A Thesis for the Degree of Ph.D. in Engineering

Implementation and Evaluation of Secured Network Infrastructure Using Content-based Router

February 2019

Graduate School of Science and Technology
Keio University

Rajitha L. TENNEKOON

Abstract

The Internet is the world's largest public network accessed by approximately half of the world population. In principle, the Internet is used as the main communication medium, which aims to transfer all kinds of user and application data. Most of the time, users need to share their sensitive and private information along the communication with the authorized communication parties. As a public network, the Internet is constantly vulnerable by the miscellaneous threats and attacks such as malware, phishing, spoofing, injections, Denial-of-Service, ransomware, and hacking, which cause the exposure of the users' private and sensitive information. These vulnerabilities have raised the demand for enhancing the functions of the Internet infrastructure to preserve confidentiality, integrity, and authentication (CIA) of their data.

The Internet use "TCP/IP" as its core communication protocol stack. The initial design of the Internet was intended to share data among networks with limited functionality. Such limitations of the core layer led to the inability to address the emerging security threats on its own. Such problems of the Internet ended up in allowing intruders to read and alter data streams. Moreover, it exposed the metadata used for the communication: i.e., packet header information. Therefore, over the past few decades, numerous studies have been conducted to study secure data transmission over public networks such as end-to-end data encryption (E2EE) protocols and tunneling protocols. Generally, a packet traverse through numerous networks between different countries among its delivery. It is a well-known fact that the security policies and requirements diverge among countries and organizations. However, the conventional methods secure data from one end to another without considering the connections in-between. Therefore, the used encryption algorithms or the keyspaces can be vulnerable among some of the intermediary links. On the contrary, tunneling encryption protocols are entirely independent of the routing flow; when data is encrypted or obscured by tunneling, it impedes analysis of traffic streams, which is the vital feature for the service-based future Internet. Moreover, it is vital to securely locate and track the adversary as early as possible after or during a network attack. However, lack of proper faster and secure traceability services is a major issue for the internet infrastructure and its users.

To this end, the dissertation proposes an approach to enhance the security over the public network communication using content-based router infrastructure. Content-based router is a next-generation novel backbone router, which can be used to analyze all packet stream transactions on its interfaces and provide extended services to end users and applications. On the contrary, the conventional routers cannot provide such services. Content-based routers use its specialized hardware and software modules to accelerate the packet processing and its services. The dissertation proposes an infrastructure to provide secured services using a novel distributed link-state routing (DLSR) protocol empowered with per-hop data encryption using the content-based router architecture. The implemented per-hop data encryption protocol is then used to provide mainly three services as the solutions for the above mention security issues, namely, 1) IP-routable entire packet encryption service, 2) neighbor data retransmission service and 3) fast and secure packet traceability service. Moreover, a core system for real-world content-based

routers, named as deep packet inspector on a router (DooR), is implemented and tested which can perform on-the-fly TCP stream reconstruction and analysis leveraging the real-time deep packet inspection capabilities of the content-based routers.

Accordingly, the structure of the dissertation is divided into three main sections: 1) implement and evaluate per-hop data encryption protocol, 2) use the developed encryption protocol to provide secured services for public networks using content-based routers 3) leverages the content-based router infrastructure with hardware and software acceleration of DooR. The first two sections are implemented and evaluated under simulation-based environment using network simulator3 (ns-3). The DooR is implemented on real-world Linux-based high-performance hardware and tested in real-world networks. In conclusion, the proposed secured services yield better performance compared to the conventional methods together with better, flexible security to its users and the real-world DooR implementation guarantees the practicality of the proposed services via its hardware and software accelerations minimizing the packet processing delays in the content-based routers.

Acknowledgments

At various stages throughout the Ph.D. process, I had interesting discussions, support, and valuable feedback from my supervisor Prof. Hiroaki Nishi. I would like to express my sincere gratitude to him for the valuable guidance and support throughout these years.

I would like to give my heartfelt gratitude to Mrs. Yuko Nishi for the endless support during this study. Moreover, I would like to acknowledge all West Laboratory members, especially Janaka Wijekoon, Shanaka Prageeth, Yuchi Nakamura, Erwin Harahap, Kenichi Takagiwa, Shinichi Ishida, Fumito Yamaguchi, and Tomoya Imanishi for great support and help during the time I spent in West Laboratory. I would like to acknowledge my friends Kasun Prasanga and Kazuki Tanida for the great support.

A very special thanks go to my dearest family, my dearest brother and my lovely wife Shanika Ekanayake, the courage and strength you guys gave me makes the man who I am today.

Finally, I acknowledge the following people and institutions for the precious contribution to the success of this thesis work.

- Prof. Hiroshi Shigeno, Associate Prof. Takahiro Yakoh, and Associate Prof. Hiroki Matsutani for the valuable comments, guidelines, and, above all, for the advice to make the dissertation a success.
- MEXT (Ministry of Education, Culture, Sports, Science, and Technology), Otsuki Scholarship, and JASSO Scholarship for financial support throughout the period at Grad. School of Science and Technology, Keio Univ., Japan.
- KLL research grant and Keio University Doctorate Student Grant-in-Aid program for the support during the thesis study.
- RocketFuel, and NII for research support in network typologies.

Rajitha L. TENNEKOON

February, 2019

Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 The motivation

Humans evolved as the most intelligent creatures of the planet mainly due to the human brain and the communications between the human species. Humans invented machines to ease their work and increase efficiency and productivity. Charles Babbage proposed the first general mechanical computer called Analytical Engine in 1832 [1]. Working on the computers, John von Neumann who was a physicist and mathematician identified that the data and instructions could be conflicted when is in the same memory. In 1945, he introduced the stored program concept, which serves as the basis for almost all modern computers. The computers revolutionized the human race by boosting the processing, storage, and efficiency of the human work making them go beyond their thinking limits allowing them to make impossible a possibility.

Communication plays a big part as the humans share their ideas and can work together to achieve the same goal by communicating with each other. At the start, people used sign language to communicate and express their ideas. Then they invented languages where they can express their ideas and feelings both verbally and written manner. With the invention of the computer, they implement computer languages both to interact and communicate with the computers. After that, they want to enhance communication by connecting with each other over distances. This resulted in the invention of wired and wireless transmission media.

In August 1962, Licklider J. [2] discussed his concept of "Galactic Network" in series of written memos. He projected a global network of interconnected computers, which can be accessed by anyone from anywhere. As the head of the computer research, he spearheaded his successors (Ivan Sutherland, Bob Taylor, and Lawrence G. Roberts) in DARPA (started October 1962) developing his concept of networking. In July 1961, Leonard Kleinrock published the first paper on packet switching theory [3] followed by the first book [4] on this area in 1964. Afterward, he convinced Roberts on using packets replacing the circuits. This game-changing step opened the gates to the world of computer networking towards using packets to exchange information over the computer networks as the very first foundation of the field of computer networking.

In 1966, Roberts developed his computer networking concept together with DARPA and published his plan for the "ARPANET" in 1967. Then followed by the RFQ publication in August 1968, regarding the very first packet switching device named as the interface message processors (IMP's) [5]. Afterward, ARPANET was connected to Stanford Research Institute (SRI), and the very first host-to-host message was sent between these two networks. As denoted in Figure 1.1 [6], computers were continuously added to ARPANET, and a mandatory requirement was forming to have a proper host-to-host communication protocol. As a result, Network Working Group (NWG) proposed the Network Control Protocol (NCP) in December 1970 [7]. In 1972, the initial

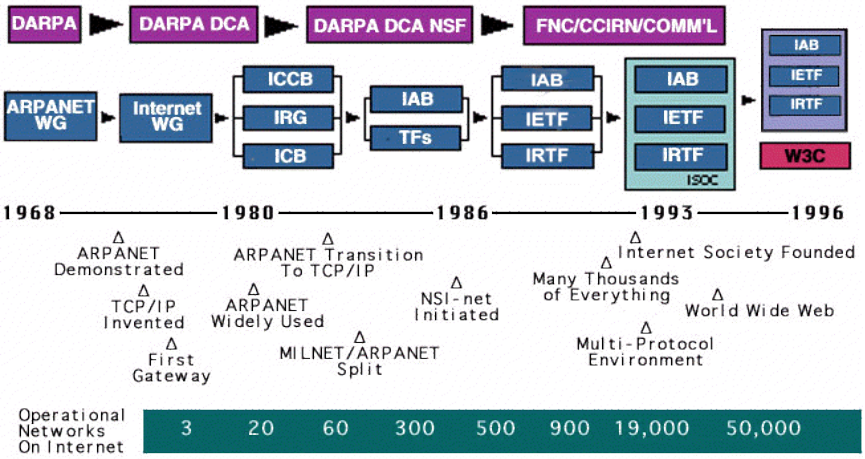


Figure 1-1 Operational network growth of the earlier Internet

application using the new infrastructure called, electronic mail was introduced [8]. In July 1972, Roberts developed the email utility program which was the starting of the new era of email communication [9]. Finally, the revolutionary idea of the "Galactic Network" was practically implemented as ARPANET and this was eventually transformed into the world's largest network, the Internet.

Due to the tremendous growth of the Internet, NCP was not sufficient to co-op with the new applications and services due to its limitations such as no ability to address machines or networks, lack of end-to-end reliability and no error control. This lead to the implementation of the well-known protocol suite, TCP/IP [10] [11] [12]. On January 1983, ARPANET was migrated from NCP to TCP/IP which is the underlying protocol stack used by the current Internet infrastructure. From then onwards, TCP/IP became the main protocol which ARPANET runs on. Afterward, OSI reference model was introduced [13] in 1989 as a framework for developing protocol standards.

Eventually, ARPANET grew to a public network interconnecting all the continents and was renamed as, the Internet. In a very short period of time, the Internet became the world largest network interconnecting users all over the world. The rapid growth of the Internet usage

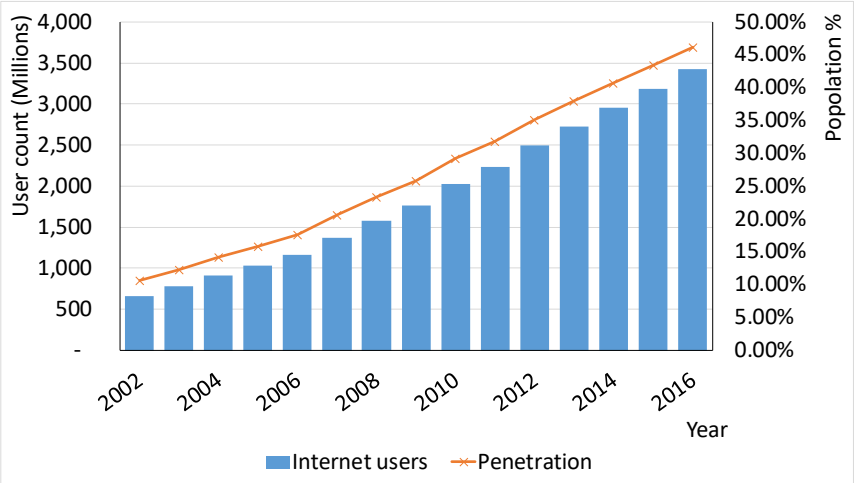


Figure 1-2 Growth of the Internet users within last two decades

motivates the developers to develop user applications satisfying the emerging requirements by the users to handle numerous types of data. Figure 1-2 [14], shows the exponential growth of the Internet users over the last two decades. It clearly illustrates how the Internet has become one of the most valuable resources used by the people all around the world.

As the enormous growth of all kinds of "people-to-people" traffic over the Internet, so does the threats both from the Internet and to the Internet started to grow as well. The main objective of the ARPANET inventors is to make the Internet public as much as possible where the core routing infrastructure is hidden from the end users. Due to this fact, even the attackers and hackers who intend to access unsecured private data passing through the Internet was able to gain access to such sensitive data.

The core protocol stack: "TCP/IP," which is utilized as the principal communication stack by the Internet, was not designed to handle security-related issues because limited functionality was expected from the Internet [15] [16]. On the other hand, the Internet is used by a majority of the worldwide population as their main communication media. It is a well-known fact that the Internet comprises of millions of public networks. In fact, public networks are less secure than private networks. Yet, it is used to transmit sensitive data belonging to millions of users across the world. However, the Internet's inherent problem of inadequate data security permits intruders to read and alter data streams, and most importantly, the metadata used for the communication, i.e., packet header information [17]. Secure data transmission should include confidentiality, integrity, and authentication (CIA) of the transmitted data [18]. According to the Internet security threat report by Symantec [19], there is a 600% increase in attacks against Internet of Things (IoT) devices during the one year from 2016 to 2017. Moreover, they claim that there is a 13% increase in reported vulnerabilities and 1 in 13 web requests lead to malware which is up 3% since 2016. Therefore, over the past few decades, numerous studies have been conducted to study secure data transmission over public networks [20] [21] [22] [23].

Because sensitive data traverses through the Internet, people are demanding data security and privacy for their sensitive data. To this end, various security countermeasures have been invented and implemented in the current Internet infrastructure. A few major and well-known security protocols such as MACsec (IEEE 802.1AE) [24], Secure Sockets Layer (SSL) [25], Transport Layer Security (TLS) [26], Point-to-Point Tunneling Protocol (PPTP) [27], and Internet Protocol Security (IPSec) [28] have been implemented to provide data security. These methods can only provide end-to-end data security independent of the routing protocol and the security requirements of the intermediate networks. Public networks over different countries enforce different security policies which led to use different crypto algorithms over the networks. However, it is impossible to change such parameters dynamically between the data transmission paths when using the conventional encryption methods. Moreover, if the end users use less secure end-to-end encryption algorithms, the data will be vulnerable while traversing through the less secure data networks. Because the Internet comprises millions of networks that are connected by routers, most of the data travel through these network routers. We argue that the network router is one of the vital devices that can preserve data security and privacy during data transmission through the Internet. However, regular routers are incapable of providing adequate data security.

To this end, the content-based router, which is a new generation backbone router, was proposed by the research team at Nishi Laboratory in Keio University [29] [30]. The content-based router has a high-throughput database. It stores squeezed data obtained from all transactions on its interfaces using a regular expression-based string matching filter. Also, it can provide APIs for accessing stored contents to enrich services. Takagiwa et al. showed that it is capable of handling 2 Gbps throughput [31], and that its hardware acceleration is capable of extending the processing throughput five times faster than the software-based acceleration [32].

However, to provide adequate security services to end users and applications, content-based router requires a proper infrastructure which can leverage the functionalities to provide high-end security. The infrastructure must contain an underlying security protocol which can provide confidentiality, data integrity, authentication and non-repudiation to the traversing data. Nevertheless, the protocol must be able to allow content-based routers to analyze packet contents in order to provide content-based services while providing adequate security to the transmitting data. Moreover, it must be flexible and scalable enough to provide extend services required by the end-users to satisfy their needs. Finally, to achieve such tasks practically, both flexible and powerful content-based router is mandatory with sufficient hardware and software acceleration capabilities.

As illustrated in Figure 1-3, the main objective of this dissertation is to implement and evaluate a novel secured network infrastructure using a content-based router. As to answer the problems mentioned above, this study presents mainly four security services.

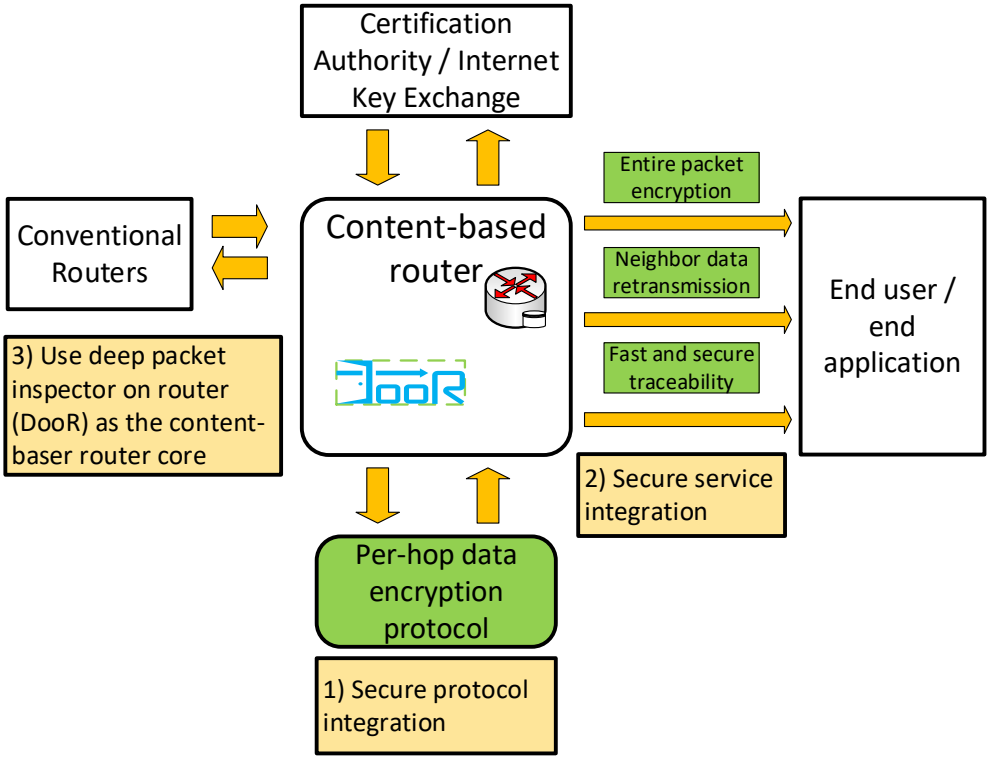


Figure 1-3 Secured infrastructure using content-based router

1. Per-hop data encryption
2. IP routable entire packet encryption
3. Neighbor data retransmission
4. Fast and secure packet traceability

Finally, a prototype of an on-the-fly packet analysis software for used as the core of the real-world hardware content-based routers is proposed, implemented and evaluated (in Chapter 5).

Figure 1-4 shows the places addressed by this dissertation in the domain secured network infrastructures. As highlighted in orange, the primary purpose of the dissertation is to enhance both the security and the secured services of the current infrastructure using the content-based routers. The proposed routing protocol is extended to provide these four services. Moreover, the proposed packet analysis system is used as the content-based router core to enhance the proposed secured architecture using both hardware and software acceleration of the content-based routers.

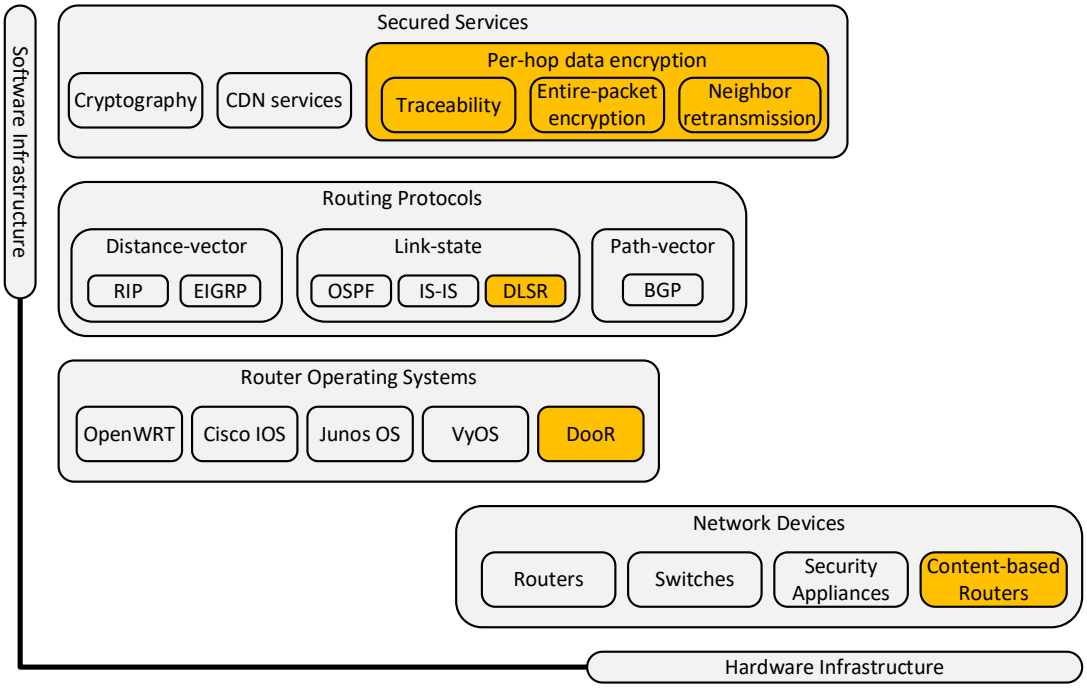


Figure 1-4 The place the dissertation resides within the current context

1.2 Research goals

The following are the main research goals of this study:

1) To design proper flexible and efficient security protocol as the infrastructure foundation.

2) To develop secured services to address entire packet encryption, recovering corrupted packets, and efficient traceability service.

3) To implement a practical core software to operate as the core software for content-based routers which can provide above services to the real-world using the secured infrastructure.

1.3 Objectives

In order to reach the above goals, the study proposed, implemented and evaluated the following six portions which integrate into the proposed secured infrastructure.

1) Link-state routing protocol

2) Per-hop data encryption protocol

3) IP routable entire packet encryption service

4) Neighbor data retransmission service

5) Fast and secure packet traceability service

6) On-the-fly deep packet inspection solution as the content-based router core software

1.4 Contributions

This work makes the following novel contributions:

- It extends the ns-3 network simulator by introducing a distributed link-state routing protocol. This protocol can be used and extended by the research community according to their requirements.
- It describes an implementation of a per-hop data encryption protocol which can be used as the foundation of the proposed secured infrastructure. Furthermore, three more services are implemented and evaluated using the per-hop encryption protocol as extended services. These implementations can be used by the ns-3 research community to understand how to merge a service with a routing protocol to provide extended services to users inside ns-3 simulator.
- It describes an implementation of a real-world deep packet inspection software for

content-based routers which acts as the core software for content-based routers. It provides a flexible, scalable and efficient core software solution for content-based routers using conventional PC hardware. Moreover, it introduces a novel on-the-fly TCP stream reconstruction methodology without the requirement of storing any packets.

- It describes an evaluation of using Intel quick assist adaptors to offload the crypto workloads. This shows the feasibility of using the hardware-based accelerators to offload the process intensive crypto operations achieving significant increase in performance and efficiency of standard platform solutions.

1.5 Dissertation structure

The structure of the dissertation is illustrated in Figure 1-5. Table 1-1 denotes a brief description of each chapter. As denoted in Figure 1-5, Chapter 2 explains the background studies associated with this dissertation. It provides a detailed explanation on the routers, routing protocols, cryptographic protocols, and the content-based router. Moreover, the chapter briefly explains the vulnerabilities of the conventional protocols and the conventional routers and how the proposed methods will resolve those issues using the content-based routers. In the end, the chapter discusses the related works.

Chapter 3 introduces the distributed link-state routing (DLSR) protocol and per-hop data encryption service implementations together with the evaluations. How the above two implementations were merged into a per-hop data encryption protocol together with its advantages are explained in the starting of Chapter 4. Then it leads to introduce three novel security services based on the per-hop data encryption protocol: namely IP routable entire packet encryption service, neighbor data retransmission service, and fast and secure packet traceability service. This chapter includes a complete design and implementation details of each proposed service under simulation environment together with the extensive evaluations and the corresponding test results.

Chapter 5 allows the proposed services in Chapters 3, and 4 to incorporate with the real-world via introducing the on-the-fly stream analysis system: namely deep packet inspector on a router (DooR). Chapter 5 deeply explains the design and implementation details of DooR together with the evaluations deploying the implementation under real-world networks. At the end of this chapter, the Intel QAT adaptors are evaluated on offloading CPU intensive crypto workloads.

Finally, Chapter 6 concludes the dissertation followed by future works.

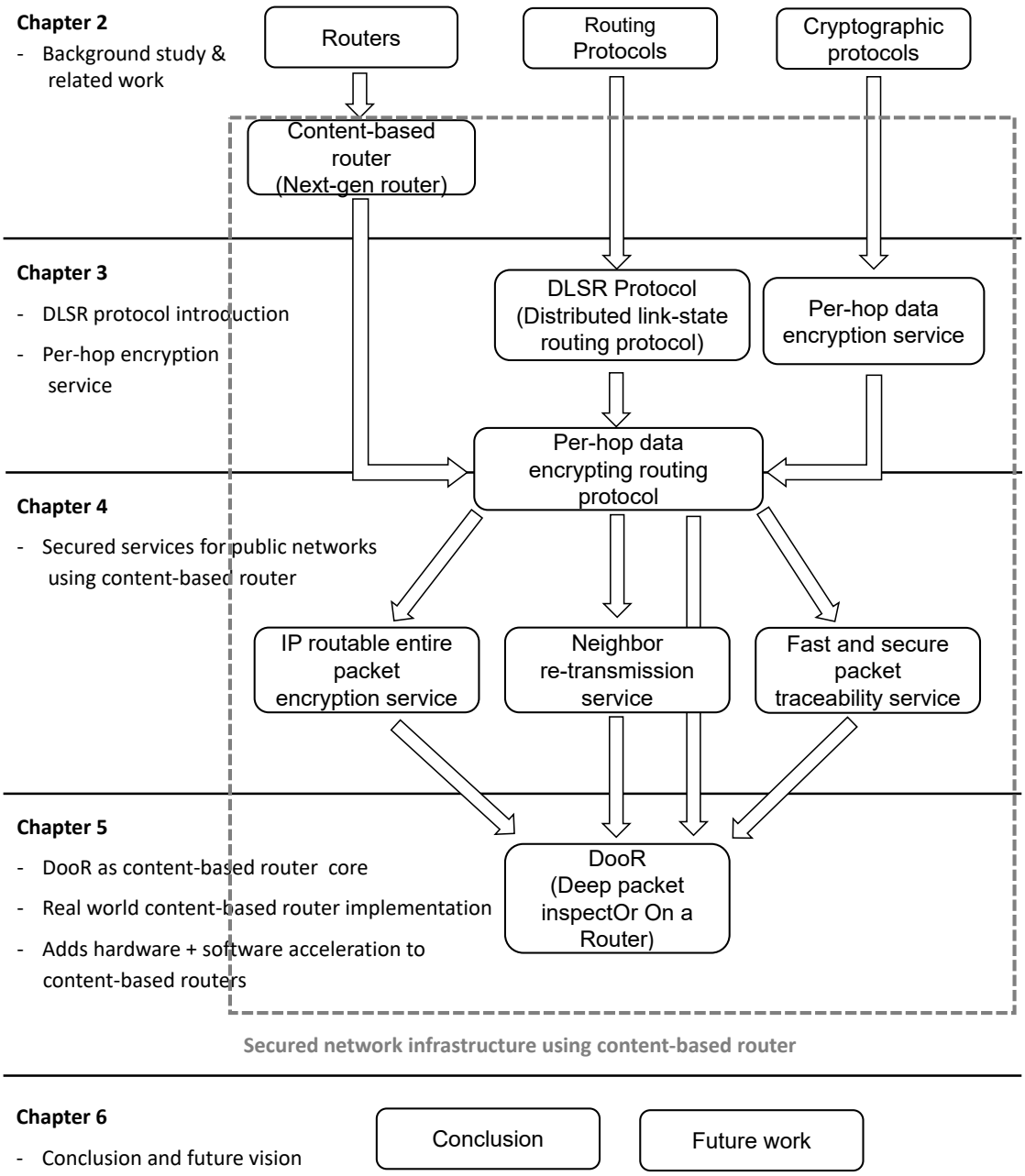


Figure 1-5 Dissertation structure

Table 1-1 Chapter description

Chapter 2	Purpose	Survey the background information about routers, content-based router, routing protocols, and cryptographic protocols followed by the background study.
Chapter 3	Purpose	Implementation, simulation and evaluation of per-hop data encryption protocol.
	Objectives	1) Find a simulator to implement the proposed architecture. 2) Design and implement a distributed link-state routing protocol 3) Design and implement per-hop data encryption service.
	Proposed Methods	Use the well-known Network Simulator 3(ns-3) to implement the proposed routing protocol and the per-hop data encryption service.
	Achievement	1) Designed and implemented a novel distributed link-state routing (DLSR) protocol on ns-3. 2) Successfully implemented the per-hop data encryption service. 3) Tested and evaluated the implementations for functionality and effectiveness. The implementations are openly published in [128].
Chapter 4	Purpose	Implementation, simulation and evaluation of Secured services for public networks.
	Objectives	Merge the DLSR with the per-hop data encryption service into per-hop data encryption protocol. Use the per-hop data encryption protocol to provide the following 3 services. 1) IP routable entire packet encryption service 2) Neighbor data retransmission service 3) Fast and secure packet traceability service
	Proposed Methods	Enhance the implemented per-hop data encryption protocol to provide extended security services.
	Achievement	1) Successfully designed, implemented and integrated the required three secured services to the per-hop data encryption protocol. 2) Tested and evaluated the implementations for functionality and effectiveness. The implementations are openly published in [78] [86] [80].

Chapter 5	Purpose	Real-world implementation and evaluation of the deep packet inspector (DooR) as the content-based router core which will leverage the above simulated services to the real-world implementations.
	Objectives	1) Use hardware and software acceleration to implement efficient, robust and flexible deep packet inspection system (named as DooR). 2) Provide on-the-fly TCP stream reconstruction service. 3) Allow third party users to deploy custom services using DooR. 4) Evaluate Intel QAT hardware accelerator for offloading crypto workloads.
	Proposed Methods	Implement DooR with both hardware and software acceleration to perform on the fly DPI and TCP stream reconstruction. 1) Use Intel DPDK library's software acceleration for achieve high throughput packet analysis. 2) Use Intel HyperScan to achieve efficient high speed string matching to extract features from the reconstructed TCP streams. Evaluate the Intel QAT accelerator to offload the encryption, decryption, sign and verification security workloads which can be used with the proposed methods.
	Achievement	1) Successfully designed, implemented, and evaluated the DooR system. 2) Successfully deployed DooR system in SINET4 and Interop events (2004/2005) to analyze real-time network data. 3) Successfully allowed third party users to connect to DooR system and deploy their services to gain real-time stream analysis information to provide third party services to end users (Deployed and tested in the Interop events). 4) Successfully evaluated the QAT adaptors for symmetric/asymmetric encryption and hash algorithms.
Chapter 6	Purpose	Conclusion and future work.
	Achievements	States the final conclusion on the implemented secured infrastructure followed by the future works.

Chapter 2 Background study and related work

2.1 Evolution of the Internet

Among the evolution of the Internet, routers emerged to provide services beyond the network layer in order to address complex issues such as security, data analysis purposes, provide content-based routing, etc. However, such processes require to perform packer analysis which urge excess resources such as processing power, storage, and memory. This led the router manufactures to develop and integrate specialized modules like multicore packet processors, high-speed smart network adaptors, high-performance cache modules, and storage modules. Nowadays, the main router manufacturers like Cisco, Alaxala, and Juniper, also allow the connectivity from the third-party applications to perform packet manipulation through provided APIs [33] [34]. Moreover, Cisco now provides extended services to users from their edge routers, such as firewalls [35], session border controlling [36], WAN acceleration [37], and load balancing [38]. This idea is followed by the vendors Motorola, Hitachi, and Juniper proposing to place their routers to the IPS network edges in order to provide services to end users [39].

Cisco is proposing to leverage their edge router architecture with fog computing [40] allowing packet processing on Internet of Everything (IoE) [41]. Furthermore, Hitachi proposed the concept of WAN accelerator [42] which used to accelerate the TCP connections via intercepting them at the network edges. Moreover, the well-known mobile platform developer, Qualcomm, has proposed a content caching router architecture which manages smart house devices [39]. They also introduced a router which performs content prepositioning for bandwidth-heavy content using Akamai software in consumer electronics show (CES-2014) [43].

Recently, the software-based router architectures have been rapidly developed as well. The well-known Vyatta which is a software-based virtual router has been changed to VyOS [44]. It is one of the major virtual routers used by the developers and the research community to implement services over the routers. Intel also started to leverage their software packet processing platform via introducing Intel data plane development kit (DPDK) [45] software framework. This framework can work with the Intel-based hardware such as Intel processors, Intel network interface cards [46], and Intel quick assist technology (QAT) [47] modules to directly use their hardware acceleration for the packet processing purposes. Furthermore, they released their powerful high-performance regular expression matching library, Intel HyperScan [48], making it open source. However, there is no complete solution which provides a flexible solution for integrating these technologies. Chapter 5 of this dissertation introduces the deep packet inspector on a router (DooR) system which takes advantage of this Intel architecture in leveraging the content-based router infrastructure to provide flexible user and application services using hardware and software acceleration.

2.2 Content-based Router

Content-based router is a next-generation router which can provide services such as deep packet inspection, content handling, regex-based string matching, and high throughput databases. Conventional routers are unable to provide extended services to Internet users and applications as they do not access higher layers such as session and presentation. To address this need, Hiroaki Nishi Laboratory, Keio University has proposed a content-based router, named as a service-oriented router (SoR), to provide content-based services to end users and applications (Figure 2-1). In mid-2008, the content-based router was proposed as a semantic router to enrich user services via data stream analysis [29] [49]. Using the stored information, the content-based router leverage the existing IP-based Internet infrastructure to a service-based open innovation platform. As illustrated in Figure 2-1, our laboratory has implemented real-world content-based routers both using conventional server PCs (Figure 2-1i) and conventional micro PCs (Figure 2-1ii). However, a proper on-the-fly stream analysis software is required for the proposed content-based router which can use as the core system to provide content-based services to the end users. Chapter 5 of this study proposes and evaluates the DooR system as a flexible solution.

2.3 Routing protocols

Routing protocols use routing algorithms to decide the optimal communication paths between network nodes. Routing protocols use different metric values, such as hop count, bandwidth, delay, path length, reliability, communication cost, and maximum transmission unit (MTU) to define these distances or the costs between the nodes. EIGRP [50] and OSPF [51] are the two

(I) Conventional server-based (II) Micro PC-based (Intel NUC)

Figure 2-1 Real-world content-based router implementations

major distance vector and link-state routing protocols used in the networks [52].

The proposed packet encryption services are developed in ns-3 [53] simulator and those services need to access the routing layer in order to secure the IP headers. However, the ns-3 simulator lacks in proper wired link-state routing protocols. Hence Section 3.1 proposes, implements and evaluates the DLSR routing protocol in ns-3. The protocol will be further extended merging with the per-hop data encryption protocol and then used for providing flexible extended security services in Chapter 4.

2.4 Cryptographic protocols

Cryptography is used by the early humans, ex. Egyptians some 4000 years ago [54], through the twentieth century where it played a crucial role in the outcomes of both world wars and towards the today's communication [55]. Data encryption standard (DES) [56], Diffie-Hellman (DH) [57] key exchange algorithm, and Rivest, Shamir, and Adleman (RSA) public-key scheme [58] are the well-known landmarks on the cryptographic inventions for the computer domain. Essentially, these algorithms preserve 1) confidentiality or privacy, 2) data integrity, 3) authentication and 4) non-repudiation, of the data.

The Internet is made out of unsecured public networks. However, the Internet is the main communication medium used by almost half of the entire world population to communicate with each other sharing their sensitive information. This made possible, with the help of the cryptographic protocols. A cryptographic protocol is a series of steps and message exchanges between multiple entities in order to achieve a specific security objective. Cryptographic protocols use cryptographic primitives such as encryption, signature functions, and hash functions to secure the communications. Some of the well-known cryptographic protocols are secure shell (SSH) [59], IPsec [60], Kerberos [61], internet key exchange (IKE) [28], transport layer security (TLS) [62], signal protocol [63], and point to point protocol (PPP) [64]. These are commonly used to implement cryptographic schemes which mitigate passive and active attacks.

Intelligent devices, i.e., content-based routers, can provide extended services such as content-based routing and deep packet inspection via analyzing the packet contents. However, when the packets are encrypted using the above methods, it impedes analysis. Moreover, conventional protocols are not flexible enough to cooperate with such devices to provide extended services. Therefore, a secure and flexible cryptographic protocol is required for a content-based router which is flexible enough to provide extended secured services. Due to the fact that routing is a per-hop based operation, and if a cryptographic protocol can act in per-hop based manner, it can be merged with the routing protocol to provide efficient and flexible cryptographic solutions with the aid of the routers. To this end, Chapter 3 of this study proposes the per-hop data encryption protocol which can use to enhanced security infrastructures via integrating further security services such as, entire packet encryption, neighbor data retransmission, and secure traceability.

2.5 Related work

2.5.1 Distributed link-state routing protocol (DLSR)

OSPF [51] and Enhanced Interior Gateway Routing Protocol (EIGRP) [50] are two of the widely used routing protocols throughout the Internet. OSPF is a link-state routing protocol, and EIGRP is a hybrid routing protocol. Both of these protocols are Cisco proprietary protocols. OSPF mainly considers the link throughput while calculating the routing metric. EIGRP uses five k values (k1-k5) to influence routes where k1 is bandwidth, k2 is load, k3 is a delay, k4 is reliability and k5 is MTU. According to [65], these values should be carefully selected via the network administrators. Otherwise, the EIGRP prevents building neighbor relationships, which cause the network convergence failures. Therefore, as explained in the EIGRP route calculation equation in [65], in default behavior, most of these k values (k2, k4, and k5) are set to zero. Nonetheless, Cisco routers use the equation and perform routing throughout the Internet in a proprietary manner calculating load (k3) and the reliability (k4) values.

