RoundRole: Unlocking the Efficiency of Multi-party Computation with Bandwidth-aware Execution

Xiaoyu Fan*§, Kun Chen†, Jiping Yu*, Xin Liu*, Yunyi Chen*, Wei Xu*§
*Tsinghua University, §Shanghai Qi Zhi Institute, †Ant Group
{fanxy98, weixu}@mail.tsinghua.edu.cn, ck413941@antgroup.com
{yjp19, cyy23}@mails.tsinghua.edu.cn, liuxin19@tsinghua.org.cn

***Abstract*—In privacy-preserving distributed computation systems like secure multi-party computation (MPC), cross-party communication is the primary bottleneck. Over the past two decades, numerous remarkable protocols have been proposed to reduce the overall communication complexity, substantially narrowing the gap between MPC and plaintext computations. However, these advances often overlook a crucial aspect: the *asymmetric* communication pattern. This imbalance results in significant bandwidth wastage, thereby "locking" the performance.**

In this paper, we propose RoundRole, a bandwidth-aware execution optimization for secret-sharing MPC. The key idea is to *decouple* the logical roles, which determine the communication patterns, from the physical nodes, which determine the bandwidth. By partitioning the overall protocol into parallel tasks and strategically mapping each logical role to a physical node for each task, RoundRole effectively allocates the communication workload in accordance with the inherent protocol communication volume and the physical bandwidth. This execution-level optimization fully leverages network resources and "unlocks" the efficiency. We integrate RoundRole on top of ABY3, one of the widely used open-source MPC frameworks. Extensive evaluations across nine protocols under six diverse network settings (with homogeneous and heterogeneous bandwidths) demonstrate significant performance improvements, achieving up to 7.1× speedups.

I. Introduction

Secure Multi-party Computation (MPC) enables multiple parties to jointly compute functions over their private inputs while revealing only the final output. This capability has made MPC increasingly relevant for privacy-preserving data analysis. Applications include collaborative data analytics [1]–[3], machine learning [4]–[7], and graph analysis [8]–[10]. One popular MPC scheme is *secret sharing (SS)* [11], which splits data into random shares that reveal nothing individually but can reconstruct the original data when combined. Building upon secret sharing, joint parties can execute a series of MPC protocols like addition, multiplication, and comparisons, thereby enabling the computation of arbitrary functions and achieving "general-purpose" MPC [12]–[14].

Because these protocols are executed interactively over a network, their performance is limited by the cross-party communications. To solve this problem, many researchers focus on reducing the communication complexities of SS protocols, either by reducing the communication rounds or the amount. Representative reductions include faster *matrix multiplication* [13], [15], [16], *sort* [17], [18], constant-round *shuffle* [8], *oblivious RAM (ORAM)* [19]–[22] and *binary search* [23] with less communications. All of these protocols have achieved significant improvements in communication complexity, making MPC more practical.

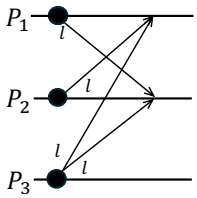


Fig. 1: Communication Pattern of Fixed-point Multiplication Protocol (ABY3 [13]).

However, these optimizations overlook another crucial aspect: *bandwidth utilization* during execution. In fact, we found that the bandwidth utilization of many state-of-the-art protocols remains low because they exhibit highly *asymmetric* communication, i.e., the communication requirements of different parties vary vastly. For example, Figure 1 shows the communication pattern of ABY3 [13] fixed-point multiplication protocol with three *parties*, P_1, P_2, and P_3. P_1 and P_2 need to receive data from both P_2/P_1 and P_3, while P_3 receives nothing but needs to send data to both P_1 and P_2. The asymmetry "locks" the efficiency in two ways: (1) It causes contentions on receiving bandwidth of P_1, P_2, and sending bandwidth of P_3. The slowest party bottlenecks the speed of the entire protocol. (2) It fails to fully utilize the *full-duplex* (both sending and receiving) network that widely exists today, wasting up to 33% of the bandwidth.

The fundamental reason for asymmetric communication is that the main focus of protocol design is to reduce the communication complexity, i.e., the number of rounds and amounts [8], [13], [20]—rather than achieving symmetric communication. Section III shows more examples of protocols with state-of-the-art complexity but significant asymmetric communication. In fact, designing protocols that also ensure symmetric communication workloads among the parties is a challenging task. Harth-Kitzerow et al. [16] address this by

Network and Distributed System Security (NDSS) Symposium 2026
23-27 February 2026, San Diego, CA, USA
ISBN 979-8-9919276-8-0
https://dx.doi.org/10.14722/ndss.2026.240052
www.ndss-symposium.org

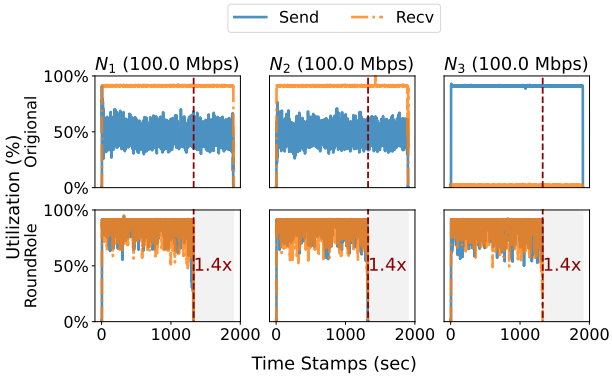


Fig. 2: Communication Traces of Fixed-point Multiplication Protocol (ABY3) without / with RoundRole.

designing specialized multiplication protocols to achieve high bandwidth utilization.

On the other hand, an essential optimization in secret sharing MPC is to use *vectorized execution* to amortize the communication latency and thus improve protocol throughput. When the vector gets large, a natural optimization is to chop up the vector into smaller *blocks* and process them in multiple parallel *tasks* [12], [24]. These vectorized and data-parallelism optimizations are prevalent in practice. Nearly all MPC frameworks and applications adopt these optimizations to achieve better performance [8], [10], [12], [13], [23], [25]–[27]. However, the asymmetric communication limits the gain of parallelization. The top three plots in Figure 2 show the network utilization of the three parties, each on a separate node, running the vectorized and parallel ABY3 fixed-point multiplication protocol above. We can observe that the sending bandwidth of P_1 and P_2 is half utilized, limited by their receiving bandwidth, while the receiving bandwidth of P_3 remains completely idle.

To address the above problem, we propose RoundRole, a simple yet effective bandwidth-aware optimization to improve the existing protocols with asymmetric communication and thus "unlock" the full potential of parallelism on these protocols. The key idea of RoundRole is accepting the communication asymmetry at the protocol level and trying to balance it by carefully mapping the parallel tasks on different computation nodes. In other words, we decouple the *logical roles*, whose communication pattern is determined by the protocol, from the *physical nodes*, whose network utilization determines the performance. This protocol-agnostic, execution-level optimization can be applied to any secret-sharing MPC protocol, thereby effectively utilizing the bandwidth of physical nodes without the need for new protocol designs as shown in [16].

As a concrete example, consider the multiplication above using a data parallel degree of 3. We partition the input vector of length l into three subvectors of size $\frac{l}{3}$, and process them in three parallel tasks. For the three tasks, we assign the logical MPC parties to the nodes using cyclic permutations: P_1, P_2 and P_3; P_3, P_1 and P_2; as well as P_2, P_3 and P_1, respectively. The benefit of this simple optimization is apparent: although communication requirements of the three parties are still uneven (determined by the protocol), the traffic from the *physical nodes* evens out, leading to much better utilization, as Figure 2 shows. We are free to permute roles because, in most secret-sharing–based protocols [13], [16], [25], [28], each node holds a share of the entire input and can therefore act as any logical party.

As a further step, RoundRole generalizes the above example to automatically compute the optimal task splits and assignments for any secret sharing protocol and network setting with negligible overhead, even if the bandwidths are heterogeneous. First, RoundRole profiles the protocol once to characterize its *communication pattern*, i.e., the amount sent and received per unit input for each logical party, which is independent of the underlying network. Then, given the available bandwidths of the nodes, it solves a tailored linear programming problem to compute an *optimal workload allocation ratio* that balances the communication workload across the physical nodes. Once the inputs are provided, RoundRole directly partitions the input vector into multiple subvectors and assigns each a proper logical-to-physical mapping according to the optimal ratios, thereby improving the bandwidth utilization (Section V).

We implement and integrate RoundRole[1] on the widely used open-source ABY3 [13], MOTION [25] MPC framework, and [16] as concrete examples. We evaluate it across six homogeneous and heterogeneous network settings with bandwidths varying from 100 Mbps to 10 Gbps, using nine comprehensive and commonly-used MPC protocols and applications with different communication patterns. Section VI shows that RoundRole can significantly improve the overall bandwidth utilization to almost 100% and thereby substantially improve the execution efficiency. Across all the 54 test cases, RoundRole achieves an average speedup of 1.7× and up to 7.1×. Also, the overhead introduced by the automatic optimization is negligible, demonstrating the practical viability of our approach.

In summary, our contributions include:

(1) We observe the asymmetric communication patterns in secret-sharing MPC protocols, which lead to substantial bandwidth wastage and suboptimal performance.

(2) We propose RoundRole, an automatic bandwidth-aware optimization method for secret-sharing MPC. By decoupling a protocol's fixed logical roles from the physical nodes, RoundRole effectively balances asymmetric communication, fully leverages available bandwidth, and accelerates execution.

(3) We integrate RoundRole on real-world MPC frameworks and conduct comprehensive evaluations on both homogeneous and heterogeneous network settings. Our results show that RoundRole can significantly improve the overall bandwidth utilization and the end-to-end performance of MPC protocols.

II. MPC Background

Secure Multi-party Computation (MPC) allows distrusting parties to compute a function on their private inputs.

[1]The code is available at https://github.com/Fannxy/RoundRole-Scheduling

Secret sharing [29] is a fundamental technique in MPC, based on which many secure applications are built [3], [8], [10], [30]. Typically, for n computation parties, SS splits the input into n shares and allocates the shares to parties, satisfying that any t parties can reconstruct the input while fewer parties learn nothing. The above is known as the (t, n)-*secret sharing scheme*. ABY3 employs $(2, 3)$-SS, where data x is split into three shares, x_1, x_2, x_3, such that $x \equiv x_1 + x_2 + x_3 \mod 2^k$ (Arithmetic shares), $x \equiv x_1 \oplus x_2 \oplus x_3$ (Boolean shares) or 3-party Yao's garbled circuit shares. Each party cyclically holds two shares, ensuring that any two parties can reconstruct x, while any single party remains uninformed. We denote the A-, B- and Y-shares of x as $[\![x]\!]^A$, $[\![x]\!]^B$ and $[\![x]\!]^Y$, respectively. When the specific sharing type is not important, we use $[\![x]\!]$.

With secret shares $[\![x]\!]$ and $[\![y]\!]$, the parties can collaboratively compute the shares of the result $[\![z]\!]$ for a variety of operations (op) using MPC protocols (OP), e.g., XOR, AND, OR, $+$, $\times$, comparisons($>, \geq, =$) and transformations between the $[\![x]\!]^A$ and $[\![x]\!]^B$ [13], [14]. The protocols guarantee both correctness, i.e., for $z = \mathsf{op}(x, y)$, $[\![z]\!] = \mathsf{OP}([\![x]\!], [\![y]\!])$, and privacy, i.e., any parties learn nothing about the secret inputs. During protocol execution, the parties only see *random* shares of the inputs. All the intermediate communication values are *random* in their *view*. Therefore, it is safe to decouple the logical "parties" from the physical computation nodes.

Outsourcing computation model. Secret-sharing protocols are frequently deployed in outsourcing scenarios where secure computation is "outsourced" to a small set of third-party servers. In these settings, data owners locally split their inputs into shares and send the shares to the servers that execute the MPC protocols and return result shares to a designated party for reconstruction [1], [3], [8]–[10], [13], [31]. During the process, the servers only see random shares and learn nothing about the privacy of the data owners. The outsourcing scenarios are broadly adopted by real-world applications, e.g., secure databases [1], [3], graph analysis [8]–[10], and secure machine learning [13], [31]. Because the servers execute an interactive protocol over a network, end-to-end efficiency is largely constrained by communication efficiency, which includes both the volume, the number of rounds, and the bandwidth utilization.

