DINT: A Dynamic Algorithm for In-band Network Telemetry

Henrique B. Brum[1], **Carlos R. P. dos Santos**[2], **Tiago C. Ferreto**[1]

[1]Faculty of Informatics, Pontifical Catholic University of Rio Grande do Sul (PUCRS)
Porto Alegre, Brazil

[2]Department of Applied Computing, Federal University of Santa Maria (UFSM)
Santa Maria, Brazil

henrique.brum@edu.pucrs.br, tiago.ferreto@pucrs.br, csantos@inf.ufsm.br

Abstract. *Network monitoring is fundamental for the correct and expected functioning of today's large computer networks. In-band Network Telemetry (INT) has become one of the main tools for collecting network information in recent years. By piggybacking information using business packets, INT can deliver real-time network statistics to monitoring engines. However, INT's fine granularity comes with a high network overhead cost. This paper focuses on balancing this trade-off between accurate monitoring and high telemetry overhead. To achieve it, we propose DINT, a Dynamic INT algorithm capable of adapting to different traffic patterns while keeping an accurate view of the network and reducing flooding it with redundant telemetry data. In our experiments, DINT presented higher adaptability compared to other techniques, providing a more accurate view of the network while requiring fewer telemetry data.*

1. Introduction

Computer networks have drastically evolved over the last years, primarily because of the global adoption of the Internet and the subsequent expansion of Cloud solutions. To properly maintain these new complex networks, it is fundamental that modern management and monitoring techniques be employed. The Software Defined Networking (SDN) paradigm has emerged as a promising solution to meet this end, as it can improve and facilitate the network operator's tasks. Following SDN's idea of separating the data plane from the control plane and the capability of programming network functions, a new branch of SDN called the Programmable Data Plane (PDP) was proposed as a way to provide even more control over the network's data plane capabilities. The Programming protocol-independent packet processors (P4) [Bosshart et al. 2014] is one of the leading technologies for controlling packet forwarding planes in networking devices.

As these new techniques started appearing, new monitoring solutions emerged in traditional SDN and PDP. Although SDN has improved the flexibility of the network measurement architecture, the flow collection and sampling methods used in conventional network measurement techniques were preserved [Tan et al. 2021]. Because of that, proposed traditional SDN monitoring frameworks continued to collect network information via pull commands (e.g., [Chowdhury et al. 2014] and [Tangari et al. 2018]). In pull-based monitoring solutions, a centralized control plane or network management system periodically queries network switches for updated statistics. With the emergence of PDP, network telemetry [Yu 2019] became a mainstream research topic encompassing different telemetry techniques. Network telemetry is the automated process for remotely collecting

and processing network information [Tan et al. 2021]. In-band network telemetry is one of the most promising variations of network telemetry. With in-band network telemetry, network information is collected by inserting network metadata in packets by switching nodes, meaning that telemetry data and user data share the same link or even the same packet.

Following the trends in network monitoring, the P4 Language Consortium (P4.org) proposed its implementation of in-band network telemetry, the In-band Network Telemetry (INT) [Kim et al. 2015]. INT is a framework that allows data packets to query switch-internal states such as queue size, link utilization, and queuing latency without the intervention of a centralized control plane. By collecting information directly from the switches, INT can achieve fine-grained real-time data plane information. Nonetheless, the minute view of the network provided by INT comes at the cost of high telemetry overhead. This high overhead exists because every switching device receiving a packet with an INT header inserts network information in the packet. High telemetry overhead can lead to poor throughput and overload the monitoring application with too much information. To tackle this issue, multiple solutions have been proposed. Still, most of them delegate too much of their logic to the control plane, which can incur latency and affect the application's performance [Kfoury et al. 2021]. Regarding the few works that employ most of the logic on the data plane, some limitations include the absence of dynamic thresholds to determine when to collect statistics (e.g., [Sheng et al. 2021]) and the lack of periodic telemetry reports (e.g., [Chowdhury et al. 2021]).

Knowing the constraints of using INT and the shortcomings of the state-of-the-art solutions, we propose an algorithm capable of accurately presenting the network state while reducing the telemetry overhead. In this work, we introduce DINT, a dynamic In-band Network Telemetry algorithm that runs directly in data plane devices. Specifically, each switch running DINT keeps track of the network's traffic, and when it detects an abrupt traffic rate change according to dynamic thresholds, the switching device updates its telemetry insertion time. To alleviate the job of the network operator of finding suitable thresholds for a vast range of possible network traffics, DINT regularly updates its thresholds to consider the latest traffic patterns.

Our major contributions are summarized as follows:

- A Dynamic In-band Network Telemetry algorithm for programmable data plane devices. DINT automatically adjusts its telemetry insertion rate according to the network traffic;
- An extensive evaluation of DINT against a naive approach, a classical monitoring solution, and a state-of-the-art mechanism;
- Source code, testing scripts, and results are available in our GitHub repository [1] to give further insights into our solution and evaluation process while also enabling reproducibility.

The rest of the paper is organized as follows. In Section 2, we present the main concepts and technologies used in our work. In Section 3, we present our algorithm and some implementation considerations. Then, we evaluate DINT and compare it with three other solutions in Section 4. We discuss the related work and compare their differences with our proposal in Section 5. Finally, we present our final remarks in Section 6.

