Application of the Divide-and-Conquer Approach to the Fragmentation and Reassembly of IPv4 Packets Based on MTU in a Network Simulator

Reinhard Alfonzo Hutabarat – 13524056
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
Email: reinhardalfonso@gmail.com, 13524056@std.stei.itb.ac.id

***Abstract*—When an IPv4 packet exceeds the Maximum Transmission Unit (MTU) of a network link, it must be fragmented into smaller units and reassembled at the destination. This paper formally analyzes this mechanism as an application of the Divide-and-Conquer (D&C) paradigm. The Divide phase decomposes the payload into MTU-aligned fragments using the 8-byte alignment constraint, producing $k = \lceil N/m \rceil$ subproblems. The Conquer phase transmits each fragment independently, with recursive re-fragmentation at intermediate routers when downstream links have smaller MTUs. The Combine phase reassembles fragments at the destination using offset-based buffer management and contiguity verification. Experiments on the MAGI System, a custom-built C++ network simulator, validate this mapping. In a multi-hop topology with MTU values of 300 and 100 bytes, a 508-byte payload was fragmented into 2 fragments at the source, recursively re-fragmented into 7 sub-fragments at the router, and faithfully reassembled at the destination with zero data loss. The results confirm that D&C guarantees correctness across heterogeneous links, with bounded overhead and predictable efficiency, demonstrating that IPv4 fragmentation is a rigorous instantiation of the Divide-and-Conquer strategy.**

***Keywords*—Divide-and-Conquer, IPv4 Fragmentation, Maximum Transmission Unit, Packet Reassembly, Recursive Fragmentation, Network Simulator**

I. Introduction

Internet Protocol version 4 (IPv4) is the foundation of data communication in modern internet networks. Every IPv4 packet transmitted across a network must comply with the maximum size limit allowed by each physical link it traverses, known as the *Maximum Transmission Unit* (MTU). In a heterogeneous network ecosystem, differences in MTU capabilities between links, for example, between a local Ethernet link (MTU of 1500 bytes) and a wireless point-to-point link (with a much smaller MTU), create an algorithmic challenge: how to transmit large packets over a link whose MTU is smaller than the packet size.

To address this issue, the IPv4 protocol implements a fragmentation and reassembly mechanism. When a packet's size exceeds the MTU of the outgoing link, the network layer algorithm splits the packet into several smaller, independent fragments. Each fragment is sent separately to the destination, and at the receiving host, the fragments are reassembled into a complete packet. This process naturally maps to the *Divide-and-Conquer* (D&C) paradigm: a large problem (the entire packet) is broken down into smaller subproblems (fragments), each subproblem is solved independently (separate transmissions), and the solutions to the subproblems are recombined (reassembly) to produce a final solution identical to the original problem.

Although IPv4 fragmentation has been documented in detail in RFC 791 [1] and networking literature [3], [4], explicit analyses that formalize this mechanism as an application of the Divide-and-Conquer algorithmic strategy are still rarely discussed. Most of the networking literature describes fragmentation in a descriptive manner, without linking it to a systematic algorithmic analysis framework. In fact, this mapping provides valuable insights: the complexity of fragmentation can be analyzed asymptotically, the recursive nature of re-fragmentation in routers mirrors that of classical D&C, and the trade-off between overhead and efficiency can be quantified mathematically.

This paper aims to formally analyze how the IPv4 fragmentation and reassembly mechanisms implement the Divide-and-Conquer paradigm, and to validate this analysis through experiments on a specially built *network simulator*. The contributions of this paper include:

1) an explicit mapping of the three phases of D&C, Divide, Conquer, and Combine, to the stages of IPv4 fragmentation and reassembly
2) an analysis of the asymptotic complexity for each phase
3) experimental evidence demonstrating the correctness of the algorithm and the recursive nature of fragmentation in multi-hop topologies.

II. Theoretical Basis

A. IPv4 Packet Structure

The Internet Protocol version 4 (IPv4) remains the foundational network layer protocol that governs how datagrams are formatted, addressed, and routed across heterogeneous internetworks. Every IPv4 datagram consists of a header, whose minimum length is 20 bytes, followed by a variable-length payload whose size may reach up to 65515 bytes.

Among the twelve header fields, four of them play a central role in the fragmentation and reassembly mechanism, namely the Total Length field, the Identification field, the Flags field, and the Fragment Offset field [1].

The *Total Length* field occupies 16 bits and encodes the entire datagram size, including both the header and the payload. Consequently, the theoretical maximum datagram size equals $2^{16} - 1 = 65535$ bytes. The *Identification* field, also 16 bits wide, is assigned by the sending host to every outgoing datagram. All fragments derived from the same original datagram inherit an identical identification value, which serves as the primary key for the destination host to group related fragments during reassembly.

The *Flags* field is a 3-bit control field whose two most significant bits are relevant to fragmentation. The *Don't Fragment* (DF) bit, when set to one, instructs every intermediate router to refrain from fragmenting the datagram. If such a datagram exceeds the Maximum Transmission Unit of an outgoing link, the router discards it and returns an ICMP "Fragmentation Needed" error message to the source. Conversely, when the DF bit is cleared, the router is permitted to fragment the datagram. The *More Fragments* (MF) bit signals whether additional fragments follow the current one. For every fragment except the last, the MF bit is set to one. For the final fragment, it is cleared to zero.