Vectorized execution. To improve the communication efficiency, current MPC applications often batch independent secure operations over search-shared vectors rather than processing individual scalars, thereby amortizing round complexity. For instance, COMBINE [32] automatically combines all the independent secure operations on secret-share scalars into one operation on secret-share vectors, i.e., combining l independent $\{\mathsf{OP}([\![x_i]\!], [\![y_i]\!])\}, i \in \{0, 1, \cdots l-1\}$ into vectorized operation $\mathsf{OP}([\![\vec{x}]\!], [\![\vec{y}]\!])$, $\vec{x} = \{x_0, x_1, \ldots x_{l-1}\}$ and $\vec{y} = \{y_0, y_1, \ldots y_{l-1}\}$. This approach reduces the total number of operations, thereby reducing the number of communication rounds required by each OP from $n \cdot R_{\mathsf{OP}}$ to R_{OP}, because the parties can communicate the vectors of shares in a single round. R_{OP} denotes the required rounds of a single OP. In this way, the communication overhead can be amortized over multiple operations, significantly improving the performance. Actually, all the current MPC frameworks and applications use vectorized execution, including [12], [13], [25], [30], [32].

III. Motivation and Asymmetric Communication Patterns in MPC

Over the past two decades, numerous innovative MPC protocols have been proposed, leading to the realization of what is now referred to as "general-purpose" MPC. However, as previously noted in Section I, several of these fundamental protocols introduce asymmetric communication patterns, resulting in inefficient utilization of communication bandwidth.

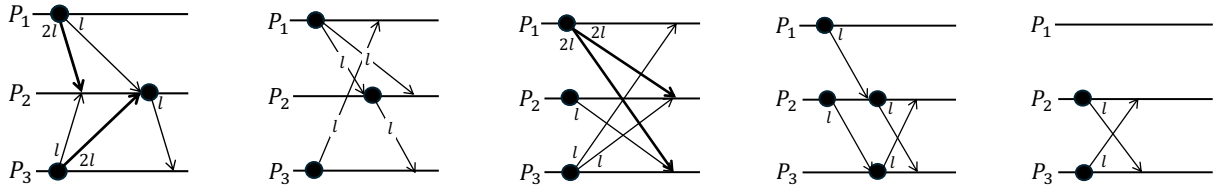
Oblivious Transfer (OT). In MPC, OT is a fundamental primitive that typically presents asymmetric communication patterns. Its functionality is to enable a *receiver* to selectively receive messages from a *sender* who owns a set of messages. The key property of OT is that the sender cannot know which message the receiver wants, and the receiver learns nothing about the messages they did not request [33]. OT forms the foundation of numerous protocols, including *private set intersection (PSI)* [34]–[36], *private information retrieval (PIR)* [37], [38], and many general-purpose building blocks like boolean to arithmetic share conversion (B2A) [13], [25].

Typically, OT involves three steps: (1) the receiver securely sends its target key (i.e., which messages to request) to the sender; (2) the sender encrypts all the messages using the key and sends all the encrypted messages to the receiver; and (3) the receiver decrypts the target messages. This process results in a highly asymmetric communication pattern, as the sender must transmit all messages, while the receiver only sends the target key. Consequently, many protocols that rely on OT inherently exhibit asymmetric communication patterns. Although traditional OT is a two-party protocol, it can be extended to a multi-party setting by appropriately assigning the roles of sender and receiver [13].

Example protocols. We identify six representative asymmetric communicating protocols based on $(2, 3)$-secret-sharing scheme, one of the most typical and efficient schemes:

(1) F-Mul (fixed-point multiplication) [13] performs multiplications of two fixed-point secret numbers, a common representation for real numbers in MPC (representing `float` or `double` as integers with fixed decimal points). F-Mul is fundamental in MPC and is widely used in many secure learning algorithms [4], [27], [39]. The communication pattern of F-Mul is illustrated in Section I.

(2) AB-Mul (arithmetic and boolean shares multiplication) [13] operates on mixed shares (i.e., $[\![x]\!]^A[\![y]\!]^B$). This protocol is typically used in `if-else` or ternary operations in MPC. For instance, one may need to select between the arithmetic shared values $[\![x_1]\!]^A$ and $[\![x_2]\!]^A$ based on a secret comparison result, i.e., $[\![c]\!]^B$, which is a boolean shared value. Specifically, we can select the larger value between $[\![x_1]\!]^A$ and $[\![x_2]\!]^A$ by computing $[\![\text{res}]\!]^A$ = $[\![x_1]\!]^A$ $[\![c]\!]^B$ + $[\![x_2]\!]^A$ $[\![\neg c]\!]^B$, where $[\![c]\!]^B = [\![x_1]\!]^A > [\![x_2]\!]^A$. This protocol is widely used in numerous fields. The communication pattern of AB-Mul is shown in Figure 3a, which leverages the 3PC OT protocol [13]. In the beginning, only P_2 receives data from both P_1 and P_3,



(a) AB-Mul (b) A2B (c) B2A (d) Shuffle (e) T-Mul

Fig. 3: Communication Patterns of Basic Protocols (for F-Mul, see Figure 1)

making its receiving bandwidth the bottleneck. Subsequently, only P_2 transmits l elements to P_3, further contributing to the asymmetric communication workload.

(3) and (4) A2B (arithmetic to boolean conversion) and its reverse, B2A. The A2B protocol converts the arithmetic shares into boolean shares, i.e., $[\![x]\!]^B$ = A2B ($[\![x]\!]^A$), which can be used to accelerate computations that rely heavily on boolean operations like sorting. B2A, on the other hand, converts boolean shares into arithmetic shares, i.e., $[\![x]\!]^A$ = B2A ($[\![x]\!]^B$). B2A is widely used in secure aggregation and summation. These two conversions are fundamental in many MPC applications, especially in secure databases [1], [27], [40]. Their communication patterns are shown in Figures 3b and 3c, respectively. B2A is similar to AB-Mul, since it also relies on 3PC OT protocol [13]. A2B is relatively simple, and its communication pattern is relatively more balanced; the only asymmetric part is the one-way communication from P_1 to P_2 in the beginning of the protocol.

(5) Shuffle [8] is the state-of-the-art 3PC semi-honest shuffle protocol that randomly permutes the input secret-shared array and outputs the shuffled result. Shuffle is widely used in secure graph analysis [8], [9]. Figure 3d shows the asymmetric communication pattern: party1 only sends l elements at the beginning and receives none, resulting in a workload that is only 33.3% and 50% of those for P_2 and P_3, respectively.

(6) T-Mul (Trio multiplication) [16] is a highly unbalanced protocol. As shown in Figure 3e, only P_2 and P_3 need to exchange l elements, while P_1 does not send or receive any data. This results in a communication workload of 0% for P_1.

Similar to F-Mul discussed in Section I, all the above protocols can benefit significantly through RoundRole. Other protocols also include Y2A, B2A in 2PC MOTION and 4PC Quad Multiplication in [16].

IV. Overview

A. Notations and Assumptions

We first introduce the notations and assumptions used in this paper. We denote the MPC protocol with n parties as Π, the size of the input vector as l. There are n independent computation nodes carrying out the computation, each equipped with a single full-duplex network interface card (NIC). We denote the ith computation node as N_i, whose sending and receiving bandwidth is B_i^{send} and B_i^{recv}, respectively. Throughout this paper, all "party" (P_i) denotes the logical role defined by the protocol, and all "node" (N_i) refers to the physical computation node.

For each instance of executing the protocol Π with an input vector of size l, we define it as a *task* T. Each T has an associated one-to-one mapping π from $[n]$ to $[n]$, assigning the logical roles to the physical nodes. E.g., $\pi = (2, 1, 3)$ means that the logical parties P_2, P_1, and P_3 are assigned to the physical nodes N_1, N_2, and N_3. We denote the task with mapping π as $T_\pi^\Pi(l)$, and the wall-clock execution time as $t_\pi^\Pi(l)$. During the execution of task $T_\pi^\Pi(l)$, we denote the data volume transmitted from party P_i to P_j as $c_{ij}^\Pi(l)$, $i, j \in [n]$, $i \neq j$, which is determined by the protocol. $[n]$ denotes the list $[n] = \{1, 2, \dots n\}$.

Following the common assumption of MPC literature [41], we assume that the communication constitutes the primary performance bottleneck rather than computation. This assumption is common because the joint parties are usually distributed across different geographical locations, where the communication bandwidth is significantly lower than the computation bandwidth. Additionally, we focus on sufficiently large inputs l, where the transmission time dominates the communication overhead rather than the latency. For simplicity, we omit the superscript Π of notations like $T_\pi(l), t_\pi(l)$, and $c_{i,j}(l)$ in the following when there is no ambiguity. When the mapping π is not specified, we assume the mapping is the identity mapping, i.e., $\pi = (1, 2, \dots n)$.

B. Motivating Examples

We first clarify the goal of RoundRole and demonstrate its effectiveness using a motivating example.

Consider the 3-party F-Mul protocol Π, whose asymmetric communication pattern is illustrated in Figure 1. If all nodes have identical bandwidth, their bandwidth is clearly underutilized. Specifically, the receiving bandwidth of the third node (N_3) remains idle, and the sending bandwidth of the first two nodes (N_1 and N_2) is approximately half utilized because the sending rate is constrained by the receiving bandwidth of the first two nodes. Figure 2 in Section I shows the bandwidth utilization of the three nodes.

RoundRole is designed to optimize the execution time through improving the bandwidth utilization. Specifically, we divide the input vector of size l into 3 equal parts, i.e., $l_1 = l_2 = l_3 = \frac{l}{3}$, and split the original task executing Π into three parallel tasks $T_{1,(1,2,3)}(\frac{l}{3}), T_{2,(3,1,2)}(\frac{l}{3})$ and $T_{3,(2,3,1)}(\frac{l}{3})$. The communication patterns for the three tasks are shown in Figure 4. Clearly, this optimization evenly balances communication across the three physical nodes, thereby enabling full

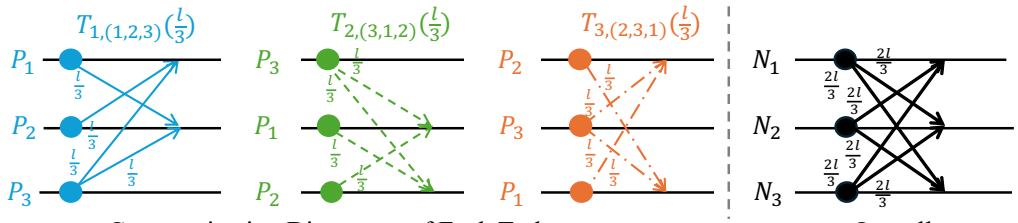


Fig. 4: Communication Pattern of F-Mul with RoundRole.

bandwidth utilization. The goal of RoundRole is to generalize the above example.

C. Problem Definition

Formally, RoundRole aims to find the execution strategy that minimizes the wall-clock execution time $t^{\Pi}(l)$ for *any* vectorized secret-sharing protocol Π. To achieve this, we divide the overall task $T(l)$ into m parallel tasks $T_{1,\pi_1}, T_{2,\pi_2}, \ldots, T_{m,\pi_m}$, where each task computes l_i elements with its logical role mapped to the physical nodes via the m mappings $\pi_1, \pi_2, \ldots \pi_m$. For simplicity, we directly denote the task as $T_{\pi_i}(l_i)$ for the ith task when there is no ambiguity.

Formally, the optimization problem is defined as follows:

$$\begin{aligned} \text{minimize} \quad & t^{\Pi}(l) = \max\{t^{\Pi}_{\pi_1}(l_1), t^{\Pi}_{\pi_2}(l_2), \ldots, t^{\Pi}_{\pi_m}(l_m)\} \\ & \qquad + t^{\Pi_{\text{aggr}}}(l_1, l_2, \ldots, l_m) \qquad (1) \\ \text{s.t.} \quad & \sum_{i=1}^{m} l_i = l, \end{aligned}$$

where $\max\{t^{\Pi}_{\pi_1}(l_1), t^{\Pi}_{\pi_2}(l_2), \ldots, t^{\Pi}_{\pi_m}(l_m)\}$ is the maximum execution time among all parallel tasks, each of which processes l_i elements and shares the bandwidth. The term $t^{\Pi_{\text{aggr}}}(l_1, l_2, \ldots, l_m)$ is the time to aggregate the results of each task using the aggregation protocol Π_{aggr}. For simplicity, we use t^{paral} and t^{aggr} to represent the two terms. Note that each task $T_{\pi_i}(l_i)$ executes the same protocol as Π on a smaller input of size $l_i \leq l$, with the logical role rounded on the physical nodes according to the mapping π_i.

The objective of RoundRole is to solve this optimization problem by determining: (1) the number of parallel tasks m; (2) the input size l_i for each task $T_{\pi_i}(l_i)$; and (3) the mapping π_i for each task $T_{\pi_i}(l_i)$, such that the overall execution time $t^{\Pi}(l)$ is minimized.