[1] DINT repository - https://github.com/HenriqueBBrum/DINT/tree/main

2. Background

The P4 language [Bosshart et al. 2014] is an open-source, domain-specific programming language for network devices, specifying how data plane devices process packets. Despite its initial focus on programming hardware or software switches, P4 has expanded and now covers a wide variety of devices such as network interface cards, network appliances, ASICs, NPUs, and FPGAs [Kaur et al. 2021]. P4 was designed with three main objectives: reconfigurability, protocol independence, and target independence. Reconfigurability is the capacity to easily change a P4 program to process packets as desired. Protocol independence means that P4 supports any protocol. Target independence refers to the fact that P4 programs are free from specific hardware features; they only need to conform to the vendor's device architecture.

INT is a monitoring framework developed by the P4 Language Consortium (P4.org) that enables transit nodes, such as switches, to embed network information in any passing packet. Using INT, a monitoring application can achieve fine-grained information about the network status without the intervention of the control plane. According to the INT specification [Group 2020], there are three agents: INT Source Node, INT Sink Node, and INT Transit Hop. INT Source Node is responsible for embedding telemetry instruction into packets. INT Transit Hop only needs to fill in telemetry metadata according to the instruction of the INT packet. INT Sink Node extracts and reports the telemetry results to a monitoring engine.

One limitation of INT is the excessive overhead it causes. Excessive overhead can be detrimental to the network's goodput. In [Chowdhury et al. 2021], the authors conducted an analytical study to measure the impact of INT data plane overhead on network goodput. Their experiments showed that even for just one hop, network goodput can be reduced by around 20%. These results show the need for techniques to accurately present the network status while keeping the network's overhead to a minimum.

3. Dynamic INT Algorithm

This section presents DINT, a dynamic monitoring algorithm based on INT framework to monitor programmable network devices. DINT aims to achieve an accurate network view while incurring low network overhead.

3.1. DINT: A Dynamic INT algorithm

DINT was designed with two goals: 1) To reduce network overhead without losing monitoring quality; 2) To keep the algorithm light enough to run it in commodity PISA devices, where the programming capabilities are limited.

DINT inserts network information in regular packets between a minimum and a maximum period, and the occurrence of changes in traffic volume updates this interval. Therefore, DINT continuously collects network information at a frequency that depends on the network's behavior. For instance, a higher frequency is used when traffic is more erratic, while a lower frequency is used when it is more steady. The algorithm presented in Algorithm 1 is responsible for updating the period that network information is collected in each programmable device and the thresholds used to check if a significant change has happened.

Algorithm 1 DINT Algorithm

Input: T_{min} and T_{max} as minimum and maximum telemetry insertion interval; α as telemetry insertion interval growth rate, threshold value Δ , positive integer *k*, SMA divisor β.

1: **if** $observation\_window\_interval >= T_{min}$ **then**
2: **if** $diff\_byte\_count > |\Delta|$ **then**
3: $\lambda = T_{min}$
4: **else**
5: $\lambda = min(\lambda \cdot \alpha,\ T_{max})$
6: **end if**
7: Δ = UpdateThreshold$(k, \beta, current_obs_byte_count)$
8: **end if**
9:
10: **if** $telemetry\_insertion\_interval >= \lambda$ **then**
11: AddTelemetryToPacket$(telemetry_data)$
12: **end if**

Algorithm 1 works as follows. At every T_{min} milliseconds (*observation_window_interval*), it calculates the amount of traffic that has passed since its previous *observation window* (*diff_byte_count*). When the difference in traffic is greater than a threshold value Δ, i.e., a significant change in the network traffic has occurred, the algorithm sets the telemetry insertion period λ to a minimum period value of T_{min}. A change in the network traffic can mean either an increase or decrease in throughput. By contrast, if the difference is smaller than Δ, i.e., no significant changes in the throughput have happened, it increases λ by α times until it reaches a maximum period value of T_{max}. After updating the telemetry insertion period, it also updates Δ to keep up with the most recent network traffic measurements by calling the *UpdateThreshold*$(...)$ function. Finally, it checks if the *telemetry window* interval has elapsed (i.e., $telemetry\_insertion\_interval >= \lambda$). In such a case, it adds telemetry information to the current packet. This algorithm executes every time a packet enters the programmable device.

3.2. Update thresholds using Moving Average

Even though fixed thresholds might work for contained and predictable traffic, their scope is limited to the presumed network behavior. Any fluctuations in the expected traffic pattern will result in either high telemetry overhead or an inaccurate view of the network. An algorithm that periodically adapts itself to the latest network information is essential to solve these issues.

To tackle this problem while considering that PISA devices have many programming limitations [Chowdhury et al. 2021] (e.g., no floating point numbers, no division, and no loops), DINT employs moving average functions to compute the dynamic thresholds. Moving average is an adequate solution to this problem since it can be programmed into a PISA device with minimum modifications and provide a clear picture of the current network's behavior. More specifically, DINT uses the Simple Moving Average (SMA) function, which is the unweighted mean of the previous *k* data entries. SMA is a method that needs minimal parameter tuning to find a satisfactory configuration.

