Accelerating Mega-Scale Satellite Network Simulation in NS-3 via MPI-based Parallelization

Haibin Song*, Tian Pan*†§, Guohao Ruan*, Yan Zheng†, Ying Wan‡, Jiao Zhang*†, Tao Huang*†, Yunjie Liu*†
*Beijing University of Posts and Telecommunications, Beijing, China
†Purple Mountain Laboratories, Nanjing, China
‡China Mobile (Suzhou) Software Technology Co., Ltd, Suzhou, China
§Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education, China
*{haibin, pan, ghruan, jiaozhang, htao, liuyj}@bupt.edu.cn †zhengyan@pmlabs.com.cn ‡wanying2@cmss.chinamobile.com

***Abstract*—Due to the high costs of low Earth orbit (LEO) satellite manufacturing and launch, as well as the complexity of in-orbit network protocol debugging, simulating and verifying satellite network protocols on the ground before satellite launch holds significant importance. Compared to the expensive emulation with one-to-one replication, simulation (*e.g.*, using ns-3) can achieve discrete event processing at a relatively lower cost by extending the wall clock time. However, very few studies have used ns-3 for LEO satellite network simulation, facing challenges such as faithfully simulating the on/off state switching of inter-satellite links (ISLs) and achieving simulation performance scalability for high-density satellite constellations. In this work, we propose a system to accelerate mega-scale LEO satellite network simulation in ns-3 via MPI-based parallelization. Specifically, we simulate ISLs based on ns-3's P2P channels/P2P remote channels and achieve run-time link connection/disconnection by implementing stateful traffic dropping inside the network interface. Then, we conduct concurrent simulation with ns-3's parallel and distributed simulation capability and partition the satellite constellation into multiple simulation processes through a hierarchical clustering algorithm and automated scripts, considering satellite locality and inter-process workload balance. Our evaluation shows significant speed improvements via parallelization, *e.g.*, a 373% speedup with 12 processes for LEO-192, and a 156% speedup with 3 processes for LEO-3072.**

I. Introduction

As ground-based network infrastructure continues to mature, an increasing amount of research redirects its focus toward expanding network coverage to diverse and challenging areas, including the ocean, sky, and space. Low Earth orbit (LEO) satellite networks, serving as a valuable complement to ground-based network infrastructure, have recently become a hot topic of interest in both academia and industry. Ongoing research delves into various aspects of LEO satellite networks, such as routing [1–3], mobility management [4, 5], congestion control [6], and more. The research of network protocols and systems typically necessitates testing and verifying new ideas in real production environments. However, considering the high costs of satellite manufacturing and launch, along with the complexity of testing and verifying network protocols in orbit, the ability to simulate or emulate satellite network functionalities on the ground before launching satellites into space would significantly reduce the research and testing costs of satellite networks, facilitating the improvement of protocol design through rapid iteration on the ground.

This work is supported by the National Natural Science Foundation of China (62372053), the research project (NO. Y2024CXT002) of China Mobile (Suzhou) Software Technology Co., Ltd.. Corresponding author: Tian Pan.

In comparison to the simulation of ground-based networks, what are the specific demands and challenges for simulating LEO satellite networks? The first issue stems from the frequent changes in the network topology of LEO satellite constellations, which typically include inter-satellite and satellite-to-ground topologies. In contrast, ground-based network topologies usually remain stable for longer periods. In other words, LEO satellite network simulation needs to faithfully and efficiently realize the frequent changes in the network topology. The second issue lies in the scalable simulation performance for large-scale satellite constellations. As satellite constellations continue to grow in size, there are large constellations with thousands or even tens of thousands of satellites [7]. Simulating such large constellations with only a single process or thread would lead to excessively long wall clock time. Consequently, fully harnessing concurrent simulation capabilities with the modern multi-core or distributed architectures becomes necessary for LEO satellite network simulation [8].