Without loss of generality, we assume the communication complexity of the protocol Π is linear to the input size l, i.e., for each pair of logical parties, P_i and P_j, the communication volume $c_{ij}(l)$ can be approximated by $c_{ij}(l) = \alpha_{ij} \cdot l$, where α_{ij} is a constant. This is a practical assumption that holds for a wide range of MPC protocols. Specifically, all the protocols in Section III satisfy this requirement. For protocols with higher communication complexity, e.g., *sort*, we can split the protocol into multiple protocol units (*shuffle*, *comparison*, and *multiplications*) and apply RoundRole to each unit separately.

D. RoundRole Overview

RoundRole solves the above problem by generalizing the motivating example in Section IV-B to achieve optimal performance on *any* network topology for a given protocol Π.

For n-party protocol, there are $n!$ possible mappings (i.e., full permutations) of the logical roles to the physical nodes. In most MPC settings, n is moderate (often $n = 2, 3$, or 4) [8], [12]–[14], [20], [21], [42]. For large n, we can adopt the *column generation* technique [43] to efficiently find a near-optimal solution without processing all $n!$ mappings. The first step of RoundRole is to determine the workload allocation ratios across all possible mappings to *minimize the maximum transmission time* on the underlying physical nodes, which determines how to split the input size l for each mapping. This optimal allocation ratio is formulated as a linear programming problem, which can be solved efficiently. As a result, we obtain $n!$ ratios $\{r_1^*, r_2^*, \cdots, r_{n!}^*\}, \sum_{i=1}^{n!} r_i^* = 1$ for the $n!$ mappings $\{\pi_1, \pi_2, \cdots, \pi_{n!}\}$, where r_i^* is the optimal ratio of the input size assigned to tasks with the ith mapping π_i (Section V-A).

Then, we determine the optimal task number m to *minimize the overall execution time* $t^{\Pi}(l)$. Each task is responsible for computing $\frac{l}{m}$ elements, and we assign the mapping to each task according to the optimal allocation ratios obtained in the first step. The selection of m requires careful consideration. On the one hand, m must be sufficiently large to fully exploit the physical network bandwidth, i.e., ensuring that enough computation resources are used to generate to-be-transmitted data quickly. On the other hand, m cannot be so large that the aggregation time t^{aggr} becomes the bottleneck (Section V-B).

Optimizing advanced applications. For advanced MPC applications/algorithms that are composed of multiple protocols, we can optimize their performance by optimizing each protocol unit separately. This separation offers the following advantages: (1) The communication pattern of each protocol unit varies, and it requires different scheduling strategies to balance the communication workload. (2) Because the inputs and outputs of each protocol unit are all secret shares, and each share is randomly distributed among the logical parties, switching the logical roles of each physical node can be done by simply changing the roles with negligible overhead. (3) Many MPC applications employ common protocol units, e.g., *multiplication*, *arith-to-boolean* conversion, allowing us to predefine and reuse the optimization strategies for these units across different applications.

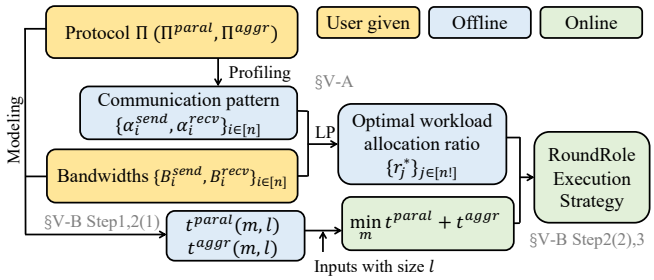


Fig. 5: RoundRole Workflow

Security guarantee. RoundRole preserves the same security guarantee as the original protocol Π under the *real-ideal paradigm* [44], including all the adversary assumptions. Intuitively, RoundRole only reassigns the logical roles on the physical nodes across independent tasks without modifying *any* communication routines of Π. Because the reassignment is data-independent, the view of the adversary corrupting the physical nodes has the same distribution as in an execution with fixed role assignments. Formally, if there exists an adversary corrupting at most t physical nodes can distinguish the RoundRole-optimized execution from its ideal counterpart under the same adversary model as Π, we can construct an adversary that distinguishes the original protocol Π from its ideal execution, contradicting the security of Π (see Appendix A).

MPC-specific design. Conventional load-balancing methods assume independent and flexible tasks without security constraints, therefore they cannot be directly applied in MPC. RoundRole exploits three MPC characteristics to improve its efficiency automatically: (1) *protocol-fixed asymmetry* in communication that is non-trivial to symmetrize without protocol-specific redesign; (2) *flexible role-node mappings* that permit re-assignment of logical parties to physical nodes; and (3) *communication-dominated cost*, where throughput improvements directly translate to speedups. RoundRole exploits the MPC-specific flexibility of mappings to balance bandwidth *across* asymmetric protocols without changing the protocols.

Application scenarios. RoundRole is applicable in any setting with asymmetric communication. However, it is particularly effective in stable communication settings, such as the semi-honest protocols listed in Section IV-B, where execution paths are fixed, and per-round traffic can be profiled and balanced to "unlock" the underutilized bandwidth. In contrast, malicious-security protocols often have numerous consistency checks, making the latency the dominant bottleneck rather than bandwidth utilization [42]. Consequently, RoundRole offers limited benefits in such scenarios.

V. RoundRole Optimization

Figure 5 summarizes the workflow of RoundRole.

A. Optimal Workload Allocation Ratio

The goal in this stage is to minimize the *maximum transmission time* executing Π on the underlying network. We achieve this in two steps: (1) we profile Π to obtain its communication pattern, i.e., the amount to be sent/received per unit of input for each logical party. Therefore, for any logical-to-physical mapping π_i, the per-node traffic is predictable; (2) we formulate a linear programming problem over $n!$ workload allocation ratios, $\{r_1, r_2, \ldots, r_{n!}\}$ (with $r_i \geq 0$ and $\sum_i r_i = 1$), where r_i is the fraction of the total inputs assigned to mapping π_i. The objective is to minimize the maximum transmission time, and the solution is the *optimal workload allocation ratio* $\{r_i^*\}$.

Characterizing the communication pattern of protocol Π. For each protocol Π, we model its communication pattern as $n(n-1)$ communication pairs, i.e., $c_{ij}(l), \forall i, j \in [n]$ and $i \neq j$. Since the communication volume is linear to the input size l, we can model the communication volume as $c_{ij}(l) = \alpha_{ij} \cdot l$, where α_{ij} is a constant factor characterizing the unit data communication cost from P_i to P_j, which is determined by the protocol Π. We then use two sets of *communication coefficients*, $\{\alpha_i^{\text{send}}\}_{i\in[n]}$ and $\{\alpha_i^{\text{recv}}\}_{i\in[n]}$, to characterize the total communication volume of the ith logical party in the sending and receiving directions, respectively. Specifically, $\alpha_i^{\text{send}} = \sum_{j\in[n]}^{j\neq i} \alpha_{ij}$ and $\alpha_i^{\text{recv}} = \sum_{j\in[n]}^{j\neq i} \alpha_{ji}$.

RoundRole uses a profiling method to estimate α_i^{send} and α_i^{recv} for each logical party P_i in protocol Π. In practice, RoundRole automatically runs the protocol Π with a set of evenly spaced input sizes and records the communication volume for each logical party in both sending and receiving directions. Then, RoundRole applies the *least squares* method to estimate the communication coefficients.

Optimal workload allocation problem. Given the communication coefficients $\{\alpha_i^{\text{send}}\}_{i\in[n]}$ and $\{\alpha_i^{\text{down}}\}_{i\in[n]}$, we can estimate the communication volume of each logical party in the sending and receiving directions given input size. Since there are $n!$ possible mappings of the logical roles to the physical nodes, we define the *optimal workload allocation problem* as finding the optimal input size ratios $\{r_1^*, r_2^*, \ldots, r_{n!}^*\}$ for all $n!$ possible mappings $\{\pi_1, \pi_2, \ldots, \pi_{n!}\}$, with the objective of minimizing the maximum transmission time across all the physical nodes. Note that the transmission time is normalized to the input size l. Formally, we define the problem as follows:

$$
\begin{aligned}
\text{minimize} \quad & \bar{t}_{\text{trans}}^{\text{paral}} && (2)\\
\text{s.t.} \quad & \sum_{j=1}^{n!} \alpha_{\pi_j(i)}^{\text{send}} \cdot r_j \leq B_i^{\text{send}} \cdot \bar{t}_{\text{trans}}^{\text{paral}}, \quad i = 1, 2, \ldots, n,\\
& \sum_{j=1}^{n!} \alpha_{\pi_j(i)}^{\text{recv}} \cdot r_j \leq B_i^{\text{recv}} \cdot \bar{t}_{\text{trans}}^{\text{paral}}, \quad i = 1, 2, \ldots, n,\\
& \sum_{j=1}^{n!} r_j = 1, \quad r_j \geq 0, \quad j = 1, 2, \ldots, n!,
\end{aligned}
$$

where another decision variable $\bar{t}_{\text{trans}}^{\text{paral}}$ represents the normalized transmission time across all the physical nodes N_i, $i \in [n]$:

$$
\bar{t}_{\text{trans}}^{\text{paral}} = \max\left\{ \max_{i\in[n]} \frac{C_i^{\text{send}}}{B_i^{\text{send}}}, \max_{i\in[n]} \frac{C_i^{\text{recv}}}{B_i^{\text{recv}}} \right\}, \qquad (3)
$$

where C_i^{send} and C_i^{recv} are the normalized total transmission volume of the physical node N_i (each node carries $n!$ tasks, each with one possible mapping) in the sending and receiving directions, respectively. Specifically,

$$C_i^{\text{send}} = \sum_{k=1}^{n!} \sum_{j=1}^{n} c_{\pi_k(i)\pi_k(j)}(r_k) = \sum_{k=1}^{n!} \alpha_{\pi_k(i)}^{\text{send}} \cdot r_k, \tag{4}$$

$$C_i^{\text{recv}} = \sum_{k=1}^{n!} \sum_{j=1}^{n} c_{\pi_k(j)\pi_k(i)}(r_k) = \sum_{k=1}^{n!} \alpha_{\pi_k(i)}^{\text{recv}} \cdot r_k. \tag{5}$$

We can efficiently obtain the optimal ratios $\{r_1^*, r_2^*, \ldots, r_{n!}^*\}$ and $\bar{t}_{\text{trans}}^{\text{paral}}$ in Equation 2 using the *simplex* algorithm.

B. Optimal Execution Strategy

The goal of the second stage is to find the optimal m parallel tasks minimizing the overall execution time. RoundRole achieves this through the following steps:

Step1: Deciding the $\bar{m}$ parallel tasks that can fully utilize the physical bandwidth. For a given protocol Π, we assume that the bandwidth utilization is linear with the task numbers m when the bandwidth is not saturated, i.e., less than 100%, which is a common assumption in practice with TCP protocols [45]. Since we mainly focus on cases where the input size is large enough to make the transmission time the bottleneck, we profile the maximum sending and receiving bandwidth utilization u_i^{send} and u_i^{recv} of a single task for a large input size on each physical node N_i, $i \in [n]$. Specifically, for each mapping $\pi_j, j \in [n!]$, we concurrently launch a task using an input size of $r_j^* \cdot L$, where L is chosen to be sufficiently large to ensure a stable transmission rate. We then measure the stable sending and receiving bandwidth utilization for each physical node N_i as u_{ij}^{send} and u_{ij}^{recv} with $n!$ concurrent tasks. Then, we can obtain the average per-task sending and receiving bandwidth utilization $u_i^{\text{send}} = \frac{1}{n!}\sum_{j=1}^{n!} u_{ij}^{\text{send}}$ and $u_i^{\text{recv}} = \frac{1}{n!}\sum_{j=1}^{n!} u_{ij}^{\text{recv}}$.

We can then simply obtain the task number $\bar{m}$ that can fully utilize the physical bandwidth of the underlying network as:

$$\bar{m} = \max\left\{\max_{i\in[n]} \frac{1}{u_i^{\text{send}}}, \max_{i\in[n]} \frac{1}{u_i^{\text{recv}}}\right\}. \tag{6}$$

Step2: Reducing $\bar{m}$ to optimal m to avoid excessive aggregation overhead. While $\bar{m}$ tasks make full use of the bandwidth, too many tasks may lead to excessive aggregation costs. Therefore, it is crucial to reduce the number of tasks to balance transmission time and aggregation costs. This step includes two sub-steps: (1) an offline, input-size-independent modeling to approximate the parallel execution (t^{paral}) and aggregation (t^{aggr}) times; (2) an online computation to determine the optimal m minimizing $t^{\text{paral}} + t^{\text{aggr}}$ given the real inputs.

