The efficiency of a network depends on how well the network is planned and what traffic engineering strategies are deployed. A clear understanding of how traffic traverses the network and associated properties of intermediate nodes along paths within the network must be known accurately to ensure effective network planning and traffic engineering strategies are deployed. A large body of research has been published on the importance of the traffic matrix to network planning, which in essence attempts to establish a relationship between traffic flows and network paths within a network domain, see for example [77–80].

Passive based traffic matrix estimation approaches generally focus on inference of traffic flows through the network based on edge to edge traffic demands and routing state models. Traffic demands are estimated through monitoring traffic entering and exiting the network at the ingress and egress points respectively. Based on internal network routing state, the trajectory of traffic flows can be inferred [77, 78]. Such approaches are generally time consuming and inaccurate as dynamic routing, link failures and load balancing policies can all affect the trajectory of traffic flows through a network, thus having an impact on the dimensioning of the network [79, 80]. Active approaches such as traceroute [36] inject control traffic into the network to identify paths between nodes across the network. However such approaches cannot directly relate the trajectory of control traffic with actual user data traffic [37]. Packet sampling based traffic matrix inference [60] uses hash functions to track the flow of traffic through nodes within a network.

In this chapter, we present a novel network monitoring approach for IPv6 networks, referred as intrinsic monitoring. The approach advocates collecting network monitoring information within the IP packet header structure as the packet traverses nodes within the network. This approach can allow direct monitoring of both the

spatial flow of traffic through the network, as well as node related information of the intermediate nodes along the traffic flow path. We offer a detailed description of the approach and demonstrate its effectiveness in reducing the overhead of path discovery in comparison to the popular traceroute tool. We also discuss the implications of such an approach in comparison to other techniques developed for relating traffic flows to network paths, such as the process of estimating the traffic matrix.

The chapter is arranged as follows: Section 4.1 presents our intrinsic monitoring framework which describes the proposed IPv6 hop-by-hop monitoring header format and capabilities. The section also discusses a number of important considerations that must be accounted for when using the approach, such as packet processing and MTU. Section 4.2 provides an in depth discussion of how intrinsic monitoring can be used within a number of traffic monitoring scenarios, such as spatial traffic flow monitoring and path discovery. Section 4.3 offers an initial experimental evaluation of the approach, comparing both traffic overhead imposed on the network and total delay in retrieving monitoring results, to the traceroute active monitoring tool used for path discovery. Section 4.4 presents the security consideration for the intrinsic monitoring approach. Finally, Section 4.5 summaries the contributions of the work and discusses future work that will be taken.

4.1 Intrinsic Monitoring Framework

We now present the intrinsic monitoring framework. The main enabler of this approach is the introduction of an additional IPv6 hop-by-hop extension header.

4.1.1 Hop-by-Hop Monitoring Header

In accordance with the IETF RFC 2460 [81], the IPv6 hop-by-hop extension header must be processed by every node along a packet's delivery path. It can be used to carry optional information that must be examined by each node. In our proposed approach, the extension header is composed of a monitoring header and a set of collected performance metric records. This defined header format is shown in Table 4.1.1. Each field in the header is defined in Table 4.2 (and the tables therein).

It is flexible to use the IPv6 hop-by-hop extension header to collect monitoring data. First, the *hop-by-hop monitoring extension header* can indicate all performance metrics it is interested in collecting. Example identifier values could be number of packets forwarded, timestamps, congestion points, etc. Second, it can also specify the nodes it is interested in monitoring. Routing header options can be used together to monitor designated intermediate nodes. Third, an IPv6 address

Table 4.1: IPv6 Hop-by-hop Monitoring Extension Header Format

Next Hdr	**Hdr Ext Len**	**Option type**	**Option Len**
Sequence Number		**Monitoring Header Len**	
Record Count	**Num of Metrics**	**Flags**	**Record Hdr Length**
Identifier 0	**Identifier 1**	**...**	**Padding**
IPv6 address(optional)			
Num of supported Metrics	**Record Length**	**Identifier 0**	**Identifier 1**
Identifier 2	**Identifier 3**	**...**	**Padding**
Identifier0 Data			
Identifier1 Data			
...			
IdentifierN Data			
...			
Num of supported Metrics	**Record Length**	**Identifier 0**	**Identifier 1**
Identifier 2	**Identifier 3**	**...**	**Padding**
Identifier0 Data			
Identifier1 Data			
...			
IdentifierN Data			

can also be included in the *hop-by-hop monitoring extension header* to indicate a specific node and only the information from this node be collected. Finally, it is also possible to collect all the intermediate nodes information because IPv6 hop-by-hop extension header will be processed by every router along the path.

4.1.2 Processing Procedures

In a typical scenario, there is one ingress node, one egress node, and multiple intermediate nodes as shown in Fig. 4.1. The procedures can be summarized as follows: the management node initiates monitoring process by sending the ingress node a monitoring trigger packet, which includes the interested monitoring performance metrics. The ingress node creates a hop-by-hop monitoring extension header, which indicates the interested performance metrics to be collected.

An intermediate router receives the packet with the hop-by-hop monitoring extension header, it recognisees it is a monitoring packet. After decapsulating the packet, it collects its performance metric values according to the requirements and generates a monitoring record. It then inserts this record into the header data field of the hop-by-hop extension header. Finally, the egress node receives the monitoring packet and processes records one after another. The monitoring record will be processed by the egress node one by one until it finishes or error occurs. Then the collected information will be reported to the collector.

Table 4.2: Header Field Definitions

Option Type	**8-bit integer identifying this option as the monitoring information collection option. The IPv6 option must start with 001 indicating that routers should skip this option if they do not support it and that the data in this option may change en route to the destination.**
Option Len	**If the opt length is 0, then the monitoring header field is used to define the length of the monitoring header, else it indicates the monitoring header length in bytes.**
Sequence number	**16-bit unsigned integer for each flow used to distinguish this packet from other monitoring packets.**
Monitoring header length	**16-bit unsigned integer used to indicate the length of the monitoring header in bytes.**
Record Count	**Number of reporting nodes.**
Num of Metrics	**Number of performance metrics this packet would like to monitor.**
Flags	**See Flags definitions (Table 4.3 and 4.4).**
Record Hdr Length	**The length of the record header.**
IPv6 address	**Optional field, in case only one specific node information is needed, this IP address indicates this specific node's IP address.**
Num of supported Metrics	**It indicates how many performance monitoring metrics are supported on this hop. In reality, this hop may not be able to support all the metrics indicated.**
Record Length	**The record length in 8-bytes.**
Identifiers	**Identifier Value.**
Identifiers Data	**Monitored Data.**

Table 4.3: Flag Format

I	T	RH	S	R	Resv

Table 4.4: Flag Definitions

I	**Interface, 2 bits. If I = 1 then incoming interface IP address should be collected; if I = 2 then outgoing interface IP address should be collected; if I = 3 then both incoming interface and outgoing interface IP addresses should be collected.**
T	**Timestamp, 2 bits. If T =1, time stamp at incoming interface should be collected; if T = 2, time stamp at outgoing interface should be collected; if T = 3, time stamp at both interfaces should be collected.**
RH	**Use together with routing headers to collect information from specific nodes where the addresses are indicated in routing headers.**
S	**IP address is indicated. Information should be collected only for this IP address.**
R	**R = 0 indicates that the router should use a normal resolution (1 millisecond or less) while R = 1 indicates a higher resolution finer than 1 millisecond is desired if available.**

4.1.3 MTU Considerations

If adding information to the packet would make it exceed the MTU, the router does not add the information but moves the collected monitoring information to the payload of an reporting ICMP message and sends it directly to the management node (without collecting any more information). The collected monitoring information to date is removed from the monitoring packet and the monitoring packet is forwarded along its usual path to egress node collecting the required information. The management node will have to know how to combine these two (or more) records.

MTU discovered by the sender can also be included into the header to facilitate the intermediate router to detect packet size. The intermediate router can dump the collected data records to an ICMP packet which will be sent back to the management node and the packet will continue being forwarded to the destination.

4.1.4 Connections Between Collector and Routers

We can possibly look at a reliable TCP connection to send and receive both between management node and ingress node, and between egress node and management node. One possibility could be that the management node sets up a monitoring ses-

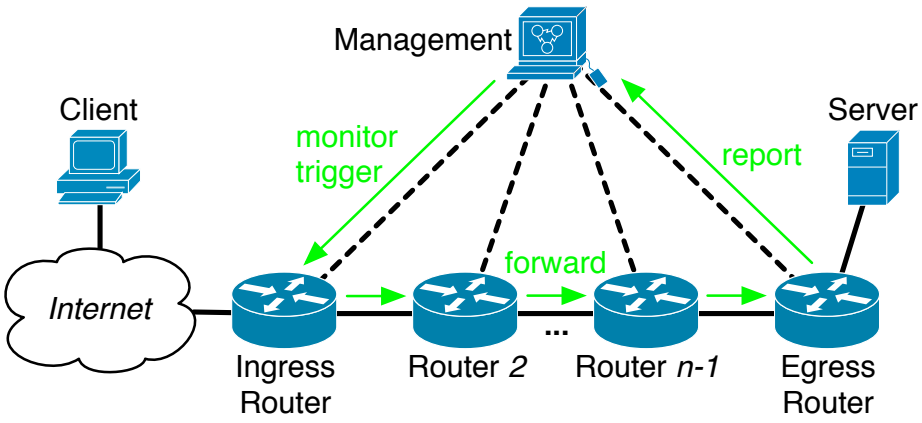


Figure 4.1: Packet processing procedures.

sion where one message is sent to ingress node with instructions for the monitoring session and details on monitoring data to be collected and frequency of packets to be pushed into the network between ingress node and egress node.

Another approach is to use destination option extension header to communicate between the management node and the end nodes of the path. In contrast to the IPv6 hop-by-hop extension header, which must be processed by every node along a packet's forwarding path, the destination option extension header will only be processed by the destination node. It could be used for trigger packet carrying same monitoring information as the above message, and report packet back to management node.

4.2 Application of Intrinsic Monitoring

4.2.1 Spatial Flow of Traffic

We have already presented approaches to monitor the spatial flow of traffic through a network and highlighted its importance to traffic engineering processes. In this section we discuss how such monitoring can be performed directly with our proposed intrinsic monitoring framework. By identifying the type of traffic flows to be monitored within the network, the edge node can identify which packets the monitoring header is to be inserted into. To ensure that not all packets within this flow are subjected to monitoring, a sampling strategy can be employed, such as the approaches outlined by the PSAMP group [82, 83].

As the selected packets move from node to node, the path will be recorded into

the hop-by-hop extension header data field. When the monitored packet reaches the egress edge of the network, the hop-by-hop option is removed along with all collected data, and the packet is routed onwards to its final destination. By using this approach, we can guarantee that the path recorded in the packet is the exact path taken by the traffic. We can then perform a comparison over a set of packets within the flow to analyze load balancing between paths within the network.

In comparison to method proposed by [60], no additional information regarding the characteristics of the flow need to be communicated to the intermediate nodes, thus reducing the overhead associated with setting up such a monitoring process. Trajectory sampling can also suffer from scalability issues as the approach requires configuration of each intermediate node along the path.

4.2.2 Path Discovery

Our approach of intrinsic monitoring can easily overcome the issue of discovering network paths under load balancing scenarios. As demonstrated in Fig. 4.2 as the intrinsic monitoring hop-by-hop option is processed by an intermediate node, the IP address of the ingress interface is recorded into the hop-by-hop header data field. The packet is processed at each intermediate hop along the path. When the packet reached the egress node, the stack of IP addresses stored within the hop-by-hop header data field are extracted and reported to a collector. As can be seen in 4.3, the approach avoids the possibility of identifying a non-existent path as the exact path the packet followed is recorded.

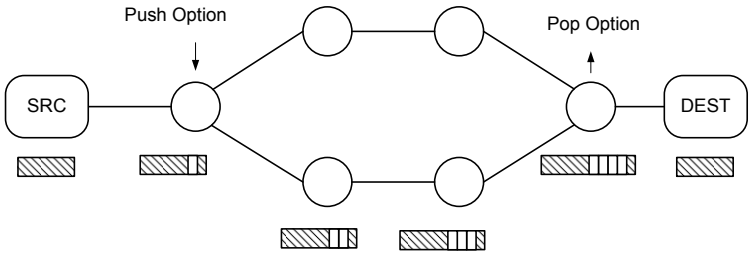


Figure 4.2: Intrinsically monitoring of traffic path under load sharing.

4.3 Performance Evaluation

We implement our intrinsic monitoring functionalities in the Linux kernel. Ping6 is changed to construct the monitoring header. The interested monitoring information are input as parameters to initialize the identifier fields. The Linux kernel

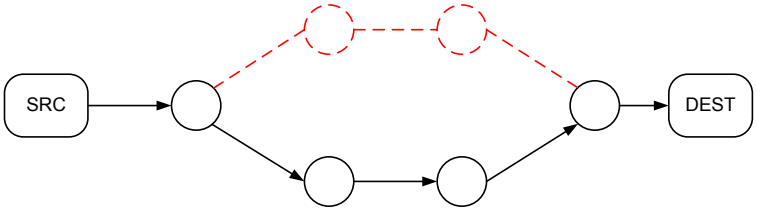


Figure 4.3: Correct traffic path monitored with no false links.

is also extended to support the processing of the intrinsic monitoring hop-by-hop extension headers. In the IPv6 forwarder of the kernel, we collect the monitoring information as indicated in the identifiers fields of the monitoring hop-by-hop header. The ICMPv6 module is changed to dump the collected information to the ICMPv6 data portion, and return it back to the collector once the packet reaches the receiver. In following experiments, we suppose the source itself is the collector. We compare the traceroute and intrinsic monitoring approach in terms of delay and traffic overhead to show that our approach is efficient and scalable.

Traceroute sends UDP probes to high-numbered destination ports. It uses the reception of an ICMP port unreachable message to indicate that the destination has been reached and probing is complete. The use of high-numbered port minimizes the chances of accidentally probing an open service on a machine, which would prevent an ICMP port unreachable message from signaling completion to traceroute. Derivatives of this method are found on most Unix systems including MACOS X, FreeBSD, and popular distributions of Linux. By default, this UDP-based method has a destination port value 33434. A weakness of this probe method is that some firewalls block these probes from reaching their intended destinations as a product of blocking unknown traffic by default, reducing the utility of traceroute.

4.3.1 Experimental Setup

We have one Windows XP SP3 machine equipped with Pentium Duo-Core 3.00GHz processor, 500G hard disk, and 4GB RAM. In the experiment, we used virtual machine technology to facilitate the tests. Each virtual machine image has a size of 10G bytes and is configured with two Ethernet cards. After re-compiling kernel, at least 176M bytes memory is required for each image in order to make it run. We have built up 18 virtual machine images. One of the machines runs Wireshark to monitor data packets on the network. Except one destination D and one sender S, all the other 15 machines are set up as intermediate nodes.

During the test, each intermediate node i has equipped with two interfaces,

their IPv6 addresses are configured as $2001 : i - 1 : i : 4004 : 219 : i/64$ and $2001 : i : i + 1 : 4004 : 219 :: i/64$, so that for each intermediate node i, one interface is in the same network segment with node $i - 1$ and the other one is in the same network segment with an interface from node $i + 1$. From the source to the destination, the data packet has to go through all the intermediate nodes, so it is 16 hops away from the source to the destination. Except the IP address configuration, the routing configuration is also important. For node i, we configure the default route's next hop as node $i + 1$. Another route is also added for the source S as the destination address, the next hop address is configured to node $i - 1$. For the source S and destination node D, their default gateways are set to the closest intermediate nodes. This configuration allows us to easily test from the source node to every hop without changing any routing and IPv6 addresses configurations.

4.3.2 Traffic Overload Comparison

To validate the proposed technique, we calculate the traffic overhead in the following manner. In a network, we suppose the destination node is N hops away from the source node. At the beginning, the source node issues 3 UDP packets with TTL setting to 1, and then gradually increases TTL, each time TTL is increased by 1. In the Linux Kernel 2.6.28, the UDP packets sending out by the source is 78 bytes, including the first 14 bytes Ethernet header, IPv6 header of 40 bytes, UDP header of 8 bytes, as well as the data portion of 16 bytes. Each hop will reply back with three ICMP messages, note that upon arrival of the UDP packets at the destination, the ICMP messages triggered are ICMP port unreachable messages, before that, the messages are all TTL timeout messages. There will be $3N$ ICMPv6 messages in total. Each ICMP packet size is 126 bytes, including the Ethernet header (14 bytes), IPv6 Header (40 bytes), ICMP Header (8 bytes) and the data portion (64 bytes). Therefore, the total of data traversing the network is $612N$ bytes.

However, in our monitoring approach, one monitoring packet is enough. In terms of the traffic overhead, even if only consider the final length of the largest message, that is, the monitoring packet is having all the intermediate router IP addresses, the length of the message is only $20N + 70$. Inside this message there is an Ethernet header (14 bytes), an IPv6 header (40 bytes), the generic hop-by-hop header (4 bytes), monitoring header (12 bytes), the monitored information (16 bytes) and the record header (4 bytes) per record.

We compared the traffic overhead generated by our method and traditional traceroute, we only generate a ratio of $(20N + 70)/612N$, which is less than 5% of the traffic overload incurred by traceroute if the hop number is larger than 6. We can reduce 95% of traffic overhead and still provide the accurate measurement for most of the Internet measurement. Even taking into account network situations

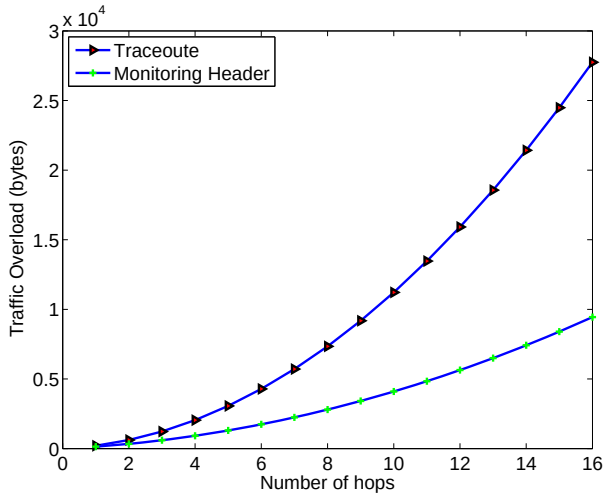


Figure 4.4: Traffic overload of intrinsic monitoring verses traceroute.

such as network congestion, we may send up to 3 packets, the traffic overload generated by our approach is still less than 15% of the current traceroute method for most of the measurements.

