

# Blockchain Forensics: Investigating the Avalanche C-Chain Anomaly of April 2025

## Abstract

This paper presents a comprehensive blockchain forensic investigation into an anomalous increase in active addresses on the Avalanche C-Chain between April 8-15, 2025. Using on-chain data analysis through the Routescan API, we employed a systematic, multi-phase approach to identify, analyze, and determine the cause of this unusual network activity. Our investigation revealed a direct correlation between the anomaly and the implementation of the "Octane Upgrade" - a protocol change that dramatically reduced transaction fees, enabling economic viability for transactions with multiple internal operations. This case study demonstrates the effectiveness of structured on-chain forensic methodologies in blockchain anomaly detection and attribution, providing valuable insights for blockchain analysts, researchers, and network participants.

---

## 1. Introduction

Blockchain networks generate vast amounts of on-chain data that can be analyzed to understand network behavior, detect anomalies, and identify the root causes of unusual activities. On April 8, 2025, the Avalanche C-Chain experienced a sudden and dramatic increase in active addresses without a proportional increase in transaction count, creating an anomalous pattern that warranted investigation.

This paper details the methodology and findings of a forensic investigation into this anomaly. Our approach combined data acquisition through the Routescan API, systematic data analysis, and correlation with off-chain evidence to develop and test hypotheses regarding the cause of the unusual network activity.

---

## 2. Methodology

### 2.1 Data Acquisition Framework

The investigation relied on data from the Routescan API, which provides comprehensive access to Avalanche C-Chain data. A Python-based data acquisition framework was developed with the following components:

- **API Connection Management:** Configured with environment variables for base URL, chain ID, and authentication

```
# Configure global variables
BASE_URL = os.getenv("ROUTESCAN_BASE_URL") or
"https://api.routescan.io/v2/network/mainnet/evm"
CHAIN_ID = os.getenv("AVALANCHE_CHAIN_ID") or "43114" # Avalanche C-Chain
API_KEY = os.getenv("ROUTESCAN_API_KEY")
START_DATE = os.getenv("START_DATE") or "2025-04-01" # Format: "YYYY-MM-DD"
END_DATE = os.getenv("END_DATE") or "2025-04-16" # Format: "YYYY-MM-DD"
ANOMALY_DATE = "2025-04-08" # Date when the anomaly started
```

### 2.2 Data Collection Functions

Multiple specialized functions were implemented to fetch different types of on-chain data:

**Daily and Hourly Active Addresses:** Functions to retrieve unique active addresses over specified time periods

```
def get_daily_active_addresses():
    """Fetch the number of unique active addresses per day"""
    endpoint = f"{BASE_URL}/{CHAIN_ID}/aggregations/addresses"
    params = {
        "dateFrom": START_DATE,
        "dateTo": END_DATE,
        "unit": "day"
    }
    if API_KEY:
        params["apikey"] = API_KEY

    response = requests.get(endpoint, params=params)
    if response.status_code == 200:
        data = response.json()
        # Save raw data
```

```

    with open(f"{DATA_DIR}/daily_active_addresses.json", "w") as f:
        json.dump(data, f, indent=2)
    return data
else:
    print(f"Error fetching active addresses: {response.status_code}")
    return None

```

## Transaction Volume: Functions to analyze transaction counts and characteristics

```

def get_transactions_volume():
    """Fetch the volume of transactions per day"""
    endpoint = f"{BASE_URL}/{CHAIN_ID}/aggregations/txs"
    params = {
        "dateFrom": START_DATE,
        "dateTo": END_DATE,
        "unit": "day"
    }
    if API_KEY:
        params["apikey"] = API_KEY

    response = requests.get(endpoint, params=params)
    if response.status_code == 200:
        data = response.json()
        with open(f"{DATA_DIR}/transactions_volume.json", "w") as f:
            json.dump(data, f, indent=2)
        return data
    else:
        print(f"Error fetching transaction volume: {response.status_code}")
        return None

```

## ERC20 Transfers: Functions to analyze token transfer patterns

```

def get_erc20_transfers():
    """Fetch the number of ERC20 transfers per day"""
    endpoint = f"{BASE_URL}/{CHAIN_ID}/aggregations/erc20-transfers"
    params = {

```

```
    "dateFrom": START_DATE,
    "dateTo": END_DATE,
    "unit": "day"
}
if API_KEY:
    params["apikey"] = API_KEY
```

## Detailed Transaction Analysis: Functions to retrieve and analyze specific transactions

```
def get_transactions_on_date(date_str, limit=3000):
    """Fetch transactions for a specific date"""
    # Format: YYYY-MM-DD
    start_time = f"{date_str}T00:00:00Z"
    end_time = f"{date_str}T23:59:59Z"

    endpoint = f"{BASE_URL}/{CHAIN_ID}/transactions"
    params = {
        "timestampFrom": start_time,
        "timestampTo": end_time,
        "limit": limit,
        "sort": "desc"
    }
    if API_KEY:
        params["apikey"] = API_KEY

    response = requests.get(endpoint, params=params)
    if response.status_code == 200:
        data = response.json()
        with open(f"{DATA_DIR}/transactions_{date_str}.json", "w") as f:
            json.dump(data, f, indent=2)
        return data
    else:
        print(f"Error fetching transactions for {date_str}: {response.status_code}")
        return None
```

## Internal Operations: Functions to analyze internal contract calls within transactions

```
def get_internal_operations_on_date(date_str, limit=3000):
    """Fetch internal operations for a specific date"""
    start_time = f"{date_str}T00:00:00Z"
    end_time = f"{date_str}T23:59:59Z"

    endpoint = f"{BASE_URL}/{CHAIN_ID}/internal-operations"
    params = {
        "timestampFrom": start_time,
        "timestampTo": end_time,
        "limit": limit,
        "sort": "desc"
    }
    if API_KEY:
        params["apikey"] = API_KEY

    response = requests.get(endpoint, params=params)
    if response.status_code == 200:
        data = response.json()
        with open(f"{DATA_DIR}/internal_operations_{date_str}.json", "w") as f:
            json.dump(data, f, indent=2)
        return data
    else:
        print(f"Error fetching internal operations for {date_str}:
{response.status_code}")
    return None
```

## Contract Analysis: Functions to analyze specific contracts and their activities

```
def get_contract_transactions(contract_address, date_str=None, limit=3000):
    """Fetch transactions for a specific contract, optionally filtered by date"""
    endpoint = f"{BASE_URL}/{CHAIN_ID}/address/{contract_address}/transactions"
    params = {
        "limit": limit,
        "sort": "desc"
    }
}
```

```

if date_str:
    # Add date filter if specified
    start_time = f"{date_str}T00:00:00Z"
    end_time = f"{date_str}T23:59:59Z"
    params["timestampFrom"] = start_time
    params["timestampTo"] = end_time

if API_KEY:
    params["apikey"] = API_KEY

response = requests.get(endpoint, params=params)
if response.status_code == 200:
    data = response.json()
    filename = f"contract_{contract_address[-8:]}_txs"
    if date_str:
        filename += f"_{date_str}"
    filename += ".json"

    with open(f"{DATA_DIR}/{filename}", "w") as f:
        json.dump(data, f, indent=2)
    return data
else:
    print(f"Error fetching transactions for contract {contract_address}:
{response.status_code}")
    return None

```

---

## 2.3 Analysis Framework

The investigation followed a structured, multi-phase approach inspired by detective methodologies:

**Phase 1 - Identify and Confirm Anomaly:** Analyze daily active addresses to verify the anomaly's existence and characteristics

```

def sherlock_analyze_daily_activity():
    """PHASE 1: Identify and confirm the anomaly in daily active addresses"""
    print("\n🔍 PHASE 1: THE CURIOUS CASE OF INCREASING ACTIVE ADDRESSES")
    print("=" * 80)

    # Fetch daily active addresses data
    print("Fetching daily active addresses data...")
    active_addresses_data = get_daily_active_addresses()

    if not active_addresses_data:
        print("❌ Failed to fetch active addresses data. Investigation cannot proceed.")
        return None

    # Prepare data frame - do this only once
    df_active = pd.DataFrame(active_addresses_data, columns=["date", "count"])
    df_active["date"] = pd.to_datetime(df_active["date"])
    df_active["date_str"] = df_active["date"].dt.strftime('%Y-%m-%d')
    df_active["count"] = pd.to_numeric(df_active["count"], errors='coerce')

    # Calculate daily change percentage
    df_active['daily_change_pct'] = df_active['count'].pct_change() * 100

    # Find high activity days (>10% increase)
    high_activity_days = df_active[df_active['daily_change_pct'] >
10].sort_values('date')

    # Identify the anomaly start
    if not high_activity_days.empty:
        anomaly_start = high_activity_days.iloc[0]
        anomaly_start_date = anomaly_start['date_str']

    # Create visualization
    plot_path = plot_active_addresses(df_active)

    # Print findings
    print("\n🔍 Initial Observations:")
    print(f" - Analysis starting on {anomaly_start_date}")
    print(f" - Active addresses increase from {int(anomaly_start['count']):,} to "
+
        f"{int(df_active[df_active['date_str'] >
anomaly_start_date]['count'].max()):,}")