Nowadays, the main concern for all the Internet-based services are network delays. The current distance vector routing protocols such as Routing Information Protocol (RIP) [66], RIPv2 [67] or the link-state routing protocols such as OSPF [51] does not consider the link propagation delays while calculating the route metrics. Therefore, I propose a distributed link-state routing protocol (DLSR) in a way which it also considers the link propagation delays while calculating the routing metrics (further explained in Section 3.4). This enables DLSR to select the least cost paths which are calculated using both the least distance together with the least delay to a particular destination. Therefore, via considering the link propagation delays, DLSR can provide a much accurate routing metric over the RIP or OSPF routing protocols which the time critical services can take the advantage by obtaining the least delay paths to destinations. Moreover, the DLSR route calculation process is distributed among all the routers where each router calculates the total network routes themselves allowing more flexibility and mitigating the single point of failure scenarios.

Larger LANs can have many routing devices. Therefore, those devices can generate heavy as control-packet traffic flooded across the network, i.e., link-state advertisements (LSAs).To improve this potential traffic problem, OSPF [51] uses designated routers (DR) on all multi-access networks (broadcast and non-broadcast multi-access (NBMA) networks types). Likewise, OSPF routing protocol processes and distributes the link-state updates by an elected centralized DR, and a backup DR incase of failover. The DR distributes the relevant routing updates to all the other nodes in a topology. Changing the DR is a disruptive process where DR keeps track of all the acknowledged router details and the link-state information. Therefore, in case the DRs went down, it would take a lot of time together with protocol messages to take over the DR roles. Such time-consuming complex election process can be mitigated using the distributed route calculation of the proposed DLSR protocol.

In particular, ns-2 and ns-3 are widely used for the experiments and researches in the Mobile Ad hoc Networks (MANETs). All of the existing routing protocols implemented in ns-3 such as;

AODV [68], DSDV [69], DSR [70] and OLSR [71] are all optimized and operates on wireless networks. Although ns-3 provides rich wireless network testing environments, it only provides three routing modules for wired network topologies named as default/static routing module, the global routing module, and a RIPng [72] routing module. First two of these modules are static routing modules where the details of the routing tables are gathered and stored only at the start of the simulation. Therefore, these protocols cannot adapt to link changes after the initialization. On the other hand, RIPng is the only available dynamic distance vector routing protocol in ns-3. Despite these facts, ns-3 lacks dynamic-wired routing protocols. Specifically there are no proper link-state routing protocols available in ns-3.

The main objective of the DLSR is to simulate a dynamic link-state routing protocol for the content-based routers using ns-3 simulator. While the ns-3 users can use DLSR as the routing protocol for their wired network simulations under ns-3, they can also develop applications/services for content-based routers, integrate them to DLSR, and test them under ns-3. Moreover, the DLSR can be used as the foundation of the link-state routing protocol development under ns-3, and both the users and the developers can develop novel routing protocols using DLSR as the baseline or extend the DLSR according to their needs. Chapter 3 deeply explains and discusses the implementation of DLSR protocol.

2.5.2 Per-hop data encryption

Nowadays, the Internet has become the primary medium for communication. Therefore, people send their sensitive data through public networks over the Internet. Owing to the lack of security in these networks, users and applications transmit their sensitive data through secured channels. MACsec (IEEE 802.1AE) [24], Secure Sockets Layer (SSL) [25], Transport Layer Security (TLS) [62], the Point-to-Point Tunneling Protocol (PPTP) [27] and Internet Protocol security (IPsec) [60] are the main privacy-preserving protocols used by the users and applications. Whenever data is encrypted or obscured by tunneling, it impedes analysis. This is applicable to SSL [25], IPsec [60], multicast, and various forms of non-secure tunneling [73]. In each of these methods, the keyspace remains constant throughout communication because it cannot be changed once it is set by the original sender.

To this end, this study proposes per-hop data encryption service which encrypts data dynamically in a per-hop manner. One of the main advantages of the proposed method is that the users can use dynamic key lengths together with different encryption algorithms among the hops, as per their preference. This allows a packet to be encrypted using different keyspaces and different encryption algorithms among the hops. The keys and their properties are managed by the content-based routers and clients; otherwise, the applications can request their requirements regarding the key constraints from the content-based routers which are tailored to their needs. This facilitates encryption using the dynamic keyspaces as required.

Moreover, because the content-based routers manage the keys, they act as an intermediate authorized router or gateway. Content-based routers can help in analyzing the packet contents, identifying and mitigating, or preventing threats such as intrusions and virus attacks; it can stop

the breach earlier than the existing detection or prevention methods. Furthermore, securing the IoT nodes where it is difficult to install antivirus software in small ubiquitous devices that have limited processing performance, memory, and battery.

2.5.3 IP-routable entire packet encryption

Onion routing or The onion router (Tor) was introduced where servers that are voluntarily operated to direct Internet traffic through a random sequence of nodes to help conceal users' communication with other services on the Internet [74] [75]. Nowadays, Tor plays a major role in network security by anonymizing users and concealing their identities.

Michael G. et al. explained anonymous connections and onion routing describing few vulnerabilities of onion routing [76]. According to this study, onion routing provides anonymous connections but not anonymous communication. However, the proposed method provides both via encrypting identification information from the data stream. Moreover, each onion layer comprises dedicated header information, thereby increasing the packet size, whereas the proposed method uses only one IP header irrespective of the number of content-based routers which the packet traverses through. Additionally, according to this study, onion routing's overhead is mainly due to public key cryptography and is incurred while setting up an anonymous connection. According to [76], onion routing is vulnerable to traffic analysis attacks. The proposed method can encrypt and hide the original packet stream information (such as port numbers and IP address details), thereby mitigating identifying patterns in the data streams using shared key cryptography.

As per the security according to [77], Tor encrypts the connection but not the data. It also mentions that the main drawback of Tor is the exit node's vulnerability. Therefore, it is highly advisable not to transmit unencrypted data over the Tor network because the information may be accessible to an intruder while the data is on the last node. However, in the proposed method, the source encrypts the entire packet and then transmits it to the content-based router. Likewise, the last content-based router transmits an encrypted packet to the destination node as well (encrypted using the key pair in-between the destination and next hop content-based router). Similarly, the exit node vulnerability can be totally mitigated by the proposed method [78].

According to Raheem A. [79], solutions similar to onion routing provide a high degree of route un-traceability; however, their process introduces considerable delays and inefficiencies in terms of bandwidth and energy utilization. The proposed traceability method does not implement such considerably high delays. Additionally, it can provide fast and secured packet traceability [80], which can be very useful in situations such as tracking the source of an identified network attack.

The plain headers of the protocols such as IP and TCP cause security vulnerabilities such as TCP sequence number prediction, TCP blind spoofing, SYN flooding, session hijacking, cross-site scripting, man-in-the-middle attacks, and DNS protocol attacks. With the diligent application of the right types of algorithms, it is possible for an attacker to guess the sequence of numbers that TCP assigns to a stream of data packets. Knowing the next number in a transmission sequence, an attacker may potentially "step in" an ongoing communication and pose

as the originator of the message. In TCP blind spoofing, an attacker can guess both the sequence number and port number of an ongoing communication session. They are then in a position to execute an injection attack by inserting corrupted or fraudulent data, or malicious code or malware into the stream.

Under the protocol rules, a client or server receiving these requests is required to respond to them to keep the communication alive. This requirement is the basis of a SYN flooding attack, wherein multiple SYN packets are spoofed using a bogus source address and further sent to a targeted server. These attack types use the TCP/IP header fields to perform the attacks. The proposed entire packet encryption method can mitigate such types of attacks by hiding the sensitive information of the original TCP/IP header.

In session hijacking, an attacker may capture data packets and gain full access to an HTTP session using a packet sniffer, a tool for detecting the presence and movement of data packets. If there is weak authentication between a web server and its clients, the attacker may assume full control of the client's rights, switching the communication to one directly between them and the targeted server. However, unlike conventional methods, because the packets transmitted by the proposed entire packet encryption method are entirely encrypted, the sniffers will not be able to see or access the original sender and receiver information together with the sensitive information of the TCP header to perform session hijacking attacks.

By spoofing an IP address, an attacker may intercept an ongoing transmission and impersonate the entity (person or a bot) involved in the communication, and steer valuable data toward themselves or misinformation and malware toward the recipient. These types of man-in-the-middle attacks are a common threat over public networks such as the Internet. The proposed entire packet encryption method can overcome such an attack by encapsulating sensitive data in the packet headers and trailers together with the packet data. Therefore, although the attackers and sniffers can sniff the data, they cannot analyze or obtain the original/sensitive information regarding the packet streams.

Fundamentally, encryption blinds intrusion detection and prevention systems. However, with the help of the proposed per-hop data encryption, content-based routers can analyze the authorized encrypted packet streams without disabling encryption over the permitted streams. Merging the proposed per-hop data encryption service with the proposed DLSR protocol, content-based routers are able to secure the sensitive data in the packet headers and trailers.

2.5.4 Neighbor data retransmission

TCP is the well-established and widely used reliable data transmission protocol through the Internet. While IP takes care of handling the actual delivery of the data, TCP takes care of keeping track of the individual units of data or packets that a message is divided into for efficient routing through the Internet.

When the TCP transmits a segment containing data, it puts a copy on a retransmission queue and starts the retransmission timer. When the acknowledgment for that data segment is received,

the segment is deleted from the queue. If the acknowledgment is not received before the timer runs out, the segment is retransmitted [81]. This retransmission timer value is defined as retransmission timeout (RTO). Basically, this TCP retransmission method is retransmitting packets all the way from the sender. In order to send the packet again, the retransmission timer should be expired. Therefore, this traditional method should wait for a minimum of the RTO value to retransmit the data. As a result, the receiving end will receive the retransmitted data with a delay of RTO plus the end-to-end link delays and the intermediate processing delays of the intermediate devices. Moreover, in some cases, when a timeout occurs, all the unacknowledged data in the output window is retransmitted. This will result retransmitting already delivered packets where the acknowledgments are on the way as well.

The conventional TCP uses acknowledgments to recover from data that is damaged, lost, duplicated, or delivered out of order by the Internet communication system. TCP attaches sequence numbers to the packets and requiring a positive acknowledgment (ACK) from the receiving TCP. If the ACK is not received within a timeout interval, the data is retransmitted. The receiver will use this sequence number to acknowledge the received packets, to eliminate duplicates and to correctly order segments that may be received out of order. Moreover, TCP uses different types of ACKs depending on the implementation and the situation. In the positive acknowledgment with retransmission (PAR) method [81], the receiver is sending ACKs to the sender regarding the successively received packets. If an ACK for a particular data is not received before a particular time period, the sender will retransmit the packet. In this method, the packets will be retransmitted for lost ACKs or delayed ACKs as well. In a negative acknowledgment method [82], the acknowledgments are sent to the packets which need to be retransmitted. In the selective acknowledgment method, the receiver will notify using both negative and positive acknowledgments appropriately depending on the situation to the sender.

There are also other retransmission methods implemented by researches such as; Wu, H, Li, Z and Tanigawa, Y, where all the methods were implemented for the wireless and mobile networks [83] [84] [85]. Most of the researches are trying to improve the retransmission index of the traditional methods. Moreover, lack of the retransmission methods for wired networks highlights the requirement of novel retransmission method for the wired networks. To this end, this study proposes a novel neighbor retransmission method for public networks using the features of the content-based routers [86].

2.5.5 Fast and secure packet traceability

Numerous approaches to providing packet traceability service using source IP addresses have been implemented. ICMP-based packet traceability methods have likewise been implemented. Bellovin, for example, implemented an ICMP-based trace-back method that can be used in network attacks [87]. In this method, routers are configured with a technology called ITrace. All routers configured with this method are programmed to select a packet from every 20,000 packets and to send an ICMP trace message that is destined to the same host as the selected packet's. However, the trace packets will only be sent from the ITrace service installed routers. To construct the path, the destination can use all packets generated by the routers with the received

time-to-live (TTL) values.

Allison et al. [88] proved that this ICMP-based trace-back method is efficient and reasonably secure. They concluded that, during an attack in which a massive number of packets are generated, such as in a distributed denial-of-service (DDoS) attack, the efficacy of this ICMP-based method is significantly reduced because the probability of the usable ITrace packets is reduced. Therefore, the ITrace method is unusable in DDoS types of attacks, which use a massive number of packets.

Probabilistic packet marking is a common traceability method introduced by Savage et al. in [89]. In Ref. [90], Kihong et al. evaluated the effectiveness of IP trace-back using probabilistic packet marking under a DDoS attack. Song et al. extended this method by integrating an authentication technique. In addition, Adler [91] discussed the tradeoffs of probabilistic packet marking for the IP trace-back method. In the probabilistic packet-marking method, the routers are programmed to randomly mark the packets using a fixed probability. The victim must collect a large number of packets to reconstruct the complete trace because the trace is randomly attached to the packets according to the used probability.

Stone introduced a feasible traceability method that involves attaching a special tracking router (TR) to the networks [92]. The premise of this method is that the TR serves as a type of proxy server that monitors all the packets. In this method, all traffic from the edge routers is routed through the TR using generic route encapsulation tunnels (GREs). The existing routing infrastructure does not need to be changed or updated to use this method; the network links become congested because the GREs are made using the existing networks. In addition, the system proposed in Ref. [92] uses single administrative domains and requires a method to connect all the TRs when communication is needed beyond the Internet service provider area.

The major drawback of the above methods is that they require analysis of large numbers of packets to obtain the traceability data. Moreover, they depend on the probability or a random static numeric value to define the trace; consequently, a higher probability exists that the methods miss the actual trace-required packets to be used in the trace process.

Existing protocols use end-to-end encryption mechanisms, such as SSL [25] and TLS [62], when they must securely send a packet through networks. When using such end-to-end encryption technologies, intermediate devices, such as even routers, cannot tamper or read the data. This integrates higher security while denying data appends to the packet, which is required for providing services such as traceability. Therefore, providing services such as traceability is a complex challenge using existing routing protocols. Moreover, such services are difficult to provide when using end-to-end encryption technologies. Therefore, to address these issues, the proposed secured traceability method employs the implemented per-hop data encryption method. The main drawback in using per-hop data encryption technology is the processing delay in encryption/decryption processes, which occurs in every content-based router through which the packet flows. Nonetheless, this delay can be drastically reduced or minimize into minor tolerable values by using hardware-based content-based routers which has proper hardware and software acceleration.

2.5.6 Deep packet inspector on a router (DooR)

During the past few decades, the rapid growth of the Internet has confirmed that it is the main communication media of the global population [93]. Simultaneously, network traffic has continued to increase [20]. Studies have found that more than 85% of packets traveling on the Internet are based on the transmission control protocol/internet protocol (TCP/IP) [12]. Among the many Internet services, hypertext transfer protocol (HTTP) has become the preferred protocol. Thus, analyzing HTTP traffic has been the focus of many recent studies [23] [94] [95] [96] [97] [98] [99] [100]. Most of these studies have analyzed traffic using captured pcap files. However, on the fly stream analysis methods are necessary to provide internet-based services such as content-based routing, detecting network attacks, and obtaining real-time statistical information. Moreover, because all web-based services use HTTP as their main communication protocol, analyzing web content and providing statistical information with respect to web access can be immensely beneficial to websites and web-based services such as content prediction and content suggestion.

Wireshark [101] and tcpflow [102] are two of the main software programs used for packet analysis. However, both of these programs are used for passive TCP session reconstruction. Nevertheless, researchers and developers have developed their own software to perform active TCP stream reconstruction. These include JUSTNIFFER [103], TCP Session Reconstruction Tool [104], and Scalable and Flexible Traffic Analysis Platform (SF-TAP) [105]. JUSTNIFFER is a packet-sniffing software developed to analyze traffic and extract all intercepted files from HTTP traffic. It can use both live traffic and traffic captured and converted to libpcap format. The TCP Session Reconstruction Tool, which is available in code project, is a packet sniffer developed to reconstruct HTTP streams from packet capture files (pcap files); these pcap files are obtained using programs such as tcpdump or Wireshark. Finally, the SF-TAP [105] traffic analysis platform is a multicore application-level stream analyzer. It is a feasible and scalable multicore application that can be used to analyze streams. However, the main problem of SF-TAP is that it requires multiple PCs that operate at multiple levels of the analysis process. Furthermore, because SF-TAP uses netmap-based packet capturing, SF-TAP cannot attain the high-speed and low-latency packet capturing achieved by libraries such as the Intel data plane development kit (DPDK).

To this end, this study proposes an on-the-fly stream analysis software, named as deep packet inspector on a router (DooR). DooR uses the libraries of Intel DPDK framework to accelerate packet processing workloads. DooR is flexible enough to run in conventional PCs with Intel hardware such as Intel-based CPU and NICs. SF-TAP can only perform stream extraction when DooR can perform TCP stream reconstruction, deep packet inspection (DPI), string matching, and packet routing functionalities. Snort [106] is a well-known open source network intrusion prevention system (NIPS) and network intrusion detection system (NIDS). Snort also performs packet payload analysis for streams such as HTTP. However, Snort software does not utilize DPDK and is very complicated to use as compared to DooR. Moreover, DooR also supports multicore for scalability, and at the moment Snort does not support multicore.

In addition, the aforementioned software does not provide adequate or easy flexibility to users in customizing and modifying the program. Moreover, existing TCP stream reconstruction

software analyzes packets individually or analyzes TCP streams after receiving all packets of a stream. Therefore, these software programs buffer all packets of streams until the analysis starts, resulting in larger memory profiles and offline packet analysis properties. In this sense, DooR provides a very flexible DPI solution using a low-memory profile by storing only the states of the reconstructing streams without storing the packets or the packet contents. Simultaneously, it performs on-the-fly packets analysis. DooR is easily configurable according to the user or application requirements. It can be used to analyze all types of protocols or packets by easily expanding the code based on user requirements. This provides flexibility to analyze any kind of traffic according to user or application requirements. To speed up the data transmission, most of the HTTP traffic is encoded and compressed during the transmission using technologies such as Chunk encoding and Gzip compression. DooR is capable of on-the-fly decoding chunk encoding and decompressing Gzip compressed packet payloads beforehand analyzing the packet payloads. DooR only stores decode and decompression processes states while analyzing packet streams and uses the saved states to analyze the streams continuously. Deep details on implementation and testing of DooR software are presented in Chapter 5.

Chapter 3 Per-hop data encryption service

3.1 Motivation

Nowadays, many researches are conducted based on Internet infrastructure and Internet-based services. Routing protocols play a major role in the Internet handling and providing the end-to-end packet delivery service over the networks. However, the Internet is not a suitable testbed to be used for implementation and test novel routing protocols because it is unrealistic to implement test nodes all over the world and give assumed traffic congestion and communication troubles. Therefore, simulators such as ns-2, ns-3, OPNET, NetSim, are introduced for the researchers and developers. Among these simulators, ns-3 [53], which is the successor of ns-2 [107], is one of the well-known network simulators due to the facts that it is open source.

ns-3 supports most of the existing communication media as its modules, and it supports Linux-based packet structure similar to the Linux network stack. It is fully modularized according to the real world, and though it is a simulator, it provides the architecture and environment that is more realistic than any other network simulator currently available [53]. Therefore, the protocols and services developed by ns-3 can easily migrate and integrate to the real world. Moreover, ns-3 is rapidly developed by many community developers and managed as an open source project in the past several years. ns-3 has eight IPv4 unicast routing protocols and three IPv6 routing protocols. However, the dynamic routing protocols such as, OLSR [71], AODV [68], and DSDV [69] are MANET routing protocols which are implemented for wireless networks. [67] and [72] are the only currently available dynamic distance vector routing protocols in ns-3. Currently there are no link-state routing protocol available in ns-3.

In order to implement a secured infrastructure and its services, it required a routing protocol and a secured packet transmission method as the foundation. Initially, the content-based routers must communicate with each other and should be able to route packets through the networks, using a routing protocol. First half of this chapter propose and implement the distributed link-state routing (DLSR) protocol as a link-state routing protocol for ns-3. The main three motivations of the DLSR protocol are 1) to implement the use link propagation delays together with the link bandwidth in metric calculation, 3) to distribute the routing table generation to all the routers, and 3) to implement a distributed link-state routing protocol in ns-3.

Content-based routers analyze packet content to provide extended services to the end users. When users use the conventional encryption methods such as end-to-end encryption and tunneling, they impedes analysis. Hence, the content-based routers are not able to provide content-based services. However, content-based routers must use cryptographic protocols as the users' demands security to their sensitive data. Public networks over different countries enforce different security policies which led to use different crypto algorithms over the networks.

However, it is impossible to change such parameters dynamically between the data transmission paths when using the conventional encryption methods. Moreover, if the end users use less secure end-to-end encryption algorithms, the data will be vulnerable while traversing through the less secure data networks. To-this-end, the proposed per-hop data encryption protocol can tolerate such enforced security policies via dynamically changing the encryption algorithms and keyspaces between the hops according to the security policies among the links [110].For the proposed secured infrastructure, a proper underlying encryption infrastructure is required to provide all the proposed services. As the conventional encryption methods (As explained in the Chapter 2.6) does not allow the content-based routers the flexibility to provide such extended services, second part of this chapter proposes a per-hop based data encryption service to encrypt data among content-based routers.

Besides, the proposed protocol and all the services are implemented and simulated using the ns-3 simulator. As the simulation testbed, Intel Core i7 3.2GHz CPU, 24 GB RAM, and CentOS 6.5 64-bit operating system is used for all the proposed protocol and service simulations in this chapter. All the cryptographic operations are performed in real-time inside the ns-3 simulator using the Crypto++ (Cryptopp) library [108]. For the explanations, hereafter content-based routers are used as routers, and the general routers are referred to as conventional routers.

3.2 Distributed link-state routing (DLSR) protocol

DLSR protocol is a link-state routing protocol that primarily uses the Dijkstra's algorithm [109] to calculate the least cost paths to the nodes in a network. The well-known open shortest path first (OSPF) [51] routing protocol processes and distributes the link-state updates by an elected centralized node called designated router (DR). The DR distributes the relevant routing updates to all the other nodes in a topology. Unlike the centralized route calculation in OSPF, DLSR distributes the route calculation to all of the nodes. Simply, in DLSR, all the nodes in the topology calculates their routing tables using the received link-state updates. As the Internet is a distributed network, DLSR is proposed as a distributed routing protocol. Moreover, DLSR extends the ordinary Dijkstra's algorithm to use both link throughput and link propagation delay as its metrics while calculating the route metric value for a particular network.

3.2.1 DLSR protocol implementation

DLSR routing protocol was implemented using the C++ [110] programming language under the simulator ns-3. DLSR protocol uses UDP [111] as the transport layer protocol and UDP port 200 to distribute its management packets. The main modules of DLSR are illustrated in Figure 3.1. According to Figure 3.1, DLSR uses mainly three types of tables to manage network routes, namely, (1) neighbor table, (2) link-state table and (3) routing table. The neighbor table is used to store and manage the neighbor details, which contains the details (such as; Interface IP address, network device address, metric, default gateway and lifetime) of the directly connected interfaces of a particular router. This table is mainly managed by the "hello process" of the protocol. Link-state table stores and manages the link-state updates sent by routers in the network. In other

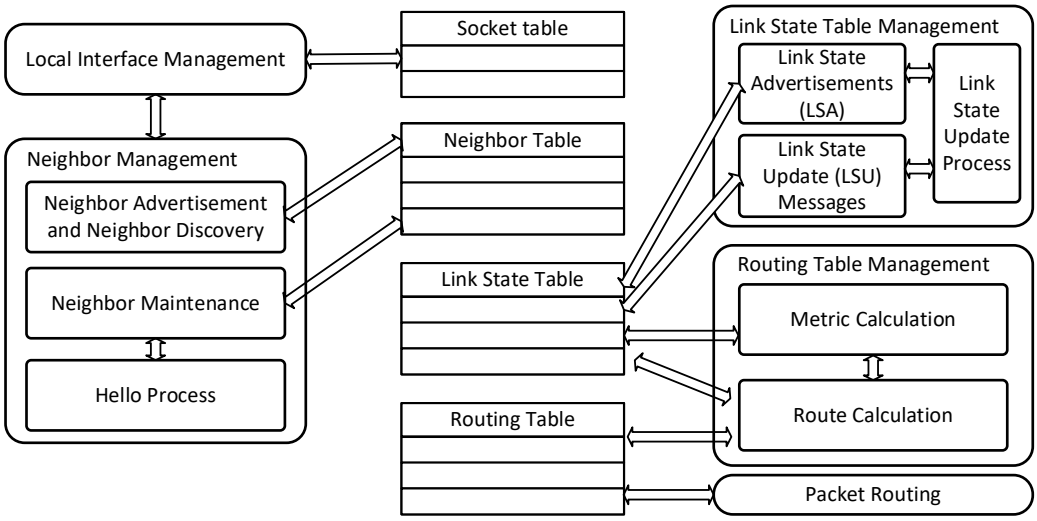


Figure 3-1 Main modules of DLSR routing protocol

```
|      0        |      1        |      2        |       3       |
 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|  Packet Type  |          Packet Size          |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|        Sequence Number        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

(I) Main packet header

```
|      0        |      1        |      2        |       3       |
 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Advertising Router-ID                     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|              Neighbor Router Interface IP Address             |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Neighbor Router Interface Network Mask            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

(II) Hello packet header

```
|      0        |      1        |      2        |       3       |
 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                     Originated Router-ID                      |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| No. of Updates|
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+

|                      Neighbor Router-ID                       |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Neighbor Router Interface Network Address           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|             Neighbor Router Interface Network Mask            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                Neighbor Router Interface Metric               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

(III) Link-state update packet header

Figure 3-2 Packet headers used by DLSR protocol

words, the link-state table contains the neighbor details of all the routers in the target topology. Finally, the routing table contains the calculated routing entries used to route the packets. The routing table is generated using the link-state table with Dijkstra's algorithm [109] combined with the DLSR route calculation algorithm illustrated in pseudocode 1 (explained under metric calculation in section 3.1.2.1).

In order to send route updates and to communicate with the routers in the topology, DLSR protocol mainly uses three types of packet headers as illustrated in Figure 3.2. In the headers illustrated in Figure 3.2, all the lengths of the fields are denoted in bytes. As shown in Figure 3.2i DLSR uses the main packet header to identify and distinguish the packet types. Therefore, all the other management packets are attached to this primary header. According to Figure 3.2(I), the packet type is used to identify the type of the message (hello, link-state update, link-state down the update, etc.). Packet size contains the total size of the packet headers including the appended sub-header sizes and their payloads. Sequence number field is used to validate the packet consistency, mainly during the route update processing. The main modules in Figure 3.1 are explained in detail hereafter.

3.2.1.1 Local interface management module

All the local interfaces attached to a DLSR node are managed through a socket table. DLSR socket table contains all the local link details of a node such as; the socket address, connected interface details, and the link status. During the protocol initialization, DLSR adds all the directly attached interface details of the nodes to the particular nodes' socket table. These link details are updated whenever a local link-state change occurs (up/down). This socket table information is used to send neighbor hello packets during neighbor management process and to select the appropriate local interfaces in sending DLSR updates (Figure 3.1). Moreover, when a node needs to deliver a packet to one of its local interfaces, DLSR uses the socket table information to obtain the local interface details.

3.2.1.2 Neighbor management module

DLSR's neighbor management process is much like the OSPF neighbor management process. The main difference is DLSR uses shorter and simpler headers than OSPF due to the distributed route calculation process. The distributed route calculation process reduces the overall bandwidth usage while sending the updates and the usage of processing resources while processing the updates. DLSR's neighbor management has two main parts. Initially, it performs neighbor advertisement and neighbor discovery. Secondly, DLSR's neighbor management maintains a neighbor table including adding, updating and deleting the neighbor records. DLSR nodes store and manage all the neighbor details in a neighbor table (Figure 3.1). The neighbor table contains information such as the neighbor router identifier (ID), neighbor interface IP address, neighbor interface network mask, route details to the neighbor and the neighbor entry status.

3.2.1.2.1 Neighbor advertisement and neighbor discovery

Upon generating the socket table, DLSR nodes generate their neighbor tables using the

neighbor discovery process. Once the routing protocol is started, each node starts to send hello request packets to their neighbors to identify and retrieve the details of connected neighbor interfaces. Once a hello request packet is received by a node, it must reply to the hello request message using a hello reply message. Nodes only add the neighbor details after receiving the hello reply message to a sent hello request. This is to verify the links as bidirectional links similar with the OSPF protocol. Packet format shown in Figure 3.2(II) is used for the neighbor hello process. It contains the node ID of the advertising node in the advertising router ID field, advertising interface IP address in the neighbor router's IP address field, and the interface subnet mask in the neighbor router interface network mask field. The hello packet type (request/reply) is identified using the packet type field in the DLSR packet header (Figure 3.2(I)).

3.2.1.2.2 Neighbor maintenance

Operating as a link-state protocol, identifying the interface failures is crucial for DLSR nodes. Therefore, the periodic hello advertising time of DLSR is set to 10 seconds. If a hello reply message is not received for $2 * \textit{hello interval}$ (20 seconds), named as the hello interval expiration time, the node assumes the particular connected neighbor interface is down. Then the node who sent the hello request message triggers a link-state down update propagating the detected link failure to all its neighbors. This mitigates the undetected neighbor link down problems which further explains in the section 3.1.4.2, link-state update (LSU) messages. Using the hello process, DLSR generates and manages its neighbor tables which contain all the neighbor details of a particular node.

3.2.1.3 Link-state table management module

DLSR routing protocol contains link-state (topology table) and routing table as the routing tables. Link-state table of a node contains the neighbor details of all the nodes in the topology which used to routing table generation, and the routing table contains the least cost routes to all the nodes in the topology that used for packet routing. DLSR uses two types of link-state packet types for route management namely, link-state advertisements (LSAs) and link-state updates (LSUs). LSAs are used to advertise the link-states of nodes during the initialization and while sending periodical updates to maintain update consistency. LSUs are used to advertise link-state updates/changes such as link up/down events.

Packet header shown in Figure 3.2(III) is used to propagate both of these packet types. DLSR stacks all the link details regarding a particular node in the same link-state packet to reduce the number of packets to be used in the link-state advertising process. The first two fields, originated router ID and the number of updates contain the advertising router identifier and the number of link-state advertisement entries attached with the update, respectively. All the interface details are individually defined using the final four fields of the packet header (Figure 3.2(III)). Therefore, number of updates times of these four fields is attached in each update packet. Each router interface detail contains information such as (Figure 3.2(III)); neighbor router-ID (neighbor router identifier), neighbor router interface network address (network address of the interface), and the neighbor router interface network mask (consists the subnet mask of the connected interface) followed by the neighbor router interface metric value which is the calculated metric for that

particular interface. Similarly, with the hello packets, the packet type is identified using the packet type field in the DLSR packet header (Figure 3.2(I)).

3.2.1.3.1 Link-state advertisements (LSA)

LSA messages are used to advertise the local link details of a node to all the nodes in the network. Initially, LSA messages are sent after the protocol initialization advertising all the links of nodes soon after the hello process. Secondly, nodes send their link-state updates periodically to maintain the update consistency via updating the sequence numbers. All the received LSAs are stored in the link-state tables after validating them. The consistency of the LSAs is maintained by using the periodic link-state update process which triggers in every 30 minutes.

Whenever LSAs updates regarding a node are received with a newer sequence number, the sequence number and the lifetime values of the existing matching link-state entries are updated. A periodic LSA purge interval value ($> periodic\ LSA\ interval$) is set by the network administrator depending on the network properties. If any LSA update for a particular node is not received till this LSA interval, all the link-state entries regarding that particular node are removed from the link-state table. Any LSA messages received with an equal or older sequence numbers are discarded mitigating the message loops.

3.2.1.3.2 Link-state updates (LSU)

As a link-state routing protocol, DLSR only sends LSUs on detecting link-state changes. Let us assume a link between node-A and node-B where the node-A's interface connected to Node-B is changing state from up to down, and this is the only connection among node-A and node-B to reach each other. First, Node-A updates its socket table changing the relevant socket table entry state to down. Secondly, it removes both the corresponding neighbor table and the link-state table entries. Afterward, the node-A recalculates its routing table in order to update the routes affected by the interface down. Finally, it sends the link-state down update to all of its neighbors. If by any chance, if node-B cannot detect the link down via its interface, all the routes through node-B's interface routes through node-A are then become invalid.

This scenario is mitigated by using the "hello interval expiration time." If a node did not receive a hello advertisement before the hello interval expiration time ($2 * hello\ interval$), the node assumes the interface in the other end is down and it sends the updated link-state down update to all of its neighbors informing the link down/unreachability. This allows both of the network portions (both A and B) of a link to be updated regarding the link down state and notify the total topology regarding the link-state change. The two messages sent from node-A and node-B can individually identify using the originated router ID field in the link-state update packet header as shown in Figure 3.2(II). All the link down updates are marked with a metric of "0" and in DLSR metric of value "0" is treated as an infinite metric.

3.2.2 Routing table management module

DLSR route packets according to the generated routes using the routing table entries, which created using the received link-state updates. DLSR uses its own route metric value which is

calculated using an extended Dijkstra's algorithm.

3.2.2.1 Metric Calculation

The metric calculation formula is illustrated in equation (3-1). DLSR metric value also contains the link delays together with the bandwidth allowing DLSR to calculate a much accurate metric value than RIP or OSPF. The services, which depend on the link delays (services that transmit time-sensitive data) such as haptic and voice/video services can get the benefit from this in selecting the least delay paths in delivering packets to destinations. According to the equation (1), best practice in selecting a reference bandwidth is raise the reference bandwidth to a more reasonable number in the context of currently available link speeds. Therefore, the reference bandwidth ($Bandwidth_{Ref}$) for DLSR is taken as 10^7 (10 Gbps) due to the fact that the current network link speeds are working in the range of Gbps. DLSR route metric is enhanced by adding the link delay to the equation compared to the OSPF protocol metric. This allows to use a more precise metric value than OSPF. According to equation 3-1, $Bandwidth_{Link}$ is the bandwidth value in Gbps of the link, which need to calculate the metric value, and $Link\ Delay$ is the delay of the link in microseconds. According to the equation 3-1, lowest calculated metric value is the better metric value or the least cost path to a particular destination.

$$Metric = \frac{Bandwidth_{Ref}}{Bandwidth_{Link}} + Link\ Delay \tag{3-1}$$

DLSR routing protocol allows nodes to calculate their own routing tables regarding the total network using the received link-state updates. For this, DLSR uses an extended version of Dijkstra's algorithm as shown in Pseudocode 3.1. Using this algorithm, DLSR routing protocol calculates the least cost paths to all the nodes of a network. DLSR uses the calculated routing table entries in order to route the packets to other nodes or the socket table to deliver the packets locally to its interfaces. Pseudocode 3.1 is briefly explained hereafter.

According to Pseudocode 3.1, DLSR first clears the routing table and removes any expired link-state updates in the link-state table (line 1-2). Then the algorithm takes all the neighbor table entries and adds the network entries omitting the broadcast network entries to the routing table, as they are directly connected least cost networks of the neighbor nodes (line 4-8). If there are entries in the link-state table (line 10), all those entries are added to the adjacency list used in the Dijkstra's algorithm (line 11-20). Dijkstra's algorithm requires a node list containing all the nodes in the received link-state updates. This node list generation is performed at the same time while DLSR is generating the adjacency (line 17-19). Then it runs the Dijkstra's algorithm to calculate all the least cost paths to all the nodes in the network from the current node using the received link-state updates (line 22). The calculated Dijkstra's data contain individual least cost paths to all the nodes in the network (from the current node) and the least cost paths to reach those particular nodes from the current node. Moreover, the path data contains all the intermediate nodes along the path to reach that particular destined node.

In the next step, DLSR iterates through the total node list (line 24), which created previously (line 17-19), and obtains the shortest path to the current node of the current iteration from the total node list using Dijkstra's algorithm (line 25-28). Then it obtains all the neighbor network

Pseudocode 3.1 Routing table generation