Based on the underlying mechanisms, existing network testbeds can be classified into two categories: emulation and simulation. The former involves a one-to-one mapping of the original system's hardware and software components, providing a more accurate representation of the real-world systems [9–11]. The latter uses models and discrete event processing to replicate the behavior of the original system without emulating it at a one-to-one level [12, 13]. Although emulation can more faithfully reflect the behavior of real systems, it requires the construction of a significant amount of hardware to handle actual network traffic, resulting in higher human labor costs and hardware investments. In emulation, the hardware investments will grow proportionally with the actual size of the emulated network. If hardware resources are insufficient, issues such as packet loss may arise. By contrast, based on discrete event processing, simulation can "exchange time for resources". That is, if simulation hardware resources are insufficient, one can flexibly extend wall clock time to wait for the completion of discrete event processing. As a well-known network simulator, ns-3 [12] is utilized for simulating numerous ground-based network systems. How-

ICC 2024 - IEEE International Conference on Communications | 978-1-7281-9054-9/24/$31.00 ©2024 IEEE | DOI: 10.1109/ICC51166.2024.10622934

ever, there has been limited usage of ns-3 in simulating LEO satellite networks. In the recently proposed ns-3-leo [13], the authors develop a module for ns-3, which includes models for the network mobility and link characteristics of LEO satellite constellations. However, its inter-satellite links (ISLs) are simulated under a broadcast mode where a satellite can broadcast to all surrounding satellites within a predefined distance. This differs from real-world ISL implementations, which use directed beams or even laser light with traffic transmitted to one destination at a time. Moreover, ns-3-leo follows a single-process simulation model, which may have scalability issues for large-sized constellations.

Considering the cost advantages of simulation over emulation, in this study, we continue to use ns-3 for LEO satellite network simulation. To address the limitations of ns-3-leo, we make improvements in the way of ISL simulation as well as the simulation performance scalability. Specifically, we leverage ns-3's point-to-point (P2P) channels to precisely simulate ISLs, replacing ns-3-leo's broadcast-based ISL implementation. Considering that the P2P channel cannot change the link on/off state during run-time through commands like "detach", we make a workaround with traffic dropping inside the network interface. Based on the predictable movement patterns of the satellite constellation, we build a state machine for each satellite to predict its link on/off states. During the transition of the state machine, we conduct a function return in the "TransmitStart()" function to achieve link disconnection or disable the previous function return for link reconnection.

To enhance the scalability of simulation, we utilize ns-3's parallel and distributed simulation capability by partitioning a simulation task into multiple subtasks and connecting them with ns-3's P2P remote channels implemented with MPI (Message Parsing Interface) [14]. In practice, when dividing a constellation simulation task across multiple processes based on orbital planes, all inter-satellite link on/off events occur on the inter-process P2P remote channels. Since the P2P remote channels also do not support run-time connection/disconnection, we apply the proposed link on/off mechanism for the P2P channels to the P2P remote channels. Furthermore, we propose a solution for optimally partitioning a constellation into multiple processes based on a hierarchical clustering algorithm to cluster nearby satellites and balance simulation workload. Finally, using scripts for an exhaustive search, we can determine the optimal number of processes for the best parallel simulation performance.

Our major contributions are summarized as follows.

- We propose a method for simulating ISLs based on ns-3's P2P channels/P2P remote channels. It can achieve link connection/disconnection by implementing stateful traffic dropping inside the network interface. This method is closer to the reality than broadcast-based ISLs in ns-3-leo, and the workaround to achieve link on/off does not require extensive modifications to ns-3's underlying code.
- Leveraging ns-3's parallel and distributed simulation capability, we partition the simulation of an LEO satellite constellation into multiple concurrent processes connected via P2P remote channels, thus achieving scalable simulation. We propose a partitioning strategy for the constellations based on hierarchical clustering and write scripts for searching the optimal number of partitions.
- The evaluation shows significant speed improvements in ns-3 via MPI-based parallelization, *e.g.*, using 12 processes results in a 373% speedup for LEO-192 while using 3 processes leads to a 156% speedup for LEO-3072.