Since the division of run-time values is impossible in the P4 language, the right shift operation is used to replace it. Although it is feasible to divide compile-time values, the average for the SMA calculation is restricted to comprising only the last 2^n data measurements, where *n* is used in the right shift operation. The *UpdateThreshold*$(...)$ function of Algorithm 1 uses SMA and is described in Algorithm 2.

Algorithm 2 Update Threshold

Input: Positive integer *k*, SMA divisor β, Byte count of the current observation window $current\_obs\_byte\_count$.

Output: SMA_k/β

1: $sum = sum + current\_obs\_byte\_count$
2: $obs\_window\_count = obs\_window\_count + 1$
3: **if** $obs\_window\_count == k$ **then**
4: $mean = sum >> n$ {n is $2^n = k$}
5: $\Delta = mean/\beta$
6: $sum = 0$
7: $obs\_window\_count = 0$
8: **end if**
9: **return** Δ

Algorithm 2 is called every time a packet enters the *observation window* in Algorithm 1. This means that at every *k-th observation window*, the Δ value is updated to reflect the average of the *k* previous *observation windows*. The value returned by SMA is an estimated guess on how the network traffic is behaving and how it might behave. Knowing this, the new Δ is going to be the *I-th* part of the SMA, meaning, for example, that if the SMA of the last *k observation windows* reported a network throughput of 100MB/s, and β is 10, the new Δ is going to be 10MB/s. Otherwise, the new Δ would be too big for the current traffic trend. Table 1 presents a description of DINT's parameters.

Table 1. DINT parameters

Parameter	Description
T_{min}	Minimum telemetry insertion period
T_{max}	Maximum telemetry insertion period
α	Period multiplier
k	The number of observation windows for the SMA
β	Fraction to consider as the new Δ from the SMA

3.3. Implementation Considerations

While INT [Kim et al. 2015] is a robust framework, we developed a simpler variation of INT-MD (INT eMbedded Data) that only monitors network throughput. For more complex topologies and requirements, INT is preferred, but our implementation was adequate for the goals of this paper. In our adaptation depicted in Figure 1, all programmable devices are an INT source, INT transit, and INT sink simultaneously. They can insert telemetry headers, add network information, and extract telemetry headers for monitoring purposes. Like INT, we use a similar header that carries meta-information about the

telemetry headers (INT header) and a stack of headers carrying network metadata (INT metadata header). When a packet with an INT header arrives at the last switch before the final host, the original packet is cloned. The monitoring information is removed from the original packet before forwarding it to the final host. The cloned packet has its payload removed, and only the monitoring information is forwarded to the monitor. Table 2 presents the description and the number of bits used by the DINT metadata header fields.

Table 2. DINT metadata header

Field	Bits used	Description
Bottom of Stack	1	Identifier of the last metadata header
Switch ID	7	Identifier of a device
Egress Port ID	32	Identifier of a packet's egress port
Amount bytes	32	Amount of bytes counted since last report
Previous telemetry time	48	Previous telemetry collection timestamp
Current telemetry time	48	Current telemetry collection timestamp

A final consideration is that in our variation, data plane packets do not have INT headers by default, only when there is an INT metadata header. This is mentioned because the state-of-the-art technique used for comparison in the evaluation section allows INT headers without any metadata header. Figure 1 presents an example of DINT's workflow. The INT header (black rectangle) is inserted by switch SW1, as it is the first to insert network telemetry into the packet. Since each switch decides when to send network statistics, in this example, only switches SW1 and SW3 insert a telemetry metadata header (purple rectangle for SW1 and orange rectangle for SW3) in the incoming packet. This is an improvement upon previous solutions that always added network information when a packet had an INT metadata header since it reduces the overall telemetry overhead while maintaining an accurate network view.

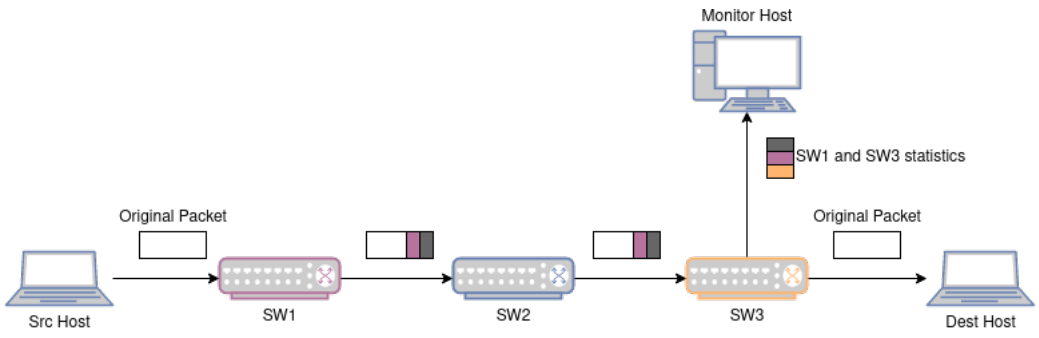

Figure 1. DINT's workflow example

4. Evaluation

In this section, we first describe our experimental setup and the evaluation methodology. Next, we define our evaluation metrics, followed by a demonstration that most parameters of DINT are not traffic-dependent but are general configurations of how accurate the monitoring needs to be at the cost of increased telemetry overhead. Finally, we compare DINT against a naive monitoring approach, a classical monitoring technique [Kim et al. 2018], and a state-of-the-art mechanism [Chowdhury et al. 2021].

