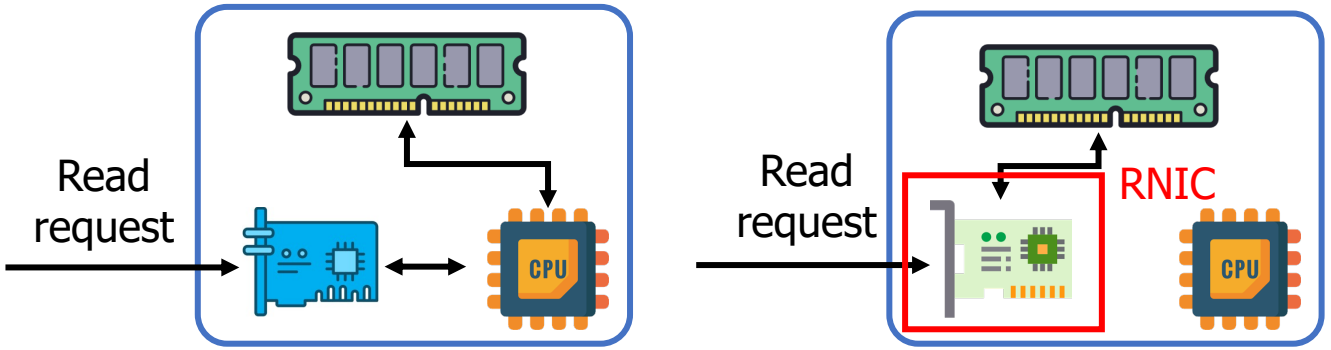
Bedrock: Programmable Network Support for Secure RDMA Systems

Jiarong Xing, Kuo-Feng Hsu, Yiming Qiu,
Ziyang Yang, Hongyi Liu, Ang Chen

RDMA: Remote Direct Memory Access



Traditional protocols

RDMA

RDMA over Converged Ethernet v2 (RoCEv2)