In Fig. 4.4, we compare the generated traffic by traceroute to our proposed approach. The monitoring machine captures all the generated packets during the test, because the monitoring machine is in the same virtual network with all the machines. We show the traffic in terms of bytes from 1 hop away until 16 hops away from the source, and demonstrate that the total traffic overhead is approximately 30% of that generated by traceroute.

4.3.3 Delay Comparison

In our approach, although we introduce a small processing delay for each hop, but we no longer use the approach trying one hop by one hop further. Thus, we have the advantage in terms of time for obtaining the final result, as well as traffic overload. The only disadvantage is that each node has to deal with this monitoring message. However, we make the Linux kernel processing overhead is also relatively small, because every time the insertion of information is after the fixed record header. During the memory operation, each time a fixed length of the memory is moved ahead, and only the data before and including monitoring header is moved. For traceroute, requirements for functionality is more important than the performance requirements, it is not particularly care about the little delay.

Assume that from source S to destination D, there are $N-1$ intermediate

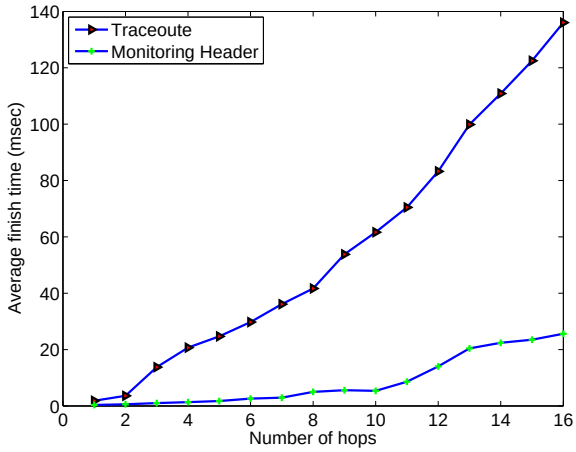


Figure 4.5: Round trip delay of intrinsic monitoring verses traceroute.

nodes. The destination is the nth node. We suppose from S to D, each pair of neighbor nodes has delay t_i, where $i = 1 \ldots N$. The traceroute method has delay $2t_1+2(t_1+t_2)+2(t_1+t_2+t_3)+2(t_1+t_2+t_3+\ldots+t_N) = 2[Nt_1+(N-1)t_2+(N-2)t_3+\ldots+t_N]$, in contrast, our approach takes $t_1+t_2+t_3+\ldots+t_N$. If we suppose the neighbour delay t_i is roughly same and denote it as $t_{avg(i)}$. Then the traceroute method will take $(1+2+\ldots+N)t_{avg(i)} = N(N+1)t_{avg(i)}/2$; our approach takes $Nt_{avg(i)}$. Our approach will only generate a ratio of $2/(N+1)$. Take a 16-node scenario, it only takes 12.5% time compared to the traceroute method.

In Fig. 4.5, we compare the total finish time by the traceroute approach and our intrinsic monitoring approach. The total finish time is the whole run time starting from the beginning of the monitoring process and until the monitoring process finishes. The result shows that the finish time of our approach increases much slower than the traceroute approach when the hop number increases.

4.4 Security Considerations for Intrinsic Monitoring

In this section we discuss a number of security considerations for the intrinsic monitoring approach. A secure method for collecting monitored network information utilizing the IPv6 hop-by-hop extension header was proposed. We propose an ICMPv6 extension which initiates the key distribution and security association negotiation. This approach uses the existing IPsec framework. We also propose a light-weight information encryption and authentication scheme to securely transport collected monitoring information.

Information security is critical for intrinsic monitoring. We focus our security considerations on the confidentiality and integrity of monitored data and authentication of origin. Confidentiality means stored or transmitted monitored data cannot be read or altered by an unauthorized party. ISPs consider monitored data proprietary and are unwilling to disclose their internal network structure and performance information details. Integrity means any alteration of transmitted or stored monitored data can be detected. The receiver should be able to detect altered monitored data to prevent misinterpreting the meaning of the original data or even malicious attacks. Authentication of origin ensures the monitored data from trusted sources.

There are few requirements for doing the authentication and encryption to achieve the information security purpose. First, the amount of encrypted data should be reduced to a minimum because encryption is computationally expensive. Each network node should only encrypt the data it inserts to reduce the encryption overhead. The second is to limit the authentication overhead to a minimum. The network node should only authenticate the data it inserts to reduce the authentication overhead. Both the encryption and authentication is computationally expensive, so it is important to use suitable encryption and authentication algorithms to reduce the requirements for computing resource.

4.4.1 Protecting the Hop-by-hop Extension Header

A *hop-by-hop monitoring extension header* is composed of a header and multiple performance metric records as shown in Table 4.5. This design aligns all the data into a 32-bit word boundary.

Now we present our method of protecting the *hop-by-hop monitoring extension header*. As Internet Key Exchange version 2 protocol (IKEv2) [84] is integrated as a core component of IPv6 protocol stacks, we assume that it is supported by every network node. A protocol, for example an extended ICMPv6 protocol, initiates IKEv2 protocol for all nodes along the path in order to create IPsec Security Associations (SAs) with the destination node. SA includes keys, key lifetime, encryption and authentication algorithm, and related parameters.

IKEv2 offers a reliable and efficient key exchange scheme to provide a secure data transport. It also defines procedures to establish, negotiate, modify and delete SAs. The advantage of IKEv2 is that it enables on-demand creation of keys for SAs and facilitates the use of keys in a large distributed system.

Our method is a combined encryption and authentication method. After the SAs are created, each node along the path shares a secret key with the destination node. Asymmetric algorithms use a significant amount of computational resources in comparison to their symmetric counterparts. Therefore, they are not chosen to encrypt monitored data.

Table 4.5: IPv6 Hop-by-hop Monitoring Extension Header Format

Next Hdr	**Hdr Ext Len**	**Option type**	**MTU Len**
Sequence Number			
Record Count	**Num of Metrics**	**Flags**	**Record Hdr Length**
Identifier 0	**Identifier 1**	**...**	**Padding**
IPv6 address(optional)			
SPI			
Node Position 0	**Num of supported Metrics**	**Authentication data length**	**Record Length**
Identifier 0	**Identifier 1**	**...**	**Padding**
Identifier0 Data			
Identifier1 Data			
...			
IdentifierN Data			
Padding and padding length			
Authentication Data			
...			
SPI			
Node Position N	**Num of supported Metrics**	**Authentication data length**	**Record Length**
Identifier 0	**Identifier 1**	**...**	**Padding**
Identifier0 Data			
Identifier1 Data			
...			
IdentifierN Data			
Padding and padding length			
Authentication Data			

The symmetric encryption algorithms use a single key to encrypt and decrypt the data. The advantage is that symmetric algorithms use significantly less computational resources than their asymmetric counterparts. This single key is exchanged securely before the secure communication using key management protocols in IPsec framework. Typical key sizes are 64, 128, or 192 bits. Fig. 4.6 shows how a shared secret key is used for protecting confidentiality of each record. The data portion starting from *node position* until *padding length* field (inclusive) is encrypted.

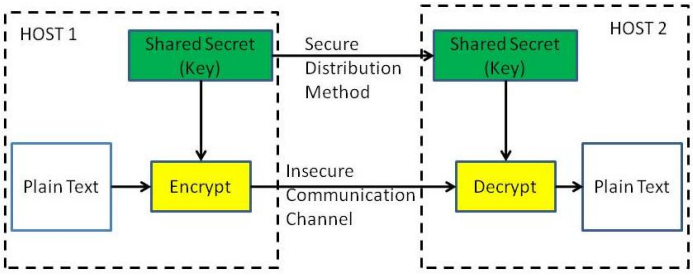


Figure 4.6: Symmetric cryptography.

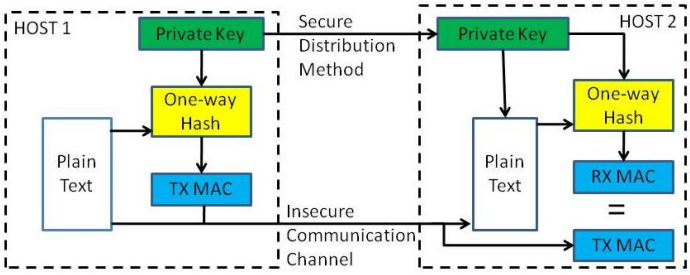


Figure 4.7: Message authentication code.

Message Authentication Code (MAC) algorithms are used to ensure authentication and data integrity. The MAC is the *authentication data* field of each record in the IPv6 *hop-by-hop monitoring extension header*, which is computed from the monitored data using the shared key. The authenticated data starts from *SPI* until *padding length* field (inclusive). After the receiver gets the message, it recomputes the *authentication data* field using the authentication algorithm and shared key, then compares the result with the received *authentication data* field. If they are equal, the authentication is successful. The process is shown in Fig. 4.7. The shared secure key authenticates the sender, and the hashed result ensures data integrity.

In a typical scenario, there are one sender, one receiver, and multiple intermediate nodes. The procedures can be summarized as follows: The sender initiates monitoring by creating a *hop-by-hop monitoring extension header*, which indicates the interested performance metrics. The sending pattern will be specified according to the monitoring purposes, e.g., from a specific application.

When an intermediate router receives the packet with *hop-by-hop monitoring extension header*, it recognisees it as a monitoring packet. After decapsulating the packet, it collects its performance metric values according to the requirements and generates a monitoring record. It then encrypts this record using the shared key. The authentication data will be generated for the encrypted record only. Finally, the encrypted data and the authentication data will be inserted into the *hop-by-hop monitoring extension header* as a record following the format defined in Table 4.5.

The destination node receives the monitoring packet and processes records one after another. Each record is associated with a *SPI* field, which identifies the SA to which this monitoring record belongs. The encryption and decryption is very computationally expensive, so the decryption should be done for the receiver after the successful authentication. The destination node authenticates the monitoring record using the key from corresponding SA, applies the hash algorithm to the encrypted monitoring record, and if the results match, then both the authenticity of the sender and the integrity of this monitoring record are assured. Then the decryption algorithm and key associated with the SA are used to decrypt the monitoring record. The monitoring record will be processed by the destination node one by one until it finishes or error occurs.

4.5 Summary

Nowadays network monitoring systems are crucial to communication networks. They periodically collect network performance metric values, identify performance anomalies, and determine root causes for the problems. Their effectiveness and efficiency determine the quality of network services. The most important performance metrics include connectivity, delay, packet loss rate, location of congested network nodes, and bandwidth information.

In this chapter, we have proposed a method for collecting monitored network information using a newly proposed IPv6 hop-by-hop extension header. We discussed how the approach can be used to directly map traffic flows to network paths. We demonstrated how our approach performs in comparison to an active based network monitoring tool called traceroute. The results demonstrate that our approach only generates less than 5% of the traffic generated by traceroute, at only around 12% of the time taken by traceroute, to retrieve the results for a 16-node network.

The extension header which can be easily extended to include a wide range of monitored information that can be collected within the node. Using such an approach can more effectively monitor network conditions as traffic is traversing the network. In the future we wish to extend the approach to collect additional information such as congestion related information, time stamping, dropped packets, etc. The objective of this research is to increase the monitoring capabilities within the network and to allow advanced monitoring of both traffic flows within the network and intermediate nodes along the path. We envisage that such monitoring can greatly improve the efficiently of network planning and traffic engineering strategies in the future.

We also have proposed a secure method for collecting monitored network information utilizing the IPv6 hop-by-hop extension header. We suppose that nodes are deployed with IPsec framework support, and a protocol, such as extended ICMPv6, initiates the key distribution and SA negotiation between the nodes along the flow transfer path and the destination node. Based on that, we have also discussed a light-weight information encryption and authentication scheme to securely transport collected monitoring information. To our knowledge, it is the first try to offer a secure IPv6 hop-by-hop extension header for all kinds of monitoring purpose, which is a significant step for intrinsic monitoring towards real deployment.

Chapter 5

Intrinsic Monitoring within an IPv6 Network: Mapping Node Specific information to Network Paths

Network monitoring systems play a crucial role in managing communication networks. They collect a wide range of network related metrics so that the network manager can carry out essential management functions such as Fault, Configuration, Accounting, Performance and Security (FCAPS). Their effectiveness and efficiency have a direct impact on the quality of network services.

Network managers typically collect metrics from administrated routers at regular intervals and in a centralized manner. At the same time, network managers are trying to assure the quality of delivery of premium services for their most valued customers, where a primary concern is ensuring that the network performance on paths to their premium customers is within a range of pre-defined criteria, according to service level agreement (SLA). To associate these metrics with particular paths requires offline processing, and with conventional monitoring tools it is difficult to get a timely view of a specific path's performance, as the collected data will likely be already outdated due to the dynamic nature of networks.

There are many readily available methods for collection of information for relating to paths within a network, such as using the ICMP based traceroute utility. However, this approach is limited to connectivity information only, and there is a high overhead associated with supervising paths using traceroute. When associating metrics to a path in the network, a query must be generated locally for each network device, discovering the IP address of the network device. Given the object identifier (OID) of SNMP MIB, specific metrics can be queried and associated to the path.

We define a path-based monitoring protocol for collecting node specific MIB metrics on a path within a network between any two routers. The protocol requires that the IPv6 Router Alert option is supported by all routers (nodes) in the net-

work core. The Router Alert based protocol can utilize a router's SNMP MIB, so that the collected monitoring information can be as general as the SNMP protocol. Performance metrics defined by OIDs can be used to aid in the assessment of the performance of an individual path. The protocol being considered is being designed to augment, not replace, current monitoring practices.

By employing the proposed monitoring scheme, a network manager can efficiently discover the path, and collect information from each of the nodes on the path. The approach can potentially be used across management domains as discussed in [85], but is restricted to a single administrative domain. A management domain in this context is considered as one centralized management system and all the IPv6 routers that it manages. An administrative domain is the complete set of management domains of a network provider.

Our approach has the potential to get a more accurate representation of the path and can retrieve metrics more efficiently through the invocation of a single message. We utilize SNMP OIDs for the collection of system information, and a network manager has the flexibility to specify what OIDs are to be collected, including both standard OIDs and vendor specific extensions, which also future proof this protocol mechanism.

Our contributions in this work are summarized as follows.

- First, we propose a number of packet structures, used for different purposes, including the triggering of intrinsic monitoring, and the processing, as well as reporting of packets.
- Second, we describe a set of procedures for the processing of each of the packet structures mentioned above.
- Third, we define a model that allows to determine the maximum number of monitoring packets to send, and the network overhead this would cause, using a Poisson stochastic process taking network failures into account.
- Fourth, we employ the existing SNMP OID naming structure to identify node metrics for collection within the protocol.
- Finally, we present an implementation of our scheme, and an evaluation in terms of time reduction on a virtual network based on VMware virtual machines. We also provide a theoretical analysis of both traffic and time overhead for both our proposed approach and the conventional SNMP collection method.

The remainder of this chapter is organized as follows. Section 5.1 outlines the system design context in which our protocol is evaluated. Section 5.2 outlines the

design of our packet structures and Section 5.3 describes the packet processing procedures. Section 5.4 presents our implementation. Section 5.5 discusses theoretically calculated monitoring parameters to ensure collection reliability. Section 5.6 outlines network model assumptions and shows the results of our evaluation on the performance of the intrinsic monitoring. Finally, we conclude the work with discussion and present future work in Section 5.7.

5.1 System Design

The intrinsic monitoring technique is viewed as a lightweight metric collection protocol. The performance metrics for all nodes along a specific path can be monitored between a source-destination pair by inserting specific information into selected IPv6 packets of a data flow. By initiating an extension header at a source, and updating the packet data at all intermediate nodes along the source-destination path, a destination node can report the collected monitoring information to the management node, so that it can conduct a performance evaluation of select nodes in a network based upon the reported data in the IPv6 packet.

From the system design viewpoint, the process is an extension of the `SNMP Get` command, where the extension command can be described as `SNMP Get Source Dest OIDs` compared to the original `SNMP Get Dest OIDs`. The original `SNMP Get` command can only obtain the specified destination's OID values, but by specifying the source node, as well as the destination node, we can obtain all OID's values along the path. Each node along the path retrieves the information from the system and appends the additional information into the monitoring packets.

Using this approach, any transport layer protocol could be utilized for monitoring purpose. Furthermore, a new ICMPv6 type could be defined, following the idea that a packet's payload should solely consist of the collected monitoring records. The OIDs in the extension header are not protected, but it would be important to offer some levels of the securities for the collected data inside the packet payload.

In order to make the protocol easy to implement from the router's perspective, monitoring packet must be obvious so that it can be processed in an efficient manner at the router's link card. To facilitate this, monitoring packets is carried directly following of an router alert. Encapsulation of the monitoring packet directly following router alert has several advantages. First, it makes the protocol more suitable for a kernel implementation. Second, the use of a separate protocol number provides scope to allow measurement packets to request to be queued as if they are another type of packet in order to understand how the network is likely to treat these packets compared with packets of other types. Third, the use of a sepa-

rate protocol number allows for more flexible filtering and may avoid measurement traffic being blocked by administrative policies designed to block DoS attacks.

Other system design issues and assumptions also need to be considered as listed below.

