Towards Low-Latency Byzantine Agreement Protocols Using RDMA

DSN Workshop on Byzantine Consensus and Resilient Blockchains

Signe Rüsch, Ines Messadi, Rüdiger Kapitza, 2018-06-25
ruesch@ibr.cs.tu-bs.de
Technische Universität Braunschweig, Germany
Blockchain and Cryptocurrencies

- **Permissionless**: Proof-of-Work for ordering agreement
  - Scalability and energy consumption issues

- **Permissioned**: e.g. for companies’ SCM
  - Blocks can be created by dedicated nodes in data centers
  - Crash-fault tolerant protocols: Hyperledger Fabric with Kafka

---

| Block n          | Block n+1         | Block n+2         |
|------------------|-------------------|-------------------|
| tx1              | tx1               | tx1               |
| tx2              | tx2               | tx2               |
| ...              | ...               | ...               |
| Hash h(n-1)      | Hash h(n)         | Hash h(n+1)       |

---

Byzantine Agreement Protocols Using RDMA
Blockchain and Cryptocurrencies

- **Permissionless:** Proof-of-Work for ordering agreement
  - Scalability and energy consumption issues

- **Permissioned:** e.g. for companies’ SCM
  - Blocks can be created by dedicated nodes in data centers
  - Crash-fault tolerant protocols: Hyperledger Fabric with Kafka
    → Additional security of **Byzantine fault tolerant (BFT)** protocols!
BFT Protocols

- $3f + 1$ nodes reach \textit{consensus} on order of requests
- High \textit{throughput} requirements: blockchain to replace company’s database
- Multiple rounds of message exchanges
- Broadcast steps
  \rightarrow \text{High message complexity and latency!}
BFT Protocols

- Message complexity optimization focusing on protocol level
  - E.g. hybrid BFT protocols
- Current BFT protocols achieve necessary throughput
  - ≈1 Million operations/second (Behl et al., EuroSys’17)
BFT Protocols

- Message complexity optimization focusing on protocol level
  - E.g. hybrid BFT protocols
- Current BFT protocols achieve necessary throughput
  - \( \approx 1 \) Million operations/second (Behl et al., EuroSys’17)

Our focus: reduce latency on network layer with technology available in data centers!
TCP Overhead

- Two intermediate **data copy** steps per host
  - Application → kernel → network
  - Network → kernel → application
- >50% of TCP latency due to data copying (Frey et al., ICDCS’09)
TCP Overhead

- Two intermediate **data copy** steps per host
  - Application $\rightarrow$ kernel $\rightarrow$ network
  - Network $\rightarrow$ kernel $\rightarrow$ application
- >50% of TCP latency due to data copying (Frey et al., ICDCS’09)

Reduce latency of BFT protocols with **Remote Direct Memory Access (RDMA)** communication framework!
Overview

- Remote Direct Memory Access
- Design of RUBIN
- Evaluation of RUBIN
- Conclusion
Remote Direct Memory Access

- **Zero-copy** communication protocol
- Kernel bypassing
- Data transfer directly into remote memory
- Applications **register memory** with RDMA NIC
- Message-oriented and asynchronous operations
- Often employed in **data centers**
Remote Direct Memory Access

- **Zero-copy** communication protocol
- Kernel bypassing
- Data transfer directly into remote memory
- Applications **register memory** with RDMA NIC
- Message-oriented and asynchronous operations
- Often employed in **data centers**

Low latency, high throughput, CPU efficient!
But possible **security issues** due to direct memory access?
RDMA Consensus Protocols

DARE (Poke et al, HPDC’15)
- RDMA-tailored SMR protocol
- Achieve low latency in replica communication

APUS (Wang et al., SoCC’17)
- Combine RDMA with Paxos
- Scalability regarding concurrent connections

Derecho (Jha et al., 2017)
- C++ library for replicated crash-fault tolerant services built on Paxos
RDMA Consensus Protocols

**DARE** (Poke et al, HPDC’15)
- RDMA-tailored SMR protocol
- Achieve low latency in replica communication

**APUS** (Wang et al., SoCC’17)
- Combine RDMA with Paxos
- **Scalability** regarding concurrent connections

**Derecho** (Jha et al., 2017)
- C++ library for replicated crash-fault tolerant services built on Paxos

→ Only crash faults are considered, **no previous work** on BFT! How to implement **RDMA communication for BFT frameworks**?
Requirements