The *Fragment Offset* field spans 13 bits and indicates the position of the fragment's payload relative to the beginning of the original datagram payload. Crucially, this field is measured in units of 8 bytes. Therefore, every fragment's payload, with the possible exception of the last fragment, must be an exact multiple of 8 bytes. This alignment constraint directly determines the maximum usable payload per fragment. Given a link with MTU M and a standard IPv4 header of $H = 20$ bytes, the maximum payload per fragment m is computed as:

$$m = \left\lfloor \frac{M - H}{8} \right\rfloor \times 8 \tag{1}$$

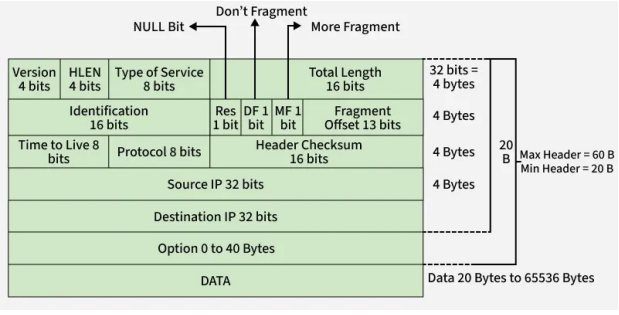


Fig. 1. IPv4 header structure. The identification, flags, and fragmentation offset fields are directly involved in the fragmentation and reassembly process. [Source: https://www.geeksforgeeks.org/computer-networks/what-is-ipv4/]

Figure 1 illustrates the IPv4 header layout with the fragmentation-related fields highlighted. The interplay among these fields ensures that fragments can be independently routed, correctly grouped at the destination, and accurately repositioned to reconstruct the original payload.

B. Maximum Transmission Unit

The Maximum Transmission Unit (MTU) is defined as the largest protocol data unit that a given data link layer technology can carry in a single frame transmission [4], [6], [9]. The MTU is an inherent property of the underlying physical and data link layer medium, and it varies substantially across different network technologies. Table I enumerates representative MTU values for common link types.

TABLE I
TYPICAL MTU VALUES FOR COMMON NETWORK TECHNOLOGIES

Network Technology	MTU (bytes)	Typical Usage
Ethernet (IEEE 802.3)	1500	LAN, data center
IEEE 802.5 Token Ring	4464	Legacy LAN
PPP	576	Dial-up, serial links
X.25	576	Public data networks
Loopback interface	65536	Local host testing
Jumbo Frames	9000	High-performance LAN

The existence of diverse MTU values reflects a fundamental engineering trade-off. A larger MTU reduces the per-byte overhead because the fixed header cost is amortized over a greater payload, thereby improving bandwidth utilization. Conversely, a smaller MTU reduces the probability that a frame is corrupted by bit errors during transmission, since shorter frames are statistically less susceptible to noise. It also diminishes the queuing delay at intermediate routers, which is particularly relevant for real-time traffic [5].

When an IPv4 datagram traverses a path composed of multiple links with heterogeneous MTU values, it may encounter a link whose MTU is smaller than the datagram size. At that point, the router connected to the outgoing link must fragment the datagram. RFC 791 mandates that every host must be prepared to accept a datagram of up to 576 bytes, which establishes the minimum reassembly buffer size [1]. If the original datagram exceeds this minimum, fragmentation becomes unavoidable on constrained links. Moreover, a fragment produced at one router may itself exceed the MTU of a subsequent link, triggering a *re-fragmentation* event. This recursive fragmentation is a direct consequence of the progressive narrowing of MTU along the transmission path.

C. The Divide-and-Conquer Paradigm

The Divide-and-Conquer (D&C) paradigm constitutes one of the most fundamental algorithm design strategies in computer science. Historically rooted in the military strategy known as *divide ut imperes* ("divide so that you may rule"), this approach was adapted into computational problem solving and has since become a cornerstone of algorithm theory [3], [7].

The paradigm decomposes the solution process into three sequential phases. In the Divide phase, the original problem of size n is partitioned into r subproblems $P_1, P_2, \ldots, P_r$,

each of which shares the same structural characteristics as the original problem but with a strictly smaller size. In the Conquer phase, each subproblem is solved individually. If the subproblem is sufficiently small, it is solved directly by a trivial method. Otherwise, the D&C strategy is applied recursively. In the Combine phase, the solutions of all subproblems are merged to produce the solution of the original problem [2].

The general structure of a D&C algorithm is expressed in Algorithm 1.

Algorithm 1 Divide-and-Conquer (General Form)

Require: Problem P of size n
Ensure: Solution of P
if $n \leq n_0$ **then**
  // Base case: problem is small enough
  Solve P directly
else
  Divide P into subproblems $P_1, P_2, \ldots, P_r$ of sizes $n_1, n_2, \ldots, n_r$
  for each subproblem P_i **do**
    Conquer: invoke DIVIDEANDCONQUER(P_i, n_i) recursively
  end for
  Combine the solutions of $P_1, P_2, \ldots, P_r$ into the solution of P
end if

The time complexity of a D&C algorithm is governed by the recurrence relation:

$$T(n) = \begin{cases} g(n), & \text{if } n \leq n_0 \\ T(n_1) + T(n_2) + \cdots + T(n_r) + f(n), & \text{if } n > n_0 \end{cases} \tag{2}$$

where $g(n)$ denotes the cost of solving a base case directly, and $f(n)$ represents the cost of the Combine phase. The Divide phase is typically absorbed into $O(1)$ and thus omitted from the recurrence [3]. In the common scenario where the problem is bisected into two equal subproblems of size $n/2$, the recurrence simplifies to:

$$T(n) = \begin{cases} g(n), & \text{if } n \leq n_0 \\ 2\,T(n/2) + f(n), & \text{if } n > n_0 \end{cases} \tag{3}$$