- Scalability - When the path is very long, the size of the monitored data for the path can cause a packet overflow. Solutions for large networks could make use of a segmentation approach, taking path MTU size into account. Another way of dealing of the collected monitoring information is to pre-allocate the length for the packet. One field "path pointer" could be used to identify where in the packet the next record should be inserted. For each record inserted, the "path pointer" is incremented by the amount of space that record uses. In stead of having a growing packet size, which causes MTU problems, pre-allocation of the monitoring record space will be a bit inefficient but reduce complexity, because the routers alone the path have to forward the padding part of the packets. In IPv6 network, the minimum transmission unit is 1280 bytes according to RFC 2460.

- Border Routers - Border routers will be set to prevent forwarding of the packet outside the network. The solution considered is limited to one administrative domain.

- Traffic Processing - Hop-by-hop options are slow to process, and core routers typically forward traffic as fast as possible in hardware. The monitoring packets will take the slow processing path within routing equipment, and the collection of the metrics will be carried out in software by the management processor of the router. A consequence of this is that the monitoring protocol is not expected to be used to collect accurate delay values for packet processing. However, there are other key metrics that will be useful when associated to a path.

- Protocol compatibility - It is possible to have multiple hop-by-hop headers. We use the standardized router alert header, which is compatible with current existing hop-by-hop extension headers, such as jumbo frame extension header.

5.2 Packet Design

We present the format for monitoring packets, trigger packets, and report packets in Table 5.1, 5.2, and 5.3 respectively. In accordance with the IETF RFC 2460 [81], the IPv6 Router Alert header must be processed by every node along a packet's

forwarding path. In our proposed approach, it is used to carry performance metric identifiers information that must be examined by each node. The monitoring packet is composed of a monitoring header and a set of collected performance records, which are appended to the end of a packet. The proposed format for a monitoring packet is shown in Table 5.2. A description of the fields for the packet formats is given in Table 5.4.

As the header size is limited, we are only storing the OID for identifying the performance metrics to monitor within the header. Any collected performance data will be stored in the packet payload, because payload size of the packet can be extended up to the path MTU. We are currently only supporting OID values of the MIB-2 structure, and every identifier value is stored using 4-bytes left aligned, implicitly prefixed with the common part `1.3.6.1.2.1`.

Considering the path MTU, a maximum of 24 individual OIDs is supported, and we use 24 bits within the data records to indicate if an OID is supported at a specific node. We also align every OID value to 4 byte boundaries. Record length is a multiple of 4, because we align all monitored data to 4-bytes boundaries. The record length is calculated by taking the size of the collected monitoring data from the system, plus the size of a fixed record header.

The "protocol queue" filed could be defined to identify how the packet should be prioritized at routers that maintain priority queues. The "protocol queue" could be set to, for example, IPPROTO_TCP or IPPROTO_UDP, which allows the Router Alert packets to be queued based on these protocols.

Table 5.1: IPv6 Hop-by-hop Monitoring Trigger Packet Format

Next Hdr	**Hdr Ext Len**	**Option type**	**Option Len**
Protocol Num		**Sequence Number**	
Destination IPv6 Address			
MonHdrLen	**OidNum**	**Flags**	**ProtoQueue**
Identifier 0			
...			
Identifier N			

5.3 Packet Processing

The monitoring protocol requires special processing handling the Router Alert packets as illustrated in Fig. 5.1.

The proposed operations are as follows.

1. In response to an alarm or a customer trouble ticket, the operator could send a trigger message to a source router indicating a path by specifying the destination node of the path, and the metrics to be collected. The metrics are

Table 5.2: IPv6 Hop-by-hop Monitoring Extension Header Format

Next Hdr	Hdr Ext Len	Option type	Option Len
Protocol Num		Sequence Number	
Management IPv6 Address			
MonHdrLen	OidNum	Flags	ProtoQueue
Identifier 0			
. . .			
Identifier N			
Transportation layer data			
Record Length	Supported Identifiers Flags		
Identifier0 Data			
Identifier1 Data			
. . .			
IdentifierN Data			
. . .			
Record Length	Supported Identifiers Flags		
Identifier0 Data			
Identifier1 Data			
. . .			
IdentifierN Data			

Table 5.3: IPv6 Hop-by-hop Monitoring Report Packet Format

Total Data Len		Sequence Number	
MonHdrLen	OidNum	Flags	ProtoQueue
Identifier 0			
. . .			
Identifier N			
Transportation layer data			
Record Length	Supported Identifiers Flags		
Identifier0 Data			
Identifier1 Data			
. . .			
IdentifierN Data			
. . .			
Record Length	Supported Identifiers Flags		
Identifier0 Data			
Identifier1 Data			
. . .			
IdentifierN Data			

defined as a set of one or more OIDs. The IP address of the management node, namely where the collected data should be reported, is also included in the trigger message.

2. At the source router of the measured path, a new monitoring message is created with the monitoring Router Alert set as the option type. Every intermediate router is responsible for collecting local monitoring data as specified

Table 5.4: Header Field Definitions

Option Type	**8-bit integer identifying this option as the monitoring information collection option. The IPv6 option must start with 001 indicating that routers should skip this option if they do not support it, and that the data in this option may change en route to the destination.**
Option Len	**Defines the length of the protocol number, and in our case it needs to always contain the value 2.**
Protocol Number	**A fixed 16-bit monitoring protocol number, we use arbitrary values for testing purposes (this needs to be assigned by IANA)**
Sequence number	**16-bit unsigned integer used to distinguish this packet from other monitoring packets.**
Monitoring header length	**8-bit unsigned integer used to indicate the length of the monitoring header in bytes.**
OidNum	**The maximum number of performance metric values that this packet will potentially contain.**
Flags	**See Flags definitions (Table 5.5 and 5.6).**
ProtoQueue	**The pseudo protocol number for queuing purpose.**
Record Length	**The record length in 4-byte steps.**
Supported Identifiers Flags	**Indicates how many performance monitoring metrics are supported on this hop, as in reality, this hop may not be able to support all the metrics indicated.**
Identifiers	**The last 4 bytes of the OID that is being monitored.**
Identifiers Data	**The monitored data (currently also restricted to 4 bytes).**

Table 5.5: Flag Format

I	R	Resv

by the OIDs given in the header. This will collect the actual OID values along the path. The monitored information is appended to the payload of the monitoring message.

Table 5.6: Flag Definitions

I	**Interface, 2 bits. If I = 1 then incoming interface IP address should be collected; if I = 2 then outgoing interface IP address should be collected; if I = 3 then both incoming interface and outgoing interface IP addresses should be collected.**
R	**Use together with routing headers to collect information from specific nodes where the addresses are indicated in routing headers.**

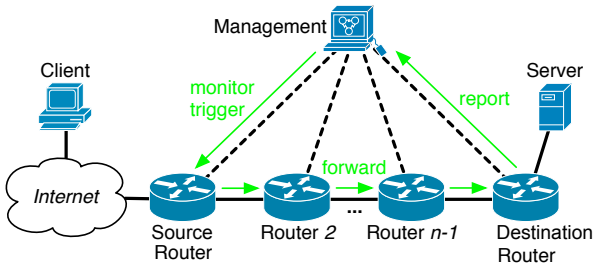


Figure 5.1: Packet processing procedures.

3. The collecting process is repeated at each hop until the destination node is reached. If the appended information for a message would cause the path MTU to be exceeded at an intermediate router, then that router would create a report message transmitting the collected metrics so far back to the management node, clearing the stored values, and adding its own data as specified by the OIDs before sending the packet to the next hop.

4. The destination node will intercept the packet with Monitoring Router Alert set. Recognizing that this node is the final node, the packet will be processed and all collected data will be forwarded to the management node.

5.4 Implementation

We developed a proof-of-concept implementation for the monitoring protocol. The system consists of the following modules:

1. Controller, which handles triggering of the monitoring activity, identifies the source and destination addresses to probe, issues measurement requests to

the corresponding source node, and manages the collection of monitoring information. The controller is located in the management node.

2. Translator, which translates the monitoring trigger to the intrinsic monitoring packet and initializes the monitoring process. When receiving a completed monitoring packet at the destination node, it translates the completed monitoring packet into a report message and sends to the management node. In Fig. 5.1, translators are located at the source and destination routers.

3. Collector, which exists at each node along the path. Its responsibility is to retrieve values for the monitored metrics, and append them to the monitoring packet before forwarding it. The collector is implemented by modifying the IPv6 module of Linux kernel.

4. Analyzer, which processes the collected information and infers the network performance based on the obtained measurement data. The analyzer is located in the management node.

To cope with packet loss in the monitored network, we might send more than one packet. This is determined by a parameter referred to as the "sending frequency". A requested monitoring activity is scheduled by the management node, and executed according to the current sending frequency. When a set of measurements has finished, the results are sent back to the management node. The analyzer runs continuously, and will process the incoming monitoring data to infer the performance of devices along the taken path.

Any data packet, independent of the employed transport layer protocol could be used as a monitoring packet. In our implementation, we use ICMPv6 `Echo Request` packets for obtaining monitoring information. On receiving a monitoring packet at an intermediary router, we obtain the outgoing interface and retrieve the specific OID values directly from data structures in the kernel space after the kernel routing process has finished. The retrieved information is saved in a buffer until all monitoring data has being collected, only then we append it to the packet. For every MIB OID, we implement a function that is able to obtain the value from kernel data structures. These functions are called by means of function pointers. The OIDs to be retrieved are sorted in an increasing order, and a binary search algorithm is used to identify the appropriate function to call.

When a monitoring packet arrives at an intermediate node, the "Supported Identifier Flags" field are initially all set to zero. If a particular OID is not supported in the system, the corresponding bit is set to 1. Using this mechanism, support for various OIDs can be introduced gradually into a network, starting with a small number of features at the beginning, and adding more features over time.

Appending monitoring data will cause an error in the overall packet's checksum. We presume that the transport-layer checksum is not evaluated by the router, and that the destination router could process the piggybacked monitoring data earlier than the destination host. After stripping the monitoring data, the checksum will be correct again, as well as the length of the packet.

5.5 Determination of Monitoring Parameters

The following section defines a mathematical model for calculating the sending frequency, determination of the number of packets used per measurement, and the overall number of measurements that need to be conducted within a day. To do this we model an unreliable network using a homogeneous Poisson process [86].

Network failures can be modeled as a sequence of arrivals of discrete entities, leading to the usage of two equivalent representations: counting processes and interarrival time processes [86]. A counting process $N(t)_{t=0..\infty}$ is a continuous, integer-valued stochastic process, where $N(t)$ expresses the number of arrivals in the time interval $(0, t]$. An interarrival time process is a non-negative random sequence A_n, where $A_n = T_n - T_{n-1}$ indicates the length of the interval separating arrivals $n-1$ and n. This failure model is largely defined by the nature of the stochastic processes $N(t)$ and A_n chosen. The two kind of processes are related through the following equation:

$$\{N(t) = n\} = \{T_n \leq t < T_{n+1}\} = \{\sum_{k=1}^{n} A_k \leq t < \sum_{k=1}^{n+1} A_k\} \tag{5.1}$$

The number of failures $N(t)$, which occur in a computer network over the time interval $[0, t)$, can be described by a homogeneous Poisson process $\{N(t), t \geq 0\}$. On average, there is a failure after every t hours, i.e., the intensity of the process is equal to $\lambda = t^{-1}$.

By the homogenous Poisson distribution, we should have same distribution for any periods of time [86, 87]. The number of events in any interval of length t is Poisson distributed with mean λt. That is, for all $s, t \geq 0$

$$P\{N(t+s) - N(s) = n\} = e^{-\lambda t}\frac{(\lambda t)^n}{n!} \tag{5.2}$$

The probability of n network failures occurs during any t hours is calculated using the Formula 5.2 given above. We presume that there is a failure after every half an hour [88], and during any 24 hours, there are most likely 48 network failures occurring.

The link state variable Z_e is a Bernoulli random variable that takes value one with probability α_e if the packet is transmitted over link e and value zero with probability $1 - \alpha_e$ if the packet is lost on the link, α_e is called the success rate or packet delivery rate of link e, and $1 - \alpha_e$ is called the loss rate of link e. The outcome variable L_k is also a Bernoulli random variable, which takes value one if the packet successfully reaches node k.

$$L_k = \prod_{e \in \mathcal{P}(s,k)} Z_e \tag{5.3}$$

where $\mathcal{P}(s, k)$ denotes the sequence of links that connect node s to node k on the forwarding path. In the consideration of how many monitoring packets we need to send out to make sure that at least one monitoring packet will reach the destination, we denote p as the probability of this happening at least once for n packets, and q as the probability that no packet reaches the destination.

$$p = 1 - q = 1 - (1 - \alpha_e)^n \rightsquigarrow 1 \tag{5.4}$$

When $(1 - \alpha)^n$ approaches 0, we get $x = 3.42$ if we want the a value of 99.999% for p and a 1% link loss rate. If the packet loss rate is 0.01%, then the model dictates that it takes 1.73 packets to successfully reach the destination. We could set the maximum sending packet number to two to make sure that at least one packet reaches the destination.

5.6 Performance Evaluation

To evaluate the protocol, we carried out both a theoretical and an experimental evaluation. For the theoretical evaluation, we based our network parameter assumptions on a real network topology as stated in Section 5.6.1. We define the setup of our experimental testbed on Section 5.6.2. The performance evaluation focuses on two main aspects: Time overhead discussed in Section 5.6.3 and traffic overhead discussed in Section 5.6.4.

5.6.1 Network Parameters

We take the AS1755 [16] as our experimental ISP topology, where the diameter of this network is 11 hops. The delay for any pair of two nodes is following a T distribution with μ 25.5351 ms and σ 12.0704. In our case, we neglect the extra delays introduced by network congestion, etc. Hop numbers of any two nodes follow a normal distribution with μ 4.94948 and σ 2.09043. That means that we take 5 hops as the average distance between any two nodes. The average delay

between any two nodes is 25.5351 ms and the average distance is 5 hops, that means the average delay between two adjacent nodes is around 5ms.

5.6.2 Experimental Setup

We have one Windows XP SP3 machine equipped with Pentium Duo-Core 3.00GHz processor, and 4GB RAM. In the experiment, we used Virtual Machine (VM) technology to facilitate the tests using 15 individual VMs. Each VM image is configured with two Ethernet cards. One of the machines runs Wireshark to monitor data packets on the network. Except one destination D and one sender S, all the other 12 machines are set up as intermediate nodes.

During the test, each intermediate node i has equipped with two interfaces, their IPv6 addresses are configured as $2001 : i-1 : i : 4004 : 219 : i/64$ and $2001 : i : i+1 : 4004 : 219 :: i/64$, so that for each intermediate node i, one interface is in the same network segment with node $i-1$ and the other one is in the same network segment with an interface from node $i+1$. From the source to the destination, the data packet has to go through all the intermediate nodes, so it is 13 hops away from the source to the destination. Additionally to the IP address configuration, the routing configuration is also important. For node i, we configure the default route's next hop as node $i+1$. Another route is added, taking the source S as the destination address, and the next hop address configured as node $i-1$. For the source S and destination node D, their default gateways are set to the closest intermediate nodes. This configuration allows us to easily conduct packet forwarding from the source node to every hop without changing any routing and IPv6 addresses configurations.

5.6.3 Time Comparison

When using conventional SNMP to obtain performance metrics from a specific path, we need to use traceroute for path discovery first, followed by a number of `SNMP Get` requests sent out to each node along the path to obtain the values. Traceroute needs to be initiated by the first node in the path (in our case the source router). During this process, we consider the time for conducting traceroute and the average `SNMP Get` time, which includes inquiry forwarding, data retrieval and data feedback. `SNMP Get` processes can be done in parallel, and the data analysis could start once the first SNMP packet has arrived at the management node.

We denote T_t as the time for traceroute, and T_{s_i} as the time for obtaining OID value(s) from a specific node i. We define $T_s = max(T_{s_i})$. Traceroute will only report the route information back to the initiating node, so the source router has to collect the discovered path information and transfer it back to the management

node. Most likely management node and initiating node are not the same, so we have to also consider $T_{trigger}$ for the time it takes to initiate a traceroute command at the source router, and T_{report} as the time that it takes to send the results back to the management node. Then the overall duration for collecting all monitoring data at the management node is:

$$T_{conventional} = T_{trigger} + T_t + T_{report} + T_s \tag{5.5}$$

For the intrinsic monitoring approach, the time to retrieve the same information would mostly depend on the time T_{trip} that the intrinsic monitoring packet takes to travel from the source to the destination node. This is given by the formula:

$$T_{intrinsic} = T_{trigger} + T_{trip} + T_{report} \tag{5.6}$$

For simplicity, we neglect the marginal differences due to the varying processing mechanisms at the devices, and assume that the $T_{trigger}$ values, as well as the T_{report} values for both approaches are equal. The time difference $T_{difference}$ between both approaches can then be calculated using:

$$T_{difference} = T_t + T_s - T_{trip} \tag{5.7}$$

According to properties of AS1755, we assume an average hop number of any pair inside the network is $n_h = 5$ hops. Therefore, we use n_h for the distance between the management node and any node on the path. For referring to the number of hops from source node to destination node, we use h. The link delay δ is assumed to be 5 ms for all links. The calculation for the overall time of SNMP approach is then given by:

$$\begin{aligned} T_{conventional} &= 2\delta + \cdots + 2(h-1)\delta + n_h \cdot 2\delta \\ &= h(h-1)\delta + 2n_h\delta \end{aligned} \tag{5.8}$$

In contrast to the conventional measurement approach, the time needed with intrinsic monitoring can be calculated as:

$$T_{intrinsic} = (h-1)\delta \tag{5.9}$$

Using the formulas 5.8 and 5.9, the time consumption for both approaches in the AS1755 topology is shown in Fig. 5.2. From the formulation and as depicted in the figure, our approach demonstrates a linear growth of time taken to complete measurements as the number of hops from source node to destination node increases, where the conventional measurement approach demonstrates polynomial growth of time.