Azure

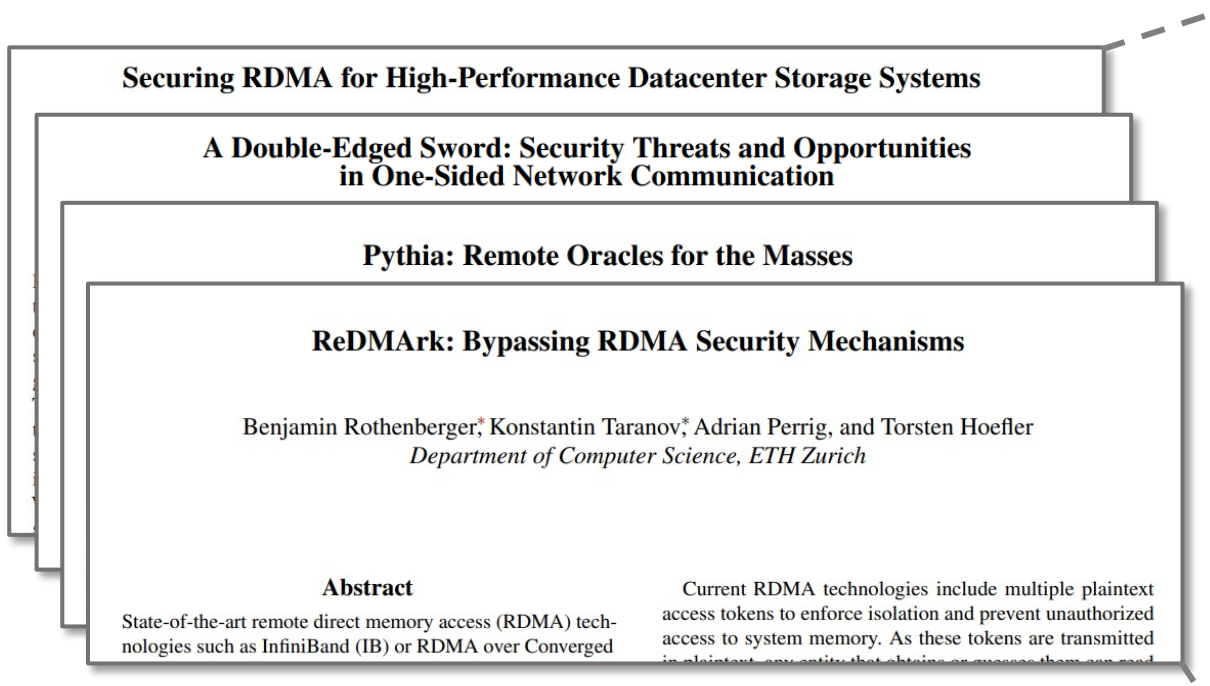
aws

Alibaba Cloud

Google Cloud

- Traditional protocols involve software processing on remote CPUs
- RDMA enables reading/writing remote memory with CPU bypassing
  - RNIC (RDMA NIC) performs DMA to application's memory
  - Can run over Ethernet with RoCEv2
  - Has been widely deployed in clouds

Motivation: RDMA is insecure

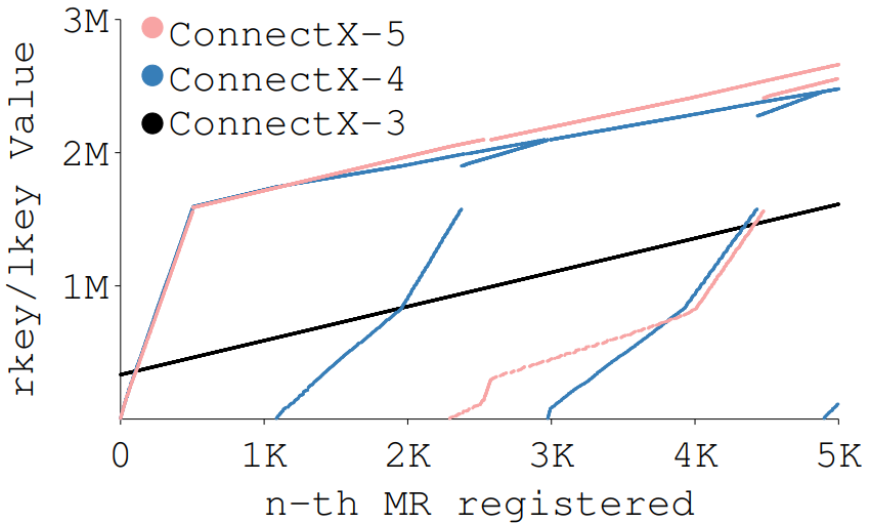


Inadequate security feature	#	Attack type	Attack method
Source authentication	S1	Illegal memory access	QP spoof
	S2	Illegal memory access	Packet injection
	S3	Denial of service	Seq number error
	S4	Denial of service	QP error
Access control	S5	Illegal memory access	Access control bypass
Monitoring and logging	S6	Denial of service	QP exhaustion
	S7	Perf. degradation	BW exhaustion
	S8	Side channel	Evict + Reload
	S9	Data exfiltration	RDMA read

- RDMA is designed for private usage with minimal security support
  - Designed for HPC in private clusters; widely deployed in public clouds now
- Recent studies have revealed lots of vulnerabilities

Insufficient RDMA-native security support

RDMA security mechanisms	Vulnerabilities	Length
Queue pair number (QPN)	Not randomly generated	24 bits
Remote memory access key (rKey)	Not randomly generated	32 bits
Packet sequence number (PSN)	Fixed initial values	24 bits