(1) *Offline modeling.* Since the communication overhead is the main bottleneck, we can simplify the parallel execution time t^{paral} as the maximum sending and receiving time of all the physical nodes. The normalized transmission time, $\bar{t}_{\text{trans}}^{\text{paral}}$ is obtained in Equation 3 of the first step (Section V-A), which is the *ideal* transmission time of the constrained link without considering the bandwidth utilization. Assume u^* is the corresponding bandwidth utilization of the constrained link, i.e., the *argmax* result in Equation 3. We can then approximate the *practical* parallel execution time t^{paral} as

$$t^{\text{paral}} \approx \bar{t}_{\text{trans}}^{\text{paral}} \cdot l \cdot \min(m \cdot u^*, 1), \tag{7}$$

where $\min(m \cdot u^*, 1)$ represents the bandwidth utilization using m tasks.

The aggregation time t_{aggr} is the time to aggregate the results from all tasks. Without loss of generality, we assume the aggregation procedure is performed in a binary tree structure, where the results from m tasks are aggregated in pairs. The user can also define other aggregation methods whenever applicable. The aggregation time can be approximated as the sum of the time taken for all the $\lceil \log(m) \rceil$ rounds of the aggregations. Ignoring the variance among parallel tasks for simplicity, the aggregation time can be approximated as:

$$\begin{aligned} t^{\text{aggr}} &\approx \sum_{i=0}^{\lceil \log_2 m \rceil} t^{\Pi_{\text{aggr}}}\left(2^i \cdot \frac{l}{m} \cdot \left\lceil \frac{m}{2^i} \right\rceil\right) \\ &\approx \lceil \log_2 m \rceil t^{\Pi_{\text{aggr}}}(l), \end{aligned} \tag{8}$$

We apply a profiling-based method to estimate the aggregation cost $t^{\Pi_{\text{aggr}}}(l)$ for different input sizes l. Specifically, we sequentially launch multiple pairs of tasks executing the aggregation protocol Π with a set of linearly spaced input sizes, measure the execution time, and then apply the least squares method to fit the execution time with a quadratic polynomial function. For simplicity, all the aggregation tasks are executed using the mapping with the highest optimal allocation ratio, i.e., $\pi_{i^*}, i^* = \text{argmax}_{i\in[n!]}(r_i^*)$.

(2) *Online computation of optimal m.* Given input size l, we can efficiently determine the optimal task number m minimizing $t^{\text{paral}} + t^{\text{aggr}}$ by searching $m \in [1, \min(\bar{m}, m_{\text{max}})]$, where m_{max} is the maximum parallelism supported by all the physical nodes.

Step3: Obtaining the optimal execution strategy for m parallel tasks. Finally, we can obtain the optimal execution strategy, minimizing the overall execution time $t^{\Pi}(l)$. We split the input size l into m equal parts, i.e., $l_i = \frac{l}{m}, i \in [m]$, and assign the mappings to m tasks according to the optimal workload allocation ratios from Section V-A. Specifically, we allocate $\pi_j, j \in [n!]$ to $r_j^* * m$ tasks, rounding to the nearest integer when necessary.

Optimization cost and dynamic adaptation. The automatic optimization includes two phases: (1) offline profiling to model the protocol communications and the network; (2) online phase to determine the optimal task number and execution strategy given real inputs. Only the second sub-step in Step2 and Step3 require real-time computations, while all the other profiling can be completed offline. Empirically, RoundRole only requires milliseconds to configure the optimal tasks with corresponding mappings, as shown in Section VI-C, introducing negligible overhead during the online phase. Because the allocation only depends on the network bandwidth and

TABLE I: Network Settings

Network Configurations		N_1	N_2	N_3
Homogeneous	Homo1	100 Mbps	100 Mbps	100 Mbps
	Homo2	1 Gbps	1 Gbps	1 Gbps
	Homo3	10 Gbps	10 Gbps	10 Gbps
Heterogeneous	Hetero1	1 Gbps	2 Gbps	2 Gbps
	Hetero2	1 Gbps	10 Gbps	10 Gbps
	Hetero3	1 Gbps	4 Gbps	5 Gbps

the protocol communication pattern, RoundRole can adapt to dynamic network conditions by tracing bandwidth changes and rerunning the allocation accordingly when significant network changes are detected without introducing extra profiling overhead.

VI. Evaluation

In this section, we evaluate the performance of RoundRole on the protocols discussed in Section III and three real-world algorithms. We want to answer the following questions:

1) How much execution time can RoundRole improve in these protocols and algorithms?
2) How much bandwidth utilization can RoundRole improve?
3) Does RoundRole automatically determine execution strategies that provide efficient performance across various protocols, algorithms, and network settings with negligible overhead?

A. Evaluation Setup

Implementation of RoundRole. We implement RoundRole as a separate module that invokes the provided protocols for profiling and execution. We integrate RoundRole on top of ABY3 [13], a widely used open-source MPC framework. Notably, using RoundRole is straightforward in practice, because it only invokes the existing function interfaces in parallel, with no modifications needed to the original protocol.

Testbeds. We employ six network settings with bandwidths varying from 100 Mbps (WAN) to 10 Gbps (LAN). In order to evaluate the performance of RoundRole across different network settings, we consider both homogeneous (every node has the same bandwidth) and heterogeneous settings (nodes have different bandwidths). Table I provides the detailed configurations. Each node is equipped with 96 Intel(R) Xeon(R) Gold 6330 CPU and 320 GB RAM. The networks are set up using `mininet` [46]. By default, the network latency is set to 0.2ms unless otherwise specified.

Micro-benchmarks. We use all six basic protocols discussed in Section III as micro-benchmarks, including F-Mul, AB-Mul, A2B, B2A, Shuffle, and T-Mul. Since Shuffle and T-Mul are not provided in the ABY3 codebase, we implement them directly following the original papers [8] and [16], respectively. The other protocols are directly sourced from the ABY3 codebase [13] without any modifications.

Real-world algorithm benchmarks. To further validate the practicality of RoundRole, we include three real-world algorithms. We choose these algorithms not only because they are widely utilized but also because they use a variety of MPC operations that require diverse communication patterns. The three algorithms include:

(1) The ORAM algorithm (specifically, Square-root ORAM [19]) is a representative *distributed ORAM* that can be implemented on any secret-sharing scheme. It incorporates multiple Shuffle protocols that contribute to the asymmetric communication patterns. We implement the ORAM algorithm following [19] and replace its original *Waksman* shuffle with the more efficient Shuffle protocol [8].

(2) The sorting algorithm is implemented following Hamada et al. [17], which implements a secure quicksort that primarily utilizes Shuffle, AB-Mul and greater-than comparisons.

(3) The Logistic Regression (LR) algorithm comes in the ABY3 codebase [13]. Each iteration of LR incorporates two F-Mul operations and one B2A operation.

Evaluation methods and metrics. We measure the performance of RoundRole using two metrics: (1) the *makespan* (i.e., end-to-end completion time of each protocol, measured by wall-clock), and (2) the bandwidth utilization of each protocol in order to evaluate the source of acceleration. To be succinct, we report the utilization of the bottleneck node (i.e., the one with the lowest bandwidth) in the heterogeneous settings. While in the homogeneous settings, we report the average utilization across all the nodes. For each measurement, we use an input vector size of 1.4 billion 64-bit secret integers unless otherwise specified. We use such a large input size to ensure that the network operations are in a steady state, especially for the 10 Gbps bandwidth (i.e., Homo3). Each protocol/algorithm is run 5 times, and the average results are reported.

Baselines. We use the same implementations of the basic protocols and the real-world algorithms without applying RoundRole as baselines. Note that we allow these protocols to use the same degree of data parallelism for a fair comparison. The only difference between baseline and RoundRole is the automatically defined mappings of logical roles to physical nodes for each parallel task. For T-Mul, we also implement the optimized high-throughput execution strategy following [16], and use it as the T-Mul* baseline.

B. Micro-benchmark Results

In this section, we present the micro-benchmark results. For most protocols, the aggregation cost is negligible because it only concatenates outputs from each task, except for the Shuffle protocol. To focus on optimizing the parallel execution of asymmetric-communicating protocols, we omit the aggregation cost of Shuffle in the main micro-benchmark results. A complete end-to-end evaluation of Shuffle, including its aggregation phase, is provided in Section VI-C.

Homogeneous network settings. Figure 6 shows an overview of the makespan comparison of the six basic protocols across six network settings, with and without using RoundRole. First, from the top row (homogeneous settings), we can see that:

(1) RoundRole significantly improves the makespan of all the basic protocols across all the homogeneous network settings. Specifically, RoundRole achieves 1.1× to 2.8× speedups

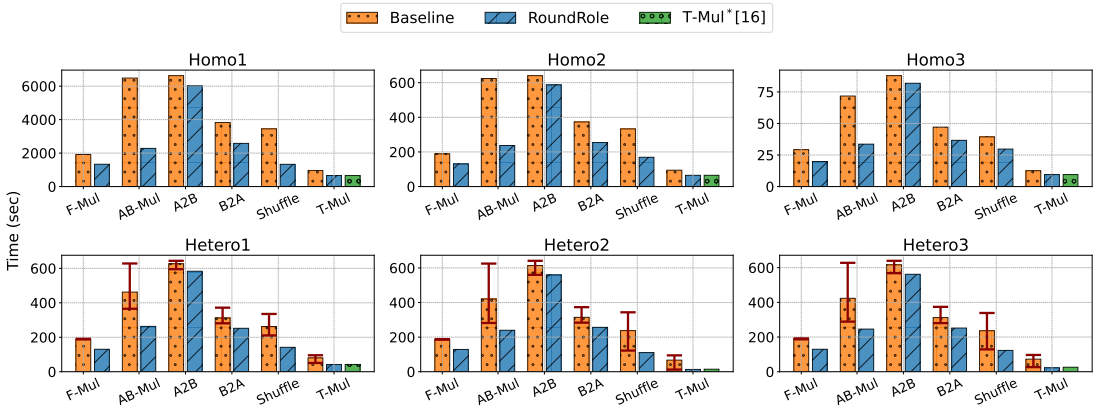


Fig. 6: Makespan Comparison of the Micro-benchmarks (Lower is better). (For the baseline methods on the heterogeneous settings, we evaluate their performance on all the different $n!$ logical-to-physical mappings and present the best, average, and worst records.)

in makespan, with an average speedup of 1.6× across all protocols. We provide a more detailed analysis below.

(2) With all homogeneous network settings (top row), RoundRole works best for AB-Mul, achieving up to 2.8×, 2.6×, and 2.1× speedups on Homo1, Homo2, and Homo3, respectively. The reason is that AB-Mul is highly imbalanced in communication, where the receiving bandwidth for P_1 and P_3, as well as the sending bandwidth of P_2 are mostly underutilized during a large portion of the protocol execution. Also, since both P_1 and P_3 are sending data to P_2, the receiving bandwidth of P_2 becomes the bottleneck and therefore constrains the sending rate of P_1 and P_3, making these two parties underutilized.

To empirically show this, we plot the bandwidth utilization at different time points during the execution of AB-Mul in Figure 7b on Homo1 [2]. We clearly observe that without RoundRole, at least one sending/receiving bandwidth is idle during the execution of AB-Mul. With RoundRole, the *full duplex* bandwidths are almost 95+% utilized, resulting in a significantly shorter makespan.

(3) In contrast, A2B has a much more balanced communication pattern and thus RoundRole only improves its performance by a small margin with a 1.1× speedup. From Figure 7c, we can see that the original bandwidth utilization is already high, except for a short phase where P_1 sends more elements to P_2 (see Figure 7c in Section III for details). Even in this case, RoundRole achieves improvement.

(4) While RoundRole achieves similar speedups for almost all protocols across different bandwidth settings, the speedup of Shuffle gets lower with higher bandwidth. This is because Shuffle uses many separate rounds of small-sized communication that introduce a longer warm-up period on higher-bandwidth networks. As a result of this fixed overhead, RoundRole can only improve the performance by 1.3× on 10 Gbps (Homo3), while the improvement is as much as 2.6× on 100 Mbps (Homo1).

[2]We only include Homo1 here due to space limits. For other settings, see Appendix B.