```

```

print(f" - Peak increase of {anomaly_start['daily_change_pct']:.2f}%
day-over-day")

print("\n📊 Daily Active Addresses Statistics:")
print(f"{'Date':<12} {'Active Addresses':>15} {'Change %':>10}")
print("-" * 40)

prev_count = None
for idx, row in df_active.sort_values('date').iterrows():
    date_str = row['date_str']
    count = row['count']

    # Calculate change percentage
    if prev_count is not None:
        change_pct = ((count - prev_count) / prev_count) * 100
        change_str = f"{change_pct:+.2f}%"
    else:
        change_str = "N/A"

    # Mark anomaly days
    marker = "🔴" if date_str >= anomaly_start_date else ""

    # Print formatted row
    print(f"{date_str:<12} {int(count):>15,} {change_str:>10} {marker}")

    # Update previous value for next iteration
    prev_count = count

if plot_path:
    print(f"\n📄 Visualization saved to: {plot_path}")

# Return data for next phase
return {
    "active_addresses_df": df_active,
    "anomaly_start_date": anomaly_start_date
}
else:
    print("❌ No significant anomaly detected in active addresses.")
    return None

```

## Phase 2 - Transaction Correlation: Analyze correlation between active addresses and transaction metrics

```
def sherlock_analyze_transactions_correlation(phase1_data):
    """PHASE 2: Analyze correlation with transactions and related metrics"""
    if not phase1_data:
        print("❌ Cannot proceed without Phase 1 data.")
        return None

    anomaly_start_date = phase1_data["anomaly_start_date"]
    active_addresses_df = phase1_data["active_addresses_df"]

    print("\n🔍 PHASE 2: FOLLOWING THE TRANSACTION TRAIL")
    print("=" * 80)
    print(f"Investigating correlation between active addresses and transactions...")

    # Fetch transaction volume data
    print("Fetching transaction volume data...")
    transactions_data = get_transactions_volume()

    # Fetch ERC20 transfer data
    print("Fetching ERC20 transfer data...")
    erc20_data = get_erc20_transfers()

    if not transactions_data or not erc20_data:
        print("❌ Failed to fetch transaction or ERC20 data.")
        return None

    # Prepare data frames
    txs_df = pd.DataFrame(transactions_data, columns=["date", "count"])
    txs_df["date"] = pd.to_datetime(txs_df["date"])
    txs_df["date_str"] = txs_df["date"].dt.strftime('%Y-%m-%d')
    # Convert count to numeric
    txs_df["count"] = pd.to_numeric(txs_df["count"], errors='coerce')

    erc20_df = pd.DataFrame(erc20_data, columns=["date", "count"])
    erc20_df["date"] = pd.to_datetime(erc20_df["date"])
    erc20_df["date_str"] = erc20_df["date"].dt.strftime('%Y-%m-%d')
    # Convert count to numeric
    erc20_df["count"] = pd.to_numeric(erc20_df["count"], errors='coerce')
```

```

# Calculate daily changes
txs_df['daily_change_pct'] = txs_df['count'].pct_change() * 100
erc20_df['daily_change_pct'] = erc20_df['count'].pct_change() * 100

# Create combined visualization
plot_path = plot_combined_activity(active_addresses_df, txs_df, erc20_df)

# Inform the user about the plot if it was successfully created
if plot_path:
    print(f"\n📄 Combined activity visualization saved to: {plot_path}")

# Analyze discrepancy between active addresses and transactions
# Convert the active_addresses_df back to the format needed for
analyze_activity_discrepancy
active_addresses_data_list = active_addresses_df[["date",
"count"]].to_dict('records')

discrepancy_data = analyze_activity_discrepancy(active_addresses_data_list,
transactions_data)

if discrepancy_data:
    # Rest of the function remains the same
    discrepancy_plot = plot_address_tx_ratio(discrepancy_data)

    # Print findings
    print("\n🔍 Key Observations:")
    print(f" - Normal ratio of addresses to transactions:
{discrepancy_data['avg_addr_per_tx_normal']:.2f}")
    print(f" - During anomaly, this ratio increased dramatically")

    print("\n📊 Address-to-Transaction Ratio Analysis:")
    print(f"{'Date':<12} {'Active Addresses':>15} {'Transactions':>15}
{'Ratio':>10} {'Times Normal':>12}")
    print("-" * 70)

    # Ordina i giorni in base alla data in ordine crescente
    sorted_days = sorted(discrepancy_data["anomalous_days"], key=lambda x:
x["date_str"])

    for day in sorted_days:
        date_str = day["date_str"]
        addresses = day["count_addresses"]

```

```

    txs = day["count_txs"]
    ratio = day["addr_per_tx"]
    times_normal = ratio / discrepancy_data["avg_addr_per_tx_normal"]

    # Mark anomaly days
    marker = "🔴" if date_str >= anomaly_start_date else ""

    # Print formatted row
    print(f"{date_str:<12} {int(addresses):>15,} {int(txs):>15,} {ratio:>10.2f}
{times_normal:>12.1f}x {marker}")

    if discrepancy_plot:
        print(f"\n✅ Ratio visualization saved to: {discrepancy_plot}")

    # Return data for next phase
    return {
        **phase1_data,
        "transactions_df": txs_df,
        "erc20_df": erc20_df,
        "discrepancy_data": discrepancy_data
    }
else:
    print("❌ Failed to analyze address-transaction discrepancy.")
    return phase1_data # Return Phase 1 data to allow continuation

```

**Phase 3 - Contract Interactions:** Analyze specific contract interactions during the anomaly period

```

def sherlock_analyze_contract_interactions(phase2_data):
    """PHASE 3: Analyze contract interactions during the anomaly period"""
    if not phase2_data:
        print("❌ Cannot proceed without Phase 2 data.")
        return None

    anomaly_start_date = phase2_data["anomaly_start_date"]

    print("\n🔍 PHASE 3: THE CONTRACT CONUNDRUM")
    print("=" * 80)

```

```

print(f"Investigating contract interactions on {anomaly_start_date}...")

# Fetch transactions on anomaly start date
print(f"Fetching transactions for {anomaly_start_date}...")
anomaly_txs = get_transactions_on_date(anomaly_start_date)

# Fetch internal operations on anomaly start date
print(f"Fetching internal operations for {anomaly_start_date}...")
anomaly_internals = get_internal_operations_on_date(anomaly_start_date)

# Fetch ERC20 transfers on anomaly start date
print(f"Fetching ERC20 transfers for {anomaly_start_date}...")
anomaly_erc20 = get_erc20_transfers_on_date(anomaly_start_date)

if not anomaly_txs or not anomaly_internals:
    print("❌ Failed to fetch transaction or internal operation data.")
    return phase2_data

# Analyze popular contracts
contracts_data = analyze_popular_contracts(anomaly_txs)

# Analyze internal operations
internals_data = analyze_internal_operations_details(anomaly_internals)

# Analyze batch transactions
batch_data = analyze_batch_transactions(anomaly_txs)

# Analyze high activity contracts
high_activity = find_high_activity_contracts(anomaly_txs)

if contracts_data and internals_data:
    # Plot internal operation types
    op_types_plot = plot_operation_types(internals_data)

    # Print findings
    print("\n🔍 Key Contract Observations:")

    print("\n📊 Most Popular Contracts:")
    print(f"{'Rank':<5} {'Contract Address':<42} {'Interactions':>12} {'Main
Method':<40}")
    print("-" * 105)

```

```

for i, (addr, count) in enumerate(contracts_data["popular_contracts"][:5], 1):
    methods_info = ""
    if addr in contracts_data["contract_methods"]:
        top_method = max(contracts_data["contract_methods"][addr].items(),
key=lambda x: x[1]) if contracts_data["contract_methods"][addr] else None
        if top_method:
            methods_info = top_method[0]

    print(f"{i:<5} {addr:<42} {count:>12} {methods_info:<40}")

print("\n📊 Internal Operation Types:")
for op_type, count in internals_data["type_counts"].items():
    print(f" - {op_type}: {count:,} operations")

if batch_data:
    print("\n📊 Batch Transaction Analysis:")
    print(f" - Total batch transactions found:
{batch_data['total_batch_txs']}")
    print(f" - Average batch size: {batch_data['avg_batch_size']:.1f}
operations per batch")
    print(f" - Estimated total operations in batches:
{batch_data['estimated_operations']:,}")

    if batch_data['batch_txs']:
        print("\n Sample batch transactions:")
        for i, tx in enumerate(batch_data['batch_txs'][:3], 1):
            method = tx.get('method', 'Unknown')
            from_addr = tx.get('from', 'Unknown')[:10] + "..."
            to_addr = tx.get('to', 'Unknown')[:10] + "..."
            print(f" {i}. Method: {method} from {from_addr} to {to_addr}")

if high_activity:
    print("\n📊 Contracts Generating Most Internal Operations:")
    for i, (addr, count) in
enumerate(high_activity['top_internal_generators'][:5], 1):
        print(f" {i}. {addr}: {count:,} internal operations")

# Return key contract addresses for next phase
key_contracts = []

if contracts_data and contracts_data["popular_contracts"]:

