DINT: Fast In-Kernel Distributed Transactions with eBPF

* Yang Zhou  
* Co-primary author  
Harvard University

* Xingyu Xiang  
Peking University

Matthew Kiley  
Harvard University

Sowmya Dharanipragada  
Cornell University

Minlan Yu  
Harvard University
Distributed Protocols for High Availability

Hadoop

Kubernetes

YouTube

Electrode [NSDI’23]

Etcd running distributed consensus protocols

Spanner running distributed transaction protocols

This work
Distributed Transactions inside a Datacenter

- **OCC transaction** clients
  - Backups
  - Primary

- **txn_read_lock**(R, W) // read read-set (R) versions + values, // lock write-set (W)
  // Update write-set locally
  - **txn_validate**(R) // validate read-set versions
  - **txn_log**(W) // log write-set updates
  - **txn_commit** // update backups+ primary

**Trend:** storing states **in memory** or persistent memory

Distributed transactions are **network IO-intensive**
Kernel Networking: High Kernel Overhead

sendto (int sockfd, const void *buf[len], size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addrlen) -> ssize_t

event_SYSCALL_64_after_hwframe
do_syscall_64
__x64_sys_sendto
__sys_sendto
sock_sendmsg
inet_sendmsg
udp_sendmsg
ip_make_skb
ip_route...
udp_send_skb
ip_send_skb
ip_output
ip_finish_output
ip_finish_output
neigh...
eigh_hh_output
dev_queue_xmit
dev_queue_xmit
dev_xmit_skb
sch direct xmit
dev_hard_start_xmit
mlx5e_xmit

90% CPU time
Kernel Networking: High Kernel Overhead

[1] Experiment setting: TATP workload for the OCC transaction protocol with 3-way primary-backup replication and 3-way sharding, using UDP sockets from Linux kernel 6.1.0

Dist. protocols
- Socket layer
- Transport layer
- ... (omitted)
- Traffic control
- NIC driver
- NIC hardware

OCC distributed transactions

92% CPU time

Only around $\frac{1}{10}$ is on NIC driver
Kernel Bypass: Not a Panacea

DPDK (Data Plane Development Kit) or RDMA:
- Customized networking stacks in user space or NIC
- Busy polling instead of costly interrupt
Kernel Bypass: Not a Panacea

**Dist. protocols**

- Transport layer
- ...
- User-space driver

**Kernel net. stack**

- NIC hardware

**DPDK (Data Plane Development Kit) or RDMA:**

- High performance
- Dedicated resources (eg, busy-polling cores)
- Security vulnerabilities (user manages NICs)\(^1,2\)

---

DINT\textsuperscript{1}: Application-Customized Networking Stacks

Dist. protocols
- Socket layer
- Transport layer
- Traffic control
- NIC driver
- NIC hardware

Application logic
- Protocol requests
- Protocol responses

Packet batching amortizes interrupt overhead (eg, Linux NAPI)

Application states
- Update

- High performance
- Resource sharing: interrupt-driven
- Secure: kernel manages NICs

[1] DINT: an archaic word, meaning force and power
How to Guarantee Kernel Safety?

eBPF (extended Berkeley Packet Filter) to safely run programs in kernel at runtime

- Guaranteeing safety via static verification
- Originally for packet filtering and monitoring

---

![Diagram]

**eBPF-offloaded logic**

- Protocol requests
- Modify and forward back as responses

**Dist. protocols**
- Socket layer
- Transport layer
- ... Traffic control
- NIC driver
- NIC hardware

**eBPF maps** (eg, array, queue)

**Update**
Challenge of Kernel Offloads with eBPF

• eBPF programming model is **constrained** because of static verification
  - Limited # of instructions, bounded loops, static memory allocation

• Distributed protocols are **complex**
  - Some rare cases are too complex for eBPF: eg, failure, message loss, malloc

Offload common cases, which tend to be “simple”  
Leave rare cases to user space

DINT Overall Architecture

- Request parser
- Transaction server
- Bookkeeping
- Lock manager
- KV store
- Log manager
- Maintaining spilled KVs
- UDP sockets

Common path
Rare path

Transaction Client
Request
Response
Kernel space
User space
Offloading Lock Manager

OCC transaction clients

Fail-and-retry (rare cases)

Hash(LockID)

Compare-and-swap atomics

Hash(LockID’)

Primary

Kernel space

Shared eBPF array

Lock state

Waiting queue?

Hard for eBPF (no malloc)
Offloading KV Store

Common cases: most KVs are small in typical workloads
- Dozens of bytes in transactional workloads (eg, TATP, SmallBank)
- Statically-allocated eBPF map to store small KVs and avoid malloc

Can we achieve fast GET and PUT?
How to achieve **fast GET** especially for non-existing keys?

GETs for non-existing keys always check user space

- **Kernel bucket**
- **User bucket**
- **Bloom filter**
- **BF records**

**Bloom Filter**:
- An approximate data structure that quickly tells whether a key is in a set
- Using a **fixed size** of bitset array

---

Offloading KV Store

How to achieve fast GET especially for non-existing keys?

- User bucket
- Kernel bucket
- Bloom filter
- BF records overflow keys

KV deletion?

Reconstruct BF upon deletion

Removes unnecessary checks for non-existing keys
Offloading KV Store

How to achieve fast GET especially for non-existing keys?

Bloom filter to record overflow keys

- Write-back cache for fast PUT
- Lock sharing for fast locking
- Per-core circular logs for fast logging
- Piggybacking states on packets for fast user-kernel synchronization
- ...

Offloading Log Manager

User space replays the logs on failure (rare cases)

Primary
Kernel space

Per-CPU eBPF ringbufs (overwritable)
System Optimization: Interrupt Scheduling

Separating interrupt-handling cores and rare-case handling cores

⇒ Avoiding user-kernel context switching overhead

Kernel space

User space

Interrupts

Handling rare cases

2.4× throughput
Implementation & Evaluation

• 2.1K lines eBPF and 4.3K C++ for OCC and 2PL distributed transactions
  – 3-way primary-backup replication, 3-way sharding
  – 6.1K lines of C++ for baselines

• Experiment setup:
  – CloudLab r650 (10 clients, 3 servers) running unmodified Linux kernel 6.1.0
  – TATP for OCC, SmallBank for 2PL

• Open source: https://github.com/DINT-NSDI24/DINT
Tail Latency vs. Throughput

- Linux Kernel
- DINT
- Caladan (kernel-bypass)

Packet copy
- 16× higher tput over kernel
- 1.9× higher tput over Caladan
- 16% higher tail latency

Throughput (Mtps)

99th-tail latency (µs)

22× on SmallBank
Interrupt-core consolidation may help
DINT Conclusion

We enable application-customized kernel networking stacks with:

- eBPF offloads for common cases, while user space for rare cases
- distributed transaction offloads, but generalizable to many distributed protocols

Kernel bypass: high performance

Kernel: resource sharing, security

Thank you!
yangzhou@g.harvard.edu