NETWORK MONITORING IN DATA LINK

Umika Saroha, Anchal Kumari,
Department of Computer Science and Engineering
National Institute of Technology Karnataka
Surathkal, Mangalore, India
7678416475,6299349700
umikasaroha.211cs161@nitk.edu.in,anchalkumari211cs105@nitk.edu.in

July 18, 2023

Abstract

A. Background of the problem statement

Network monitoring in the data link layer involves analyzing and capturing network traffic at the link layer of the OSI model. The data link layer is responsible for framing packets, providing error detection, and ensuring reliable transmission of data over the physical network.

There are several tools and techniques used for network monitoring at the data link layer, including: Packet sniffers Network probes Protocol analyzer Port mirroring [1, 2]

B. Challenges and issues of the problem statement

there are several problems and issues regarding the context for example, Encryption: In many modern communication protocols, such as HTTPS and SSH, we use encryption to protect data in transit. but sometimes the encryption can make it difficult or impossible to monitor network traffic at the data link layer, as the encrypted data is unreadable. The volume of data: Networks can generate a large volume of data, which makes it difficult to capture and analyze all traffic. Performance impact: This impact can result in increased latency, dropped packets, or other issues that can affect the network's performance. Cost: This cost can be a significant barrier for smaller organizations or those with limited budgets. Complexity: Some monitoring tools, such as flow-based monitoring, can require complex configurations and may require specialized knowledge or expertise to implement and maintain. Security and privacy concerns: Monitoring network traffic can potentially expose sensitive data or information, which can be a concern for organizations that handle confidential or proprietary data. Additionally, some monitoring tools can be vulnerable to security threats or attacks, which can compromise the integrity of the network or the data being monitored

C. Existing approaches or methods and their issues

Port Mirroring: widely available in switches TAP: reliable at high speeds Packet Capture: unfettered access to full network data during analysis Pattern Matching DPI: easy detection rule development Event-Based DPI: requires more complicated implementation than pattern-matching DPI Flow Observation: privacy packet payload data not used Quality of Service (QoS) is a technique used to prioritize network traffic and allocate network resources to ensure that critical network applications have sufficient bandwidth and low latency. QoS can be implemented at various layers of the network stack, including the data link layer. [1, 3–5]

D. Your problem statement

Several authors have proposed solutions to mitigate data link layer attacks. Each mitigation technique was entirely different from the others and has its own disadvantages. [2]

E. Objectives of the proposed work

Prime focus of the project is to identify the attack at the data link layer and check its possibility to merge together all attacks in one place using QOS. Not all the data link layer attacks can be identified by IDS For the current work various solutions already researched are put together to analyze attacks at the data link layer. Further customized code can be written to detect data link layer attacks is possible. [6–9]

Literature Review

A. Existing Approaches To Solve The Above Problem

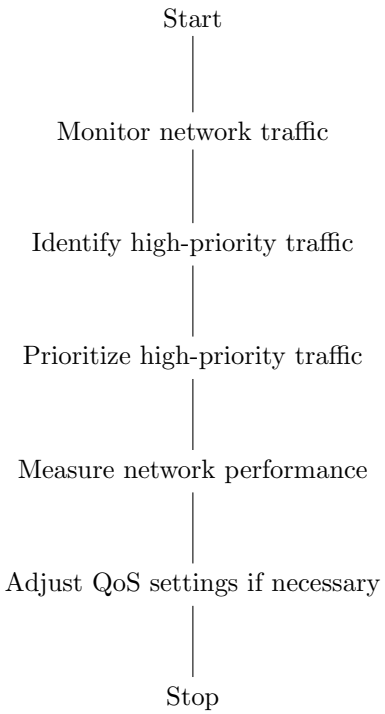
Quality of Service (QoS) is a technique used to prioritize network traffic and allocate network resources to ensure that critical network applications have sufficient bandwidth and low latency. QoS can be implemented at various layers of the network stack, including the data link layer. To implement QoS for improving network monitoring at the data link layer, the following steps can be taken: Identify critical applications: Identify the applications that require high priority and low latency. For example, video streaming applications, VoIP, or real-time gaming applications require low latency and high bandwidth. Define traffic classes: Define traffic classes based on the priority of the applications. For example, define a high-priority traffic class for real-time applications and a low-priority traffic class for non-real-time applications. Assign priorities: Assign priorities to each traffic class. For example, assign a higher priority to the high-priority traffic class and a lower priority to the low-priority traffic class. Define bandwidth allocation: Allocate bandwidth to each traffic class based on their priority. For example, allocate more bandwidth to the high-priority traffic class than the low-priority traffic class. Implement queuing: Implement a queuing mechanism to handle traffic congestion. Use a priority queuing mechanism where packets from high-priority traffic classes are dequeued before packets from low-priority traffic classes.

