Enabling In-Network Computation in Remote Procedure Calls

Bohan Zhao*, Wenfei Wu**, Wei Xu*

**Tsinghua University, **Peking University*

清華大學
Tsinghua University

NSDI 2023

NetRPC: a General INC-enabled RPC System

- **Motivation:**

  In-network computation (INC) is beneficial to system performance but difficult to program

- **Goal:**

  Make INC easy to use for normal applications with little performance loss

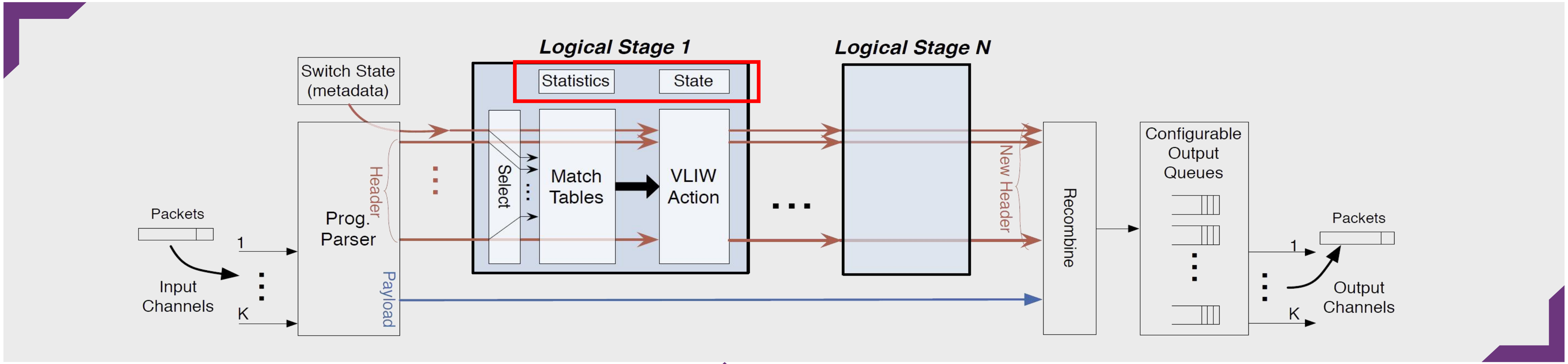
- **Metrics:**

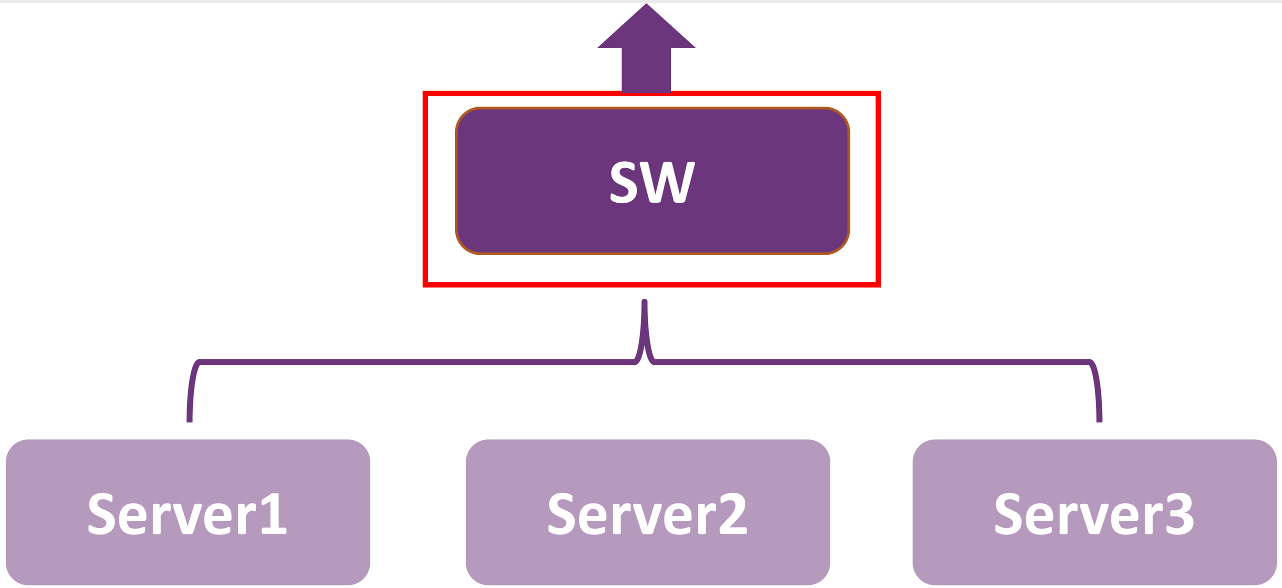
  Reduce lines of code of INC applications by up to 97%