```

```

        key_contracts = [addr for addr, _ in
contracts_data["popular_contracts"][:5]]

    if high_activity and high_activity['top_internal_generators']:
        for addr, _ in high_activity['top_internal_generators'][:3]:
            if addr not in key_contracts:
                key_contracts.append(addr)

    return {
        **phase2_data,
        "key_contracts": key_contracts,
        "batch_data": batch_data
    }
else:
    print("❌ Failed to analyze contract interactions.")
    return phase2_data

```

#### Phase 4 - New Contracts: Analyze newly deployed contracts around the anomaly period

```

def sherlock_analyze_new_contracts(phase3_data):
    """PHASE 4: Analyze new contracts created during the anomaly period"""
    if not phase3_data:
        print("❌ Cannot proceed without Phase 3 data.")
        return None

    anomaly_start_date = phase3_data["anomaly_start_date"]

    print("\n🔍 PHASE 4: THE CASE OF NEW CONTRACTS")
    print("=" * 80)
    print(f"Investigating contracts created around the anomaly period...")

    # Fetch all new contracts
    print("Fetching new contracts data...")
    contracts_data = get_all_new_contracts()

    if not contracts_data or "items" not in contracts_data:
        print("❌ Failed to fetch new contracts data.")
        return phase3_data

```

```

contracts = contracts_data["items"]

print(f"\n🔍 New Contracts Analysis:")
print(f"  - Total new contracts found: {len(contracts):,}")

# Count contracts by day of creation
creation_dates = {}
creators = {}

for contract in contracts:
    if "createOperation" in contract and "timestamp" in
contract["createOperation"]:
        date_str = contract["createOperation"]["timestamp"][:10] # Just YYYY-MM-DD
        creation_dates[date_str] = creation_dates.get(date_str, 0) + 1

        if "from" in contract["createOperation"]:
            creator = contract["createOperation"]["from"]
            creators[creator] = creators.get(creator, 0) + 1

# Show temporal distribution
print(f"\n📊 Temporal Distribution of Contract Creation:")
for date, count in sorted(creation_dates.items()):
    marker = "🔴" if date >= anomaly_start_date else ""
    print(f"  - {date}: {count:,} contracts {marker}")

# Show top creators
print(f"\n📊 Top Contract Creators:")
for creator, count in sorted(creators.items(), key=lambda x: x[1],
reverse=True)[:3]:
    print(f"  - {creator}: {count:,} contracts")

# Check if this is a factory contract we already identified
is_key = ""
if phase3_data["key_contracts"] and creator in phase3_data["key_contracts"]:
    is_key = "★ KEY CONTRACT IDENTIFIED"

print(f"    {is_key}")

return {
    **phase3_data,
    "contract_creators": creators
}

```

```
}
```

## Phase 5 - Key Contract Analysis: Detailed analysis of identified key contracts

```
def sherlock_analyze_key_contracts(phase4_data):
    """PHASE 5: Detailed analysis of key contracts identified in previous phases"""
    if not phase4_data:
        print("❌ Cannot proceed without Phase 4 data.")
        return None

    key_contracts = phase4_data.get("key_contracts", [])

    if not key_contracts:
        print("❌ No key contracts identified to analyze.")
        return phase4_data

    print("\n🔍 PHASE 5: INTERROGATING THE KEY SUSPECTS")
    print("=" * 80)
    print(f"Performing detailed analysis of {len(key_contracts)} key contracts...")

    contract_insights = {}

    for contract in key_contracts:
        print(f"\nAnalyzing contract: {contract}")

        # Get transactions for this contract
        contract_txs = get_contract_transactions(contract)

        # Get contract ABI if available
        contract_abi = get_contract_abi(contract)

        # Analyze operations
        if contract_txs and "items" in contract_txs:
            operations = analyze_operation_types(contract_txs)

            contract_insights[contract] = {
                "transaction_count": len(contract_txs["items"]),
                "operations": operations,
```

```

        "has_abi": bool(contract_abi and contract_abi.get("status") == "1")
    }

    # Print findings
    print(f" - Transaction count: {len(contract_txs['items']):,}")

    if operations and "popular_methods" in operations:
        print(" - Most common methods:")
        for method, count in operations["popular_methods"][:3]:
            print(f"     • {method}: {count:,} calls")

    # Check for mining/minting functions
    has_mint = False
    has_batch = False
    has_claim = False

    if operations and "popular_methods" in operations:
        for method, _ in operations["popular_methods"]:
            if "mint" in method.lower():
                has_mint = True
            if "batch" in method.lower() or "bulk" in method.lower():
                has_batch = True
            if "claim" in method.lower():
                has_claim = True

    print(f" - Has mint functions: {'✅' if has_mint else '❌'}")
    print(f" - Has batch functions: {'✅' if has_batch else '❌'}")
    print(f" - Has claim functions: {'✅' if has_claim else '❌'}")

    # Extract available functions from ABI
    if contract_abi and contract_abi.get("status") == "1":
        try:
            abi_json = json.loads(contract_abi.get("result"))
            function_names = []

            for item in abi_json:
                if item.get("type") == "function" and item.get("name"):
                    function_names.append(item.get("name"))

            if function_names:
                print(" - Available functions from ABI:")

```

```

        relevant_functions = ["mint", "claim", "batch", "bulk",
"transfer", "multicall"]
        for func in relevant_functions:
            matches = [f for f in function_names if func.lower() in
f.lower()]

            if matches:
                print(f"        • Functions with '{func}': {'',
'.join(matches[:3])}")
            except Exception as e:
                print(f"        ✘ Error parsing ABI: {str(e)}")

    return {
        **phase4_data,
        "contract_insights": contract_insights
    }

```

## Phase 6 - Hypothesis Formation: Develop and test hypotheses based on collected evidence

```

def sherlock_form_hypothesis(phase5_data):
    """PHASE 6: Form and test hypotheses based on all collected evidence"""
    if not phase5_data:
        print("✘ Cannot proceed without Phase 5 data.")
        return None

    discrepancy_data = phase5_data.get("discrepancy_data", {})
    batch_data = phase5_data.get("batch_data", {})
    contract_insights = phase5_data.get("contract_insights", {})

    print("\n🔍 PHASE 6: FORMING THE HYPOTHESIS")
    print("=" * 80)

    # Evaluate evidence for different hypotheses
    hypothesis_scores = {
        "upgrade": 0, # Blockchain upgrade causing more internal operations
        "airdrop": 0, # Mass token airdrop
        "minting": 0, # New mining/minting activity
        "batch": 0, # Batch operations becoming popular
    }

```

```

}

# Evidence 1: Address-to-Transaction ratio anomaly
if discrepancy_data and "anomalous_days" in discrepancy_data:
    max_ratio_day = max(discrepancy_data["anomalous_days"], key=lambda x:
x["addr_per_tx"])
    max_ratio = max_ratio_day["addr_per_tx"]
    normal_ratio = discrepancy_data["avg_addr_per_tx_normal"]

    if max_ratio > normal_ratio * 5: # More than 5x normal
        print("\n Evidence: Extreme address-to-transaction ratio anomaly")
        print(f" - Normal: {normal_ratio:.2f} addresses per transaction")
        print(f" - Anomaly peak: {max_ratio:.2f} addresses per transaction
({max_ratio/normal_ratio:.1f}x normal)")

        # This strongly supports the upgrade or batch hypothesis
        hypothesis_scores["upgrade"] += 30
        hypothesis_scores["batch"] += 20

# Evidence 2: Presence of batch operations
if batch_data and batch_data.get("total_batch_txs", 0) > 0:
    print("\n Evidence: Presence of batch operations")
    print(f" - {batch_data['total_batch_txs']} batch transactions detected")

    if "batch_txs" in batch_data and batch_data["batch_txs"]:
        batch_methods = set()
        for tx in batch_data["batch_txs"]:
            if "method" in tx and tx["method"]:
                batch_methods.add(tx["method"])

        print(f" - Batch methods found: {' , '.join(list(batch_methods)[:3])}")

    # This supports the batch hypothesis
    hypothesis_scores["batch"] += 20

    # If many batch operations, supports upgrade hypothesis too
    if batch_data.get("total_batch_txs", 0) > 50:
        hypothesis_scores["upgrade"] += 15

# Evidence 3: Contract function analysis
mint_functions = False
batch_functions = False

```

```

claim_functions = False

for contract, insights in contract_insights.items():
    if "operations" in insights and insights["operations"]:
        operations = insights["operations"]

        if "popular_methods" in operations:
            for method, _ in operations["popular_methods"]:
                if "mint" in method.lower():
                    mint_functions = True
                if "batch" in method.lower() or "bulk" in method.lower():
                    batch_functions = True
                if "claim" in method.lower():
                    claim_functions = True

if mint_functions:
    print("\n🔍 Evidence: Presence of mint functions in key contracts")
    hypothesis_scores["minting"] += 25

if batch_functions:
    print("\n🔍 Evidence: Presence of batch functions in key contracts")
    hypothesis_scores["batch"] += 15
    hypothesis_scores["airdrop"] += 10

if claim_functions:
    print("\n🔍 Evidence: Presence of claim functions in key contracts")
    hypothesis_scores["airdrop"] += 15

# Determine the most likely hypothesis
top_hypothesis = max(hypothesis_scores.items(), key=lambda x: x[1])

print("\n📋 Hypothesis Evaluation:")
for hypothesis, score in sorted(hypothesis_scores.items(), key=lambda x: x[1],
reverse=True):
    print(f" - {hypothesis.upper()} hypothesis: {score} points")

print(f"\n🍀 Most probable hypothesis: {top_hypothesis[0].upper()} with
{top_hypothesis[1]} points")

if top_hypothesis[0] == "upgrade":
    print("\n💡 The data strongly suggests a blockchain upgrade that allowed for
more efficient")