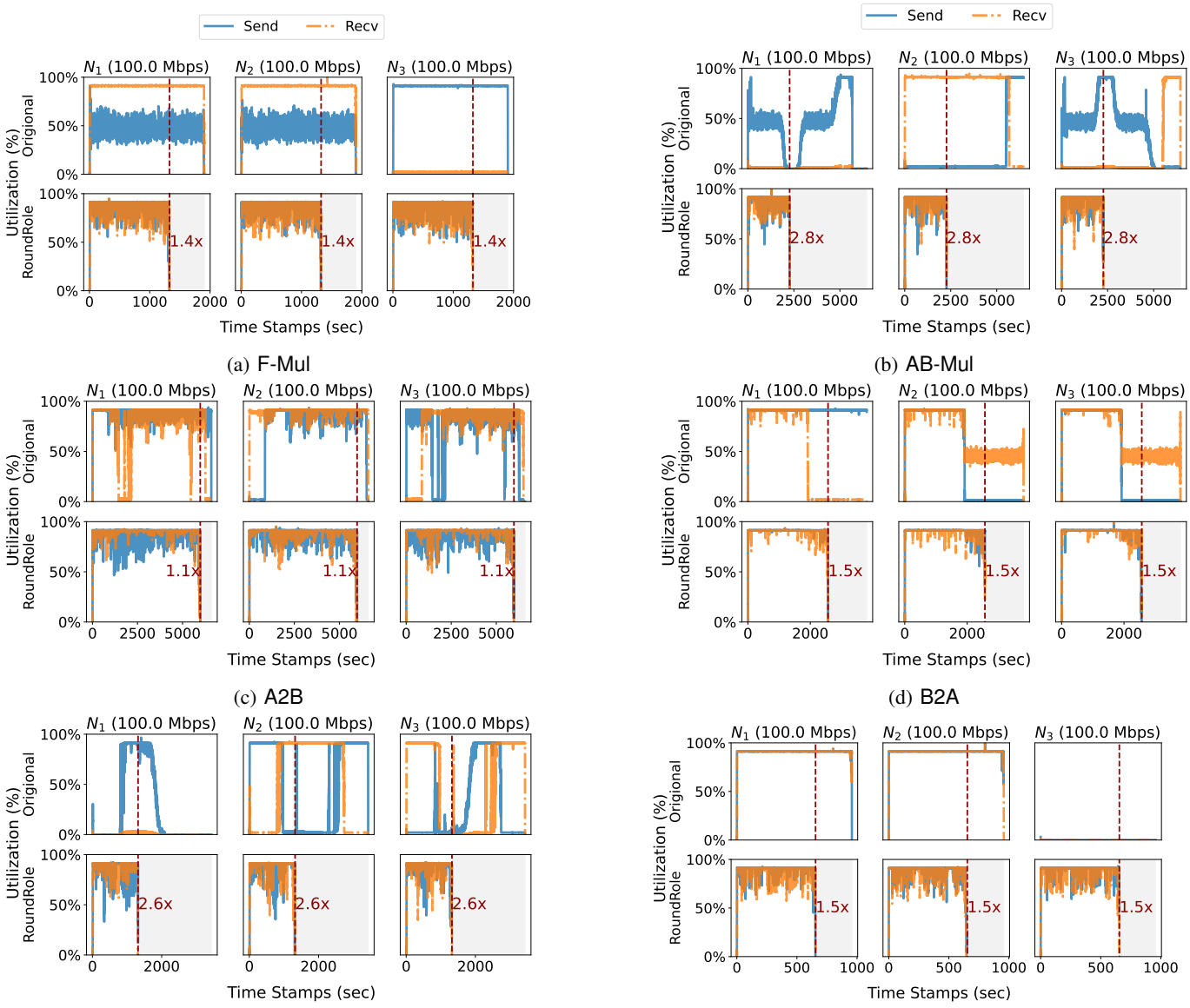
Heterogeneous network settings. The bottom line of Figure 6 shows the makespan comparison on heterogeneous settings. Since the baseline methods present varied performance with different logical-to-physical mappings, we measure the best, average, and worst wall-clock execution time of the baselines across all the $n!$ logical-to-physical mappings. We can see:

(1) Baseline methods exhibit significant performance variation, especially for protocols with highly asymmetric communication patterns like AB-Mul and T-Mul. In Hetero2, the bandwidth varies from 1 Gbps to 10 Gbps, and the makespan of AB-Mul and T-Mul varies by 2.2× and 7.1×, respectively. This variation arises because the communication workload differs across logical parties, and the performance is highly sensitive to which party is mapped to which node. Consequently, these asymmetric protocols require careful manual tuning to avoid worst-case performance.

(2) In contrast, RoundRole consistently identifies the optimal execution strategy and achieves an average speedup of 1.9× across all 36 baselines (6 protocols and $3! = 6$ logical-to-physical mappings). Also, by task splitting and fine-grained workload allocation, RoundRole consistently performs better than the best result for each protocol. On the most unbalanced Hetero2 setting, RoundRole achieves the best speedup of 7.1× for T-Mul.

Comparison with the optimized T-Mul* baseline. Among all the asymmetric basic protocols, T-Mul is designed to achieve high throughput by Harth-Kitzerow et al. [16]. Specifically, as Section IV-B describes, T-Mul only involves two communications between a single pair of the parties, thereby, it can be optimized manually to achieve high throughput by simply letting the communication happen in every pair of the parties. We also treat it as the T-Mul* baseline and compare its performance with RoundRole in Figure 6 and Figure 8. We can see that for this special protocol, RoundRole achieves the same performance as the optimized result automatically.

Improvements on bandwidth utilization. As Figure 8 indicates, RoundRole improves the overall performance by increasing the bandwidth utilization of the basic protocols.



(a) F-Mul (b) AB-Mul (c) A2B (d) B2A (e) Shuffle (f) T-Mul

Fig. 7: Communication Traces of the Micro-benchmark Protocols without / with RoundRole.

TABLE II: Workload Allocation Ratios on Hetero1.

Π	All Mappings					
	1,2,3	1,3,2	2,1,3	2,3,1	3,1,2	3,2,1
F-Mul				67%	33%	
AB-Mul	67%			33%		
A2B						100%
B2A		34%			66%	
Shuffle	87%	9%			4%	
T-Mul	58%		21%		21%	

TABLE III: Optimization Overhead on Homo1.

Overhead	F-Mul	AB-Mul	A2B	B2A	Shuffle	T-Mul
Offline (sec)	18.5	44.7	52.8	31.9	5.8	2.6
Online (ms)	55.6	56.3	66.0	53.8	34.9	55.5

Figure 8 shows the bandwidth utilization of all the basic protocols with or without RoundRole on six network settings. We have the following conclusions:

(1) In the first two homogeneous settings, the bandwidth utilization is almost 100% except Shuffle on Homo2, for the same reason above (i.e., the multiple rounds of small-sized communication increase the warm-up period, especially for networks with higher bandwidth. The empirical bandwidth utilization figures are provided in Appendix B).

(2) In the most constrained bandwidth setting, i.e., Homo1, which aligns with the most commonly adopted WAN setting in MPC literature, RoundRole achieves an average of 100.0% utilization across all the basic protocols, which is 34.3% higher than the baseline methods. In Homo3, the average utilization

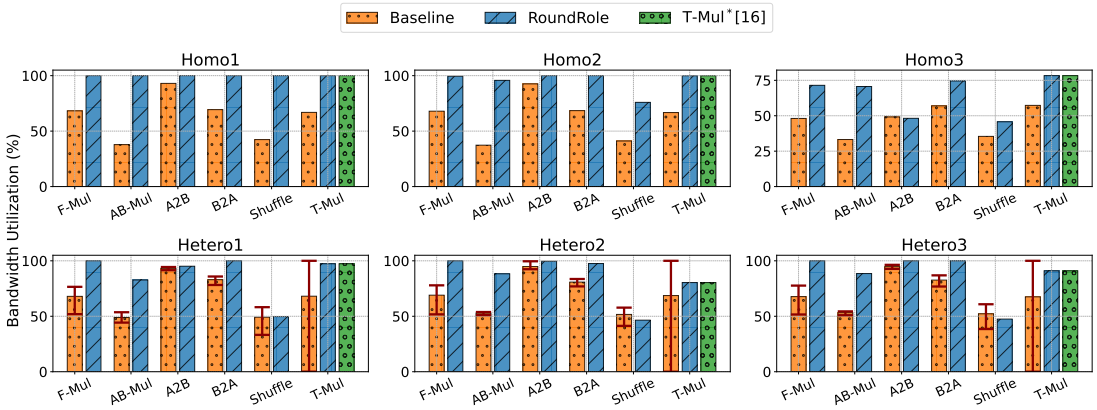
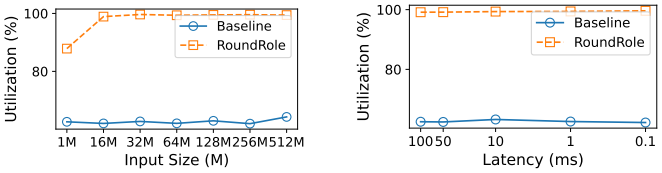


Fig. 8: Bandwidth Utilization of the Micro-benchmarks (Higher is better) (The bandwidth utilization of the heterogeneous settings is measured on the bottleneck node. Similar to Figure 6, the records for baseline methods on heterogeneous show the best, average, and worst results on all the different $n!$ logical-to-physical mappings.).



(a) with Varied Input Size. (b) with Varied Latency.

Fig. 9: Bandwidth Utilization Analysis.

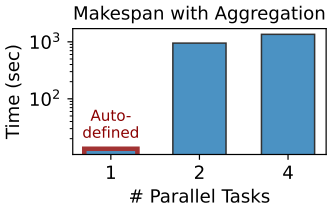


Fig. 10: Auto-defined Parallelism of Shuffle with Aggregation Cost on Homo1.

is only 62.1% because the high-speed 10 Gbps network transfers a significant amount of data during the warm-up phase. Nevertheless, RoundRole still achieves nearly 100% utilization during the steady state. Detailed communication traces are provided in Appendix B. As a result, the overall bandwidth utilization appears lower, despite the network's capacity to support higher sustained throughput.

(3) For all the protocols except Shuffle and T-Mul, RoundRole achieves almost full bandwidth utilization on the bottleneck node in the heterogeneous settings. Specifically, RoundRole achieves 94.4% on the left four protocols in all three heterogeneous settings, which is 14.8% higher than the baseline methods on average. For Shuffle and T-Mul protocols where the communication workload varies significantly across different logical parties, RoundRole can intelligently allocate the parties with higher workloads to the nodes with higher bandwidth, thereby making the bottleneck node not the bottleneck, which we will discuss next in Section VI-C.

TABLE IV: Makespan Speedups Using RoundRole.

	Homo1	Homo2	Homo3	Hetero1	Hetero2	Hetero3
ORAM	1.5	1.4	1.1	$1.4^{0.2}_{0.3}$	$1.5^{0.3}_{0.5}$	$1.5^{0.3}_{0.4}$
Sort	1.3	1.3	1.2	$1.2^{0.3}_{0.2}$	$1.2^{0.3}_{0.2}$	$1.3^{0.4}_{0.3}$
LR	1.5	1.4	1.3	$1.4^{0.0}_{0.0}$	$1.4^{0.1}_{0.1}$	$1.4^{0.1}_{0.1}$

Performance with varied input sizes and latency. Figure 9 shows the average bandwidth utilization of all the protocols in Figure 16 with or without RoundRole on Homo1 with varied input sizes (1M to 512M) and network latencies (100ms to 0.1ms). Figure 9a and Figure 9b fix the latency and the input size to the defaults in Section VI-A, respectively. We can see that the bandwidth utilization of RoundRole remains at the full capacity level except for processing 1M inputs, demonstrating the practicality of RoundRole. The 1M inputs waste around 12.2% bandwidth because they finish execution before reaching full capacity. In contrast, the baseline utilization is consistently limited to around 62.6%, regardless of latency or data volume. This is because its performance is constrained by the bandwidths of the first two nodes.

C. Automatic Optimization

Mapping logical roles to physical nodes automatically with negligible overhead. It is obvious that RoundRole achieves the above improvements by automatically mapping the logical roles to physical nodes properly. We then provide a detailed analysis of the workload allocation ratios using Hetero1 setting as an example. Table II shows the auto-solved optimal workload on Hetero1 setting. We can see that for Shuffle where the communication workload of P_1 is three times lower than P_2 and P_3, RoundRole allocates 96% of the tasks with P_2 and P_3 on the nodes with larger bandwidths (i.e., N_2 and N_3) to make full use of the network resources. Similarly, for T-Mul that only P_2 and P_3 need to communicate, RoundRole allocates 58% of the tasks with P_2 and P_3 on the nodes with

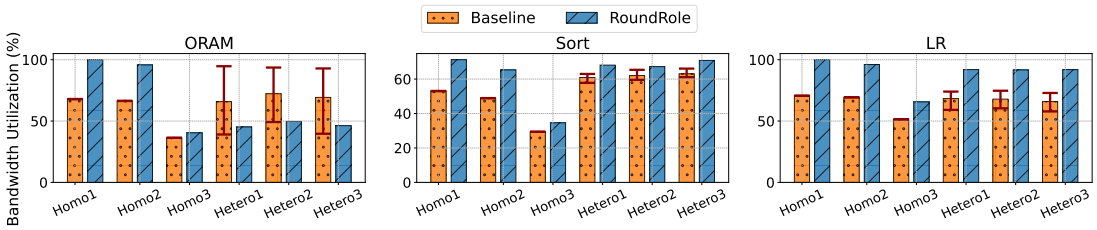


Fig. 11: Bandwidth Utilization of Real-world Algorithms with and without RoundRole. (The bandwidth utilization of the heterogeneous settings are measure on the bottleneck node, i.e., the first node with the lowest bandwidth.)