Several classical algorithms exemplify this paradigm. *Merge Sort* divides an array at its midpoint, recursively sorts each half, and merges the two sorted halves in $O(n)$ time, yielding an overall complexity of $O(n \log n)$ [3]. *Quick Sort* partitions the array around a pivot element such that all elements in the left partition are less than or equal to the pivot, and all elements in the right partition are greater, then recursively sorts each partition [3]. The *binary exponentiation* algorithm computes a^n by exploiting the identity $a^n = a^{n/2} \cdot a^{n/2}$, reducing the complexity from $O(n)$ under brute force to $O(\log n)$ [3]. The *Master Theorem* provides a convenient method for solving recurrence relations of the form $T(n) = aT(n/b) + cn^d$ without iterative expansion [3]:

$$T(n) = \begin{cases} O(n^d), & \text{if } a < b^d \\ O(n^d \log n), & \text{if } a = b^d \\ O(n^{\log_b a}), & \text{if } a > b^d \end{cases} \tag{4}$$

D. IPv4 Fragmentation as Divide-and-Conquer

The process of IPv4 fragmentation and reassembly constitutes a natural instantiation of the Divide-and-Conquer paradigm. The correspondence between the three D&C phases and the network operations is as follows.

Divide Phase. Consider a datagram whose payload has size N bytes that must be transmitted across a link with MTU M. If the total datagram size $N + H$ exceeds M, where H denotes the IPv4 header length, the router must decompose the payload into smaller blocks. The maximum usable payload per fragment is determined by the 8-byte alignment constraint:

$$m = \left\lfloor \frac{M - H}{8} \right\rfloor \times 8 \tag{5}$$

The total number of fragments k produced is:

$$k = \left\lceil \frac{N}{m} \right\rceil \tag{6}$$

Each fragment F_i, for $i = 1, 2, \ldots, k$, carries a payload of size at most m bytes, a freshly constructed IPv4 header of H bytes, and the appropriate Fragment Offset and MF flag values. The base case of this decomposition occurs when $N \leq m$, meaning the payload fits within a single fragment and no further division is necessary.

Conquer Phase. Each fragment F_i is transmitted independently through the network. The connectionless nature of IP guarantees no ordering, no delivery assurance, and no route consistency among fragments. Each fragment may traverse a distinct sequence of routers, experience different propagation delays, and potentially be subjected to further fragmentation at intermediate routers if the next link has a smaller MTU. This recursive re-fragmentation mirrors the recursive invocation of a D&C algorithm on a subproblem that remains too large after the initial division.

Combine Phase. At the destination host, a reassembly buffer collects incoming fragments. The Identification field groups fragments originating from the same datagram, while the Fragment Offset field determines the correct byte position of each fragment's payload within the reconstructed datagram. The reassembly is complete when all bytes from offset zero to the total payload length are received contiguously, which is signaled by the arrival of the fragment with MF $= 0$. The destination host then delivers the reassembled datagram to the transport layer.

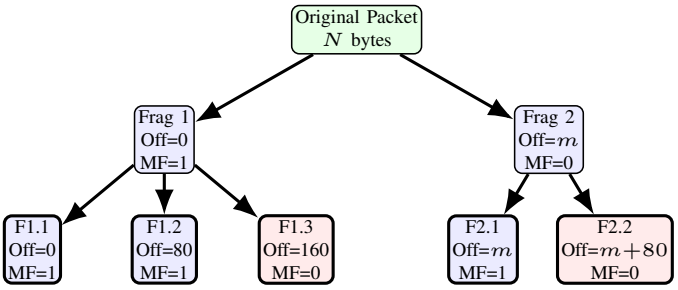


Fig. 2. The D&C recursion tree for IPv4 fragmentation. The original packet is first divided at the source host (H1). Fragments that still exceed the MTU of a downstream link are recursively re-fragmented at intermediate routers (R1). Red-shaded nodes represent the final fragments that reach the destination without further division.

Figure 2 depicts the recursion tree that arises when a packet undergoes fragmentation at the source host followed by re-fragmentation at an intermediate router. The structure is directly analogous to a D&C recursion tree, the root represents the original problem, the intermediate nodes represent subproblems produced by successive divisions, and the leaf nodes represent base cases that can be transmitted without further decomposition.

The overhead introduced by fragmentation is quantifiable. Each fragment carries its own H-byte header, so the total number of bytes transmitted across all fragments is:

$$B_{\text{total}} = k \cdot H + N \tag{7}$$

The transmission efficiency, defined as the ratio of useful payload to total bytes transmitted, is therefore:

$$\eta = \frac{N}{k \cdot H + N} = \frac{N}{\lceil N/m \rceil \cdot H + N} \tag{8}$$

As the MTU decreases, the number of fragments k increases, and the efficiency η decreases accordingly. Nevertheless, the D&C structure guarantees that the original payload is faithfully reconstructed at the destination, provided that all fragments arrive within the reassembly timeout period and no fragment is lost in transit.

III. Algorithm Analysis

A. Fragmentation Algorithm

The fragmentation algorithm operates as the Divide phase of the Divide-and-Conquer paradigm. It serves a dual role: as a boundary checker that determines whether the problem has reached the base case, and as the executor of the recursive decomposition when the recursive case applies.

Base Case. When the algorithm detects that the total datagram size (payload plus header) is less than or equal to the MTU of the outgoing interface ($N \leq M$), the division operation is classified as irrelevant. The packet can be transmitted directly to the physical medium without further manipulation.

Recursive Case. When $N > M$, the protocol enters the exception handling procedure. The first step is to verify the status of the Don't Fragment (DF) bit in the header. If the DF flag indicates that fragmentation is prohibited (DF = 1), the transmission is aborted; the packet is dropped and the algorithm returns an error notification using the ICMP protocol (Type 3, Code 4: Fragmentation Needed and DF Set) back to the source address. However, if fragmentation is permitted (DF = 0), the algorithm executes the iterative cutting of the payload into discrete unit blocks with absolute length $m = \lfloor (M - 20)/8 \rfloor \times 8$.