Monitor network performance: Monitor network performance to ensure that the QoS policies are being followed and the critical applications are receiving sufficient bandwidth and low latency.

By implementing QoS, network monitoring at the data link layer can be improved by ensuring that critical applications have sufficient bandwidth and low latency, leading to better network performance and user experience.

Algorithm 1 Algorithm to implement QoS for data link layer network monitoring

Data: Critical applications, traffic classes, priorities, bandwidth allocation
Result: Improved network monitoring at the data link layer
Identify critical applications: Identify the applications that require high priority and low latency;
Define traffic classes: Define traffic classes based on the priority of the applications;
Assign priorities: Assign priorities to each traffic class;
Define bandwidth allocation: Allocate bandwidth to each traffic class based on their priority;
Implement queuing: Implement a queuing mechanism to handle traffic congestion;
Monitor network performance: Monitor network performance to ensure that the QoS policies are being followed and the critical applications are receiving sufficient bandwidth and low latency;



Pros:

Efficient use of network resources: QoS at the data link layer allows network administrators to prioritize traffic and allocate resources based on the needs of specific applications. This can result in more efficient use of

network bandwidth and less congestion.

Improved application performance: By giving priority to critical applications, QoS can improve the performance of these applications and reduce latency. This can be particularly important for real-time applications like video conferencing and VoIP.

Better network management: QoS at the data link layer can provide network administrators with more control over the network, making it easier to manage and troubleshoot issues.

Cons:

Increased complexity: Implementing QoS at the data link layer can be complex and require specialized knowledge. It may also require additional hardware and software.

Limited scope: QoS at the data link layer is limited to the local network, and cannot be used to manage traffic on the wider internet.

Increased overhead: QoS at the data link layer can add overhead to the network, which can reduce overall network performance.

Interoperability issues: QoS implementations can vary between different vendors and devices, which can cause interoperability issues and make it difficult to manage a heterogeneous network.

B. comparative study

Here's an improved algorithm for implementing QoS at the data link layer: Identify critical applications: Identify the applications that require high priority and low latency. For example, video conferencing applications, VoIP, or real-time gaming applications require low latency and high bandwidth. Define traffic classes: Define traffic classes based on the priority of the applications. For example, define a high-priority traffic class for real-time applications and a low-priority traffic class for non-real-time applications. Consider defining additional traffic classes to provide finer control over bandwidth allocation. Assign priorities: Assign priorities to each traffic class. For example, assign a higher priority to the high-priority traffic class and a lower priority to the low-priority traffic class. Consider assigning different priorities to different traffic classes based on their requirements. Define bandwidth allocation: Allocate bandwidth to each traffic class based on their priority. For example, allocate more bandwidth to the high-priority traffic class than the low-priority traffic class. Consider using dynamic bandwidth allocation to adjust bandwidth allocation based on network conditions. Implement queuing: Implement a queuing mechanism to handle traffic congestion. Use a priority queuing mechanism where packets from high-priority traffic classes are dequeued before packets from low-priority traffic classes. Consider using weighted fair queuing or other queuing mechanisms to provide more control over bandwidth allocation. Monitor network performance: Monitor network performance to ensure that the QoS policies are being followed and the critical applications are receiving sufficient bandwidth and low latency. Consider using network monitoring tools to track bandwidth usage, latency, and packet loss. Adapt to changing network conditions: Adapt the QoS policies to changing network conditions. Consider using dynamic bandwidth allocation and queuing mechanisms to adjust QoS policies based on network conditions such as congestion or link failure.