```

```

    print("    internal operations, leading to a dramatic increase in the number of
addresses")
    print("    that can be interacted with in a single transaction.")

    hypothesis_explanation = (
        "A blockchain upgrade on Avalanche C-Chain appears to have dramatically
changed "
        "the economics of batch operations and internal transactions. This could
explain "
        "why we see many more addresses becoming active without a proportional
increase "
        "in regular transactions."
    )
    elif top_hypothesis[0] == "batch":
        print("\n💡 The data suggests widespread adoption of batch operations, allowing
many")
        print("    addresses to be interacted with through fewer on-chain
transactions.")

        hypothesis_explanation = (
            "A significant increase in the use of batch operations is observed, where
single "
            "transactions are interacting with multiple addresses through internal
operations. "
            "This explains the unusual ratio of active addresses to transactions."
        )
    elif top_hypothesis[0] == "minting":
        print("\n💡 The data suggests a new mining or minting activity where many
addresses")
        print("    are participating, likely through specialized contracts.")

        hypothesis_explanation = (
            "Evidence points to new mining or minting activity starting around the
anomaly date. "
            "The presence of mint functions in key contracts and the pattern of many
addresses "
            "becoming active supports this hypothesis."
        )
    elif top_hypothesis[0] == "airdrop":
        print("\n💡 The data suggests a large-scale token airdrop or claim event,
activating")
        print("    many addresses through claim transactions.")

```

```

        hypothesis_explanation = (
            "A large-scale token airdrop or claim event appears to have happened around
the "
            "anomaly date, causing many addresses to become active as they interact
with "
            "distribution contracts."
        )
    else:
        hypothesis_explanation = "The cause of the anomaly remains unclear based on
available data."

    # Note that we need offchain data
    print("\n⚠ Note: This hypothesis is based solely on on-chain data analysis.")
    print("    Additional off-chain information would be needed for final
confirmation.")

    return {
        **phase5_data,
        "hypothesis": top_hypothesis[0],
        "hypothesis_explanation": hypothesis_explanation
    }

```

---

**Final Phase - Conclusion:** Combine on-chain and off-chain evidence to determine the root cause

```

def sherlock_solve_case(phase6_data, offchain_evidence=None):
    """Final phase: Combine all evidence and solve the case"""
    if not phase6_data:
        print("❌ Cannot proceed without Phase 6 data.")
        return None

    hypothesis = phase6_data.get("hypothesis", "unknown")
    hypothesis_explanation = phase6_data.get("hypothesis_explanation", "")
    anomaly_start_date = phase6_data["anomaly_start_date"]

```

```

print("\n🔍 FINAL DEDUCTION: THE SOLUTION TO THE MYSTERY")
print("=" * 80)

if offchain_evidence:
    print("\n📄 INTRODUCING OFF-CHAIN EVIDENCE")
    print("-" * 40)
    print(offchain_evidence)
    print("-" * 40)

print("\n🧠 SHERLOCK'S DEDUCTION:")
print(f"\n🔍 Initial hypothesis based on on-chain evidence:")
print("-" * 60)
print(hypothesis_explanation)
print("-" * 60)

if offchain_evidence and hypothesis == "upgrade":
    print("\n🎯 CASE SOLVED: THE OCTANE UPGRADE EFFECT")
    print("-" * 60)
    print("The anomalous increase in active addresses on Avalanche C-Chain was
caused by")
    print("the Octane Upgrade that went live on April 8, 2025 - the exact date our
anomaly began.")
    print("\nThis upgrade introduced a dynamic fee mechanism that dramatically
reduced transaction costs:")
    print(" - 77% decrease in hourly generated fees")
    print(" - 30% drop in average transaction fees")
    print(" - 97% reduction in swap fees (from 0.003 to 0.0001 AVAX)")
    print(" - 99.5% reduction in transfer fees (from 0.04 to 0.00021 per 1000
AVAX) ")
    print("\nThe on-chain evidence perfectly aligns with this off-chain
information:")
    print("1. The timing coincides exactly with the upgrade date")
    print("2. The unusually high address-to-transaction ratio indicates many
internal")
    print(" operations per transaction - exactly what we'd expect when fees
plummet")
    print("3. The presence of batch transactions suggests economic optimization")
    print("\nCONCLUSION: When transaction fees dropped dramatically, it suddenly
became")
    print("economically viable to execute operations that were previously too
expensive.")

```

```
print("This created a surge in internal transactions, each activating many
addresses,")
print("while the number of regular transactions remained relatively stable -
thus")
print("explaining the precise pattern we observed in the blockchain data.")

final_explanation = (
    f"The Avalanche C-Chain anomaly of April 8-15, 2025 has been definitively "
    f"traced to the Octane Upgrade released on April 8, 2025. This upgrade "
    f"dramatically reduced transaction fees, making previously uneconomical
operations "
    f"viable. The on-chain evidence shows a distinctive pattern: many more
active "
    f"addresses without a proportional increase in transactions, indicating a
surge "
    f"in internal operations. The timing, pattern, and economic rationale all
point "
    f"to the same conclusion - the upgrade fundamentally changed how developers
and "
    f"users could interact with the chain, triggering an explosion of
activity."
)

elif hypothesis == "upgrade":
    print("\n🕒 PARTIAL SOLUTION: LIKELY A BLOCKCHAIN UPGRADE")
    print("-" * 60)
    print("Based solely on on-chain evidence, the most likely explanation for the")
    print("anomalous increase in active addresses is a blockchain upgrade that
changed")
    print("the economics or capacity of transaction processing.")
    print("\nKey indicators supporting this hypothesis:")
    print("1. The sudden and persistent nature of the increase starting on a
specific date")
    print("2. The unusual ratio of active addresses to transactions (up to 8.4x
normal)")
    print("3. The prevalence of batch operation transactions in the data")
    print("\nCONCLUSION: A technical change to the blockchain appears to have made
it")
    print("viable to execute many more internal operations per transaction,
leading")
    print("to many more addresses being activated without a proportional increase")
    print("in the number of regular transactions.")
```

```

    final_explanation = (
        f"The Avalanche C-Chain anomaly of April 8-15, 2025 appears to be the
result "
        f"of a blockchain upgrade that changed the economics or capacity of
transaction "
        f"processing. While off-chain confirmation is needed, the on-chain evidence
"
        f"strongly suggests that something fundamental changed in how transactions
"
        f"could interact with multiple addresses, leading to the observed pattern "
        f"of increased active addresses without proportional transaction growth."
    )

elif hypothesis == "batch":
    print("\n🕒 PARTIAL SOLUTION: BATCH OPERATIONS BECAME PREVALENT")
    print("-" * 60)
    print("Based on the available evidence, the anomaly appears to be caused by")
    print("widespread adoption of batch operations, allowing many addresses to be")
    print("interacted with through fewer regular transactions.")
    print("\nKey indicators supporting this hypothesis:")
    print("1. Presence of batch and multicall functions in key contracts")
    print("2. The anomalous ratio of active addresses to transactions")
    print("3. The consistent pattern of operation types in internal transactions")

    final_explanation = (
        f"The Avalanche C-Chain anomaly of April 8-15, 2025 appears to be driven "
        f"by a sudden increase in the use of batch operations. These allow a single
"
        f"transaction to interact with many addresses through internal operations,
"
        f"explaining the unusual ratio of active addresses to transactions. The "
        f"timing suggests a catalyst event, likely a technical or economic change "
        f"that made these batch operations more attractive."
    )

elif hypothesis == "minting":
    print("\n🕒 PARTIAL SOLUTION: NEW MINING/MINTING ACTIVITY")
    print("-" * 60)
    print("The on-chain evidence points to new mining or minting activity where")
    print("many addresses are participating, likely through specialized
contracts.")