II. Background and Motivation

A. Low Efficiency of Single-Process NS-3 Simulation

To show the lack of scalability in single-process ns-3 simulation, we measure the performance of ns-3-leo as the scale of simulation nodes increases. The experiment is conducted on an Intel Gold 5220 CPU @ 2.20GHz, with the selection of a pair of ground terminals for traffic transmission. The constellation runs the AODV routing protocol. Fig. 1 shows the wall clock time of LEO satellite constellation simulation with ns-3-leo. When simulating a 1033-node constellation for 100s with a packet forwarding rate of 1000pps, the wall clock time reaches 10 hours. When simulating a 4225-node constellation for 100s with a packet forwarding rate of 10pps, the wall clock time reaches 153 hours. It is evident that (1) ns-3-leo's broadcast-based ISL implementation consumes substantial computational resources despite featuring complex ISL modeling; (2) with single-process simulation, the expansion of constellation scale leads to non-scalable simulation performance.

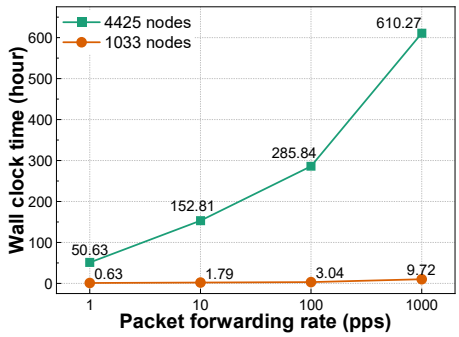

Fig. 1. Wall clock time of simulating two constellations with ns-3-leo [13].

B. NS-3's Parallel and Distributed Simulation

To accelerate simulation efficiency, many simulation tools provide parallel simulation capabilities. For example, ns-3's parallel and distributed simulation involves leveraging multiple logical processes to execute simulations concurrently. This approach utilizes parallel processing techniques to distribute the computational workload across multiple computing resources. Each process operates independently, simulating a portion of the network or system. Communication between processes is facilitated through message passing mechanisms (*e.g.*, MPI). By partitioning the simulation workload and running it in parallel, ns-3 can exploit the computational resources available in multi-core systems or distributed computing environments, thereby accelerating simulation performance and enabling the simulation of larger and more complex networks or systems.

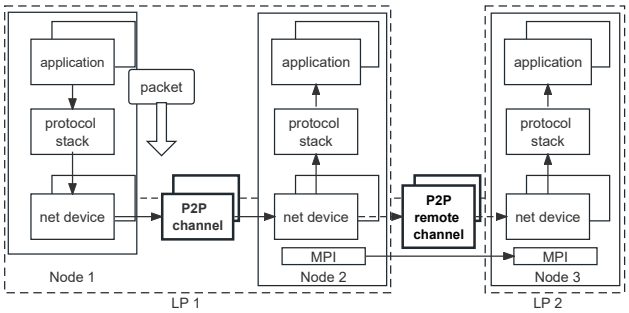

Fig. 2. In ns-3's parallel simulation, P2P channels are for ISLs within a logical process, while P2P remote channels are for ISLs across logical processes.

In ns-3, we can use a P2P channel to connect two network devices (*e.g.*, satellites) if they are within the same logical process. For parallel simulation with multiple logical processes, if two network devices are separated into two logical processes, we have to use a P2P remote channel to connect them, as shown in Fig. 2. For parallel simulation, data exchange across logical processes occurs from time to time for traffic transmission and simulation synchronization. During parallel simulation, ns-3 has the ability of lookahead, which allows each logical process to predict future events and independently advance its simulation time without waiting for frequent synchronization with other logical processes, thus minimizing synchronization overhead and improving simulation efficiency. Once a logical process computes all events within its lookahead phase, it enters the synchronization phase. At this phase, messages are exchanged between logical processes, which contain packets, timestamps, shared data structures, *etc*.

With the above parallel simulation mechanisms, ns-3 is capable of accelerating network simulation with fixed topologies, as P2P channels/P2P remote channels can be established before simulation and remain unchanged afterward. However, due to the highly dynamic nature of ISLs in LEO satellite networks, both intra-process P2P channels and inter-process P2P remote channels need to support run-time disconnection and reconnection. For instance, if we divide an LEO satellite constellation across multiple logical processes based on orbital planes, all inter-satellite link on/off events will occur on the inter-process P2P remote channels. Unfortunately, ns-3 currently does not provide the link detach operation or something similar for both P2P channels and P2P remote channels, posing a major challenge for LEO satellite network simulation.

