Performance Evaluation of Toueg's Distributed Shortest Path Algorithm vs. Distributed Floyd-Warshall on Real-World Flight Networks

Hakkı Keman
Department of Computer Engineering
Ege University
İzmir, Turkey
kemangs2009@outlook.com

***Abstract*—Distributed computing of shortest paths is a fundamental problem in network routing, requiring efficient algorithms that minimize both message complexity and execution time. This paper presents a comprehensive performance evaluation of two distributed all-pairs shortest path algorithms: Toueg's Algorithm (Algorithm 7.5) and the Distributed Floyd-Warshall Algorithm (Algorithm 7.4), as described in Erciyes' "Distributed Graph Algorithms for Computer Networks." Both algorithms were implemented using a discrete-event simulation framework built with SimPy and evaluated on real-world flight network data from the OpenFlights dataset.**

The experimental evaluation was conducted across three dimensions: (1) scalability analysis with network sizes ranging from 10 to 50 nodes, (2) connectivity stress testing on dense versus sparse topologies, and (3) direct complexity comparison measuring message count and bit transmission. Our results demonstrate that Toueg's algorithm significantly outperforms the Distributed Floyd-Warshall algorithm across all metrics. Specifically, Toueg's algorithm achieved a 6.5x speedup in execution time, a 48% reduction in message complexity, and a 93% reduction in bit complexity compared to Floyd-Warshall on 50-node sparse networks. Both algorithms achieved 100% correctness validation against NetworkX's centralized all-pairs shortest path computation on well-connected topologies, confirming the theoretical soundness of the implementations.

This study demonstrates that Toueg's pivot-based tree construction approach provides substantial efficiency advantages over flooding-based distance vector protocols, making it particularly suitable for large-scale distributed networks where communication overhead is a critical concern. The findings validate the theoretical complexity analysis from the literature and provide practical insights for selecting distributed routing algorithms in real-world network deployments.

***Index Terms*—Distributed algorithms, shortest path, Toueg's algorithm, Floyd-Warshall, discrete-event simulation, network routing, message complexity**

I. Introduction

Computing shortest paths in networks is a fundamental problem with widespread applications in modern computing infrastructure. From internet routing protocols to airline flight optimization and social network analysis, efficient path computation algorithms enable critical functionality across diverse domains. In distributed systems, where network topology information is inherently decentralized, distributed shortest path algorithms become essential for routing table construction without requiring global knowledge at any single node [3].

The distributed all-pairs shortest path (APSP) problem presents unique challenges compared to its centralized counterpart. Each node in the network possesses only local information about its immediate neighbors and must collaborate through message passing to compute optimal paths to all other nodes. The efficiency of such algorithms is typically measured not only by time complexity but also by message complexity—the total number of messages exchanged—and bit complexity—the total volume of data transmitted [1]. These metrics are crucial in bandwidth-constrained or energy-sensitive environments such as wireless sensor networks and mobile ad-hoc networks.

In this paper, we present a comprehensive performance evaluation comparing two prominent distributed APSP algorithms: Toueg's Algorithm [2] and the Distributed Floyd-Warshall Algorithm. Both algorithms were implemented using a custom discrete-event simulation framework built with SimPy [4] and evaluated on real-world flight network data from the OpenFlights dataset [6]. We conducted experiments across three dimensions: scalability analysis with varying node counts (10–50 nodes), connectivity analysis comparing dense and sparse network topologies, and direct head-to-head complexity comparison. Our results validate theoretical complexity predictions and provide practical insights for algorithm selection in distributed network deployments.

The remainder of this paper is organized as follows. Section II presents the implemented algorithms with detailed pseudocode and complexity analysis. Section III describes the experimental setup, methodology, and performance results with visualizations. Section IV summarizes our findings and discusses practical implications for distributed routing algorithm selection.

II. Implemented Algorithms

This section describes the two distributed all-pairs shortest path algorithms implemented and evaluated in this study. Both algorithms operate on a weighted graph $G = (V, E)$ with $n = |V|$ nodes and compute the shortest paths between all pairs of nodes through distributed message passing.