The actual C++ implementation of the fragmentation algorithm from the MAGI System network simulator (`layer3/IPv4Packet.cpp`) is shown in Figure 3.

```cpp
vector<IPv4Packet> IPv4Packet::fragment(size_t mtu) const {
    vector<IPv4Packet> fragments;

    size_t header_size = static_cast<size_t>(version_ihl & 0x0F) * 4;
    if (header_size == 0) header_size = IPV4_HEADER_SIZE;

    size_t total_size = header_size + payload_.size();
    if (total_size <= mtu) {
        fragments.emplace_back(src_ip, dst_ip, protocol,
            payload_, ttl, identification, flags_fragment_offset);
        fragments.back().tos = tos;
        return fragments;  // Base case: no fragmentation needed
    }

    if (flags_fragment_offset & IPV4_FLAG_DF) {
        throw PacketException(
            "The packet has the DF flag set, so it must not be fragmented");
    }

    if (mtu <= header_size + 8) {
        throw PacketException("The MTU is too small for IPv4 fragmentation");
    }

    // 8-byte alignment constraint
    size_t max_payload = ((mtu - header_size) / 8) * 8;
    if (max_payload == 0) {
        throw PacketException("The payload per fragment is 0");
    }

    uint16_t base_offset = get_fragment_offset_units();
    bool original_mf = has_more_fragments();

    size_t offset = 0;
    while (offset < payload_.size()) {
        size_t remaining = payload_.size() - offset;
        size_t chunk = min(max_payload, remaining);

        bool is_last = (offset + chunk == payload_.size());

        vector<uint8_t> frag_payload(
            payload_.begin() + static_cast<ptrdiff_t>(offset),
            payload_.begin() + static_cast<ptrdiff_t>(offset + chunk));

        uint16_t offset_units =
            static_cast<uint16_t>(base_offset + (offset / 8));
        uint16_t flags = offset_units & IPV4_OFFSET_MASK;

        if (!is_last || original_mf) {
            flags |= IPV4_FLAG_MF;  // More Fragments
        }

        IPv4Packet frag(src_ip, dst_ip, protocol,
            move(frag_payload), ttl, identification, flags);
        frag.tos = tos;
        fragments.push_back(move(frag));

        offset += chunk;
    }
    return fragments;
}
```

Fig. 3. IPv4 Fragmentation Algorithm (Divide Phase)

Complexity Analysis of the Divide Phase. The computational time delegated to decomposing the packet is asymptotically proportional to the number of fragments produced. If we define N as the original payload size and m as the maximum payload per fragment, the algorithm executes $k = \lceil N/m \rceil$ iterations. Thus, the time complexity of this phase is in the deterministic order $O(N/m)$. Since the payload copying size per iteration is statically bounded by m, the aggregate copying time scales linearly with N.

From a space complexity perspective, this geometric cutting operation exploits the routing subsystem's memory structure by demanding additional RAM buffer allocation to insert replicated IPv4 headers per new fragment. This operation consumes

asymptotically $O(N/m)$ additional memory space (precisely $20 \times \lceil N/m \rceil$ bytes) while the persistence queue resides in the source device's egress interface buffer, representing a non-trivial spatial memory burden.

B. Reassembly Algorithm

The reassembly process at the destination fundamentally operates at a complexity level far exceeding the fragmentation mechanism. This is due to the nature of IP as a connectionless (stateless) medium that provides no delivery ordering guarantees. Packet fragments have a high probability of arriving randomly or out-of-order, experiencing duplication at the switch level, or disappearing entirely (packet loss) due to intermediate router queue congestion.

The reassembly key is a 4-tuple: $(src\_ip, dst\_ip, protocol, identification)$. This tuple uniquely identifies all fragments belonging to the same original datagram. The actual C++ implementation from the MAGI System network simulator (`core/Host.cpp`) is shown in Figure 4.

```cpp
bool Host::handle_fragment_and_reassemble(
    const IPv4Packet& ip,
    std::vector<uint8_t>& reassembled_payload) {

    // Reassembly key: (src_ip, dst_ip, protocol, identification)
    std::string key = make_reassembly_key(ip);
    ReassemblyBuffer& buffer = reassembly_buffers_[key];

    size_t offset =
        static_cast<size_t>(ip.get_fragment_offset_bytes());
    buffer.fragments[offset] = ip.get_payload();

    // Terminal marker: last fragment (MF=0) defines total size
    if (!ip.has_more_fragments()) {
        buffer.has_last_fragment = true;
        buffer.total_payload_size =
            offset + ip.get_payload().size();
    }

    if (!buffer.has_last_fragment) {
        return false;  // Wait for more fragments
    }

    // Contiguity check: iterate sorted offsets, verify no gaps
    size_t cursor = 0;
    for (const auto& [frag_off, frag_payload]
         : buffer.fragments) {
        if (frag_off != cursor) {
            return false;  // Gap detected
        }
        cursor += frag_payload.size();
    }

    if (cursor < buffer.total_payload_size) {
        return false;  // Incomplete
    }

    // Combine: concatenate all fragments in order
    reassembled_payload.clear();
    reassembled_payload.reserve(buffer.total_payload_size);
    for (const auto& [frag_off, frag_payload]
         : buffer.fragments) {
        reassembled_payload.insert(
            reassembled_payload.end(),
            frag_payload.begin(), frag_payload.end());
    }

    reassembly_buffers_.erase(key);
    return true;
}
```

Fig. 4. IPv4 Reassembly Algorithm (Combine Phase)

Complexity Analysis of the Combine Phase. Through the implementation of the reassembly buffer with offset-based fragment storage, the Combine phase avoids duplicate memory movement. The contiguity check iterates over the sorted fragment map once, yielding $O(k)$ time for the verification step where k is the number of fragments. The final concatenation of all fragments is $O(N)$ where N is the total payload size. Overall, the reassembly time complexity is $O(k + N)$. The space complexity is $O(N)$ for the reassembly buffer that stores all fragment payloads until the complete datagram is reconstructed.