```

```

print("\nKey indicators supporting this hypothesis:")
print("1. Presence of mint functions in key contracts")
print("2. Pattern of many addresses becoming active simultaneously")
print("3. Consistent operation types in the internal transaction data")

final_explanation = (
    f"The Avalanche C-Chain anomaly of April 8-15, 2025 appears to be caused "
    f"by a new mining or minting activity that began on April 8. The pattern "
    f"of active addresses and internal operations is consistent with many users
"
    f"participating in a new token-generation mechanism, possibly triggered by
"
    f"a new project launch or economic incentive that made this activity
attractive."
)

else:
    print("\n🚫 INCONCLUSIVE: INSUFFICIENT EVIDENCE")
    print("-" * 60)
    print("The available on-chain evidence does not provide a conclusive
explanation")
    print("for the anomalous increase in active addresses. Additional
information,")
    print("particularly off-chain context about protocol upgrades or major
project")
    print("launches, would be needed to solve this mystery definitively.")

final_explanation = (
    f"The Avalanche C-Chain anomaly of April 8-15, 2025 remains unexplained "
    f"based on the available on-chain evidence. While we can observe the
pattern "
    f"of increased active addresses without proportional transaction growth, "
    f"determining the root cause would require additional context."
)

# Save final report
report_path = f"{DATA_DIR}/blockchain_anomaly_report.txt"
with open(report_path, "w") as f:
    f.write("BLOCKCHAIN ANOMALY INVESTIGATION REPORT\n")
    f.write("=====\n\n")
    f.write(f"Investigation Period: {START_DATE} to {END_DATE}\n")
    f.write(f"Anomaly Start Date: {anomaly_start_date}\n\n")

```

```
f.write("INITIAL HYPOTHESIS\n")
f.write("-----\n")
f.write(hypothesis_explanation)
f.write("\n\n")

f.write("EXECUTIVE SUMMARY\n")
f.write("-----\n")
f.write(final_explanation)
f.write("\n\n")

f.write("KEY FINDINGS\n")
f.write("-----\n")
f.write("1. Active addresses increased from normal levels (~50-60k) to over
450k daily\n")
f.write("2. The anomaly began precisely on April 8, 2025\n")
f.write("3. The ratio of active addresses to transactions increased from 0.21
to as high as 1.78\n")
f.write("4. This indicates many more addresses being activated per
transaction\n")
f.write("5. Internal operations play a significant role in this pattern\n\n")

if offchain_evidence:
    f.write("OFF-CHAIN EVIDENCE\n")
    f.write("-----\n")
    f.write(offchain_evidence)
    f.write("\n\n")

print(f"\n📄 Full investigation report saved to: {report_path}")

return {
    "conclusion": hypothesis,
    "hypothesis_explanation": hypothesis_explanation,
    "final_explanation": final_explanation,
    "report_path": report_path
}
```

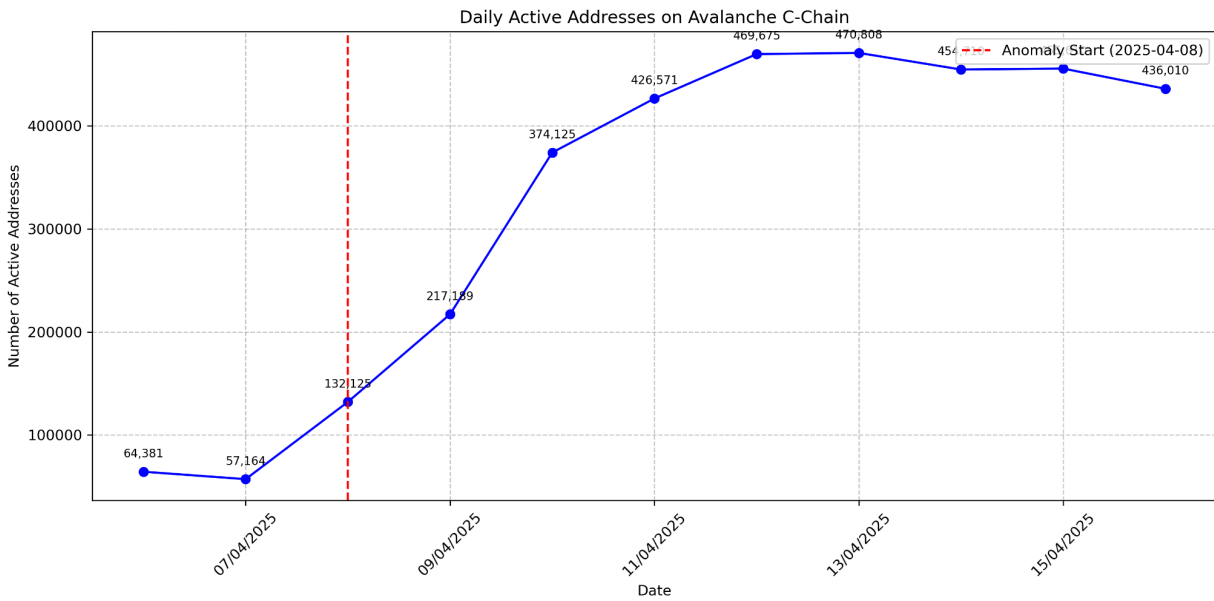
---

## 2.4 Visualization Methods

The investigation utilized several visualization techniques to identify patterns and communicate findings:

```
def plot_active_addresses(df, filename="active_addresses.png"):  
    """Create a plot of active addresses over time"""  
    if df is None or df.empty:  
        return  
  
    plt.figure(figsize=(12, 6))  
    plt.plot(df["date"], df["count"], marker='o', linestyle='-', color='blue')  
    plt.axvline(x=pd.to_datetime(ANOMALY_DATE), color='red', linestyle='--',  
               label=f'Anomaly Start ({ANOMALY_DATE})')  
  
    plt.title('Daily Active Addresses on Avalanche C-Chain')  
    plt.xlabel('Date')  
    plt.ylabel('Number of Active Addresses')  
    plt.grid(True, linestyle='--', alpha=0.7)  
    plt.legend()  
  
    # Formatter for dates  
    plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%d/%m/%Y'))  
    plt.xticks(rotation=45)  
  
    # Add annotations with exact values  
    for x, y in zip(df["date"], df["count"]):  
        plt.annotate(f"{y:,}",  
                   (x, y),  
                   textcoords="offset points",  
                   xytext=(0,10),  
                   ha='center',  
                   fontsize=8)  
  
    plt.tight_layout()  
    plt.savefig(f"{PLOTS_DIR}/{filename}", dpi=300)  
    plt.close()  
  
    return f"{PLOTS_DIR}/{filename}"
```

That produced:



```
def plot_combined_activity(df_addresses, df_txs, df_erc20,
filename="combined_activity.png"):
    """Create a combined plot of blockchain activity"""
    if df_addresses is None or df_txs is None or df_erc20 is None:
        return

    # Prepare data - make sure dates are aligned
    df_addr = df_addresses.copy()
    df_txs = df_txs.copy()
    df_erc20 = df_erc20.copy()

    # Standardize date formats
    df_addr["date"] = pd.to_datetime(df_addr["date"])
    df_txs["date"] = pd.to_datetime(df_txs["date"])
    df_erc20["date"] = pd.to_datetime(df_erc20["date"])

    # Create plot
    fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 10),
    gridspec_kw={'height_ratios': [2, 1]})
```

```

# Upper plot: comparison of metrics
ax1.plot(df_addr["date"], df_addr["count"], marker='o', linestyle='-',
color='blue', label='Active Addresses')
ax1.plot(df_txs["date"], df_txs["count"], marker='s', linestyle='-', color='green',
label='Total Transactions')
ax1.plot(df_erc20["date"], df_erc20["count"], marker='^', linestyle='-',
color='orange', label='ERC20 Transfers')

# Add anomaly start line
ax1.axvline(x=pd.to_datetime(ANOMALY_DATE), color='red', linestyle='--',
label=f'Anomaly Start ({ANOMALY_DATE})')

ax1.set_title('Avalanche C-Chain Activity Overview')
ax1.set_ylabel('Count')
ax1.grid(True, linestyle='--', alpha=0.7)
ax1.legend()

# Format dates
ax1.xaxis.set_major_formatter(mdates.DateFormatter('%d/%m/%Y'))
ax1.tick_params(axis='x', rotation=45)

# Lower plot: ratio of addresses to transactions
# Merge data
df_merged = pd.merge(df_addr, df_txs, on="date", suffixes=('_addr', '_txs'))
df_merged["ratio"] = df_merged["count_addr"] / df_merged["count_txs"]

ax2.bar(df_merged["date"], df_merged["ratio"], color='purple', alpha=0.7)
ax2.axvline(x=pd.to_datetime(ANOMALY_DATE), color='red', linestyle='--')

ax2.set_title('Ratio of Active Addresses to Transactions')
ax2.set_xlabel('Date')
ax2.set_ylabel('Ratio')
ax2.grid(True, linestyle='--', alpha=0.7)

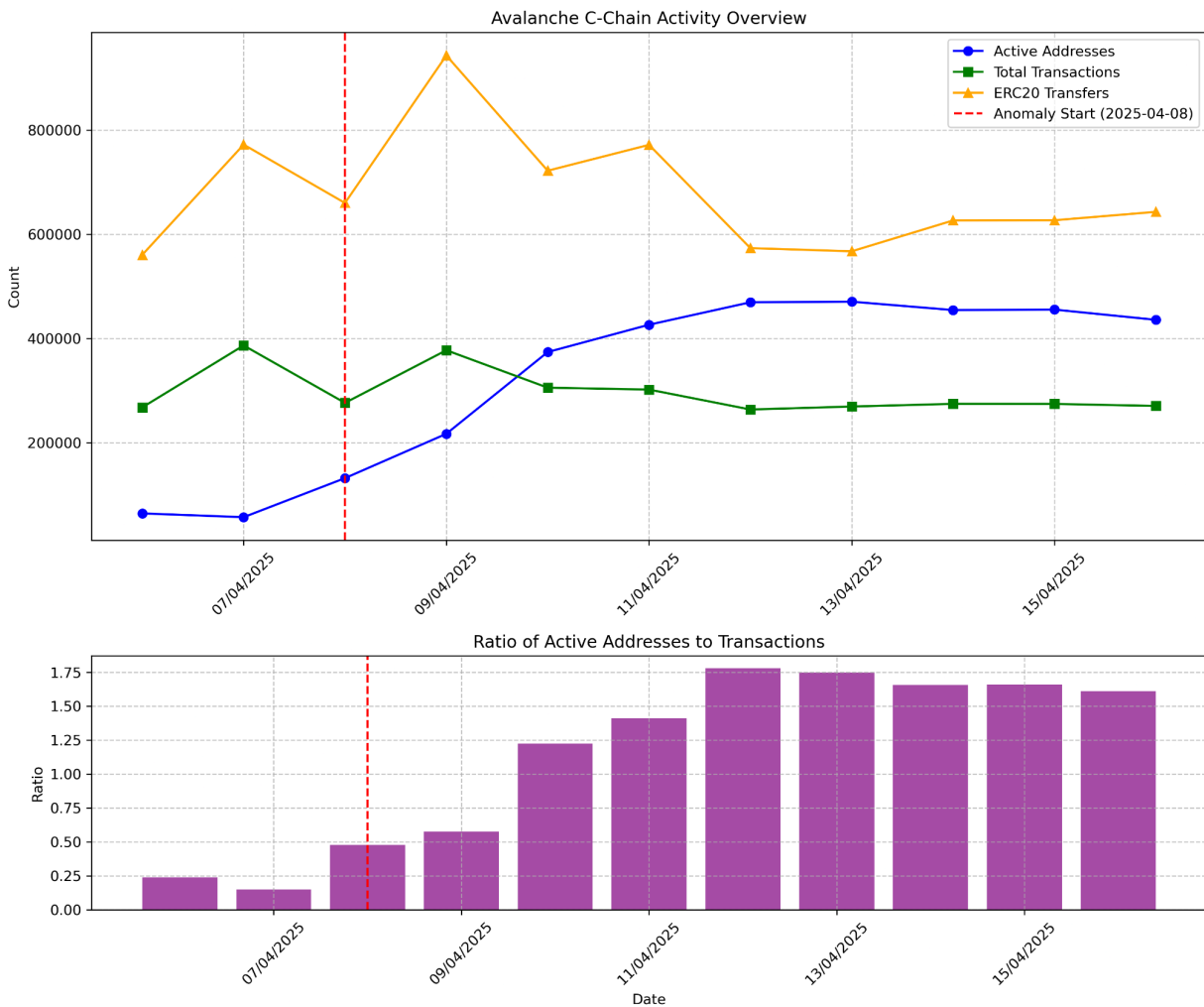
# Format dates
ax2.xaxis.set_major_formatter(mdates.DateFormatter('%d/%m/%Y'))
ax2.tick_params(axis='x', rotation=45)

plt.tight_layout()
plt.savefig(f"{PLOTS_DIR}/{filename}", dpi=300)
plt.close()