4.1. Experiment Setup

Our simulation environment is an Ubuntu 20.04 virtual machine provided by the P4 language GitHub repository. We use the default tools installed in the Ubuntu VM to build our network prototype. Mininet emulates the specified network topology, and the reference P4 software switch, BMv2, is used to enable data-plane programmability within the Mininet switches. Our network prototype is a full mesh topology, as depicted in Fig. 2. It consists of one monitoring host (red hexagon), four switches (blue rectangles), and five end-user hosts (green circles). The monitoring host is connected to all switches, the switches are connected to all other switches, and each user host is only connected to one switch.

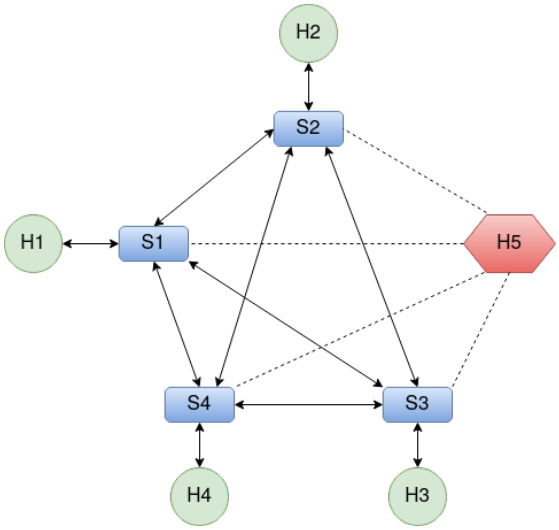

Figure 2. Experiment topology.

For the workload used in the experiments, we adapted the traffic pattern described in Payless [Chowdhury et al. 2014]. Fig. 3 illustrates the duration, the communicating pair, and the throughput for each flow of the evaluation traffic. After defining the desired traffic pattern, we captured the real traffic of one simulation with *tshark* and reused it for all subsequent tests with *tcpreplay*. Reusing the same traffic guarantees that every test has the same input, and if any substantial divergence happens, it is due to the algorithm being tested or the network emulator.

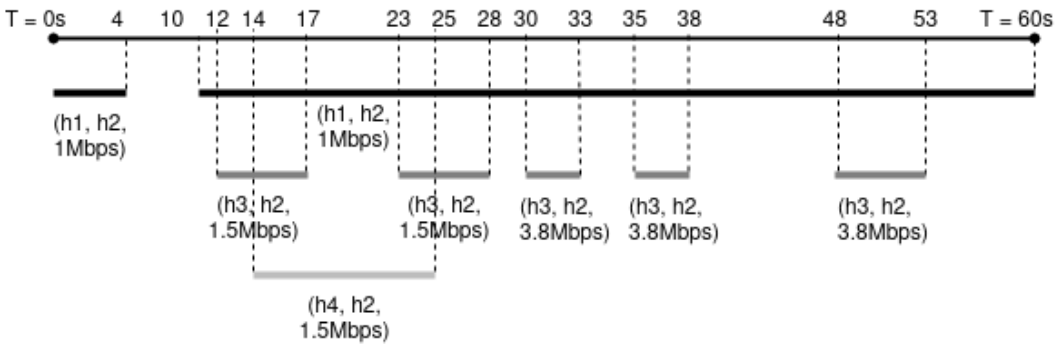

Figure 3. Timing diagram of the evaluation traffic

4.2. Evaluation Metrics

We used the following metrics in our experiments:

Network throughput Network throughput is measured as the amount of traffic flowing from a specific source to a specific destination at a certain point in time and measured in *Mbps* units. For this evaluation, we report the throughput of the switch *S2* (Fig 2).

Normalized Root Mean Squared Error (NRMSE) We evaluate a monitoring technique's accuracy by measuring the telemetry information's deviation from the ground truth using NRMSE. To compute the NMRSE, we first measure the real and the telemetry reported throughput and use them to calculate the Root Mean Squared Error (RMSE). Then, we normalize the RMSE value by dividing it by the difference between the maximum and minimum values of the real traffic data and obtain the NRMSE.

Telemetry Overhead We define telemetry overhead as the total amount in bytes of all telemetry headers used to carry out network information. INT headers with no INT metadata headers are discarded.

4.3. Evaluation of DINT Parameters

Our first set of results aims to demonstrate that most parameters of DINT are not traffic-dependent but are simply a trade-off on how accurate the monitoring is required to be at the cost of increased telemetry overhead. Moreover, the results obtained in this evaluation are used in Section 4.4 when comparing DINT with other monitoring methods. DINT has five possible parameters: the minimum telemetry insertion interval T_{min}; the maximum telemetry insertion interval T_{max}; the rate at which our telemetry insertion interval grows α; the number of previous data measurements used in the SMA function *k*; and, β as the relation between the SMA result and the new Δ. Despite the need to set these parameters, just *k* has a relatively more complicated tuning process.