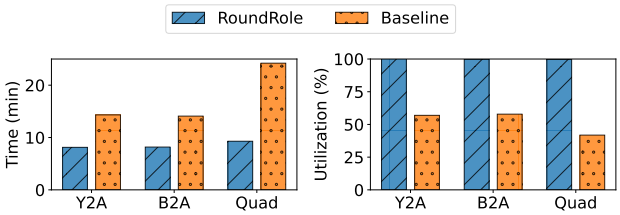


Fig. 12: Performance on other MPC protocols

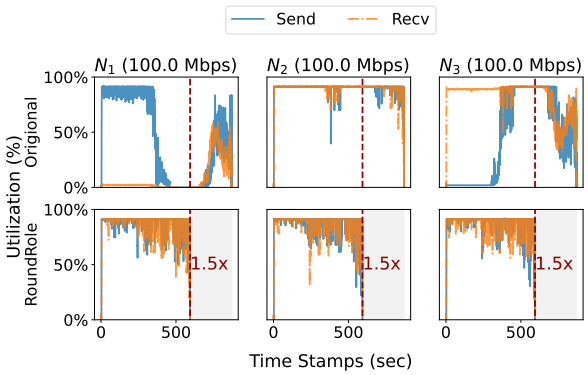


Fig. 13: ORAM Communications without / with RoundRole.

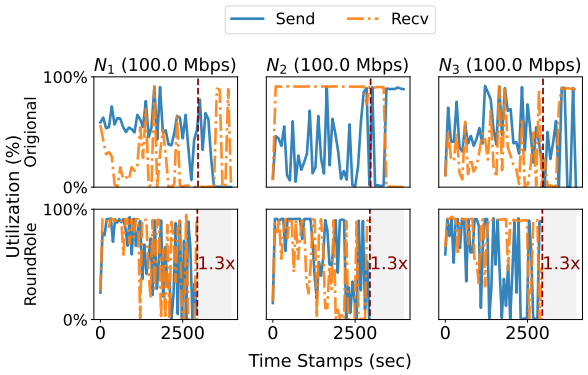


Fig. 14: Sort Communications without / with RoundRole.

2 Gbps bandwidth, i.e., N_2 and N_3, while the other 42% of the tasks are allocated to leverage the sending and receiving bandwidths of N_1.

The practical automatic configuration overhead is negligible, as shown in Table III. Even on the slowest network, i.e., Homo1, the offline profiling cost is less than 1 minute. All the online computation only costs about one second.

Protocols with non-negligible aggregation cost. In this section, we evaluate the performance of RoundRole on protocols with non-negligible aggregation costs to demonstrate that RoundRole can obtain the optimal end-to-end performance by automatically reducing the parallel task numbers to avoid the aggregation cost becoming the bottleneck.

We use Shuffle protocol as an example because it is the only protocol with a non-negligible aggregation cost in our micro-benchmarks. Specifically, we implement the *merge shuffle* protocol [47], which merges two randomly shuffled lists into one, guaranteeing that the final distribution of the merged list remains random. We use this protocol as the aggregation protocol for Shuffle. However, the aggregation cost is orders of magnitude larger than the cost of the state-of-the-art Shuffle protocol proposed by Araki et al. [8], any extra parallelism harms the performance significantly. In this case, RoundRole can automatically reduce the number of parallel tasks to 1, obtaining the best performance, as shown in Figure 10

Adaptation on other MPC protocols. To validate the generality of RoundRole, we also adapt it to two other MPC frameworks with different numbers of parties, including the 2PC Y2A, B2A protocols through MOTION [25] and 4PC Quad Multiplication protocol (Quad) through [16]. The comparison of the bandwidth utilization and performance of these protocols is shown in Figure 12. Similar to the results in Section VI-B, RoundRole consistently improves the end-to-end efficiency by leveraging the wasted bandwidth.

D. Real-world Algorithms

We also evaluate RoundRole using three commonly used real-world algorithms to demonstrate its practical utility. Table IV and Figure 11 summarize the speedup and bandwidth utilization improvements of RoundRole against the baseline on the three algorithms across all six network settings. We can see that RoundRole achieves speedups on all the algorithms across all the network settings, the average speedup being 1.3×.

ORAM (Initialization). The ORAM initialization includes multiple Shuffle protocols that are highly imbalanced. As shown in Figure 13, P_1 and P_3 are mainly sending or receiving data to/from P_2, and P_2 takes the highest communication workload. Similar to Shuffle, RoundRole achieves significant speedups by (1) balancing the communication workload across all three physical nodes on the homogeneous settings, and (2) intelligently allocating the parties with higher workloads to the

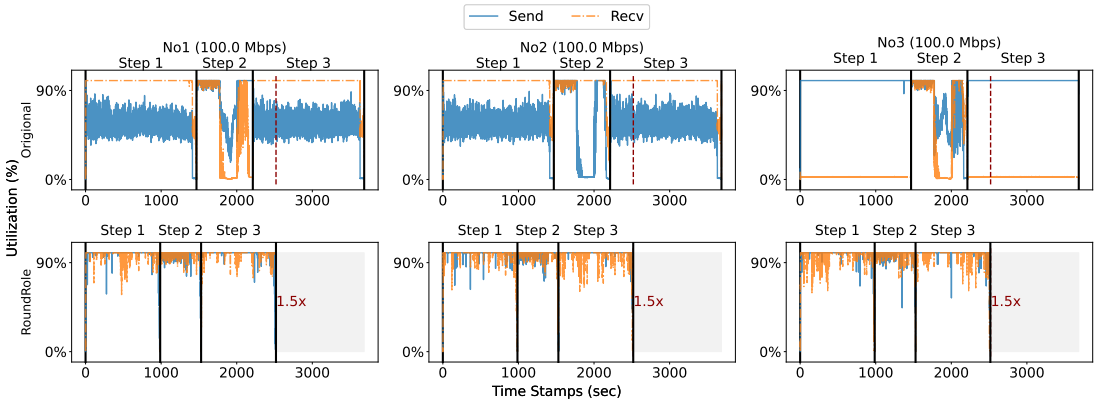


Fig. 15: Logistic Regression Communications without / with RoundRole.

nodes with higher bandwidth on the heterogeneous settings. This explains the lower bandwidth utilization of ORAM on the bottleneck node (Figure 11) and up to 1.8× speedup in the heterogeneous settings.

Sorting. Figure 14 shows the communication pattern of the sorting algorithm with and without RoundRole. Unlike ORAM, the communication pattern of sorting is more complex. The imbalanced communication part happens at the beginning of the algorithm. As shown in Figure 14, the bandwidth is almost half-utilized except for the receiving bandwidth of the second node. The following steps are the $\log(l)$ rounds of comparisons and element swapping, leading to complex communication patterns. In this case, RoundRole can mainly improve the bandwidth utilization from the beginning of the algorithm, resulting in 1.3× speedup on Homo1.

Logistic regression (LR). For algorithms with multiple asymmetric protocol components, RoundRole can significantly improve the performance by optimizing the communication patterns of each component. Each iteration of the LR computation incorporates three steps: (1) F-Mul, (2) comparison and B2A, as well as (3) F-Mul operations. All three steps are asymmetric and, therefore, can be optimized by RoundRole. As shown in Figure 15, on the Homo1 setting, the bandwidth utilization of each iteration of the LR algorithm can be optimized to almost 100% with RoundRole, leading to a 1.5× speedup each iteration.

VII. Related Work

Load-balanced MPC is recently proposed by Lu et al. [48], which aims to achieve load balance on heterogeneous resources for the *server-aided garbled circuit (GC)* settings, where a series of parties construct a segment of the same circuit with different sizes according to their resources, i.e., network bandwidth, and send the circuit to the aided server, who evaluates the circuits. As the main communication cost of GC is the initial circuit transmission, the load balance is achieved straightforwardly by splitting the circuit into segments of different sizes.

Comparably, secret sharing incurs much more complicated communication patterns, where each communication varies for different operations, as Section III illustrates. The first step to achieve load balance in secret sharing is presented in Harth-Kitzerow et al. [16], who construct two protocols with simple communication patterns for 3PC and 4PC multiplications and optimize these two customized protocols to fit various network settings for high throughput. As a comparison, RoundRole is a generally applicable method that can optimize *any* secret sharing protocols with asymmetric communication patterns, including their customized protocols.

Goyal et al. [49] theoretically define the asymmetric MPC problem, focusing on the differing latencies among parties and the resulting security challenges. This is orthogonal to our work, as we focus on the communication patterns of secret sharing protocols and varied bandwidth.

Other protocol-agnostic optimizations. Currently, there exist many researchers focusing on optimizing the execution of secret-sharing MPC from the system perspective, which can be categorized into two groups: (1) Optimizations aiming to reduce the communication rounds, such as COMBINE [32] and MP-SPDZ [12], [50]. These methods exploit *vectorization* to combine multiple independent operations into one large vectorized operation and evaluate these operations in the same communication round to amortize the communication latency; and (2) Scheduling that aims to find the optimal operation assignment graph to minimize the total execution time, such as Silph [39], HyCC [51], and costCO [52]. Each operation in MPC can be implemented using different protocols, e.g., arithmetic (A), boolean (B), and Yao's garbled circuit (Y), each of which has its own "comfort zone", i.e., efficient operations. Data can be transformed among these protocols at an extra cost. The above efforts aim to find the optimal protocol assignment for a given computation task, minimizing the total execution time.

VIII. Conclusion

Over the years, researchers have developed ingenious protocols to optimize communication complexity in MPC. However, many of these protocol designs overlook a critical aspect: asymmetric communication, which can severely degrade actual network efficiency and, thus, overall performance. Since it is

challenging to address asymmetry at the protocol design level, we decouple the protocol's logical roles from the physical nodes and optimize task split and assignment at the execution level. This approach enables MPC designers to focus on protocol innovation without worrying about practical communication inefficiencies, as RoundRole automatically optimizes execution to achieve optimal performance.

We hope that our work highlights the necessity of improving MPC efficiency from multiple angles—not just by reducing communication rounds or volumes but also by improving execution-level performance. In future work, we will explore and mitigate other sub-optimal aspects of MPC execution to further advance its practicality.

IX. Acknowledgments

We thank Yusi Chen, Xiaowei Zhu and the anonymous reviewers for their help during the design and implementation of this paper. This work is supported in part by the National Key R&D Program of China 2023YFC3304802, National Natural Science Foundation of China (NSFC) Grant U2268202 and 62176135.

References

[1] J. Liagouris, V. Kalavri, M. Faisal, and M. Varia, "{SECRECY}: Secure collaborative analytics in untrusted clouds," in *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2023.

[2] R. A. Popa, C. M. Redfield, N. Zeldovich, and H. Balakrishnan, "Cryptdb: Protecting confidentiality with encrypted query processing," in *Proceedings of the twenty-third ACM symposium on operating systems principles (OSDI)*, 2011.

[3] N. Volgushev, M. Schwarzkopf, B. Getchell, M. Varia, A. Lapets, and A. Bestavros, "Conclave: secure multi-party computation on big data," in *Proceedings of the Fourteenth EuroSys Conference*, 2019.

[4] N. Jawalkar, K. Gupta, A. Basu, N. Chandran, D. Gupta, and R. Sharma, "Orca: Fss-based secure training and inference with gpus," in *IEEE Symposium on Security and Privacy (S & P)*, 2023.

[5] W.-j. Lu, Z. Huang, Q. Zhang, Y. Wang, and C. Hong, "Squirrel: A Scalable Secure Two-Party Computation Framework for Training Gradient Boosting Decision Tree," in *USENIX Security Symposium*, 2023.

[6] J.-L. Watson, S. Wagh, and R. A. Popa, "Piranha: A GPU platform for secure computation," in *USENIX Security Symposium*, 2022.

[7] S. Tan, B. Knott, Y. Tian, and D. J. Wu, "CryptGPU: Fast privacy-preserving machine learning on the GPU," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2021.

[8] T. Araki, J. Furukawa, K. Ohara, B. Pinkas, H. Rosemarin, and H. Tsuchida, "Secure graph analysis at scale," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[9] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi, "Graphsc: Parallel secure computation made easy," in *2015 IEEE symposium on security and privacy (S&P)*. IEEE, 2015.

[10] N. Koti, V. B. Kukkala, A. Patra, and B. Raj Gopal, "Graphiti: Secure graph computation made more scalable," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2024.

[11] D. R. Stinson, *Cryptography: theory and practice*. Chapman and Hall/CRC, 2005.