① **Easy integration** into existing BFT prototypes

② **Security** guarantees even in the presence of malicious nodes

③ **Zero-copy** communication
Easy Integration

- RDMA communication for multiple BFT frameworks
  - BFT-SMArt (Bessani et al., DSN’14)
  - UpRight (Clement et al., SOSP’09)
  - Reptor (Behl et al., Middleware’15)

- BFT frameworks very complex, e.g. Reptor:
  - Core: 50,000 LOC (Java)
  - Deployment, benchmarking: 14,000 LOC (Python)

- High development effort
  - ≈20 years of BFT research
  - Limited number of BFT frameworks
Easy Integration

- RDMA communication for multiple BFT frameworks
  - BFT-SMART (Bessani et al., DSN’14)
  - UpRight (Clement et al., SOSP’09)
  - Reptor (Behl et al., Middleware’15)

- BFT frameworks very complex, e.g. Reptor:
  - Core: 50,000 LOC (Java)
  - Deployment, benchmarking: 14,000 LOC (Python)

- High development effort
  - ≈20 years of BFT research
  - Limited number of BFT frameworks

Direct integration is far too much overhead!
## Easy Integration

- BFT frameworks often written in **Java**
- Use **Java NIO** for high-performance communication
  - With clients (BFT-SMART), replicas (UpRight), or both (Reptor)
- Frameworks optimized to reduce data copy steps
- Need suitable level of abstraction
  - Not as low-level as the native RDMA interface
  - Not as high-level as JSOR: socket interface, but intermediate data copies by default (Thirugnanapandi, 2014)
1 Easy Integration

- BFT frameworks often written in **Java**
- Use **Java NIO** for high-performance communication
  - With clients (BFT-SMART), replicas (UpRight), or both (Reptor)
- Frameworks optimized to reduce data copy steps
- Need suitable level of abstraction
  - Not as low-level as the native RDMA interface
  - Not as high-level as JSOR: socket interface, but intermediate data copies by default (Thirugnanapandi, 2014)

→ Modeled after **Java NIO**
→ Interface similar to **Java socket interface**
→ Easy switch between **RDMA and TCP** communication
Security: RDMA Semantics

Read/Write

- Used in APUS and DARE
- Fastest communication mode
- Exchange memory key specifying buffer location
- Receiver not notified
- Security risks in BFT setting: get memory key, corrupt memory

---

Server A

- RNIC
- Data Buffer

RDMA Write(data, key)

memory key exchange

Server B

- RNIC
- Data Buffer

RDMA Receive

---
## Security: RDMA Semantics

### Read/Write
- Used in APUS and DARE
- **Fastest** communication mode
- Exchange memory key specifying buffer location
- Receiver not notified
- **Security risks** in BFT setting: get memory key, corrupt memory

### Send/Receive
- **Two sides** active
- Receiver notified
- **No known** memory key
- Remote memory locations decided by application
  \[\rightarrow\] No memory corruption!

![Diagram](image-url)
Read/Write
- Used in APUS and DARE
- Fastest communication mode
- Exchange memory key specifying buffer location
- Receiver not notified
- Security risks in BFT setting: get memory key, corrupt memory

Send/Receive
- Two sides active
- Receiver notified
- No known memory key
- Remote memory locations decided by application
  → No memory corruption!

Send/Receive has higher security
  → no memory corruption and MitM attack possible!
Our Framework: **RUBIN**

- Modeled after **Java NIO** and socket interface
- **Integration** in several BFT frameworks possible
- Use RDMA **Send/Receive** semantics for security
- Integrate into **Reptor** framework
- Use **DiSNI** library for RDMA communication in Java
Rubin Components

- **RDMA Channel**: Java NIO SocketChannel with RDMA resources
**RUBIN Components**

- **RDMA Channel**: Java NIO SocketChannel with RDMA resources
- **RDMA Selector**: efficiently handle multiple channels with one thread
  - Select channels that are ready for certain events
  - Avoids expensive context switching
Rubin Components

- **RDMA Channel**: Java NIO SocketChannel with RDMA resources
- **RDMA Selector**: efficiently handle multiple channels with one thread
  - Select channels that are ready for certain events
  - Avoids expensive context switching
- **RDMA Selection Keys**: channel operation
  - Send, receive message, connection establishment
Workflow of Rubin

1. Channel registration, set interest

- Channel registration
- Set interest

List of selection keys with RDMA interest set

Registered

RDMA Selector

Event Manager

Hybrid Event Queue

DiSNI library

Processing communication events

Processing completion queue events
Workflow of Rubin

1. Channel registration, set interest
2. Selection Key creation
Workflow of Rubin

1. Channel registration, set interest
2. Selection Key creation
3. Non-/Blocking select()
 Workflow of Rubin

1. Channel registration, set interest
2. Selection Key creation
3. Non-/Blocking select()
4. Hybrid event queue, notify selector
Workflow of Rubin