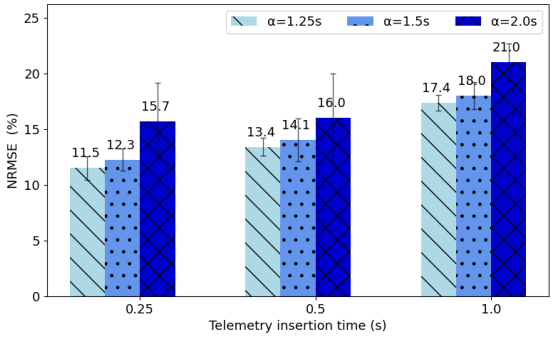

Figure 4. NMRSE of DINT varying α with different T_{min} values.

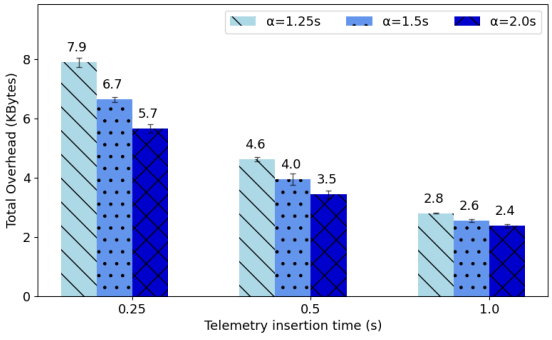

Figure 5. Telemetry Overhead varying α with different T_{min} values.

Choosing the value for these four parameters is just a question of how accurate the monitoring needs to be at the cost of increasing telemetry overhead. For T_{min}, T_{max}, and α values, smaller values incur higher accuracy, while greater values must be used if network limitations and high telemetry overhead are a problem. To give a perspective, 0.1s is a small value for T_{min} while 10s is big. For setting β, it is the opposite relation; greater values result in more accurate reports.

In Figures 4 and 5, we demonstrate this trade-off for the parameter α. As we can see in Figure 4, in all telemetry insertion time values (T_{min}), the lower the α is, the smaller the NMRSE and, correspondingly, the higher its accuracy. In Figure 5, we see the trade-off between accuracy and telemetry overhead, as lower values of α and T_{min} have a higher telemetry footprint. In both figures, for every pair, α and T_{min}, five tests were executed, and the error bar is the standard deviation of those tests. This process is repeated for all subsequent NMRSE and Telemetry Overhead graphs.

The reason that each of these four parameters presents this behavior is explained next. Since T_{min} indicates the shortest telemetry collection interval, we have finer granularity by using smaller values. The parameter T_{max} defines the maximum time the monitoring engine is allowed to be unaware of the network situation; with smaller values, the monitor receives more telemetry reports and is more accurate. The α parameter specifies how quickly should the monitoring interval go from the most accurate value to the least one; smaller α values take more time to perform this task meaning more telemetry data is collected. The β parameter adjusts what is considered a significant change in the network traffic; greater β values result in a smaller Δ, which means that the algorithm is more susceptible to smaller traffic variations, resulting in more telemetry reports and higher accuracy.

4.3.1. Choosing *k*

While tuning T_{min}, T_{max}, α and β is quite straightforward as presented before, setting parameter *k*, which is used in Algorithm 2 to define how often the Δ value is updated, is a little more complicate. We show this by executing experiments with three different *k* values.

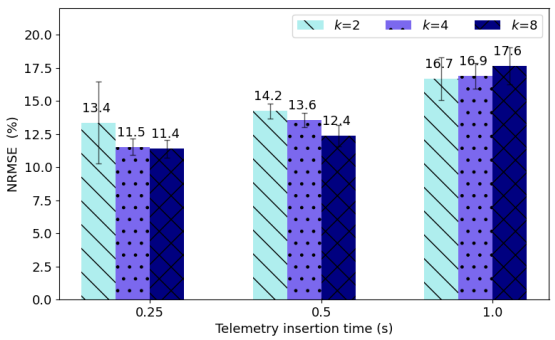

Figure 6. NMRSE of DINT varying *k* with different T_{min} values.

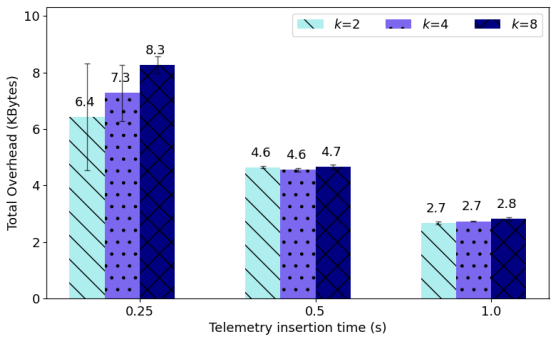

Figure 7. Telemetry overhead varying *k* with different T_{min} values.

Like the α evaluation, we have an NMRSE graph, Figure 6, and a Telemetry Overhead graph, Figure 7. In contrast to the findings of the evaluation of the previous parameters, changing *k* does not produce a clear trend. Although reducing T_{min} increases the performance in general, there is not a clear winner for *k* values. For example, *k*=8 provides the best performance with T_{min} equal to 0.25s and 0.5s but has the worst performance when T_{min} is 1.0s. A probable explanation for these results is that large *k* values with higher T_{min} take too much time to update the Δ whereas small *k* values with low

T_{min} update Δ too frequently.