[12] M. Keller, "MP-SPDZ: A versatile framework for multi-party computation," in *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2020.

[13] P. Mohassel and P. Rindal, "ABY3: A mixed protocol framework for machine learning," in *Proceedings of the ACM SIGSAC conference on computer and communications security (CCS)*, 2018.

[14] D. Demmler, T. Schneider, and M. Zohner, "ABY-A framework for efficient mixed-protocol secure two-party computation." in *The Network and Distributed System Security Symposium (NDSS)*, 2015.

[15] Q. Pang, J. Zhu, H. Möllering, W. Zheng, and T. Schneider, "Bolt: Privacy-preserving, accurate and efficient inference for transformers," in *IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2024.

[16] C. Harth-Kitzerow, A. Suresh, Y. Wang, H. Yalame, G. Carle, and M. Annavaram, "High-throughput secure multiparty computation with an honest majority in various network settings," *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2025.

[17] K. Hamada, R. Kikuchi, D. Ikarashi, K. Chida, and K. Takahashi, "Practically efficient multi-party sorting protocols from comparison sort algorithms," in *Information Security and Cryptology–ICISC 2012: 15th International Conference, Seoul, Korea, November 28-30, 2012, Revised Selected Papers 15*. Springer, 2013.

[18] G. Asharov, K. Hamada, D. Ikarashi, R. Kikuchi, A. Nof, B. Pinkas, K. Takahashi, and J. Tomida, "Efficient secure three-party sorting with applications to data analysis and heavy hitters," in *Proceedings of 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[19] S. Zahur, X. Wang, M. Raykova, A. Gascón, J. Doerner, D. Evans, and J. Katz, "Revisiting square-root oram: efficient random access in multi-party computation," in *IEEE Symposium on Security and Privacy (S&P)*, 2016.

[20] B. Falk, R. Ostrovsky, M. Shtepel, and J. Zhang, "GigaDORAM: breaking the billion address barrier," in *Proceedings of the USENIX Conference on Security Symposium (USENIX Security)*, 2023.

[21] A. Vadapalli, R. Henry, and I. Goldberg, "DuORAM: A bandwidth-efficient distributed ORAM for 2-and 3-Party computation," in *Proceedings of the USENIX Conference on Security Symposium (USENIX Security)*, 2023.

[22] X. Fan, K. Chen, J. Yu, X. Zhu, Y. Chen, H. Zhang, and W. Xu, "Goram: Graph-oriented oram for efficient ego-centric queries on federated graphs," *arXiv preprint arXiv:2410.02234*, 2024.

[23] M. Blanton and C. Yuan, "Binary search in secure computation," in *Network and Distributed System Security Symposium (NDSS)*, 2022.

[24] X. Fan, K. Chen, G. Wang, X. Zhu, H. He, X. Yong, X. Jia, Y. Li, and W. Xu, "Pair-then-aggregate: Simplified and efficient parallel programming paradigm for secure multi-party computation," in *2025 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*. IEEE, 2025.

[25] L. Braun, D. Demmler, T. Schneider, and O. Tkachenko, "Motion: A framework for mixed-protocol multi-party computation," *ACM Transactions on Privacy and Security*, 2022.

[26] J. Feng, Y. Wu, H. Sun, S. Zhang, and D. Liu, "Panther: Practical secure 2-party neural network inference," *IEEE Transactions on Information Forensics and Security (TIFS)*, 2025.

[27] X. Fan, K. Chen, G. Wang, M. Zhuang, Y. Li, and W. Xu, "NFGen: Automatic non-linear function evaluation code generator for general-purpose MPC platforms," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2022.

[28] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "{ABY2. 0}: Improved {Mixed-Protocol} secure {Two-Party} computation," in *USENIX Security Symposium (USENIX Security)*, 2021.

[29] A. Shamir, "How to share a secret," *Communications of the ACM*, 1979.

[30] J. Ma, Y. Zheng, J. Feng, D. Zhao, H. Wu, W. Fang, J. Tan, C. Yu, B. Zhang, and L. Wang, "{SecretFlow-SPU}: A performant and user-friendly framework for privacy-preserving machine learning," in *USENIX Annual Technical Conference (ATC)*, 2023.

[31] Y. Zheng, H. Duan, C. Wang, R. Wang, and S. Nepal, "Securely and efficiently outsourcing decision tree inference," *IEEE Transactions on Dependable and Secure Computing*, 2020.

[32] B. Levy, M. Ishaq, B. Sherman, L. Kennard, A. Milanova, and V. Zikas, "Combine: Compilation and backend-independent vectorization for multi-party computation," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2023.

[33] V. K. Yadav, N. Andola, S. Verma, and S. Venkatesan, "A survey of oblivious transfer protocol," *ACM Computing Surveys (CSUR)*, 2022.

[34] B. Pinkas, T. Schneider, and M. Zohner, "Faster private set intersection based on ot extension," in *Proceedings of the USENIX Conference on Security Symposium*, 2014.

[35] B. Pinkas, M. Rosulek, N. Trieu, and A. Yanai, "Spot-light: lightweight private set intersection from sparse ot extension," in *Annual International Cryptology Conference (CRYPTO)*. Springer, 2019.

[36] M. Orrù, E. Orsini, and P. Scholl, "Actively secure 1-out-of-n ot extension with application to private set intersection," in *Topics in Cryptology–CT-RSA 2017: The Cryptographers' Track at the RSA*

Conference 2017, San Francisco, CA, USA, February 14–17, 2017, Proceedings. Springer, 2017.
[37] G. Di Crescenzo, T. Malkin, and R. Ostrovsky, "Single database private information retrieval implies oblivious transfer," in *International Conference on the Theory and Application of Cryptographic Techniques (EUROCRYPT)*. Springer, 2000.
[38] Y.-C. Chang, "Single database private information retrieval with logarithmic communication," in *Information Security and Privacy (ACISP)*. Springer, 2004.
[39] E. Chen, J. Zhu, A. Ozdemir, R. S. Wahby, F. Brown, and W. Zheng, "Silph: A framework for scalable and accurate generation of hybrid mpc protocols," in *2023 IEEE Symposium on Security and Privacy (S & P)*. IEEE, 2023.
[40] M. Islam, S. S. Arora, R. Chatterjee, P. Rindal, and M. Shirvanian, "Compact: Approximating complex activation functions for secure computation," *Proceedings on Privacy Enhancing Technologies (PoPETS)*, 2024.
[41] Y. Lindell, "Secure multiparty computation," *Communications of the ACM*, 2020.
[42] A. Dalskov, D. Escudero, and M. Keller, "Fantastic four: Honest-majority four-party secure computation with malicious security," in *USENIX Security Symposium (USENIX Security)*, 2021.
[43] G. Desaulniers, J. Desrosiers, and M. M. Solomon, *Column Generation*. Springer Science & Business Media, 2006, vol. 5.
[44] R. Canetti, "Security and composition of multiparty cryptographic protocols," *Journal of CRYPTOLOGY*, vol. 13, no. 1, pp. 143–202, 2000.
[45] V. Jacobson, "Congestion avoidance and control," *ACM SIGCOMM Computer Communication Review*, 1988.
[46] K. Kaur, J. Singh, and N. S. Ghumman, "Mininet as software defined networking testing platform," in *International conference on communication, computing & systems (ICCCS)*, 2014.
[47] A. Bacher, O. Bodini, A. Hollender, and J. Lumbroso, "Mergeshuffle: A very fast, parallel random permutation algorithm," in *CEUR Workshop Proceedings*. CEUR-WS, 2018.
[48] Y. Lu, B. Zhang, and K. Ren, "Load-balanced server-aided mpc in heterogeneous computing," *IEEE Transactions on Information Forensics and Security (TIFS)*.
[49] V. Goyal, C.-D. Liu-Zhang, and R. Ostrovsky, "Asymmetric multi-party computation," in *Conference on Information-Theoretic Cryptography*, 2023.
[50] M. Keller, "How to scale multi-party computation," *Cryptology ePrint Archive*, 2024.
[51] N. Büscher, D. Demmler, S. Katzenbeisser, D. Kretzmer, and T. Schneider, "HyCC: Compilation of hybrid protocols for practical secure computation," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.
[52] V. Fang, L. Brown, W. Lin, W. Zheng, A. Panda, and R. A. Popa, "Costco: An automatic cost modeling framework for secure multi-party computation," in *2022 IEEE 7th European Symposium on Security and Privacy (Euro S&P)*. IEEE, 2022.

APPENDIX

A. Security Analysis

Theorem 1 (Security Preservation of RoundRole)**.** *For any secret-sharing protocol* Π*,* RoundRole *preserves the same security guarantee as* Π *in the presence of any adversary that can corrupt at most* t *physical nodes under the same adversary model as* Π*.*

Proof. Assume for contradiction that there exists an adversary $\mathcal{A}$ that, under the same adversary model as Π, corrupts at most t physical nodes and can distinguish the RoundRole-optimized execution from an ideal execution with non-negligible advantage.

We now construct an adversary $\mathcal{B}$ for the original protocol Π as follows.

Construction of $\mathcal{B}$:

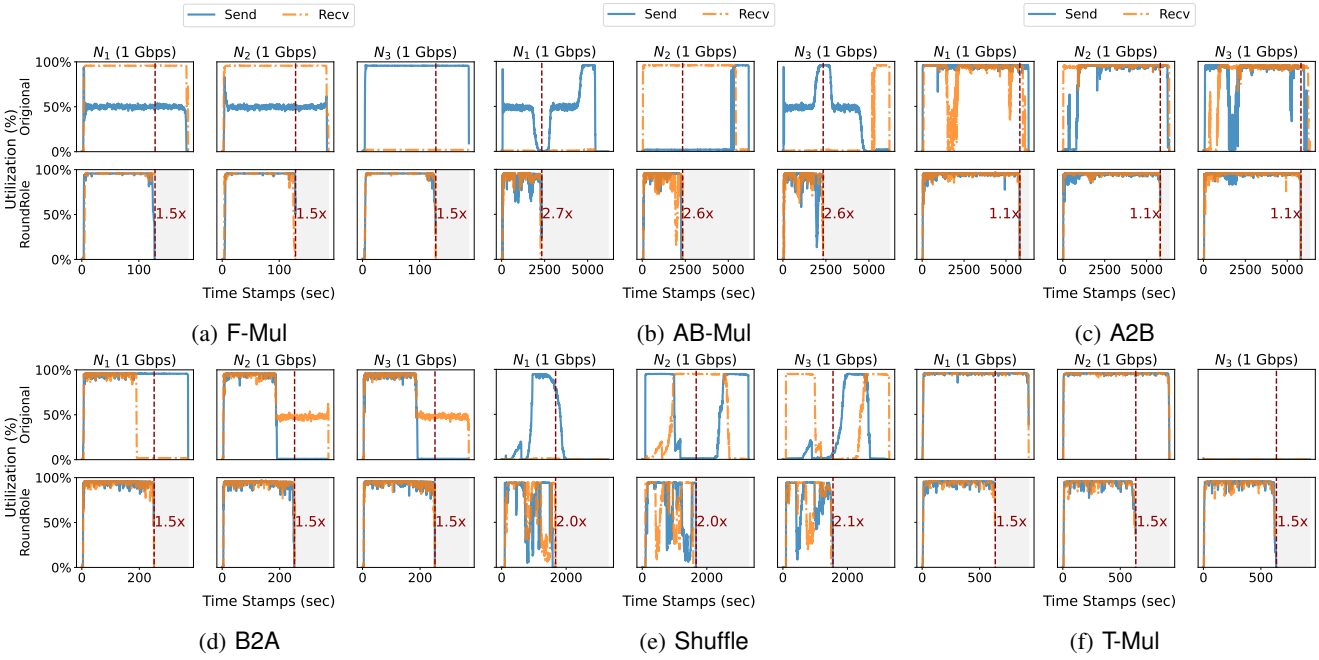
1) $\mathcal{B}$ receives as input the common reference string and any other public parameters of Π, as in the original setting.
2) $\mathcal{B}$ simulates the execution of Π from the RoundRole-optimized execution by reassigning the logical roles back to the original physical nodes using the inverse logical-to-physical mappings defined in RoundRole. The resulting process is identical to the original execution of Π because RoundRole only permutes the logical roles across independent tasks without modifying the secret-sharing mechanism.
3) $\mathcal{B}$ runs $\mathcal{A}$ on the simulated execution. Whenever $\mathcal{A}$ makes a query or receives a message in the simulated execution, $\mathcal{B}$ forwards this query/response faithfully to its own interface as defined by the protocol Π.
4) Finally, $\mathcal{B}$ outputs whatever $\mathcal{A}$ outputs.