1. Channel registration, set interest
2. Selection Key creation
3. Non-/Blocking select()
4. Hybrid event queue, notify selector
5. Select responsible channel
3 Zero-Copy Communication

- Pool of pre-allocated RDMA-registered application buffers
- Optimization: selective signaling to reduce notification overhead

→ Challenge: Buffer Copy
- Sender: register application buffers, no buffer copy
- Receiver: copy data to application buffer due to incompatibility
  → DiSNI requires direct buffers, but also heap buffers used in Reptor
Evaluation Setup

- 2 server machines: 4-core Xeon v2 CPUs and 16GB RAM
- 10Gbps switched network
- Mellanox ConnectX-3 RDMA NICs

Q1: How does RDMA communication compare to TCP?
Q2: What is the performance of RUBIN?
Evaluation Setup

- 2 server machines: 4-core Xeon v2 CPUs and 16GB RAM
- 10Gbps switched network
- Mellanox ConnectX-3 RDMA NICs

Q1: How does RDMA communication compare to TCP?
Q2: What is the performance of Rubín?

Echo server application:
- Q1: Distributed microbenchmark for RDMA Channel
- Q2: Local microbenchmark for Rubín in Reptor communication stack
RDMA Microbenchmarks – Latency

- RDMA Channel 33 – 43% lower latency than TCP
- Optimizations: 30% less latency than Send/Recv for messages <16KB
RDMA Microbenchmarks – Latency

- **RDMA Channel** 33 – 43% lower latency than TCP
- **Optimizations:** 30% less latency than Send/Recv for messages <16KB

Performance degradation due to remaining buffer copy
**RDMA Microbenchmarks – Throughput**

- RDMA Channel **33 – 43 %** higher throughput than TCP
- Optimizations: **30 %** higher throughput than Send/Recv
Rubin Microbenchmarks – Latency

- 1KB, 100KB: **19 – 20%** lower latency
- 20KB – 80KB: **20%** higher latency
Rubin Microbenchmarks – Throughput

- **Rubin** has **25 – 38%** higher throughput than TCP

Limited by **buffer copy** → remove and optimize!
Future Work

- **Zero-copy**: remove any additional data copy steps
- Reptor: evaluate **fully replicated system** with RUBIN communication
- Integration of Reptor into a **permissioned blockchain framework**
  - E.g. Hyperledger Fabric
Conclusion – RUBiN

- RDMA framework for BFT protocols
- High-level abstraction to maintain flexibility
- Easy integration: modeled after Java NIO interface
- Up to 25 – 38% higher throughput
- Next: RDMA-capable BFT ordering service in permissioned blockchain setting
Conclusion – RUBIN

- RDMA framework for BFT protocols
- High-level abstraction to maintain flexibility
- Easy integration: modeled after Java NIO interface
- Up to 25 – 38% higher throughput
- Next: RDMA-capable BFT ordering service in permissioned blockchain setting

Questions?
ruesch@ibr.cs.tu-bs.de
Backup – RDMA Communication

- OS only used to establish connection
- Queue Pair: send/receive queue holding work requests
- Work Request: information about data to be sent/received
- Completion Queue: holds events notifying application about finished operation
Backup – Reptor Buffer Management

- DiSNI requires direct buffers in native memory
- Reptor uses both direct buffers and heap buffers in JVM memory
- Remote side needs pre-prepared buffers to receive data via RDMA
- Reptor has complex buffer management scheme, often replacing buffers
  \[ \rightarrow \text{Redesign parts of buffer management} \]
Backup – Reptor

- BFT framework implementing both PBFT and Hybster
- Hybster: hybrid BFT protocol with TSS using Intel SGX
- Consensus-oriented parallelization: parallel execution of consensus instances
Backup – Security Analysis

- RDMA mechanisms: Protection Domains and memory access permissions
- Security issues mostly relevant for Read/Write communication
- Read/Write: node reads data while it is overwritten → data corruption
- Steering Tag:
  - Buffer identifier
  - MitM attacks
  - Invalidate tag to prevent legitimate access
Backup – TCP Overhead

(Frey et al., ICDCS’09)
ETB Technologies. Dell Mellanox CX324A CONNECTX-3 40Gb QSFP+ Dual Port Low Profile NIC - M9NW6. https://goo.gl/Z8pVbM

mcwiggin. Datto Data Center Shots. https://goo.gl/xZCPxd