4.4. DINT Comparison with other Monitoring Methods

After evaluating DINT's parameters and showing that tuning is required to achieve an optimal solution, we compare it with three other monitoring techniques: a naive method, a classical monitoring algorithm, and a state-of-the-art technique. The naive method inserts telemetry data into a packet based on a static interval. The classical monitoring algorithm (sINT) [Kim et al. 2018] uses the control plane to increase the telemetry insertion frequency in reaction to significant changes in consecutive INT metadata or reduce it if no significant changes have happened after a pre-defined stabilization time. DINT's main distinctions from sINT are that it dynamically updates what is considered a significant change to reflect the latest network traffic, requires no control plane intervention, and each switch decides when to insert network information into the packet.

The state-of-the-art technique (LINT) [Chowdhury et al. 2021] proposes an accuracy-adaptive and lightweight INT capable of running on programmable devices that use the same idea of significant changes and dynamic thresholds as our work. More specifically, a device running LINT tries to estimate with each incoming packet the amount of error that can be introduced at the collector if the requested telemetry data items are not piggybacked on the current packet. When the device estimates that the prediction error at the collector can go above an acceptable threshold, it inserts the current observation into the packet. Despite the similarities, DINT differs from LINT by working with time intervals, meaning it is harder to saturate the monitoring engine with excessive packet-by-packet telemetry resulting from unfit configuration. Additionally, the monitoring engine can always expect to receive the network status according to the time intervals defined, something LINT does not guarantee as it only adds telemetry information when a significant change occurs. Finally, DINT only inserts an INT metadata header when adding network information and not in every packet.

4.4.1. Implementation Details

Two considerations need to be taken before starting our evaluation. First, since per packet telemetry of network throughput is not optimal, we adapted LINT to run its algorithm at every T_{min} seconds interval, just like DINT and the static approach. This cannot be applied to sINT as its logic is done at a monitoring host, not the switches. Second, in LINT, it is unclear whether every packet has an INT header, even if the packet has no INT Metadata Header. We assume LINT allows empty INT headers, but since the telemetry overhead metric only considers telemetry reports with network information, the empty INT headers are not considered.

Finally, for the parameters of each algorithm, we used the values described next. sINTs fixed threshold makes it hard to find a good configuration. Still, we have found that if the difference between two consecutive throughput reports is greater than 0.25 Mbps and the stabilization time is 0.01 seconds, we can achieve acceptable results. For LINT, we chose the parameters with the highest accuracy reported in its paper. Our solution considers the following parameters: T_{max}=10s, α=1.25s, k=8, β=10. α=1.25s and k were chosen based on Section 4.3. We test all algorithms with three different T_{min} values 0.25s,

0.5s, and 1s and each configuration (e.g., {DINT, 0.25s}, {DINT 1.0s}, etc) is tested five times. The standard deviation of the five tests is represented by error bars in the NMRSE and Telemetry Overhead graphs.

4.4.2. Analysis of Experiment Results

Figures 8 and 9 display the contrast between the actual network traffic (blue line) and the one reported by the telemetry algorithm (red line). The gray area is the difference between the two lines (i.e., the time intervals when the monitoring engine had an incorrect network view). We present this comparison with the results obtained by DINT, in Figure 8, and the ones by LINT, in Figure 9, when T_{min} is equal to 1s.

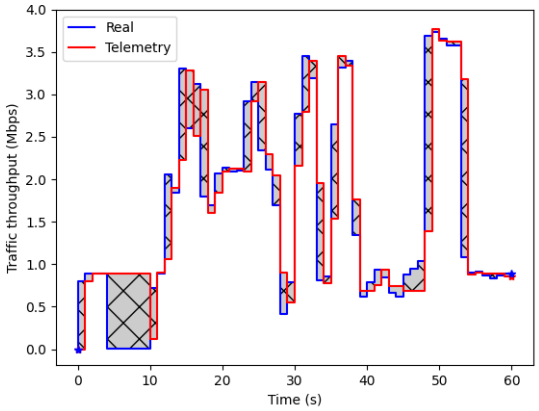

Figure 8. Network traffic using DINT with T_{min}=1s.

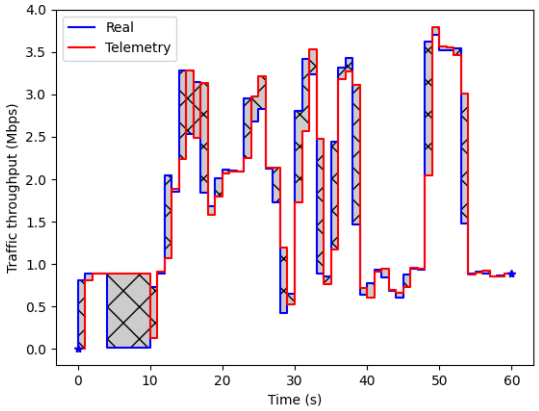

Figure 9. Network traffic using LINT with T_{min}=1s.