C. Proposed Methodolgy

To implement Quality of Service (QoS) on the data link layer, the following steps can be taken: Identify the traffic types: Different types of traffic have different QoS requirements. Identify the types of traffic that will be transmitted over the data link layer, such as voice, video, data, etc. Define the QoS parameters: Determine the QoS parameters that are important for each traffic type. For example, for voice traffic, the important QoS parameters could be latency and jitter, while for video traffic, the important parameters could be throughput and packet loss. Map traffic to classes: Based on the QoS requirements of each traffic type, map them to different traffic classes. For example, voice traffic could be mapped to a high-priority class, while data traffic could be mapped to a low-priority class. Implement QoS mechanisms: Implement QoS mechanisms to ensure that the traffic in each class receives the desired QoS. Some mechanisms that can be used include traffic shaping, packet scheduling, and congestion control. Monitor and adjust: Monitor the network to ensure that the desired QoS is being met. If not, adjust the QoS parameters or the QoS mechanisms as necessary.

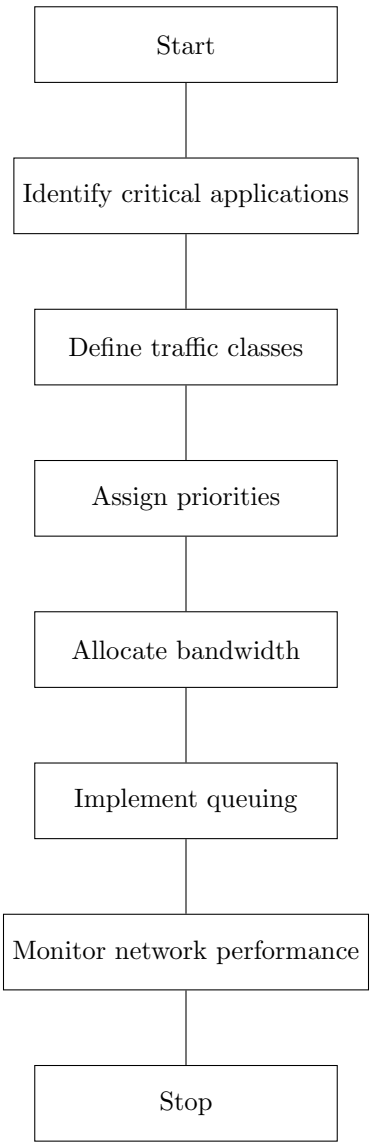


Figure 1: QoS Implementation at the Data Link Layer

1 Code Comparison

1.1 Code 1

% Code 1 output

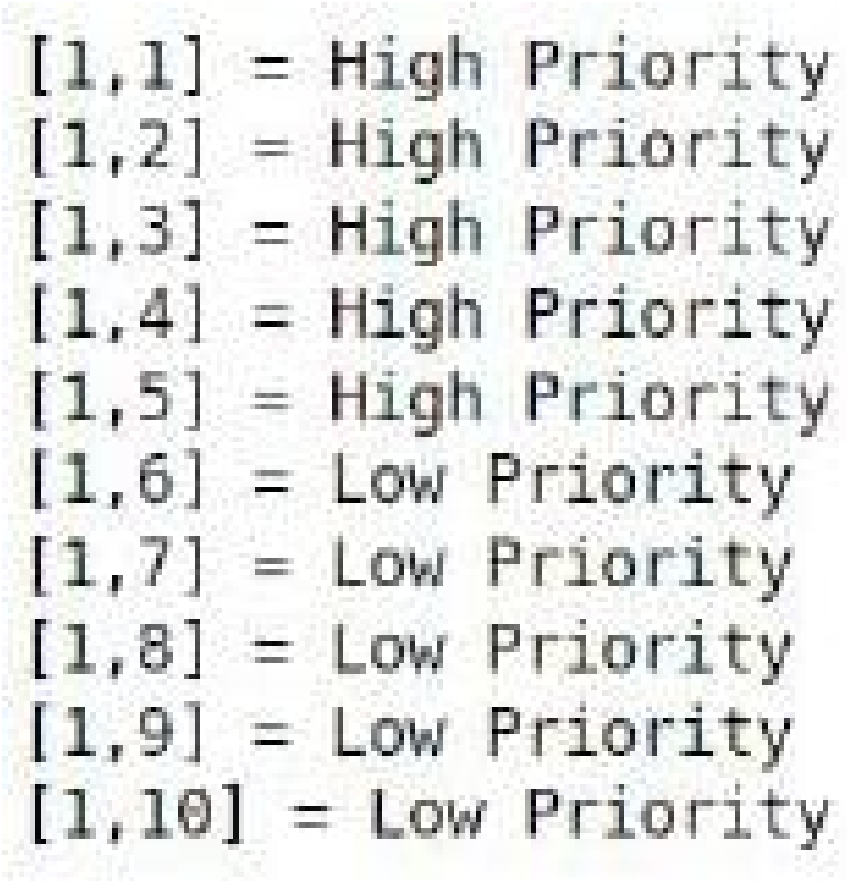


Figure 2: Output of Code 1

1.2 Code 2

% Code 2 output

```
Packet Enqueued:
High Priority
Packet Enqueued:
High Priority
Packet Enqueued:
Low Priority
Packet Enqueued:
Low Priority
Packet Enqueued:
Low Priority
Packet Enqueued:
Low Priority
Packet Enqueued:
Low Priority
Packet Enqueued:
Low Priority
Packet Enqueued:
Low Priority
Packet Enqueued:
Low Priority
Number of High Priority Packets in Queue:
1
Number of Low Priority Packets in Queue:
0
```

Figure 3: Output of Code 2

2 Explanation

Both code snippets are related to simulating and monitoring network traffic, but they have different focuses and objectives.

The first code describes a simulation of a network system with different traffic classes and implements a priority queuing mechanism to process packets based on their priority. It also allows for specific QoS policies to be applied to critical applications and provides a framework for monitoring network performance. However, the specific implementation details are not provided in the description.