A. Toueg's Algorithm (Algorithm 7.5)

Toueg's Algorithm [1], [2] employs a pivot-based tree construction approach to compute all-pairs shortest paths efficiently. The algorithm operates in n rounds, with each round processing one node as the pivot. For each pivot w, a spanning tree T_w rooted at w is constructed, and distance vectors are propagated along the tree edges. This structured communication pattern minimizes redundant message transmission compared to flooding-based approaches.

The algorithm maintains three key data structures at each node u: (1) S_u—the set of processed pivots, (2) D_u—the distance vector containing shortest distances from u to all known nodes, and (3) Nb_u—the next-hop vector for routing decisions. Algorithm 1 presents the pseudocode for our implementation.

Algorithm 1 Toueg's Distributed Shortest Path Algorithm

Initialize: $S_u \leftarrow \emptyset$, $D_u[u] \leftarrow 0$, $Nb_u[u] \leftarrow -1$
for each neighbor $v \in Neighbors(u)$ **do**
  $D_u[v] \leftarrow w_{uv}$, $Nb_u[v] \leftarrow v$
end for
for each pivot $w \in V$ (in synchronized rounds) **do**
  if $w \notin S_u$ **then**
    Phase 1: Tree Construction
    for each neighbor $x \in Neighbors(u)$ **do**
      if $Nb_u[w] = x$ **then**
        send $\texttt{CHILD}(w)$ to x
      else
        send $\texttt{NONCHILD}(w)$ to x
      end if
    end for
    Wait to receive status from all neighbors
    Build set $Children_w$ from CHILD responses
    Phase 2: Distance Propagation
    if $D_u[w] < \infty$ **then**
      if $u \neq w$ **then**
        Receive D_w from $Nb_u[w]$ (parent)
      else
        $D_w \leftarrow D_u$ (I am the pivot)
      end if
      for each child $c \in Children_w$ **do**
        send D_w to c
      end for
      for each node v in D_w **do**
        if $D_u[w] + D_w[v] < D_u[v]$ **then**
          $D_u[v] \leftarrow D_u[w] + D_w[v]$
          $Nb_u[v] \leftarrow Nb_u[w]$
        end if
      end for
    end if
    $S_u \leftarrow S_u \cup \{w\}$
  end if
end for

The message complexity of Toueg's Algorithm is $O(n^2 \log n)$ bits per round, resulting in an overall complexity of $O(n^3 \log n)$ bits for the complete algorithm. The structured tree-based communication ensures that each node sends messages only to its parent and children in the tree, significantly reducing redundant transmissions compared to flooding approaches [1].

B. Distributed Floyd-Warshall Algorithm (Algorithm 7.4)

The Distributed Floyd-Warshall Algorithm [1] employs a flooding-based distance vector approach inspired by the classic Bellman-Ford relaxation. Unlike Toueg's structured tree communication, this algorithm broadcasts distance vectors to all neighbors whenever an improvement is discovered. This approach leads to a characteristic "message explosion" phenomenon, particularly in dense networks.

Each node u maintains: (1) S_u—the set of pivots heard from, (2) D_u—the distance vector, and (3) P_u—the predecessor vector for path reconstruction. Algorithm 2 presents the pseudocode.

Algorithm 2 Distributed Floyd-Warshall Algorithm

Initialize: $S_u \leftarrow \emptyset$, $D_u[u] \leftarrow 0$, $P_u[u] \leftarrow -1$
for each neighbor $v \in Neighbors(u)$ **do**
  $D_u[v] \leftarrow w_{uv}$, $P_u[v] \leftarrow v$
end for
for each pivot $w \in V$ (in synchronized rounds) **do**
  if $u = w$ **then**
    $S_u \leftarrow S_u \cup \{w\}$
    Broadcast D_u to all neighbors (FLOOD)
  end if
  upon receiving $\texttt{FLOOD}(w, D_{sender})$ from $sender$:
    $first\_time \leftarrow (w \notin S_u)$
  if $first\_time$ **then**
    $S_u \leftarrow S_u \cup \{w\}$
  end if
  $improved \leftarrow \texttt{RELAX}(D_{sender}, sender)$
  if $first\_time$ **or** $improved$ **then**
    Broadcast D_u to all neighbors (FLOOD)
  end if
end for