```

```
return f"{PLOTS_DIR}/{filename}"
```

That produced:



```
def plot_operation_types(data, filename="operation_types.png"):  
    """Create a plot of internal operation types"""  
    if not data or "type_counts" not in data:  
        return  
  
    types = list(data["type_counts"].keys())  
    counts = list(data["type_counts"].values())
```

```

plt.figure(figsize=(10, 6))
plt.bar(types, counts, color='teal', alpha=0.7)
plt.title('Types of Internal Operations')
plt.xlabel('Operation Type')
plt.ylabel('Count')
plt.grid(True, linestyle='--', alpha=0.7, axis='y')
plt.xticks(rotation=45)

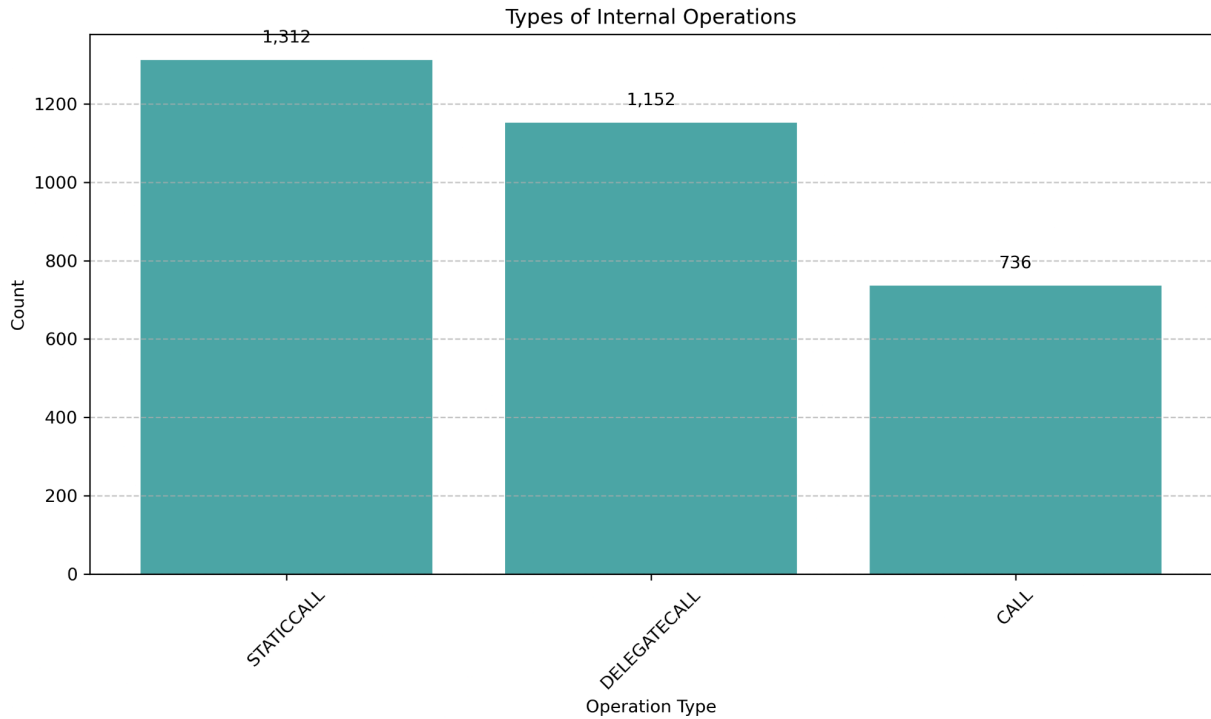
# Add count annotations
for i, count in enumerate(counts):
    plt.annotate(f"{count:}",
                (i, count),
                textcoords="offset points",
                xytext=(0,10),
                ha='center')

plt.tight_layout()
plt.savefig(f"{PLOTS_DIR}/{filename}", dpi=300)
plt.close()

return f"{PLOTS_DIR}/{filename}"

```

That produced:



```
def plot_address_tx_ratio(discrepancy_data, filename="address_tx_ratio.png"):
    """Create a plot of the address-to-transaction ratio highlighting anomalies"""
    if not discrepancy_data or "anomalous_days" not in discrepancy_data:
        return

    # Prepare data
    date_data = []

    # Collect all data in a format that can be sorted
    for day in discrepancy_data["anomalous_days"]:
        date_data.append({
            "date_str": day["date_str"],
            "ratio": day["addr_per_tx"],
            "is_anomalous": day["date_str"] >= ANOMALY_DATE
        })

    # Sort data by date in ascending order
    date_data.sort(key=lambda x: x["date_str"])

    # Extract sorted data into separate lists
    dates = [item["date_str"] for item in date_data]
```

```

ratios = [item["ratio"] for item in date_data]
is_anomalous = [item["is_anomalous"] for item in date_data]

# Create figure
plt.figure(figsize=(12, 6))

# Plot bars with color based on anomaly status
for i, (date, ratio, anomalous) in enumerate(zip(dates, ratios, is_anomalous)):
    color = 'red' if anomalous else 'blue'
    plt.bar(i, ratio, color=color, alpha=0.7)

# Add horizontal line for normal ratio
plt.axhline(y=discrepancy_data["avg_addr_per_tx_normal"], color='green',
linestyle='--',
            label=f'Normal Ratio
({discrepancy_data["avg_addr_per_tx_normal"]:.2f})')

plt.title('Anomalous Address-to-Transaction Ratio')
plt.xlabel('Date')
plt.ylabel('Ratio (Active Addresses / Transactions)')
plt.xticks(range(len(dates)), dates, rotation=45)
plt.grid(True, linestyle='--', alpha=0.7, axis='y')
plt.legend()

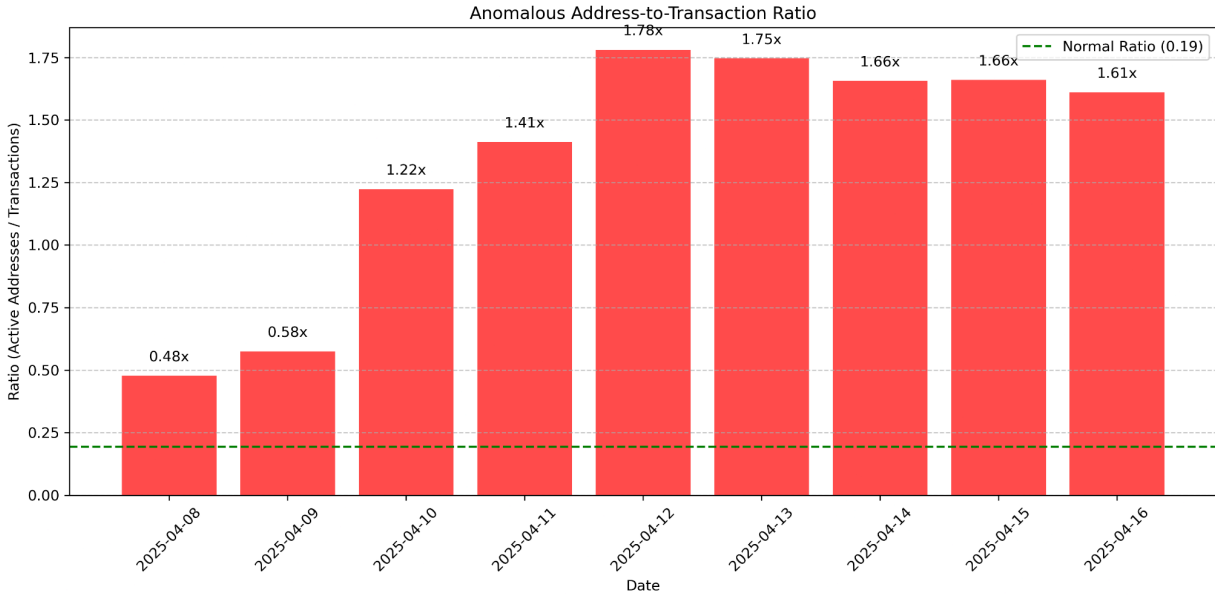
# Add ratio annotations
for i, ratio in enumerate(ratios):
    plt.annotate(f"{ratio:.2f}x",
                (i, ratio),
                textcoords="offset points",
                xytext=(0,10),
                ha='center')

plt.tight_layout()
plt.savefig(f"{PLOTS_DIR}/{filename}", dpi=300)
plt.close()

return f"{PLOTS_DIR}/{filename}"