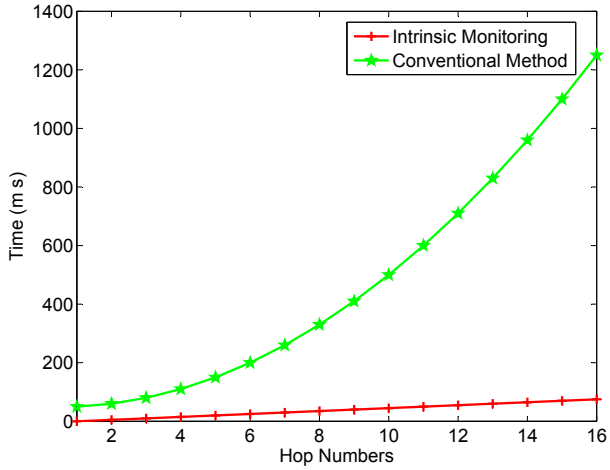


Figure 5.2: Measurement time comparisons.

Using our VM experimental setup, we measured the time taken to collect a specified number of OID metrics per node from a varying number hop counts between the source and destination nodes. We varied the number of hops from 2 hops to 13 hops. We also collected 3, 7 and 11 OIDs per hop for each experiment. The results are depicted in Fig. 5.3.

5.6.4 Data Traffic Overhead

Regarding traffic overhead, we should not only consider the generated monitoring traffic, but also consider how long these generated packets will stay in the network. We describe the generated data traffic volume v in bytes, and the time that a packet stays in the network as t. We refer to the caused data traffic overhead as vt.

We compare the traditional SNMP approach, and our intrinsic monitoring approach in terms of the traffic overhead. Suppose that we want to obtain n OIDs values. For the SNMP approach, we first calculate the overhead generated by traceroute. Note, that only the incoming interface can be obtained by using traceroute. Second, we consider the traffic overhead for SNMP Get packets that need to be routed to each every node along the path.

For traceroute, we assume that the Ping6 Echo packet length is l_e, and the Echo reply packet is l_r. The Echo packets have to be sent out in a bunch of 3 packets each time, and they have to return back to the source node. For the SNMP traffic, the RTT for any pair of nodes inside the network is 51ms, as suggested by the network properties of AS1755. We refer to the size of an SNMP Get packet as $P_{snmpGet}$,

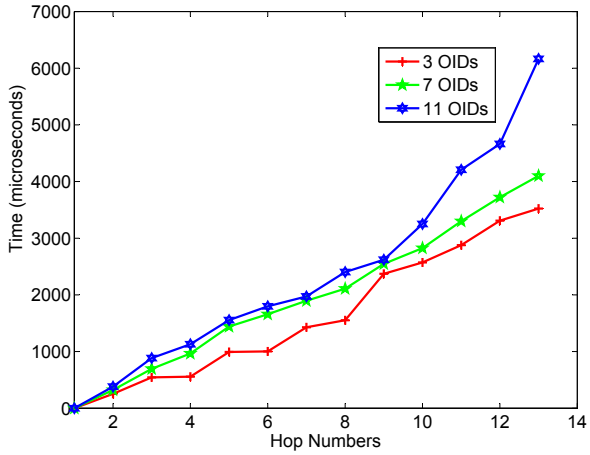


Figure 5.3: Time overhead versus hop count using intrinsic monitoring approach.

and to the corresponding response packet size as $P_{snmpResp}$.

The traffic overload introduced by SNMP is $n_h \cdot \delta \cdot (P_{snmpGet} + P_{snmpResp})$. The whole network overhead is:

$$\begin{aligned} O_{conventional} \quad = \quad & 3 \cdot l_e \cdot \delta + 3 \cdot l_r \cdot \delta + \cdots \\ & +3 \cdot l_e \cdot (h-1) \cdot \delta \\ & +3 \cdot l_r \cdot (h-1) \cdot \delta \\ & +n_h \cdot \delta \cdot (P_{snmpGet} + P_{snmpResp}) \end{aligned} \tag{5.10}$$

In our measurements, we consider each OID length as being 4 bytes, the average OID data as 4 bytes, n_h as 5 hops, and δ as 5 ms. Within the Linux implementation used for our experiments, l_e equals 78 bytes and l_r equals 126 bytes. We obtain $O_{conventional} = 5 \cdot (306h^2 - 306h + 160n + 680)$.

Using our intrinsic approach, n is greater than one and less than 24. We refer to the length of the intrinsic monitoring packet as l. The hop-by-hop header length is $3 \cdot 4 + 4 \cdot n + 16$ bytes, and each time it will increase with fixed 4 bytes record header, plus the data portion, starting from the source node. The first packet sent out from the source node will have a size of $l + 3 \cdot 4 + 4 \cdot n + 16 + 4 + 4 \cdot n$ bytes. The next forwarding will increase the packet size to $l + 3 \cdot 4 + 4 \cdot n + 16 + 2 \cdot (4 + 4 \cdot n)$ bytes. This process continues, until the packet reaches the destination with a final size of $l + 3 \cdot 4 + 4 \cdot n + 16 + (h-1) \cdot (4 + 4 * n)$. So the average network overload can be calculated as:

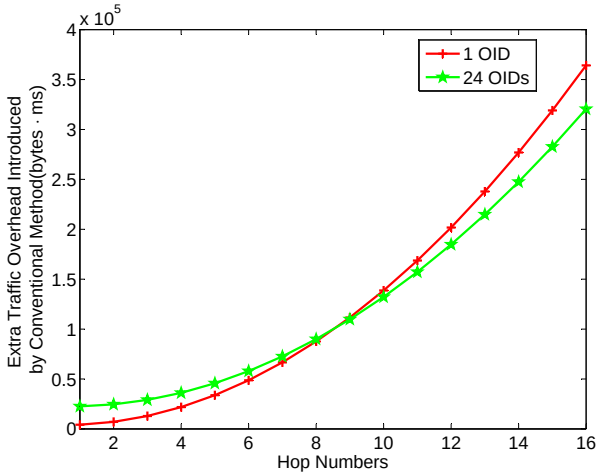


Figure 5.4: Extra network traffic overload comparisons.

$$\begin{aligned} O_{intrinsic} &= \delta * (3 \cdot 4 + 4 \cdot n + 16 + 4 + 4 \cdot n) + \cdots \\ &\quad +(3 \cdot 4 + 4 \cdot n + 16 + (h-1) \cdot (4 + 4 \cdot n)) \\ &= (h-1)(28 + 4n + 2h + 2nh) \end{aligned} \tag{5.11}$$

In Fig. 5.4 we depict the extra traffic overhead introduced by conventional methods in comparison to our approach. For a range of OID values from 1 to 24, we estimated the vt product for an increasing number of hop counts. As can be seen, our approach improves in efficiency as the number of hops increases.

5.7 Summary

In this chapter, we described a path-based monitoring protocol that can efficiently associate SNMP based metrics to a network path within a single administrative domain. The method is based on intrinsic monitoring, a lightweight metric collection protocol, by making use of the Router Alert mechanism [89].

The proposed protocol is not to replace SNMP, but rather to complement it by defining a mechanism that can efficiently discover nodes along a path and collect a set of information from each node along the path. It has not been designed to be a back door access to SNMP MIB data that bypasses SNMP access control policies. The proof-of-concept protocol has been implemented to understand the technical feasibility of the approach.

The main objective of the protocol is to investigate a method of dynamically discovering node specific OID values from static OIDs without having knowledge of local indexes. For example, the protocol can map the monitoring packet to the incoming and outgoing interface of the device it is passing through, and use this information to discover the local InterfaceIndex values appropriate to those incoming and outgoing interfaces. These index values can then be used by a management station to complete the request and identify the locally scoped information required. In contrast, SNMP requires explicit knowledge of these local InterfaceIndex values to retrieve this information before issuing the request, which is more difficult for a management station to discover for unstable network paths.

As a brief example of where this protocol can assist higher level management decisions is for IP-SLA monitoring. CISCO IP SLA can identify paths where there is a problem with latency, however, without knowing where the latency is likely to be occurring. With our approach, the protocol can quickly discover and collect relevant metrics from the nodes within the path for support of any performance or fault management decisions. The alternative is to gather potentially large volumes of data from each node either by SNMP requests and/or exporting netflow/ipfix records, followed by performing some data mining or analytic to pinpoint the problematic node.

There are several advantages. First is scalability, as it is not necessary to poll each router on the path individually. Second is the low overhead, as our approach is piggybacking the collected information with the real data, instead of generating specific SNMP Get packets. The third one is the increase in measurement efficiency and accuracy. The packet is being completely processed inside kernel space, instead of being processed by user space code, so the speed is optimal. The main advantage of our approach is that operators can rapidly associate key metrics to a troubled path in the network. Subsequently, a fast assessment can be carried out. This protocol is not meant to replace current monitoring practices, but it meant to augment them. It provides the network operator with a tool that can be used to focus monitoring on individual paths in the network, where the path is not known before hand.

This method is especially suitable for incremental deployment due to the usage of Router Alert headers, the intermediate routers would bypass the monitoring packet when it does not support the intrinsic monitoring functionalities. We believe such a monitoring protocol is desirable, given a potential advantage to network operators regarding a reduction in performance overhead of network monitoring processes. Also it is efficient, given that the paths are not known beforehand, but must be discovered by a conventional approach, e.g., using traceroute. A comprehensive experimental study exploring how such systems perform under a wide range of conditions would have considerable practical value.

Chapter 6

Loop-Free Forwarding Table Updates with Minimal Link Overflow

The intra-domain routing in IP networks today is mostly based on link-state routing protocols. In the management plane, network operators configure weights of the links, which indirectly control the selection of routing paths. In the control plane, each router computes the shortest-path to other routers with respect to the link weights.

Instead of using the current approach with centralized management and distributed control, one clean slate design suggests that the control of an autonomous system (AS) could be performed in a centralized way with direct control [29, 30]. A centralized server computes the routers' forwarding tables and controls the routers' forwarding decisions directly. A centralized control scheme simplifies the decision process, and reduces the functions required in the routers. The centralized control scheme also faces challenges and there are potential concerns, as also been identified and addressed in [29, 30]. For example, the centralized server constitutes a single point of failure and may have scalability problems. However, a logically centralized server does not exclude a distributed design.

The forwarding paths in a network may change due to link failures, equipment maintenances and reconfigurations of link weights. Studies from Iannaccone et al. have shown that link failures occur as part of everyday operation, and about 50% of the failures last longer than one minute [90]. When the forwarding paths change, the forwarding tables needs to be updated. During the update process, transient loops might occur since some routers may have updated their forwarding information bases (FIBs) while others have not. In a forwarding loop, packets are delayed and may be lost. Applications for interactive multimedia services such as VoIP are sensitive to packet delay and loss. Therefore, transient loops should be avoided whenever possible, particularly for non-urgent forwarding path changes. The events that can be considered as non-urgent include link and router maintenance, reconfigurations of link weights and restoration after failure of protected

links.

There have been several studies to avoid transient forwarding loops, using different approaches [4, 65, 66, 68, 69]. Using these approaches, transient loops will never occur and routers can reach one another during the entire transient phase. However, all these studies do not consider possible congestion and link overflow during the transient phase. Although the routers can reach one another, a large amount of traffic may flow over a few set of links during the transient phase, thus causing congestion in the network. The convergence time could take seconds to tens of seconds [4], and realtime traffic may hence still be severely impacted when link overflow occurs.

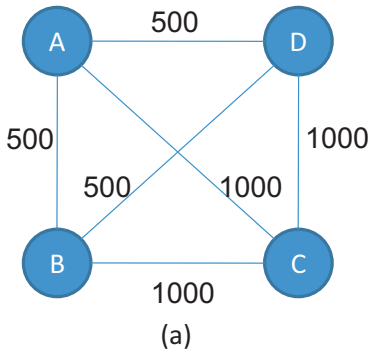
In this work, we propose an update order that avoids transient routing loops and takes the congestion and link overflow into consideration. We study the initial and final forwarding paths to obtain the *overflow-free updatable nodes*, where updating these nodes does not cause any transient loops or transient link overflow. However, overflow-free updatable nodes may not exist. Therefore, we propose an algorithm to update the forwarding tables that will cause minimum link overflow. Finally, we study the algorithm's performance on a real topology with two types of forwarding path changes. In particular, we compare our algorithm with an update order that does not consider link overflow [4]. The performance study includes both the number of update iterations and the total amount of link overflow during the transient phase.

The rest of the chapter is organized as follows. The problem is illustrated using an example in Section 6.1. Section 6.2 shows our algorithm for loop-free FIB updates that considers link overflow. The experimental setups including network topology and traffic demands are presented in Section 6.3. Section 6.4 shows the performance studies of the loop-free update algorithms. Finally, Section 6.5 concludes this chapter.

6.1 Problem Statement

In this section, we use an example to illustrate how transient loops may occur and how a loop-free update order presented in [4] may cause transient link overflow with given link capacities and traffic demands. A network topology is shown in Fig. 6.1 (a) with link capacities denoted beside each link. Fig. 6.2 (a) shows the initial forwarding graph to destination node D, which describes how traffic flows to the destination from other nodes before the forwarding table updates. Fig. 6.2 (b) shows the final forwarding graph, which presents how the traffic will flow after the updates. As can be seen in the figure, all three nodes, A, B and C need to update their next-hops to reach D. Note the numbers denoted beside the links are used for

illustrating a link overflow problem described later.



(a)

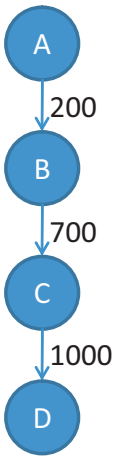
A network with link capacities shown

	D
A	200
B	500
C	300

(b)

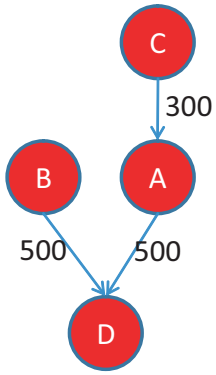
Traffic demands to node D

Figure 6.1: A network with link capacities and traffic demands.



(a)

Initial forwarding graph



(b)

Final forwarding graph

Figure 6.2: Initial and final forwarding graphs to destination node D.

If the FIB updates are not performed in an appropriate order, as when router C is updated first, then it will cause a transient loop $C \rightarrow A \rightarrow B \rightarrow C$. According

to the loop-free update order presented in [4], there are two kinds of nodes that are updatable in one iteration: the leaf nodes or those nodes whose upstream nodes have been updated in the initial forwarding graph; the nodes reach the destination directly or those nodes whose downstream nodes have already been updated in the final forwarding graph. Therefore, nodes A and B need to be updated in the first iteration (which means either of them can be updated first), and node C can be updated in the second iteration (which means C can be updated when both A and B have completed their updates). In this case, transient loops will never occur and the three potential transient forwarding graphs are shown in Fig. 6.3, where Fig. 6.3 (a) shows the potential transient graph when node A gets updated first, Fig. 6.3 (b) shows the graph when node B gets updated first, and Fig. 6.3 (c) shows the graph when both A and B have been updated.

Now we will consider the possible link overflow during the transient phase. For the network topology shown in Fig. 6.1 (a), the traffic demands towards node D are shown in Fig. 6.1 (b). We assume that there are no traffic flowing towards other destination nodes for simplicity. With the link capacities, traffic demands and forwarding graphs, we can compute the amount of flow on each link. The computed traffic flows are denoted beside each link in Figs. 6.2 and 6.3. As can be seen in Fig. 6.2, both the initial and the final forwarding graphs are overflow free, as the traffic flowing over each link are smaller than link capacities shown in Fig. 6.1. However, the potential transient graph after updating node B shown in Fig. 6.3 (b) causes transient link overflow in link $B \rightarrow D$. The link capacity is 500 while the amount of traffic that flows over the link is 700. Therefore, in this case, updating node B first causes a transient link overflow. While if node A is updated first, then B, then C, there will be no transient link overflow.

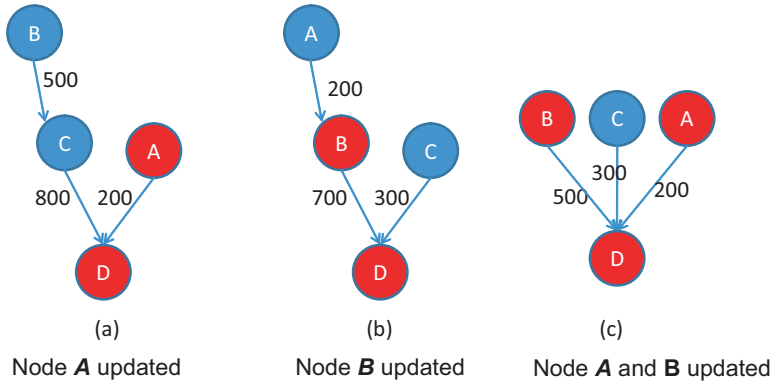


Figure 6.3: Potential transient forward graphs to destination node D.

6.2 Loop-Free FIB Updates with Minimum Link Overflow

In this section we describe how to perform loop-free FIB updates with minimum link overflow. In Section 6.1, we use a forwarding graph for a single destination as an example. However, all destinations need to be considered when studying the order of FIB updates. We use a *forwarding matrix* to specify all forwarding graphs in the network. In a forwarding matrix shown in Fig. 6.4, each row of the matrix is a FIB of a given router, and each column of the matrix shows the forwarding graph to reach a given destination.

To From	A	B	C	D
A	-	B	B	B,C
B	C	-	C	D
C	A	A	-	D
D	A	B	C	-

Figure 6.4: A forwarding matrix.

The FIB update process can be considered as performing a transition from an initial forwarding matrix M_i to a final forwarding matrix M_f. We do not presume any relationship between the two matrices except that both matrices are *consistent*, which means that there are no cycles in the forwarding paths. In addition, we assume that both the initial and final forwarding matrices are overflow free, which means that there is no link overflow in any links.

During the transition from M_i to M_f, the FIB updates are loop free if all transient forwarding matrices are consistent. Note that a FIB update can be a partial update, where only a subset of the modified forwarding entries are updated. The update order in our approach is specified as a sequence of *iterations*. At each iteration, a set of FIBs can be updated simultaneously, or in any order. When all routers

in an iteration have updated their FIBs, a new update iteration starts. Therefore, the update process is performed over a number of iterations, following a specified order. In each iteration, we need to select *updatable nodes*, which are the nodes that can be updated without causing transient loops. In particular, updatable nodes that causes minimal or no link overflow are preferred.