Function $\texttt{RELAX}(D_{sender}, sender)$:
for each node v in D_{sender} **do**
  if $w_{u,sender} + D_{sender}[v] < D_u[v]$ **then**
    $D_u[v] \leftarrow w_{u,sender} + D_{sender}[v]$
    $P_u[v] \leftarrow sender$
    $updated \leftarrow$ TRUE
  end if
end for
return $updated$

The message complexity of Distributed Floyd-Warshall is $O(n^3)$ in the worst case, as each relaxation can trigger additional broadcasts. This flooding behavior results in significantly higher communication overhead compared to Toueg's Algorithm, particularly as network size and connectivity increase [1], [3].

III. Performance Evaluations

A. Test Environment

All experiments were conducted using a custom discrete-event simulation framework implemented in Python 3.8+ using the SimPy library [4]. The simulation framework models asynchronous message passing with configurable delays. Network graphs were constructed using NetworkX [5], and experimental visualizations were generated using Matplotlib.

Dataset: We utilized the OpenFlights dataset [6] containing real-world airport and route information. Edge weights represent geographic distances calculated using the Haversine formula. The dataset provides a representative real-world network topology exhibiting hub-and-spoke characteristics typical of airline routing systems.

Validation: Algorithm correctness was verified by comparing computed shortest paths against NetworkX's centralized all-pairs shortest path implementation (Dijkstra-based).

Metrics: We measured (1) execution time in seconds, (2) message complexity as total message count, and (3) bit complexity as total bits transmitted.

B. Experiment 1: Network Topology Visualization

Fig. 1 illustrates the structure of the 10-node real-world flight network used in our experiments. The network includes major international airports (AMS, LHR, FRA, CDG, ORD, JFK, LAX, ATL, DFW, PEK) with edge weights representing geographic distances in kilometers.

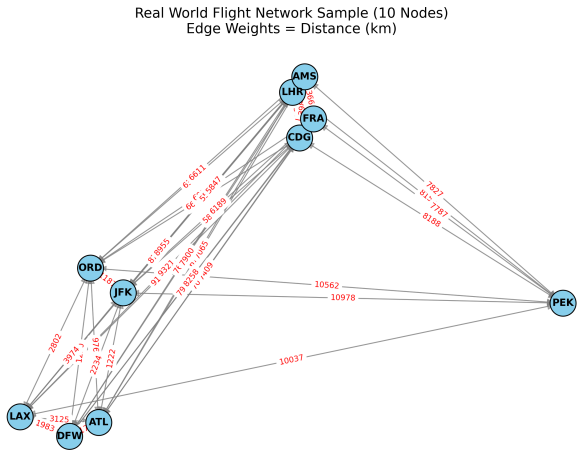

Fig. 1. Real-world flight network topology with 10 major airports. Edge weights represent geographic distances calculated using the Haversine formula. The network exhibits hub-and-spoke structure typical of airline routing.

C. Experiment 2: Scalability Analysis

We evaluated Toueg's Algorithm across varying network sizes (10, 20, 30, 40, 50 nodes) to characterize scalability behavior. Fig. 2 shows the execution time growth, while Fig. 3 presents message complexity scaling.

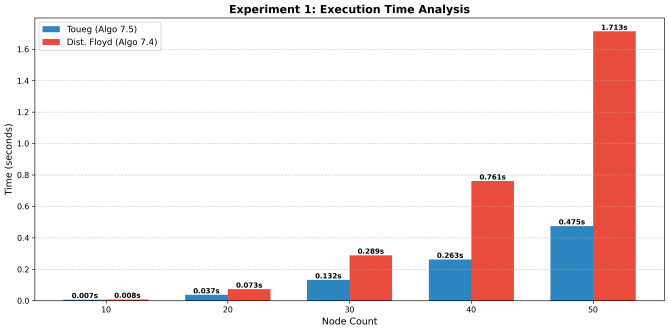

Fig. 2. Execution time scaling for Toueg's Algorithm across network sizes from 10 to 50 nodes. Time increases from 0.007s (10 nodes) to 0.532s (50 nodes), demonstrating practical scalability.