  Support most popular INC appications

  Little performance loss

INC Customizes Stateful Packet Processing





INC is Widely Used in Many Scenarios

In-Network Computation

Advantages

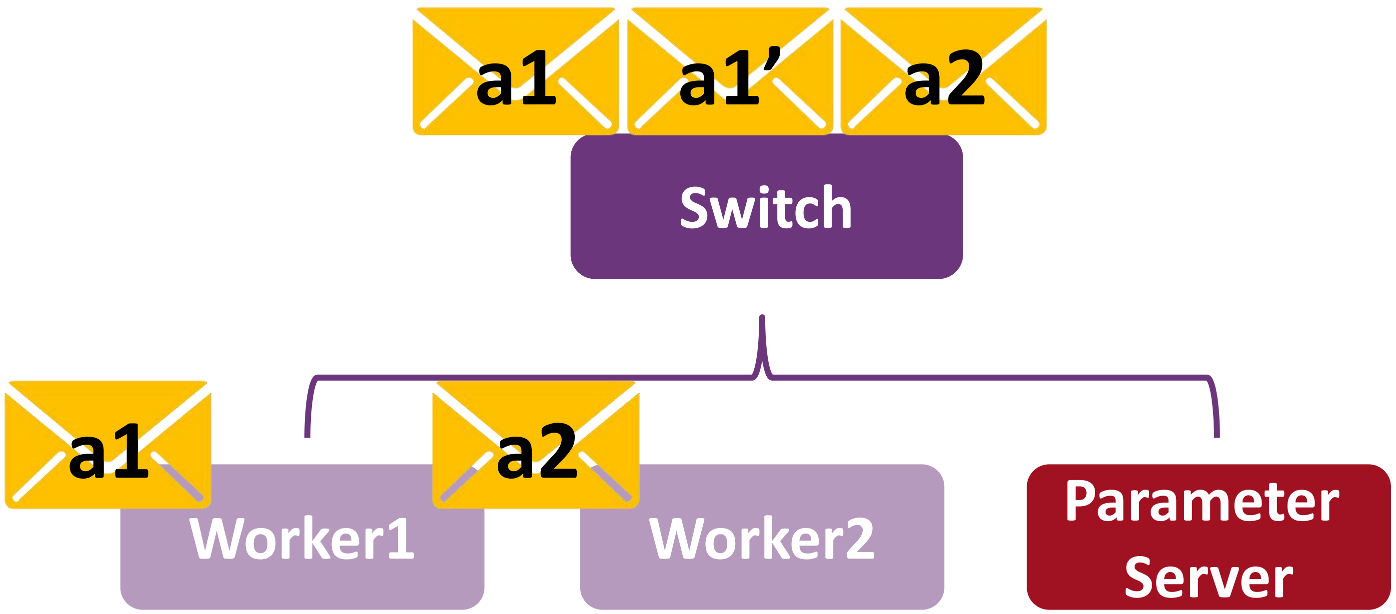
- Server Func Offloading
- Line-rate Computation
- Network Stack Simplification

Scenario

- Synchronous Aggregation
- Asynchronous Aggregation
- Key-value Caches
- Agreement