C. Complexity Comparison

Based on the algorithmic exploration above, the manifestation of D&C mapping on the IPv4 fragmentation mechanism convincingly demonstrates measurable deterministic operational trade-off consequences. Payload decomposition converts infrastructure limitations into mathematical computational burden and transmission metadata inflation (additional static headers).

Table II carefully illustrates how the de-escalation or narrowing of MTU variation transforms the totality of subproblem resolution dynamics, tested on a constant payload load scaled at 508 bytes.

TABLE II
FRAGMENTATION COMPLEXITY VS MTU (PAYLOAD = 508 BYTES)

MTU	k	m	**Overhead**	η
1500	1	1480	20 B	96.2%
576	1	556	20 B	96.3%
300	2	280	40 B	92.7%
200	3	184	60 B	89.5%
100	7	80	140 B	78.4%
68	9	48	180 B	73.9%

Note: k = number of fragments, m = max payload per fragment (bytes), η = transmission efficiency.

The table demonstrates that as the MTU decreases, the number of fragments increases, leading to higher header overhead and lower transmission efficiency. The efficiency metric η is defined as:

$$\eta = \frac{N}{k \cdot H + N} = \frac{N}{\lceil N/m \rceil \cdot H + N} \tag{9}$$

For the reassembly phase, the time complexity comparison is shown in Table III.

TABLE III
REASSEMBLY TIME COMPLEXITY COMPARISON

Approach	**Time**	**Space**
Naive (Sort + Scan)	$O(k \log k)$	$O(N)$
RFC 815 (Hole List)	$O(k^2)$	$O(N)$
RFC 815 (Interval Tree)	$O(k \log k)$	$O(N)$

Note: k = number of fragments, N = total payload size.

The RFC 815 hole descriptor approach [8] with interval tree optimization achieves the same asymptotic complexity as naive sorting but provides superior practical performance by avoid-

ing full payload movement and enabling early termination when the hole list becomes empty.

IV. Implementation and Experiment Design

A. Network Simulator Overview

The experiments in this paper are conducted using the MAGI System, a custom-built C++ network simulator that implements the OSI model from Layer 2 to Layer 7. The simulator supports Ethernet, ARP, IPv4, ICMP, TCP, UDP, HTTP, DNS, and DHCP protocols. For the purpose of this paper, only the Layer 3 components are relevant:

- `IPv4Packet` class with the `fragment()` method that implements the Divide phase
- `Host` class with a reassembly buffer that implements the Combine phase
- `Router` class with forwarding and re-fragmentation capabilities
- `Link` class with configurable MTU and propagation delay

The simulator is open-source and its source code can be accessed in the Appendix.

B. Test Topology Design

The experiment uses a single multi-hop topology designed to observe both initial fragmentation and recursive re-fragmentation. The topology consists of two hosts (H1 and H2) connected through one router (R1), as illustrated in Fig. 5.

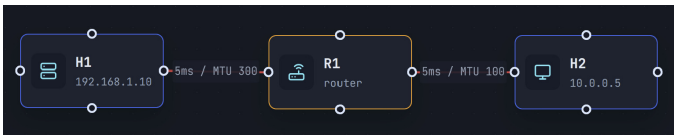


Fig. 5. Test topology: H1 sends a packet to H2 through router R1. The link H1–R1 has MTU 300 bytes, and the link R1–H2 has MTU 100 bytes. Both links have a propagation delay of 5 ms.

Host H1 is configured with IP address 192.168.1.10/24 and default gateway 192.168.1.1. Router R1 has two interfaces: port 1 at 192.168.1.1/24 (facing H1) and port 2 at 10.0.0.1/8 (facing H2). Host H2 is configured with IP address 10.0.0.5/8 and default gateway 10.0.0.1. The link between H1 and R1 has an MTU of 300 bytes, while the link between R1 and H2 has a more restrictive MTU of 100 bytes. Both links have a propagation delay of 5 ms.

This topology is specifically chosen to demonstrate recursive fragmentation. When H1 sends a packet with a payload of 508 bytes (total 528 bytes including the 20-byte IPv4 header), the packet exceeds the MTU of the first link (300 bytes) and must be fragmented at H1. The resulting fragments then arrive at R1, where the first fragment (280 bytes of payload) still exceeds the MTU of the second link (100 bytes) and must be re-fragmented at the router. This creates a two-level D&C recursion tree.

C. Experiment Scenarios

A single experiment scenario is conducted using the topology described above. H1 sends an IPv4 packet with a payload of 508 bytes to H2. The following metrics are observed and recorded:

1) Divide phase at H1: The number of fragments produced, their sizes, offsets, and MF flags. The maximum payload per fragment at H1 is $m_1 = \lfloor (300 - 20)/8 \rfloor \times 8 = 280$ bytes, yielding $k_1 = \lceil 508/280 \rceil = 2$ fragments.
2) Recursive Divide phase at R1: The first fragment (280 bytes payload) exceeds the MTU of the R1–H2 link (100 bytes), triggering re-fragmentation. The maximum payload per sub-fragment is $m_2 = \lfloor (100-20)/8 \rfloor \times 8 = 80$ bytes. The first fragment produces $\lceil 280/80 \rceil = 4$ sub-fragments, and the second fragment (228 bytes payload) produces $\lceil 228/80 \rceil = 3$ sub-fragments, for a total of 7 final fragments.
3) Combine phase at H2: Verification that all 7 fragments are received, the contiguity check passes (no gaps), and the reassembled payload matches the original 508 bytes.
4) Overhead and efficiency: The total header overhead and transmission efficiency η are computed and compared against the theoretical values from Table II.