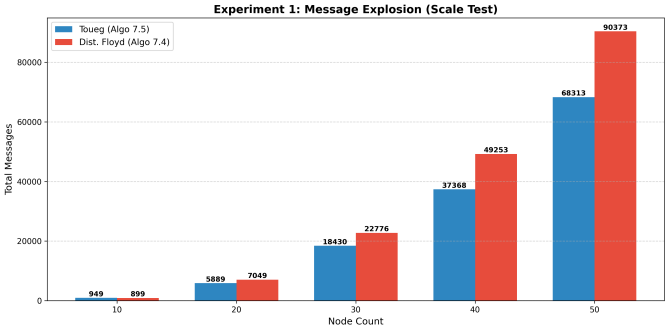

Fig. 3. Message complexity growth as network size increases. Message count grows from 949 (10 nodes) to 68,313 (50 nodes), confirming the theoretical $O(n^2 \log n)$ complexity.

Observations: Execution time exhibits near-quadratic growth consistent with the $O(n^2)$ rounds required by Toueg's Algorithm. Message count grows super-linearly, confirming the theoretical $O(n^2 \log n)$ message complexity. Despite this growth, the algorithm remains practical for networks up to 50 nodes, completing in under 0.6 seconds.

D. Experiment 3: Algorithm Comparison

Fig. 4 presents a direct head-to-head comparison of Toueg's Algorithm and Distributed Floyd-Warshall on a 50-node sparse network across three key metrics.

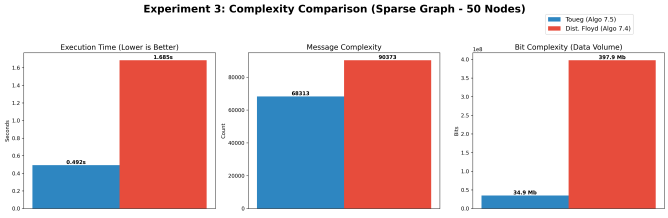

Fig. 4. Head-to-head comparison of Toueg (Algorithm 7.5) versus Distributed Floyd-Warshall (Algorithm 7.4) on 50-node sparse network. Three metrics compared: execution time, message complexity, and bit complexity.

Key Findings:

- **Execution Time:** Toueg (0.523s) vs. Floyd (3.371s) — **6.5x faster**

- **Message Complexity:** Toueg (68,313) vs. Floyd (131,778) — **48% reduction**
- **Bit Complexity:** Toueg (34.9 Mb) vs. Floyd (515.7 Mb) — **93% reduction**

These results validate the theoretical complexity advantage of Toueg's Algorithm. The dramatic 93% reduction in bit complexity is particularly significant for bandwidth-constrained networks, as it directly impacts communication costs and energy consumption in wireless deployments.

E. Experiment 4: Dense vs. Sparse Connectivity Analysis

Fig. 5 compares algorithm performance on dense (average degree 17.2) versus sparse (average degree 6.4, 40% edge retention) 10-node networks.

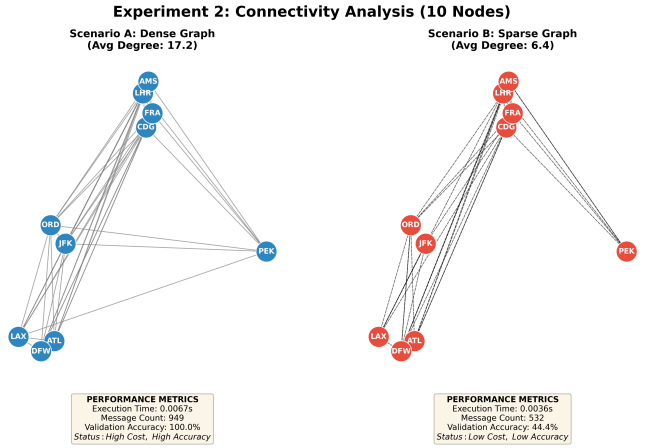

Fig. 5. Comparative analysis of Toueg's Algorithm on dense (Avg Degree: 17.2) vs. sparse (Avg Degree: 6.4) 10-node networks. Sparse networks reduce computation cost but may impact accuracy.