III. Design and Implementation

A. ISL Simulation with On/Off Link State Switching

Simulating wireless ISLs with wired links. The ns-3-leo utilizes a complex broadcast model to characterize the connectivity of ISLs, which incurs high computational overhead and fails to faithfully simulate real-world ISLs with precise pointing. Considering that our simulation primarily focuses on network and transport layer protocols, the simulation system only needs to provide ISL-level connectivity or congestion status for these protocols. Hence, we employ P2P channels (the wired link model in ns-3) to simulate wireless ISLs with precise pointing. Above P2P channels, the congestion status of ISLs can be simulated by adjusting parameters such as packet loss rate and queuing delay. Furthermore, compared to the complex broadcast model of ns-3-leo, P2P channels have less computational overhead, thereby significantly reducing the wall clock time during simulation. However, P2P channels also have limitations as they cannot characterize the protocol behaviors at the link layer or physical layer.

Satellite network interface open/close prediction. After we choose to use a wired link to simulate the wireless ISL, we need to determine its connectivity based on the real-time position of the satellite. Generally, for polar-orbit LEO satellite constellations, when a satellite is in high-latitude regions, its ISLs will be closed due to antenna tracking limitations. Based on our observation of the movement patterns of polar-orbit constellations, we find that each satellite has 6 stable neighbor satellites. For a given satellite, it is always connected with the 2 neighbor satellites on the same orbit, while alternatively connected with 2 out of the 4 satellites on neighbor orbits. For instance, in Fig. 3, satellite A is connected with {C, E} and {B, D} before and after its crossing the polar zone.

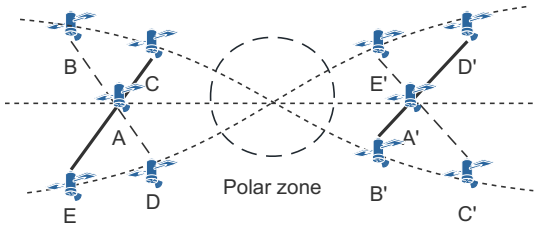

Fig. 3. The ISL switching between a satellite and its neighbor satellites in different orbits when crossing the polar zone.

Assuming for each inter-orbit satellite network interface, once its latitude exceeds a threshold, it will be closed, then we can predict the open/close state of each inter-orbit satellite network interface based on the state machine described in Fig. 4. For each inter-orbit interface, it will visit the 4 states {OPEN, POLAR1, CLOSE, POLAR2} sequentially and periodically. Among the 4 states, the interface will be closed in state POLAR1, POLAR2 and CLOSE, and reopened only in state OPEN. We predict the interface's state based on its position first, rather than directly predicting the ISL's on/off state, because an ISL may span polar and non-polar zones.

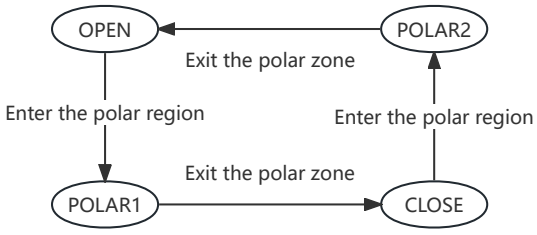

Fig. 4. The state machine of a satellite network interface (*i.e.*, netdevice in ns-3) for its open/close state prediction.

ISL on/off calculation based on associated interface states. Once we have obtained the real-time open/close state of each

interface through the state machine, we can calculate the connectivity of the corresponding ISLs. Since an ISL is associated with two interfaces and the ISL is connected only when both interfaces are turned on, we take the logical AND of the interface states to find the ISL's on/off state. It is worth noting that when any interface is in the POLAR1/POLAR2/CLOSE state, the ISL is considered to be disconnected. As mentioned, the ISL is implemented with ns-3's P2P channel/P2P remote channel, and it has four original states {INITIALIZING, IDLE, TRANSMITTING, PROPAGATING}. In order to avoid modifying the original code for the channels, we introduce a new state called "STOPPING" to represent the disconnection of an inter-orbit ISL. When both interfaces of an ISL go to the OPEN state, the ISL will transition from STOPPING to IDLE, thus entering the normal channel state transition process.