```
PROGRAM GenerateRoutingTable:
1:   Clear routing table;
2:   Purge expired entries from link state table;
3:
4:   WHILE (neighbor table contains entry)
5:        DO   IF (current entry != broadcast entry)
6:                  THEN  Add the current entry to the routing table;
7:             ENDIF;
8:   ENDWHILE;
9:
10:  IF (Number of  link state table entries > 0)
11:       THEN   Clear adjacency list;
12:            allLinkStateRoutes = Get all routes from the link state table;
13:
14:            WHILE (allLinkStateRoutes contains entry)
15:                 DO   currentLinkStateEntry = Get the next entry from allLinkStateRoutes;
16:                      Add the currentLinkStateEntry to the adjacency list;
17:                 IF (currentLinkStateEntry advertising node != This node)
18:                      THEN  Add the currentLinkStateEntry node details to the totalNodesList;
19:                 ENDIF;
20:            ENDWHILE;
21:
22:            Compute the shortest paths from this node to all the other nodes
                    using Dijkstra's algorithm(DA);
23:
24:            WHILE (totalNodesList contains entry)
25:            DO    currentNode = Get the next entry from totalNodesList;
26:                 Get the shortest path to currentNode entry from this node using DA;
27:
28:                 minDistance = Get the minimum distance to currentNode from this node;
29:                 allRoutes = Get all the neighbor network routes for the currentNode
                                 from the link state table;
30:
31:                 WHILE (allRoutes contains entry)
32:                      DO   currentRoute = Get the next entry from allRoutes;
33:                           existingRoute = Get entry from routing table network address
                                          equals to the network address of the currentRoute;
34:
35:                      IF (currentRoute metric value < existingRoute metric value)
36:                           THEN  Replace the existingRoute with the currentRoute;
37:                      ENDIF;
38:                 ENDWHILE;
39:            ENDWHILE;
40:  ENDIF;
  END.
```

details of the current iteration node using the link-state table (line 29). Finally, the algorithm adds the network details of all the neighbor networks of that node to the routing table (line 31-38). While adding the neighbor network entries, the algorithm checks for any existing entries in the routing table and replaces those entries if the route metric to the destination of the current entry is lesser than the existing routing table entry (line 32-37).

When the nodes are advertising their routing table details to other nodes, a DLSR node only sends its link-state updates that contain only the neighbor details of the advertising node. This drastically reduces the bandwidth used in the route update process where all the existing distance vector routing protocols implemented in ns-3 sends the total routing table entries of each node. Due to the very few entries in the link-state updates compared with the conventional ns-3 routing protocol route updates, DLSR also drastically reduces the processing power together with the processing time required to process the updates.

3.2.2.2 Packet routing module

Up on receiving a packet, DLSR checks its routing table for a valid route for the packet destination. If a valid entry is found, then the packet is forwarded through the interface according to the routing table entry. However, if any entry is not found, the packet is stored in a buffer for a predefined amount of time. DLSR tries to deliver the packet periodically between this buffer times. DLSR discards the buffered packets upon expiring the buffered time.

3.2.3 Evaluations

As the first evaluation, bandwidth usage for the routing updates of the DLSR routing protocol evaluated with the OSPF routing protocol. The proposed DLSR routing protocol was implemented in ns-3.23 where the OSPF protocol is simulated in ns-3-dce-1.6. Both of the protocols were simulated in the ISP RocketFuel network topology (AS3967), shown in Figure 3-3, containing 79 nodes which were generated using the ISP topology mapping engine RocketFuel [112] [113].

3.2.3.1 Throughput consumption evaluation

In this test case, the link throughput used for the route initialization and management processes were measured for each protocol. The simulation was scheduled for 30 minutes in the topology illustrated in Figure 3-3, and no other traffic was transferred throughout the time. The throughput of all the routers was measured, and the obtained test results are plotted in the graph shown in Figure 3.4(I). According to Figure 3.4(I), we can see that the DLSR protocol has consumed less throughput for its route advertising and management processes than the OSPF protocol.

In the second evaluation, both of the protocols were evaluated in grid networks for their throughput usage for route initialization and management processes while increasing the nodes of the grids. All the links of the grid network topologies were programmed with 1 Gbps link data rates and 2 ms link delays. The simulation time was programmed to 15 minutes for this evaluation. The node size of the grids was increased starting from 4 nodes then incrementing to 9, 16, 25, 36, 49, 64 and up to 81 nodes respectively. Both of the protocols were executed in each grid with the same parameters, and the total throughputs of each node were measured. Finally, the average throughputs of each grid test case were calculated and plotted in the graph as shown in Figure 3.4(I). According to Figure 3.4(II), DLSR protocol clearly consumed less throughput than

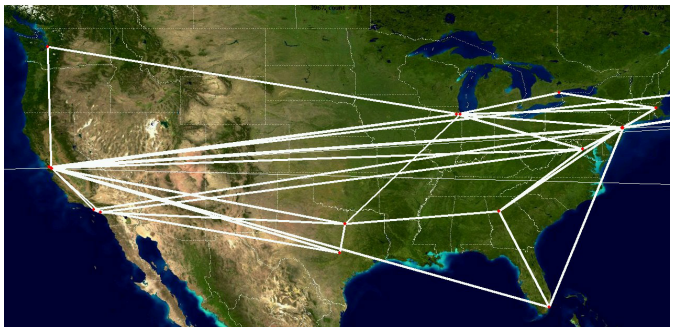
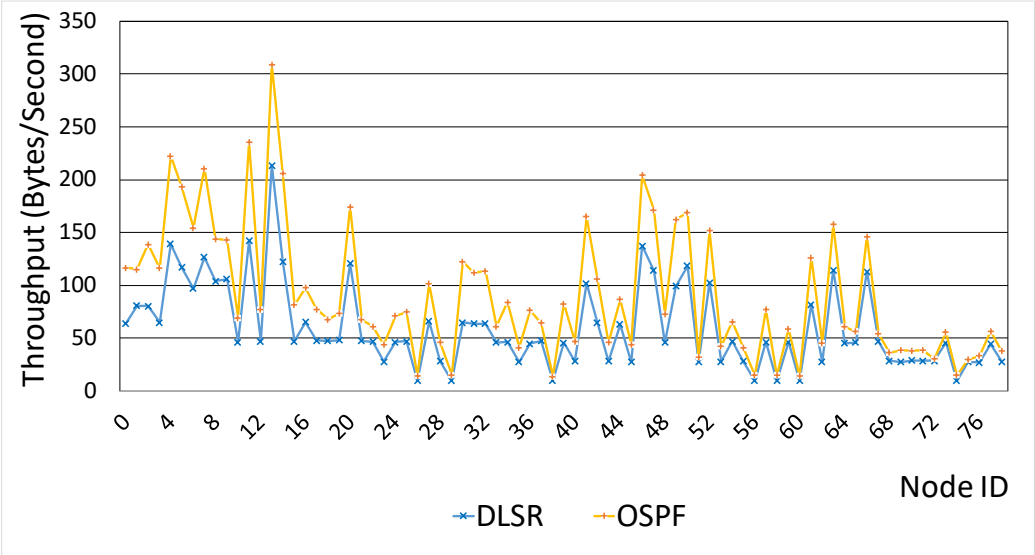
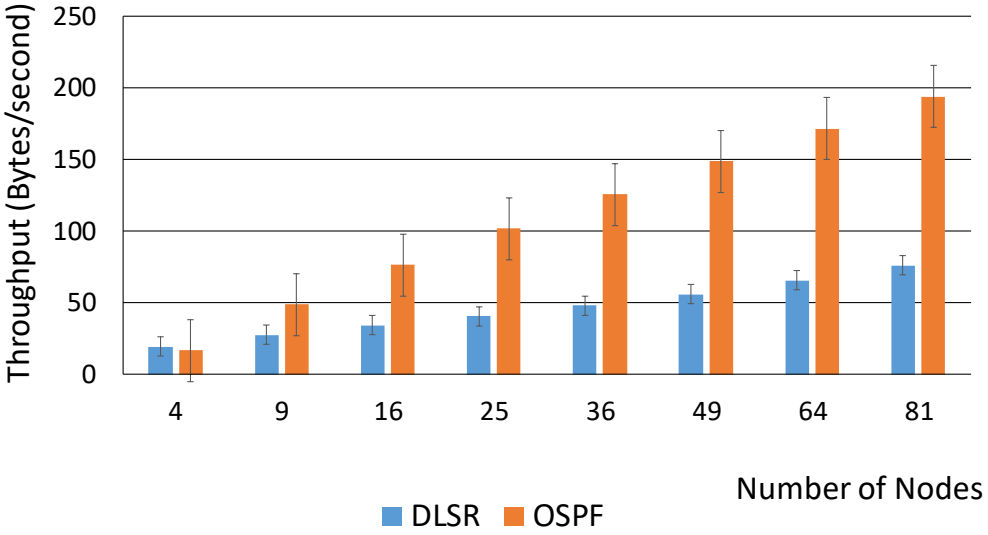


Figure 3-3 Simulation topology - Exodus USA (AS3967)



(I) AS3967 topology



(II) Grid topologies

Figure 3-4 Throughput consumption for route management

the OSPF protocol for its route management and the route update processes. Therefore, we can conclude that the DLSR protocol uses less bandwidth for route updating and management than the OSPF protocol in larger networks.

3.2.3.2 Delay variance

As the Next test case, AS3967 topology was programmed to send constant bit rate TCP packets from node #1 to node #79. Server was configured with the port number 4,000, packet size of 1,024 and inter-packet interval of 0.05 seconds. The trace files generated after executing the test case were analyzed using the TraceMetrics software [114] which is a trace file analyzer for ns- 3. The results obtained via analyzing the TCP streams are summarized in Table 3-1. According to the measured values, the IPDV variance is less in the DLSR over OSPF which

Table 3-1 TCP stream evaluation results of DLSR vs. OSPF

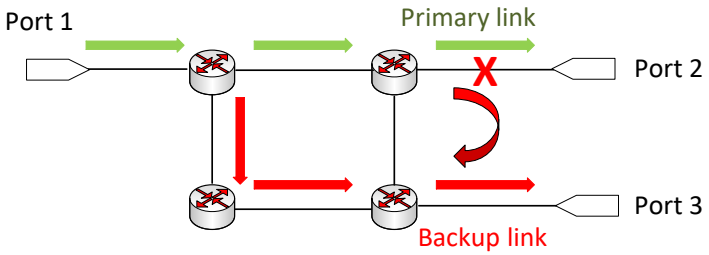
Property	DLSR	OSPF
Average Delay	0.0022014	0.0105394
Average PDV	0.0002201	0.0005394
PDV Variance	0.39006 E-7	2.5037 E-7
Average IPDV	-1.0279 E-7	-3.1790 E-7
IPDV Variance	0.44819 E-11	2.9188 E-11

results less jitter. Also the average PDV value denotes a one-way-delay variation and the test results shows that the DLSR has 59.19% less PDV than OSPF.

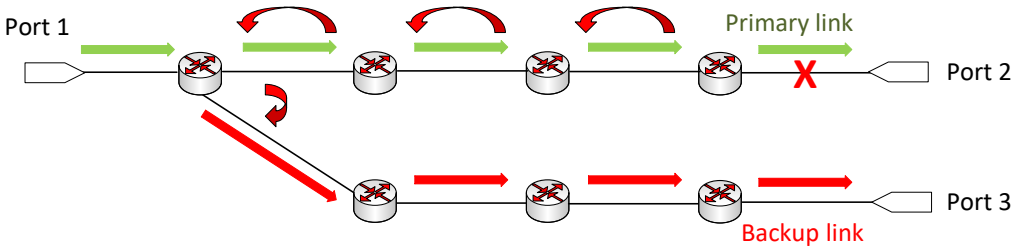
3.2.3.3 Convergence evaluation

Convergence has a direct influence on users' perception of quality. Measurement of convergence time from the moment of a service disruption to full-service restoration is a key performance indicator for service providers [115]. Therefore, as the next test case, the convergence delay of the DLSR protocol was measured. The most common classical method for determining convergence is illustrated in Figure 3-5 [115]. In this test, constant bit rate (CBR) UDP traffic was generated from port 1 which is the sender. The received packets in port 2 and port 3 were counted. In an error-free scenario, traffic flows from port 1 to port 2. In between the packet transmission, link break is simulated in port 2. Therefore, the DLSR detects the link down and recalculates the alternative path through port 3. In this scenario, the number of packets lost during the transition can be used as a measure of the convergence time.

The CBR traffic generator was programmed to generate UDP packets once in each millisecond with a payload of 100 bytes. The convergence delays were measured for two test scenarios as shown in Figure 3.5 [115]. Figure 3.5(I) denotes the simple convergence scenario, and Figure



(I) Simple convergence test case



(II) Complex convergence test case

Figure 3-5 Convergence test case topologies

3.5(II) shows the complex convergence scenario. All the links in the simulations were set with a throughput of 1 Gbps and 2 ms delay. According to the test results of the simple convergence test case, 11 packet drops were detected. Moreover, in the complex convergence test case, 17 packet drops were detected. Therefore, it concludes that the convergence times for the simple convergence test case and the complex convergence test case are 11 ms and 17 ms respectively. For OSPF the values were 14 ms and 22 ms respectively. One of the main advantage of the distributed link-state route calculation method of DLSR is that it does not require to exchange any packets to generate the routing tables once the router has all the link-state updates of all the routers in the topology. The convergence time was totally consumed for the route table recalculation process and to send the route down updates to its neighbors until the reroute point (port 1).

3.2.4 Conclusion

DLSR protocol was designed and implemented as an extendable protocol for used in content-based routers. As the content-based routers perform many functionalities than the ordinary routers, minimizing the delays and routing overhead via the routing protocol is a must. To this end, this study proposed the DLSR protocol. DLSR is a distributed routing protocol which extends Dijkstra's algorithm via adding link propagation delay as one of its metrics. The test results showed 20% average delay variance improvement together with averagely 60% less throughput consumption over larger networks compared with the conventional OSPF protocol. Moreover, according to the TraceMetrics analysis results, the DLSR protocol also results in a better PDV and IPDV values over the OSPF protocol. The convergence test results show that the DLSR only consumes 11 milliseconds and 17 milliseconds to the evaluated simple and complex convergence test cases respectively.

Lack of proper dynamic wired routing protocols in ns-3 is a huge drawback to all the developers and researchers who are willing to develop and research on wired routing infrastructure and services which are based on wired routing. DLSR can be used as a baseline to develop advanced link-state routing protocols or to implement Internet-based services on top of it. Finally, the open source, architecture in ns-3 simulator let the research community and the developers to use, verify, reproduce and review the DLSR protocol by a wider community.

3.3 Per-hop data encryption

3.3.1 Per-hop data encryption service implementation

The proposed service uses symmetric key encryption to encrypt data among content-based routers. In order to perform encryption in one content-based router and decryption in another, both the routes must share a common symmetric key. The most difficult task in encryption is transmitting a secret key through the public networks. Neighbor key exchange process performs the shared key generation and exchange among the neighbor interfaces securely.

3.3.1.1 Neighbor key exchange

The proposed service uses the standard Diffie-Hellman (DH) [57] key exchange algorithm in order to generate symmetric keys among neighbors by exchanging some values through the public network. The main advantage of this algorithm is that though the values are exchanged through the public networks, intruders who obtain the packets or sniff traffic may not be able to generate the shared secret.

Content-based routers create their neighbor tables using the hello and hello reply packets exchange process. After exchanging the hello packets, routers initialize the DH shared key distribution process with every neighbor in the neighbor table. Upon successful key exchange, the routers generate a key table and add key information, neighbor information, and its local interface information. The protocol is flexible enough to renew the keys upon the user's or application's request. This is achieved via retriggering the DH algorithm between any routers accordingly. Further, the underneath IPv4 routing protocol exchanges the routing information simultaneous to the key exchange process. Therefore, the proposed per-hop data encryption protocol is capable of exchanging both routing and DH key information between neighbors with minimum delay. Consequently, the end users are not required to perform key exchange prior to the connection initiation that is an essential requirement of the end-to-end encryption technologies. The topology inherently does this automatically.

3.3.1.2 Data encryption process

Whenever a router receives a packet or a data stream that needs to be routed, it first checks whether the packet or stream requires encryption. This capability enables the proposed service to act as an intermediate between the existing link encryption and end-to-end encryption methods as it allows both unencrypted packets and encrypted packets to be routed simultaneously. Moreover, it will enable the routers to reduce its burden in encrypting and decrypting all the packets that flow and to facilitate both encrypted and unencrypted communications simultaneously according to the client or application requirements. On the one hand, if the received packet requires the encryption service, then, the packets will be routed after being encrypted by the router. On the other hand, if the router receives an ordinary packet, which does not require encryption, then it will treat the packet as a standard packet and route it following the general routing procedure.

After receiving an encryption-required packet, the router first checks its routing table and selects the most efficient next hop required to forward the packet. Then, the router obtains the particularly shared key according to its key table. Next, it encrypts the packet content using the obtained key. Moreover, the encryption level can be selected according to the requirements of the client or the application or according to the security policy of the link.

In a communication, the proposed service only encrypts the payload and sends it to the next hop router along with the original IP header and attaching custom header information. This permits successful identification of the packet at the next hop router. The custom header will include fields that uniquely distinguish each packet type without any conflicts. Upon successful

retrieval, the next hop router, which receives the packet, will decrypt the packet using the same shared key obtained from its key table. Then, after reading the original sender and receiver details, if it is not addressed to itself, the above process is repeated after selecting the most effective and efficient neighboring hop to forward the data. Further, if the packet requires encryption, it will be encrypted using the same process, and the packet will be forwarded to the appropriate next hop router. This process continues until the final router sends the decrypted packets to the receiver.

The main advantage of this method is that the most efficient and effective path can be selected according to the real-time link conditions. This will speed up the packet transmission while optimizing the congestions in the wide area networks. In addition, the proposed service will allow simultaneous transmission of both ordinary packets of the network and the packets that need to be securely delivered via encryption. Meanwhile, the encryption level and the key size will also be configured dynamically according to the sensitivity of the data or the client and the application requirements.

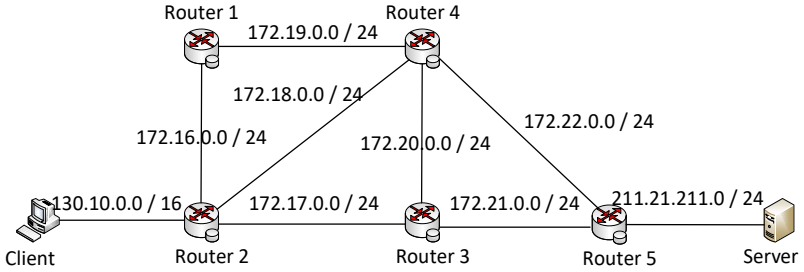
In this implementation process, a router can use different individual key lengths for all their interfaces. Nowadays, in the case of encryption, many nations have imposed restrictions on key lengths for the data traveling through their networks. The proposed method helps in overcoming this by maintaining the required key length based on these requirements in every interface as necessary. Therefore, whenever the rules change, the key lengths can be managed centrally thereby giving total control to the nations according to their requirements.

3.3.2 Evaluations

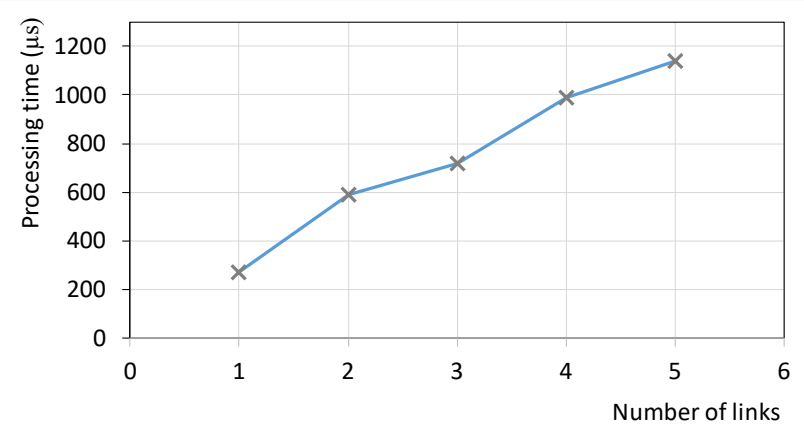
3.3.2.1 Evaluation of processing cost for key exchange

The main drawback of the proposed protocol is the delays caused by the encryption and decryption processes while routing encrypted packets. In the implementation, Advanced Encryption Standard (AES) is used as the encryption algorithm with a 256-bit key length. Further, there tends to be an initial time cost for the key exchange phase until the DH algorithm is completed. Therefore, the key table creation time has been measured in a network topology, as illustrated in Figure 3-6(I). All the links in the topology were configured with a throughput of 5Mbps and a 2 ms delay. While obtaining the test results, the total processing time consumed by the processor in executing the particular functions in microseconds using the real-time stamp counter of the CPU is used as the processing time.

According to Figure 3-6(I), the busiest router in the network is Router 4 because it is connected to five links resulting in managing a maximum number of links. Therefore, the processing time taken by Router4 in generating DH keys while increasing the links from one to five were individually measured, and the results were plotted as illustrated in Figure 3-6(II). From the results, we can substantiate that the processing time taken for the DH key generation process increases with an increase in the number of links attached to a router. When the router contained only a single link, the processing time was 268.33 μs; this increased to 1,140.48 μs when it was extended to five links. These processing times also include the processing time consumed by



(I) Simulation topology



(II) Consumed processing time for DH key exchange in Router 4

Figure 3-6 Processing time consumption evaluation for the DH key exchange process

the route propagation of the protocol that runs simultaneously with the key exchange.

3.3.2.2 Evaluation of the encryption costs

Furthermore, we measured the time taken for the encryption and decryption processes of the proposed service. The test was executed using the same topology given in Figure 3-6(I). In this test case, Client was programmed to send the encryption required packets destined to the Server as presented in Figure 3-6(I). Client was started 0.5 s after the start of the simulation. Then, Client sent a fixed number of packets at an interval of 0.02 ms. The first 0.5 s were allocated to create, and exchange the DH keys, create key tables, and propagate the routing tables of the proposed protocol.

The packets traveled through the path Router 2, Router 4 and Router 5 to reach the Server. This path was selected automatically by the protocol since it was the least cost path to reach the destination. Moreover, the main advantage of using this path is that the packet would travel through routers that contained connected links such as four, five, and three accordingly. Therefore, this resulted in measuring the processing times of Router 2, Router 4, and Router 5 where the burden of the routers increased according to the number of connected links.

We measured the processing time in routers: Router2, 4 and 5, for following scenarios: 1) process unencrypted packets, 2) process end-to-end encrypted packets, and 3) process per-hop encrypted packets. Further, we measured the processing time for 20 test cases using 2,000 to 40,000 packets, and we used average processing time for the comparison. The results comparison

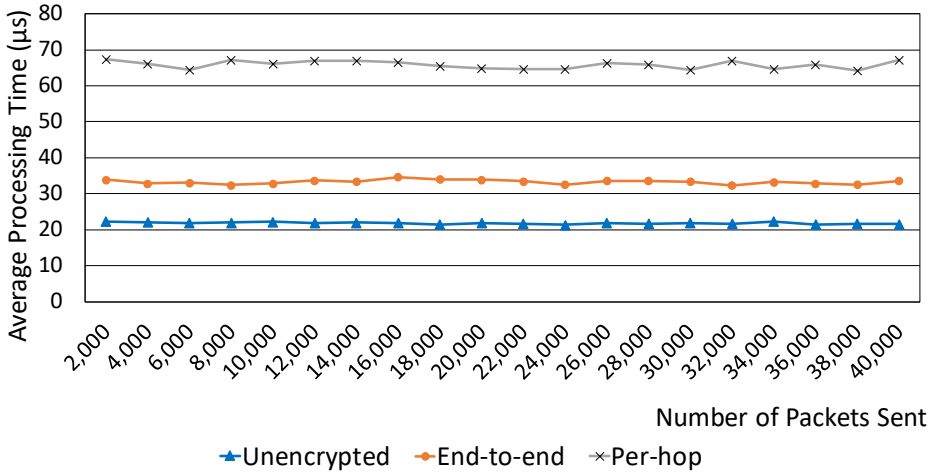


Figure 3-7 Processing costs for unencrypted, end-to-end encryption and per-hop encrypted packet transmission

is given in Figure 3-7. According to Figure 3-7, routers consumed averagely 21.85 μ s to process unencrypted packets. Furthermore, end-to-end encryption packet processing consumed averagely 33.31 μ s, which is 34.41% increment. That was expected on the routers due to the extra process consumption to manage key tables and process the end-to-end encryption packets. Finally, routers consumed averagely 65.86 μ s for the processing of per-hop encrypted packets. That is due to manage key tables, routing and, further routers have to decrypt and encrypt every packet in each hop.

Nonetheless, it is average 66.82% increment compared to unencrypted packet transmission, and 49.42% increment compared to end-to-end encrypted packet transmission. However, as every router pair uses a unique key, the original content of the packet can be obtained via decryption only by the destined router. Therefore, compared to other two methods, the proposing method shows much secure and flexible data transmission over public networks.

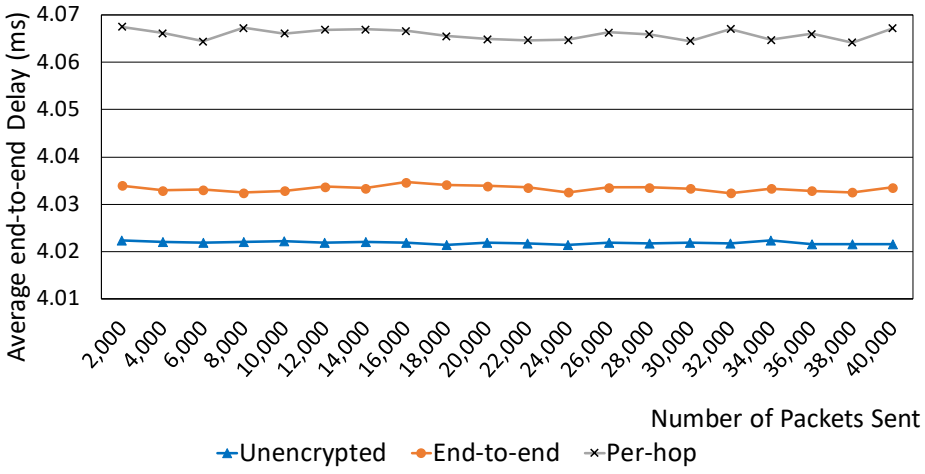


Figure 3-8 Average end-to-end delay

3.3.2.3 Evaluation of the end-to-end delays

Finally, we measured the end-to-end delay for all aforementioned cases. The results are plotted in Figure 3-8. According to the figure, an unencrypted packet consumed about 4.0219 ms to propagate from Client to Server. Nevertheless, end-to-end encrypted packet consumed about 4.0333 ms to propagate from Client to Server. The proposed method consumed about 4.0659 ms and it is 0.044 ms (0.8%) increment compared to end-to-end encryption. That is because the routers consumed additional processing time for the decryption and encryption processes in each hop. Even though the proposed method displays 0.03 ms increment of packet propagation delay, it is clear that the proposing method has the merit as the most concerning point over public networks as security over the links.

3.3.3 Conclusion

In this section, a per-hop data encryption service that can be used to transmit sensitive data over public networks was proposed using content-based routers. The proposed data encryption protocol can simultaneously provide both secured packet routing and ordinary routing according to the user requirement. Moreover, the test results showed that the proposed method can be used with end-to-end encryption as well. Furthermore, the process method is able to provide higher security to the data travers through public network by ensuring integrity and confidentiality. Additionally, the proposed per-hop data encryption service provides the flexibility to change the encryption algorithm over the communication path even during data transmission and the ability to use multiple encryption algorithms among the packet propagation path with different keyspaces. However, those flexibilities come with the cost of additional 0.044 ms end-to-end delay in data transfer.

Chapter 4 Secured services for public networks using content-based routers

4.1 Per-hop data encryption protocol

To provide extended secured services such as entire packet encryption, neighbor data retransmission, and secured packet traceability using the content-based routers, a proper foundation which can provide security from the routing layer is vital. To this end, this study proposes the per-hop data encryption protocol which merges the DLSR routing protocol and per-hop data encryption service proposed in Chapter 3. The proposed secured services in this chapter use the DLSR routing protocol to fetch the appropriate packet related details such as a packet header, neighbor, and route details. Furthermore, they use the merged per-hop data encryption service on top of the DLSR to provide the security. Moreover, this shows that the proposed per-hop encryption service can be easily integrated to and compatible with any conventional routing protocols.

Furthermore, other enhancements to the per-hop data encryption service and DLSR protocol were made during the merge process in order to extend the functionality of the proposing secured architecture. Initially, DLSR was extended to compatible with IPv6 addresses. Secondly, the merged per-hop encryption service was added with more encryption algorithms such as IDEA, DES, AES-GCM, and AES-CTRs. Additionally, a certification authority (CA) was attached to enhance the security integrating the public key cryptography or asymmetric cryptography for key exchange and verification processes. Hereafter, all the proposing services will be using this new merged and enhanced per-hop data encryption protocol.

4.1.1 CA architecture integration

During content-based router initialization, all routers generate unique public and private key pairs for each of their interfaces. To create these private and public keys, routers use the RSA [58] public key infrastructure. For this implementations, the routers generate RSA private keys and public keys with a key length of 2,048 bits. Upon successful key generation on all interfaces, the routers send all their interface public keys with the additional interface information to the CA. In addition, the routers store and manage their keys in their own key tables, where the private key is not shared or sent anywhere.

In this architecture, routers send data securely using the per-hop encryption method. In order to perform per-hop encryption, the connected neighbor routers interfaces should share the same symmetric key; or else they can use the PKI infrastructure and encrypt the sending data using the public key of the neighboring router interfaces. Upon retrieval, the retrieved interface can

then decrypt the data using the private key of that particular interface. However, the main disadvantage of using the PKI infrastructure is the encryption and decryption delays due to the extensive processing power required by these algorithms. Therefore, it is inappropriate to use such a mechanism to perform encryption and decryption where the router interfaces require higher packet service rates in public networks. The solution to this problem is to use the symmetric key encryption, where both the communicating parties share the same key. However, the main practical problem is to share the secret keys through the communication media, especially through the public networks on the Internet.

For this purpose, we propose the content-based router involvement with the CA architecture. Accordingly, once the routers generate PKI key pairs in all of their interfaces, and after sending them to the CA, the routers generate symmetric keys for all of their interfaces. For symmetric key encryption, the proposed architecture uses the AES encryption algorithm operating both in galois/counter mode (GCM) and counter mode (CTR) modes. Cisco approves AES-GCM as a next-generation encryption algorithm [116]. Moreover, these AES modes are used in the industry for gigabit-scale data streams. The key length of the interfaces can be different depending on the user, application, service, or infrastructural limitations and requirements. Once the routers have created the AES keys for all of their interfaces, they request for all their neighbor interfaces the PKI public keys from the CA. After verifying the requests, the CAs send the requested public keys to the senders. Upon retrieval, the routers encrypt the corresponding AES keys using the corresponding neighbor interface public keys using the RSA encryption algorithm; they then send the encrypted AES keys to their neighbors. If these packets are found by any person or device other than the destined receiver, such as an attacker or intermediate listener, the latter would be unable to read or decrypt the original AES keys because they would not have the PKI private keys of the public keys used to encrypt the AES keys. Consequently, only the neighbor router interface would be able to decrypt and acquire the AES key using the RSA private key.

The AES key-sending process is performed by both neighbors ends. Therefore, to avoid conflicts, the interfaces attach a timestamp value with the AES key packet when they send the keys. The neighbor interfaces agree on a key only if the neighbor has sent the key before itself. If a received packet contains a timestamp higher than its sent timestamp value, the interface discards that key packet.

4.2 IP-Routable entire-packet encryption service

4.2.1 Motivation

The proposed (section 3.2) per-hop data encryption service was not able to secure the packet headers as it did not have proper access to the routing layer. TCP/IP protocol stack expose the sensitive details in the TCP/IP headers such as IP addresses. E2EE methods can expose weak crypto parameters in the intermediate vulnerable network hops. Tor has the exit node vulnerability and the huge end-to-end delays due to the anonymization process. Moreover, such conventional methods impedes data analysis where the content-based routers cannot provide

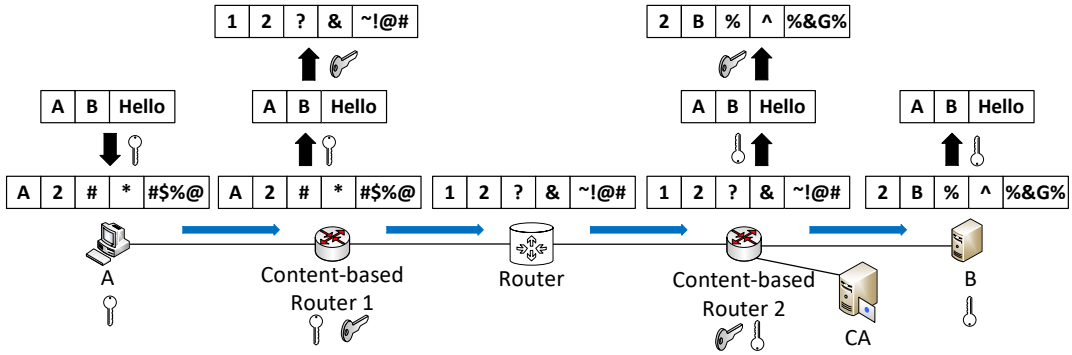


Figure 4-1 Proposed entire packet encryption service

content-based services to the end-users. Therefore, a secured and flexible entire packet encryption service is crucial for the proposed secured architecture using the content-based routers. To-this-end this section propose an IP-routable entire-packet encryption service using content-based routers.

4.2.2 IP-Routable entire-packet encryption service implementation

As illustrated in Figure 4-1, adjacent routers generate and share the link encryption keys using the CA architecture. On receiving an entire-encrypted packet, the router decrypts the packet headers using an appropriate link encryption key and obtains the original header sensitive details such as the ports, and source and destination IP addresses. It then uses those details to search the next-hop router and the particular link encryption-key. Using the retrieved information, the router encrypts the data and original header portions. After completing the encryption process, the router alters the IP header of the packet.

The router replaces the IP header's sender and receiver details of the received packet with the current router forwarding-interface IP address and the next-hop router IP address, respectively. This process allows the packet to be IP routable to any intermediate device between the current and the next-hop routers. The packet-decryption key is only known to the current and the next-hop routers; therefore, the original IP header details are only readable by these two devices. Consequently, the only IP header details that an intermediate device or packet sniffer can read from a packet is that it is transmitting from the current to the next-hop router.

4.2.3 Entire-packet encryption packet header structures

As illustrated in Figure 4-2, the proposed entire-packet encryption service uses its own header structure to transmit the encrypted packets. Figure 4-2(I) shows the header-encapsulation structure used by the proposed entire-encryption service. The entire-encrypted packet is transmitted encapsulated in its own basic header structure (Figure 4-2(II)). Finally, it is encapsulated in the IP header, thereby enabling it to be IP routable.

Initially, the proposed entire-packet encryption service uses four encryption-header types as illustrated in Figure 4-2(III), (IV), (V), and (VI). All the packet types transmitted by the

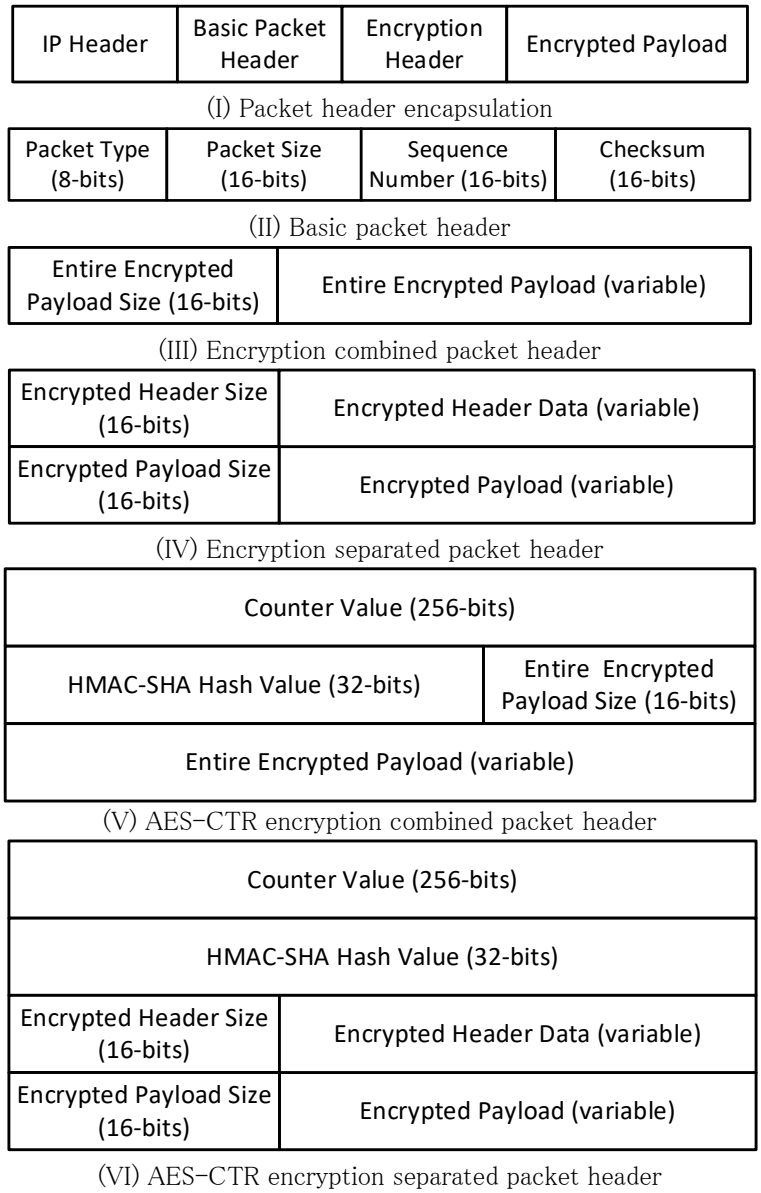


Figure 4-2 Packet header structures used by the entire packet encryption service

underlying per-hop encryption protocol are attached to the basic packet header. As shown in Figure 4-2(II), the packet type field defines the type of the current packet such as entire-encrypted combined packet or separate entire-encrypted packet. The packet size field is used to identify the transmitted packet size. The sequence number field is used to identify the sequence of the packets. This field is mainly used by the underlying routing protocol during the routing-table generation phase to check the consistency of the route-update packets. Moreover, this value is used to provide anti-replay protection to the packets transmitted by the proposed service. Finally, the checksum value is used to check the basic integrity of the data and identify if the packet is erroneous.

The implemented secure architecture uses the random-number generation algorithms available in the Crypto++ library [117] to generate the required unique random numbers. However, users

can also use the rand() function in C++ instead of the random-number generation functions available in the Crypto++ library. The proposed service uses the system time value during encryption as the distinctive value.

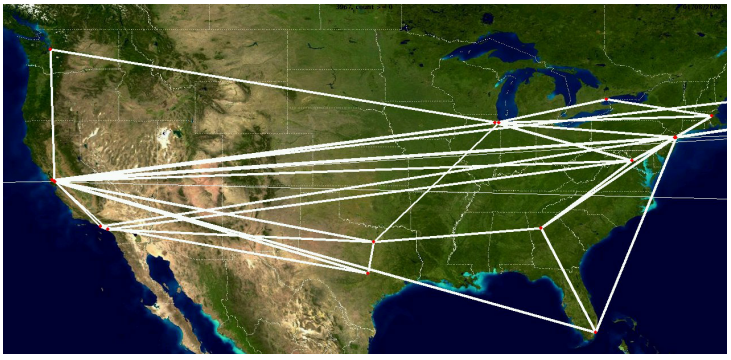
As an additional counter value is required for each encryption in the AES-CTR mode, it is mandatory to use a fast and secure method to share the same counter value for each packet between the communication parties. The proposed service attaches the counter value to the packet and sends it using the AES-CTR header structures illustrated in Figure 4-2(V) and (VI).

As the encryption key is shared using PKI and is only known to the communication parties, exposing the counter value does not cause any type of security breach. However, while the packet is in transmission through public networks, attackers or intruders can easily alter the counter value. Therefore, an appropriate integrity checking technique must be used for verification.

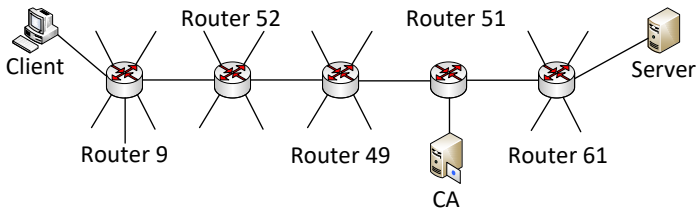
The proposed service uses hash-based message authentication code-secure hash algorithm (HMAC-SHA) to authenticate the counter value. As shown in Figure 4-2(V) and (VI), the packet header is attached with the calculated HMAC-SHA value. This value is calculated for the appended counter value using the encryption key (only known to the communicating parties); this allows the communicating parties to identify any modifications to the counter value.

The HMAC-SHA hash value field contains the hash-based message authentication code for the counter value of the current packet. Generally, the encrypted header size/encrypted payload size fields contain the size of the encrypted header/payload values, respectively. Moreover, the encrypted header data field contains the encrypted headers, and the encrypted payload/entire-encrypted payload fields contain the encrypted payloads.

Headers in Figure 4-2(I) and (IV) are used to encrypt the payload and header data as a single



(I) Simulation topology - Exodus USA (AS3967)



(II) Packet propagation path

Figure 4-3 Simulation topology information

field, whereas those in Figure 4-2(IV) and (VI) are used to encrypt the headers and data. The IP header's sensitive data is concatenated and encrypted as a single string value inside the entire-encrypted payload field or together with the data inside the encrypted header data, depending on the entire-encryption packet type.

4.2.4 Topology implementation and simulation

The proposed entire-encryption service was simulated using an internet service provider (ISP) topology (AS3967), as illustrated in Figure 4-3(I). The topology contains 79 nodes, which were generated using the RocketFuel ISP topology-mapping engine [112] [113]. The test-bed network comprised a client, content server, CA server, 79 routers, and 294 links among the routers. All links in the topology were configured with 1 Gbps bandwidth and 2 ms delay. In the implementation, IDEA, DES, and AES encryption types AES-GCM and AES-CTR were used as the encryption algorithms with 128 and 256-bit key lengths. During testing, we obtained the total processing time consumed by the processor in microseconds while executing particular functions, using the real-time stamp counter of the CPU.

4.2.5 Evaluations

4.2.5.1 Evaluation of the processing time

As illustrated in Figure 4-3(I), the client was programmed to transmit encrypted packet types destined to the server, predominantly in the following two scenarios:

1. Entire-encrypted packets (header + payload as a unit)

2. Separately entire-encrypted packets (header and payload separately)

In each scenario, three sub-scenarios were executed for each encryption method and configured to use three different test cases as mentioned below.

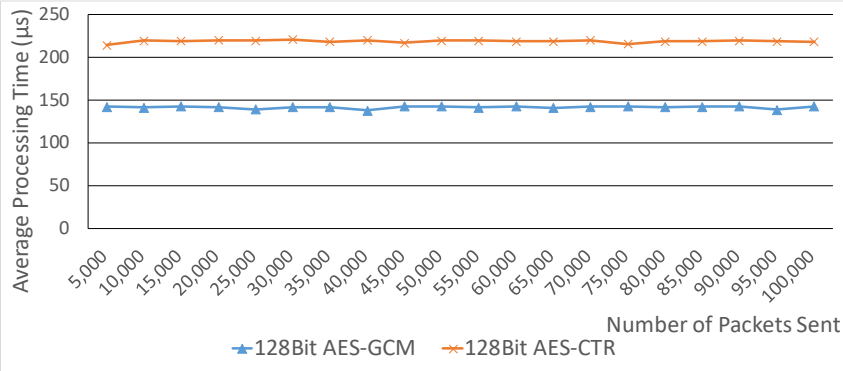
1. Measuring the average processing time (APT) for AES-GCM vs. AES-CTR encryption algorithms.

2. Measuring the APT for IDEA, DES, AES-GCM, and AES-CTR while increasing the packets per second (pps).

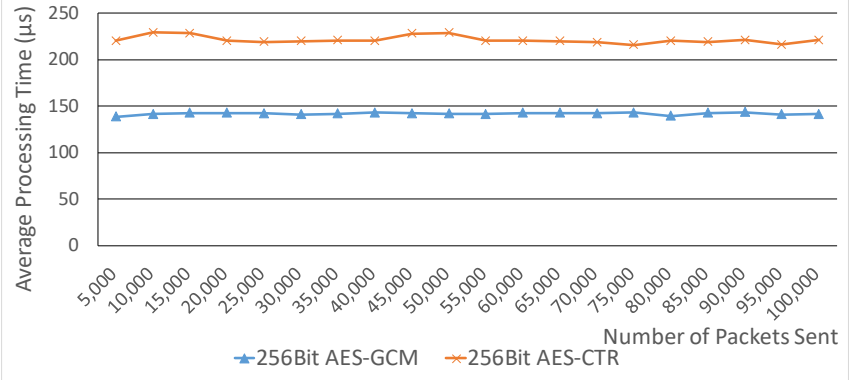
3. Measuring the throughput vs. APT for AES-GCM and AES-CTR encryption modes.

In the first test case, we measured the processing time of each router for 20 test cases, moving from 5,000 packets to 100,000 packets with a packet gap of 5,000 packets. Processing time was measured for tasks such as decryption and encryption during per-hop data encryption, serializing packet headers, de-serializing packet headers, and selecting routes.

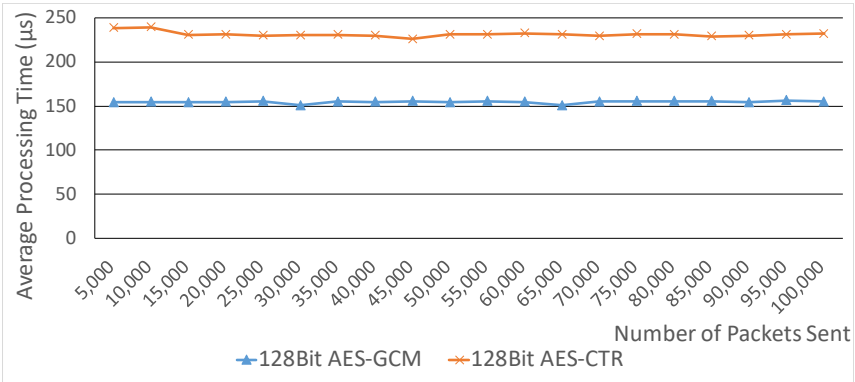
For the initial evaluation, the client was programmed to transmit two types of packets to the server in the topology shown in Figure 4-3(I). First, it transmitted entire-encrypted combined packets, which encrypted the header details together with the payload details. Secondly, it was



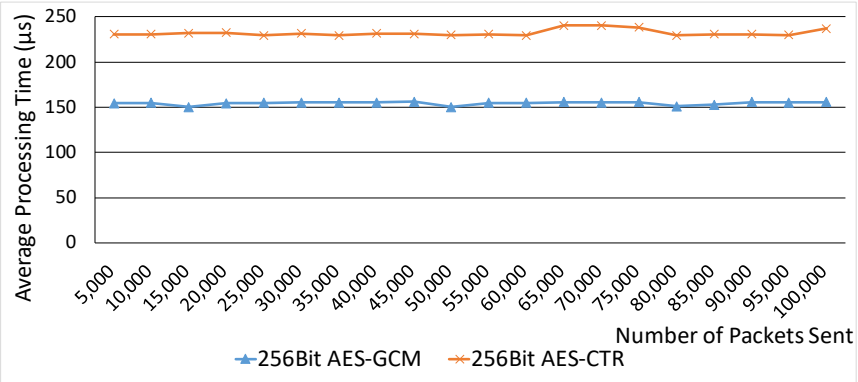
(I) 128-bit entire-encrypted packets



(II) 256-bit entire-encrypted packets



(III) 128-bit separately entire-encrypted packets



(IV) 256-bit separately entire-encrypted packets

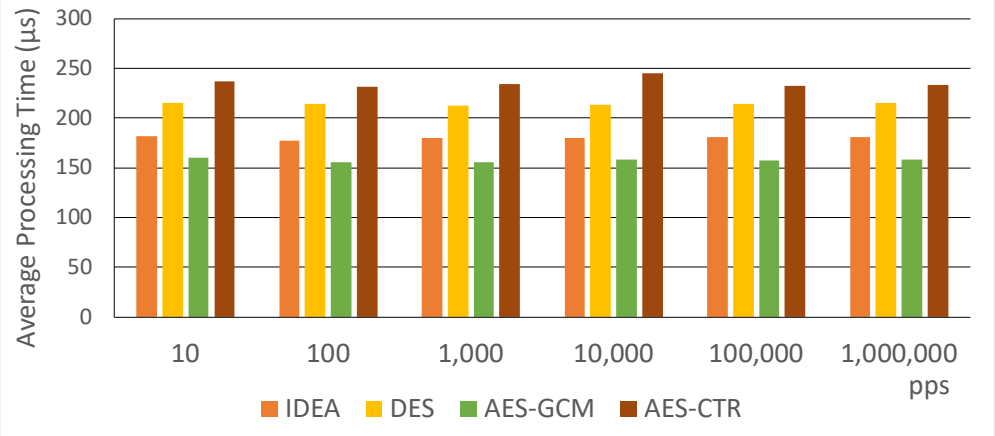
Figure 4-4 APT for AES-GCM vs. AES-CTR encrypted packets

Table 4-1 Overhead of average processing delays caused by the entire-packet encryption service: AES-GCM vs. AES-CTR

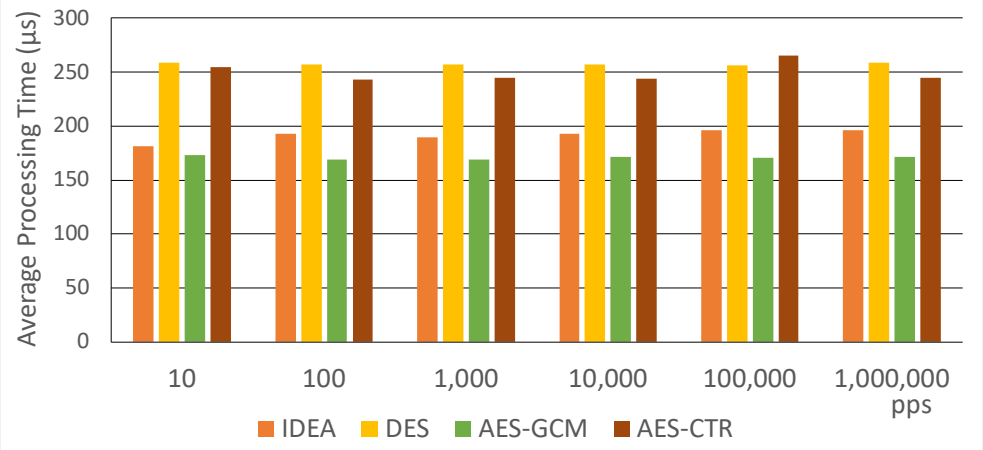
Key size	AES-GCM 128-bit	AES-CTR 128-bit	AES-GCM 256-bit	AES-CTR 256-bit
Combined encryption	141.8 μs	218.94 μs	141.92 μs	221.39 μs
Separate encryption	154.56 μs	231.3 μs	154.29 μs	232.2 μs

programmed to transmit separate entire-encrypted packets to the server, where the payload information and the header information were separately encrypted using two fields. The test cases were performed using 128-bit and 256-bit keyspaces.