To properly quantify the performance of each algorithm, we use NMRSE in Figure 10 and the total telemetry overhead in Figure 11. We also evaluate how different telemetry insertion intervals (T_{min}) perform for each method. Regarding the accuracy of each algorithm, we can see in Figure 10 that LINT is the best at T_{min} 0.25s and 0.5s while DINT has the advantage at 1.0s.

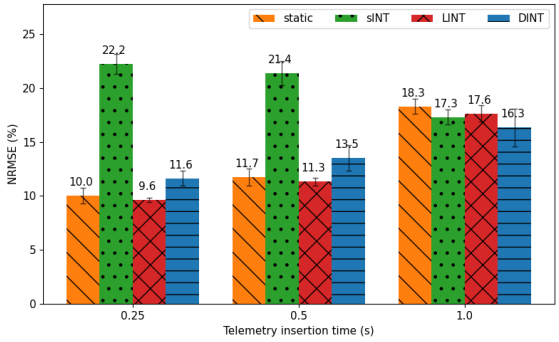

Figure 10. NMRSE of the evaluated techniques with different T_{min} values.

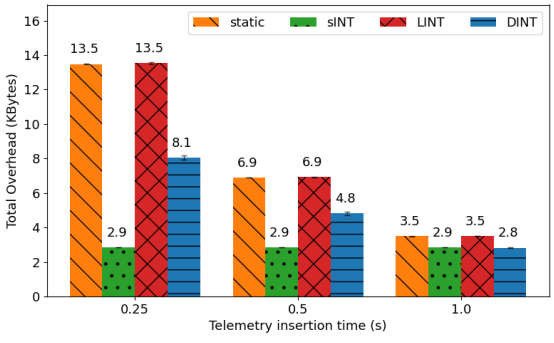

Figure 11. Telemetry overhead of the evaluated techniques with different T_{min} values.

There are some important observations to take from Figures 10 and 11. First, methods that collect network data with intervals (static, LINT, and DINT) obviously achieve better accuracy with smaller T_{min} since they collect more information. For sINT, the accuracy decreases even with the same telemetry overhead because the real data becomes more scarce. Second, the static method is outperformed by others solutions when there is a slight difference between their telemetry overheads. This happens because the static method simply collects data, while the other methods do it more intelligently by collecting when a significant event happens. When T_{min} is 1.0s, this condition is evident as all other solutions have better accuracy even with less or similar telemetry overhead. Lastly, we see that DINT can keep one of the most accurate network views while having the smallest telemetry overhead in all scenarios. For example, when T_{min} is 0.25s, DINT has 60% less telemetry overhead at just 20% less accuracy compared to the most accurate solution, i.e., LINT. Even better, when T_{min} is 0.25s, DINT outperforms all other solutions; it has the best accuracy and the lowest telemetry overhead.

5. Related Work

The advent of P4 and INT prompted the development of multiple monitoring approaches with varying end goals. In DyPro [Dallanora et al. 2022], the authors formalize the problem of ensuring that telemetry dependencies are always satisfied under monitoring application requirements as a Mixed-Integer Linear Programming (MILP) optimization model and propose a heuristic that wisely finds a high-quality solution. Although DyPro successfully satisfies telemetry dependencies according to monitoring application requirements, its main focus is to define the optimal probe packet path to collect all network information according to different applications. In contrast, DINT's main idea is to reduce network overhead by adjusting the period it collects network information.

One of INT's most challenging aspects is balancing an accurate network view while keeping telemetry overhead minimal. To achieve this, sINT [Kim et al. 2018] reduces the number of telemetry reports by adjusting the insertion ratio of the INT header according to thresholds particular to the monitored data. Despite sINT reducing telemetry overhead compared to the naive INT implementation, it requires some complex threshold tuning for each type of traffic, an issue our method overcomes through dynamic threshold reconfiguration. Parallelly, PINT [Ben Basat et al. 2020] is a probabilistic framework for in-band telemetry that provides similar visibility to INT while bounding the per-packet overhead to a user-specified value. PINT improves INT's default operation mode by the probabilistic encoding of a flow's relevant data into several packets. DINT is complementary to PINT since its purpose is to spread out information over multiple packets to minimize the per-packet overhead, while DINT determines when to collect network information to reduce the overall telemetry overhead.

Still finding ways to achieve this balance, the authors of *Sel-INT* [Tang et al. 2019] proposed and designed a run-time programmable selective INT scheme based on Protocol Oblivious Forwarding (POF) and implemented it in the OpenvSwitch (OVS) platform. More specifically, the POF controller installs INT-related flow tables in the POF switches on the forwarding path of a flow. The controller determines the sampling rate by analyzing historical INT data with the Fourier transform. Cui et al. proposes SPT [Cui et al. 2022], a sketch-based in-band network telemetry measurement framework. SPT uses multiple techniques to reduce network overhead and provide flexible control. Some

of the techniques used are sketch-guided elephant flow selection, compression of control instructions, and decomposition of measurement tasks of one packet into separate measurement tasks carried by multiple sequential packets. In contrast, DINT works directly on the data plane, meaning there is no communication delay between the switches and the controller.