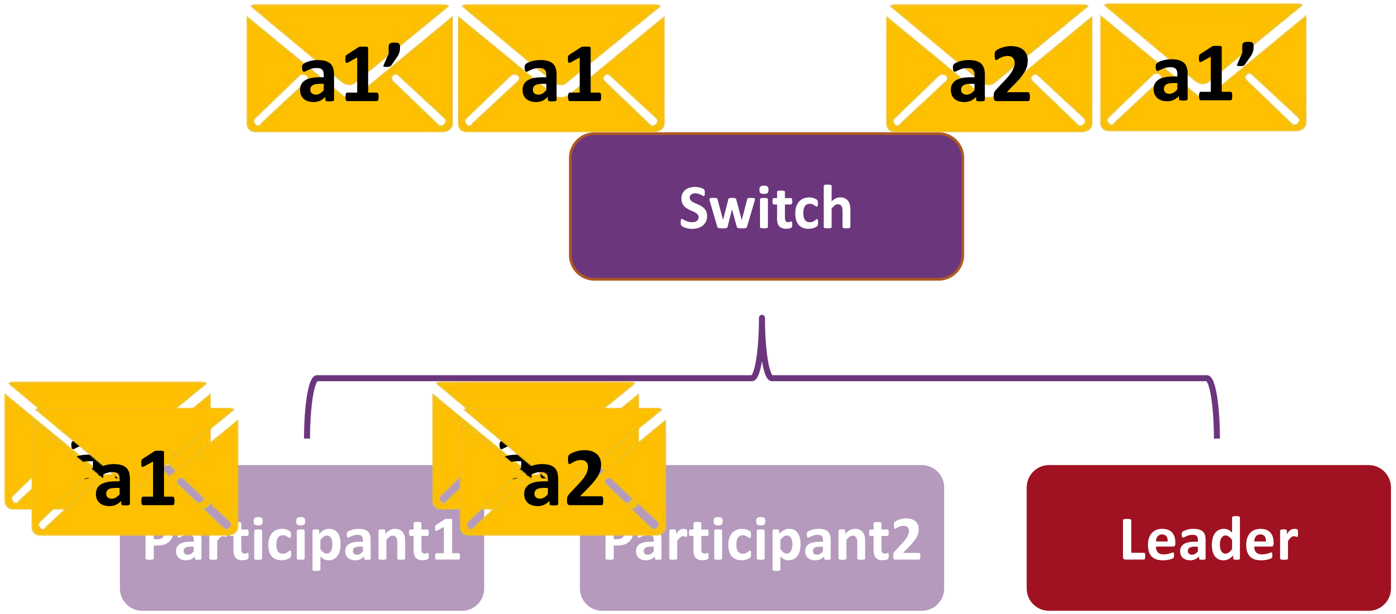
INC Provides Higher Throughput

- Eliminate incast to reduce traffic, especially for distributed training

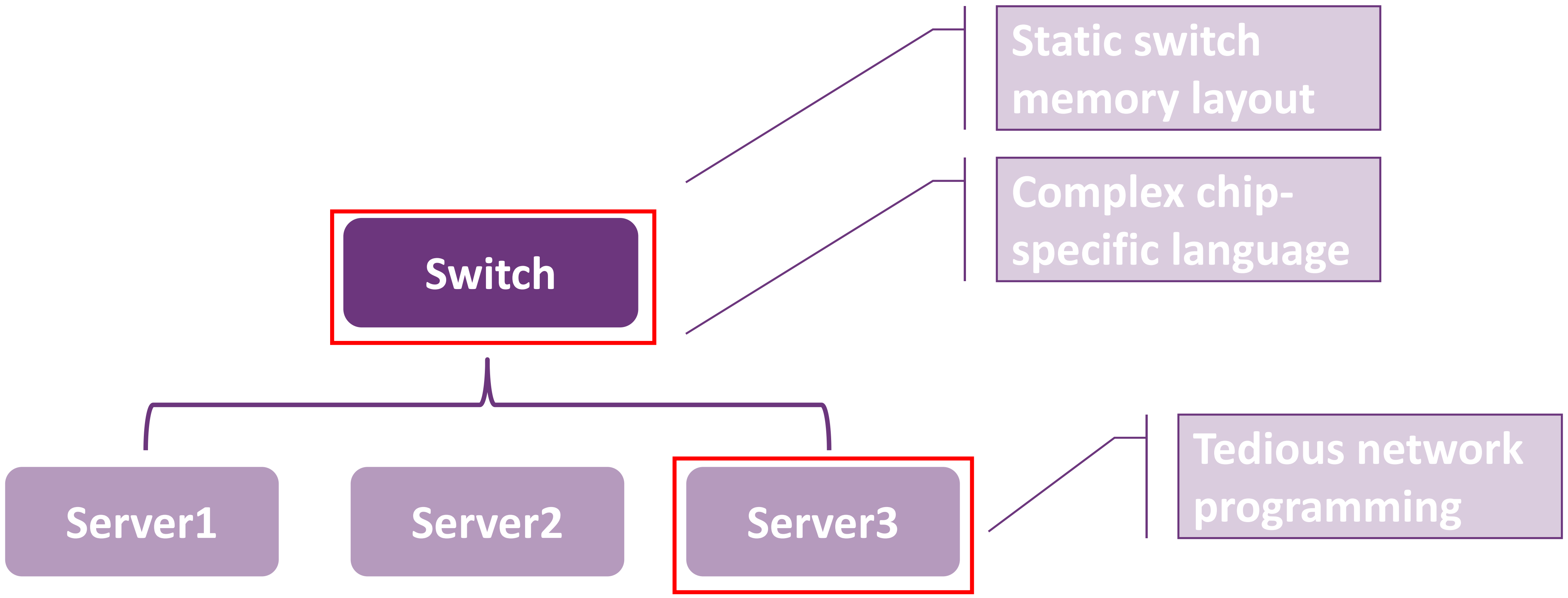


INC Provides Lower Delay

