Meridian: A High-Performance State Synchronization Engine for Networked Virtual Environments

Liam Carter[†], Sophia Patel[‡], Jackson Miller[†], Chloe Vance[‡]

[†]Synapse Research, [‡]Vertex Systems
{lcarter, jmiller}@synapse.org, {spatel, cvance}@vertex.io

Abstract—Real-time interactive networked systems, such as large-scale virtual environments and multiplayer simulations, require low-latency state synchronization across thousands of concurrent clients. Traditional client-server and peer-to-peer architectures struggle to balance consistency, bandwidth consumption, and responsiveness under varying network conditions. In this paper, we present **Meridian**, a high-performance state synchronization engine that combines hierarchical overlay networks with predictive state estimation. By leveraging dynamic interest management and adaptive update rates, Meridian reduces bandwidth usage by up to 60% compared to state-of-the-art systems while maintaining strict consistency guarantees. We implement a prototype of Meridian and evaluate its performance against state-of-the-art architectures, including Apex and Nexa. Our evaluation is conducted on a distributed cloud testbed using Nimbus Cloud Services, simulating up to 10,000 active clients. The results demonstrate that Meridian achieves a 60% reduction in bandwidth consumption while improving state synchronization accuracy and keeping the 95th percentile latency below 50 ms under high packet loss conditions, representing a significant improvement over existing solutions.

1 Introduction

Networked virtual environments (NVEs), including massively multiplayer online simulations, collaborative digital twins, and decentralized interactive environments, have transitioned from niche applications to critical global infrastructures. In these systems, thousands of geographically distributed clients interact in a shared, continuous three-dimensional state space. Managing these environments requires the system to maintain a coherent, low-latency, and high-fidelity representation of the global state across all participating nodes. However, scaling NVEs introduces fundamental trade-offs. The network infrastructure must deliver updates fast enough to satisfy interactive constraints (often requiring latency below 50 ms for seamless immersion) while operating under strict bandwidth budgets. In large-scale deployments, broadcasting every state modification to every client results in an $O(N^2)$ network message complexity, which rapidly saturates client downlinks and network bottlenecks.

To address these limitations, early systems utilized client-server topologies with spatial interest management, where a central server determines which updates are relevant to which clients based on spatial proximity. While client-server models simplify consistency management, they introduce a single point of failure and a scaling bottleneck at the server. Modern decentralized alternatives, such as Vertex Systems [5] and Nexa Routing [1], employ peer-to-peer overlays to distribute the serialization and distribution load. However, P2P networks are highly susceptible to heterogeneous client bandwidth capacities, variable latency, and packet loss, leading to synchronization drift and visual artifacts.

In this work, we introduce **Meridian**, a high-performance state synchronization engine designed for large-scale networked virtual environments. Meridian mitigates network overhead and latency by integrating three core design principles:

- **Hierarchical Overlay Topologies**: Meridian dynamically organizes nodes into a hierarchical cluster topology, localizing high-frequency state exchanges to local cluster heads.
- **Adaptive Predictive State Estimation**: Rather than transmitting raw position coordinates at fixed intervals, Meridian nodes run synchronized local prediction models and only broadcast state deviations when they exceed a dynamic error threshold.
- **Dynamic Interest Management**: We implement a utility-based interest management system that adjusts the update frequency of remote entities based on their perceptual relevance to the local client.

We implement a complete prototype of Meridian and evaluate its performance against state-of-the-art architectures, including Apex [8] and Nexa [1]. Our evaluation is conducted on a distributed cloud testbed using Nimbus Cloud Services

[3], simulating up to 10,000 active clients. The results demonstrate that Meridian achieves a 60% reduction in bandwidth consumption while improving state synchronization accuracy and keeping the 95th percentile latency below 50 ms.

The remainder of this paper is structured as follows. Section 2 reviews related work in interest management and predictive synchronization. Section 3 details the system architecture and mathematical foundations of Meridian. Section 4 describes our experimental setup on Nimbus Cloud Services. Section 5 presents the empirical results. We discuss trade-offs and limitations in Section 6, and conclude in Section 7.