Our approach to the problem of finding an update order in a forwarding matrix is divided into studying individual forwarding graphs first. Thereafter, by combining update orders in all forwarding graphs, an update order for the forwarding matrix is derived.

6.2.1 Loop-free and Overflow-free Update in a Single Forwarding Graph

In this subsection we state and prove a theorem for selecting *overflow-free updatable nodes*, which are the nodes that can be updated without causing transient loops and link overflow.

We first define *downstream* and *upstream* relation, which will be used later in the theorem. In a forwarding graph, a node N_a is an *upstream node* of node N_b if N_a uses N_b to reach the destination. If N_a is an upstream node of N_b, N_b is a *downstream node* of N_a. Note that the destination node is not a downstream node of any node according to our definition.

Theorem 1 (Overflow-free updatable nodes): In a transition from an initial forwarding graph T_i to a final forwarding graph T_f, where both initial and final forwarding graphs are consistent and overflow free, a node update does not cause a transient loop in any transient forwarding graph T_t, and does not cause transient link overflow, if it fulfils the two following conditions:

- In graph T_t, the node is a leaf node or all its upstream nodes have been updated.
- In graph T_f, the nodes reaches the destination directly, or in T_f, all its downstream nodes together with all their upstream nodes have already been updated in graph T_t.

Proof. Consider a non-updated node N_a that fulfils both conditions. We assume that updating N_a results in a transient overflow in one of the links $N_a \rightarrow N_0$, $N_0 \rightarrow N_1$, ... , until the destination node. Because all upstream nodes have already been updated, so all the upstream nodes will be kept as same until the final graph. All the downstream nodes and their upstream nodes are also updated so they will be also be kept in the final graph. The overflow will remain in the final graph.

But as we assume there is no overflow in the final forwarding graph, there is a contradiction.

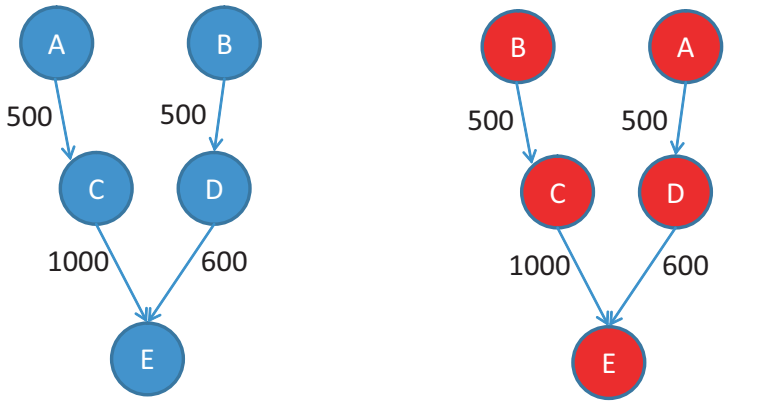
The conditions in our theorem are stronger than the condition for loop-free updates presented in [4], where the loop-free property is already proven. □

As an example, we study the problem stated in Section 6.1 using our theorem. According to the theorem, in the first iteration, only node A fulfils the overflow-free updatable condition, as it is a leaf node in the initial forwarding graph, and it reaches the destination directly in the final forwarding graph. When node A is updated, in the transient forwarding graph shown in Fig. 6.3 (a), node B fulfils the condition and can be updated as it is now a leaf node in the transient graph and it reaches the destination directly in the final graph. Finally in the last iteration node C fulfils the condition and can be updated. Using this update order, link overflow does not occur.

6.2.2 Loop-free Updates with Minimal Link Overflow in a Single Forwarding Graph

To successfully update from an initial forwarding graph to a final forwarding graph without overflow, it requires the existence of at least one *overflow-free updatable node* in each iteration. However, overflow-free updatable nodes may not exist, with an example shown in Fig. 6.5. In the figure, the initial and final forwarding graphs are shown, and it can be observed that only routers A and B need to be updated, as routers C and D's next-hops remain the same after the update. However, both nodes do not satisfy the conditions for overflow-free updates according to the theorem. In particular, if we study the link capacities and traffic demands in Fig. 6.6, it could be observed that link-overflow will always occur. As shown in the potential transient forwarding graphs in Fig. 6.7, if node A is to be updated first, the link overflow will occur at link $D \rightarrow E$, where the link capacity is 1000 and the traffic flow is 1100. If node B is to be updated first, the link overflow will occur at link $C \rightarrow E$, where the link capacity is 1000 and the traffic flow is 1500. In this case, it is better to update node A first since it will cause a smaller link overflow.

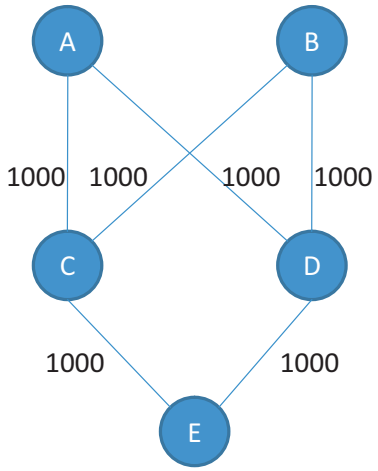
The idea behind our algorithm is to choose a loop-free updatable node that will cause minimal link overflow. The definition for *updatable nodes* that provide loop-free transition is given in [4]. Basically, updating all leaf nodes are loop-free and updating all nodes that reach the destination directly or through updated nodes are also loop-free. Therefore, there is always an updatable node in a transient forwarding graph, the update process can always proceed and eventually it will reach the final forwarding graph.



(a) Initial forwarding graph

(b) Final forwarding graph

Figure 6.5: Initial and final forwarding graphs to node E.



	E
A	500
B	500
C	500
D	100

(a) A network with link capacities shown

(b) Traffic demands to node E

Figure 6.6: A network with link capacities and traffic demands.

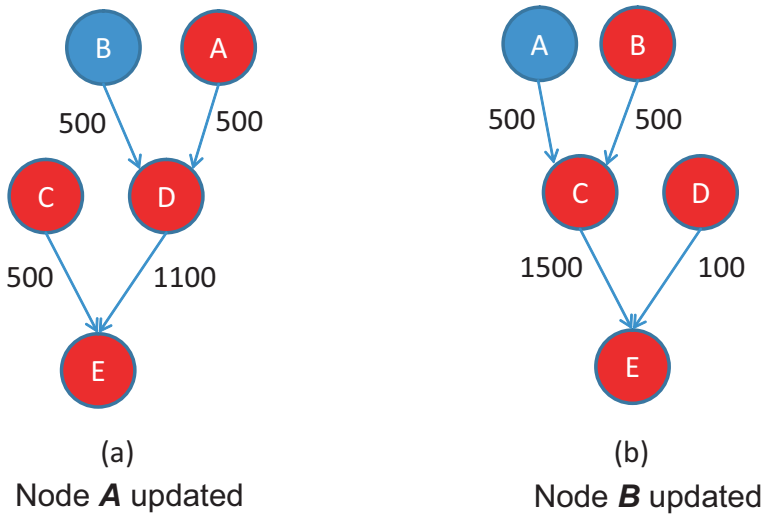


Figure 6.7: Potential transient forwarding graphs to node E.

Among the updatable nodes, choosing a node that will cause minimal link overflow is important. The general idea to choose the node to update is as follows. To analyze a node N, we first compute the sum of all its traffic flows F_N to the destination, this not only include its own traffic demand, but also include the demands from all its upstream nodes. Next we compute the minimal available link capacities L_N on its downstream links in the final forwarding graph. Thereafter, we compute the potential link overflow O_N for updating node N by subtracting L_N from F_N. Finally, among all updatable nodes, we should choose the node with the minimum O_N to update. If this O_N is non-positive, it means link overflow will not occur, otherwise link overflow will occur with the amount specified by O_N.

We present an example based on the forwarding graphs in Fig. 6.6. To choose among nodes A and B to update, we compute F_A and F_B first, both are 500 as they do not have upstream nodes and both of their traffic demands to E are 500. The minimal link capacity L_A is 400 as the link $D \rightarrow E$ has this available capacity, L_B, on the hand is 0 as the link $C \rightarrow E$ is fully utilized. Therefore, O_A is 100 and O_B is 500, and node A should be updated first.

6.2.3 Updating the Forwarding Matrix

After studying forwarding graphs for all destination nodes, we could derive the entries in the forwarding matrix that can be updated. Thereafter, the centralized

server can distribute the updatable entries to the routers. When the acknowledgments from these updates arrive, a new update iteration starts.

6.3 Experimental Setup

In this section, we present the experimental setup for our simulation study. We implemented a Java simulation tool according to our algorithm to perform the study. The simulation tool takes a network topology, a traffic demand matrix, and the initial and final forwarding matrices as inputs, then it compares our algorithm with the update algorithm presented in [4]. In the following subsections, we present how the inputs of our algorithm are generated.

6.3.1 Network Topology

In the experiment we use a real topology, AS1755, measured in Rocketfuel [91]. This is the Ebone network in Europe and contains 87 routers and 161 links. In addition, we use inferred link weights measured in [92].

6.3.2 Traffic Demands

The traffic demands we use are based on the Newton's gravity model of migration. The model is initially for migration of people between different towns. As the size of the towns increases, there will be an increase in movement between them. The further apart the two towns are, the movement between them will be less. The gravity model and its modified versions have been widely used in communication networks. In our demand matrix generation, each node is given two uniformly distributed random variables o and i for outgoing and incoming traffic. The traffic demand from node s to t is specified as follows:

$$D(s,t) = \alpha \frac{o_s i_t}{d(s,t)^2},$$

where $d(s,t)$ denotes the distance and a constant α is used to scale the demands to appropriate values. We set α to an appropriate value so that the average link utilization is about 60%. In the short time period of FIB updates, we assume the traffic demands are not changing.

6.3.3 Initial and Final Forwarding Matrices

We use the Dijkstra's shortest-path algorithm [93] to compute the forwarding paths. We compute the paths before and after the link down events, and derive the initial and final forwarding graphs. The considered link down events include both single link down and five links down events.

6.4 Performance Evaluation

One of the performance metrics of an update algorithm is the convergence time. Ideally, we want to convergence from an initial setting to a final setting as fast as possible. In particular, there should be no transient loop or transient link overflow. As transient link overflow might be inevitable so the amount of overflow could also be a performance metric.

If we analyze the convergence time in more detail, it could be observed the convergence time for an update process is the sum of update time in all update iterations. In each iteration, the update time includes the time to distribute the FIB, the time for routers to install the FIB on the line card, and the time for routers to acknowledge the update. If we assume these numbers are somehow similar for all routers, then it is the number of update iterations that has a large impact on the convergence time.

In the following subsection, we present the number of update iterations and the total amount of transient overflow. In addition, we present the number of changed FIBs and the possibility of finding at least one overflow-free updatable node in an iteration. We randomly select the down links and run the simulation for 100 times. In particular, we sort the experiments results in increasing order in the figure.

6.4.1 Number of Changed FIBs

When the forwarding paths change in a network, not all FIBs are affected by the changes. Therefore, we first present the number of changed FIBs. Since the number of changed FIBs varies for different changes in network topology, we study the number of changed FIBs in single link down events and five simultaneous link down events. In Fig. 6.8, the number of changed FIBs corresponding to these two types of events are shown. The link down events occur at random locations. Therefore, the number of changed FIBs also varies in different simulation runs. Each curve shows the results of 100 simulation runs for a specific type of event. As can be seen in the figure, single link down events only cause a small number of FIBs to change or even no change if the failed link is redundant. While five simultaneous link down events occur, the number of changed FIBs are much higher.

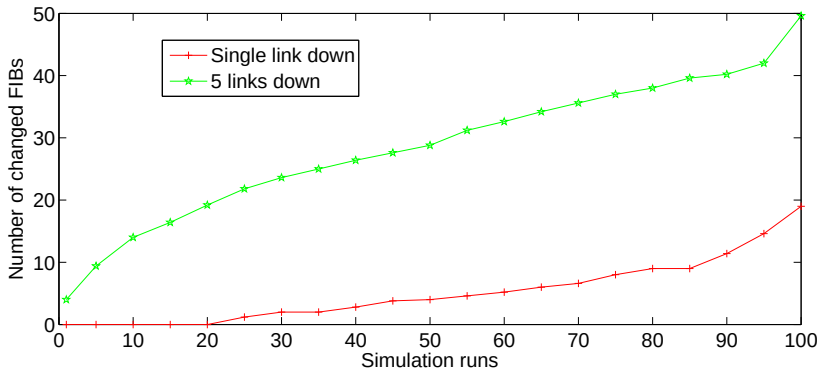


Figure 6.8: Number of changed FIBs from single link and 5 links failures.

6.4.2 Number of Update Iterations

We present the number of FIB update iterations in Fig. 6.9, using both our approach and an alternative approach presented in [4]. It could be observed that using our approach, more iterations are required to update the forwarding tables. Also, the number of iterations required for the five link down events is always larger than the single link down events.

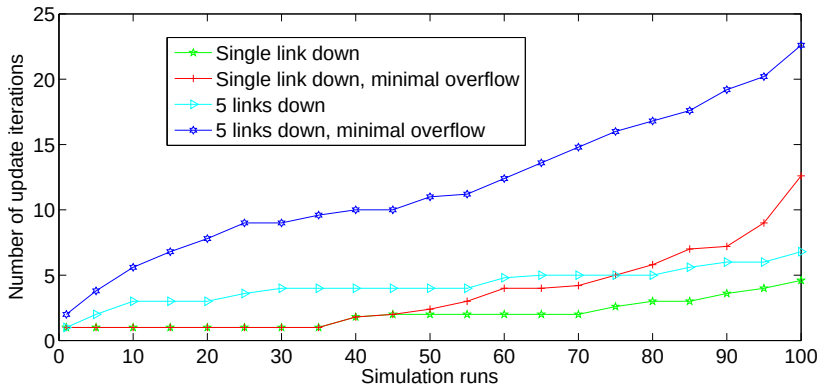


Figure 6.9: Number of update iterations, our approach vs an alternative approach in [4].

The increased iterations required to update all FIBs will cause longer convergence time. For the single link down event, around 50% of cases have same itera-

tions numbers which leads no convergence time difference. In the worst case, the iteration number using our approach could be around three times of the opposing approach, which is acceptable for the non-urgent forwarding path changes.

6.4.3 Probability of Finding Overflow-free Updatable Nodes

The probability of finding at least one overflow-free updatable node is shown in Fig. 6.10. This value is obtained by comparing the number of iterations that have at least one overflow-free updatable node to the total iteration number. As can be seen in the figure, in single link down events, for more than 30% of the cases the possibility is 100%, which means that there always exist a overflow-free updatable nodes in all iterations, and there will be no link overflow. In other cases, the probability is between 30% and 100%. Also, in five links down events, the probability of finding overflow-free updatable nodes are lower.

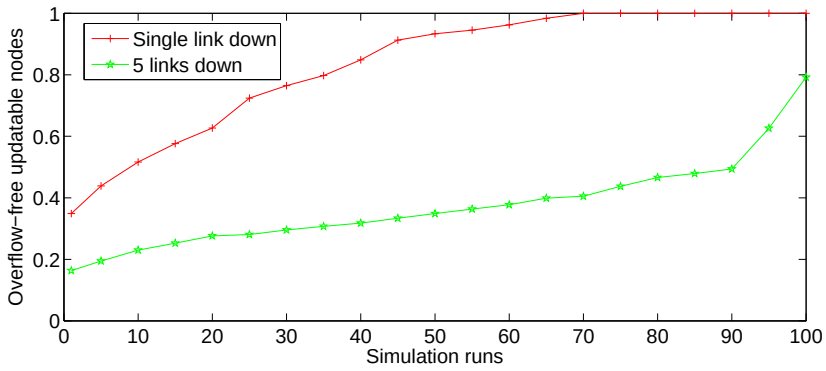


Figure 6.10: Probability of finding overflow-free updatable nodes.

6.4.4 Transient Traffic Overflow

Fig. 6.11 shows the total amount of traffic overflow during the entire transient phase. The numbers are the sum of overflow in all links in all transient forwarding matrices. As can be seen in the figure, using our approach, the amount of traffic overflow is significantly lower than using the update order that does not consider link overflow.

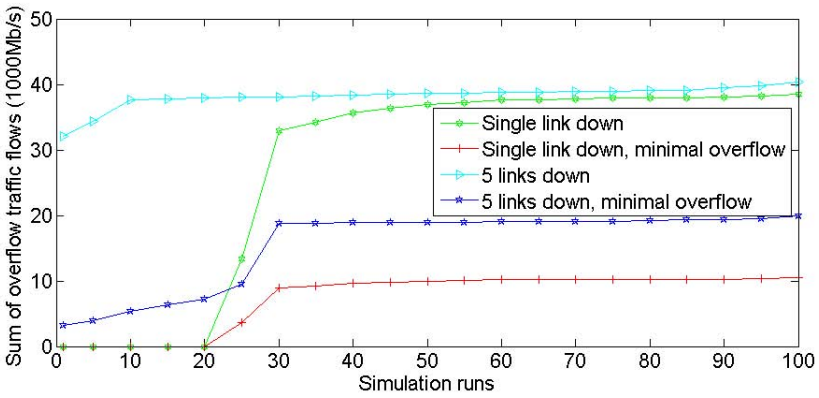


Figure 6.11: The amount of traffic overflow during the transient phase.

6.5 Summary

When the forwarding paths in a network change, the forwarding tables in the routers need to be updated. During the update process, transient loops may occur if it is not performed in an appropriate order. In this chapter, we present an approach to perform loop-free forwarding table updates and at the same time tries to minimize transient link overflow.

We first present an order that is loop-free and overflow free. However, this order may not exist as there may be no nodes that satisfy the overflow-free updatable conditions. In particular, we have proved that link overflow can be inevitable with an example. Thereafter, we propose a modified update order that tries to minimize the link overflow.