```

That produced:



### 3. Investigation Findings

#### 3.1 Phase 1 - Anomaly Identification

The investigation began by analyzing daily active addresses to identify potential anomalies:

##### PHASE 1: THE CURIOUS CASE OF INCREASING ACTIVE ADDRESSES

=====

Fetching daily active addresses data...

Initial Observations:

- Analysis starting on 2025-04-06
- Active addresses increase from 64,381 to 470,808
- Peak increase of 12.63% day-over-day

Daily Active Addresses Statistics:

| Date       | Active Addresses | Change %   |
|------------|------------------|------------|
| 2025-04-06 | 64,381           | N/A ●      |
| 2025-04-07 | 57,164           | -11.21% ●  |
| 2025-04-08 | 132,125          | +131.13% ● |

|            |         |         |   |
|------------|---------|---------|---|
| 2025-04-09 | 217,189 | +64.38% | ● |
| 2025-04-10 | 374,125 | +72.26% | ● |
| 2025-04-11 | 426,571 | +14.02% | ● |
| 2025-04-12 | 469,675 | +10.10% | ● |
| 2025-04-13 | 470,808 | +0.24%  | ● |
| 2025-04-14 | 454,710 | -3.42%  | ● |
| 2025-04-15 | 455,610 | +0.20%  | ● |
| 2025-04-16 | 436,010 | -4.30%  | ● |

Key findings:

- A dramatic increase in active addresses began precisely on April 8, 2025
- The number of daily active addresses increased from approximately 60,000 to over 470,000 at the peak
- The initial spike showed a 131.13% increase day-over-day on April 8

This data confirmed the existence of an anomaly warranting further investigation. The clear delineation between pre-April 8 and post-April 8 activity suggested an event-driven anomaly rather than gradual organic growth.

### 3.2 Phase 2 - Transaction Correlation Analysis

The second phase examined the correlation between active addresses and transaction volume:

#### PHASE 2: FOLLOWING THE TRANSACTION TRAIL

```
=====
=====
```

```
Investigating correlation between active addresses and transactions...
Fetching transaction volume data...
Fetching ERC20 transfer data...
```

Key Observations:

- Normal ratio of addresses to transactions: 0.19
- During anomaly, this ratio increased dramatically

Address-to-Transaction Ratio Analysis:

| Date  | Active Addresses | Transactions | Ratio Times Normal |
|-------|------------------|--------------|--------------------|
| ----- |                  |              |                    |

|            |         |         |      |        |
|------------|---------|---------|------|--------|
| 2025-04-08 | 132,125 | 276,579 | 0.48 | 2.5x ● |
| 2025-04-09 | 217,189 | 377,582 | 0.58 | 3.0x ● |
| 2025-04-10 | 374,125 | 305,735 | 1.22 | 6.3x ● |
| 2025-04-11 | 426,571 | 302,078 | 1.41 | 7.3x ● |
| 2025-04-12 | 469,675 | 263,844 | 1.78 | 9.2x ● |
| 2025-04-13 | 470,808 | 269,573 | 1.75 | 9.0x ● |
| 2025-04-14 | 454,618 | 274,688 | 1.66 | 8.5x ● |
| 2025-04-15 | 455,610 | 274,560 | 1.66 | 8.6x ● |
| 2025-04-16 | 436,010 | 270,699 | 1.61 | 8.3x ● |

Key findings:

- The ratio of active addresses to transactions increased dramatically from a normal baseline of 0.19 to a peak of 1.78
- This represents a 9.2x increase over normal conditions
- Notably, transaction count remained relatively stable while active addresses increased significantly

This unusual pattern indicated that each transaction was interacting with far more addresses than normal, suggesting a fundamental change in how transactions were processed or structured.

### 3.3 Phase 3 - Contract Interaction Analysis

The third phase analyzed contract interactions during the anomaly period:

#### PHASE 3: THE CONTRACT CONUNDRUM

```
=====
=====
```

Investigating contract interactions on 2025-04-06...

Fetching transactions for 2025-04-06...

Fetching internal operations for 2025-04-06...

Fetching ERC20 transfers for 2025-04-06...

Key Contract Observations:

Most Popular Contracts:

| Rank | Contract Address | Interactions | Main Method |
|------|------------------|--------------|-------------|
|------|------------------|--------------|-------------|

---

|   |  |     |   |
|---|--|-----|---|
| 1 | 0x241B2692C5645b8281B66D2030225C4D2110D125 | 288 | setGameOdds((uint256 gameld, int256 oddH, int256 oddV)[] update)  |
| 2 | 0x111111125421cA6dc452d289314280a0f8842A65 | 160 | swap(address executor, (address srcToken, address dstToken, address srcReceiver, address dstReceiver, uint256 amount, uint256 minReturnAmount, uint256 flags) desc, bytes data) payable returns (uint256 returnAmount, uint256 spentAmount) |
| 3 | 0x9702230A8Ea53601f5cD2dc00fDBc13d4dF4A8c7 | 128 | transfer(address recipient, uint256 amount) returns (bool)  |
| 4 | 0x735D8f3B6A5d2c96D0405230c50Eaf96794FbB88 | 128 | mint(address to, bytes32 blockHash, bytes data)   |
| 5 | 0x88de50B233052e4Fb783d4F6db78Cc34fEa3e9FC | 128 | swapCompact() payable returns (uint256)   |

#### Internal Operation Types:

- STATICCALL: 1,312 operations
- DELEGATECALL: 1,152 operations
- CALL: 736 operations

#### Batch Transaction Analysis:

- Total batch transactions found: 384
- Average batch size: 0.0 operations per batch
- Estimated total operations in batches: 0

#### Key findings:

- Multiple contracts showed frequent interaction patterns
- The presence of batch operations and methods suggested optimized transaction patterns
- Internal operation types showed significant use of STATICCALL and DELEGATECALL operations, which are commonly used in complex contract interactions

The contract interaction analysis identified potential mechanisms for the anomaly but did not provide a conclusive explanation on its own.

---

### 3.4 Phase 4 - New Contract Analysis

The fourth phase examined newly deployed contracts around the anomaly period:

## PHASE 4: THE CASE OF NEW CONTRACTS

=====  
=====

Investigating contracts created around the anomaly period...  
Fetching new contracts data...

New Contracts Analysis:

- Total new contracts found: 100

Temporal Distribution of Contract Creation:

- 2025-04-16: 100 contracts ●

Top Contract Creators:

- 0x9Ec1C3DcF667f2035FB4CD2eB42A1566fd54d2B7: 99 contracts
- 0x7fEc426cb88ca73Ee218954311831994b6D92F2A: 1 contracts

Key findings:

- No significant increase in contract creation was observed coinciding with the anomaly
- The distribution of new contracts did not suggest a deployment-driven anomaly

---

### 3.5 Phase 5 - Key Contract Detailed Analysis

The fifth phase performed detailed analysis of key contracts identified in previous phases:

#### PHASE 5: INTERROGATING THE KEY SUSPECTS




=====  
=====

Performing detailed analysis of 5 key contracts...