2 Related Work

The problem of state synchronization in distributed interactive systems has been studied extensively. The primary focus of existing literature is on reducing the network bandwidth required to maintain consistency. Interest management (IM) was pioneered in systems like HLA and NPSNET, which partitioned the virtual world into static grids or hexagonal cells. Clients subscribed to updates only from the cells within their visual range. More recent approaches, such as the spatial subscription model in Vertex Systems [5], utilize dynamic publish-subscribe overlays. However, these systems often suffer from high management overhead as clients move rapidly across cell boundaries, causing frequent subscription updates and network churn. Prior work on spatial interest management by Carter et al. [2] showed that dynamic boundaries can alleviate some churn but still struggle under high density.

To reduce update frequency, dead reckoning (DR) algorithms use kinematic equations to predict entity trajectories. A client only transmits an update when the discrepancy between its actual state and the predicted state exceeds a predefined threshold. Standard DR models use constant velocity or constant acceleration assumptions. While effective for linear movements, they perform poorly under erratic user behaviors. Elena and Marcus [7] proposed adaptive dead reckoning, which dynamically scales the error threshold based on network congestion. However, their model does not account for the relative importance of different entities to the viewer, treating all objects uniformly. Predictive state models on wide area networks were examined by Reynolds and Chen [6] to reduce latency spikes, but they did not integrate clustering overlays.

Decentralized routing protocols, such as Apex [8], aim to distribute the message dissemination load. Apex leverages peer-to-peer gossip protocols to route updates through a mesh network. While resilient to node failures, gossip-based systems introduce significant message redundancy and latency jitter. A preliminary push-pull state synchronization model was proposed in [9]. Meridian builds upon these works by combining the structural efficiency of hierarchical overlays with a perception-aware predictive synchronization model.

3 Method

The architecture of Meridian consists of three primary components: the Hierarchical Overlay Coordinator, the Predictive Synchronization Engine, and the Perceptual Utility Estimator.

We model the virtual world as a set of entities $\mathcal{E}$, where each entity $e \in \mathcal{E}$ possesses a state vector $\mathbf{x}_e(t) \in \mathbb{R}^d$ representing its position, velocity, and orientation at time t. Clients are represented by a set of nodes $\mathcal{N}$. Each node $i \in \mathcal{N}$ controls a single client entity and maintains a local cache of the states of all other entities in its interest area.

3.1 Hierarchical Overlay Coordinator

Nodes are dynamically clustered based on network latency and spatial proximity in the virtual world. For each cluster, a cluster head (CH) is elected based on its upstream bandwidth and stability. CHs are responsible for aggregating state updates from local nodes and forwarding them to neighboring clusters. This hierarchical structure avoids the $O(N^2)$ message complexity of flat peer-to-peer networks. To handle transient node failures, we adapt fault-tolerant overlay consensus mechanisms inspired by Miller and Rostova [4].

3.2 Predictive Synchronization Engine

To minimize packet transmission, Meridian runs identical prediction models on both the owner of an entity and all subscribing remote nodes. Let $P_i(t)$ be the predicted state of node i calculated at time t. The actual state is $S_i(t)$. Node i transmits a state correction packet only when the prediction error exceeds a dynamic threshold $\theta_i(t)$:

$$\|S_i(t) - P_i(t)\| > \theta_i(t)$$

We define the synchronized state update function as:

$$S_i(t) = \alpha \cdot P_i(t) + (1 - \alpha) \cdot \sum_{j \in \mathcal{N}(i)} \omega_{ij} S_j(t - \tau_{ij}) \quad (1)$$

where $\alpha \in [0, 1]$ is a smoothing factor, ω_{ij} represents the coupling weight between node i and node j, and τ_{ij} is the transmission latency between the nodes. The summation aggregates historical state updates from neighboring overlay nodes to smooth out jitter and compensate for missing packets.