Since RoundRole only permutes the logical roles across independent tasks without modifying the secret-sharing mechanism, the view of each physical node in the simulated execution is identically distributed to its view in the execution of Π. Therefore, if $\mathcal{A}$ distinguishes the RoundRole-optimized execution from an ideal execution, then $\mathcal{B}$ will distinguish the original protocol Π from its ideal execution with the same non-negligible advantage. This contradicts the assumed security of Π.

Thus, no such adversary $\mathcal{A}$ exists, and RoundRole preserves the same security guarantee as the original protocol Π. □

B. Communication Patterns on Different Network Settings

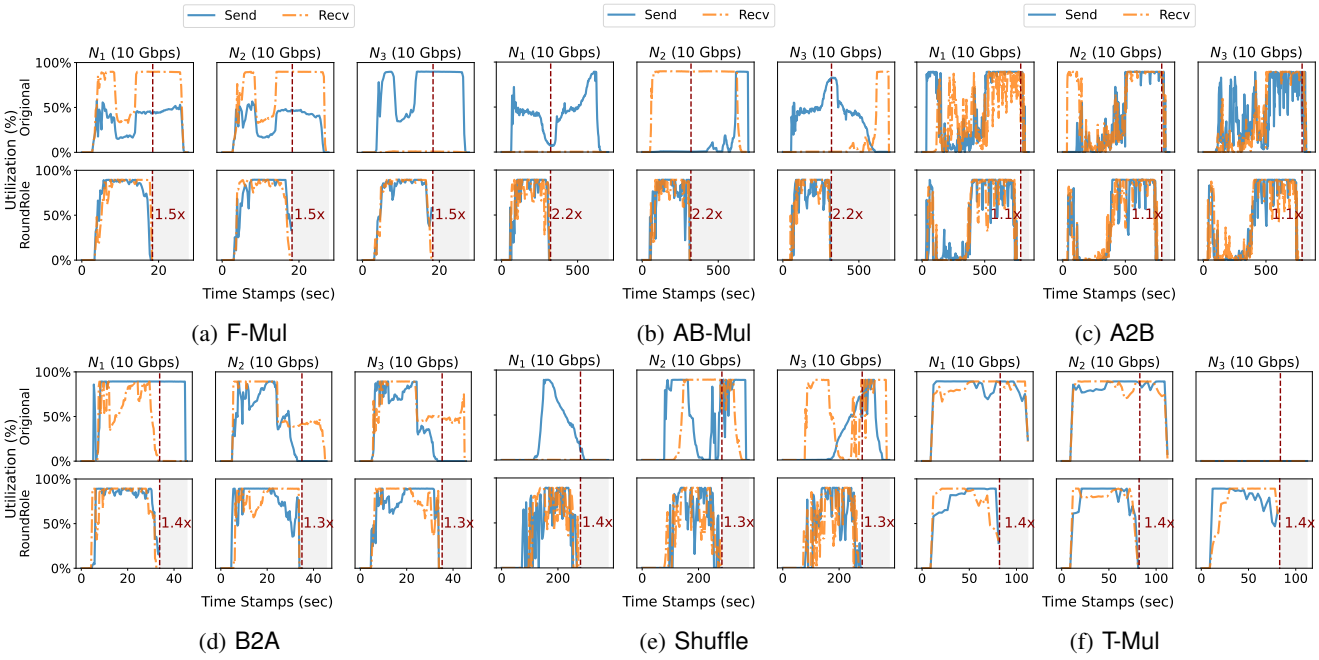
Figure 16 to Figure 20 show the communication traces of the micro-benchmark protocols with and without RoundRole on different network settings.



(a) F-Mul (b) AB-Mul (c) A2B

(d) B2A (e) Shuffle (f) T-Mul

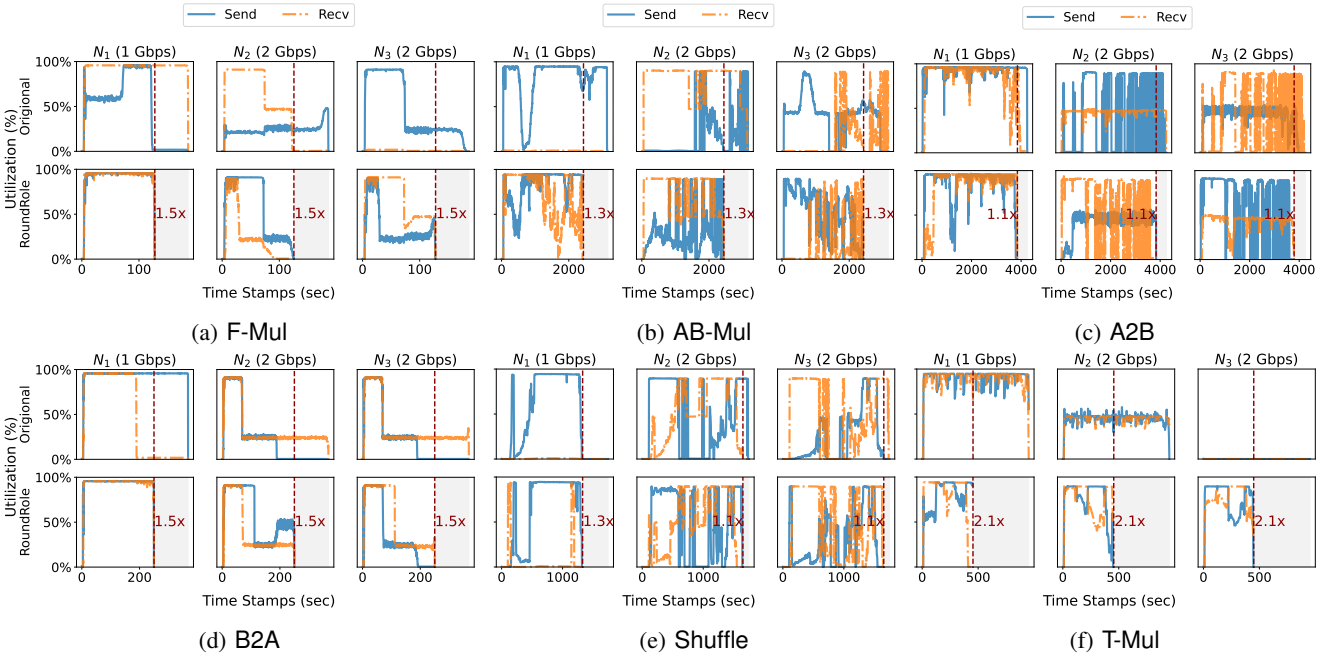
Fig. 16: Communication Traces of the Micro-benchmark Protocols w/o RoundRole on Homo2.



(a) F-Mul (b) AB-Mul (c) A2B

(d) B2A (e) Shuffle (f) T-Mul

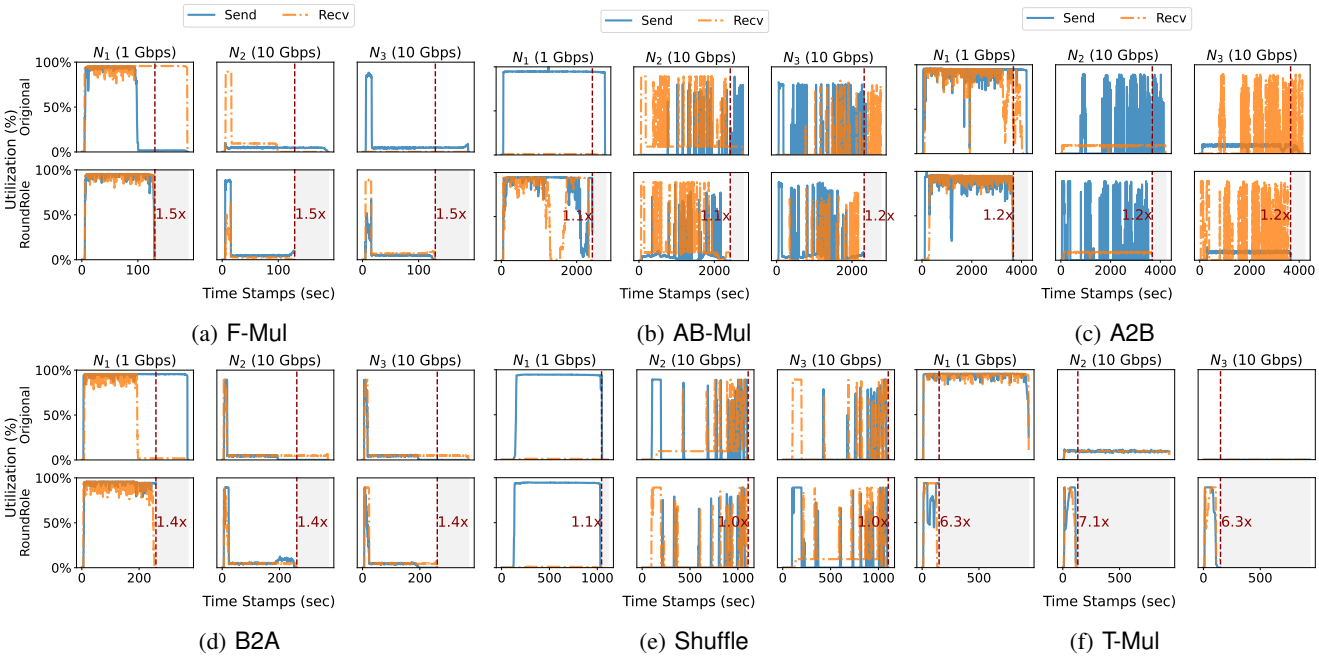
Fig. 17: Communication Traces of the Micro-benchmark Protocols w/o RoundRole on Homo3.



(a) F-Mul (b) AB-Mul (c) A2B

(d) B2A (e) Shuffle (f) T-Mul

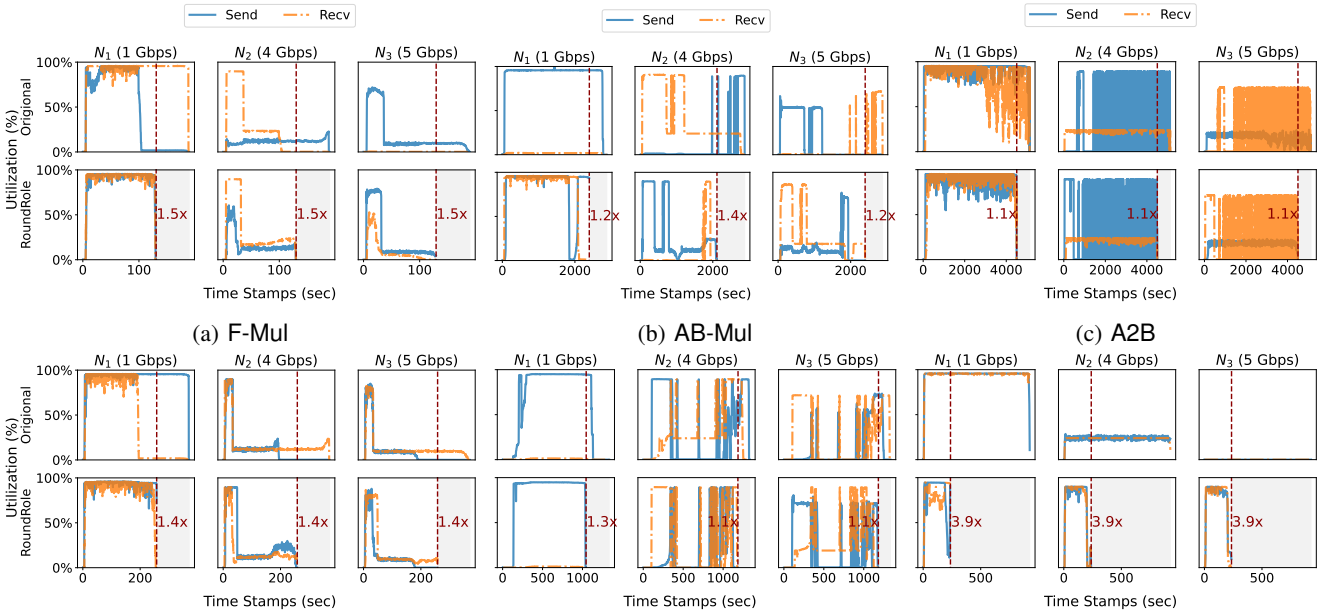
Fig. 18: Communication Traces of the Micro-benchmark Protocols w/o RoundRole on Hetero1.



(a) F-Mul (b) AB-Mul (c) A2B

(d) B2A (e) Shuffle (f) T-Mul

Fig. 19: Communication Traces of the Micro-benchmark Protocols w/o RoundRole on Hetero2.



(a) F-Mul (b) AB-Mul (c) A2B

(d) B2A (e) Shuffle (f) T-Mul

Fig. 20: Communication Traces of the Micro-benchmark Protocols w/o RoundRole on Hetero3.