The CLI output from the simulator is captured at each stage to provide evidence of the D&C phases in action.

V. Results and Discussion

A. Fragmentation Process Analysis

The first phase observed is the Divide phase at the source host H1. When H1 attempts to send a 508-byte payload (528 bytes total with the 20-byte IPv4 header) toward H2, the outgoing link MTU of 300 bytes forces fragmentation. The CLI output from the simulator is shown in Fig. 6.

```
Magi>  H1 ping 10.0.0.5 500
[H1] Memulai ping ke 10.0.0.5 payload=500bytes
[H1] Target berbeda subnet, ARP diarahkan ke gateway 192.168.1.1

[H1] +=======================================================+
[H1] |           DIVIDE PHASE: IPv4 Fragmentation            |
[H1] +=======================================================+
[H1] | Original packet: 528 bytes (header=20 + payload=508)
[H1] | Outgoing link MTU: 300 bytes
[H1] | Max payload/fragment: floor((300 - 20) / 8) x 8 = 280 bytes
[H1] | Number of fragments: ceil(508 / 280) = 2
[H1] |-------------------------------------------------------|
[H1] | Fragment #1: offset=0, size=300 bytes (h=20+p=280), MF=1
[H1] | Fragment #2: offset=280, size=248 bytes (h=20+p=228), MF=0
[H1] |-------------------------------------------------------|
[H1] | Total overhead: 2 x 20 = 40 bytes extra headers
[H1] +=======================================================+
```

Fig. 6. CLI output of the DIVIDE PHASE at H1. The 508-byte payload is split into 2 fragments with max payload $m = \lfloor (300 - 20)/8 \rfloor \times 8 = 280$ bytes.

The output confirms the theoretical formula. With $N = 508$ and $m = 280$:

$$k = \left\lceil \frac{508}{280} \right\rceil = 2 \tag{10}$$

Fragment #1 carries 280 bytes of payload at offset 0 with MF = 1, and Fragment #2 carries the remaining 228 bytes at offset 280 with MF = 0. The total header overhead is $2 \times 20 = 40$ bytes, yielding a transmission efficiency of $\eta = 508/(508 + 40) = 92.7\%$, which matches the theoretical value in Table II for MTU = 300.

This output directly maps to the Divide phase of D&C: the original problem (508-byte payload) is decomposed into 2 subproblems (fragments), each with its own header metadata, ready for independent transmission.

B. Conquer Phase: Independent Fragment Transmission

After the Divide phase produces the fragments, each fragment enters the Conquer phase, where it is transmitted independently through the network. The CLI output of the Conquer phase at H1 is shown in Fig. 7.

```
[H1] +=====================================================+
[H1] |         CONQUER PHASE: Independent Transmission     |
[H1] +=====================================================+
[H1] | Queuing Fragment #1 (pending ARP resolution)
[H1] | Queuing Fragment #2 (pending ARP resolution)
[H1] | 2 fragments queued, awaiting ARP Reply
[H1] +=====================================================+
```

Fig. 7. CLI output of the CONQUER PHASE at H1. Both fragments are queued independently and dispatched after ARP resolution.

The output shows that both fragments are queued separately, each pending ARP resolution for the next-hop gateway (192.168.1.1). Once the ARP reply is received, both fragments are dispatched independently. This behavior directly mirrors the Conquer phase of D&C: each subproblem (fragment) is solved independently, with no dependency on the other fragments. The connectionless nature of IP means that each fragment could theoretically take a different route, experience different delays, or arrive in a different order at the destination.

The Conquer phase also occurs at H2 during the return path (ICMP Echo Reply). Since the R1–H2 link has MTU = 100 bytes, H2 must fragment its 508-byte reply into 7 fragments. The CLI output is shown in Fig. 8.

```
[H2] +=====================================================+
[H2] |         CONQUER PHASE: Independent Transmission     |
[H2] +=====================================================+
[H2] | Sending Fragment #1 -> next-hop 10.0.0.1
[H2] | Sending Fragment #2 -> next-hop 10.0.0.1
[H2] | Sending Fragment #3 -> next-hop 10.0.0.1
[H2] | Sending Fragment #4 -> next-hop 10.0.0.1
[H2] | Sending Fragment #5 -> next-hop 10.0.0.1
[H2] | Sending Fragment #6 -> next-hop 10.0.0.1
[H2] | Sending Fragment #7 -> next-hop 10.0.0.1
[H2] | All 7 fragments dispatched independently
[H2] +=====================================================+
```

Fig. 8. CLI output of the CONQUER PHASE at H2 (return path). All 7 fragments are dispatched independently toward the next-hop router.

This output confirms that all 7 fragments are dispatched independently, each routed toward the next-hop 10.0.0.1 (R1). The Conquer phase at H2 produces more fragments than at H1 because the MTU of the R1–H2 link (100 bytes) is more restrictive than the H1–R1 link (300 bytes). Nevertheless, the D&C structure ensures that each fragment is a self-contained, independently routable IP packet, and the Combine phase at H1 will correctly reassemble all 7 fragments back into the original 508-byte payload.