We have evaluated our algorithm on a real topology, with two types of forwarding paths change. The performance evaluation metrics include number of changed FIBs, number of update iterations and total amount of link overflow. Comparing our approach with the one presented in [4], our approach requires more update iterations, which will cause a longer convergence time, however, our approach achieves much smaller link overflow, which is a clear advantage.

Our work in transient routing loops is the first approach to consider congestion and link overflow. We believe that the idea of considering link overflow is interesting and future work will focus on investigating the update algorithms on more topologies, with varying traffic demands and types of forwarding paths changes.

Chapter 7

On Minimizing Network Wide Node Update Time

Communications networks are becoming increasingly complex and heterogeneous, at the same time, Internet traffic is rapidly expanding and also becoming more varied and complex. The value of communicating network traffic globally is becoming increasingly important and so is ensuring that these networks function optimally. In the maintenance of these networks, often nodes need to be rebooted or brought off line for software updates, configuration updates or hardware maintenance [12, 13]. ISPs often run the same protocols and use equipment from the same vendor network-wide, increasing the probability that a bug causes simultaneous failures [14, 15]. The extreme case is that all nodes in the network need to be rebooted for the maintenance / upgrade.

A simple and straightforward way is to update one node after another. This approach can achieve the best performance in terms of the affected traffic volumes. However, typically this solution is not optimal as it can take a long time to finish the update and for a large network it is not a scalable solution. We assume one router takes ten minutes to update, including the configuration time. For a network containing hundreds of nodes, it can easily take a few days to update all the nodes [13]. There is also an overhead associated to rebooting each node that can cause instability in routing.

A time-effective way is to update all the nodes in one update, in particular at the time when there is the least network traffic in the network. This method is the best in terms of time. However, this method does not consider the traffic the network carries which is potentially very valuable to the network operator. In this scenario, all network traffic will be lost. The challenge that we investigate in this chapter is how to make use of the network traffic properties among network nodes to effectively reboot the entire network, with a constraint that the network can still accommodate a certain percentage of network load.

The rest of the chapter is structured as follows. Section 7.1 describes the problem we target to solve. Rebooting algorithm we propose is explained in detail in Section 7.2. Section 7.3 introduces the experiments setup and the traffic matrix model we use; Section 7.4 describes the experiments we conducted and their anal-

ysis. Finally, Section 7.5 concludes this work and introduces directions for future work.

7.1 Problem Statement

To model the network, we use a graph $G = (V, E)$, where V is the set of nodes, E is the set of edges. Given this graph, we can construct a traffic matrix, which denotes the amount of data traffic transmitted between every pair of nodes in a network. To reboot all nodes in a network, we select a subset of nodes S_i to reboot at a point in time, where $S_1 \cup S_2 \cup \cdots \cup S_N = V$. By choosing an iteration sequence $S = S_1, S_2, \cdots, S_N$, such that during each iteration, all nodes in S_i are rebooted. This iteration will have an impact on overall affected traffic p% and we are interested in limiting p% to an upper bound.

Consider we have a network with N nodes, we also have all links with weights and link capacity information. Based on the topology, we can generate traffic demands so that the average link utilization is for example 30%. Consider we want to bring down node A for maintenance. Of course all traffic from node A and to node A is affected, we call this traffic self-associated traffic. For other traffic passing through node A, we can use OSPF to re-compute paths for that traffic. Ideally, after re-directing traffic, there is sufficient bandwidth available on the other links to support the re-routed traffic. In this case, we consider the lost traffic is only the self-associated traffic of the selected updatable nodes. If some traffic (traffic not from or to node A) will need to be dropped due to overflow after redirecting, the we consider that the upgrade causes the network to lose both the self-associated traffic and overflow traffic. The algorithm we have specified can be used to calculate how many nodes we can choose to reboot in parallel that preserves an upper bound on the total affected traffic. We investigate how different upper bounds on affected traffic impact the selection of updatable nodes on each iteration.

7.2 Algorithm

The number of ways to partition a set of n objects into k groups is a Stirling numbers of the second kind, denoted as $S(n, k)$ or $\left\{ {n \atop k} \right\}$. $S(n, k)$ can be calculated using the following explicit formula:

$$S(n,k) = \frac{1}{k!} \sum_{j=0}^{k} (-1)^{k-j} \binom{k}{j} j^n \tag{7.1}$$

Considering a network with n nodes, the possible sequences of rebooting r is

$$r = S(n,1) + \cdots + S(n,n) \tag{7.2}$$

which is the sum of the all possible partitions. If we have a 100 nodes network, r is $4.7585 \cdot 10^{115}$; it is impossible to check all the possibilities, no mention about even larger networks.

The intuition is that we try to find a set of nodes during each iteration that are closely related. By closely related we mean, nodes that send traffic to each other more than their other neighbor nodes. We then update these nodes together, because if one node is rebooting, the most related nodes will see the largest drop in traffic processing, so it is better to update all closely related nodes at the same time. Part of the loss of the network traffic is unavoidable when routers reboot themselves. Another reason is that some links have to carry more traffic during the rebooting phase and are overflowed.

To select a set of nodes in each iteration, the first step is to select the node with the most self-associated traffic, which means the sum of its sending and receiving traffic is the largest in the whole network. Since each node needs to update, its own generated traffic and received traffic has to be lost. Each time a node is selected, we test to see if the self-associated traffic and lost traffic (due to overflow) are more than the expected value (p%). We then gradually add the relevant nodes until such time that the threshold of p% is reached. For the case of minimal loss, p% is set to zero. The algorithm is shown as pseudo code in algorithm 1, algorithm 2 and algorithm 3.

Algorithm 1 UpdateNetwork

```
1:  procedure MAIN
2:      S ← Graph
3:      while S ≠ ∅ do
4:                                  ▷ Obtain the updatable nodes for each iteration
5:          updatablenodes ← obtainUpdatablenodes(S);
6:                                                  ▷ Compute overflowed traffic
7:          computeOverflow(updatablenodes);
8:                                              ▷ Compute average link utilization
9:          computeLinkUtilization(updatablenodes);
10:         S ← S − updatablenodes;
11:     end while
12: end procedure
```

If nodes are updated one by one, the self-associated traffic for each node and overflow traffic are affected, which is the minimum traffic that can be lost because

Algorithm 2 Compute Updatable Nodes

```
1: procedure COMPUTEUPDATABLENODES
2:     compute self-associated traffic for every node;
3:     updatableNodes = {the node associated with largest traffic volume};
4:     affectedTraffic = computeSelfTraffic(updatableNodes);
5:     while affectedTraffic ≤ targetTraffic do
6:         find the most related node r_i for each node i in updatablenodes;
7:         r = max{r_i};
8:         updatableNodes ← {updatableNodes, r}
9:         overflow = computeOverlow(updatablenodes);
10:        affectedTraffic =
11:        computeSelfTraffic(updatableNodes);
12:        if overflow + affectedTraffic > target then
13:            updatableNodes ← {updatableNodes} - r;
14:            return updatableNodes;
15:        end if
16:    end while
17:    return updatableNodes;
18: end procedure
```

each node has to be updated once anyway. However, the speed of this process is too slow and not scalable for large networks. The objective of the algorithm is to maintain some level of the affected network traffic (p%) and at the same time to update as many nodes as possible. The algorithm dictates that we pick up the node with the biggest traffic associated with it first. From the simulation result of taking nodes down one by one, we would know the affected network traffic by this single node, and then we try to bring down the most related node and so on.

In selecting the nodes for an iteration, the first selected node A is trivially selected, and the next node B is also trivially selected (next biggest neighbor). However, further nodes in the iteration could be neighbors of A or neighbors of B. The choice is made by iterating through all neighbors and selecting those that have the least impact on affected traffic threshold p%. Thus we are aiming to select that set of nodes, that when rebooted, will cause the least amount of overflow in the network. The strategy for deciding on the next node is breadth first search which aims to select nodes most related to A. If an iteration of nodes are selected that cause major routing problems in the network (i.e., segment the network), then the sequence is rolled back by one node. This is important as selected nodes are powered off simultaneously and thus any network segmentation would effectively render the network inactive.

Algorithm 3 Compute Overflow Traffic

```
1: procedure COMPUTEOVERLOW(updatablenodes)
2:     delete all the related links;
3:     set all related traffic demands to zero;
4:     G = G − {updatableNodes};
5:     find all shortest pathes between any two nodes in G;
6:     for each traffic demand d do
7:         run OSPF routing protocol to get shortest path;
8:         compute the link capacity bottleneck b of the path;
9:         update residual link capacity of G;
10:        if d > b then
11:            overflow ← overflow + d − b;
12:        end if
13:    end for
14:    return overflow;
15: end procedure
```

7.3 Test Setup

In this section, we present the experimental setup for our simulation study. We implemented a Java and Matlab simulation tool according to our algorithm to perform the study. In the following subsections, we present how the inputs of our algorithm are generated.

7.3.1 Network Topology

In the experiment we use a real ISP topology, which is measured in Rocketfuel based on the traceroute technique [91]. Some of their measured topologies have later been verified by the ISPs to prove their correctness. The topology we used is from AS 1755, the Ebone network in Europe, which consists of 87 backbone routers and 161 links. In addition, we use inferred link weights measured in [94].

7.3.2 Traffic Demands

The traffic demands we use are based on the Newton's gravity model of migration. The model is initially for migration of people between different towns. As the size of the towns increases, there will be an increase in movement between them. The further apart the two towns are, the movement between them will be less. The gravity model and its modified versions have been widely used in communication

networks [8, 95]. In our demand matrix generation, each node is given two uniformly distributed random variables o and i for outgoing and incoming traffic. The traffic demand from node s to t is specified as follows:

$$D(s,t) = \alpha \frac{o_s i_t}{d(s,t)^2},$$

where $d(s,t)$ denotes the distance and a constant α is used to scale the demands to appropriate values. We set α to an appropriate value so that the average link utilization is about 60% and 10%, representing traffic demands in the peak time and the night time. In the time period of network nodes updates, we assume the traffic demands are not changing.

7.4 Performance Evaluation

In this section we evaluate the performance of our proposed algorithm and validate its properties, namely convergence time and average link utilization. In addition, we obtained the updatable nodes for each iteration and update time for the whole network. We also examine the tradeoff between the amount of network traffic affected versus the length of the update process.

Performance evaluation includes two metrics. The first one is the update time, which is proportional to the number of update iterations. The second performance metric is the acceptable network traffic accommodated inside the network. We compare the result with the base method, which takes down the network one node at a time. The base method offers the best network capacity. However, the updatable time is much longer compared to our method. We try to offer the network capacity at some level by adjusting the percentage of affected traffic, and we obtain the sequences of nodes we want to update.

For the 10% average link utilization case, there is no link overflow before we start updating nodes. However, for the 60% average link utilization, some links are heavily loaded and link overflow does exist. It is due to the nature of the traffic matrix computed from the gravity model, the traffic demand between some nearby neighbors is much more than that of the far away nodes. We count the overflowed traffic as the original affected traffic and compare it with the new value of that after taking down network nodes.

During the experiments conducted with the Ebone backbone topology, we obtained the result by setting the percentage of average link utilization to 10% and 60% respectively. We also repeated the experiments for varying values of p%, namely the amount of affected traffic that the ISP is willing to cope with. Ebone

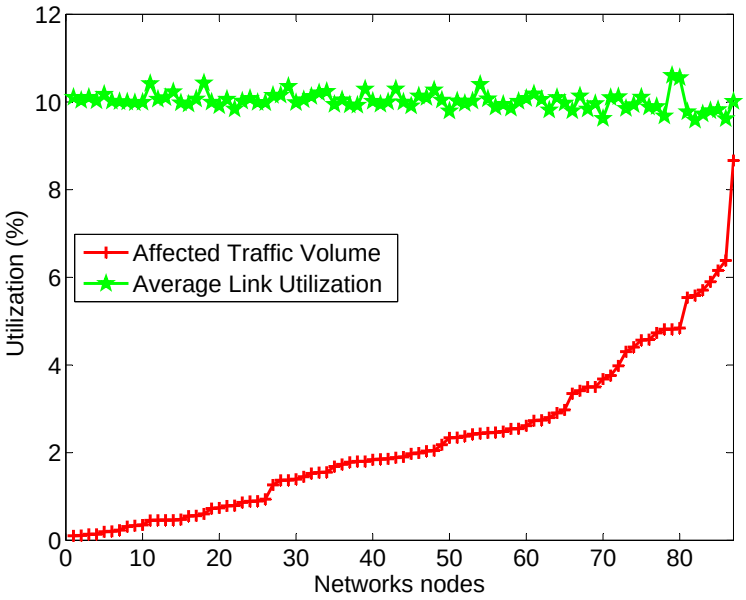


Figure 7.1: Update node one by one in the 10% average link utilization.

experimental results are shown in Fig. 7.1 and Fig. 7.2, for the case where each node is taken down one by one.

We can see that the general link utilization would not change much after one node is down. Generally, the average link utilization is reduced because the traffic associated with the node itself has disappeared. There might be a few nodes that are critical in routing traffic in the network, and after these nodes are down, the network traffic is significantly affected compared to other nodes. After sorting out the affected traffic volume in an increasing order, it goes up steadily compared to the total traffic matrix.

In the 10% average link utilization case, there is no link overflow on any links. This means that all remaining traffic can be routed around after the node is down. The figures include the traffic lost due to the outage of the nodes (i.e., the traffic to/from downed nodes).

Now we show the evaluation for the average link utilization cases 10% and 60%. The results are shown in Fig. 7.3 and Fig. 7.4 respectively. In these two cases the target affected traffic is 10%. This means that during the network wide update, up to 10% of traffic may be lost due to overflow and self-associated traffic loss.

The first observation is that there are less iterations when up to 10% of the

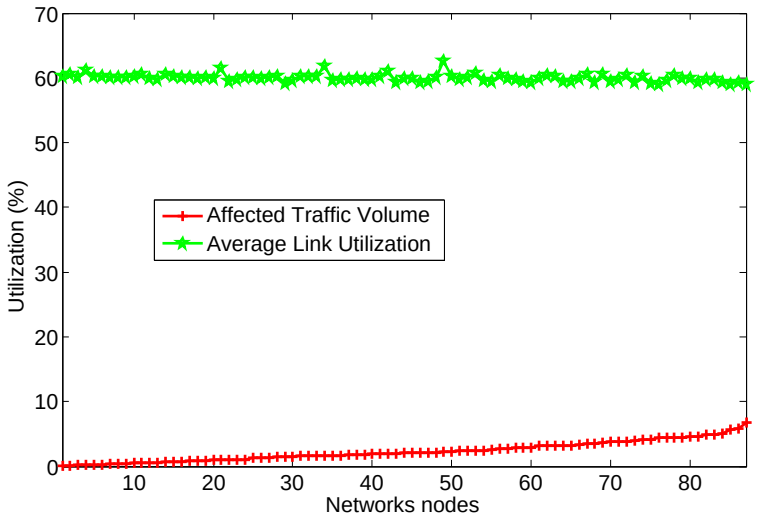


Figure 7.2: Update node one by one in the 60% average link utilization.

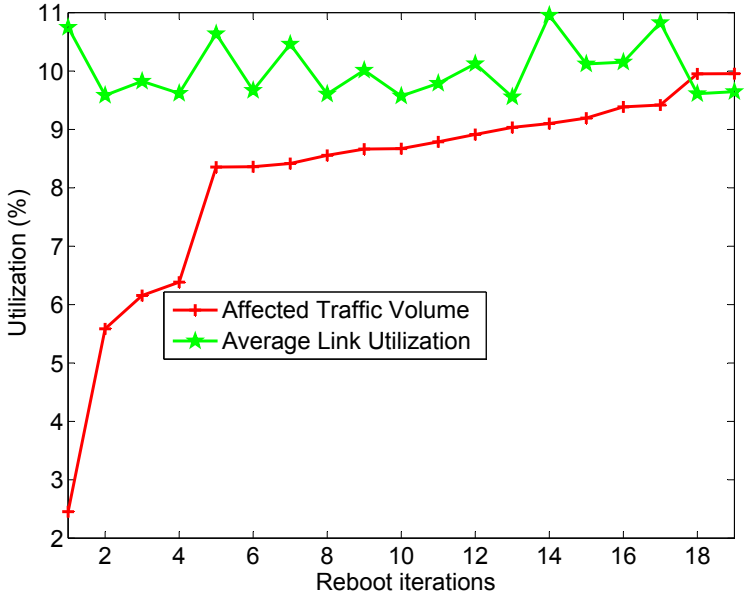


Figure 7.3: Affected traffic and average link utilization during the update process in 10% link utilization network.

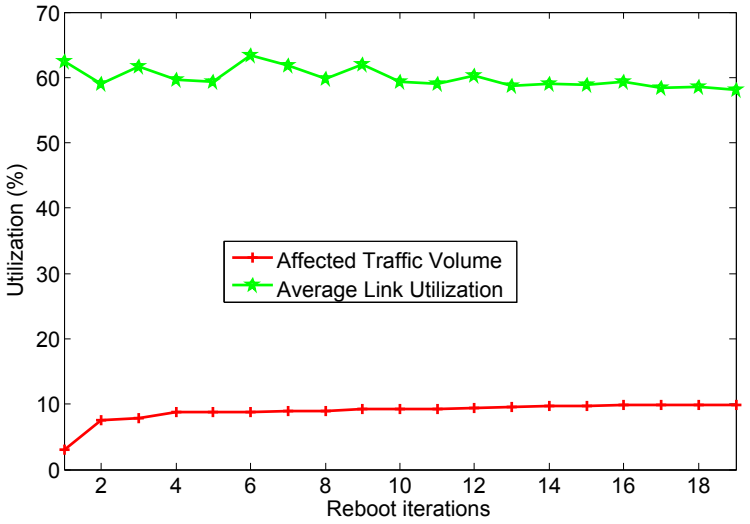


Figure 7.4: Affected traffic and average link utilization during the update process in 60% link utilization network.

traffic can be affected. Note in Fig. 7.4 that the affected traffic of 10% is achieved while there are only 19 iterations carried out, even in a relatively heavy loaded network. The experiment shows that the algorithm can be used to maintain an upper bound on the amount of affected traffic in the network.