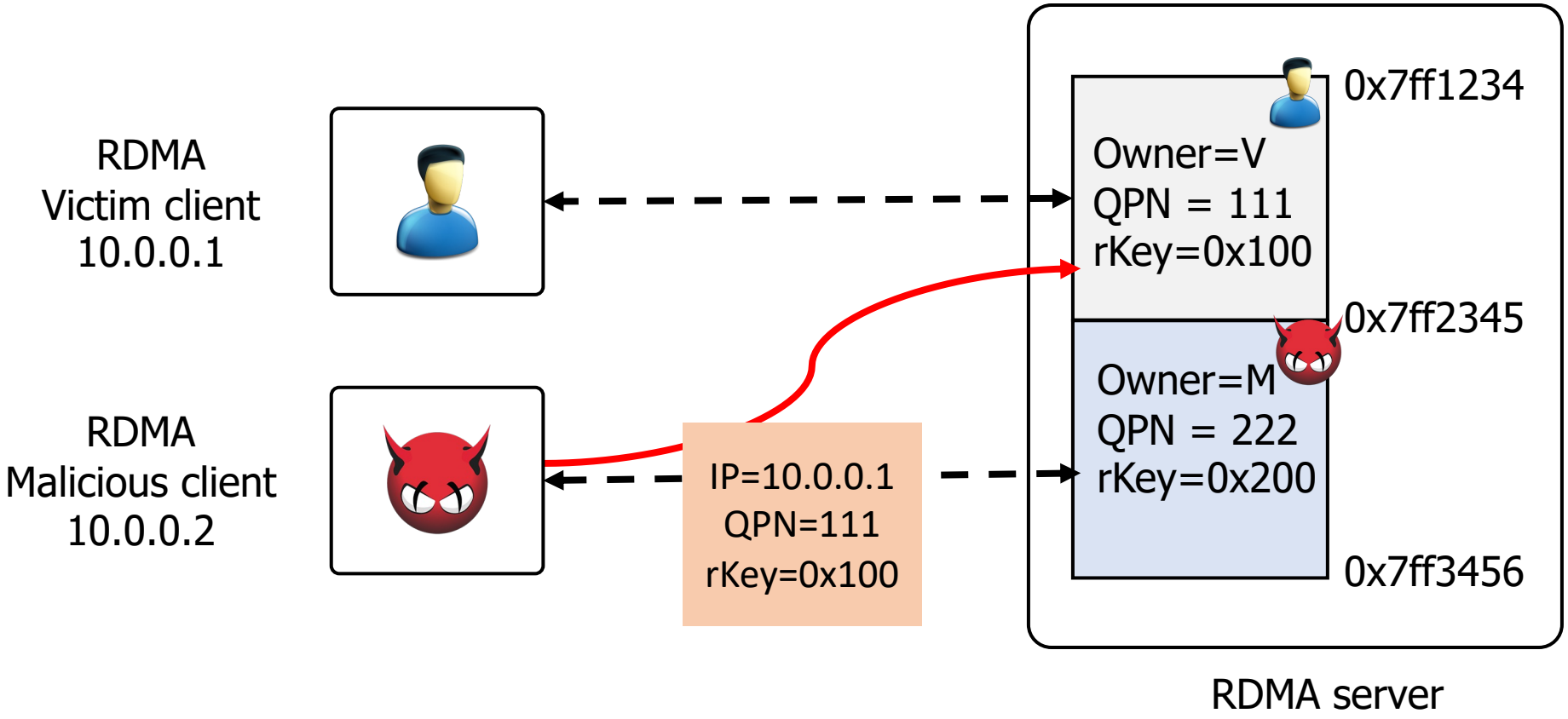


rKey value generated by RNICs [1]

- RDMA is unencrypted; uses simple credentials (e.g., QPN, rKey)
  - QPN and rKey are not randomly generated
  - PSN has fixed initial values
  - Easy to enumerate all possible values

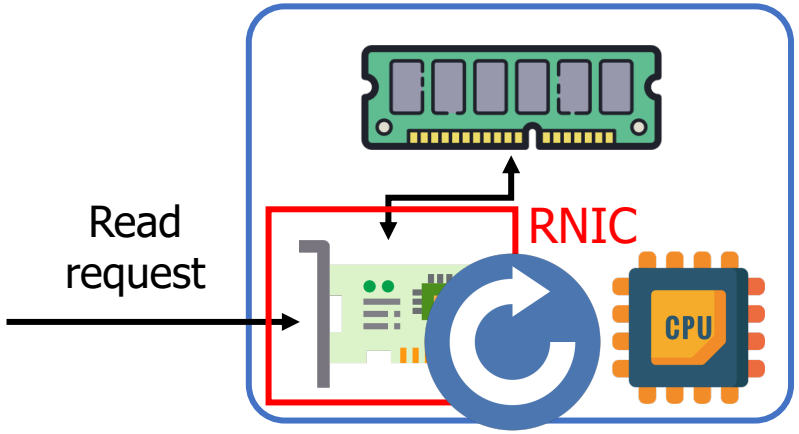
[1] Tsai, Shin-Yeh, and Yiying Zhang. "A Double-Edged Sword: Security Threats and Opportunities in One-Sided Network Communication." *(HotCloud 19)*.