C. Reassembly Correctness Verification

The Combine phase occurs at the destination host H2, where all fragments are collected and reassembled. The CLI output is shown in Fig. 9.

```
[R1] +=====================================================+
[R1] |     DIVIDE PHASE: Router Re-Fragmentation            |
[R1] +=====================================================+
[R1] | Incoming packet to 10.0.0.5 exceeds MTU 100
[R1] | Re-fragmenting into 4 fragments
[R1] |-----------------------------------------------------|
[R1] | Fragment #1: offset=0, size=100 bytes (h=20+p=80), MF=1
[R1] | Fragment #2: offset=80, size=100 bytes (h=20+p=80), MF=1
[R1] | Fragment #3: offset=160, size=100 bytes (h=20+p=80), MF=1
[R1] | Fragment #4: offset=240, size=60 bytes (h=20+p=40), MF=1
[R1] +=====================================================+

... (skip the ARP and fragment forwarding lines in between) ...

[R1] +=====================================================+
[R1] |     DIVIDE PHASE: Router Re-Fragmentation            |
[R1] +=====================================================+
[R1] | Incoming packet to 10.0.0.5 exceeds MTU 100
[R1] | Re-fragmenting into 3 fragments
[R1] |-----------------------------------------------------|
[R1] | Fragment #1: offset=280, size=100 bytes (h=20+p=80), MF=1
[R1] | Fragment #2: offset=360, size=100 bytes (h=20+p=80), MF=1
[R1] | Fragment #3: offset=440, size=88 bytes (h=20+p=68), MF=0
[R1] +=====================================================+
```

Fig. 9. CLI output of the COMBINE PHASE at H2. All 7 fragments are received, the contiguity check passes, and the reassembled payload is 508 bytes.

The output shows that all 7 fragments arrive at H2 with offsets 0, 80, 160, 240, 280, 360, and 440. The buffer fills contiguously from byte 0 to byte 507, confirmed by the progress bar `[####...####] 508/508 bytes`. The contiguity check reports `PASSED (no gaps detected)`, and the reassembled payload size of 508 bytes exactly matches the original payload sent by H1.

This verifies the correctness of the Combine phase: the D&C paradigm guarantees that the reassembled solution is bitwise identical to the original problem. The offset-based buffer management correctly orders fragments regardless of arrival sequence, and the MF = 0 flag on the last fragment (offset 440, size 68) signals the completion of the datagram.

D. Multi-Hop Re-fragmentation Results

The most critical observation is the recursive nature of fragmentation at router R1. After H1 produces 2 fragments, the first fragment (280 bytes payload, offset 0) arrives at R1 and must be forwarded toward H2 via a link with MTU = 100 bytes. Since $280 + 20 > 100$, R1 must re-fragment this fragment. The CLI output is shown in Fig. 10.

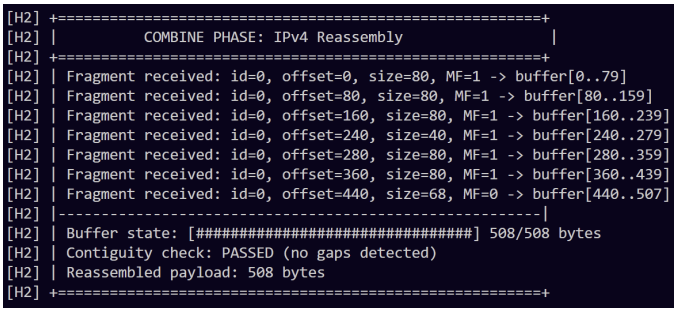


Fig. 10. CLI output of the recursive DIVIDE PHASE at R1. The first fragment (280 bytes payload) is re-fragmented into 4 sub-fragments, and the second fragment (228 bytes payload) is re-fragmented into 3 sub-fragments, producing 7 final fragments.

The output shows two separate re-fragmentation events at R1:

First fragment (payload = 280 bytes, offset = 0): With $m_2 = \lfloor (100-20)/8 \rfloor \times 8 = 80$ bytes, this fragment is divided into $\lceil 280/80 \rceil = 4$ sub-fragments at offsets 0, 80, 160, and 240.

Second fragment (payload = 228 bytes, offset = 280): This fragment is divided into $\lceil 228/80 \rceil = 3$ sub-fragments at offsets 280, 360, and 440.

The total number of final fragments reaching H2 is $4+3=7$, which is identical to the number of fragments that would have been produced if H1 had directly used the most restrictive MTU of 100 bytes ($\lceil 508/80 \rceil = 7$). This demonstrates that recursive re-fragmentation at intermediate routers is mathematically equivalent to a single-level fragmentation at the most constrained link, validating the recursive structure of the D&C paradigm.

The recursion tree from Fig. 2 is thus confirmed experimentally: the root problem (508 bytes) is divided at H1 into 2 subproblems, each of which is further divided at R1 into 4 and 3 leaf subproblems respectively, yielding 7 base cases that can traverse the narrowest link.

E. Efficiency and Overhead Analysis

The fragmentation process introduces additional header overhead at each level of the recursion tree. Table IV summarizes the overhead at each stage of the experiment.

TABLE IV
OVERHEAD AND EFFICIENCY AT EACH STAGE

Stage	Fragments	Overhead	η
H1 Divide (MTU = 300)	2	40 B	92.7%
R1 Re-fragment (MTU = 100)	7	140 B	78.4%

Note: Overhead = $k \times 20$ bytes (IPv4 header per fragment).
$\eta = N/(N + \text{overhead})$.

At the first level (H1), the overhead is modest: only 40 extra bytes for 2 fragments. However, after recursive re-fragmentation at R1, the total overhead grows to $7 \times 20 = 140$ bytes, reducing efficiency to 78.4%. This matches the theoretical efficiency for MTU = 100 in Table II.

The key insight is that the D&C structure guarantees correctness regardless of the number of recursion levels, but each level of division adds header overhead. The trade-off is deterministic and predictable: smaller MTU values produce more fragments and lower efficiency, but the original payload is always faithfully reconstructed at the destination as long as all fragments arrive. The simulator output confirms that the reassembled payload at H2 (508 bytes) is identical to the original payload sent by H1, validating that the D&C approach to IPv4 fragmentation ensures reliable data transmission across heterogeneous network links at the cost of bounded, quantifiable overhead.