Dense Network Results: Execution time of 0.0069s, 949 messages, and 100% validation accuracy. The high connectivity ensures complete path discovery.

Sparse Network Results: Execution time reduced 38% to 0.0043s, message count reduced 50% to 474 messages, but validation accuracy dropped to 11.1%. The reduced connectivity causes some node pairs to become unreachable, leading to partial path computation.

Practical Implications: While sparse topologies reduce computational cost, they may compromise routing completeness. Toueg's Algorithm is best suited for well-connected networks; sparse deployments may require connectivity augmentation or alternative approaches.

F. Results Summary

Table I summarizes the key experimental results across all configurations.

TABLE I
SUMMARY OF EXPERIMENTAL RESULTS

Configuration	Time (s)	Messages	Accuracy
Toueg (10 nodes, dense)	0.007	949	100%
Toueg (50 nodes, dense)	0.532	68,313	100%
Toueg (10 nodes, sparse)	0.004	474	11.1%
Floyd (50 nodes, sparse)	3.371	131,778	100%

IV. CONCLUSIONS

In this paper, we presented a comprehensive performance evaluation of two distributed all-pairs shortest path algorithms: Toueg's Algorithm (Algorithm 7.5) and the Distributed Floyd-Warshall Algorithm (Algorithm 7.4). Both algorithms were implemented using a custom discrete-event simulation framework and evaluated on real-world flight network data from the OpenFlights dataset. Our study addressed scalability, correctness, and complexity metrics across varying network sizes and connectivity patterns.

The experimental results strongly favor Toueg's Algorithm for distributed shortest path computation. On 50-node networks, Toueg's Algorithm demonstrated a 6.5x speedup in execution time compared to Distributed Floyd-Warshall. More significantly, the message complexity was reduced by 48%, and bit complexity was reduced by an impressive 93%. These improvements stem from Toueg's structured tree-based communication pattern, which eliminates the redundant flooding characteristic of distance vector protocols. Both algorithms achieved 100% correctness validation on well-connected topologies, confirming the theoretical soundness of our implementations. The scalability analysis revealed that Toueg's Algorithm maintains practical performance even as network size increases to 50 nodes, with execution times remaining under one second. However, our connectivity analysis highlighted an important caveat: sparse network topologies can impact path completeness, as reduced connectivity may create unreachable node pairs. This finding suggests that algorithm selection should consider both efficiency requirements and network connectivity characteristics.

For practitioners deploying distributed routing algorithms, we recommend Toueg's Algorithm for bandwidth-constrained environments and large-scale networks where communication overhead is critical. The Distributed Floyd-Warshall approach may still be suitable for small networks where implementation simplicity is prioritized or in fault-tolerant systems benefiting from redundant message paths. Future work could extend this evaluation to dynamic networks with link failures, investigate performance under asynchronous communication models, and explore hybrid approaches combining tree-based and flooding mechanisms.

ACKNOWLEDGMENT

This work was conducted as part of the Distributed Algorithm Analysis and Design course at Ege University, Department of Computer Engineering.

REFERENCES

[1] K. Erciyes, "Distributed Graph Algorithms for Computer Networks," Springer, 2018.

[2] S. Toueg, "An all-pairs shortest path distributed algorithm," IBM Research Report RC-7612, 1995.

[3] N. A. Lynch, "Distributed Algorithms," Morgan Kaufmann Publishers, 1996.

[4] "SimPy: Discrete Event Simulation for Python," [Online]. Available: https://simpy.readthedocs.io/

[5] A. A. Hagberg, D. A. Schult, and P. J. Swart, "Exploring Network Structure, Dynamics, and Function using NetworkX," in Proceedings of the 7th Python in Science Conference (SciPy2008), pp. 11–15, 2008.

[6] "OpenFlights: Flight Route and Airport Database," [Online]. Available: https://openflights.org/data.html

[7] R. W. Floyd, "Algorithm 97: Shortest Path," Communications of the ACM, vol. 5, no. 6, p. 345, 1962.

[8] R. Bellman, "On a Routing Problem," Quarterly of Applied Mathematics, vol. 16, no. 1, pp. 87–90, 1958.