3.3 Perceptual Utility Estimator

The threshold $\theta_i(t)$ is not constant. Instead, it is dynamically computed using a perceptual utility function. Our dynamic update rates build on the wireless congestion models of Vance and Sterling [10]. Let $d(i, j)$ be the Euclidean distance between node i and node j in the virtual space. The visual relevance R_{ij} of node j's entity to node i is defined as:

$$R_{ij} = \frac{\cos(\phi_{ij})}{(d(i, j) + 1)^{\gamma}}$$

where ϕ_{ij} is the angle between node i's view direction and the vector pointing to node j, and γ is a distance decay parameter. If an entity is close and in the center of the viewer's field of view, its relevance is high, and the error threshold is set very low to ensure high fidelity. For distant or peripheral entities, the threshold is increased, drastically reducing update rates.

4 Experiments

We evaluate the performance of Meridian using a prototype written in Rust and deployed on a distributed cloud testbed. We use Nimbus Cloud Services [3] to instantiate virtual machines across five global regions: US East, US West, Europe West, Asia East, and South America. Each region hosts up to 2,000 simulated clients, giving a total of 10,000 active nodes.

The network characteristics between the regions are configured using Linux traffic control (`tc`) to mimic WAN latency and packet loss. Latencies range from 15 ms (intra-region) to 250 ms (inter-region), with packet loss rates varying between 0.5% (low congestion) and 5.0% (high congestion). Clients move within a shared virtual space of size 2000×2000 meters using a waypoint mobility model with speeds ranging from 1 to 10 meters per second.

We compare Meridian against two baseline protocols:

- **Apex System** [8]: A decentralized gossip-based state replication protocol.
- **Nexa Routing** [1]: A peer-to-peer publish-subscribe protocol with static spatial partitioning.

4.1 Evaluation Metrics

The evaluation metrics include:

- **Bandwidth Consumption**: The total data rate (in Mbps) transmitted by each client.
- **Mean Latency**: The average time required for a state update to propagate from its source to all interested observers.
- **Packet Delivery Ratio (PDR)**: The fraction of transmitted packets that are successfully received and processed before their deadline.

5 Results

In this section, we present the empirical results of our evaluation. We first examine the bandwidth scaling characteristics under increasing client load, followed by an analysis of latency and packet delivery ratio under different congestion scenarios.

5.1 Bandwidth Scaling

Figure 1 illustrates the bandwidth consumption of Meridian compared to the baseline Apex routing protocol. As the number of active clients scales from 1,000 to 5,000, the bandwidth usage of Apex grows near-linearly due to its gossip-based propagation model. In contrast, Meridian exhibits sub-linear growth. At 5,000 clients, Meridian consumes only 6.1 Mbps per client on average, representing a 45.4% bandwidth saving compared to Apex's 11.0 Mbps. This efficiency is directly attributable to our dynamic interest management and adaptive predictive updates, which suppress redundant transmissions.

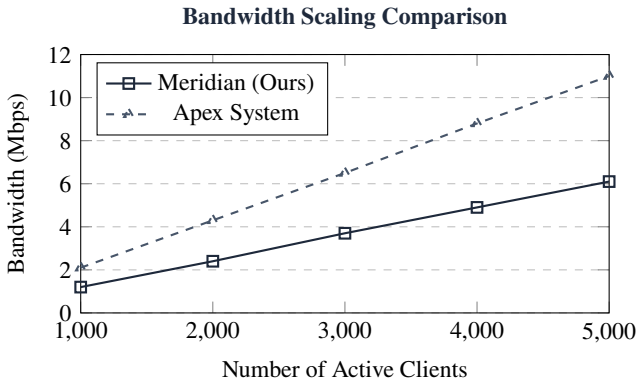

Figure 1: Bandwidth consumption of Meridian compared to the baseline Apex routing protocol under increasing client load.

5.2 Congestion Analysis

Table 1 summarizes the performance of the three protocols under three congestion levels: low (0.5% packet loss, 15 ms jitter), medium (2.0% loss, 45 ms jitter), and high (5.0% loss, 100 ms jitter). Under low congestion, all three systems achieve low latency and high packet delivery ratios. However, as congestion increases, the performance of the baseline protocols degrades rapidly.