Example: Illegal memory access



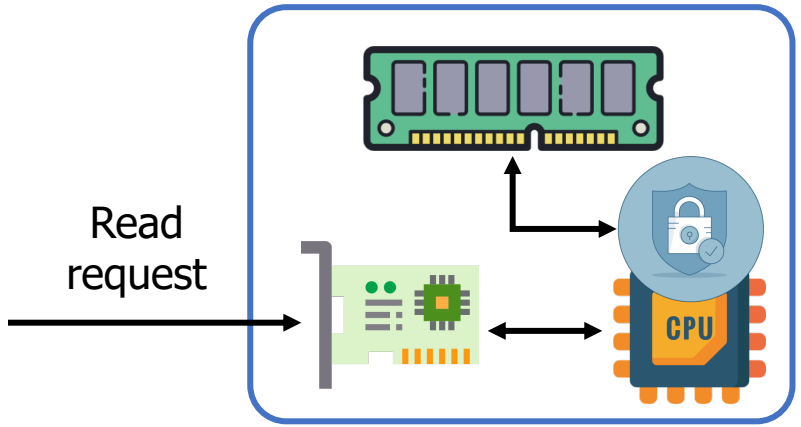
- An attacker can access memory owned by others with right credentials

Challenges of end host solutions



Redesign RDMA protocol and RNIC

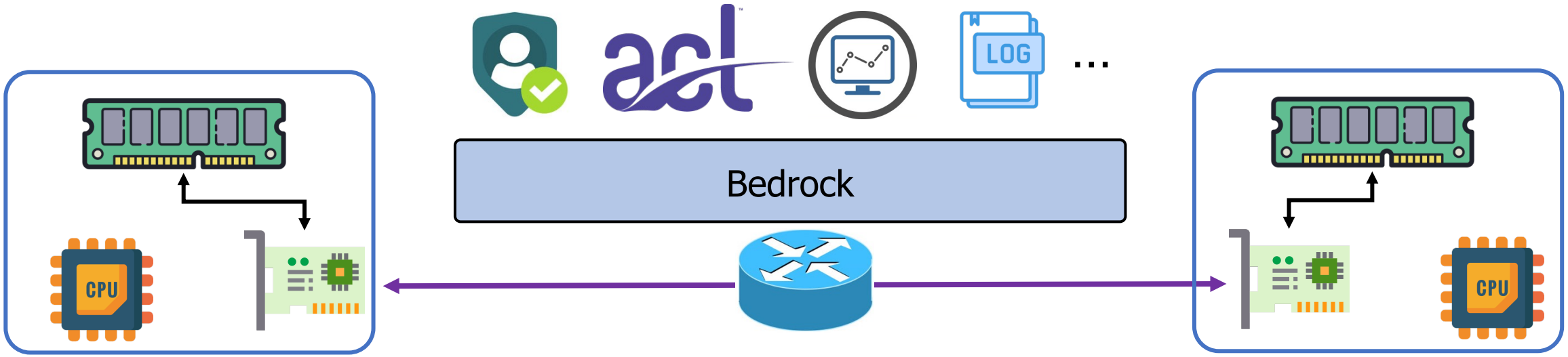
Requires intrusive changes of existing RDMA systems



Software security patch on CPUs

Negates the benefit of CPU bypassing

Bedrock: RDMA security support in the network



- Bedrock provides RDMA security support **in the network**
  - A platform with several security services
  - Can be extended to support more security defenses
  - Preserves CPU bypassing on the hosts
  - Can be deployed immediately