ISL on/off implementation in ns-3. After obtaining the on/off state of an ISL, we can simulate the disconnection of the ISL in ns-3 by actively dropping packets. Specifically, the sender interface's netdevice will call the ScheduleWithContext() function or the SendPacket() function within the TransmitStart() function to send packets within or across simulation logical process. When we determine that the ISL needs to be disconnected (*i.e.*, channel.state = STOPPING), we can return early in TransmitStart(), thereby avoiding executing packet transmission in ScheduleWithContext() or SendPacket(), thus achieving active packet dropping on the netdevice.

B. LEO Satellite Constellation Partition

Hierarchical clustering-based constellation partition. After providing the mechanism for partitioning constellations into multiple processes, it is equally important to specify the partition strategy. Clearly, during partitioning, we should strive to group adjacent satellites into the same process to minimize inter-process communication overhead. Furthermore, we should aim to distribute the traffic processing load as evenly as possible across different processes to fully leverage the computing power on multi-core or distributed nodes.

Accordingly, we utilize the existing hierarchical clustering algorithm [15] to partition the LEO satellite constellations. Hierarchical clustering is a method used in cluster analysis to group similar objects into clusters or groups. It builds a hierarchy of clusters by either iteratively merging smaller clusters into larger ones (agglomerative approach) or by splitting larger clusters into smaller ones (divisive approach). Here, we adopt the agglomerative approach for satellite constellation partition. In the agglomerative hierarchical clustering approach, the algorithm takes the following general steps: 1. each data point starts as its own cluster; 2. at each step, the two closest clusters are merged into a larger cluster; 3. this process continues until all data points belong to a single cluster.

For our specific satellite constellation partition scenario, initially, we let each satellite in an M*N constellation (where M is the number of orbits and N is the number of satellites per orbit) form a cluster, and assign a weight w to each satellite based on its traffic load. Then, we continuously calculate the distance between two clusters considering both the inter-satellite physical distance and the two satellites' traffic load, and merge the closest two clusters. This process is iterated repeatedly until the number of clusters decreases from the initial total number of satellites to the desired number of partitions we need to create. The distance between two clusters can be calculated as

$$distance_{1,2} = w_1 \cdot w_2 \cdot \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

where, w_1, w_2 are traffic load at 2 satellites, and (x_1, y_1), (x_2, y_2) are the coordinates of 2 satellites in the constellation. According to the distance equation, the satellites closer together with lighter traffic loads will be clustered first, ensuring locality during clustering and maintaining balanced traffic loads across different clusters (thus avoiding scenarios where two heavily loaded satellites run on the same process). When two clusters are merged into one cluster, the weight of the new cluster is the sum of the weights of the two previous clusters, and the position of the new cluster is the midpoint of the line connecting the positions of the two previous clusters.

Automated scripts for finding the optimal partition number. When we choose different numbers of partitions, the simulation performance may be different. This depends on many factors, such as the size of the constellation, the traffic model, the number of CPU cores or distributed nodes, and so on. To find the optimal partition number for the best simulation performance, we write a script as the outer loop, continuously testing different numbers of partitions' simulation performance, thereby finding the optimal partition number for performance for each parallel simulation instance.

IV. Evaluation

A. Experimental Settings

In order to evaluate the impact of different factors on the parallel simulation efficiency as well as the costs of parallel simulation, we construct M*N polar-orbit constellations and select a pair of satellites for traffic generation and reception. The LEO satellite constellations run the AODV routing protocol. The experiments are conducted on a server with an Intel Gold 5220 CPU @ 2.20GHz, 16GB of RAM, and Ubuntu 18.04. The parameter setting by default is listed in Table I.