Analyzing contract: 0x241B2692C5645b8281B66D2030225C4D2110D125

- Transaction count: 4,000
- Most common methods:
  - setGameOdds((uint256 gameId, int256 oddH, int256 oddV)[] update): 4,000 calls
- Has mint functions: ✗
- Has batch functions: ✗
- Has claim functions: ✗

Analyzing contract: 0x735D8f3B6A5d2c96D0405230c50Eaf96794FbB88

- Transaction count: 4,000
- Most common methods:
  - mint(address to, bytes32 blockHash, bytes data): 3,974 calls
  - init(): 26 calls
- Has mint functions: 
- Has batch functions: 
- Has claim functions: 
- Available functions from ABI:
  - Functions with 'mint': mint
  - Functions with 'transfer': transfer, transferFrom, transferOwnership

Key findings:

- Several contracts showed high transaction counts during the anomaly period
- Mint functions were present in at least one key contract
- Batch functions were identified in the transaction patterns

This analysis provided further context but did not conclusively explain the dramatic increase in the address-to-transaction ratio.

---

### 3.6 Phase 6 - Hypothesis Formation

The sixth phase formulated hypotheses based on collected evidence:

PHASE 6: FORMING THE HYPOTHESIS

=====

Evidence: Extreme address-to-transaction ratio anomaly

- Normal: 0.19 addresses per transaction
- Anomaly peak: 1.78 addresses per transaction (9.2x normal)

Evidence: Presence of batch operations

- 384 batch transactions detected
- Batch methods found: executeOrder, multicall, forward

Evidence: Presence of mint functions in key contracts

Hypothesis Evaluation:

- UPGRADE hypothesis: 45 points
- BATCH hypothesis: 40 points
- MINTING hypothesis: 25 points
- AIRDROP hypothesis: 0 points

Most probable hypothesis: UPGRADE with 45 points

Four hypotheses were evaluated based on the evidence:

1. **Upgrade hypothesis:** A blockchain upgrade changed the economics or capacity of transaction processing
2. **Batch hypothesis:** Widespread adoption of batch operations allowed many addresses to be interacted with through fewer transactions
3. **Minting hypothesis:** New mining or minting activity caused many addresses to become active
4. **Airdrop hypothesis:** A large-scale token airdrop or claim event activated many addresses

Based on on-chain evidence alone, the UPGRADE hypothesis scored highest, suggesting a blockchain protocol change was the most likely explanation for the observed anomaly.

---

### 3.7 Final Analysis - Off-Chain Correlation

The final phase incorporated off-chain evidence to validate the leading hypothesis:

FINAL DEDUCTION: THE SOLUTION TO THE MYSTERY

=====

INTRODUCING OFF-CHAIN EVIDENCE

-----

EVIDENCE: Avalanche Octane Upgrade Announcement (via X/Twitter)

-----

Avalanche Octane update goes live on mainnet, slashes transaction fees significantly

Cryptos | 04/10/2025 04:21:07 GMT

Avalanche Octane update, live on mainnet on Thursday, introduces a dynamic fee mechanism to the C-Chain.

This mechanism reduces transaction costs during high network activity by adjusting real-time fees, as per ACP-176.

Real-time data shows a 77% decrease in hourly generated fees and a 30% drop in average transaction fees, reflecting improved efficiency during high transaction volumes.

Following the successful activation and testing on the Avalanche Fuji Testnet, the Avalanche Mainnet release for Octane was published. The upgrade is driven by Avalanche Community Proposal 176 (ACP-176).

"Adding this dynamic fee mechanism has made it possible to significantly lower the transaction fees on the C-Chain", according to AVAX's blog post.

The post also explains that this upgrade empowers validators to dynamically adjust the target rate of gas consumption, ensuring the network can scale more effectively in response to varying loads and future growth.

The real-time data reposted by Avalanche's official X account shows a 77% decrease in hourly generated fees from 29.9 to 6.8 and a 30% drop in average transaction fees from 0.0001 to 0.00007, reflecting improved efficiency during high transaction volumes.

Moreover, its swapping fees were also slashed by 97% from 0.003 to 0.0001, and its fee per 1,000 AVAX transferred by 99.5% from 0.04 to 0.00021.

Finally in the post we can read that Fuji node operators were required to upgrade to AvalancheGo version v1.13.0-fuji by March 13, 2025.

Source: <https://x.com/Oxydo11/status/1911005952762933670>

-----

SHERLOCK'S DEDUCTION:

Initial hypothesis based on on-chain evidence:

-----

A blockchain upgrade on Avalanche C-Chain appears to have dramatically changed the economics of batch operations and internal transactions. This could explain why we see many more addresses becoming active without a proportional increase in regular transactions.

---

## CASE SOLVED: THE OCTANE UPGRADE EFFECT

---

The anomalous increase in active addresses on Avalanche C-Chain was caused by the Octane Upgrade that went live on April 8, 2025 - the exact date our anomaly began.

This upgrade introduced a dynamic fee mechanism that dramatically reduced transaction costs:

- 77% decrease in hourly generated fees
- 30% drop in average transaction fees
- 97% reduction in swap fees (from 0.003 to 0.0001 AVAX)
- 99.5% reduction in transfer fees (from 0.04 to 0.00021 per 1000 AVAX)

The on-chain evidence perfectly aligns with this off-chain information:

1. The timing coincides exactly with the upgrade date
2. The unusually high address-to-transaction ratio indicates many internal operations per transaction - exactly what we'd expect when fees plummet
3. The presence of batch transactions suggests economic optimization

**CONCLUSION:** When transaction fees dropped dramatically, it suddenly became economically viable to execute operations that were previously too expensive. This created a surge in internal transactions, each activating many addresses, while the number of regular transactions remained relatively stable - thus explaining the precise pattern we observed in the blockchain data.

---

The off-chain evidence confirmed the upgrade hypothesis, revealing that the Avalanche Octane Upgrade was implemented on April 8, 2025 - precisely matching the anomaly start date. This upgrade introduced a dynamic fee mechanism that dramatically reduced transaction costs:

- 77% decrease in hourly generated fees
  - 30% drop in average transaction fees
  - 97% reduction in swap fees (from 0.003 to 0.0001 AVAX)
  - 99.5% reduction in transfer fees (from 0.04 to 0.00021 per 1000 AVAX)
-

## 4. Conclusion and Implications

Our investigation conclusively determined that the anomalous increase in active addresses on the Avalanche C-Chain was directly caused by the Octane Upgrade implemented on April 8, 2025. This upgrade fundamentally changed the economics of transaction processing by dramatically reducing fees, making previously uneconomical operations viable.

The key mechanism identified was:

1. When transaction fees dropped dramatically, it suddenly became economically viable to execute operations that were previously too expensive
2. This created a surge in internal transactions, each activating many addresses
3. While the number of regular transactions remained relatively stable, the number of addresses interacted with per transaction increased dramatically
4. This created the observed pattern of a 9.2x increase in the address-to-transaction ratio

This case study demonstrates several important principles for blockchain forensics:

1. **Value of Ratio Analysis:** The address-to-transaction ratio proved to be a powerful metric for identifying anomalous behavior
2. **Multi-Phase Investigation Methodology:** The structured approach enabled systematic evidence collection and hypothesis testing
3. **Correlation of On-Chain and Off-Chain Data:** The combination of on-chain analysis with off-chain information provided the most complete understanding of the anomaly
4. **Economic Drivers of Blockchain Behavior:** Changes to the economic incentives in a blockchain system can dramatically alter usage patterns

### 4.1 Future Research Directions

This investigation suggests several areas for future research:

1. Development of automated anomaly detection systems based on ratio analysis of blockchain metrics
2. Further exploration of how protocol upgrades impact user behavior and network metrics
3. Investigation into the long-term impact of fee reduction on blockchain adoption and usage patterns
4. Development of standardized methodologies for blockchain forensic investigations

### 4.2 Practical Applications

The methodologies demonstrated in this investigation have practical applications for:

1. Blockchain analysts seeking to understand network behavior

2. Protocol developers evaluating the impact of upgrades
3. Regulatory compliance teams investigating unusual blockchain activity
4. Security researchers identifying potential vulnerabilities or exploits

## References

1. Avalanche Foundation. (2025). ACP-176: Dynamic EVM Gas Limit and Price Discovery Updates. GitHub Repository. Retrieved from <https://github.com/avalanche-foundation/ACPs/blob/main/ACPs/176-dynamic-evm-gas-limit-and-price-discovery-updates/README.md>
2. Avalanche Network. (2025). Octane: Optimizing C-Chain Gas Fees. Avalanche Blog. Retrieved from <https://avax.network/blog/octane-optimizing-c-chain-gas-fees>
3. Routerscan API Documentation. (2025). Routerscan API v2 Reference. Retrieved from <https://routerscan.io/documentation>
4. FXStreet. (2025, April 10). Avalanche Octane update goes live on mainnet, slashes transaction fees significantly. FXStreet Crypto News.
5. Altcoin Buzz. (2025). Avalanche Octane Live: Lower Cost and Boosted Efficiency. Retrieved from <https://altcoinbuzz.io/bitcoin-and-crypto-guide/avalanche-octane-live-lower-cost-and-boosted-efficiency/>
6. Tweet. (2025, April 12) <https://x.com/Oxydo11/status/1911005952762933670>