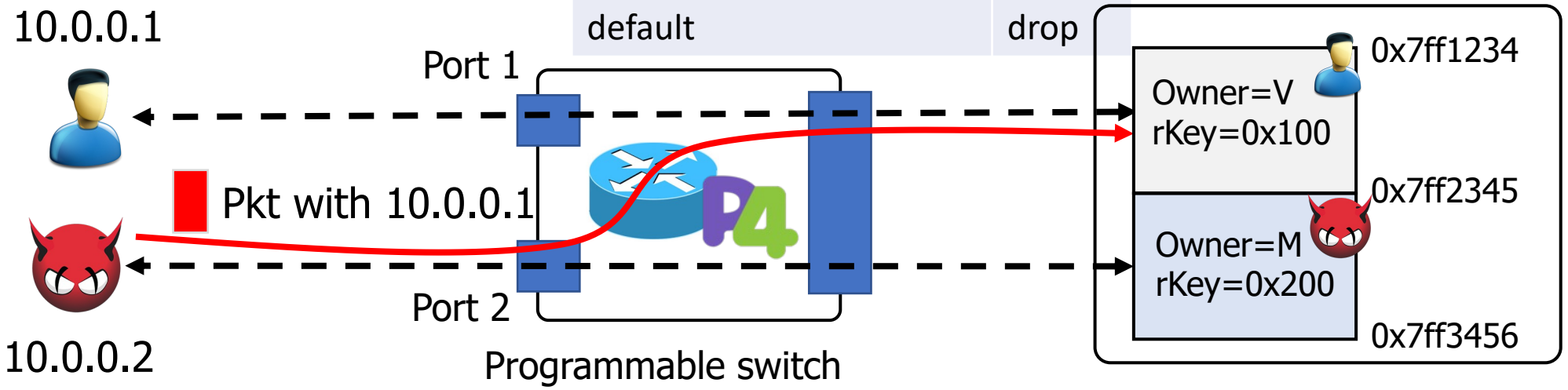
Key enabler: Programmable switches


```
table rdma_log_tab {
    key = {
      rdma.qpn:  exact;
      rdma.addr: range;
      rdma.op:   exact;
    }
    actions = {Log; DoNothing;}
}
```

- Programmable switches are commercially available
- Programmed with high-level language, e.g., P4
  - RDMA header parser, match-action tables, registers, ...
- Run at linespeed (Tbps)
- Have been widely used to build security defenses for traditional protocols:
  - Jaqen-Security'21, FlowLens-NDSS'21, Ripple-Security'21, etc.
- We use it to provide security support for RDMA.

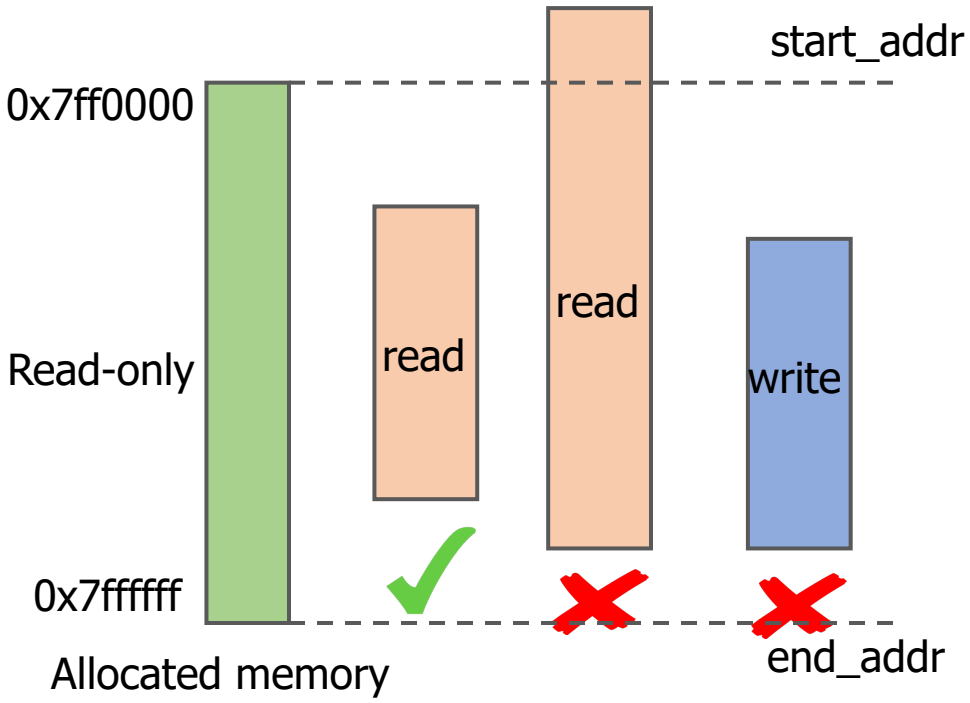
Source authentication using network invariants