TABLE I
Parameter Setting by Default of NS-3 Parallel Simulation for LEO Satellite Constellations

Major Parameter	Value
Number of satellite orbits (M)	6-48
Number of satellites in each orbit (N)	8-64
Packet forwarding rate	100pps
Packet size	1000B
Simulation time	200s
Polar zone boundary	70 degree
Sending node	Satellite at (M, N/4+1)
Receiving node	Satellite at (1, 1)
Data rate of the channel	5Mbps
Latency of the channel	5ms

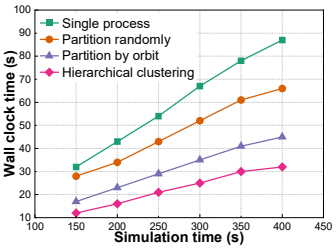

Fig. 5. The acceleration effect of different partition strategies (LEO-48).

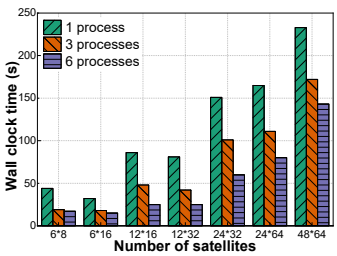

Fig. 6. The acceleration effect at different satellite constellation scales.

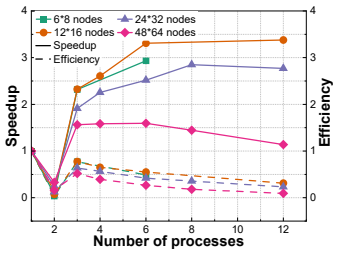

Fig. 7. The acceleration effect with different numbers of logical processes.

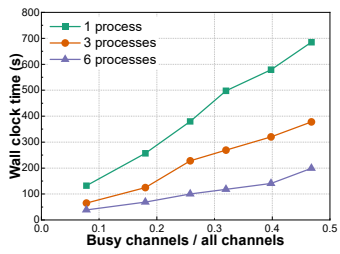

Fig. 8. The acceleration effect by adding more busy channels (LEO-48).

B. Impact of Factors on Parallel Simulation Efficiency

Partition strategies. In Fig. 5, we compare the simulation efficiency of different partition strategies, including single process, partition randomly, partition by orbit, and hierarchical clustering. It can be observed that the hierarchical clustering-based partition yields the best result because it simultaneously considers satellite locality and balancing workload across different partitions. Furthermore, the slight variations in the slope of these curves are noticeable, leading to an accumulation of differences between multi-process and single-process simulations over time. The underlying reason is that multi-process simulations take longer during the initialization phase as they need to initialize within each process. However, as time progresses, the advantages of multi-process simulations become increasingly apparent.

Network scales. Fig. 6 illustrates the impact of varying network scales on simulation efficiency as the number of orbits or satellites within each orbit incrementally increases. The findings indicate that when the number of in-orbit satellites grows, the total wall clock time does not increase significantly, as the number of hops for packets from source to destination has not increased too much. Conversely, increasing the number of orbits results in a proportional increase in the number of hops for packets from source to destination, consequently prolonging the total wall clock time. Additionally, it can be seen that, as the number of satellites increases, the acceleration effect of parallel simulation on ns-3 gradually diminishes.

Numbers of logical processes. Fig. 7 shows the speedup and efficiency of ns-3 parallel simulation of satellite constellations with different numbers of logical processes, where the speedup is defined as $S_p = T_1/T_p$, p refers to the number of processes, T_1 refers to the time for the single-process simulation, T_p refers to the time for the parallel simulation with p processes. Efficiency is another measure of performance, defined as $E_p = S_p/p = T_1/(p \cdot T_p)$. From Fig. 7, we can see that when the number of processes is small, increasing the number of processes leads to better speedup in parallel simulation. However, as the number of processes surpasses a threshold, the speedup effect starts to diminish due to the increasing overhead of synchronization between processes. For mega-scale constellations (*e.g.*, LEO-3072), the maximum speedup is achieved with a small number of processes (*e.g.*, 3). For smaller constellations, the maximum speedup is achieved with a larger number of processes. The efficiency of all constellations reaches maximum at 3 processes. Besides, there is a noise point on Fig. 7: when the number of processes is 2, the speedup of parallel simulation is unexpectedly lower compared to single-process simulation. After extensive debugging, we discover that when there are 2 processes, ns-3 still schedules the simulation tasks of both processes on a single CPU core, leading to degradation in performance. Currently, we are unaware of the exact reason behind the ns-3's scheduling behavior when the number of processes is 2.