Fig. 7.5 illustrates the impact the algorithm has on the number of iterations given a specified threshold of affected network traffic capacity. It shows that the number of iterations can be reduced, and thus the time to upgrade every node in the network. Specifically, in the case of 5% affected traffic, with 60% link utilization there is a reduction by 50% in the update time. The case of 20% affected traffic capacity shows the best performance if the priority for the network operator is to rapidly upgrade the network nodes. In Fig. 7.6 we show a comparison of the actual reduction in network capacity to calculate the upgrade process versus the upper bound on affect traffic. It shows that the algorithm can perform under these restrictions effectively. Given these results, network operators can have more control over the time it takes to upgrade the nodes in their network and can have also give priority to affected network traffic depending on the urgency of the network upgrade.

The sequence of iterations generated during the experiment are listed in Table 7.1 and Table 7.2. These apply to an affected traffic bound of 10% for network

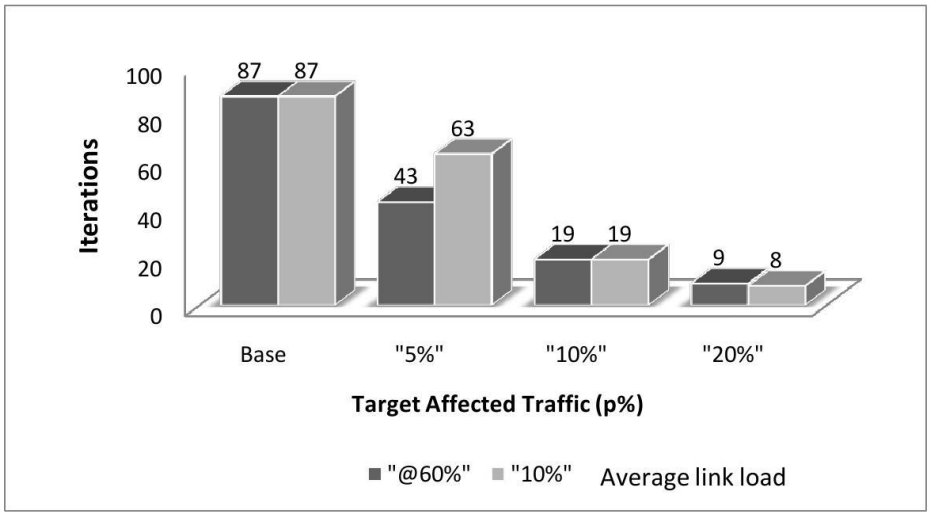


Figure 7.5: Comparison of the number of iterations vs. affected traffic.

link utilization of 10% and 60% respectively.

7.5 Summary

Network operators are finding it difficult to avoid network wide upgrades due to system maintenance and the increased demand for capacity. We studied the issue of rebooting the whole network in order to facilitate network upgrades. We proposed an optimal rebooting algorithm based on the idea of simultaneously rebooting traffic related nodes. The algorithm presented takes as a parameter an upper bound on the volume of affected traffic to calculate the upgrade sequence. For our test topology, different affected traffic bounds were investigated to test how efficiently a network upgrade could be completed. Our simulation studies show that while maintaining the network affected traffic to p% percentage of the total network traffic, our method provides an efficient way of rebooting nodes, hence the time of rebooting the whole network. Future work will investigate the impact of topologies of various sizes to performance and a more through analysis of the scalability of the algorithm.

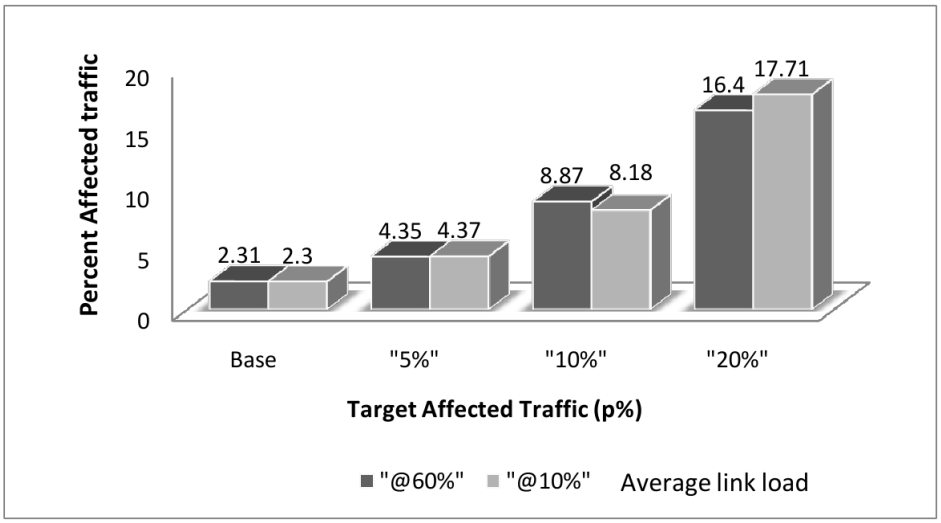


Figure 7.6: Comparison of the actual reduction in capacity vs. the target affected traffic.

Table 7.1: Updatable Nodes Sequences in 60% Link Utilization Network.

Iteration	Updatable nodes
1	6 79
2	8 19
3	35 10
4	40 36
5	46 71 58
6	33 32 31
7	24 28 37
8	1 11 2
9	44 67 45
10	73 26 80 27
11	47 9 75 87 20
12	4 81 66 64
13	29 22 21 41
14	30 83 74 57 70 68 69
15	38 43 42 23 85 86
16	34 48 82 65 39 55
17	16 62 12 15 17 56 14 60 61 13 49 63
18	72 78 84 77 52 59 51 76 53 25 54
19	5 7 50 18 3

Table 7.2: Updatable Nodes Sequences in 10% Link Utilization Network.

Iteration	Updatable nodes
1	1
2	6
3	8
4	28 24
5	2 11
6	19
7	67 45 44
8	41 43 42 38 37
9	4 40
10	83 74 26 73
11	22 21 29 27 30
12	46 71 72 58
13	12 17 15 16 62 61 14 56 60 63
14	25 54 53 55 57 70
15	10 35 20 5 7
16	33 31 34 32 48 82
17	9 47 87 75 76 59 52 51 77 78 84 81
18	66 65 64 39 36 23 85 86 49 13 50 69 68
19	80 79 18 3

Chapter 8

Conclusion and Future Work

In this thesis, network monitoring and fault management have been investigated. For the network monitoring, path-based monitoring provided a general approach to collect performance metrics on a path within a network between any two routers, which can aid in the assessment of the performance of any individual path. In the fault management, two issues were studied. The work on loop-free updates illustrates how routers can be updated in an order to avoid transient loops and link overflow. In particular, this approach can be applied to any routing protocols and any type of forwarding path changes. The study on the network wide reboot provided an algorithm to efficiently reboot the whole network. Performance results show a significant improvement in terms of overall upgrade time and scalability which indicates the properties of this efficient solution with respect to the minimizing impact on network traffic.

The proposed network monitoring, fault management and recovery schemes are evaluated through extensive analysis and experiments. Results show that our monitoring approach only generates less than 5% of the traffic generated by traceroute, at only around 12% of the time taken by traceroute, to retrieve information for a 16-node network, our fault management method can achieve zero transient loop with minimal link overflow, and our fault recovery scheme can significantly reduce rebooting time (86.78% lower that the traditional approach by rebooting network node one by one).

The results presented in this thesis are useful for building next-generation IP networks. For IPv6 networks, this study has pointed out the features in IPv6 protocols that could be exploited or extended for purposes of designing or building better networks and services. The study in network management also suggests that centralization of the control functions might be an interesting alternative to current network architecture. A centralized control scheme brings important benefits as it simplifies the decision process and reduces the functions required in routers.

There remain open issues to address; in particular, several interesting research problems have been identified and are currently under study:

1. For the path-based monitoring, interesting future work is to (1) improve the current systems (2) conduct measurement in different environments (3) ex-

plore new applications of path-based monitoring (4) identify the fundamental performance metrics that are needed for next-generation networks and applications. This effort hopefully can identify the general principles that can be used to guide the design and development of path-based monitoring and management components for the next-generation network architectures.

2. Several security issues will need further investigation: 1) Due to the expensive computational cost of the IKEv2 exchange, which mainly includes the Diffie-Hellman key exchange, certificate handling and the authentication processing, IKEv2 requires quite some time for session establishment. The monitoring may conduct periodically to collect performance information to ensure the optimized operations of communication networks. Essential future work will seek fast re-establishment of the sessions when session expires or network problems occur. 2) An efficient and optimized compression method for monitored information in order to disseminate collected performance records efficiently.

3. To minimize the network wide rebooting time and link overflow of loop-free forwarding table updates, results obtained here are based on the assumption that the traffic demands are constant during the update period. It would be interesting to study whether this has some effect on the results. Also a different traffic model could be used to verify the benefits of the proposed methods.

4. For the algorithm of minimizing the network wide rebooting time, one could focus on its optimality or expected performance. Elastic Tree algorithm [70] could be modified to produce rebooting schedules. For instance, after each run of the algorithm, all previously selected nodes can be marked as "unavailable" so that they are not selected again.

5. Instead of rebooting a wired network, a wireless network with nodes on a 2D map is considered where each node has a coverage area. There are mobile users located in this 2D map in accordance to some kind of distributions. The problem could be that there are n maintenance teams to update the nodes (software, hardware updates, etc.) and minimization of the service disruption and update time during the update process is desired. To solve this problem, base stations' location, capacity, coverage, and mobile user's demands and so on need to be studied.

Bibliography

[1] A. Davy, K. Quinn, L. Shi *et al.*, "Autonomic monitoring node functionality and dissemination of information," EFIPSANS Project Deliverable, December 2008.

[2] N. Tcholtchev, A. Davy *et al.*, "Components and mechanisms for autonomic fault management," EFIPSANS Project Deliverable, December 2009.

[3] F. Li and M. Thottan, "End-to-end service quality measurement using source-routed probes," in *INFOCOM 2006. 25th IEEE International Conference on Computer Communications. Proceedings*, april 2006, pp. 1 –12.

[4] J. Fu, P. Sjödin, and G. Karlsson, "Loop-free updates of forwarding tables," *IEEE Transactions on Network and Service Management*, vol. 5, no. 1, pp. 22–35, March.

[5] ITU-T, "TMN Management Functions," *Telecommunication Standardization sector of ITU*, 2000. [Online]. Available: http://www.itu.int/rec/T-REC-M.3400-200002-I/en

[6] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg, "Fast accurate computation of large-scale ip traffic matrices from link loads," in *SIGMETRICS '03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*. New York, NY, USA: ACM, 2003, pp. 206–217.

[7] Y. Zhang, M. Roughan, C. Lund, and D. Donoho, "An information-theoretic approach to traffic matrix estimation," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. New York, NY, USA: ACM, 2003, pp. 301–312.

[8] A. Medina, N. Taft, K. Salamatian, S. Bhattacharyya, and C. Diot, "Traffic matrix estimation: existing techniques and new directions," *SIGCOMM Comput. Commun. Rev.*, vol. 32, no. 4, pp. 161–174, 2002.

[9] V. Paxson, "Towards a framework for defining internet performance metrics," in *Proc. INET '96*, 1996.

[10] J. J. Garcia-Lunes-Aceves, "Loop-free routing using diffusing computations," vol. 1, no. 1. Piscataway, NJ, USA: IEEE Press, 1993, pp. 130–141.

[11] J. J. Garcia-Luna-Aceves, "A unified approach to loop-free routing using distance vectors or link states," in *SIGCOMM '89: Symposium proceedings on Communications architectures & protocols*. New York, NY, USA: ACM, 1989, pp. 212–223.

[12] E. Keller, M. Yu, M. Caesar, and J. Rexford, "Virtually Eliminating Router Bugs," *in ACM CoNEXT*, 2009.

[13] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford, "Virtual routers on the move: live router migration as a network-management primitive," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 4, pp. 231–242, 2008.

[14] J. Duffy, "BGP bug bites Juniper software," *In Network World*, December 2007.

[15] J. Evers, "Trio of Cisco flaws may threathen networks," *In CNET News*, January 2007.

[16] N. Spring, R. Mahajan, and D. Wetherall, "Measuring ISP Topologies with Rocketfuel," in *In Proc. ACM SIGCOMM*, 2002, pp. 133–145.

[17] B. Cain, S. Deering, I. Kouvelas, B. Fenner, and A. Thyagarajan, "Internet Group Management Protocol, Version 3," RFC 3376 (Proposed Standard), Internet Engineering Task Force, Oct. 2002, updated by RFC 4604. [Online]. Available: http://www.ietf.org/rfc/rfc3376.txt

[18] H. Liu, W. Cao, and H. Asaeda, "Lightweight Internet Group Management Protocol Version 3 (IGMPv3) and Multicast Listener Discovery Version 2 (MLDv2) Protocols," RFC 5790 (Proposed Standard), Internet Engineering Task Force, Feb. 2010. [Online]. Available: http://www.ietf.org/rfc/rfc5790.txt

[19] Y. Rekhter, T. Li, and S. Hares, "A Border Gateway Protocol 4 (BGP-4)," RFC 4271 (Draft Standard), Internet Engineering Task Force, Jan. 2006. [Online]. Available: http://www.ietf.org/rfc/rfc4271.txt

[20] G. Malkin and R. Minnear, "RIPng for IPv6," RFC 2080 (Proposed Standard), Internet Engineering Task Force, Jan. 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2080.txt

[21] R. Coltun, D. Ferguson, J. Moy, and A. Lindem, "OSPF for IPv6," RFC 5340 (Proposed Standard), Internet Engineering Task Force, Jul. 2008. [Online]. Available: http://www.ietf.org/rfc/rfc5340.txt

[22] Y. Rekhter, "IPv6 Address Specific BGP Extended Community Attribute," RFC 5701 (Proposed Standard), Internet Engineering Task Force, Nov. 2009. [Online]. Available: http://www.ietf.org/rfc/rfc5701.txt

[23] S. Kent, "IP Encapsulating Security Payload (ESP)," RFC 4303 (Proposed Standard), Dec. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4303.txt

[24] ——, "IP Authentication Header," RFC 4302 (Proposed Standard), Dec. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4302.txt

[25] S. Hagen, *IPv6 Essentials*. O'Reilly Media, Inc., 2006.

[26] D. Johnson, C. Perkins, and J. Arkko, "Mobility Support in IPv6," RFC 3775 (Proposed Standard), Internet Engineering Task Force, Jun. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3775.txt

[27] C. Partridge and A. Jackson, "IPv6 Router Alert Option," RFC 2711 (Proposed Standard), Internet Engineering Task Force, Oct. 1999. [Online]. Available: http://www.ietf.org/rfc/rfc2711.txt

[28] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin, "Resource ReSerVation Protocol (RSVP)," *IETF Network Working Group RFC 2205*, 1997, available: 08-10-2008, online at: http://www.ietf.org/rfc/rfc2205.txt. [Online]. Available: http://www.ietf.org/rfc/rfc2205.txt

[29] J. Rexford, A. Greenberg, G. Hjalmtysson, D. Maltz, A. Myers, G. Xie, J. Zhan, and H. Zhang, "Network-wide decision making: Toward a wafer-thin control plane," in *ACM SIGCOMM HotNets Workshop*, November 2004.

[30] A. Greenberg, G. Hjalmtysson, D. Maltz, A. Myers, J. Rexford, G. Xie, H. Yan, J. Zhan, and H. Zhang, "Clean slate 4d approach to network control and management," in *ACM SIGCOMM Computer Communication Review*, vol. 35, no. 5, October 2005.

[31] A. Csaszar, G. Enyedi, G. Retvari, M. Hidell, and P. Sjodin, "Converging the evolution of router architectures and ip networks," *Network, IEEE*, vol. 21, no. 4, pp. 8 –14, july-august 2007.

[32] R. Chaparadza, "Requirements for a generic autonomic network architecture (gana), suitable for standardizable specifications of decision-making-elements (dmes) for diverse networking environments," *The International Engineering Consortium (IEC), the Annual Review in Communications*, vol. 61, December 2008.

[33] J. Bell, "Understand the autonomic manager concept," Feb 2004. [Online]. Available: http://www.ibm.com/developerworks/library/ac-amconcept/

[34] "Resilinets project definitions." [Online]. Available: https://wiki.ittc.ku.edu/resilinets_wiki/index.php/Definitions

[35] S. Rai, B. Mukherjee, and O. Deshpande, "Ip resilience within an autonomous system: current approaches, challenges, and future directions," *Communications Magazine, IEEE*, vol. 43, no. 10, pp. 142 – 149, oct. 2005.

[36] A. Durand and L. Toutain, "Ipv6 traceroute option," *IPv6 Working Group Internet Draft*, vol. http://www.join.uni-muenster.de/Dokumente/drafts/draft-durand-ipv6-traceroute-00.txt, June 1997.

[37] B. Augustin, X. Cuvellier, B. Orgogozo, F. Viger, T. Friedman, M. Latapy, C. Magnien, and R. Teixeira, "Avoiding traceroute anomalies with Paris traceroute," *6th ACM SIGCOMM conference on Internet measurement*, pp. 153–158, 2006.

[38] J. Quittek, T. Zseby, B. Claise, and S. Zander, "Requirements for IP Flow Information Export (IPFIX)," RFC 3917 (Informational), Internet Engineering Task Force, Oct. 2004. [Online]. Available: http://www.ietf.org/rfc/rfc3917.txt

[39] R. Stewart, "Stream Control Transmission Protocol," RFC 4960 (Proposed Standard), Internet Engineering Task Force, Sep. 2007. [Online]. Available: http://www.ietf.org/rfc/rfc4960.txt

[40] N. Hu, E. L. Li, Z. M. Mao, P. Steenkiste, and J. Wang, "Locating internet bottlenecks: algorithms, measurements, and implications," in *SIGCOMM*, 2004, pp. 41–54.