Src IP	Incoming port	Action
10.0.0.1	1	accept
10.0.0.2	2	accept
...		
default		drop



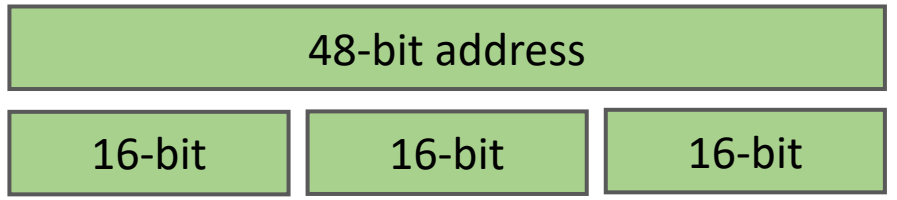
- Key idea: Using topological invariants beyond attackers' control
  - Cloud network is trusted; network operators control the invariants (e.g., incoming port)
- Spoofed packets coming from wrong topological location will be dropped
- Attackers on the same machine? Invariants from qp creation (using eBPF)

Flexible access control




```
table rdma_acl_tab {
    key = {
      rdma.s_addr: range;
      rdma.e_addr: range;
      rdma.op:     exact;
    }
    actions = {Allow; Deny;}
}
```

ACL table on the switch



Range partition

- Ensure requests accessing the memory correctly
- Determine containing relationship of two ranges
- Hardware limits make it challenging
  - Need to partition ranges to fit into the switch (20 bits per range field)
  - Rule compression (rule granularity, table decomposition, exploiting SRAM)

Regain visibility by monitoring and logging

Monitoring:

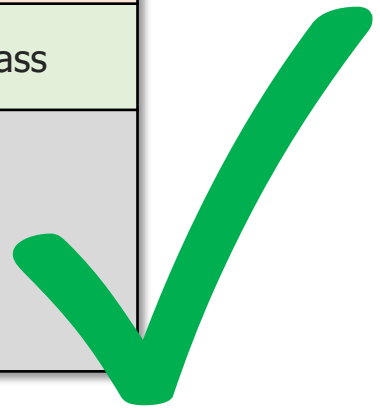
On-path traffic monitoring to detect anomaly

Logging:

Extract key info to logging servers for forensics

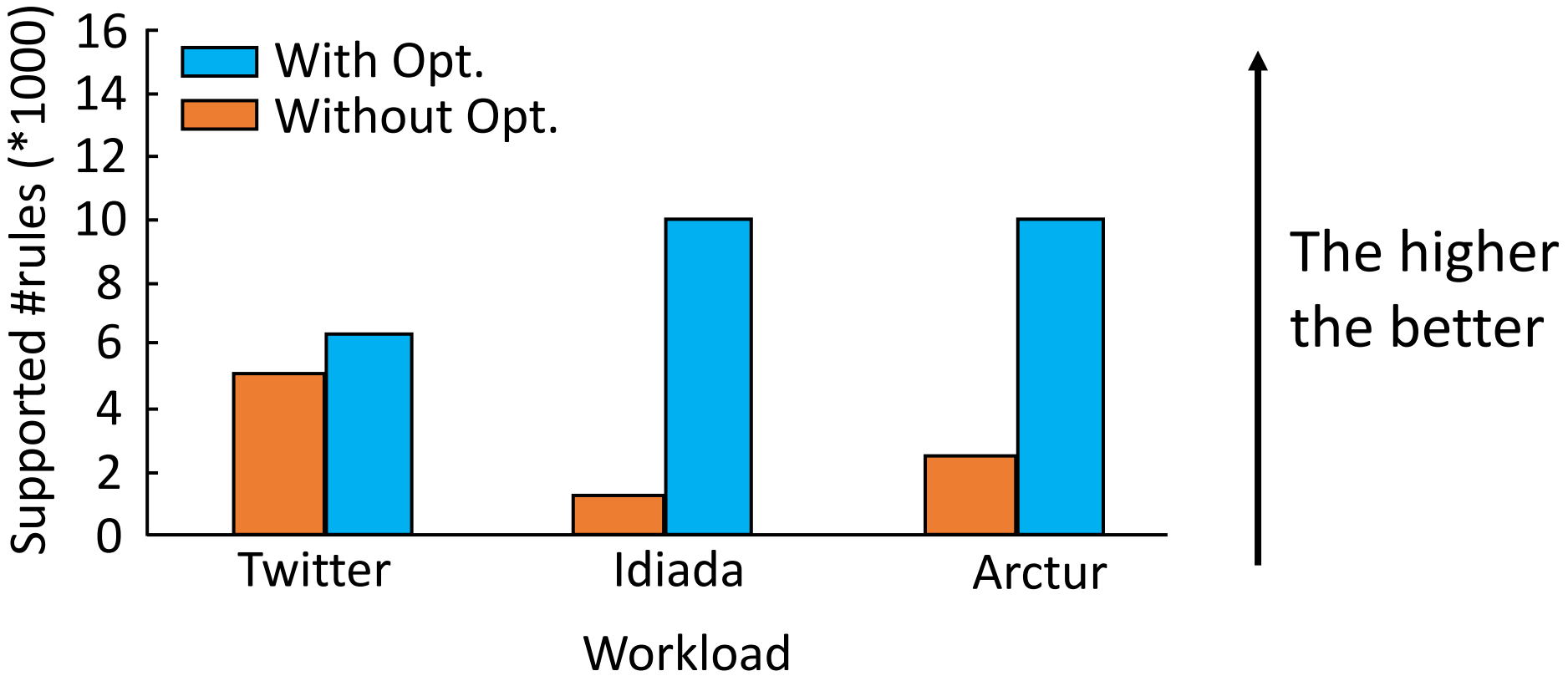
Bedrock defends against nine attacks

Inadequate security feature	#	Attack type	Attack method
Source authentication	S1	Illegal memory access	QP spoof
	S2	Illegal memory access	Packet injection
	S3	Denial of service	Seq number error
	S4	Denial of service	QP error
Access control	S5	Illegal memory access	Access control bypass
Monitoring and logging	S6	Denial of service	QP exhaustion
	S7	Perf. degradation	BW exhaustion
	S8	Side channel	Evict + Reload
	S9	Data exfiltration	RDMA read



- Experimental testbed:
  - NIC on all machines: Mellanox ConnectX-4 MT27710 25Gbps (with RoCEv2)
  - Switch: Wedge 100BF-32X Tofino switch

Access control rule compression



- Need to enhance security despite limited resources
- Naïve implementation without optimization only supports 5000 ACL rules
- Bedrock's optimizations increase ACL rules by 7x

Conclusion

- Motivation: Mitigating RDMA vulnerabilities
- Opportunity: Programmable switches
  - Complement the missing on-path defense for RDMA
- **Bedrock: In-network RDMA security suite**
  - Support authentication, access control, monitoring, logging
  - Effective against attacks
  - Immediately deployable
  - Extensible against future attacks
- Evaluation:
  - Mitigates nine RDMA attacks effectively with minimal perf. overhead!
- Source code: https://github.com/alex1230608/Bedrock

Thanks!