In all the cases, the client was started five seconds after the simulation began. It was programmed to transmit a fixed number of packets at intervals of 0.02 s. The first five seconds were allocated to initializing, generating, and exchanging the PKI and AES keys; updating the key values in the routing tables, and propagating the routing tables of the underlying routing protocol. These processes were automatically executed by the underlying routing protocol



(I) APT for entire-encrypted packets



(II) APT for separately encrypted packets

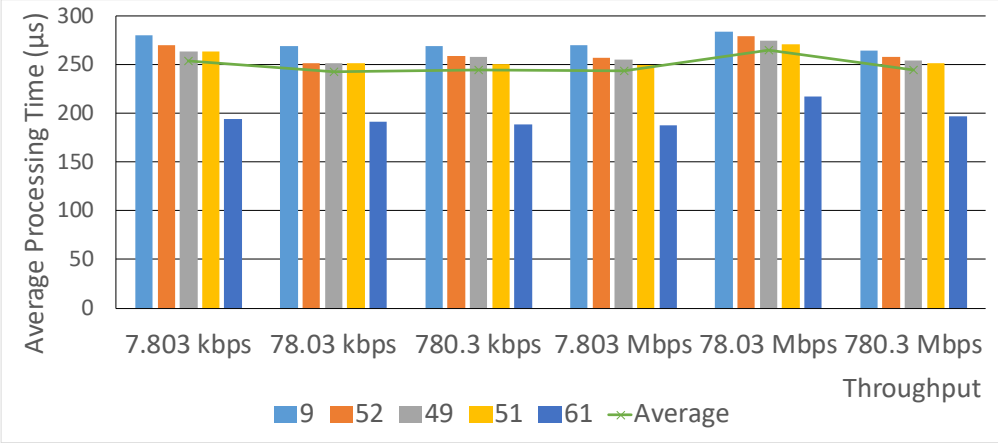
Figure 4-5 APT for encrypted packets

regardless of the network topology or number of routers in any attached topology. The APT consumptions were measured, and the results were plotted using graphs as illustrated in Figure 4-4. The test results are summarized in Table 4-1.

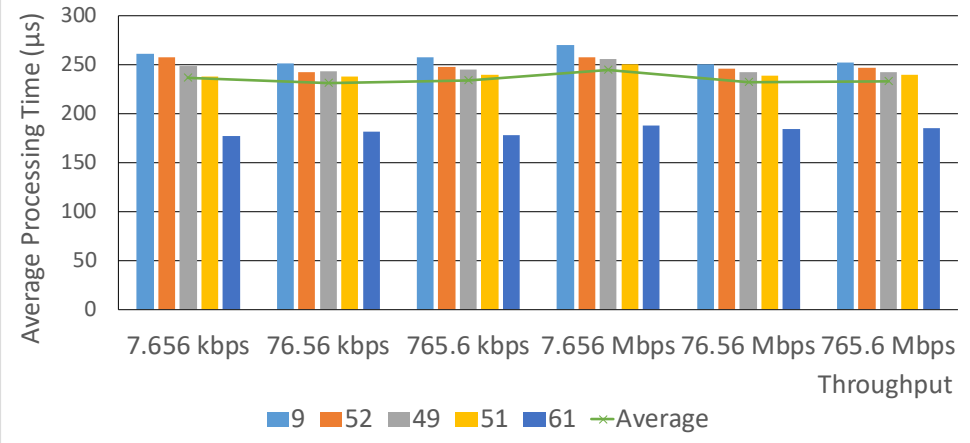
4.2.5.2 Evaluation of the throughput vs. average processing time

Further, the simulation was programmed to transmit the same two types of entire-encryption packets using different pps values. For this test case, the APTs for IDEA, DES, AES-GCM, and AES-CTR encryption algorithms were measured while increasing the pps 10 times in each case; starting with a value of 10 pps and ending with 1,000,000 using a 256-bit keyspace. The main objective of this test case was to measure APT of the proposed entire-packet encryption service for high-speed packet transmission and throughput during the packet transmission. The test results obtained for the APT values vs. the pps for the entire-encrypted combined packets are plotted in the graph shown in Figure 4-5(I). Furthermore, the test results for the separate entire-encrypted packets are plotted in Figure 4-5(II).

For the next test case, the APT for different throughputs was measured between each routers in the packet-propagation path for each encryption algorithm. The test results for the AES-CTR



(I) AES-CTR entire-encrypted packets



(II) AES-CTR separately entire-encrypted packets

Figure 4-6 Throughput vs. APT for AES-CTR encryption in each router

Table 4-2 Summary of evaluation results

	pps	10	100	1000	10000	100000	1000000
IDEA	Throughput	4.922 kbps	49.22 kbps	492.2 kbps	4.922 Mbps	49.22 Mbps	492.2 Mbps
	APT (μs)	181.84	177.26	180.05	179.55	180.88	181.26
IDEA separate	Throughput	5.147 kbps	51.47 kbps	514.7 kbps	5.147 Mbps	51.47 Mbps	514.7 Mbps
	APT (μs)	180.88	192.79	189.36	192.86	196.19	196.03
DES	Throughput	4.922 kbps	49.22 kbps	492.2 kbps	4.922 Mbps	49.22 Mbps	492.2 Mbps
	APT (μs)	214.76	213.87	212.31	213.52	214.36	214.94
DES separate	Throughput	5.078 kbps	50.78 kbps	507.8 kbps	5.078 Mbps	50.78 Mbps	507.8 Mbps
	APT (μs)	258.99	256.73	256.71	257.02	256.16	258.82
AES-GCM	Throughput	4.96 kbps	49.6 kbps	496 kbps	4.96 Mbps	49.6 Mbps	496 Mbps
	APT (μs)	160.46	155.47	155.72	157.87	157.71	158.14
AES-GCM separate	Throughput	5.225 kbps	52.25 kbps	522.5 kbps	5.225 Mbps	52.25 Mbps	522.5 Mbps
	APT (μs)	173.15	168.96	169.19	171.21	170.42	171.15
AES-CTR	Throughput	7.656 kbps	76.56 kbps	765.6 kbps	7.656 Mbps	76.56 Mbps	765.6 Mbps
	APT (μs)	236.74	231.71	233.80	244.59	232.55	233.46
AES-CTR separate	Throughput	7.803 kbps	78.03 kbps	780.3 kbps	7.803 Mbps	78.03 Mbps	780.3 Mbps
	APT (μs)	254.29	243.10	245.03	244.08	265.38	245.05

encryption algorithm with a 256-bit keyspace are plotted in the graph illustrated in Figure 4-6. The measured test results for the entire-encrypted combined packets are shown in Figure 4-6(I), and the results for the separate entire-encrypted packets are plotted in Figure 4-6(II). Both the above test results are summarized in Table 4-2. The throughput values for the two cases vary because the header sizes are different for in the same combined header structure than when encrypting the data and packet header separately.

Further, we measured the processing cost for the route-calculation process without any encryption to measure the pure processing cost for packet routing. The test results are summarized in Table 4-3. According to the test results for the given topology, the average

Table 4-3 Overhead of processing delays caused by entire-packet encryption service over packet routing

Trace type	IDEA	IDEA_ Sep	DES	DES_ Sep	AES-GCM	AES-GCM_ Sep	AES-CTR	AES-CTR_ Sep
APT (μs)	180.14	191.35	213.96	257.41	157.56	170.68	235.48	249.49
Diff. over pkt routing (μs)	106.03	117.25	139.85	183.3	83.45	96.57	161.37	175.38
Percentage	58.86%	61.27%	65.36%	71.21%	52.96%	56.58%	68.53%	70.3%

processing cost for packet routing was 74.11 μs. Therefore, all the above results include 74.11 μs of routing cost, which is consumed by the underlying routing protocol.

4.2.5.3 Results discussion

In the graphs illustrated in Figure 4-6(I) and Figure 4-6(II) regarding test case 3, only the AES-CTR encryption mode results were plotted to reduce a large number of graphs. The complete test results for all the encryption algorithms: IDEA, DES, and AES-GCM are summarized in Table 4-3. The differences and percentage increments of the entire-packet encryption service versus plain packet routing are compared in Table 4-3. The test results in Table 4-2 and Table 4-3 were measured with a 256-bit keyspace for all the encryption algorithms.

According to Table 4-2, encryption data lengths vary according to the encryption algorithm; therefore, throughput values for a constant pps value also change according to the encryption algorithm. Furthermore, the average processing time for the entire-encrypted combined packets is lower than that for the separate entire-encrypted packets because separate entire-encrypted packets must perform the encryption and decryption processes twice separately for the data and header. However, any of the two encryption methods can be selected and used depending on the user or application requirements.

Additionally, the proposed separate entire-encrypted header structures (Figure 4-2(IV) and (VI)) can also be used in a scenario where the payload is end-to-end encrypted, and the header is encrypted with per-hop data encryption to reduce the total encryption and decryption delays. The proposed per-hop data-encryption service is highly flexible and can be used with any of the above-mentioned combinations among links according to the user, application, or service requirements.

The proposed service displayed average APT of 141.8 and 141.92 μs for AES-GCM mode, and 154.56 and 154.29 μs for AES-CTR mode of packet-propagation delays per hop for 128-bit and 256-bit key lengths, respectively, while transmitting the entire-encrypted combined packets. These processing times increased to 218.94 and 221.39 μs for AES-GCM mode, and 231.3 and 232.2 μs for AES-CTR mode for 128-bit and 256-bit keyspaces, respectively, while transmitting separate entire-encrypted packets.

According to the International Telegraph Union Telecommunication Standardization Sector (ITU-T) recommendation G.114 [118], if the total transmission delay over networks is maintained at less than 150 ms, most applications will not be affected significantly. In the test results, the highest processing delay of 265 μs (0.265 ms) was caused by the AES-CTR mode while transmitting the separate entire-encrypted packets with keyspace of 256-bit. Therefore, a packet should pass over 566 hops on average, without considering the link delays to generate a delay greater than 150 ms.

Even with the link costs, the encryption delay caused by the proposed method is small and tolerable over public networks because link delays are of the order of milliseconds whereas processing delays of microseconds. Therefore, we conclude that even with the link and other packet-propagation delays, the proposed entire-packet encryption service can provide complete

security to the packets and their headers without causing noticeable end-to-end delays over public networks.

Moreover, the existing Internet infrastructure does not require a total change to use the proposed method. Adding routers to networks according to the network size is sufficient to deploy all the proposed services. Additionally, the proposed service can be provided easily via extending the existing routing protocols. The Linux-based is used as the cryptographic library for all cryptographic implementations [117]. Therefore, the proposed method operates much more accurately as a real-world implementation. Furthermore, this enables an easier migration of the total infrastructure to the real-world environment.

4.2.6 Conclusion

Preserving CIA during data transmission is essential while communicating through public networks. Conventional methods either lack securing the sensitive details in the packet headers or expose those details at exit points. In this study, we introduced an entire-packet encryption service that employs a per-hop data encryption method to provide a secure packet-encryption service over public networks. It allows users or applications to dynamically select encryption algorithms together with different keyspaces between the hops. Generally, it secures the packet payloads together with the IP headers, and still allows the packet to be IP routable.

The main limitation of the model is that there must be content-based routers at least at the edge of the networks of the communication parties. Adding more content-based routers to the networks provides more security to the communication. The content-based routers interfaces must also be configured with appropriate keyspaces and encryption algorithms according to the security requirements of the connected networks.

The test results demonstrated that the proposed method was flexible to use with encryption algorithms such as IDEA, DES, AES-GCM, and AES-CTR. AES-CTR mode is a proven method to be used with Gbps-rate packet streams while providing authentication to the packets and can overcome lost or reordered packet scenarios. According to the test results, the proposed entire-packet encryption method causes delays considerably shorter than 150 ms. Therefore, we can conclude that the proposed method can be used for most of the general applications.

4.3 Neighbor data retransmission service

4.3.1 Motivation

Over the past few decades, many studies have been conducted regarding the data retransmission methods and analyzing various data retransmission methods [119] [120] [121] [122]. On the other hand, the Internet is used by a majority of the worldwide population as their main communication media. It is a well-known fact that the Internet comprises of millions of public networks. In fact, public networks are less secure and highly congested than private networks. Yet, it is widely used to transmit sensitive data belonging to millions of users across the world.

The Internet's inherent problem of higher congestion rates due to the massive traffic it tolerates, allows the packets to be frequently retransmitted or dropped in-between the data transmission.

Almost all of the research conducted in the past few decades on data retransmission is mainly focused on improving the existing retransmission methods in existing protocols [119] [120] [121] [122]. Many researches use the conventional ARQ based data retransmission methods, which retransmit the data all the way from the sending end. This introduces an higher retransmission delays as it should again transmit the data from the original sender to the receiver, through the same intermediate networks which still can be congested from other ends. Systems which depend on real-time data such as sensor network based solutions will experience difficulties using the conventional retransmission methods due to the higher retransmission delays.

When users or applications transmit data through the networks, those packets may be corrupted for facts such as; due to the noise or due to the collisions in the transmission lines. In a reliable data transmission protocol, all the data packets will be acknowledged upon the retrieval by the receiver. When a packet is not acknowledged after a certain feasible time period, the protocol will define it as a transmission failure and it will retransmit the packet, or the receiver will send a retransmission request. TCP protocol uses the ARQ method to provide data retransmission [123]. In the existing data retransmission methods, when a device identifies an error in a data bit of a transmitted packet, it will send a retransmission request to the sending node or the data will be automatically retransmitted once the RTO is reached at the sending end. In this traditional way, the packets are always retransmitted from the original sender.

Therefore, the retransmission packets should again be transmitted all the way through the network from the sender. The main disadvantage in this method is the retransmission delay occurred while sending again the packets all the way from the original sender. Obviously, there will be several hops in-between the sender and the receiver. In the networks, there are link delays which cause by the transmission media and the distance. When the retransmitted packet is sending all the way from the originator, the final received packet will be affected by this link delay twice plus the RTO delay as the packet is retransmitted from the sender. If the packet is from a packet stream or where the transmission order is important as in video/audio stream data or sensor network data, this retransmission delay will result in drastic issues in data transmission, where in most of these cases the conventional data retransmission methods cannot be used altogether.

This study proposes a novel neighbor data retransmission service that is capable of providing availability and integrity using the content-based routers. The implemented neighbor data retransmission service consists of the following: (1) buffering the service required packets in routers, (2) managing the packet buffers according to the user or application requirements, and (3) retransmitting corrupted data from the neighbor routers.

4.3.2 Neighbor data retransmission service implementation

In the traditional TCP retransmission method, it is retransmitting packets mainly after the retransmission timer is expired and the sender initiates the retransmission. Basically TCP

retransmission time out (RTO) value is calculated using the round-trip time (RTT) value. Until the RTT is defined, the default RTO value is 2.5 seconds [82]. Therefore, if the receiver is many hops away from the sender, the RTO delay will mainly be affected by the link delays between the hops. According to the TCP retransmission timer [82], the initial RTO value is calculated using the following Formula (4-1).

$$RTO = SRTT + \max(G, K * RTTVAR) \quad (4\text{-}1)$$

In the above formula, SRTT is smoothed round-trip time, G is clock granularity value, and RTTVAR is round-trip time variation. In this scenario, as we know the link delays, we assume that the first RTT measurement is made. According to [82], when the first RTT measurement is made the values were set as follow.

- SRTT = R
- RTTVAR = R/2
- RTO = SRTT + max (G, K*RTTVAR) , where K = 4

In the testbed topology, in order to reach the destination, a packet should pass through 9 links and 8 routers. Through this path the SRTT value is $(9 * 0.02) * 2$ s which is 0.36 s. Therefore, the value of RTTVAR is 0.18s. For the TCP sessions, value is G is 0.1s, and the K is equal to 4. Finally, we can calculate the value of RTO as 1.08. If P denotes the packet processing delay taken by a router and D_L denotes the delay of the link, the initial retransmission delay of the traditional TCP (D_{TCP}) for this scenario can be calculated by Formula (4-2).

$$D_{TCP} = RTO + (D_L * 9) + (P * 8) \quad (4\text{-}2)$$

If we consider the situation while using the neighbor data retransmission method in the same topology under the same conditions, we can calculate the neighbor data retransmission delay (DNBR) using the formula (4-3).

$$D_{TCP} = 1.26 + 8P \quad (4\text{-}3)$$

$$D_{NBR} = (D_L * (9 + 2)) + (P * 10) \quad (4\text{-}4)$$

In neighbor data retransmission, there is no waiting for a timer to expire. Therefore, the 1.08s RTO value is emitted. According to Formula (4-3) only two more link delays and two additional processing delays at routers are affected. In this method, when a checksum error is detected in a certain packet, that packet will be discarded, and it will send a retransmission request from the previous router. Once that request receives to the previous router, it will get the corresponding packet from the packet buffer, and it will send it back to the requesting router. Therefore, there will be two additional link delays effected to the retransmission time as one for transmitting the retransmission request packet and the second one is the link delay taken to transmit the retransmit packet withdrawn from the retransmission buffer. Moreover, as the link delay for this process is also 0.02 s, Formula (4-4) can be solved to Formula (4-5).

$$D_{NBR} = 1.22 + 10P \quad (4\text{-}5)$$

In the real world scenario, the processing time taken by the router to process a packet, denoted

by P in the above equations, is measured in microseconds (μs). Therefore, the main delay effecting in a packet transmission is the link delays in-between the hops. Furthermore, in neighbor data retransmission method, the P value will be slightly higher as it requires to buffer the service required packets before forwarding them.

The neighbor data retransmission method buffers all the required packets routed through a router. Whenever a bit error is detected, the buffered packet can be requested and retransmitted from the neighbor routers. This method reduces the higher data retransmission delays caused due to requesting the packets from the sender in the conventional retransmission methods. The implementation uses the Adler-32 algorithm to calculate the checksum values of the packets in order to check the data integrity and to identify the bit errors.

Before storing or forwarding a packet, the router will first analyze the checksum value of the packet. If the packet checksum matches with the calculated checksum of the packet content, the router will store the packet in its buffer, and it will forward the packet to its destination through the most feasible path. If it detects a checksum error, the router will discard the packet, and it will request the packet via sending a neighbor data retransmission request to the previous hop router.

Once a router receives a neighbor data retransmission request, it will check whether the packet is available in its packet buffer. If the packet is available in the buffer, it will send the packet to the requesting router. Moreover, if the packet is unavailable in its packet buffer, it will send retransmission requests to its neighbor routers. This method will mainly reduce two types of retransmission delays over conventional retransmission methods: (1) delay to propogate the retransmission request to the original sender, and (2) delay to retransmit the packet from the original sender to the destination.

4.3.3 Retransmission packet buffer

The packet buffer is the heart of the neighbor data retransmission service. In order to increase the efficiency and to reduce the data retransmission delays, routers should buffer a feasible amount of packets in their buffers throughout a reasonable time period. This process will add a processing overhead to the device, and a processing delay to the service required packets as it will take some time to buffer the packets in the memory before forwarding them. Moreover, the specific hardware and software architectures of the content-based routers will be used to further reduce these delays.

If a router with a large number of interfaces try to buffer all the packets it routes, then soon it will run out of memory. Therefore, the routing protocol needs to identify the most critical and service required packets and then it should only buffer these packets. Moreover, the protocol can be programmed to buffer the packets considering the number of packets or buffering the packets for a considerable time period as the buffer limits. The buffering method can be selected depending on the device limitations or the application requirements. The proposed service is recommended for services which require reliable data communication together with continues packet streams, transferring small sized data packets such as sensor networks which need to

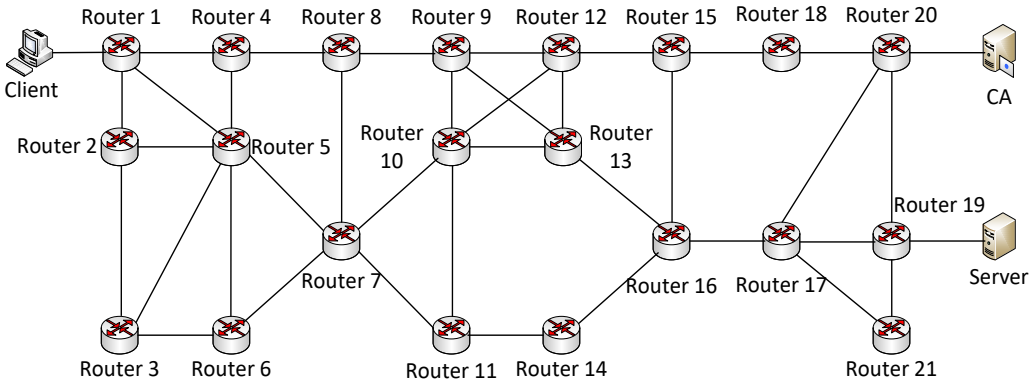


Figure 4-7 Simulation topology

transfer a large number of small-sized sensor data packets with a minimum delay while providing reliability.

4.3.4 Topology implementation and simulation

The main limitation of the proposed service is the delays caused by the packet buffering methods and the additional processing, power and memory overheads required by neighbor data retransmission buffers. The experiment is executed in the topology illustrated in Figure 4-7. As illustrated in Figure 4-7, all links in the topology were configured with a throughput of 5 Mbps and a 2 ms delay. While acquiring the test results, we obtained the total processing time consumed by the processor in microseconds when executing the particular functions using the real-time stamp counter of the CPU. In this topology, all the routers were configured as content-based routers. The client is formatted to send packets destined to the server.

When the number of packets buffered in the router is increasing, the processing time to process the packets will gradually increase. This is mainly due to the processing and memory requirements for the buffer management functionalities. In order to evaluate this scenario, we have programmed the routers in the topology, denoted by Figure 4-7, with two test cases as with limited packet buffers and then with unlimited packet buffers. The end-to-end delay of the packet delivery path is 0.18 s (0.02 * 9). The packets were sent at a rate of 50 packets per second. By considering these values, in the limited packet buffer scenario, the routers were programmed with a queue size of 500 packets and the maximum packet queue timeout of 0.2 seconds.

In the unlimited queue scenario, the testbed is formatted with both unlimited queue size and unlimited packet expiration time. The simulation was executed for 10 cases starting from 5,000 packets and ending with 50,000 packets with a packet gap of 5,000 packets.

4.3.5 Evaluations

The processing time of the eight routers was measured for each case, and the average value is used for the plotting. The test results were graphed as illustrated in Figure 4-8. According to Figure 4-8, we can clearly conclude that the queue size is directly affected to the router packet

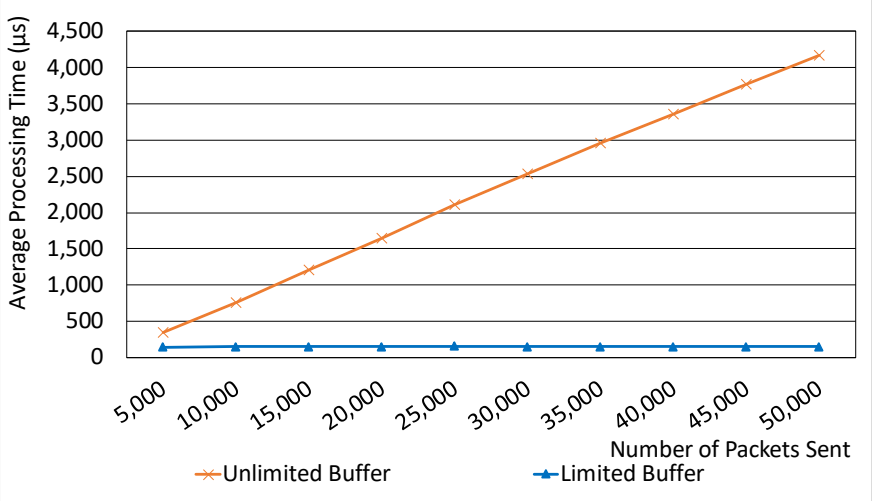


Figure 4-8 APT values for unlimited packet buffer and limited packet buffer

processing time. In the unlimited buffer scenario, when the number of packets is increasing, the processing time is also increased gradually. In the limited buffer scenario, the average processing time was 147.3 μs. In the unlimited buffer scenario, average processing time was 2,285.86 μs for the measured packet range. Moreover, the processing time is increasing with the number of packets buffered. We can conclude from these test results that the routers should be programmed with a feasible queue size and packet expiration time depending on the device limits or application requirements.

As the next evaluation, the topology illustrated in Figure 4-7 was programmed to send packets with limited packet buffering and without packet buffering. All the routers were programmed with the same above used limited queue parameters for the buffering enabled scenario. The processing time taken by each router was measured through the packet delivery path. The client was started 5 s after the start of the simulation. Then, the client sends a fixed number of packets at an

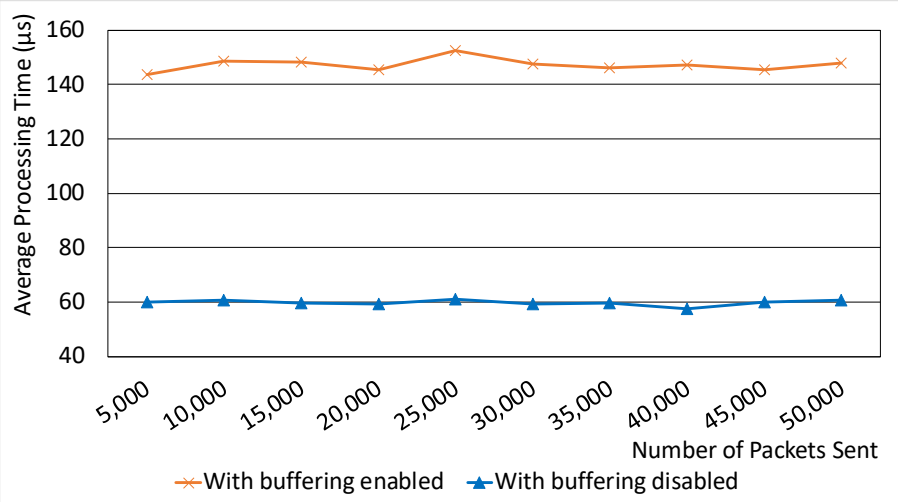


Figure 4-9 Processing time for routing packets with limited packet buffering and without packet buffering

interval of 0.2 s. We measured the processing time taken by each router for 10 test cases starting from 5,000 packets to 50,000 packets with a packet gap of 5,000 packets, and we used average processing time (APT) for the comparison. The obtained test results are plotted in the graph shown in Figure 4-9.

By analyzing the graph in Figure 4-9, for the normal packet processing routers have consumed an APT value of 59.75 μs. When the routers were buffering the packets for the neighbor data retransmission service, the average processing time was 147.3 μs. Therefore, it will cost an additional 87.55 μs of per router processing time per packet in order to provide the neighbor data retransmission service. This will be the average processing overhead per packet for a router.

4.3.6 Results discussion

Now using these test results, we can approximately calculate the retransmission times using the above-discussed Formula (4-3) and Formula (4-5). By using the 59.75 μs as the value of P for the Formula (4-3), we can calculate the D_{TCP} value as 1.260478s. By using the 147.3 μs as the value of P for the Formula (4-5), we can calculate the D_{PRE} value as 0.221473s. According to these results, the conventional TCP data retransmission service will generally cost additional 1.039005s for an initial data retransmission compared to the proposed neighbor data retransmission service.

For the final evaluation, we implemented the prototype method to simulate the traditional ARQ based retransmission method in ns-3. The topology was configured to compare the AQR based retransmission method used in TCP with the proposed neighbor retransmission method. In order to perform data retransmission, data checksum errors should be present. In order to overcome this, the topology in Figure 4-7 was configured to invent static packet errors or header checksum errors in routers among the data delivery path. In this test scenario, when a packet is transmitting through the network and once it received at the error inventing router, the router will buffer the original packet and will send the packet changing the checksum value to the next router. Therefore, the next hop router will detect the checksum error and send a retransmission request.

In the ARQ based method, the router sends the retransmission request back to the sender wherein the neighbor retransmission method the router will send the retransmission request to the neighbor router. The errors were separately invented in the routers; Router 4, Router 8, Router 9, Router 12, Router 15, Router 18 and Router 19 for individual scenarios. In each scenario, the packet processing delay at each router for each packet was measured, and the end-to-end packet delays for each case was calculated. For each scenario, the testbed was programmed to send 10,000 packets from client to server. The implemented ARQ based prototype method was not configured with RTO timers and it was just configured to send the retransmission request to the sender upon detecting the checksum error.

The total average end-to-end delays were measured for the two cases and are plotted in the graph illustrated in Figure 4-10. According to Figure 4-10, the proposed neighbor data retransmission method had the same retransmission delay regardless of the location of the error detected router in the testbed topology. This is because the packet will be always retransmitted

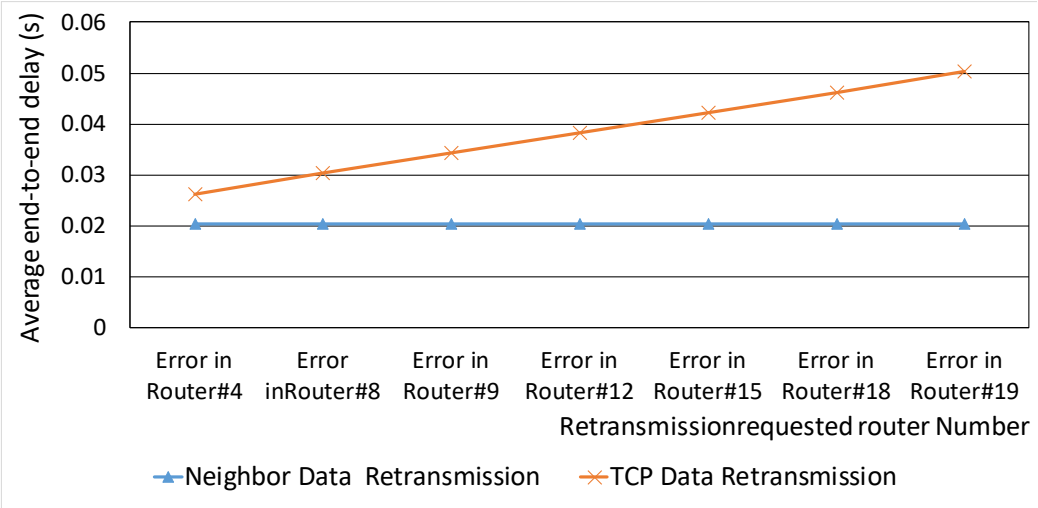


Figure 4-10 End-to-end packet delay in neighbor retransmission method vs. ARQ based TCP retransmission method

from the neighbor router if the packet is available. In the conventional ARQ method, the retransmission delay is increased sequentially while the distance/hop count to the retransmission requesting router is increased from the sender. This is obvious as the affected link delays are increasing while the distance from the sender to the error detecting node is increasing.

If the evaluation considered the RTO calculation for the ARQ method, the end-to-end delay would be further increased. This is because the RTO values are calculated in seconds, and the link delays and the router processing delays are calculated respectively in milliseconds and micro seconds. The graphs illustrated in Figure 4-8 and Figure 4-10 is similar, yet the test results and the evaluated perspectives in the two test cases are totally different. According to the results of Figure 4-10, when considering the end-to-end delay, the proposed neighbor data retransmission method has a higher advantage over the conventional ARQ based retransmission method.

4.3.7 Conclusion

In this section, we introduced a neighbor data retransmission service that can be used to retransmit corrupted data over public networks. The proposed service is flexible enough to set the retransmission parameters such as; expiration times and buffer sizes according to the device limitations or application requirements. Moreover, the test results showed that the end-to-end retransmission delay of the proposed method is 80.43% reduced than the delay caused by the conventional ARQ method. In order to get the maximum efficiency out of the proposed method, the routers must be configured with reasonable buffer lengths, and buffer timeout values depending on the available router resources and the packet serve ratios of the router.

The efficiency of the proposed method is increasing while the number of hops is increasing from the error detected node to the source. However, those flexibilities come with the cost of additional 87.55 μs of per router processing time per packet buffering overhead in the proposed method. This processing power and memory constraints can be further reduced by using the

appropriate buffer size and the packet expiration time together with hardware routers. The proposed retransmission method will be a feasible method for packet streams with less packet sizes and time-sensitive data streams where higher retransmission delays of the conventional methods are higher to construct the packet streams such as sensor network data streams.

Nowadays, sensor networks are playing a massive role in most of the day-to-day automation systems. These networks contain thousands of sensors that will transmit the sensor data continuously through the networks. The accuracy and the effectiveness of these systems totally depend on the amount of the sequentially received data packets. Therefore, most of the sensor data should be transmitted with a minimum delay. For these types of networks, neighbor data retransmission service can provide an effective and efficient solution via drastically reducing the data retransmission delays.

4.4 Fast and secure traceability service

4.4.1 Motivation

Network attacks, such as phishing attacks [124] and DDoS attacks [125], are a major threat to users, networks, and network resources. Therefore, it is critical to locate and track the adversary through a network attack prevention and mitigation process. After tracing an attacker, he/she can be blocked from accessing the network and resources, thereby preventing further attack propagation. Traceroute is a widely used traceability method [126] that employs the Internet communication message protocol (ICMP) and user datagram protocol (UDP) to determine the path a packet has traveled through networks. However, some intermediate network devices, such as firewalls and proxy servers, drop the ICMP-based traceroute packets to ensure the security of the network. Therefore, it is impossible to obtain the complete trace using ICMP packets [127]. Moreover, this service cannot verify the requested node where the ping packets are dropped because there can be ICMP attacks on the servers. ICMP packets are not a reliable method of obtaining the packet trace because the packet delivery path can differ while the ICMP and normal packets are being transmitted depending on link-states. Moreover, ICMP packets are not designed to carry other payloads while they transfer through the networks. To address these issues, this section propose a traceability service that can be used to trace any packet type as they propagate through public networks.

To address this issue, this section propose a novel traceability packet header structure that can be attached to both transmission control protocol (TCP), and UDP packets upon request to provide traceability service. The trace details can be selected as the router ID, IP addresses, or MAC addresses of the router interfaces through which the packet has arrived and forwarded in. Moreover, the proposed traceability method use per-hop data encryption service to secure attached traceability data protecting the sensitive trace date from any unauthorized access or modification.

The proposed traceability service includes the following: (i) providing of link-state routing, (ii) managing of key tables, (iii) providing of per-hop data encryption, (iv) working with a certification

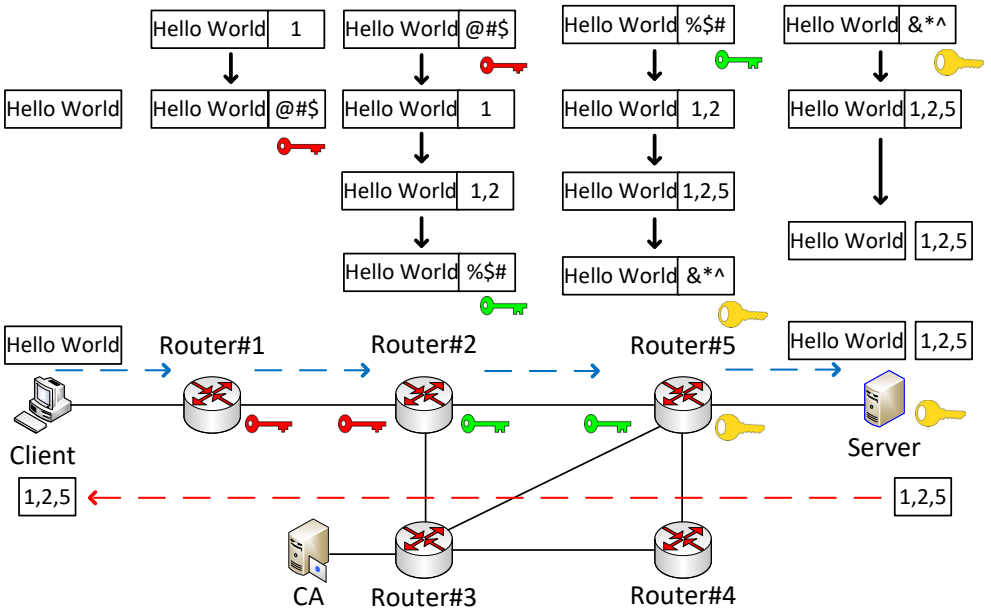


Figure 4-11 Proposed packet traceability service

authority (CA)-like infrastructure, and (v) providing of a secure and flexible traceability service over public networks. The operation of this traceability service is illustrated in Figure 4-11. As shown in the figure, all routers have a unique advanced encryption standard (AES) encryption keys to be shared with their connecting neighbor interfaces. For the key sharing process, the routers use the CA architecture. The key sizes combined with the encryption algorithms can vary between hops because they are unique sharing-independent keys. To provide the traceability service, we implemented a UDP-based custom packet header, which is detailed in section 4-4-2. This custom header is used to attach the traceability data. As illustrated in Figure 4-11, this custom header can be attached to existing UDP or TCP-based packets traversing the Internet.

When the client must send a traceability-required packet to the server, as depicted in the Figure 4-11, the client sends the traceability required packet along with the UDP-based packet with the payload 'Hello World.' The traceability data are attached to the packets while the packets are propagating to the server, and finally, the server sends back the complete trace back to the server. Initially, when Router 1 retrieves the packet, the latter identifies the packet as a traceability-required packet and attaches the custom header by appending its ID. (Here, the ID is the router identification number, which is 1 for Router 1.) Then, Router 1 searches for the best route to forward the packet from its routing table and encrypts the traceability header using the shared AES key with the next hop router according to the selected route details. The per-hop data encryption method [128] is used to encrypt the traceability data. Router 1 then forwards the packet to the selected next-hop Router 2.

Upon retrieval of the packet, Router 2 decrypts the traceability packet header using the retrieve interface details and obtains the plain traceability header. It then appends its traceability details to the custom header and performs the per-hop encryption according to the route to the packet destination; it then forwards the packet accordingly. Finally, the server retrieves the UDP payload together with the attached traceability data. Routers can attach their respective IDs, IP

addresses of the interfaces, or MAC addresses of the interfaces through which the packet was retrieved and forwarded.

After retrieving the traceability data, the server returns the trace data to the sender. While providing secure traceability, this reply packet is returned to the sender in encrypted form using the sender's public key. Therefore, the reply packet can be read only by the sender, who is the only one with the private key required for the public key infrastructure (PKI) decryption process for retrieving the traceability data.

4.4.2 Fast and secure traceability service implementation

To provide a secure per-hop encrypted traceability service, the proposed method uses the underlying routing protocol and per-hop data encryption method. The method can verify the service-requesting users or the services themselves using the CA infrastructure. The main advantage of the proposed traceability packets is that they can be attached to the data, which must be transmitted by the user or application with the trace packet. The traceability data of the packets is updated in real time as they propagate through public networks. This service can provide the traceability details primarily by using the following three pieces of information:

- Router identifiers (IDs)
- IP addresses of the received and forwarding interfaces
- MAC addresses of the received and forwarding interfaces

The traceability options can be chosen by the application or user that requires the traceability service. While a required trace packet is transmitted through the network, the routers add the incoming and forwarding interface or router details as required by the user or application. Furthermore, the protocol is sufficiently flexible to provide the traceability information as either a plain trace or an encrypted trace. If the user or application selects the plain trace, the protocol transfers the trace information attached with the packets in plain text format; if the secure transfer is selected, the protocol encrypts the trace using the AES-GCM or AES-CTR modes appropriately with per-hop encryption.

If the user or application selects router ID as the traceability option, when the packet arrives at a certain router, it attaches its router ID to the traceability header. If the selection is the interface IP address, the routers attach both IP addresses of the received and forwarded interfaces, which the packet received and should be forwarded. Similarly, if the selection is the interface MAC address, the routers attach both MAC addresses of the received and forwarded interfaces of the router from which the packet arrived and should be forwarded. The outputs of the implementation during receipt of the traceability packet are illustrated in Figure 4-12.

Figure 4-12(I) shows the output or the trace data received when the trace option is configured as the router ID. The output clearly shows all the IDs of the routers through which the packet transmitted. Figure 4-12(II) illustrates the output when the topology is configured to provide the IP addresses of all the router interfaces the packet received and forwarded. Finally, Figure 4-