The second code is a simulation of a network monitoring system that tracks the performance of various traffic classes and critical applications on the network. It updates network latency and bandwidth utilization metrics based on the processed packets. The code provides more specific details about the variables used for monitoring and the process of updating the metrics.

Based on the descriptions, the second code appears to be better in terms of providing clarity and specific information about the network monitoring process. It explicitly defines the monitoring variables and describes how they are updated during the simulation.

2.1 Code 1

The given code is a simulation that models a network system with multiple traffic classes and implements quality of service (QoS) policies based on the priorities of the applications.

Here's a step-by-step explanation of the code:

1. The criticalApplications variable is an array that specifies the critical applications. In the example, applications 1, 3, and 5 are considered critical.

2. The trafficClasses variable assigns a traffic class to each application based on its priority. In this example, traffic class 1 is for high-priority applications, traffic class 2 is for medium-priority applications, and traffic class

3 is for low-priority applications.

3. The priorities variable assigns a priority to each traffic class. In the example, priority 1 is assigned to traffic class 1, priority 2 to traffic class 2, and priority 3 to traffic class 3.

4. The bandwidthAllocation variable defines the allocation of bandwidth for each traffic class. In the example, 50 percent of the bandwidth is allocated to traffic class 1, 30 percent to traffic class 2, and 20 percent to traffic class 3.

5. The priorityQueue variable is a cell array that represents the queues for each traffic class. In this simulation, a priority queuing mechanism is used, where packets from high-priority traffic classes are dequeued before packets from low-priority traffic classes.

6. The queueLengths variable is an array that stores the current lengths of each queue.

7. The packetProcessingTime variable tracks the total processing time of packets in the simulation.

8. The simulation loop iterates over each traffic class and simulates packet arrival and processing.

9. For each iteration, the code increments the queue length for the corresponding traffic class to simulate packet arrival.

10. The packetProcessingTime is incremented to simulate packet processing.

11. The code checks, if the current application is critical by using the, is member function to check if the iteration number is present in the criticalApplications array. If it is, a message indicates that a packet from a critical application is being processed.