Number of busy channels. In Fig. 8, we let more pairs of satellites send and receive traffic to create more busy channels in an LEO-48 constellation and evaluate the corresponding wall clock time for parallel simulation with different number of logical processes. The evaluation illustrates that when the number of busy channels grows, the advantages of parallel simulation become increasingly significant as the efficiency of single-process simulation is throttled by the limited performance of a single CPU core.

Traffic rates. In Fig. 9, we gradually increase the sending rate of traffic from 50pps to 250pps, and then observe the performance when allocating different numbers of logical processors for parallel simulation. The results show that when the traffic sending rate is low, due to the relatively large communication and synchronization overhead between multiple simulation processes, the wall clock time of multi-process (12 processes) simulation is actually higher than that of single-process simulation. However, as the traffic sending rate gradually increases, the efficiency of single-process simulation begins to lag behind multi-process simulation. The slopes of the curves representing different simulation parallelism indicate that increasing the number of simulation processes can effectively handle high-throughput simulation traffic.

C. Costs of Parallel Simulation

Memory consumption. Fig. 10 illustrates the memory consumption of single-process and multi-process LEO satellite network simulations under different numbers of processes and satellite nodes in ns-3. From the figure, it can be seen that as the number of nodes increases, the total memory consumption of the simulation also increases; similarly, as the number of simulation processes increases, the total memory consumption of the simulation also increases. Taking the LEO-3072 constellation as an example, using 12 simulation processes consumes 23.8% more memory than a single process. As shown in Fig. 7, LEO-3072 achieves the best speedup when the number of

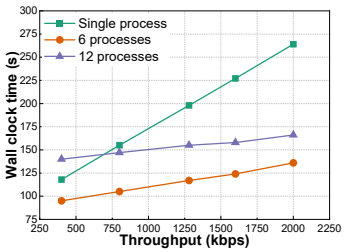

Fig. 9. The acceleration effect under different traffic rates (LEO-3072).

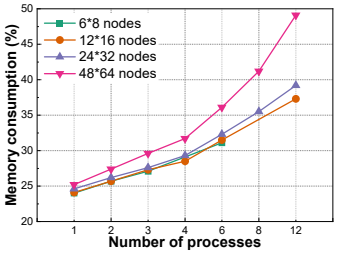

Fig. 10. The memory usage with different numbers of logical processes.

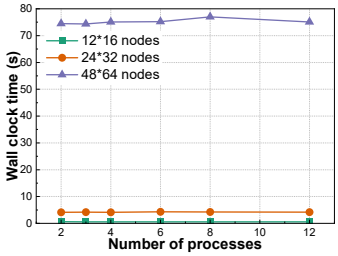

Fig. 11. Time for executing the hierarchical clustering algorithm.

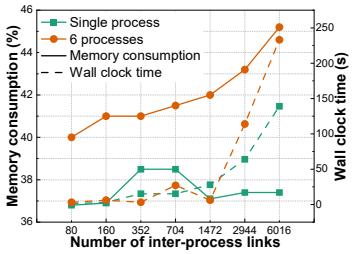

Fig. 12. The memory usage and acceleration effect with different numbers of inter-process links without traffic.

processes is set to 3, with a memory increase of 4.4% (*i.e.*, 0.8GB) compared to a single process.

Time for hierarchical clustering execution. Fig. 11 displays the time required for the hierarchical clustering algorithm to partition for satellite constellations of different sizes. When the number of satellite nodes is 3072, the algorithm's execution time is less than 80s; when the number of satellite nodes is lower in other cases, the execution time is less than 10s. From the figure, it can be observed that the number of clustered nodes is the primary factor affecting the algorithm's execution time. The number of logical processes has a relatively minor impact on the algorithm's execution time.

Cost of inter-process links. Fig. 12 explores the impact of inter-process link (*i.e.*, P2P remote channel) number on memory consumption and wall clock time when there is no data plane traffic in the links. The more links between processes, the greater the overhead of synchronization and communication between these processes. Even if there is no data plane traffic in the links between processes, parallel simulation with ns-3 still requires some additional memory. As the number of links between processes increases, the wall clock time of parallel simulation also becomes longer. In contrast, single-process ns-3 simulation achieves higher efficiency with less inter-process synchronization overhead.