Even though using a centralized controller to determine the telemetry sampling rate allows for more complex algorithms, the communication delay between the switching devices and the controller can be a cause for concern. To tackle this issue, LINT [Chowdhury et al. 2021] proposes an accuracy-adaptive and Lightweight INT mechanism that uses the Exponentially Weighted Moving Average (EWMA) to keep up with network status. One of the limitations of LINT is that it works packet-by-packet, which can lead to substantial telemetry overhead if an optimal configuration is not found. Moreover, suppose no significant change happens in LINT; In that case, the monitoring engine stops receiving information and is oblivious to the latest network status. Differently, DINT works by time intervals, meaning it's harder to saturate the monitoring engine with excessive telemetry resulting from unfit configuration and the monitoring engine can always expect to receive the network status according to the time intervals defined. In DeltaINT [Sheng et al. 2021], the authors propose to reduce INT overhead by selectively carrying network states only when their values change significantly with a data plane native solution. By using a sketch data structure, it can monitor multiple network states. However, DeltaINT thresholds are static and are not updated according to the latest network status, in contrast to DINT's functioning.

6. Conclusion

Balancing the amount of telemetry data collected while keeping an accurate network view is crucial for monitoring applications. In this work, we achieve this goal by proposing DINT, a dynamic and efficient in-band network telemetry algorithm. DINT can adapt itself to send more network metadata when a significant change occurs and actively updates what is considered a significant change based on the latest network traffic. We have evaluated and compared DINT's performance with three other INT-based techniques, including a state-of-the-art one. Our comparison has shown that DINT is the most adaptive, as it always has one of the most accurate network views, even though it uses the least amount of telemetry data in every scenario.

In future work, we plan to evaluate DINT with more realistic network traffics and topologies scenarios to consolidate it as a proper monitoring solution. Finally, we also intend to use DINT with an implementation of the INT specification [Group 2020] since INT is the main in-band telemetry framework for P4.

References

Ben Basat, R., Ramanathan, S., Li, Y., Antichi, G., Yu, M., and Mitzenmacher, M. (2020). Pint: Probabilistic in-band network telemetry. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*, pages 662–680.

Bosshart, P., Daly, D., Gibb, G., Izzard, M., McKeown, N., Rexford, J., Schlesinger, C., Talayco, D., Vahdat, A., Varghese, G., et al. (2014). P4: Programming protocol-

independent packet processors. *ACM SIGCOMM Computer Communication Review*, 44(3):87–95.

Chowdhury, S. R., Bari, M. F., Ahmed, R., and Boutaba, R. (2014). Payless: A low cost network monitoring framework for software defined networks. In *2014 IEEE Network Operations and Management Symposium (NOMS)*, pages 1–9. IEEE.

Chowdhury, S. R., Boutaba, R., and François, J. (2021). Lint: Accuracy-adaptive and lightweight in-band network telemetry. In *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 349–357. IEEE.

Cui, M., Li, X., Wang, Y., Niu, T., and Yang, F. (2022). Spt: sketch-based polling in-band network telemetry. In *NOMS 2022-2022 IEEE/IFIP Network Operations and Management Symposium*, pages 1–7. IEEE.

Dallanora, L. M., Castro, A. G., da Costa Filho, R. I., Rossi, F. D., Lorenzon, A. F., and Luizelli, M. C. (2022). Dypro: Dynamic probing planning for in-band network telemetry. In *2022 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE.

Group, T. P. A. W. (2020). In-band network telemetry (int) data plane spec- ification.

Kaur, S., Kumar, K., and Aggarwal, N. (2021). A review on p4-programmable data planes: Architecture, research efforts, and future directions. *Computer Communications*, 170:109–129.

Kfoury, E. F., Crichigno, J., and Bou-Harb, E. (2021). An exhaustive survey on p4 programmable data plane switches: Taxonomy, applications, challenges, and future trends. *IEEE Access*, 9:87094–87155.

Kim, C., Sivaraman, A., Katta, N., Bas, A., Dixit, A., and Wobker, L. J. (2015). In-band network telemetry via programmable dataplanes. In *ACM SIGCOMM*, volume 15.

Kim, Y., Suh, D., and Pack, S. (2018). Selective in-band network telemetry for overhead reduction. In *2018 IEEE 7th International Conference on Cloud Networking (CloudNet)*, pages 1–3. IEEE.

Sheng, S., Huang, Q., and Lee, P. P. (2021). Deltaint: Toward general in-band network telemetry with extremely low bandwidth overhead. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–11. IEEE.

Tan, L., Su, W., Zhang, W., Lv, J., Zhang, Z., Miao, J., Liu, X., and Li, N. (2021). In-band network telemetry: A survey. *Computer Networks*, 186:107763.

Tang, S., Li, D., Niu, B., Peng, J., and Zhu, Z. (2019). Sel-int: A runtime-programmable selective in-band network telemetry system. *IEEE transactions on network and service management*, 17(2):708–721.

Tangari, G., Tuncer, D., Charalambides, M., Qi, Y., and Pavlou, G. (2018). Self-adaptive decentralized monitoring in software-defined networks. *IEEE Transactions on Network and Service Management*, 15(4):1277–1291.

Yu, M. (2019). Network telemetry: towards a top-down approach. *ACM SIGCOMM Computer Communication Review*, 49(1):11–17.