```
Received Trace:
1 +2065599.0ns
4 +2094399.0ns
8 +2123200.0ns
9 +2152000.0ns
13 +2180800.0ns
16 +2211199.0ns
17 +2241599.0ns
19 +2271999.0ns
Server +2302399.0ns
```

(I) Receiving router ID as the trace

```
Received Trace: 11.0.0.1 11.0.0.2
172.16.2.1 +2075199.0ns
172.16.2.2 172.16.12.2 +2132799.0ns
172.16.12.1 172.16.14.1 +2195199.0ns
172.16.14.2 172.16.26.2 +2259200.0ns
172.16.26.1 172.16.27.2 +2323199.0ns
172.16.27.1 172.16.29.1 +2387199.0ns
172.16.29.2 172.16.34.2 +2451200.0ns
172.16.34.1 211.21.211.1 +2515199.0ns
211.21.211.2 +2580800.0ns
```

(II) Receiving IP addresses as the trace

```
Received Trace:02-06-00:00:00:00:00:47
02-06-00:00:00:00:00:48 02-06-00:00:00:00:00:03 +2099199.0ns
02-06-00:00:00:00:00:04 02-06-00:00:00:00:00:18 +2287998.0ns
02-06-00:00:00:00:00:17 02-06-00:00:00:00:00:1b +2304000.0ns
02-06-00:00:00:00:00:1c 02-06-00:00:00:00:00:34 +2406400.0ns
02-06-00:00:00:00:00:33 02-06-00:00:00:00:00:36 +2508800.0ns
02-06-00:00:00:00:00:35 02-06-00:00:00:00:00:39 +2611200.0ns
02-06-00:00:00:00:00:3a 02-06-00:00:00:00:00:44 +2713599.0ns
02-06-00:00:00:00:00:43 02-06-00:00:00:00:00:49 +2815999.0ns
02-06-00:00:00:00:00:52 +2918399.0ns
```

(III) Receiving MAC addresses as the trace

Figure 4-12 Outputs of trace receipt

12(III) depicts the output when the trace option is programmed as the MAC addresses of all the router interfaces the packet received and forwarded. As shown in Figure 4-12, all the traceability details are attached with the per-hop transmit delay consumed for each hop. From these values, we can calculate the RTT values as given in the traceroute output.

When the topology is programmed to provide the encrypted traceability service, all the above details are encrypted using the per-hop data encryption method. Therefore, the transmitted trace

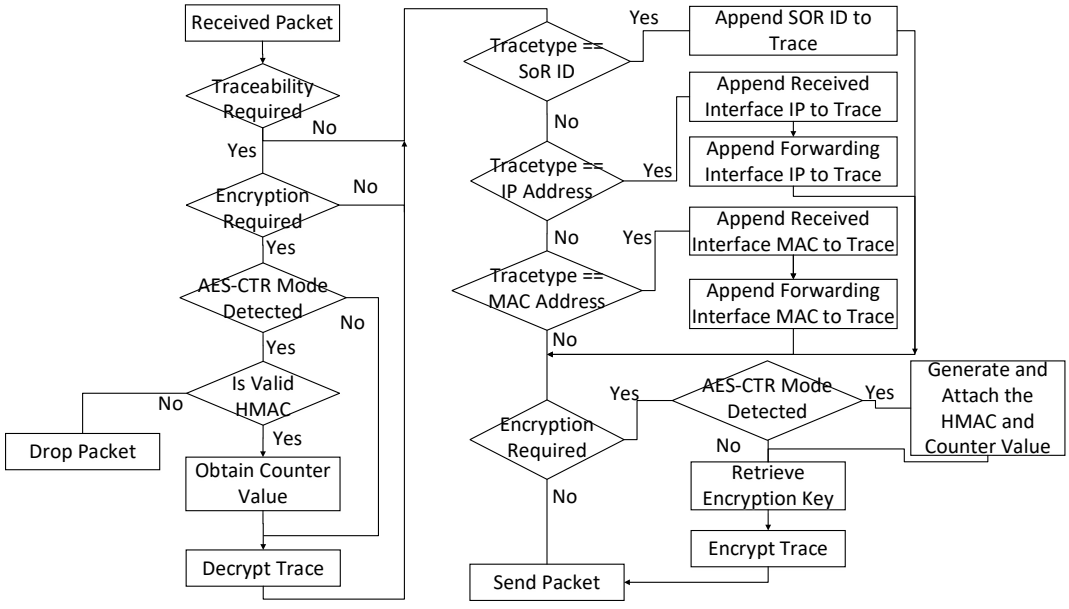


Figure 4-13 Traceability process flowchart

information is unreadable in the between the state of the transmission. From the received trace information, the receiving user or application can read and identify the details of the routers or interfaces that the packet has transferred through the network. This process is illustrated in the flowchart shown in Figure 4-13.

If the trace details are encrypted, the encrypted trace packets are decrypted using the appropriate AES mode before appending the trace details in every router, which is required by the per-hop data encryption method. After appending the traceability information, the router searches and obtains the most feasible route that the packet should be forwarded through from its routing table. Then, it obtains the next hop router details from that route and retrieves the encryption key for that router from its key table.

Finally, the router encrypts the packet using the obtained encryption key and forwards the packet to the corresponding router. Only the destined router can decrypt the trace because the destined router alone shares the symmetric AES encryption key, which is used to encrypt the packet, with the router that encrypted the packet. Additional HMACSHA validation is performed when the AES-CTR mode is used before the decryption process. The counter value together with an HMAC-SHA to authenticate the counter value is transmitted attached to each packet using the header shown in Figure 4-14(II). If the HMAC validation fails for a particular received packet, the packet is dropped and will not proceed with the decryption, as the received counter value is not authenticated. When using the AES-CTR mode for encryption, a new counter value is generated together with the HMAC value for the new counter value using the corresponding AES encryption key. Then both the counter and HMAC values are attached to the encrypted trace packet before sending to the next hop router.

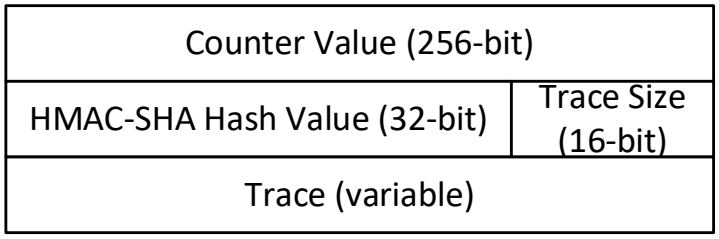
4.4.3 Packet traceability packet header structures

To provide this packet traceability service, the per-hop data encryption protocol uses its own header structure. To attach and update the traceability details, the packet must be decrypted and encrypted by the routers at each hop or interface using the per-hop encryption method. If we attempt to divide each trace information piece into separate fields, each of them is required to separately decrypt and encrypt. Because each encryption and decryption process consumes considerable processing time, it results in the addition of a larger additional end-to-end delay as encryption overhead to the packet delivery.

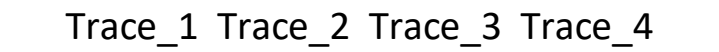
Moreover, as we perform per-hop encryption, all secured fields should be decrypted and encrypted at each hop, making it the main drawback in the per-hop data encryption method. The main problem while providing traceability is that we cannot define the number of hosts that a particular packet should travel to arrive at its destination. This totally depends on the network architecture and locations of the sender and receiver in that network topology, as well as on the packet delivery path selected by the routing protocol at the given moment. Therefore, we cannot confirm the number of entries that should be contained in the traceability header.

Two methods can be used for adding the traceability information to a packet. The most basic method employs an array structure to append the trace information. However, the main problem

(I) AES-GCM traceability header



(II) AES-CTR traceability header



(III) Traceability field structure

Figure 4-14 Packet header structured used for traceability service

is that we must define a fixed size as the size of the array before using the array structure. As a solution, we can use a packet structure with a maximum number of elements it can accompany. This implementation uses all the space of a packet, which would often otherwise be wasted, whenever the trace is less than the total length. This makes it an inefficient approach. Moreover, if we use an array structure, the implementation cannot attach the traceability details to existing packets.

The second method employs a vector that can have a dynamic size, making it much more efficient and effective than using an array. As mentioned earlier, in the per-hop data encryption method, we must decrypt and encrypt the data at each hop. Consequently, if we use an array or a vector, every element in them must be separately decrypted and encrypted at every hop, resulting in a higher decrypt and encrypt processing delay. Furthermore, the additional field length is important when considering the TCP segmentation size because the additional length may cause further segmentation of the packet. Therefore, a novel, feasible, flexible method is proposed and used in the implementation. The header structures and its field structure used for the traceability service are illustrated in Figure 4-14. These header structures are suggested for overcoming all of the drawbacks mentioned above.

Figure 4-14(I) shows the header structure used by the proposed traceability service while using AES-GCM mode as the encryption method. The Length field is used to define the length of the string for denoting the traceability data. Basically all the traceability information is stored in the Trace part of the Figure 4-14(I) AES-GCM header structure. All trace information is stored there as a string value. This enables the traceability packet header to be at once decrypted and encrypted at each hop rather than performing individual decryption and encryption cycles for individual trace information. Figure 4-14(II) shows the header structure used by the AES-CTR mode. In the AES counter mode (AES-CTR), the packet payloads are encrypted using the symmetric AES key together with a onetime used counter value.

According to [129], when using the AES-CTR mode, each packet must be encrypted using a

unique counter value together with the key value, and the counter values must not be reused. In order to generate the random counter value, this implementation uses the random number generating function available in the Crypto++ library [117]. Users can also use the rand() function in C++, which generates random numbers for this task according to their requirements. However, if the users are using the rand() function, it should be initialized with a distinctive value. In this implementation, system time during the particular encryption operation is used as the distinctive value while using the rand(). Network administrators can preprogram the counter generation method appropriately depending on their requirements or depending on the application requirements.

The main challenge in the AES-CTR mode is to exchange the counter values securely and reliably among the communication parties for each packet. For this purpose, the header structure shown in Figure 4-14(II) is used. Using this header structure, the proposed traceability service attaches the counter value together with each traceability packet. The Counter Value field contains the counter value used by the encryption party to encrypt the traceability data in the particular packet. While exchanging the counter, the intruders and attackers can alter the counter values, or, because of bit errors, the counter value can be changed in the transmission. Therefore, proper error checking and authentication methods must be used in the traceability packets.

In order to overcome these problems, HMAC-SHA Value field in the traceability packet is used as illustrated in Figure 4-14(II). In this proposed traceability method, HMAC-SHA hash-based message authentication code is used in order to provide authentication. When sending the counter value with the packets, it ensures that the decryptor can generate the key stream required for the decryption even when some of the packets are lost or reordered. The symmetric AES key, which is exchanged using the PKI, is used in the HMAC generation. As the AES key value is only known by the neighbor routers, a forged HMAC value can be easily identifiable over a valid HMAC value. Trace Size field defines the length of the string for denoting the traceability data. Traceability information is stored inside the Trace portion of the AES-CTR header. The encrypted traceability data are also stored in string format similar to the AES-GCM header.

The structure of the Trace fields in both the AES-GCM and AES-CTR headers is illustrated in Figure 4-14(III). Once a router requires the attachment of the traceability information to a received traceability packet, it appends the trace information as the router ID or interfaces details to the string separated by a tab, as shown in Figure 4-14(III). Then the traceability data is encrypted using the appropriate encryption method before transmitting the packet.

The proposed traceability service can provide traceability information in three ways, as mentioned previously: router ID, IP addresses, or MAC Addresses. Although all IPv4 addresses are denoted in dotted decimal format, they are actually represented by integer values. Therefore, to further reduce the size of the string, the implementation employs the denoted integer values of particular IPv4 addresses. Because all traceability information are arranged in the same string, all the routers that use this service must perform the decryption and encryption steps once per packet to minimize the overall encryption and decryption delays.

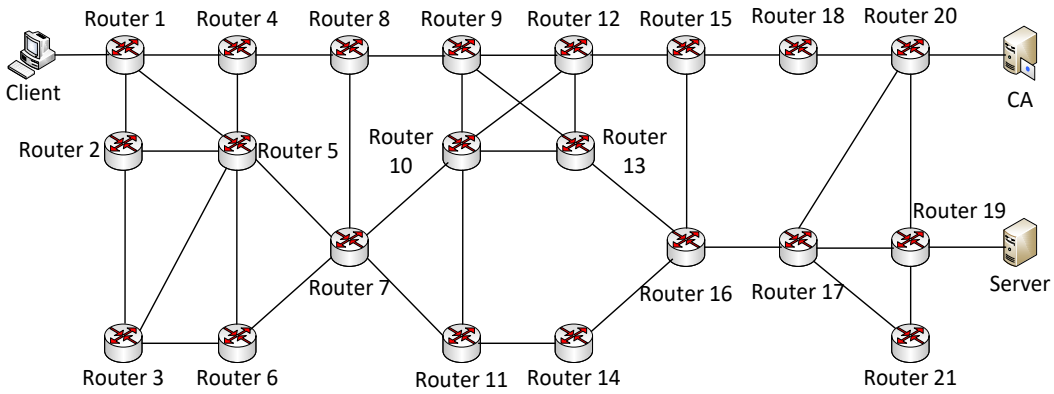


Figure 4-15 Simulation topology

4.4.4 Topology implementation and simulation

The proposed protocol was simulated in a network topology implemented in the ns-3 simulator, as shown in Figure 4-15. The topology was implemented to cover most network topology types, such as mesh and star. The testbed network contained a total of 20 routers and a client, a server, and a CA. All links in the topology were configured with a bandwidth of 5 Mbps and a delay of 2 ms. The client was programmed to send packets to the server with a gap of 0.02 ms. The packet propagating path was not hardcoded and was automatically selected by the underlying routing protocol via the highest metric path.

The main advantage of this method is that the most efficient and effective path can be selected according to the real-time link conditions. This speeds up the packet transmission while optimizing congestion conditions in wide-area networks. In addition, the protocol enables simultaneous transmission of both ordinary packets of the network and the packets that must be securely delivered via encryption. Meanwhile, the encryption level and key sizes are also dynamically configured according to the sensitivity of the data or the client and application requirements.

In this implementation process, the routers can use different individual key lengths for all their interfaces. Nowadays, in the case of encryption, many countries have imposed restrictions on key lengths for the data traveling through their networks. The proposed method helps in overcoming such constraints by maintaining the required key length based on these requirements in every interface as needed. Therefore, whenever the rules change, the key lengths can be centrally managed, thereby giving total control to the respective countries according to their individual requirements.

The main limitations of the proposed protocol were the delays caused by the encryption and decryption processes while routing encrypted packets and the additional processing delays and memory required by neighbor data retransmission buffers.

In the implementation, AES encryption types AES-GCM and AES-CTR are used as the encryption algorithms with 128-bit and 256-bit key lengths. As illustrated in Figure 4-15, all

links in the topology were configured with a throughput of 5 Mbps and a 2 ms delay. While acquiring the test results, we obtained the total processing time consumed by the processor in microseconds when executing the particular functions using the real-time stamp counter of the CPU.

The experiment was executed in the topology illustrated in Figure 4-15. The client was programmed to send packets destined to the server. To reach the server, the packets traveled through the paths Router 1, Router 4, Router 8, Router 9, Router 13, Router 16, Router 17, and Router 19. The underlying link-state routing protocol automatically selected this path because it was one of the least expensive paths to reach the destination. The proposed method could have provided extended security to the packets transferred through the public networks; however, the per-hop data encryption process would have incurred processing overhead during the decryption and encryption processes in each hop. Therefore, the main evaluation was performed by evaluating the processing times incurred by the routers.

The testbed was programmed to send packets from the client destined to the server. The protocol automatically selected the most optimized path, which had the least cost/metric. The testbed was programmed to send traceability-requested packets predominantly in two scenarios:

1. Unencrypted (plain) traceability packets
2. Encrypted traceability packets.

In each scenario, three sub-scenarios were executed for each encryption methods and configured to use the three different traceability options as outlined below.

1. Sending the router ID as the trace
2. Sending the IP addresses of the received and forwarding interfaces as the trace
3. Sending the MAC addresses of the received and forwarding interfaces as the trace

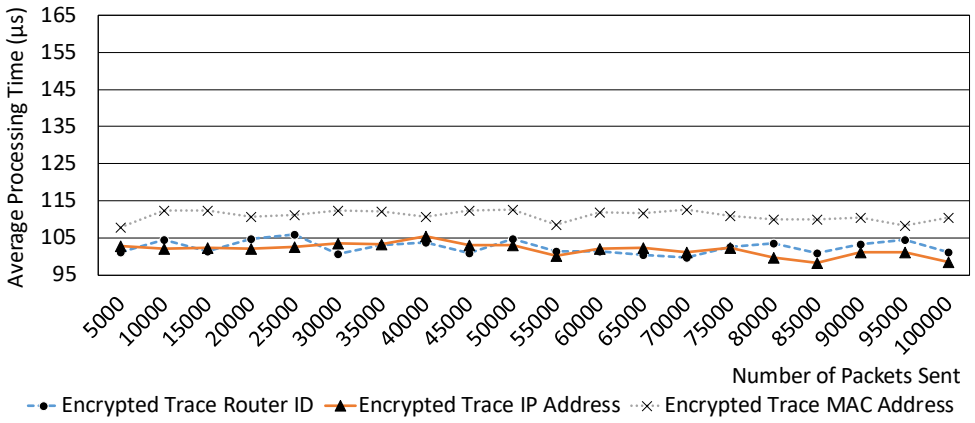
The processing time of each router was measured for 20 test cases, starting from 5,000 packets to 100,000 packets, with a packet gap of 5,000 packets. We used the average processing time for the comparison. Processing time was used for tasks such as decryption and encryption processing in per-hop data encryption, serializing packet headers, deserializing packet headers, and selecting routes.

4.4.5 Evaluations

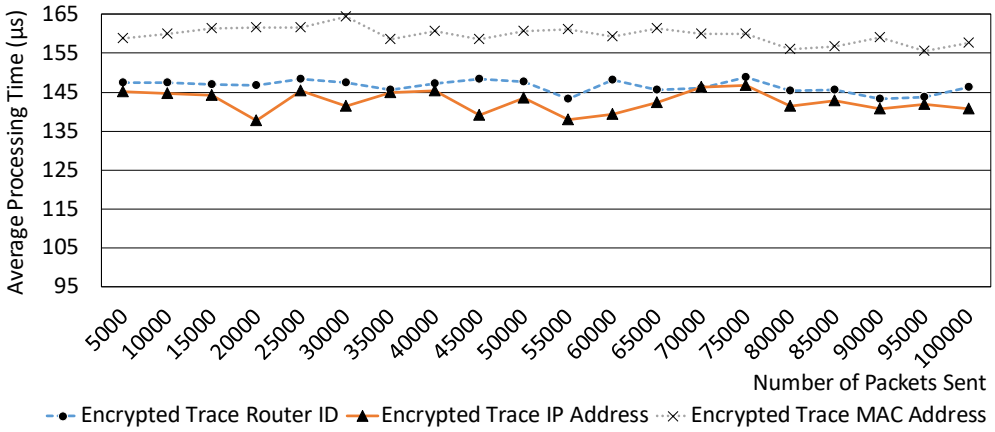
For the initial evaluation, the client was programmed to send three types of packets destined to the server in the topology shown in Figure 4-15. First, the client was programmed to send plain traceability packets with no encryption using the router ID as the trace information. In the next scenario, the client was programmed to send plain traceability packets to the server using the IP addresses as the trace information. Finally, the client was programmed to send plain traceability packets destined to the server using the MAC addresses as the trace information. In each scenario, the processing time incurred by the routers for packet processing was measured

along the packet delivery path.

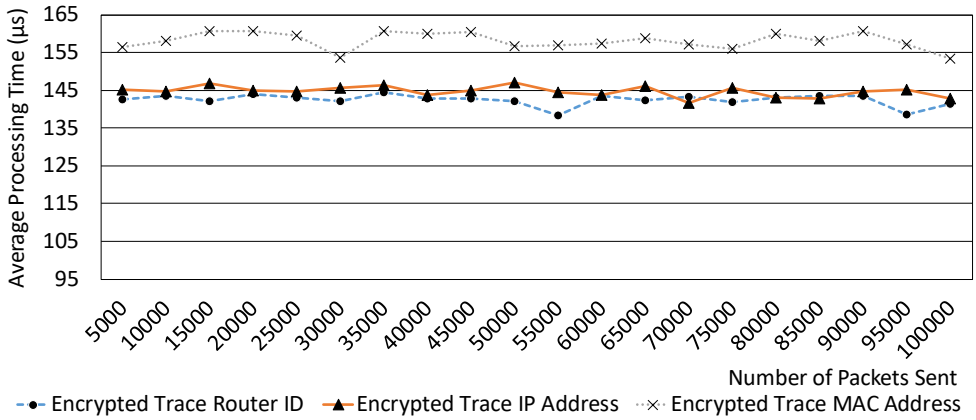
The client was started 5 s after the simulation began. The client was programmed to send a fixed number of packets at an interval of 0.02 s. The first 5 s were allocated to initializing, generating, and exchanging the PKI and AES keys, creating the key tables, and propagating the



(I) Plain traceability packets

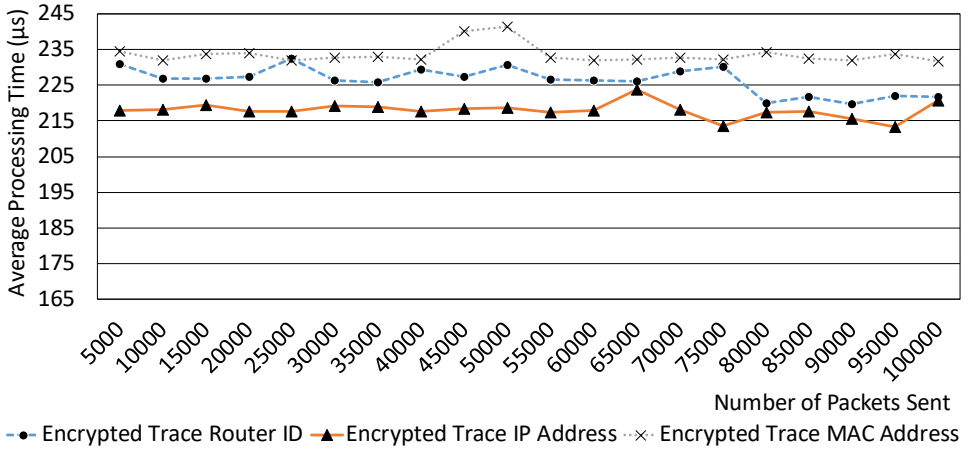


(II) 128-bits AES-GCM encrypted traceability packets

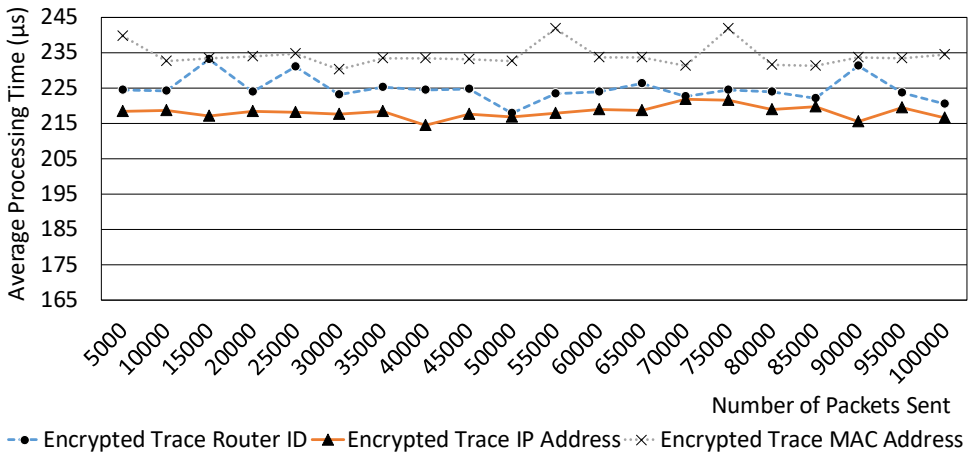


(III) 256-bits AES-GCM encrypted traceability packets

Figure 4-16 APT for packets



(I) 128-bits AES-CTR encrypted traceability packets



(II) 256-bits AES-CTR encrypted traceability packets

Figure 4-17 APT for AES-CTR encrypted packets

routing tables of the routing protocol. These processes were automatically executed by the underlying routing protocol that was implemented, regardless of the network topology and the number of routers in any given topology. The average processing times incurred by the routers for the above scenarios were measured, and the results were plotted in the graphs, as illustrated in Figures 4-16 and 4-17.

According to information in Figure 4-16, the average processing time for a plain trace to provide a trace by router IDs was 102.48 μ s. The average processing time for plain trace packets to provide a trace by IP addresses was 101.85 μ s. The average processing time for plain trace packets to provide a trace by MAC addresses was 110.98 μ s.

Then, the simulation was programmed to send per-hop encrypted packets using the same parameters, such as packet size, link throughputs, and the link delays for two cases, using AES-GCM and AES-CTR encryption algorithms each with respective key lengths of 128 bits and 256 bits. The obtained test results for the AES-GCM encrypted trace were plotted in the graph shown in Figures 4-16(II) and 4-16(III). The obtained results for the AEC-CTR encryption are plotted in Figure 4-17.

Table 4-4 Summary of evaluation results (encrypted trace average values)

Trace type	Plain trace average	AES-GCM (128 bits) (µs)	AES-CTR (128 bits) (µs)	AES-GCM (256 bits) (µs)	AES-CTR (256 bits) (µs)
Node ID	102.48	146.51	226.39	142.5	224.76
IP Address	101.85	142.59	217.96	144.72	218.24
MAC Address	110.98	159.68	233.6	158.16	234.23

According to Figure 4-16(II), the routers consumed an average of 146.51, 142.59, and 159.68 μs to process AES-GCM (128-bits key) encrypted packets with router ID, IP address, and MAC address as the trace value, respectively. Moreover, according to Figure 4-16(III), the routers on average consumed 142.5, 144.72, and 158.16 μs to process AES-GCM (256-bit key) encrypted packets with router ID, IP addresses, and MAC addresses as the trace values, respectively.

According to Figure 4-17(I), the routers consumed an average of 226.39, 217.96, and 233.6 μs to process AES-CTR (128-bit key) encrypted packets with router ID, IP address, and MAC addresses as the trace value, respectively. Moreover, according to Figure 4-17(II), the routers on average consumed 224.76, 218.24, and 234.23 μs to process AES-CTR (256-bit key) encrypted packets with router ID, IP addresses, and MAC addresses as the trace values, respectively. Then we measured the processing cost for the route calculation process without any encryption in order to measure the pure processing cost for routing. According to the test results, for the given topology, the average processing cost for packet routing was 65.02 μs. Therefore, all the above results give 65.02 μs as the routing cost, which is consumed by the underlying routing protocol.

4.4.6 Results discussion

A summary of the results is shown in Table 4-4, and the difference and the percentage increment of the encryption processes over the plain traceability packet processing are compared in Table 4-5. When using the 128-bit AES-GCM encryption method for per-hop data encryption, an average of 30.06%, 28.57%, and 30.5% of additional per-packet processing overhead was

Table 4-5 Overhead processing delays caused by the encryption traceability over the plain traceability

Trace type	Diff with Plain (AES-GCM 128 bits µs)	% increment AES-GCM (128 bits) (%)	Diff with Plain (AES-CTR 128 bits µs)	% increment AES-CTR (128 bits) (%)	Diff with Plain (AES-GCM 256 bits µs)	% increment AES-GCM (256 bits) (%)	Diff with Plain (AES-CTR 256 bits µs)	% increment AES-CTR (256 bits) (%)
Node ID	44.04	30.06	123.91	54.73	40.02	28.09	122.28	54.41
IP Address	40.74	28.57	116.11	53.27	42.88	29.63	116.39	53.33
MAC Address	48.69	30.5	122.62	52.49	47.18	29.83	123.25	52.62

incurred over that of the plain trace when traceability service was provided using the router ID, IP addresses, and MAC addresses, respectively. While using AES-CTR mode with the 128-bit keyspace, the additional encryption overheads were increased to 54.73%, 53.27%, and 52.49% for trace options router ID, IP addresses, and MAC addresses, respectively.

When using a 256-bit AES-GCM encryption key length for per-hop data encryption, the processing increment was 28.09%, 29.63%, and 29.83% when implemented using the router ID, IP addresses, and MAC addresses as the traceability option, respectively. Finally, 54.41%, 53.33%, and 52.62% of encryption increments were shown for the 256-bit AES-CTR mode for trace options router ID, IP addresses, and MAC addresses, respectively.

Based on the test results, after considering all the three traceability options, an average of 44.49 and 43.36 μs of additional processing overhead (averaging all three traceability options) per packet per hop was added when using the proposed traceability service with 128-bit and 256-bit AES-GCM keyspaces, respectively. Moreover, those values for the AES-CTR mode was 120.88 and 120.64 μs for 128-bit and 256-bit keyspaces, respectively

According to ITU-T Recommendation G.114 [118], if the total transmission delays over the networks are kept less than 150 ms, most applications would not be significantly affected. According to the test results, the highest processing delay of around 235 μs (0.235 ms) was caused by the AES-CTR mode while using a keyspace of 256 bits. Therefore, a packet should pass over around 638 hops without considering the link delays to generate a delay larger than 150 ms. Even with the link costs, the encryption delay caused by the proposed method is significantly small and tolerable over public networks. Therefore, we can conclude that even with the link and other packet propagation delays, the proposed traceability method can provide secured traceability service without effecting end-to-end delays of notification over the public networks.

Every neighboring router interface pair uses a unique key in both of the encryption methods, so the original trace details of the packet could be obtained via decryption only by the destined router. Therefore, the proposed method could provide higher security to the traceability data transmitting over public networks. The proposed service displayed an average of 44.49 and 43.36 μs for AES-GCM mode and 120.88 and 120.64 μs for AES-CTR mode increments of packet propagation delays per hop for 128-bit and 256-bit key lengths, respectively. Hence, it is evident that the proposed method with the AES-CTR mode provided the highest security over public networks due to the random counter value it used for each encryption round. Moreover, it was shown that the proposed traceability service could be used, or the trace details attached, to any packets transferring through the public networks.

Additionally, the user or application can select the traceability options along with the encryption algorithms and encryption key lengths as preferred along the data transmission path. Furthermore, network administrators can implement their preferred constraints regarding the encryption algorithms and key lengths according to country-specific policies. In short, the proposed traceability service can provide all these flexible aspects to users, applications, and network administrators.

Table 4-6 Trace retrieve delays for the proposed method (from client to server as in Figure 4-15)

Trace type	Plain trace (ms)	AES-GCM 128 bits (ms)	AES-CTR 128 bits (ms)	AES-GCM 256 bits (ms)	AES-CTR 256 bits (ms)
Node ID	37.34	37.69	38.51	37.66	38.54
IP Address	37.33	37.66	38.49	37.68	38.5
MAC Address	37.41	37.8	38.6	37.79	38.63

For the final evaluation, we implemented the prototype method to simulate the traditional traceroute method in ns-3. We used the ICMP packet service in ns-3 and the same testbed as shown in Figure 4-15. The client was programmed to trace the server. According to the obtained test results, the simulated trace route method on average consumed a total of 185.95 ms to obtain the trace to the server. In contrast, with the same testbed configurations, the proposed traceability method consumed total delays as shown in Table 4-6 to obtain an encrypted trace to the server. A significant difference between the two methods is clearly evident. This is because the standard trace route method obtained the trace details by incrementing the TTL value in a hop-by-hop manner relative to the source (client). With the proposed method, on the other hand, the traceability data was obtained and attached while the packet was transmitting through the network. Furthermore, in the proposed method there was no repeat process to report back to the source in each hop. This prevents the huge delay caused by link delays.

However, for this evaluation, we used an unencrypted return trace packet from the destination to the source that contained the trace details in the proposed service. Hence, using the CA architecture, the proposed service can return this packet in encrypted form using the sender's public key. The main advantage of this mechanism is that it introduces no additional link delays and provides a fully secure traceability service to end users.

Moreover, the existing Internet infrastructure does not require to be totally changed to use the proposed method. Only adding routers to the networks according to the network size is sufficient to deploy all the services. As mentioned in Section 2.3, the content-based router is a middleware that can be implemented on a Cisco AXP and Juniper JunosV App Engine. Therefore, routers can be easily added to the existing routers as a module without any additional infrastructure change. Also, the proposed service can be provided easily by extending the existing routing protocols. Linux-based Crypto++ library is used as the cryptographic library for all the cryptographic implementations [108]. Therefore, the proposed method operates much more accurately as a real-world implementation. Furthermore, this enables a much easier migration of the total infrastructure to the real-world environment. The proposed traceability service consists of the following: (i) merging with underlying link-state routing protocols, (ii) managing key tables, (iii) providing per-hop data encryption, (iv) working with a CA-like infrastructure, and (v) providing fast, secure and flexible traceability services over public networks.

4.4.7 Conclusion