V. Conclusion

In this work, we accelerate mega-scale satellite network simulation in ns-3 via MPI-based parallelization. Specifically, we simulate ISLs based on ns-3's P2P channels/P2P remote channels and achieve run-time link connection/disconnection in LEO satellite constellations by implementing stateful traffic dropping inside the network interface. To scale the simulation performance, we leverage ns-3's parallel and distributed simulation capability and partition the satellite constellation into multiple simulation processes through a hierarchical clustering algorithm and automated search scripts, considering satellite locality and inter-process workload balance. Our evaluation shows significant speed improvements in satellite network simulation via ns-3's parallelization.

References

[1] M. Handley, "Delay is not an option: Low latency routing in space," in *Proceedings of the 17th ACM Workshop on Hot Topics in Networks*, 2018, pp. 85–91.

[2] T. Pan, T. Huang, X. Li, Y. Chen, W. Xue, and Y. Liu, "Opspf: Orbit prediction shortest path first routing for resilient leo satellite networks," in *ICC 2019-2019 IEEE International Conference on Communications (ICC)*. IEEE, 2019, pp. 1–6.

[3] G. Ruan, T. Pan, C. Lu, Z. Luo, H. Wang, J. Zhang, Y. Shen, T. Huang, and Y. Liu, "Lightweight route flooding via flooding topology pruning for leo satellite networks," in *ICC 2022-IEEE International Conference on Communications*. IEEE, 2022, pp. 1149–1154.

[4] J. Hu, T. Pan, Y. Chen, X. Zhang, T. Huang, and Y. Liu, "Lisp-leo: Location/identity separation-based mobility management for leo satellite networks," in *GLOBECOM 2022-2022 IEEE Global Communications Conference*. IEEE, 2022, pp. 1558–1563.

[5] S. Ji, M. Sheng, D. Zhou, W. Bai, Q. Cao, and J. Li, "Flexible and distributed mobility management for integrated terrestrial-satellite networks: Challenges, architectures, and approaches," *IEEE Network*, vol. 35, no. 4, pp. 73–81, 2021.

[6] Z. Wang, J. Zhang, Y. Zhang, T. Pan, and T. Huang, "A transport control protocol for low earth orbit satellite networks based on link information estimation," *Computer Research and Development*, vol. 60, no. 8, pp. 1846 – 1857, 2023.

[7] F. Michel, M. Trevisan, D. Giordano, and O. Bonaventure, "A first look at starlink performance," in *Proceedings of the 22nd ACM Internet Measurement Conference*, 2022, pp. 130–136.

[8] S. Bai, H. Zheng, C. Tian, X. Wang, C. Liu, X. Jin, F. Xiao, Q. Xiang, W. Dou, and G. Chen, "Unison: A parallel-efficient and user-transparent network simulation kernel," 2024.

[9] T. Pan, X.-C. Li, W.-H. Xue, Z.-Z. Bian, T. Huang, and Y.-J. Liu, "A docker-based leo satellite network testbed," *Chinese Journal of Computers*, vol. 45, no. 9, pp. 2029 – 2046, 2022.

[10] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1–6.

[11] Z. Lai, H. Li, Y. Deng, Q. Wu, J. Liu, Y. Li, J. Li, L. Liu, W. Liu, and J. Wu, "{StarryNet}: Empowering researchers to evaluate futuristic integrated space and terrestrial networks," in *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, 2023, pp. 1309–1324.

[12] T. R. Henderson, M. Lacage, G. F. Riley, C. Dowell, and J. Kopena, "Network simulations with the ns-3 simulator," *SIGCOMM demonstration*, vol. 14, no. 14, p. 527, 2008.

[13] T. Schubert, L. Wolf, and U. Kulau, "ns-3-leo: Evaluation tool for satellite swarm communication protocols," *IEEE access*, vol. 10, pp. 11 527–11 537, 2022.

[14] J. Pelkey and G. F. Riley, "Distributed simulation with mpi in ns-3." in *SIMUTools*, 2011, pp. 410–414.

[15] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: an overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012.