12. The code enters a while loop to process packets from higher-priority queues. It checks if the queue length of the current priority is greater than zero. If it is, a message is displayed indicating that a packet from the corresponding traffic class is being processed. The queue length is then decremented, simulating packet processing.

13. After processing all the higher-priority packets, the code moves to the next iteration of the simulation loop.

14. The network performance metrics can be updated within the simulation loop to monitor factors such as network latency, bandwidth utilization, and other relevant metrics.

Overall, the code models a network system with different traffic classes and implements a priority queuing mechanism to process packets based on their priority. It also allows for specific QoS policies to be applied to critical applications and provides a framework for monitoring network performance.

2.2 Code 2

This code is a simulation of a network monitoring system that monitors the performance of various traffic classes and critical applications on the network. Let's step through the code.

1. The code starts by defining some variables. - 'criticalApplications': list of application numbers that are considered critical. - 'trafficClasses': A list of traffic class numbers for each packet. - "Priorities": List of priorities assigned to each traffic class. - 'bandwidthAllocation': List of bandwidth allocated to each traffic class in percentage.

2. The two monitoring variables 'networkLatency' and 'bandwidthUtilization' are initialized with a null vector of size equal to the maximum traffic class number.

3. The simulation loop iterates through each traffic class and takes the next step. - Get the traffic class number, priority and bandwidth allocation for the current iteration. - If the current traffic class is in the 'criticalApplications' list, you will see a message saying that packets from critical applications are being processed. - Update monitoring variables whether application critical or not. - 'networkLatency': Increment the index value corresponding to the current traffic class number to indicate that the packet was processed for this class.

- ‘bandwidthUtilization‘: Measures the total bandwidth usage for that class by adding the bandwidth allocated to the current traffic class to the index value corresponding to the current traffic class number.

4. When the simulation cycle is complete, the code displays the network monitoring results. It loops through each traffic class again and prints the following information for each class: - Traffic class number. - Network Delay: The number of packets processed for this traffic class (for critical applications). - Bandwidth usage: Percentage of total bandwidth usage for this traffic class. Thus, the code simulates packet processing for different traffic classes and critical applications on the network. Track and monitor network latency and bandwidth usage for each traffic class to provide insight into network performance.

References

[1] R. Shanker, A. Singh, Analysis of network attacks at data link layer and its mitigation (2021). doi:10.1109/ICCS54944.2021.00061.

[2] J. Svoboda, I. Ghafir, V. Prenosil, Network monitoring approaches: An overview, International Journal of Advances in Computer Networks and Its Security– IJCNS 5 (2015) 88–93.

[3] Snort, https://en.wikipedia.org/wiki/Snort_(software).

[4] IDS, https://www.geeksforgeeks.org/intrusion-detection-system-ids/amp/.

[5] Error detection, https://www.geeksforgeeks.org/error-detection-in-computer-networks/amp/.

[6] Snort, https://www.geeksforgeeks.org/what-is-snort/.

[7] Snort, https://www.snort.org/.

[8] Packet Capture, https://www.varonis.com/blog/packet-capture.

[9] Pattern matching DPI, https://ieeexplore.ieee.org/document/4545432.

[10] Port Mirroring, https://www.techopedia.com/definition/16134/port-mirroring.

[11] Tap, https://insights.profitap.com/what-are-network-taps.

[12] Framing Packets, https://www.baeldung.com/cs/networking-packet-fragment-frame-datagram-segment#:~:text=While%20a%20packet%20is%20the,length%20and%20variable%2Dlength%20frames.

[13] Mitigation Technique, https://www.hypr.com/security-encyclopedia/mitigation#:~:text=Mitigation%2C%20or%20Attack%20Mitigation%2C%20is,activism%2C%20retribution%2C%20or%20mischief.

[14] Protocol Analyzer, https://www.geeksforgeeks.org/what-is-protocol-analyzer/.

[15] Packet Sniffers, https://www.kaspersky.com/resource-center/definitions/what-is-a-packet-sniffer.

[16] J. Conard, Services and protocols of the data link layer (1983). doi:10.1109/PROC.1983.12781.

****** END ******

Note:

- Include your reference details in the ref. bib file of the shared project
- References to be referred to within the content as [1–16]