This section introduced a traceability service that employs a per-hop data encryption method to provide a secured traceability service on public networks. The proposed traceability service can provide plain traceability service and encrypted traceability service for added security. The traceability service is sufficiently flexible to provide the trace information in three methods as the router ID, the IP addresses of the received and forwarding interface addresses, and MAC addresses of the received and forwarding interface addresses according to the user or application requirements. Moreover, the encryption keyspaces together with the encryption algorithms among the hops can be dynamically selected according to the network or application requirements.

The proposed method can provide the traceability service to any UDP or TCP packet by attaching the trace packet header to the packet. It can simultaneously provide higher security to both the data and traceability information when the packets traverse public networks with the implemented per-hop data encryption protocol. The proposed architecture is additionally capable of using variable key lengths and different encryption algorithms between the hops of routers. This may drastically reduce encryption and decryption delays, which are caused when using higher keyspaces in routers that have low resources or routers in congested environments.

This can solve the issue of key constraint policies that countries may apply to their public networks. These flexible attributes come with maximum costs of additional 43.36 and 120.64 μs (average of all three traceability options) of average processing overhead per packet per hop when the proposed traceability service is used with 256-bit keyspaces, and 44.49 and 120.88 μs for the 128-bit keyspaces for AES-GCM and AES-CTR encryption modes, respectively. The proposed traceability service consumed a maximum of 37.79 and 38.39 ms to provide the end-to-end trace using AES-GCM and AES-CTR modes, respectively, with 256-bit key lengths in the evaluated network topology, where it took 185.95 ms in the conventional traceroute method. Therefore, the proposed secured traceability method can provide about 79% faster performance over the conventional traceroute method with added security.

Nevertheless, the proposed method implies no vulnerabilities or drawbacks in the existing ICMP-based traceability methods such as intermediate device blockings, non-inline trace details, and unsecure traceability data. The implemented AES-CTR mode is a proven method to be used with Gbps class packet streams while providing authentication to the packets and can overcome lost or reordered packet scenarios. As the proposed traceability methods incur much less delays, much less than 150 ms, most applications would not be affected [118], and we can conclude that the proposed method can be used as a secure and a reliable traceability method over the public networks. Finally, because the proposed traceability service is implemented in routers, we can validate and authenticate the service easily and effectively by using the CA architecture.

Chapter 5 Hardware-based router implementation

5.1 Motivation

Generally, security services consume many resources compared to general services as they required to run operations which are CPU intensive such as string matching and DPI. If incorporated, content-based routers consume a huge amount of performance in order to provide the security mentioned above services. Therefore, we need a proper optimized high-end solution in order to counteract the performance bottleneck. To this end, I propose the software deep packet inspector on a router (DooR) as the core software for the content-based routers. The main objective of this chapter is to implement and evaluate the DooR software. DooR is designed and developed from the ground up focusing to act as the router core providing content-based services. It is highly flexible to communicate with user applications, and users can also develop modules to the router through DooR. DooR uses the software acceleration of the Intel DPDK library [45] to accelerate packet processing workloads. Moreover, it uses the Intel optimized high-speed Intel HyperScan string matching library to achieve gigabit range throughput for string matching.

DooR is proposed as a high efficient on-the-fly packet analysis software which can tolerate Gbps rate throughputs depending on the configured functionalities. The main advantage of DooR is that it runs on any conventional Linux-based computer with Intel-based commodity hardware (processors and Network interface cards (NICs)). DooR stores only the required states of a particular packet after the DPI process completes. DooR also saves the state information in the processes such as chunk encoding, Gzip decoding and string matching to the appropriate TCP stream data. For each consecutive packet received for use by an existing stream, the saved states are retrieved from memory and DooR resumes the DPI process using that saved state information. DooR then performs chunk decoding, Gzip decompression and string matching on the packet content using a preconfigured rule set. DooR is implemented with Aho-Corasick-based string matching and Intel HyperScan string matching which can delivering 160 Gbps DPI throughput.

The matched rule information is stored based on the user-configured method such as a MySQL database, LevelDB key-value store, memory file, or hard disk file. After analyzing the packets, DooR uses its longest-prefix-match (LPM)-based routing tables or the Patricia-Trie-based routing tables in order to route the packets. With the aid of the Intel Xeon multicore architecture combined with the Intel architecture (IA) of Intel-based NICs, DooR can provide flexible, feasible, and high-speed DPI services and routing solutions to its users and applications.

Table 5-1 Software library versions

Library Name	Version
DPDK	1.8.0/2.0.0/2.1.0/2.2.0/16.04
gcc	gcc-4.4.7-16.el6.x86_64
glibc	glibc-2.12-1.166.el6_7.3.x86_64, glibc-2.12-1.166.el6_7.3.i686
python	python-2.6.6-64.el6.x86_64
MySQL	mysql-5.1.73-5.el6_6.x86_64 / 10.1.10-MariaDB
LevelDB	leveldb-1.7.0-2.el6.i686
tokyocabinet	tokyocabinet-1.4.33-6.el6
zlib	zlib-1.2.3-29.el6.x86_64

5.2 Deep packet inspector on a router (DooR)

5.2.1 DooR implementation

When incorporated with content-based routers, DooR acts as a packet routing software implemented in commodity hardware, which performs on the fly DPI over the contents of the routed packets. DooR is developed using the C programming language with the libraries and drivers of Intel DPDK [45] in a Linux environment. Using the Intel DPDK library and its drivers, DooR can achieve faster packet processing over 20-Gbps throughput using the hardware-based acceleration of the Intel Xeon processors and Intel NICs. However, the achievable throughput is highly dependent on various conditions such as traffic density, rule set size (in string matching), hardware configurations (CPU, Memory, NIC, etc.), and the type of database server used. Table 5-1 shows the software libraries and version information used for this implementation.

5.2.1.1 Multicore architecture

As illustrated in Figure 5-1, DooR was implemented using a multicore architecture that enables it to perform TCP stream reconstruction and string matching functions tolerating higher throughputs by dividing the workloads among CPU cores. As shown in Figure 5-1, one CPU core per NIC is assigned separately as the input/output (I/O) RX cores (or reader cores) and I/O TX cores (or writer cores). In addition, many CPU cores, depending on user requirements, are assigned as deep packet inspection cores (DPI cores). Another CPU core is allocated as the status core, where the status core displays the DooR program status periodically and runs the

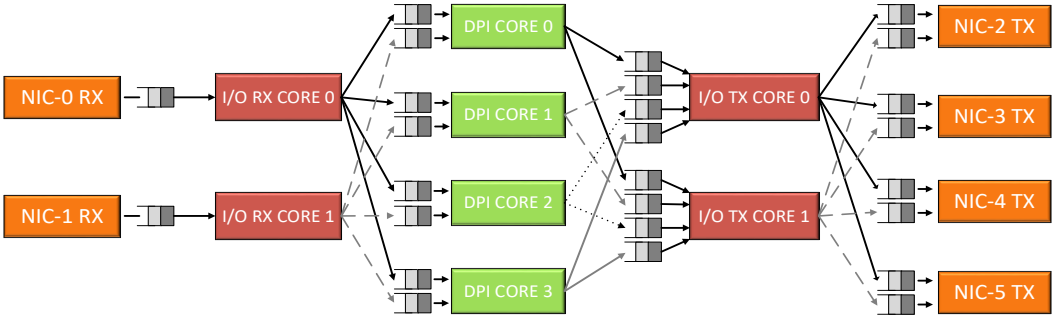


Figure 5-1 DooR architecture

TCP stream timeout manager.

According to Figure 5-1, I/O RX cores are responsible for reading the packets from the NICs and forwarding the packets to the DPI cores after performing the flow abstraction. When a packet arrives at the NIC RX queues, the connected I/O RX core(s) (reader cores) read the packets from the RX queues and redirects the packets of the same stream to the corresponding DPI cores' RX queue. The flow abstraction and the stream identification processes of the I/O cores are explained under the sub section HTTP Traffic Filter and Stream Classifier. Then the DPI cores read the packets form their corresponding attached RX queues. DPI cores perform the tasks such as DPI up to OSI layer 7, TCP stream reconstruction, string matching, and saving the extracted information to the configured database.

After these DPI processes, DPI cores perform packet routing according to the user configured routing method. The packet routing process obtains the NIC port which the packet to be forwarded according to the destination address from the routing tables. Afterward, DPI cores forward the packets to the appropriate RX queues of the I/O TX cores (writer cores). Finally, I/O TX cores read their RX queues and send the received packets to the connected TX-NICs.

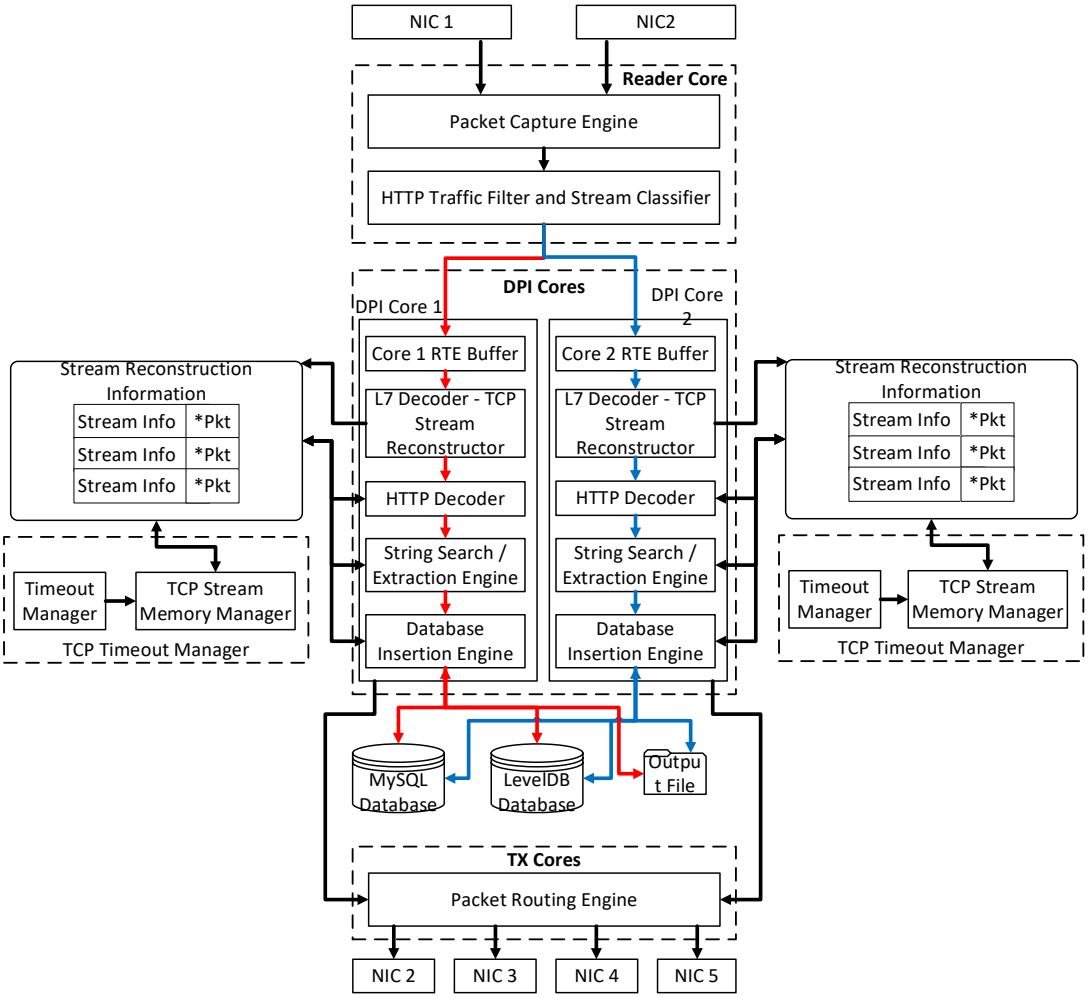


Figure 5-2 Main modules of DooR

The DooR is totally configurable by the users, and they can assign all the NICs and the CPU cores together with the routing or database saving settings according to their requirements or provisional to the hardware configurations of their PCs (see the documentation for further details). As illustrated in Figure 5-2, DooR can mainly divide into the following eight modules.

5.2.1.2 Load balancing packets between DPI cores

Figure 5-1 also reveals that I/O RX cores are responsible for reading the packets from the NICs and forwarding the packets to the DPI cores after performing flow abstraction based on Layer 4 information performing load balancing the received packets among the configured DPI cores. Reader cores read the pair source IP address-port and destination IP address-port values from the received packet and generate a hash value by performing the XOR operation over the four values. The hash value is generated by considering the number of DPI cores used by the program.

DooR uses the last significant bits (LSBs) of the calculated mask and extracts the bits equal to two to the power value of the total assigned DPI core numbers. This value is used as the corresponding DPI core number. The main advantage of this method is that it does not require storing and managing any details regarding the streams in order to direct the packets of the same stream to the same DPI core. This overall process reduces the processing power in managing the stored stream data in data structures such as queues and vectors, thus increasing the total throughput of the reader cores. A single reader core also can be used to read packets from multiple NICs and writing data to a particular NIC port after analysis acting as both a reader and a writer core. The current DooR version was tested with Intel X710-DA2 (2×10Gbps ports), Intel X710-DA4 (4×10Gbps ports), and Intel 82599ES (2×10Gbps ports) NICs.

5.2.1.3 Packet capture engine

This module captures the packets from the attached Intel-based NICs using the Intel DPDK library which allows DooR to directly copy the packets from the NICs to the user space, bypassing the kernel space. This process totally removes the delays caused by going through the kernel space while copying the packet to the userspace (zero copy). DooR stores the captured packets in its own DPDK based run-time environment ring buffers (RTE ring buffers) until the total DPI process finishes. All the packets are retrieved through polling method where it does not interrupt the CPU process unlike in interrupt-based method.

Reader cores are responsible for reading the packets from the NICs and delivering them to the corresponding DPI cores depending on the Layer 4 information. The reader core and writer core assignment are totally configured according to the user given parameters. A single reader core also can be used to read packets from multiple NICs and acting as both the reader and writer core. The current DooR version was tested with Intel XL710 (2×40Gbps ports), Intel X710 (2×10Gbps ports), Intel I350 (4×1Gbps ports), and Intel 82599ES (2×10Gbps ports) NICs. Test results proved that the packet capture engine was capable of reading 20Gbps (2×10Gbps) data rates without any packet drop by only using one reader core.

5.2.1.4 HTTP Traffic Filter and Stream Classifier

As illustrated in Figure 5-3 flowchart, the reader core fetches the packets from the attached NICs (via packet capture engine), and the HTTP traffic filter and stream classifier module analyze

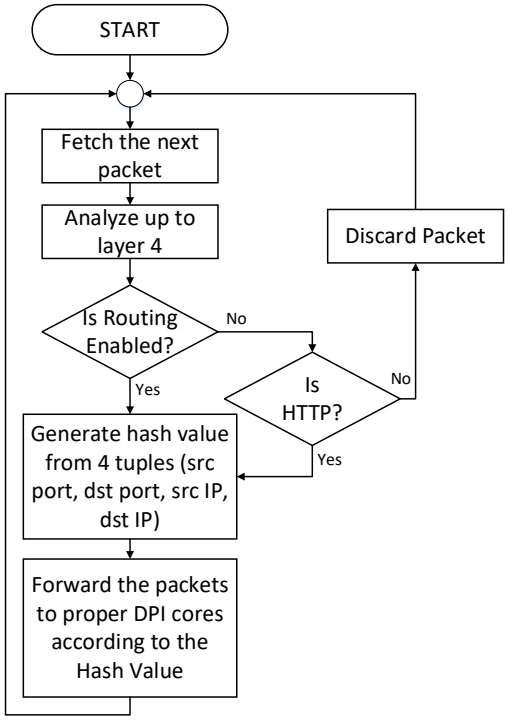


Figure 5-3 Reader core operation flowchart

them up to layer 4 (transport layer). The basic method is to store the stream information and forward the packets to the DPI cores using an appropriate load balancing method. However, from the initial DooR implementation, we found that in order to store, retrieving and managing these kind of information using data structures take so much CPU cycles hence, reducing the throughput of the reader cores resulting incoming packet drops. In order to overcome the stream load balancing/classifying problem, DooR was implemented with a load balancing mechanism using a generated hash value as explained below. This is an on-the-fly stream abstraction method, and it totally omits the requirements of storing any information regarding the streams hence providing faster and feasible load balancing.

The reader core reads the pair of source IP address-port and destination IP address-port values and generates a hash value by performing the XOR operation over the four values. The hash value is generated considering the number of DPI cores used by the program. DooR uses the last significant bits (LSBs) of the calculated mask and extracts the bits which equal to the two to the power value of the total assigned DPI core numbers. This value is used as the corresponding DPI core number. The main advantage of this method is that it does not require to store and manage any details regarding the streams in order to direct the packets of the same stream to the DPI core. This overall process reduces the overall processing power hence increases the total throughput of the reader core.

Then the packet is forwarded to the RTE queues of the corresponding DPI cores' according to the calculated hash value. As the same hash value is generated for the same value pairs, the reader core does not require to store any information regarding the packet forwarding yet enabling the ability to forward all the packets of a particular stream to the same DPI core.

5.2.1.5 TCP stream reconstruction engines

TCP stream reconstruction engine reconstructs the TCP connections on the fly analyzing the packets up to Layer 7 in DPI cores. Therefore, as illustrated in the context switching process in Figure 5-4, DooR saves the intermediate states of the currently analyzing packet upon finishing the analysis (as shown in stream reconstruction information in Figure 5-2), in order to resume the TCP stream reconstruction process afterward receiving the next packet of the stream. As shown in Figure 5-4, stream reconstruction information consists the reconstructed TCP stream data of a particular DPI core. A, B and C are the currently reconstructing streams where the relevant info consists the information regarding the current TCP stream such as the stream direction, source and destination IP address and port number pairs, packet count, etc. Interstate details contain the information regarding the state of the TCP streams such as current sequence number, lifetime, previous string matching state, etc.

Moreover, each TCP stream contains a pointer to the currently inspecting packet (ex. *A2, *B1, *C2). According to Figure 5-4, the program is currently processing the packet C2. Therefore, DooR loads all the relevant info and interstate information regarding the TCP stream C and saves a pointer (*C2) in the stream reconstruction information. This pointer is used to obtain the information to perform further DPI functions such as; HTTP stream reconstruction and string matching. DooR uses a hash key and a port map key for individually identify the TCP connections. The hash key is created using the value pairs Source/Destination, IP address/Port values and it is used in searching both the TCP connections and streams. The largest port number of a particular stream (Ex: if the communication is from port 1234 to port 80 then port 1234

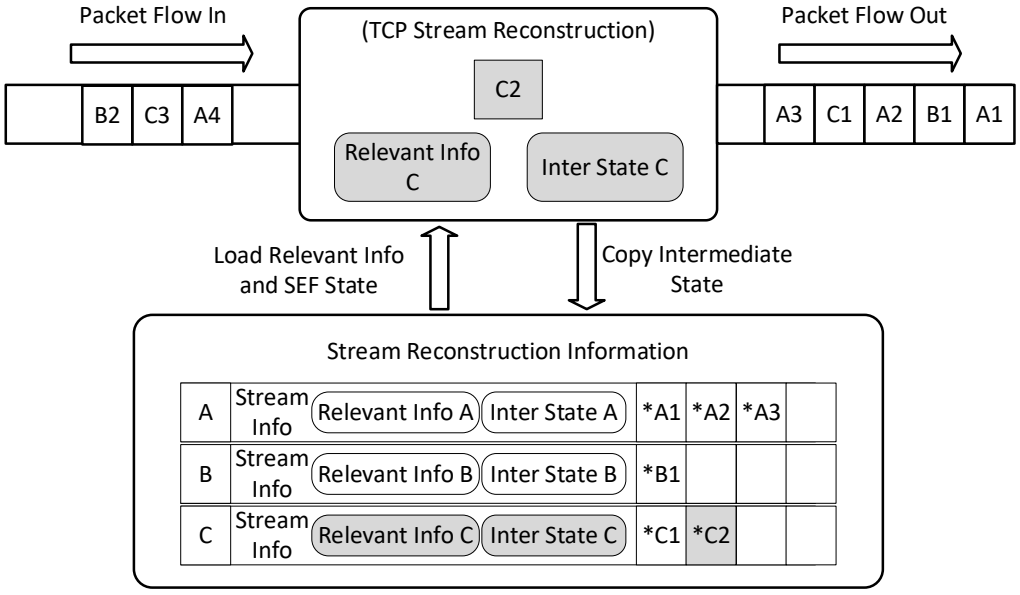


Figure 5-4 Context switching process

becomes the port map key for that stream) is selected as the port map key for a particular stream.

TCP connection creation is detected using the SYN exchange in communication, and when a SYN is detected without an ACK, DooR creates a new TCP connection using the packet details. In TCP connections, the packets are sent in streams, and a new stream is identified when the communication direction is changed. The TCP connection termination is identified via the FIN or RST flags of the packets.

The TCP stream reconstruction process used by DooR is illustrated in the flowchart in Figure 5-5. According to the Figure 5-5 flow chart, once the packet C2 (of Figure 5-4) receives at the TCP stream reconstruction engine, it searches the TCP connections using the hash key in order

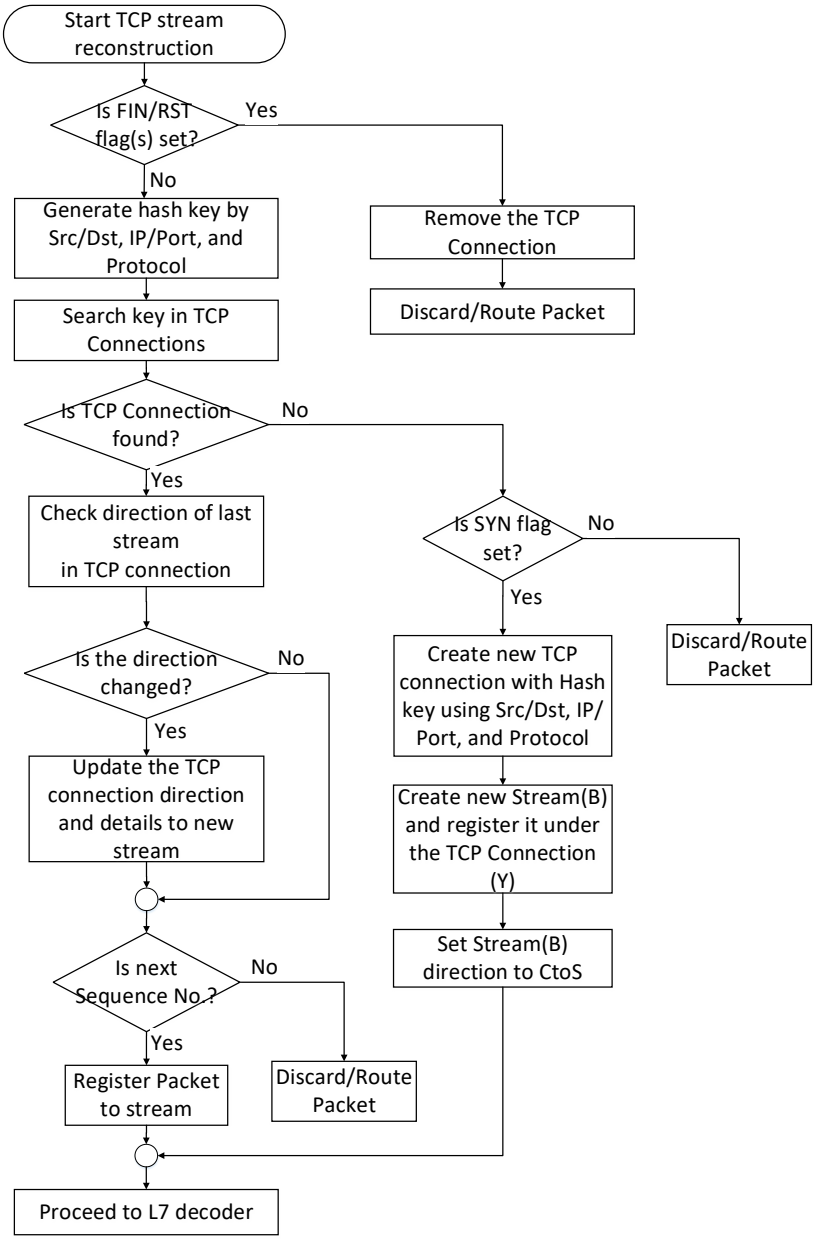


Figure 5-5 TCP stream reconstruction process flowchart

to find whether the packet belongs to an existing TCP connection. If a TCP connection is not found, DooR creates a new hash key by reversing/swapping the source port and destination port. Usually, a stream starts from client-to-server and after requesting the data, server replies to the client. In this reply process, the stream direction is changed from client-to-server to server-to-client. Therefore, if the TCP stream is not found, DooR searches the TCP connections swapping the port numbers. If still the TCP connection is not found, DooR checks whether it is a new TCP connection by checking whether the SYN flag is set in the packet. If the SYN flag is unset, the packet is forwarded (if routing is enabled) or discarded according to the configuration. If SYN flag is set DooR creates a new TCP connection and adds it to the TCP connections using the packet data and moves to the next step, checking whether the FIN/RST flags are set or not, as represented in Figure 5-5.

If the TCP connection is found, the interstate and the relevant information are loaded from the retrieved stream. First, the stream direction is checked, and if the stream direction is toggled, it creates a new stream. If the direction is the same, the stream is identified as continuing and proceed to reconstruct the stream. Here, the sequence number of the packet is considered, and in order to proceed with the stream reconstruction, it must be the consecutive sequence number. If the sequence number is consecutive with the last packet, the packet is registered with the stream, and the layer 7 decoder proceeds with the packet content decoding. Finally, if one of the FIN or RST flags is detected in the packet, DooR detects the TCP connection termination and marks the appropriate TCP connection state as END. This concludes the TCP stream reconstruction process and the packet is forwarded to the HTTP decoder.

With the aid of the stream mentioned above reconstruction method, DooR is able to reconstruct the TCP streams on-the-fly with minimum memory usage. Hence, the main disadvantage of this method is that DooR is not able to tolerate out-of-order packets as it processes the packets on-the-fly only storing the latest packet state information. Therefore, it waits till the next packet arrival without proceeding the stream reconstruction, and this affects the stream reconstruction process in scenarios such as TCP retransmission and out-of-order packets.

5.2.1.6 HTTP decoder

HTTP decoder retrieves information and resumes the Layer 7 decode process using the pointer values created by the TCP reconstruction engine. For this implementation, DooR is configured only to analyze HTTP content. DooR searched for the HTTP Headers inside the packets, and if found, the header details are extracted, and the details regarding the HTTP streams are stored. This stored information is then used to reconstruct the HTTP stream. All the allocated memory is managed and freed up on finishing the corresponding processes. Mainly the memory is freed upon successive stream termination (after receiving FIN). In any case, if the FIN is not received, the expired stream data are removed by the TCP timeout manager which is explained under TCP timeout manager sub section.

DooR is capable of decompressing the HTTP content using Gzip and Deflate algorithms and decoding the HTTP chuck encodings on-the-fly. DooR only stores the states of the

decompression without any content to continue the decompression and decoding. None of the existing implementations are capable of performing such tasks on-the-fly without storing or buffering the packet contents.

5.2.1.7 ·String matching and extraction engine

String search/extraction engine searches the packet content decoded by the L7 decoder with the Aho-Corasick ruleset. On detecting a match, DooR stores the important details regarding the match such as timestamp, rule ID, rule, matched position, stream details, packet details, etc., using the Database insertion engine. If a rule is partly matched at the end of a packet, the state of the string matching is saved in the context of the stream. The context is loaded, and string matching is resumed from the saved position subsequent arrival of the next packet of the stream (Figure 5-4). This is one of the major advantages of DooR string matching function as it can resume the string matching states towards the consecutive packet payloads.

5.2.1.7.1 Aho-Corasick string matching algorithm

DooR uses the Aho-Corasick algorithm developed by Mischa Sandberg [130]. DooR has extended the string matching functions in this general Aho-Corasick implementation in order to tolerate with the packet-based string matching. We used some rules defined in the well-known intrusion detection system, snort rule set for this evaluation [106]. DooR generates individual Aho-Corasick trees for each DPI cores.

5.2.1.7.2 Intel HyperScan

DooR is also integrated with the regular expression-based high-speed string matching algorithm by Intel, named as Intel HyperScan [48]. According to the Intel evaluations [131] [132], Intel HyperScan can scale from 1Gbps to 160Gbps string matching performance.

5.2.1.8 TCP timeout manager

Apart from these main types of cores, DooR uses another core type called status core. The status core is used to display DooR program status periodically and to execute the TCP timeout manager. Whenever a new packet belonging to an existing stream is received, the lifetime of the TCP connection is changed to the current time. If no packets are received for a particular TCP stream within 5 mins, the TCP timeout manager identifies the TCP connection as an expired connection. It then removes the entire TCP connection, including all stream details belonging to it, from memory. This process consumes considerable processing power, as it must go through all TCP connections in all DPI cores. If the TCP timeout function is executed in all DPI cores, it results in extensive packet drops in DPI cores. Therefore, the TCP timeout function is periodically executed through the status core whenever it displays the program status. By periodically freeing all expired TCP connections, the TCP timeout manager utilizes the memory used by DooR.

Ordinary TCP streams end within a short period, and the end of the streams is notified to the communication parties using the TCP finish (FIN) flags. However, because of occurrences such as missing packets or a packet drop (containing the TCP FIN or reset (RST) flag), stream

termination of such TCP streams cannot be detected by DooR. Therefore, those TCP connections remain in memory together with the stream details, thus causing an increase in memory usage. Such allocated memory remains in the program runtime. The TCP timeout manager monitors the lifetime of all TCP connections and removes any TCP connection that remains idle for a predefined period.

According to RFC 793 [10], a TCP connection changes its state to TIME-WAIT once a node sends the FIN packet to terminate a connection, and then waits 2×the maximum segment lifetime (MSL) to receive the acknowledgment (ACK) from the other communication party. If an ACK is not received until the TIME-WAIT expires, the communication party terminates the TCP connection and changes the state to CLOSED. This process mitigates missing connection termination packet problems during the TCP communication. The general MSL time value is 2 mins, and it sets the default TIME-WAIT timer value to 4 mins. Considering these facts, DooR uses a TCP timeout value of 5 mins.

Whenever a new packet belongs to an existing stream is received, both the lifetime value of the TCP connection is changed to the current time. If there are no packets received for a particular TCP stream for 5minutes, TCP timeout manager identifying the TCP connection as an expired connection. Then it removes the entire TCP connection including all the stream details belongs to that TCP connection from memory. This process consumes a considerable processing power as it needs to go through all the TCP connections in all the DPI cores. If the TCP timeout function is executed in all the DPI cores, it results in a huge number of packet drops in DPI cores. Therefore, the TCP timeout function is executed via the status core whenever it displays the status. TCP timeout manager utilizes the memory used by DooR by freeing all the memory of the expired TCP connections periodically.

5.2.1.9 Database insertion engine

Database insertion engine saves the extracted information by the String search/extraction engine to the configured saving method. DooR is configurable to save the extracted information into MySQL database, LevelDB key-value store, and to files which either resides in the memory or hard disk drive. Whenever a string match is detected, DB insertion engine stores the matched information such as timestamp, rule ID, rule, matched position, stream details, packet details, etc., to the configured database/file. All of these options are configurable according to the user, application or service requirements.

5.2.1.10 Packet routing engine

DPI cores retrieve packet routing details according to the user-configured routing method after inspecting the packets. DooR performs packet routing mainly using one of the available two methods: the longest prefix match (LPM) and Patricia Trie-based routing. Both of these methods support IPv4 and IPv6 address types. DooR uses separate routing tables to store IPv4 and IPv6 entries. However, DooR provides route add, edit, and delete methods to the users in order to manage the routing tables and to configure the routers as conventional routers. For our evaluations, we use real-world border gateway protocol routing tables provided by the University

Table 5-2 Testbed configurations

Property	Testbed1		Testbed2
PC Type	DPI PC	Packet Generator	DPI PC
Processor	Intel(R) Xeon(R) E5-2697 v2 @ 2.70GHz *2	Intel(R) Core(TM) i7-4790 @ 3.60GHz	Intel(R) Xeon(R) E5-2643 v3 @ 3.40GHz *2
RAM	256GB	16GB	128GB
HDD	1TB SSD + 4TB RAID0	512 SSD	1TB SATA
Huge Page Memory	10GB	4GB	10GB
NIC	Intel X710-DA2 (2*10-Gbps Intel X710-DA4 (4*10-Gbps ports)	Intel 82599ES (2*10-Gbps ports)	Intel X710-DA2 (2*10-Gbps ports), Intel X710-DA4 (4*10-Gbps ports)
OS	CentOS 6.7	CentOS 6.7	CentOS 6.7
Kernel Version	2.6.32-573.8.1.el6.x86_64	2.6.32-573.8.1.el6.x86_64	2.6.32-573.8.1.el6.x86_64

of Oregon route views project (November 2017) [133]. The IPv4 table contained 560,190 entries and the IPv6 table contained 26,634 entries. The packet route retrieving process obtains the NIC port to which the packets are to be forwarded according to the destination address from the routing tables. Afterward, DPI cores place the packets to the corresponding input queues of the I/O TX cores (writer cores).