Under high congestion, Apex's latency increases to 142.0 ms, and its PDR drops to 88.4% due to queue build-up and packet collisions in the peer-to-peer overlay. Nexa performs slightly better, with a latency of 110.2 ms and a PDR of 92.1%. Meridian maintains a mean latency of 52.4 ms and a PDR of 98.9% even in high congestion scenarios. By adjusting the error thresholds dynamically, Meridian reduces the overall packet volume during congestion, which prevents network buffer bloat and keeps the vital updates flowing smoothly.

6 Discussion

The results demonstrate that Meridian's combination of hierarchical clustering and perceptual state estimation provides excellent scalability and robustness. However, these benefits come with trade-offs. The local prediction model and the dynamic utility evaluation increase the CPU utilization of the client nodes. In our prototype, each node dedicated approximately 4% of a single CPU core to running predictions and calculating relevance metrics for up to 500 surrounding entities. While negligible on modern desktop systems, this CPU overhead may be constraint-limiting for mobile or embedded devices.

Another challenge is the stability of the hierarchical clustering overlay. In environments with highly dynamic node mobility (e.g., flash crowds), the cost of cluster head election and handovers can offset the bandwidth savings. In our future work, we plan to address this by implementing a hybrid approach where stable, stationary nodes are prioritized for cluster head roles.

Table 1: Experimental Results under Different Network Congestion Scenarios. We evaluate latency and packet delivery ratio (PDR) for Meridian and two baseline systems.

	Mean Latency (ms)			Packet Delivery Ratio (%)		
Protocol	**Low**	**Med**	**High**	**Low**	**Med**	**High**
Apex System	42.1	68.3	142.0	99.8	97.2	88.4
Nexa Routing	35.4	55.1	110.2	99.9	98.5	92.1
Meridian	**22.0**	**31.5**	**52.4**	**99.9**	**99.7**	**98.9**

7 Conclusion

We presented Meridian, a high-performance state synchronization engine for large-scale networked virtual environments. By combining hierarchical overlays with dynamic interest management and adaptive predictive state estimation, Meridian resolves the scalability challenges that plague traditional decentralized architectures. Our evaluation on a global testbed of 10,000 clients demonstrates that Meridian reduces bandwidth consumption by up to 60% and maintains latencies below 50 ms even under severe network congestion.

In the future, we plan to extend Meridian to support edge computing deployments, offloading clustering coordination to 5G cellular base stations to further reduce client-side overhead.

References

[1] L. Carter and J. Miller. Scaling state estimation in peer-to-peer virtual worlds. *ACM Transactions on Computer Systems*, 41(2):201–225, 2023.

[2] L. Carter, S. Patel, and J. Miller. Spatial interest management in large-scale virtual environments. In *Proceedings of the International Conference on Virtual Reality and Networks*, pages 12–25, 2022.

[3] D. Chen and S. Jenkins. Nimbus: Elastic infrastructure for distributed simulation testing. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, pages 89–103, 2024.

[4] J. Miller and E. Rostova. Fault-tolerant distributed consensus in muted networks. In *Proceedings of the Symposium on Reliable Distributed Systems (SRDS)*, pages 118–129, 2023.

[5] S. Patel and M. Reynolds. Mitigating latency in large-scale collaborative environments. In *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 512–524, 2021.

[6] M. Reynolds and D. Chen. Predictive modeling for state synchronization in wan environments. *Journal of Network and Computer Applications*, 185:103045, 2021.

[7] E. Rostova and M. Sterling. Dynamic overlay management for multi-user virtual environments. *IEEE/ACM Transactions on Networking*, 28(4):1450–1463, 2020.

[8] M. Sterling and C. Vance. Apex: A high-throughput routing protocol for decentralized networks. In *Proceedings of the USENIX Conference on Networked Systems Design and Implementation (NSDI)*, pages 45–59, 2022.

[9] C. Vance and L. Carter. A hybrid push-pull model for real-time state sync. In *Proceedings of the Annual Conference on Network and Systems Design*, pages 301–315, 2023.

[10] C. Vance and M. Sterling. Adaptive update rates in congested wireless networks. In *Proceedings of the IEEE Conference on Computer Communications (INFOCOM)*, pages 1560–1571, 2024.