VI. CONCLUSION

This paper has formally demonstrated that the IPv4 fragmentation and reassembly mechanism is a direct and rigorous application of the Divide-and-Conquer (D&C) algorithmic paradigm. The three phases of D&C map precisely to the three stages of packet transmission across heterogeneous network links.

In the Divide phase, a datagram whose size exceeds the MTU of the outgoing link is decomposed into smaller fragments. The maximum payload per fragment is determined by the 8-byte alignment constraint $m = \lfloor (M-H)/8 \rfloor \times 8$, and the total number of fragments is $k = \lceil N/m \rceil$. Each fragment becomes an independent subproblem with its own IPv4 header, Fragment Offset, and MF flag. When a fragment still exceeds the MTU of a downstream link, the router recursively applies the same division, producing a multi-level D&C recursion tree.

In the Conquer phase, each fragment is transmitted independently through the network. The connectionless nature of IP means that fragments may take different routes, experience different delays, and arrive in arbitrary order. Each fragment is a self-contained, independently routable IP packet, and no coordination between fragments is required during transmission.

In the Combine phase, the destination host collects all fragments using the Identification field as the grouping key and the Fragment Offset field as the ordering mechanism. A reassembly buffer accumulates the fragments, and a contiguity check verifies that no data gaps exist. When all bytes from offset zero to the total payload length are received, the original datagram is reconstructed and delivered to the transport layer.

The experimental results from the MAGI System network simulator confirm the correctness of this D&C mapping. In the test topology (H1–R1–H2 with MTU values of 300 and 100 bytes), a 508-byte payload was fragmented into 2 fragments at H1, recursively re-fragmented into 7 sub-fragments at R1, and faithfully reassembled at H2 with a contiguity check that reported `PASSED`. The reassembled payload of 508 bytes was bitwise identical to the original, validating that the D&C structure guarantees correctness regardless of the number of recursion levels.

The overhead introduced by fragmentation is bounded and predictable. Each additional fragment adds a fixed 20-byte IPv4 header, and the transmission efficiency $\eta = N/(\lceil N/m \rceil \cdot$

$H + N$) decreases monotonically as the MTU decreases. For the experiment, the efficiency dropped from 92.7% at MTU = 300 to 78.4% at MTU = 100, but the original payload was always faithfully reconstructed.

Future work could explore optimization strategies to minimize fragmentation overhead. Path MTU Discovery (PMTUD) allows the sender to determine the smallest MTU along the path before transmission, potentially avoiding fragmentation entirely. Additionally, comparing the D&C approach with other algorithm paradigms for similar problems, such as greedy or dynamic programming approaches to packet scheduling, could provide further insights into the trade-offs between correctness, efficiency, and complexity in network protocol design.

APPENDIX

The program source code can be accessed here and this document repositories can be accessed here.

ACKNOWLEDGMENT

First of all, the author would like to express gratitude and thanks to God Almighty for His blessings and grace, which have given the author the opportunity to write and complete this paper as well and as thoroughly as possible. The author would also like to thank his family for their continuous support and encouragement during both difficult and happy times. The author would also like to express his deepest gratitude to the lecturer of IF2211 K-02, Ir. Rila Mandala, M.Eng., Ph.D., who has always shared his knowledge and taught the author many things that will prepare him for the future. In addition, I would like to thank my classmates in K-02 who have provided various dynamics both inside and outside the classroom. I would also like to thank all of my Artificial Intelligence friends who have helped me brainstorm ideas and paraphrase my poor English. Lastly, thank you to myself for continuing to learn and striving to complete this paper. I truly enjoyed this course so much that my hair kept growing, which signifies the wealth of knowledge I gained from this course.

If you are working on something that you really care about, you don't have to be pushed. The vision pulls you

Steve Jobs

REFERENCES

[1] J. Postel, "Internet Protocol," *RFC 791*, Internet Engineering Task Force, Sep. 1981. [Online]. Available: https://www.rfc-editor.org/rfc/rfc791

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA: MIT Press, 2009.

[3] R. Munir, "Algoritma Divide and Conquer (Bagian 1)," *Informatika STEI ITB*, Des. 19, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/07-Algoritma-Divide-and-Conquer-(2026)-Bagian1.pdf

[4] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*, 8th ed. Hoboken, NJ: Pearson, 2021.

[5] W. R. Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, 2nd ed. Boston, MA: Addison-Wesley, 2011.

[6] D. E. Comer, *Internetworking with TCP/IP Vol.1: Principles, Protocols, and Architecture*, 6th ed. Hoboken, NJ: Pearson, 2021.

[7] R. Munir, "Algoritma Divide and Conquer (Bagian 2)," *Informatika STEI ITB*, Des. 19, 2025. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/08-Algoritma-Divide-and-Conquer-(2026)-Bagian2.pdf

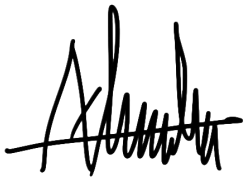
[8] D. D. Clark and J. Reed, "A Fragmentation Control Protocol for Internet Datagrams," *RFC 815*, Internet Engineering Task Force, Jun. 1982. [Online]. Available: https://www.rfc-editor.org/rfc/rfc815

[9] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*, 6th ed. Hoboken, NJ: Pearson, 2021.

STATEMENT OF ORIGINALITY

I hereby declare that this paper is my own writing, not an adaptation, or translation of someone else's paper, and not plagiarized.

Bandung, 19 June 2026



Reinhard Alfonzo Hutabarat, 13524056