Finally, the writer cores read the packets from their inbound queues and sends the received packets through the connected TX-NICs. DooR is completely configurable by users, and they can assign all the NICs and determine CPU core configurations together with the routing or database saving settings based on their requirements or provisional to the hardware configurations of their PCs.

The DooR system was evaluated using campus network traffic and core network traffic. All the test cases were executed using the testbed configurations listed in Table 5-2.

5.2.2 Evaluations

5.2.2.1 Evaluate the speed and throughput in packet reading (constant bit rate (CBR) traffic via DPDK-pktgen)

For the first test case, Testbed1 (Table 5-2) was configured using two PCs. The first PC was the DPI node and the second acted as the packet generator. The 10-Gbps NIC ports in the DPI node were directly connected to the NIC ports in the packet generator node. Thus, the main purpose of this testbed was to evaluate the throughput of the DooR reader cores for 10-Gbps and 20-Gbps throughputs using the dpdk-pktgen packet generator. The DPI, string matching, and routing functionalities of the DooR program were disabled during this test case. We configured the dpdk-pktgen to send packets with payload sizes of 500, 1,000, and 1,500 bytes first on a single port and then on both ports. The packets were captured and counted in the analysis node.

Table 5-3 Test results: Test case1

Packet Size	500 Bytes		1,000 Bytes		1,500 Bytes	
# of Packets Sent on Port1 (in Millions)	1,500 M	1,500 M	800 M	800 M	500 M	500 M
# of Packets Sent on Port2 (in Millions)	0	1,500 M	0	800 M	0	500 M
Throughput (MBits/s)	9,999	9,999*2	5,410	5,410*2	4,455	4,455*2
Pkts/s (in Millions)	2.4 M	2.4 M *2	800 M	800 M *2	0.822 M	0.822 M *2
Total Packets Received	1,500 M	3,000 M	800 M	1,600 M	500 M	1,000 M

Table 5-4 Test results: Throughput analysis

Property	Values
Port 1 Throughput (Mbps)	~500-2,000
Port 1 Pkt/s	491,992.43
Port 2 Throughput (Mbps)	500-2000
Port 2 Pkt/s	478,433.94
Received Packets I/O Rx Cores	1,342,334,131
Received Packets DPI Cores	1,342,334,131
No. of String Matches	279,681,012

The results of this test case are summarized in Table 5-3. The packet generator was able to generate packets at maximum data rate of 9.999 Gbps (10 Gbps) per NIC port during the test scenarios. The test results reveal that a single reader core was able to capture all sent packets of all three packet sizes. Thus, we can conclude that the DooR architecture can capture traffic in 10-Gbps throughput using one NIC port and can capture traffic in 20-Gbps throughput while using two NIC ports. However, only a single CPU core was allocated as the reader core to handle both NIC ports during all evaluations of this test case. Assigning individual CPU cores for each NIC port reduces the number of packet drops while achieving extremely high throughputs such as 40/80 Gbps. However, DooR is sufficiently flexible for use with appropriate CPU core configurations and NIC port assignments based on their requirements.

5.2.2.2 Evaluate DooR throughput (HTTP-only generic traffic)

The second test scenario was executed to assess the throughput of the DooR program using generic HTTP traffic captured from our laboratory (Hiroaki Nishi Laboratory, Keio University). The average packet size of the used traffic was 565.78 bytes. For this evaluation, the traffic was injected using both ports for the following two cases measuring the drop rate and memory usage for TCP stream reconstruction while performing HyperScan string matching and routing using a snort rule set. The memory usage for the test case is plotted in Figure 5-6. Moreover, the traffic details and other details are summarized in Table 5-4.

The traffic was injected for 4,760 s, and the average memory usage for the stream reconstruction process was 30.36Mb. Maximum memory usages were 66.95Mb and 67.06Mb. Moreover, the maximum memory consumption by the entire DooR was 8GB, and it was averaged

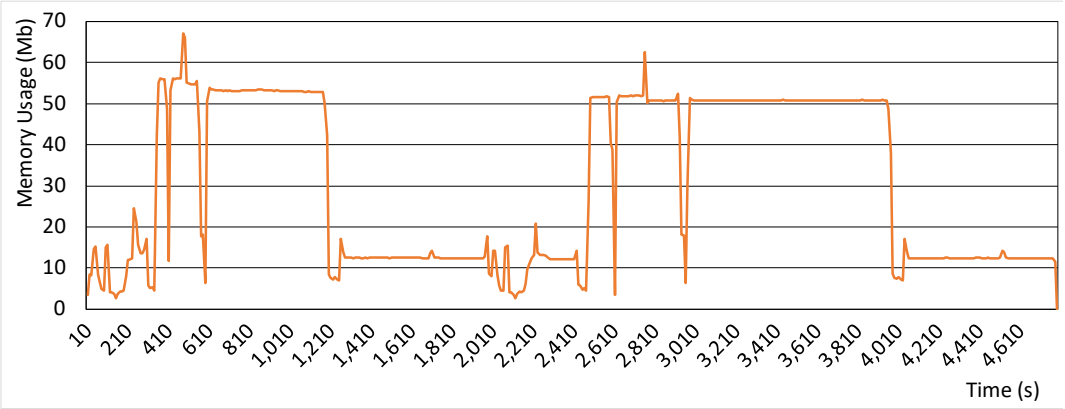


Figure 5-6 Memory usage for throughput analysis

at 6GB. Furthermore, the graph clearly shows the TCP timeout manager was able to remove all TCP connections upon expiration. For both of the throughput analysis test cases, DooR was able to handle a maximum throughput of around 4 Gbps with a 0% packet drop rate.

5.2.2.3 Evaluate DooR with core network traffic (Interop2017 event traffic)

Two of the identical nodes (Testbed 1) with the same configuration were attached to the Interop Tokyo 2017 [133] (2017/06/7-9) event, which is a leading global business technology event in Japan. During the event, we used the DooR program to analyze the event traffic on-the-fly. All traffic in the event was mirrored through the nodes where DooR program performed the functions of DPI up to application layer (L7), TCP stream reconstruction, string matching, and routing. The DooR program was configured with the snort rule set. The status core was configured to display the status periodically in 20 s.

The memory usage for the TCP stream reconstruction was measured during runtime, and the memory usage for 7,940 s (from 10:00 to 13:00 on 2017-06-08) was plotted in the graphs shown in Figure 5-7. Test results reveal that the memory usage of DooR was low (~76 Mb maximum) and the TCP timeout manager was able to release (free) the memory as expected while maintaining a low-memory profile throughout the operation. We stopped the traffic after 7,940 s and checked whether the TCP timeout manager removed the expired connections. The graph in Figure 5-7

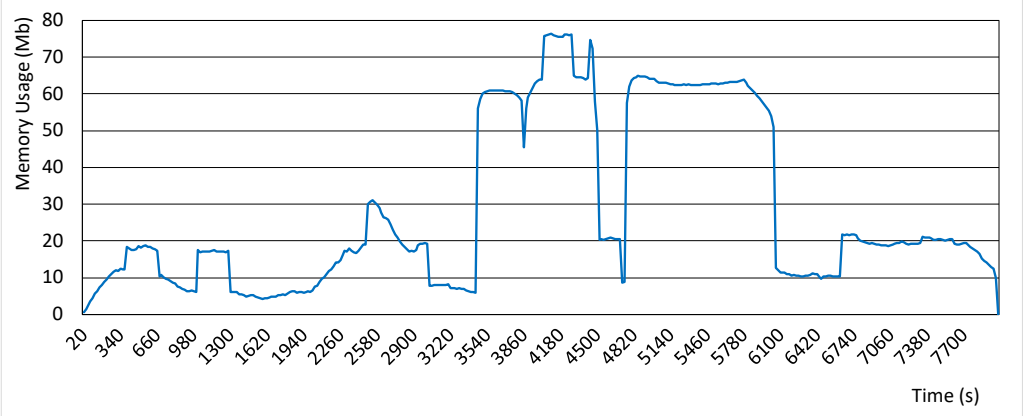


Figure 5-7 Memory usage for core network

Table 5-5 Test results: Interop 2017 event

Property	Values
Throughput (Mbps)	~300 – 1,500
Pkt/s	97,144.76
Received Packets I/O Rx Cores	671,231,540
Received Packets DPI Cores	671,231,540
No. of String Matches	141,100,429

clearly shows that the TCP timeout manager successfully removed all the expired connections. Moreover, the test results showed that the maximum memory usage of the program (for TCP stream reconstruction) for this test cases were around 76.4 Mb. The maximum memory usage for the entire system was 9 GB and averaged around 5 GB. The analyzed traffic details of the node are summarized in Table 5-5. The DooR program was configured with one reader core, four writer cores, one status core, and 16 DPI cores. During this test case, each DPI core has a reconstructed average of 550,000 streams. According to the Table 5-5 details, the DooR program was able to operate without packet drops. Therefore, we can conclude that the DooR program was able to handle core network traffic with a 0% drop rate even while performing on-the-fly TCP stream reconstruction and the string matching operations.

5.2.2.4 Evaluate DooR with campus network traffic (SINET4)

The final test scenario contained a testbed using the Testbed2 configurations in Table 5-2. This testbed was installed on one of the campus networks called SINET4 in Japan, and the traffic traces from the campus network were analyzed using this testbed. The DooR was configured with the Snort ruleset. The network traffic averaged nearly 2 Tb per hour. Similar to the previous test cases, the memory usages (for the TCP stream reconstruction process) of DooR were measured and plotted as illustrated in Figure 5-8. For this evaluation, we used traffic during the peak hours and each test evaluated for 5.4 hours.

Figure 5-8 shows the memory usage for the TCP stream reconstruction process for the campus network. The graph reveals that the maximum memory was 220 Mb. The average maximum server memory consumption values were around 14 GB. The total memory usage values were averaged

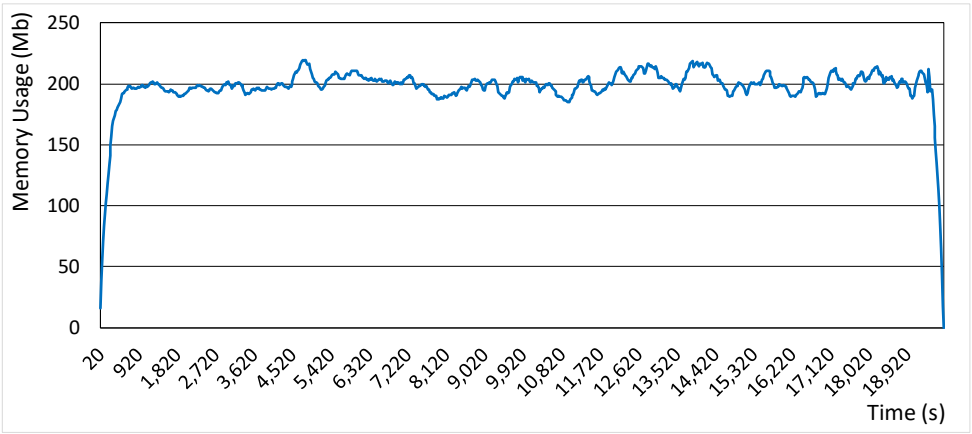


Figure 5-8 Memory usage for campus network

Table 5-6 Test results: Campus Network (SINET4)

Property	Values
Throughput (Mbps)	~500 - 2000
Pkt/s	54,726.62
Received Packets I/O Rx Cores	776,199,092
Received Packets DPI Cores	776,199,092
No. of String Matches	1,092,667,681

at 8 GB. Furthermore, the TCP timeout manager was able to free up memory, as expected, in both of the test cases upon TCP timeout expirations. In addition, the summary results from Table 5-6 shows that DooR conducted the TCP stream reconstruction, HTTP decoding, string matching, and routing processes for all the cases without any packet drop.

5.2.2.5 DooR compatibility with other implementations

For the performing DPI, research team in Hiroaki Nishi Laboratory, Keio University developed another web-based application to visualize the browsing time of the users as well as webpages viewed during the Interop events. The analysis results of the Interop event are graphically illustrated in a webpage as shown in Figure 5-9. This web-based application used the processed and saved L7 information from the DooR program. The accessed websites were ranked based on the browsing time; the top ten sites were also displayed in the web program. As shown in Figure 5-9, the web program also contained information such as the number of rules matched (number 1), a word cloud showing the matched titles in which the word size is proportional to the matched times (number 2), live stream details (number 3), search tab (number 4), HTTP access graph (number 5), interest suggestion (number 6), a browsing graph in which the circle size is

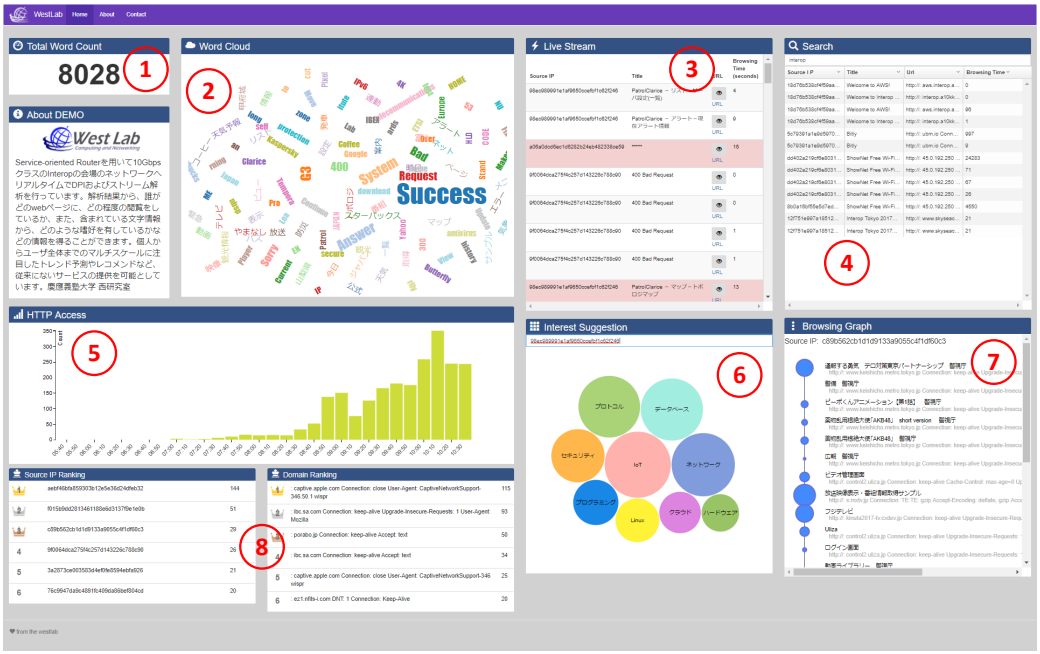


Figure 5-9 HTTP web interface used to visualize the analyzed traffic at Interop Tokyo 2017 event

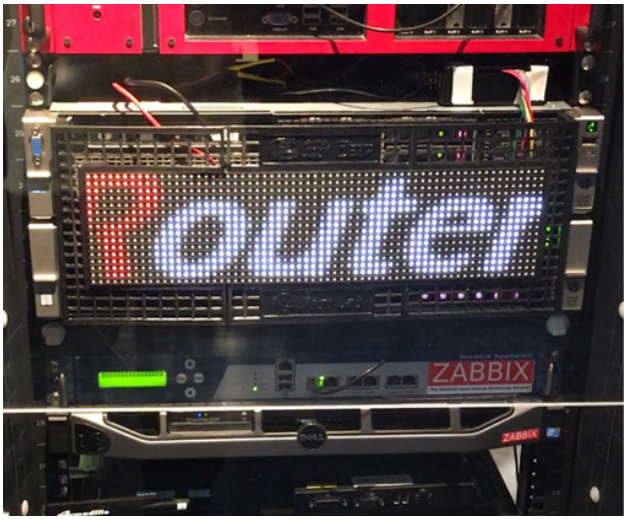


Figure 5-10 Raspberry Pi-based LCD display

proportional to the browsing time (number 7),and IP and domain ranking (number 8).

Secondly, the DooR was sending the latest updates regarding the string matching process to a Raspberry Pi-based LCD display device as denoted in Figure 5-10. This device was used to display the information such as packet count, drop rate, and the number of string matches from the DooR system on a LCD display. Finally, a HTTP-based tool was used to monitor the status of the DooR system as illustrated in Figure 5-11. As shown in Figure 5-11, this program was displaying the received information from DooR such as, packet drop ratio, packet read count, packet process count, reconstructed stream count and string matching count.

5.2.2.6 Running DooR in low performance micro PCs

During the Interop event, DooR was used to analyze the Internet traffic of the laboratory booth using an Intel NUC-based (5i7RYB) micro PC as shown in Figure 5.12. The NUC specification

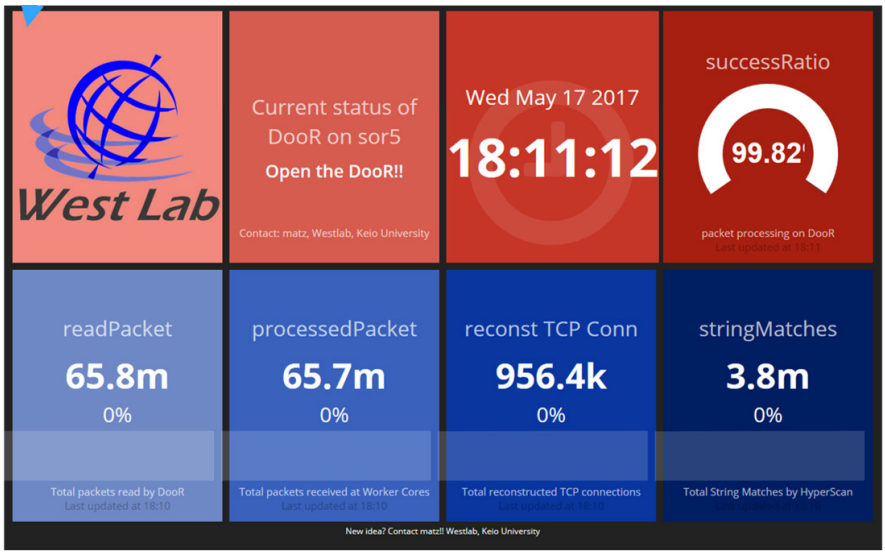


Figure 5-11 HTTP web interface used to display the status of the DooR system

Figure 5-12 Running DooR in micro PCs

was Intel Core i7-5557U processor, 16 GB memory, and 1TB SSD. The same DooR program was installed to the NUC and configured with one reader core and two worker cores. DooR program was able to run successfully analyzing all the packets and providing the output to a similar HTTP-based program as illustrated in Figure 5-9.

5.2.2.7 Results discussion

DooR is a complete software capable of performing on the fly deep packet inspection over the packets transmitting over networks. Unlike the conventional DPI solutions, DooR can perform on-the-fly TCP stream reconstruction, string matching over the packet content, extract required information and finally route the packets using its LPM-based or Patricia-Trie-based routing tables. Moreover, DooR uses the software acceleration of the DPDK library to provide high-speed packet processing with minimizing delays. DooR uses latest Intel multicore architecture in order to delegate tasks among CPU cores and to provide much efficient and scalable DPI solutions to the end users and applications over the conventional DPI solutions. There are numerous DPI solutions available where most of them are proprietary and use specific hardware devices. However, none of the existing DPI solutions to the knowledge of the authors offers such flexibility and cost efficiency as the DooR. This flexibility is brought mainly by using the Intel-based commodity hardware together with the acceleration of the open source DPDK library.

The open source nature of DooR will allow the users to alter the DooR program to tailor for their unique requirements and to the expansion of the DooR. The users can use DooR to implement high-speed DPI solutions using commodity hardware, and they can easily extend DooR to use hardware accelerated such as Intel QAT and even core processors such as Intel Xeon Phi. Finally, we can state that the DooR can be expanded for use as a routing solution, in intrusion detection and prevention systems, to provide content-based services, and to gather statistical information regarding network traffic. DooR can be used as a high performance and cost-effective packet analysis solution based on commodity hardware.

5.2.3 Conclusion

Through analyzing the test results, we can conclude that the DooR program can act as the DPI core in content-based routers while performing packet routing. Moreover, DooR succeeded in analyzing packets on the fly up to the application layer in both core and campus networks while maintaining low-memory profiles and without any packet drop. DooR can be easily used by users to analyze any types of packets in a real-time manner. It can provide DPI up to the application layer, and the analysis level can be easily configured based on user or application requirements. The users can set up, use, program, modify, and extend DooR easily according to their requirements.

Moreover, the web programs used in the Interop events concluded that the extracted information from DooR could be used by another application to provide services or to further analyze results in order to provide useful statistical information with respect to the networks and the transmitting data. Running the same version of the DooR in Intel NUC-based micro PCs demonstrated the flexibility and the scalability of the DooR system. The test results also concluded that the DooR is capable of handling a large amount of traffic even with a very large amount of string matching rules using its multicore architecture. Finally, the DooR program will open the doors to the new dimensions in packet analysis for the research community via the open source publications, and they will be able to test, run and deploy DooR using cost-effective commodity hardware with IA.

5.3 Intel Quick Assist Technology for hardware-based crypto acceleration

Nowadays, TLS acts as the backbone protocol for Internet security providing the foundation to expand the security everywhere within the network. Data security is a major service which protects the user information. However, as the security services always greedy for the resources, the main trade-off of providing appropriate security always drive by the factors; cost and performance. To this end, Intel has introduced the Intel Quick Assist Technology (QAT) [134] adaptors as cost-effective cryptographic and compression accelerators for the mainstream servers.

The main drawback of the proposed security services is also the delays caused by the process-intensive security processes such as encryption, decryption, and verification. These processes consume the extensive amount of CPU cycles to provide the security. As the proposed infrastructure use the per-hop data encryption for all the services, a proper method is required to accelerate the encryption services. To this end, this section tests and evaluates the usage of Intel QAT hardware accelerators for offloading the encryption workloads. Intel QAT accelerators are capable of accelerating compression and encryption processes offloading them from the CPUs allowing CPUs to consume their CPU cycles on other tasks. For this evaluation, the testbed 1 in Table 5.2 was attached with two Intel QAT 8950 adaptors, one per each CPU [135]. The QAT adaptors are capable of providing crypto operations in both synchronous and asynchronous modes.

The testbed was configured with different cipher operations to evaluate the performance of the CPU vs. QAT accelerators. The OpenSSL library with version 1.1.1 was used for the CPU operations (named as software in the evaluations).

5.3.1 Evaluation of crypto operations CPU vs. QAT

All the individual test cases were evaluated for three main scenarios, 1) using CPU, 2) using QAT synchronous mode, and 3) using QAT asynchronous mode. Both of the PKI and symmetric key encryption algorithms are evaluated as the proposed encryption architecture use both of the methods.

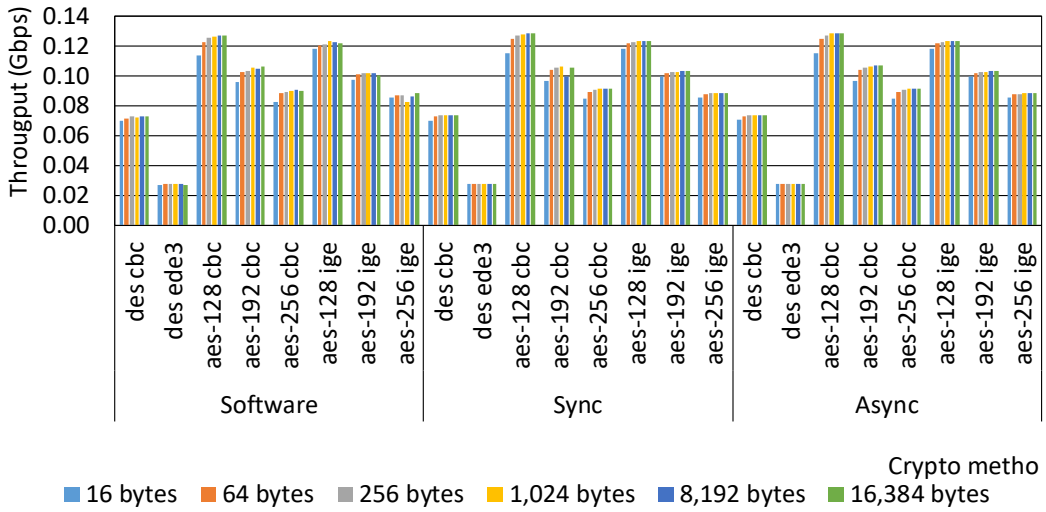


Figure 5-13 Evaluation of symmetric encryption algorithms

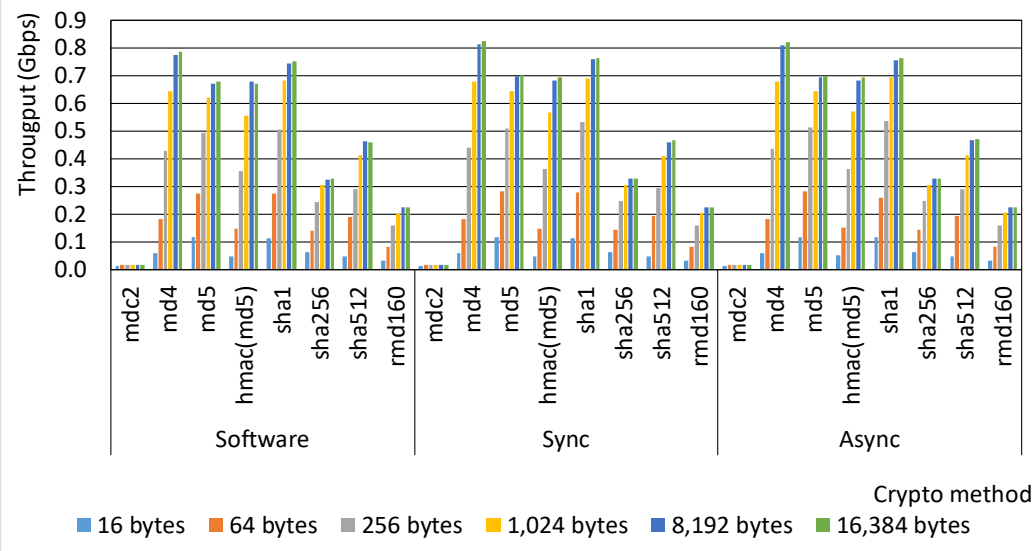


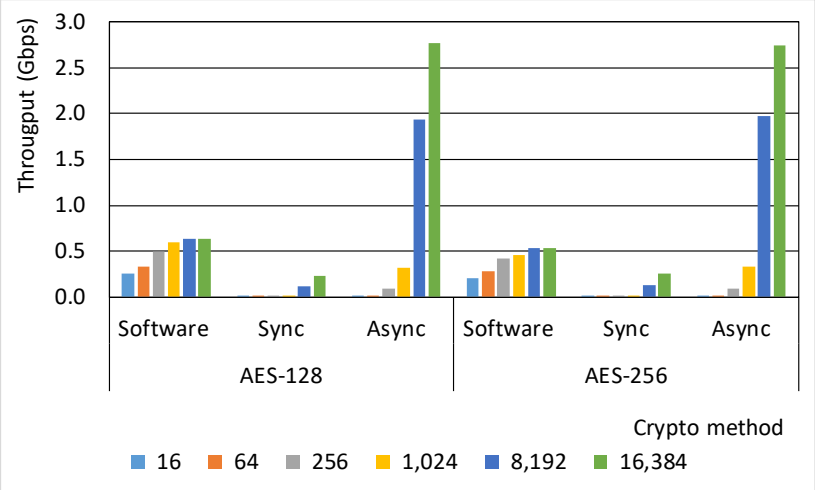
Figure 5-14 Evaluation of hash algorithms

5.3.1.1 Evaluate symmetric encryption algorithms

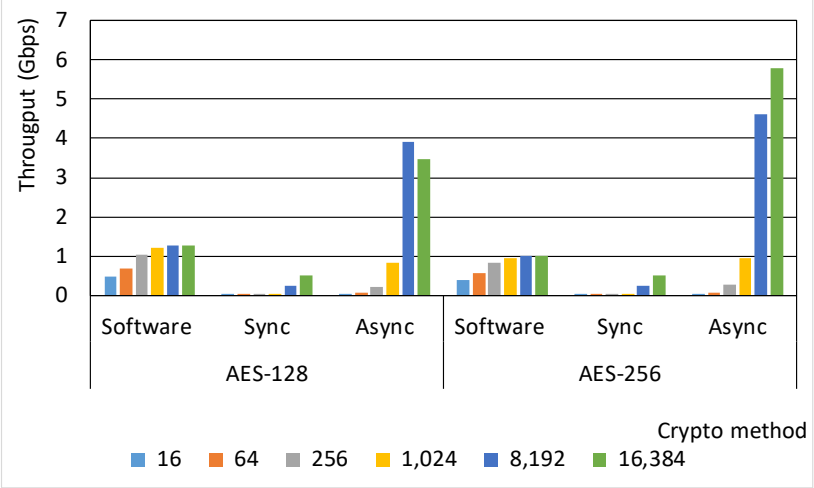
As the first evaluation, DES CBC, DES EDE3, AES-128 CBC, AES-192 CBC, AES-256 CBC, AES-128 IGE, AES-192 IGE, and AES-256 IGE symmetric encryption algorithms were measured for software, QAT synchronous, and QAT asynchronous methods. All the algorithms are evaluated for different packet sizes; 16, 64, 256, 1,024, 8,192 and 16,384 bytes. The test results are plotted in Figure 5-13.

5.3.1.2 Evaluate hash algorithms

Hash functions are widely used for digital signatures, message authentication codes (MACs), fingerprinting, and other forms of authentication, where mainly used to provide data integrity. As the second evaluation, well-known hash algorithms MDC2, MD4, MD5, HMAC (MD5), SHA1, SHA256, SHA512, and RMD160 are evaluated for different packet sizes. The test results are plotted in Figure 5-14.



(I) Using single core



(II) Using two cores

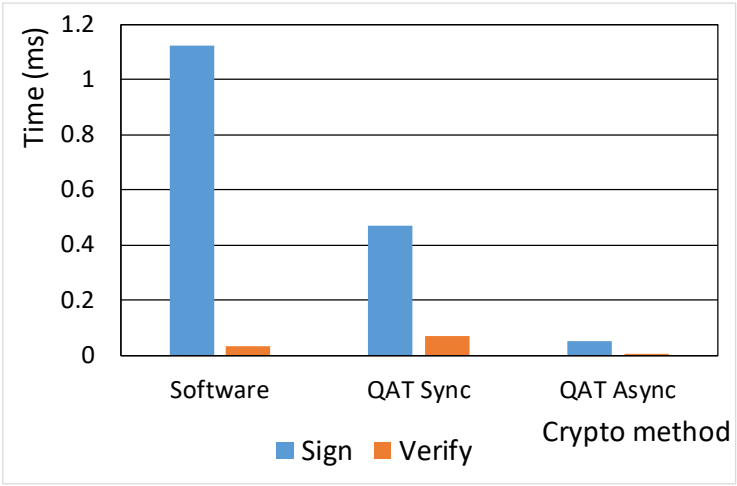
Figure 5-15 Evaluation for AES-CBC-HMAC-SHA1

5.3.1.3 Evaluate chained cipher: AES-128-CBC-HMAC-SHA1

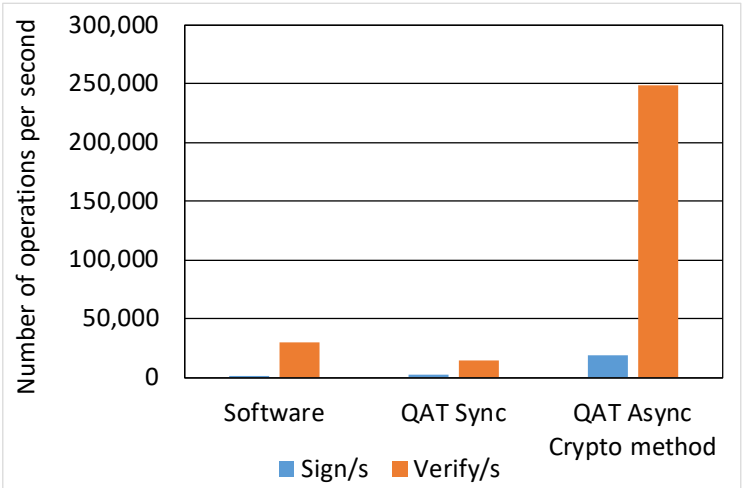
As the next evaluation, the well-known and widely used chained cipher AES-CBC algorithm is evaluated with SHA-1 (AES-CBC SHA1). The AES-CBC algorithm was evaluated for the main keyspaces which are 128-bit and 256-bit, and the test results are illustrated in Figure 5-15. Figure 5-15(I) shows the single core performance, and Figure 5-15(II) shows the performance using two cores.

5.3.1.4 Evaluate public key cryptography algorithms

As the final test case, the public key cryptography RSA algorithm was evaluated. For this evaluation, RSA encryption algorithm was configured with 2,048 keyspace. The test results were plotted and illustrated in Figure 5-16. Figure 5-16(I) shows the time consumption (milliseconds) for the sign and verification processes for each method. Figure 5-16(II) shows the performance of the each method, sign and verifications per second, of the RSA algorithm.



(I) RSA time consumption for sign and verify



(II) RSA performance for sign and verify

Figure 5-16 Evaluation of RSA algorithm

5.3.1.5 Results discussion

According to the first and second evaluations, the asynchronous mode of the QAT adaptor shows the best performance for almost all the symmetric encryption and hash algorithms. Moreover, the third test case clearly demonstrates the performance of the asynchronous mode for both the AES-CBC-HMAC-SHA1 128-bit and 256-bit keyspaces. Finally, the evaluation for the RSA cryptosystem shows a huge increase in sign and verification performance. The asynchronous mode was also capable to perform extremely higher number of signs and verifications per second compared with the software-based method.

5.3.2 Conclusion

The encryption, decryption, verification, sign, and hash calculation tasks are highly process intensive operations while providing security. This section evaluates the Intel QAT hardware accelerators which can use to offload the crypto workloads in the content-based routers. Both of the synchronous and asynchronous modes of the QAT adaptor is evaluated against the conventional software CPU based crypto operations. The test results demonstrate a significant performance gain when utilizing a high-performing asynchronous engine to efficiently manage the processing flow. For asymmetric cryptographic algorithms such as RSA-2K, there is an average demonstrable performance gain of 20 times per sign and 8 times for verification while using the asymmetric mode. Furthermore, the hardware-based acceleration is much efficient when using cipher suits such as AES-CBC-HMAC-SHA1, where it combines the encryption with hash algorithms to provide both data privacy and integrity. The test results clearly conclude that the asynchronous mode of the QAT adaptor is significantly outperform the software-based method. Moreover, by using the QAT adapter CPUs can entirely use their processing cycles for other tasks while asynchronously offloading the process intensive crypto tasks. The cost-efficient QAT adaptors will allow the content-based router applications to adopt these features and opens a significant potential to increase SSL/TLS performance more generally.

Chapter 6 Conclusion and future vision

6.1 Discussion

This study has proposed a novel secured network infrastructure using content-based routers. All the services implemented and evaluated using the ns-3 simulator. Per-hop data encryption protocol was implemented and used as the security foundation of the proposed architecture. This was achieved by merging the DLSR protocol and the per-hop data encryption service. Currently, ns-3 has numerous wireless routing protocols for MANETs such as AODV [66], DSDV [67], DSR [68], and OLSR [69]. Moreover, [67] and [72] are the only currently available dynamic distance vector routing protocols in ns-3. DLSR protocol was implemented from the scratch due to the fact that ns-3 lacks in a dynamic link-state routing protocol for wired networks. Unlike the conventional link state routing protocol OSPF [51], DLSR protocol was capable of distributing the route calculation process to all the routers. Moreover, DLSR uses both link bandwidth and the link propagation delay in the route metric calculation process. This will result in selecting the highest throughput links with the least delays while packet routing. Furthermore, via integrating the proposed services to a link-state routing protocol demonstrates the compatibility to use the proposed infrastructure with the existing routing infrastructure.

The proposed per-hop data encryption protocol is capable of mitigating the exit node vulnerability via encrypting the packets from the source to the destination in per-hop manner. The users can either select the security level they prefer by themselves or allow the content-based router to perform the key generation. If the user defines the security level, the content-based routers will route all the packets of that particular user encrypting them using appropriate encryption algorithms among the hops meeting the user requirements. If any intermediate hop required much higher encryption, the content-based routers will increase the security accordingly among the appropriate hops. This will secure the client data from using weak crypto parameters via dynamically extending the security according to the data traversing network requirements. Conventional methods will not be able to provide such dynamic security to the data.

The proposed entire packet encryption service will secure the sensitive data of the packet headers together with the payloads. Yet, it allows the packets IP routable to allow any intermediate devices to route the packets. Compared with the conventional retransmission methods, the neighbor retransmission service can used to reduce the retransmission delays drastically. Moreover, the proposed traceability service provides faster, secure and flexible traceability to any type of packets. The encrypted traceability packets will secure the trace data from any unauthorized access or modifications and the packets can be verified using the CA architecture. All of these services inherits the above mentioned advantages and the flexibilities of the underlying per-hop data encryption service.

DooR system provides a real-world environment which can used as the core software for the content-based routers to implement the above simulated services in the real-world. DooR performs on-the-fly TCP stream reconstruction, HTTP packet decoding (Gzip decompression and chunk decoding), HyperScan-based high throughput string matching and packet routing. It uses both software and hardware acceleration to provide the above functionalities tolerating higher data throughputs. DooR use context switching for all the above functionalities without storing any packet content which result a smaller memory footprint for the stream reconstruction process. Running the DooR system in Interop events showed the robustness of DooR in analyzing real-world core network traffic. Connecting with third party software showed the interoperability of the DooR and proved that the extracted information can be used to derive valuable information. Moreover, running the DooR system in micro PCs demonstrates the scalability of the system. Also DooR runs on conventional PCs with Intel architecture. This makes it a cost effective and flexible solution for constructing powerful content-based routers.

Finally, the evaluation of the Intel QAT accelerators showed the crypto overheads of the proposed services can be offloaded to the QAT adaptors to process crypto workloads faster and efficiently. This offloading process will allow the DooR to tolerate higher Gbps throughputs by saving the CPU cycles allowing the CPUs to focus and perform other tasks such as DPI and string matching.

6.2 Conclusions

This study has proposed, implemented and evaluated a secured network infrastructure using content-based routers. This work proposed addressing three major requirements for the secured infrastructure by providing:

1) proper flexible and efficient security protocol as the infrastructure foundation

2) secured services to address entire packet encryption, recovering corrupted packets, and efficient traceability service

3) a practical core software to operate as the core software for content-based routers which can provide above services to the real-world using the secured infrastructure.

In order to reach the above goals, the study proposed, implemented and evaluated the following six portions which integrate into the proposed secured infrastructure.

1) Link-state routing protocol

2) Per-hop data encryption protocol

3) IP routable entire packet encryption service

4) Neighbor data retransmission service

5) Fast and secure packet traceability service

6) On-the-fly deep packet inspection solution as the content-based router core software

The first five portions were implemented and evaluated using the well-known network simulator ns-3, and the last implementation was carried out in a real-world testbed. Initially, the distributed link-state routing protocol was implemented extending the Dijkstra's algorithm adding the link propagation delay to the metric calculation. The test results concluded that the proposed DLSR protocol has 60% less throughput conception and 20% better average delay variance over larger networks compared with the conventional OSPF protocol. Moreover, the convergence tests resulted that the proposed protocol was successfully able to recover in case of link failures.

A per-hop based data encryption service was proposed as the underlying security protocol for the proposed infrastructure. The per-hop encryption only consumed 0.03 ms in packet propagation delay where it was only a mere 0.08% increment over the conventional end-to-end encryption methods. However, the test results showed that the proposed service could be used with different encryption algorithms together with keyspaces among the links through the data propagation paths.