[41] V. J. Ribeiro, R. H. Riedi, R. G. Baraniuk, J. Navratil, and L. Cottrell, "pathchirp: Efficient Available Bandwidth Estimation for Network Paths," in *Passive and Active Measurement Workshop*, San Diego, CA, USA, April 2003.

[42] W. Stallings, *SNMP,SNMPV2,Snmpv3,and RMON 1 and 2*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1998.

[43] D. Harrington, R. Presuhn, and B. Wijnen, "An Achitecture for Describing Simple Network Management Protocol (SNMP) Management Frameworks," *IETF Network Working Group RFC 3411*, 2002, available: 08-10-2008, online at: http://www.ietf.org/rfc/rfc3411.txt. [Online]. Available: http://www.ietf.org/rfc/rfc3411.txt

[44] J. Case, M. Fedor, M. Schofstall, and J. Davin, "A Simple Network Management Protocol (SNMP)," *IETF Network Working Group RFC 1157*, 1990, available: 08-03-2008, online at: http://www.ietf.org/rfc/rfc1157.txt. [Online]. Available: http://www.ietf.org/rfc/rfc1157.txt

[45] "Snmp version 3 (snmpv3)." [Online]. Available: http://www.ibr.cs.tu-bs.de/projects/snmpv3/

[46] L. Andrey, O. Festor, A. Lahmadi, A. Pras, and J. Schönwälder, "Survey of SNMP performance analysis studies," *Int. Journal of Network Management*, vol. 19, no. 6, pp. 527–548, 2009.

[47] H. Kitamura, "Connection/link status investigation (csi) for ipv6 hop-by-hop option and icmpv6 message extension," *IETF Networking Group, Internet Draft*, 1999.

[48] M. Crocker, G. Lazarou, J. Baca, and J. Picone, "A bandwidth determination method for ipv6-based networks," *International Journal of Computers and Applications*, vol. 31, 2009.

[49] D. P. Pezaros, D. H. F. J. Garcia, R. D. Gardner, and J. S. Sventek, "In-line service measurements: An ipv6-based framework for traffic evaluation and network operations," in *IEEE NOMS 2004, Seoul Korea*, 2004.

[50] "Next steps in signaling (nsis)." [Online]. Available: http://datatracker.ietf.org/wg/nsis/charter/

[51] X. Fu, H. Schulzrinne, H. Tschofenig, C. Dickmann, and D. Hogrefe, "Overhead and performance study of the general internet signaling transport (gist) protocol," *IEEE/ACM Trans. Netw.*, vol. 17, no. 1, pp. 158–171, 2009.

[52] J. Quittek, F. Raspall, M. Brunner, and M. Martin, "Path-coupled configuration of traffic measurement," 2004. [Online]. Available: http://www.6qm.org/workshop/slides/quittek-6QM-NSIS-measurement.pdf

[53] J. Manner, R. Bless, J. Loughney, and E. E B. Davies, "Using and extending the nsis protocol family," IETF Internet Draft, April 2010. [Online]. Available: http://www.rfc-editor.org/internet-drafts/draft-ietf-nsis-ext-07.txt

[54] C. Dickmann, I. Juchem, S. Willert, and X. Fu, "A stateless ping tool for simple tests of gimps implementations," IETF Internet Draft, May 2005. [Online]. Available: http://user.informatik.uni-goettingen.de/~fu/nsis/draft-juchem-nsis-ping-tool-01.html

[55] R. Hancock, "Using the router alert option for packet interception in gist," IETF Internet Draft, November 2008. [Online]. Available: http://tools.ietf.org/html/draft-hancock-nsis-gist-rao-00

[56] "Cisco ios ip slas," December 2005. [Online]. Available: http://www.cisco.com/en/US/docs/ios/12_4/ip_sla/configuration/guide/hsoverv.html#wp1027129

[57] K. Wu and D. Reeves, "Capacity Planning of DiffServ Networks with Best-Effort and Expedited Forwarding Traffic," in *Telecommunication Systems*, vol. 25, no. 3. Springer, 2004, pp. 193–207.

[58] A. Feldmann, A. Greenberg, C. Lund, N. Reingold, J. Rexford, and F. True, "Deriving Traffic Demands for Operational IP Networks: Methodology and Experience," in *IEEE/ACM Transactions on Networking*, vol. 9, no. 3, 2001, pp. 265–279.

[59] A. Medina, C. Fraleigh, N. Taft, S. Battacharryya, and C. Diot, "A taxonomy of IP traffic matrices," in *SPIE proceedings series*, 2002, pp. 200–213.

[60] N. Duffield and M. Grossglauser, "Trajectory sampling for direct traffic observation," in *IEEE/ACM Transactions on Networking*, vol. 9, no. 3. New York, NY, USA: ACM Press, 2001, pp. 280–292.

[61] G. Sadasivan, N. Brownlee, B. Claise, and J. Quittek, "Architecture for IP Flow Information Export," *IETF IP Flow Information Export WG - Internet-Draft*, 2006, available: 08-10-2008, online at: http://www.ietf.org/internet-drafts/drafts-ietf-ipfix-architecture-12.txt.

[62] B. Claise, "Cisco Systems NetFlow Service Export Version 9," *IETF Network Working Group RFC 3954*, 2004, available: 08-10-2008, online at: http://www.ietf.org/rfc/rfc3954.txt.

[63] S. Kent and K. Seo, "Security Architecture for the Internet Protocol," RFC 4301 (Proposed Standard), Dec. 2005. [Online]. Available: http://www.ietf.org/rfc/rfc4301.txt

[64] X. Zhao, Y. Liu, L. Wang, and B. Zhang, "On the aggregatability of router forwarding tables," in *INFOCOM, 2010 Proceedings IEEE*, 14-19 2010, pp. 1 –9.

[65] S. Bryant, C. Filsfils, S. Previdi, and M. Shand, "Ip fast reroute using tunnels," in *Internet draft*. draft-bryant-ipfrr-tunnels-00.txt, May 2004, work in progress.

[66] P. Francois and O. Bonaventure, "Avoiding transient loops during igp convergence in ip networks," in *IEEE INFOCOM*, March 2005.

[67] P. Francois, O. Bonaventure, M. Shand, S. Bryant, S. Previdi, and C. Filsfils, "Loop-free convergence using ofib," Internet Draft, March 2010, draft-ietf-rtgwg-ordered-fib-03.txt.

[68] P. Francois and O. Bonaventure, "Avoiding transient loops during the convergence of link-state routing protocols," *IEEE/ACM Transactions on Networking*, vol. 15, no. 6, pp. 1280–1932, December 2007.

[69] P. Francois, M. Shand, and O. Bonaventure, "Disruption Free Topology Reconfiguration in OSPF Networks," in *IEEE Infocom*, May 2007.

[70] B. Heller, D. Underhill, and N. McKeown, "ElasticTree: Saving energy in data center networks," *NSDI 2010*, April 2010.

[71] R. Chaparadza, "A composition language for programmable traffic flow monitoring in multi-service self-managing networks," oct. 2007, pp. 1 –8.

[72] ——, "Improving the automation of network management and troubleshooting tasks by applying on-demand monitoring of protocol(s) specific traffic in multi-service networks," in *5th IEEE International Workshop on IP Operations & Management, IPOM 2005*, October 2005.

[73] R. Chaparadza, H. Coskun, and I. Schieferdecker, "Addressing some challenges in autonomic monitoring in self-managing networks," vol. 2, nov. 2005, p. 6 pp.

[74] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*. Addison-Wesley, 1994.

[75] CAIDA, "CAIDA Anonymized 2008 Internet Traces dataset," 2008, available: 08-10-2008, online at: https://data.caida.org/datasets/passive-2008/.

[76] L. Shi and A. Davy, "Intrinsic monitoring within an ipv6 network: Relating traffic flows to network paths," *International Conference on Communications (ICC)*, 2010.

[77] A. Gunnar, M. Johansson, and T. Telkamp, "Traffic Matrix Estimation on a Large IP Backbone - A Comparison on Real Data," in *Proc. of 4th ACM SIGCOMM Internet Measurement Conference*. New York, NY, USA: ACM Press, 2004, pp. 149–160.

[78] K. Papagiannaki, N. Taft, and A. Lakhina, "A distributed approach to measure IP traffic matrices," in *Proc. of the 4th ACM SIGCOMM Conference on Internet Measurement*, 2004, pp. 161–174.

[79] M. Roughan, M. Thorup, and Y. Zhang, "Traffic engineering with estimated traffic matrices," in *Proc. of the 3rd ACM SIGCOMM Conference on Internet Measurement*, 2003, pp. 248–258.

[80] Y. Zhang, M. Roughan, C. Lung, and D. L. Donoho, "Estimating Point-to-Point and Point-to-Multipoint Traffic Matrices: An Information-Theoretic Approach," in *IEEE/ACM Transactions on Networking*, vol. 13, no. 5, 2005, pp. 947–960.

[81] S. Deering and R. Hinden, "Internet protocol, version 6," *IETF Network Working Group RFC 2460*, December 1998.

[82] N. Duffield, "A Framework for Packet Selection and Reporting," *IETF Packet Sampling WG (PSAMP) - Internet-Draft*, 2004, available: 08-10-2008, online at: http://www.ietf.org/internet-drafts/draft-ietf-psamp-framework-10.txt. [Online]. Available: http://www.ietf.org/internet-drafts/draft-ietf-psamp-framework-10.txt

[83] T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall, "Sampling and Filtering Techniques for IP Packet Selection," *IETF Packet Sampling WG (psamp) - Internet-Draft*, 2007, online at: http://www.ietf.org/internet-drafts/draft-ietf-psamp-sample-tech-10.txt.

[84] C. Kaufman, "Internet Key Exchange (IKEv2) Protocol," RFC 4306 (Proposed Standard), Dec. 2005, updated by RFC 5282. [Online]. Available: http://www.ietf.org/rfc/rfc4306.txt

[85] L. Shi and A. Davy, "Security considerations for intrinsic monitoring within ipv6 networks," *IEEE 9th International Workshop on IP Operations and Management*, October 2009.

[86] S. M. Ross, *Stochastic Processes, Second Edition*. John Wiley & Sons, Inc., 1996.

[87] ——, *Introduction to Probability Models, Ninth Edition*. Orlando, FL, USA: Academic Press, Inc., 2006.

[88] G. Iannaccone, C.-n. Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, "Analysis of link failures in an ip backbone," in *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. New York, NY, USA: ACM, 2002, pp. 237–242.

[89] D. Katz, "IP Router Alert Option," RFC 2113 (Proposed Standard), Internet Engineering Task Force, Feb. 1997, updated by RFC 5350. [Online]. Available: http://www.ietf.org/rfc/rfc2113.txt

[90] G. Iannaccone, C. nee Chuah, R. Mortier, S. Bhattacharyya, and C. Diot, "Analysis of link failures in an ip backbone," in *In Proc. of the Internet Measurement Workshop*. ACM, 2002, pp. 237–242.

[91] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson, "Measuring ISP topologies with rocketfuel," *IEEE/ACM Trans. Netw.*, vol. 12, no. 1, pp. 2–16, February 2004.

[92] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring link weights using end-to-end measurements," in *IMW '02: Proceedings of the 2nd ACM SIGCOMM Workshop on Internet measurment*. New York, NY, USA: ACM, 2002, pp. 231–236.

[93] T. H. Cormen, C. Stein, R. L. Rivest, and C. E. Leiserson, *Introduction to Algorithms*. McGraw-Hill Higher Education, 2001.

[94] R. Mahajan, N. Spring, D. Wetherall, and T. Anderson, "Inferring link weights using end-to-end measurements," in *In ACM SIGCOMM Internet Measurement Workshop*, 2002, pp. 231–236.

[95] B. Fortz and M. Thorup, "Optimizing ospf/is-is weights in a changing world," *IEEE Journal on Selected Areas in Communications*, vol. 20, no. 4, pp. 756–767, May 2002.

Lei Shi

PERSONAL DATA

Address: Poplar Drive 79
Six Cross Roads
Waterford, Ireland
Voice: (+353) 877580681
E-mail: robustness@gmail.com
Sex: Male
Born: July 7, 1979

RESEARCH INTERESTS

Router architecture, Network management, IPv6 networks, Network monitoring, Fault management, Peer-to-peer video streaming, Routing protocols, Network processors

EDUCATION

University of Göttingen, Göttingen, Germany

Ph.D, Computer Networks — March, 2007 - 2010 (expected)
Advisor: Prof. Xiaoming Fu and Prof. Dieter Hogrefe, Univ. of Göttingen.

Uppsala University, Uppsala, Sweden

Master, Computer Science — September, 2005 - December, 2006
Master Thesis: The Implementation of Forwarding Elements on IXP Network processor
Advisor: Associate Professor Peter Sjödin, Royal Institute of Technology (KTH), Sweden

Chongqing University, Chongqing, China

BS with honors, Computer Science — September, 1997 - July, 2001

PROFESSIONAL EXPERIENCE

TSSG, Waterford, Ireland

Researcher — September, 2008 - Now

I work as a researcher for FP7 EFIPSANS project. The EFIPSANS project aims at exposing the features in IPv6 protocols that can be exploited or extended for the purposes of designing or building autonomic networks and services. My work focuses on the IPv6 network monitoring and currently I am involving with IETF standardization activities. Apart from the EFIPSANS project, I also participated in some proposal applications and reviews.

University of Goettingen, Goettingen, Germany

Research assistant — March, 2007 - September, 2008

My research focused on network architecture, protocols and their performance evaluation. I designed a network-processor based TCP proxy to improve the TCP performance. Besides that I also conducted some projects related with peer to peer video streaming and centralized-control IP network.

Huawei Technologies, Beijing, China

Software engineer — April, 2004 - August, 2005

Developed Quidway NetEngine 5000E Terabit Switching Router (TSR) based on IXP2800 network processors. My work focused on performance of core routers where the flexibility and programmability are achieved by network processors. I participated in designing protocols such as MPLS and IPv6 for the data plane. Other modules such as CSIX RX/TX, SPI-4 RX/TX and QoS are also analyzed and implemented. In addition, I designed and implemented the solution of sending fragment packets and jumbo frames.

HUTCHISON OPTEL Telecom Technology, Beijing, China

Software engineer February, 2001 - March, 2004

Developed the routing protocols and software for MSTP and access-routers. I transplanted communication protocols such as RIP, TCP, UDP, ICMP and IGMP to our hardware platform IMAP (intelligence multi-access platform). I also tested the whole device using traffic generator Smartbits and done some research work about MPLS. In addition, I implemented the RPR protocol for the data plane of our system and the software was successfully tested on the hardware platform.

Teaching Experience

- Praktikum-Telematik, Teaching Assistant, University of Göttingen, 2008
- Advanced Topics in Computer Networking, Teaching Assistant, University of Göttingen, 2007
- Computer Networks, Teaching Assistant, University of Göttingen, 2007

Publications

- Lei Shi, Alan Davy, David Muldowney, Steven Davy, Edzard Hoefig, Xiaoming Fu, "Intrinsic Monitoring within an IPv6 Network: Mapping Node Specific information to Network Paths", International Conference on Network and Service Management (CNSM), Niagara Falls, Canada, Oct 2010.
- Lei Shi and Alan Davy, Intrinsic Monitoring within an IPv6 Network: Relating Traffic Flows to Network Paths, IEEE International Conference on Communications (ICC), Cape Town, South Africa, May 2010.
- Lei Shi and Steve Davy, On Minimizing the Network Nodes Update Time, IEEE GLOBECOM 2010, MIAMI, USA, Dec 2010.
- Lei Xu, Lei Shi, Runxing Wang and Brendan Jennings,"A Multiple Criteria Service Composition Selection Algorithm Supporting Time-sensitive Rules", IFIP/IEEE IM 2011, Dublin, Ireland, May 2011
- Runxin Wang, Lei Shi, M*í*che*á*l *ó* Foghl*ú*, "Using Meta Approach to Learn Data with Concept Drift", International Conference on Knowledge Discovery and Information Retrieval (KDIR 2010), Valencia, Spain, Oct. 2010.
- Jun Lei, Lei Shi, Xiaoming Fu, An Experimental Analysis of Joost Peer-to-Peer VoD Service, Peer-to-Peer Networking and Applications, Volume 3, Number 4, Pages 351-362, Springer Verlag, ISSN 1936-6442, December 2010.
- Lei Shi and Alan Davy, Security Considerations for Intrinsic Monitoring within IPv6 Networks, Proceedings of 9TH IEEE International Workshop on IP Operations and Management (IPOM 2009), Venice, Italy, Oct 2009.
- Lei Shi, Jing Fu and Xiaoming Fu, Loop-Free Forwarding Table Updates with Minimal Link Overflow, IEEE International Conference on Communications (ICC), Dresden, Germany, June 2009.
- Lei Shi, Peter Sjödin, A VLAN Ethernet Backplane for Distributed Network Systems, IEEE Workshop on High Performance Switching and Routing (HPSR) 2007, New York, USA, May 2007.
- Lei Shi, The Implementation of Forwarding Elements on IXP Network processor, Technical Report, XR-EE-LCN 2006:006, School of Electrical Engineering, Royal Institute of Technology, Sweden, October 2006.
- Lei Shi and Peter Sjödin, A VLAN Ethernet Backplane for a Distributed Network System, Swedish National Computer Networking Workshop (SNCNW), 2006, Lulea, Sweden, October 2006

Computer Skills

- Strong background knowledge in IP networks;
- Languages: Ada, C, C++, Java, Pascal, Prolog, X86 Assembly language, Matlab;
- Proficient in Intel NP 2800/2400 architecture and microcode programming; Linux kernel programming;
- Development experience on Unix/Linux, VxWorks and Windows;
- Databases: Oracle and SQL Server;
- Traffic Generator & Analyzer: Smartbits, Spirent TestCenter, Agilent N2X.

Driver License

- Professional Irish 'B' driver license.

Language

- Chinese (Native), English and basic German

Honors and Awards	Honorable Mention, MCM (Mathematical Contest in Modeling), USA, 2001 Honors Graduate of Chongqing University, 2001 Honors Thesis of Chongqing University, 2001