Afterward, the DLSR protocol was merged with the per-hop data encryption service as the per-hop data encryption protocol. It allowed the secured services to access the routing layer and provide security to the routing layer as well. Moreover, it was further attached with a CA architecture to enhancing the verification, integrity, and non-repudiation of the proposed infrastructure. By using this enhanced infrastructure, the IP routable entire-packet encryption service was proposed and implemented. This service was able to totally mitigate one of the main concerns of the conventional secured protocols, namely the exit node's vulnerability. Furthermore, according to the test results, the proposed service was able to secure the sensitive information of the IP headers yet allowing them to be IP routable. The test results concluded that the proposed method was flexible to use with encryption algorithms such as IDEA, DES, AES-GCM, and AES-CTR while consuming μs level encryption and decryption overheads which are much lower than the 150 ms tolerable delay defined by the ITU-T recommendation G.114.

These encryption and decryption delays were further reduced by using the Intel QAT accelerators. The test results showed that the QAT adaptors were able to provide significant performance improvements via the asynchronous mode operations over the software-based cryptosystems.

The neighbor data retransmission service was proposed to reduce the data retransmission delays over the public networks due to data corruption. The test results concluded that the proposed neighbor retransmission method was able to reduce the retransmission delay by 80% compared to the conventional ARQ method. As the final service, a fast and secure packet traceability service was proposed and implemented. The test results concluded that the proposed service was able to provide flexible traceability details according to the user or application requirements. Furthermore, it was able to provide 79% faster performance over the conventional traceability method. Furthermore, according to the test results of the hardware-based Intel QAT crypto accelerators, these encryption and decryption delays of the proposed services can be extensively reduced by using the Intel QAT accelerators in content-based routers.

Finally, the DooR system was proposed as the on-the-fly DPI solution for the content-based routers. The DooR system was deployed and evaluated extensively in the real-world campus (SINET4) and core networks (Interop 2016 and 2017 events). The test results concluded that the DooR system used a smaller memory footprint to its on-the-fly stream reconstruction processes via context switching. Moreover, integration with the third party software during the Interop events concluded that the DooR could be successfully integrated and used by third-party software to derive meaningful on-the-fly stream analysis data to provide extended services to the users. During running for three consecutive days inside each Interop event demonstrated the robustness and the reliability of the DooR system to act as the content-based router core software. Finally, running the same DooR version in the Intel NUC micro PC proved the flexibility and the scalability of the proposed DooR system.

In conclusion, the proposed secured network infrastructure uses the per-hop data encryption protocol as the encryption protocol, and the test results proved that the proposed infrastructure could successfully provide services such as IP routable entire packet encryption, neighbor packet retransmission, and fast and secure packet traceability services. Moreover, DooR can be used to implement the above-proposed security service to the real-world content-based routers and compensate the encryption and decryption delays using its software and hardware accelerations.

6.3 Future vision

Cybersecurity is a never-ending journey while the security counter methods will grow together protecting the network infrastructures. Requirements from the users will rise for the content-based services from the network infrastructures. The exponential development of hardware and software infrastructures will boost the performance of the devices to the very end pushing them to achieve tasks much faster and much efficiently. Therefore, the content-based routers will take advantage of such high-performance software and hardware in order to provide content-based services residing inside the very core of the network infrastructures.

By leveraging the network infrastructure to such an extent, will make the prototype secured infrastructure implemented in this dissertation to perform beyond the Gigabit Ethernet networks. As the world is migrating to IPv6 infrastructure, in the future, the proposed infrastructure will be extended to IPv6 infrastructure followed by testing, evaluation, and deployment in the real-world networks. The implemented DooR system will be extended to implement by using efficient programming languages such as C++ and Rust. It will make the C-based current implementation much easier to maintain, troubleshoot and most of all memory efficient. The DooR will be extended to use Intel QAT accelerators to offload the compression and encryption workloads to gain hardware-based acceleration for such tasks. Moreover, the proposed infrastructure will be extended to perform authorized secured stream analysis, where it will be able to analyze authorized secured data streams such as authorized SSL and TLS sessions.

References

[1] "Babbage's Analytical Engine, 1834-1871. (Trial model)," Science Museum Group Collection, 1834-1871. [Online]. Available: http://collection.sciencemuseum.org.uk/objects/co62245/babbages-analytical-engine-1834-1871-trial-model-analytical-engines. [Accessed 12 12 2018].

[2] J. C. R. Licklider, "Memorandum For Members and Affiliates of the Intergalactic Computer Network," Kurzweilai.net, 11 12 2001. [Online]. Available: http://www.kurzweilai.net/memorandum-for-members-and-affiliates-of-the-intergalactic-computer-network. [Accessed 12 12 2018].

[3] Kleinrock L., and Lam S., "Packet Switching in a Multiaccess Broadcast Channel: Performance Evaluation," in *IEEE Transactions on Communications*, 1975.

[4] Kleinrock, L., Communication nets, New York: MacGraw-Hill, 1964.

[5] Wesley, C., "ARPANET IMP, Interface Message Processor," Livinginternet.com, 2000. [Online]. Available: https://www.livinginternet.com/i/ii_imp.htm. [Accessed 12 12 2018].

[6] Barry. M, "Brief History of the Internet - Internet Society," Internet Society, 1997. [Online]. Available: https://www.internetsociety.org/internet/history-internet/brief-history-internet/. [Accessed 12 12 2018].

[7] Postel, J., "NCP/TCP TRANSITION PLAN," IETF RFC 801, 11 1981. [Online]. Available: https://tools.ietf.org/html/rfc801.

[8] Spicer, D., "Raymond Tomlinson: Email Pioneer," in *IEEE Annals of the History of Computing*, 2016.

[9] Weimann, G., "Terror on the Internet," Washington D.C, 2006.

[10] "Transmission Control Protocol," IIETF RFC 793, 9 1981. [Online]. Available: https://tools.ietf.org/html/rfc793.

[11] Information Sciences Institute - University of Southern California, "Internet Protocol," IETF RFC 791, 9 1981. [Online]. Available: https://tools.ietf.org/html/rfc791.

[12] Socolofsky, T., and Kale, C., "A TCP/IP Tutorial," IETF RFC 1180, 9 1981. [Online]. Available: https://tools.ietf.org/html/rfc791.

[13] "OSI Routing Framework," ISO/IEC TR 9575:1995, 1995. [Online]. Available: https://www.iso.org/standard/25981.html. [Accessed 12 12 2018].

[14] "Google search statistics," internetlivestats, 7 2016. [Online]. Available: http://goo.gl/nz44XD. [Accessed 12 12 2018].

[15] Chambers, C., Dolske, J. and Iyer, J., "TCP/IP Security," linuxsecurity.com, 2018. [Online]. Available: http://www.linuxsecurity.com/resource_files/documentation/tcpipsecurity.html. [Accessed 12 12 2018].

[16] B. Guha and B. Mukherjee, "Network security via reverse engineering of TCP code: vulnerability analysis and proposed solutions," in *IEEE Network*, 1997.

[17] "Number of Internet Users," Internetlivestats.com, 2016. [Online]. Available:

http://www.internetlivestats.com/internet-users/. [Accessed 12 12 2018].

[18] Stapleton, J., Security without obscurity, 1 ed., CRC Press, 2014.

[19] Symantec.com, "Internet Security Threat Report," 2018. [Online]. Available: https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf. [Accessed 12 12 2018].

[20] Roberts, L., "Internet Still Growing Dramatically Says Internet Founder," Prnewswire.com, [Online]. Available: http://www.prnewswire.com/news-releases/internet-still-growingdramatically-says-internet-founder-71733967.html. [Accessed 12 12 2018].

[21] V. Annovazzi-Lodi, G. Aromataris, M. Benedetti and S. Donati, "Secure transmission network using chaotic lasers," in *Complexity in Engineering (COMPENG)*, Barcelona, 2014.

[22] S. Shalunov and B. Teitelbaum, "Bulk TCP Use and Performance on Internet2," 8 2001. [Online]. Available: http://www.internet2.edu/abilene/tcp/i2-tcp.pdf. [Accessed 12 12 2018].

[23] Ager, B. et al., "Revisiting cacheability in times of user generated content," in *IEEE Global Internet Symposium*, 2010.

[24] "Media Access Control (MAC) Security," 2011. [Online]. Available: http://standards.ieee.org/getieee802/download/802.1AEbn-2011.pdf. [Accessed 12 12 2018].

[25] Freier, A., Karlton, P., and Kocher, P., "The Secure Sockets Layer (SSL) Protocol Version 3.0," IETF RFC 6101, 08 2011. [Online]. Available: http://tools.ietf.org/html/rfc6101.

[26] Dierks, T., and Dierks, T., "Transport Layer Security (TLS) Protocol Version 1.2," IETF RFC 5246, 8 2008. [Online]. Available: http://tools.ietf.org/html/rfc5246.

[27] Hamzeh, K., et al., "Point-to-Point Tunneling Protocol (PPTP)," IETF RFC 2637, 7 1999. [Online]. Available: http://tools.ietf.org/rfc/rfc2637.txt.

[28] Frankel, S. and Krishnan, S., "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," IETF RFC 6071, 2 2011. [Online]. Available: http://tools.ietf.org/html/rfc6071. [Accessed 12 12 2018].

[29] K. Inoue and H. Nishi, "Semantic router using data stream to enrich services," Keio Univ, 2009. [Online]. Available: http://goo.gl/GmHnnV. [Accessed 12 12 2018].

[30] Janaka Wijekoon, Erwin Harahap, Hiroaki Nishi, "Service-oriented Router Simulation Module Implementation in NS2 Simulator," in *Procedia Computer Science*, 2013.

[31] Takagiwa, Kenichi; Ishida, Shinichi; Nishi, Hiroaki, "SoR-Based Programmable Network for Future Software-Defined Network," in *Computer Software and Applications Conference (COMPSAC)*, 2013.

[32] Yusuke Nishida, and Hiroaki Nishi, "Implementation of a Hardware Architecture to Support High-speed Database Insertion on the Internet," in *The 2012 International Conference on Engineering of Reconfigurable Systems and Algorithms (ERSA2012 in WORLDCOMP2012)*, 2012.

[33] "Application extension platform (axp)," Cisco, 7 2016. [Online]. Available: https: //goo.gl/Ul3afV. [Accessed 12 12 2018].

[34] J. Kelly, W. Araujo, and K. Banerjee, "Rapid service creation using the junos sdk," in *2Nd ACM SIGCOMM Workshop on Programmable Routers for Extensible Services of Tomorrow*, 2009.

[35] "Cisco ace 4700 series application control engine appliance," Cisco, 7 2016. [Online]. Available: http://goo.gl/qCBrhe. [Accessed 12 12 2018].

[36] "Session border controller - cisco 7600 series sbc," Cisco, 7 2016. [Online]. Available: http://goo.gl/bST5b3. [Accessed 12 12 2018].

[37] "Cisco ace application control engine module," Cisco, 7 2016. [Online]. Available:

http://goo.gl/0gGCRN. [Accessed 12 12 2018].

[38] "Cisco ace 4700 series application control engine appliance," Cisco, 7 2016. [Online]. Available: http://goo.gl/qCBrhe. [Accessed 12 12 2018].

[39] Higginbotham, S., "In a distributed world cache is king. why routers are becoming the new server," 7 2016. [Online]. Available: https://goo.gl/bVDvOv. [Accessed 12 12 2018].

[40] "Develop IoT business applications at the edge," Cisco , 2018. [Online]. Available: https://www.cisco.com/c/en/us/products/cloud-systems-management/iox/index.html. [Accessed 12 12 2018].

[41] "Internetofeverything," 7 2016. [Online]. Available: http://goo.gl/o6cYbK.

[42] "Hitachi to launch wan accelerator family for dramatically faster data transfer among global Offices," Hitachi, 7 2016. [Online]. Available: http://goo.gl/8M1WlY. [Accessed 12 12 2018].

[43] "Akamai and qualcomm in pact for ultrahd streaming," Beet.TV, 7 2016. [Online]. Available: http://goo.gl/K6raLy. [Accessed 12 12 2018].

[44] "VyOS - an Open Source Linux-based Network OS," Vyos.io, 2018. [Online]. Available: https://vyos.io/. [Accessed 12 12 2018].

[45] "Home - DPDK," DPDK, 2018. [Online]. Available: https://www.dpdk.org/.

[46] "Intel® Ethernet Converged Network Adapter XL710 10/40 GbE," Intel, [Online]. Available: https://www.intel.com/content/www/us/en/ethernet-products/converged-network-adapters/ethernet-xl710-brief.html. [Accessed 12 12 2018].

[47] "Intel® QuickAssist Technology (Intel® QAT) Improves Data Center," Intel, [Online]. Available: http://www.intel.com/content/www/us/en/embedded/technology/quickassist/overview.html. [Accessed 12 12 2018].

[48] "Hyperscan," Intel, 2018. [Online]. Available: https://01.org/hyperscan. [Accessed 12 12 2018].

[49] K. Inoue, D. Akashi, M. Koibuchi, H. Kawashima and H. Nishi, "Semantic router using data stream to enrich services," in *3rd International Conference on Future Internet Technologies (CFI08)*, Seoul, 2008.

[50] "Enhanced Interior Gateway Routing Protocol draft-savage-eigrp-02.txt," Network Working Group (Internet draft), 4 2014. [Online]. Available: http://tools.ietf.org/html/draft-savage-eigrp-02. [Accessed 12 12 2018].

[51] Moy J., "OSPF Version 2," IETF RFC 2328, 4 1998. [Online]. Available: http://www.ietf.org/rfc/rfc2328.txt.

[52] Routing protocols companion guide, India: Cisco Press, 2014.

[53] Abraham J., "ns-3 manual," 2013. [Online]. Available: http://www.nsnam.org/docs/manual/singlehtml/index.html. [Accessed 12 12 2018].

[54] Kahn, D, The codebreakers, New York: Scribner, 1996.

[55] Katz, J., Menezes, A., Oorschot, P. and Vanstone, S., Handbook of applied cryptography, 1 ed., Boca Raton: CRC Press, 1996.

[56] National Institute of Standards and Technology, "Data Encryption Standard (DES)," in *FIPS PUB*, 1999.

[57] Rescorla, E., "Diffie-Hellman Key Agreement Method," IETF RFC 2631, 7 1999. [Online]. Available: https://www.ietf.org/rfc/rfc2631.txt.

[58] Kaliski, B., "RSA Encryption Version 1.5," IETF RFC 2313, 3 1998. [Online]. Available: https://tools.ietf.org/html/rfc2313.

[59] Ylonen, T. and Lonvick, C., "The Secure Shell (SSH) Transport Layer Protocol," IETF RFC 4253, 1 2006. [Online]. Available: http://tools.ietf.org/html/rfc4253.

[60] Frankel, S. and Krishnan, S., "IP Security (IPsec) and Internet Key Exchange (IKE) Document Roadmap," IETF RFC 6071, 2 2011. [Online]. Available: http://tools.ietf.org/html/rfc6071.

[61] Kohl, J., and Neuman, C., "The Kerberos Network Authentication Service (V5)," IETF RFC 1510, 9 1993. [Online]. Available: https://tools.ietf.org/html/rfc1510.

[62] Dierks, T., and Dierks, T., "Transport Layer Security (TLS) Protocol Version 1.2," IETF RFC 5246, 8 2008. [Online]. Available: http://tools.ietf.org/html/rfc5246.

[63] "Signal Specifications," Signal.org, 2013. [Online]. Available: https://signal.org/docs/. [Accessed 12 12 2018].

[64] Simpson, W., "The Point-to-Point Protocol (PPP)," IETF RFC 1661, 7 1994. [Online]. Available: https://tools.ietf.org/html/rfc1661.

[65] "Enhanced Interior Gateway Routing Protocol," Cisco, 1 2015. [Online]. Available: http://www.cisco.com/c/en/us/support/docs/ip/enhanced-interior-gateway-routing-protocol-eigrp/16406-eigrp-toc.html#eigrpmetrics. [Accessed 12 12 2018].

[66] Hedrick C., "Routing Information Protocol," IETF RFC 1058, 7 1988. [Online]. Available: https://www.ietf.org/rfc/rfc1058.txt.

[67] G., Malkin, "RIP Version 2," IIETF RFC 2453, November 1998. [Online]. Available: https://tools.ietf.org/html/rfc2453.html.

[68] Perkins C.; Belding-Royer E.; Das S., "Ad hoc On-Demand Distance Vector (AODV) Routing," IETF RFC 3561, July 2003. [Online]. Available: http://www.rfc-base.org/txt/rfc-3561.txt.

[69] Khan, K.; Zaman, R.U.; Reddy, K.A.; Reddy, K.A.; Harsha, T.S., "An Efficient DSDV Routing Protocol for Wireless Mobile Ad Hoc Networks and its Performance Comparison," in *Second UKSIM European Symposium on Computer Modeling and Simulation*, 2008.

[70] Johnson D., Hu Y., and Maltz D., "The Dynamic Source Routing Protocol (DSR) for Mobile Ad Hoc Networks for IPv4," IETF RFC 4728, 2 2007. [Online]. Available: https://www.ietf.org/rfc/rfc4728.txt. [Accessed 12 12 2012].

[71] Clausen T.; Jacquet P., "Optimized Link State Routing Protocol (OLSR)," IETF RFC 3626, 10 2003. [Online]. Available: https://www.ietf.org/rfc/rfc3626.txt.

[72] G., Malkin, Xylogics, R., Minnear, "RIPng for IPv6," IIETF RFC 2080, January 1997. [Online]. Available: https://tools.ietf.org/html/rfc2080.html.

[73] Harris, S., CISSP All-in-One Exam Guide, McGraw-Hill Companies, 2008.

[74] "Tor Project," Torproject.org, [Online]. Available: https://www.torproject.org/about/overview. [Accessed 12 12 2018].

[75] Haraty, R. and Zantout, B., "The TOR data communication system," *Journal of Communications and Networks,* vol. 16, no. 4, pp. 415-420, 2014.

[76] M. G. Reed, P. F. Syverson and D. M. Goldschlag, "Anonymous connections and onion routing," *IEEE Journal on Selected Areas in Communications,* vol. 16, no. 4, pp. 482-494, 1998.

[77] Tiwari, A., "Everything About Tor: What is Tor? How Tor Works?," Fossbytes, [Online]. Available: https://fossbytes.com/everything-tor-tor-tor-works/. [Accessed 12 12 2018].

[78] R. Tennekoon, J. Wijekoon and H. Nishi, "On the Effectiveness of IP-Routable Entire-Packet Encryption Service over Public Networks," *IEEE ACCESS,* vol. 6, no. 1, pp. 73170-73179, 11 2018.

[79] Beyah, R., McNair, J. and Corbett, C., Security in Ad-hoc and Sensor Networks, 3 ed., World

Scientific Publishing Company, 2009.

[80] Tennekoon, R., Wijekoon, J., Harahap, E. and Nishi, H., "Prototype implementation of fast and secure traceability service over public networks," *IEEJ Trans Elec Electron Eng,* vol. 11, pp. 122-133, 2016.

[81] "Transmission Control Protocol," IETF RFC 793, 9 1981. [Online]. Available: http://tools.ietf.org/html/rfc793.

[82] Paxson, V., Allman, M., "Computing TCP's Retransmission Timer," IETF RFC 2988, 11 2000. [Online]. Available: http://tools.ietf.org/html/rfc2988.

[83] Wu, H.; Zheng, J., "Efficient network coding-based multicast retransmission mechanism for mobile communication networks," in *IET Communications*, 2012.

[84] Li, Z.; Sun, L.; Qiao, Z.; Ifeachor, E., "Perceived speech quality driven retransmission mechanism for wireless VoIP," in *International Conference on 3G Mobile Communication Technologies*, 2003.

[85] Tanigawa, Y.; Jong-Ok Kim; Tode, H., "Delay-Sensitive Retransmission Method Based on Network Coding in IEEE 802.11 Wireless LANs," in *Global Telecommunications Conference (GLOBECOM 2010)*, 2010.

[86] R. Tennekoon, J. Wijekoon, E. Harahap and H. Nishi, "Previous hop data retransmission service for SoR-based public networks," in *7th International Conference on Information and Automation for Sustainability*, Colombo, 2014.

[87] Bellovin, S., " ICMP traceback messages," IETF draft, 2003. [Online]. Available: https:// tools.ietf.org/html/draft-ietf-itrace-04.

[88] Mankin A, Massey D,Wu CL,Wu SF, Zhang L., "On design and evaluation of 'intention-driven icmp traceback," in *10th International Conference on Computer Communications and Networks (IC3N'2001)*, Arizona, 2001.

[89] Savage S, Wetherall D, Karlin A, Anderson T., "Practical network support for IP traceback," in *IEEE/ACM Transactions on Network 2001*, 2001.

[90] Kihong P, Heejo L., "On the effectiveness of probabilistic packet marking for IP traceback under denial of service attack," in *IEEE INFOCOM*, 2001.

[91] Adler M., "Tradeoffs in probabilistic packet marking for IP traceback," in *Proceedings of the Theory of Computing*, 2002.

[92] Stone R., "Centertrack: an IP overlay network for tracking DoS floods," in *9th USENJX Security Symposium*, 2000.

[93] "Internet Users," internetlivestats, 2014. [Online]. Available: http://www.internetlivestats.com/internet-users/. [Accessed 12 12 2018].

[94] Callahan, T., Allman, M., and Paxson, V., "A longitudinal view of http traffic," in *Conference on Passive and Active Measurement (PAM)*, 2010.

[95] Doverspike, R., and Gerber, A., "Traffic Types and Growth in Backbone Networks," in *OFC/NFOEC (invited paper)*, 2011.

[96] Erman, J., Gerber, A., Hajiaghayi, M. T., Pei, D., and Spatscheck, O., "Network aware forward caching," in *International World Wide Web Conference (WWW)*, 2009.

[97] Labovitz, C. et al., "Internet inter-domain traffic," in *ACM SIGCOMM Conference*, 2010.

[98] Maier, G., et al., "On dominant characteristics of residential broadband internet traffic," in *Internet Measurement Conf. (IMC)*, 2009.

[99] Schneider, F., et al., "The new web: characterizing ajax traffic," in *Conference on Passive and*

Active Measurement (PAM), 2008.

[100] Schneider, F., et al., "Understanding online social network usage from a network perspective," in *Internet Measurement Conf. (IMC)*, 2009.

[101] "Wireshark homepage," Wireshark, 2014. [Online]. Available: https://www.wireshark.org/. [Accessed 12 12 2018].

[102] "TCP/IP packet demultiplexer," TCPFLOW, 2015. [Online]. Available: https://github.com/simsong/tcpflow. [Accessed 12 12 2018].

[103] "Justniffer-grab-http-traffic," JUSTNIFFER, 2014. [Online]. Available: http://justniffer.sourceforge.net/#!/justniffer_grab_http_traffic. [Accessed 12 12 2018].

[104] "TCP Session Reconstruction Tool," Code Project, 2007. [Online]. Available: http://www.codeproject.com/Articles/20501/TCP-Session-Reconstruction-Tool. [Accessed 12 12 2018].

[105] Takano Y., Miura R., and Yasuda S., "SF-TAP: Scalable and Flexible Traffic Analysis Platform Running on Commodity Hardware," [Online]. Available: https://www.usenix.org/conference/lisa15/conference-program/presentation/takano[Accessed. [Accessed 12 12 2018].

[106] "Snort home page," Snort Project, 2007. [Online]. Available: https://www.snort.org/. [Accessed 12 12 2018].

[107] Chung J., Claypool M., "NS by Example," 2011. [Online]. Available: http://en.wikipedia.org/wiki/NS-2. [Accessed 12 12 2018].

[108] "Main page - Crypto++® Library 8.0," 12 2018. [Online]. Available: https://www.cryptopp.com/wiki/Main_Page. [Accessed 12 12 2018].

[109] "Dijkstra's algorithm," Rosettacode.org, 24 11 2018. [Online]. Available: http://rosettacode.org/wiki/Dijkstra's_algorithm. [Accessed 12 12 2018].

[110] Stroustrup B., The C++ Programming Language, 3 ed., Boston: Longman Publishing Co., Inc., 2000.

[111] Postel, J., "User Datagram Protocol," IETF RFC 768, 8 1980. [Online]. Available: https://tools.ietf.org/html/rfc768.

[112] Spring, N., Mahajan, R., Wetherall, D., Anderson, T., "Measuring ISP topologies with rocketfuel," in *IEEE/ACM Transactions on Networking (TON)*, 2004.

[113] "Rocketfuel internet topology database," Department of Computer Science & Engineering University of Washington, [Online]. Available: http://www.cs.washington.edu/research/networking/rocketfuel. [Accessed 12 12 2018].

[114] "Tracemetrics Team," Trace Metrics Homepage, 2013. [Online]. Available: http://www.tracemetrics.net/. [Accessed 12 12 2018].

[115] "Measuring Network Convergence Time," ixia, 2014. [Online]. Available: https://support.ixiacom.com/sites/default/files/resources/whitepaper/convergence.pdf. [Accessed 12 12 2018].

[116] "Next generation encryption," Cisco, 2012. [Online]. Available: http://www.cisco.com/web/about/security/intelligence/nextgen_crypto.html. [Accessed 12 12 2018].

[117] "Crypto++ random number generator," [Online]. Available: http://www.cryptopp.com/wiki/RandomNumberGenerator. [Accessed 12 12 2018].

[118] "One-way transmission time ITU-T Recommendation G.114," [Online]. Available:

https://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-G.114-200305-I!!PDF-E&type=items. [Accessed 12 12 2018].

[119] Rahman, I, "Nonbinary error detection codes for data retransmission and bit error rate monitoring," in *IEEE Transactions on Communications*, 1992.

[120] Yamaguchi, M.; Ito, K.; Takasaki, Y., "acket loss detection scheme for retransmission-based real-time data transfer," in *Seventh International Conference on Parallel and Distributed Systems: Workshops*, 2000.

[121] Manfredi, S.; Garofalo, F.; Di Bernardo, M., "Analysis and effects of retransmission mechanisms on data network performance," in *Proceedings of the 2004 International Symposium on Circuits and Systems*, 2004.

[122] Mengshiang Lin; Chen Kun Chuang; Ching Yao Huang, "Performance analysis of MAC retransmission in high-speed data transmission," in *Wireless Networks, Communications and Mobile Computing*, 2005.

[123] Stevens, R, "TCP Timeout and Retransmission," 2004. [Online]. Available: http://www.pcvr.nl/tcpip/tcp_time.htm. [Accessed 12 12 2018].

[124] Timm C, and Perez R., Phishing Attacks. In Seven Deadliest Social Network Attacks, Boston: Syngress, 2010, p. 43–61.

[125] Lu K, Wu D, Fan J, Todorovic S, Nucci, A., "Robust and efficient detection of DDoS attacks for large-scale internet," *Computer Networks,* 2007.

[126] "How Traceroute Works," [Online]. Available: http://www.inetdaemon.com/tutorials/troubleshooting/tools/traceroute/definition.shtml. [Accessed 12 12 2018].

[127] "CERT Advisory CA-98.01: Smurf IP Denial-of-Service Attack via Pings," Computer Emergency Response Team, [Online]. Available: http:Hwww.cert.org/advisories/CA-98.01.smurf.html. [Accessed 12 12 2018].

[128] Rajitha Tennekoon, Janaka Wijekoon, Erwin Harahap, Hiroaki Nishi, "Per-hop Data Encryption Protocol for Transmitting Data Securely Over Public Networks," in *Procedia Computer Science*, 2014.

[129] Housley, R., "Using advanced encryption standard (AES) counter mode with IPsec encapsulating security payload (ESP)," IETF RFC 3686, 2004. [Online]. Available: http://www.ipa.go.jp/security/rfc/RFC3686EN.html. [Accessed 12 12 2018].

[130] "Aho-Corasick parallel string search algorithm - GitHub Aho-Corasick Implementation," [Online]. Available: https://github.com/mischasan/aho-corasick. [Accessed 12 12 2018].

[131] "Delivering 160Gbps DPI Performance on the Intel® Xeon® Processor E5-2600 Series using HyperScan," 2013. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/160gbps-dpi-performance-using-intel-architecture-paper.pdf. [Accessed 12 12 2018].

[132] "University of Oregon Route Views Project - Home page," 2015. [Online]. Available: http://www.routeviews.org/. [Accessed 12 12 2018].

[133] "About Interop Tokyo - Interop home page," 2017. [Online]. Available: https://www.interop.jp/2017/. [Accessed 12 12 2018].

[134] "INTEL® QUICKASSIST TECHNOLOGY," Intel open source.org, 2018. [Online]. Available: https://01.org/intel-quickassist-technology. [Accessed 12 12 2018].

[135] "Intel® QuickAssist Technology - API Programmer's Guide," 12 2018. [Online]. Available:

https://01.org/sites/default/files/downloads//330684qatapiprogrammersguiderev005us.pdf. [Accessed 12 12 2018].

[136] Fredkin, E., "Trie memory," in *Commun. ACM*, 1960.

[137] Comer, D., Computer networks and internets, New York, New York: Springer, 1987, p. 368.CAM T.

[138] W. Doeringer, G. Karjoth and M. Nassehi, "Routing on longest-matching prefixes," in *IEEE/ACM Transactions on Networking*, 1996.

[139] Fuller, V., "Classless Inter-domain Routing (CIDR)," IETF RFC 4632, 8 2006. [Online]. Available: https://tools.ietf.org/html/rfc4632.

[140] Bahadur N., Uber, S., Kini, S., and Medved, J., "RIB Information Model," IETF RFC 8430, 9 2018. [Online]. Available: https://tools.ietf.org/html/rfc8430.

[141] Stallings, W., Data and computer communications, NewJursey: Pearson Prentice Hall, 2007.

[142] Ager, B., Schneider, F., Kim, J., and Feldmann, A., "Revisiting cacheability in times of user generated content," in *IEEE Global Internet Symposium*, 2010.