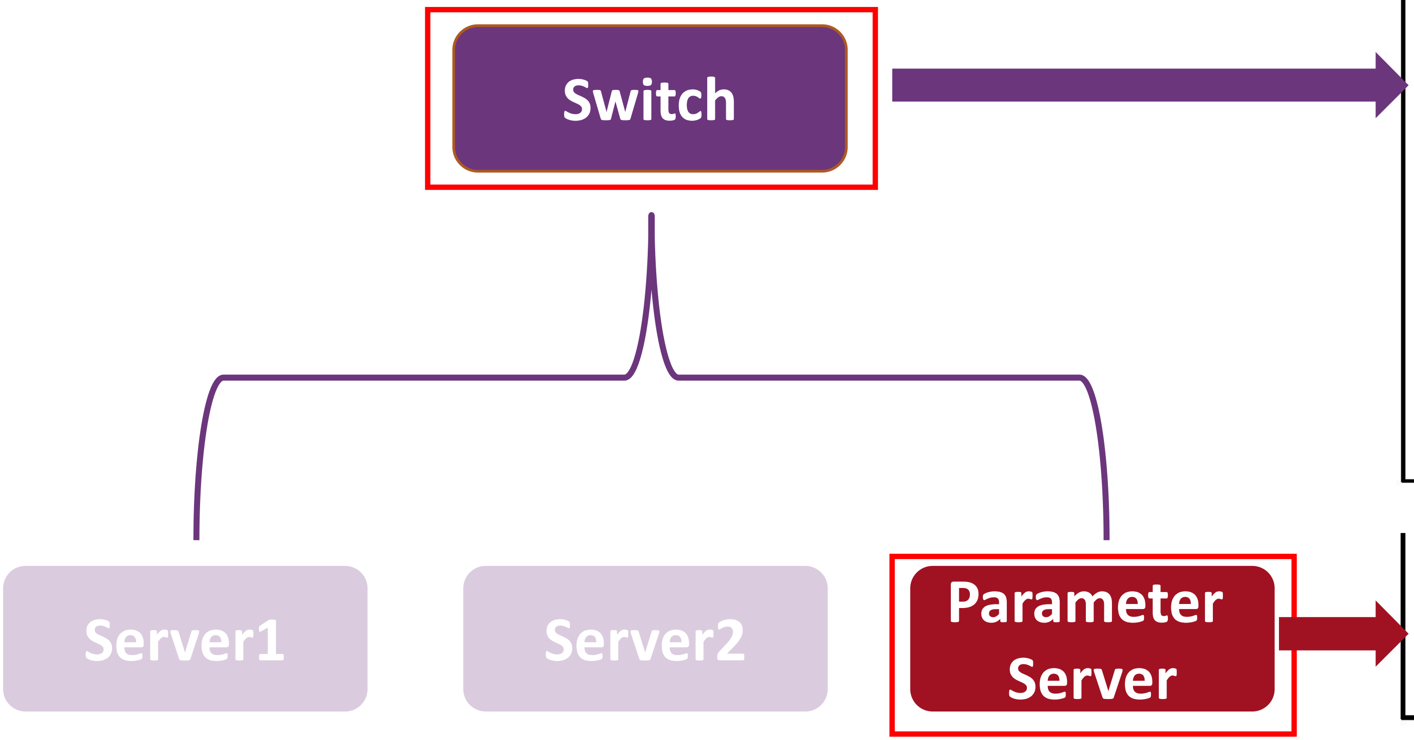
- Reduce the hops of round trip, useful for agreement applications



Challenges of Developing INC Application



P4 Programming is Much More Complex than Software

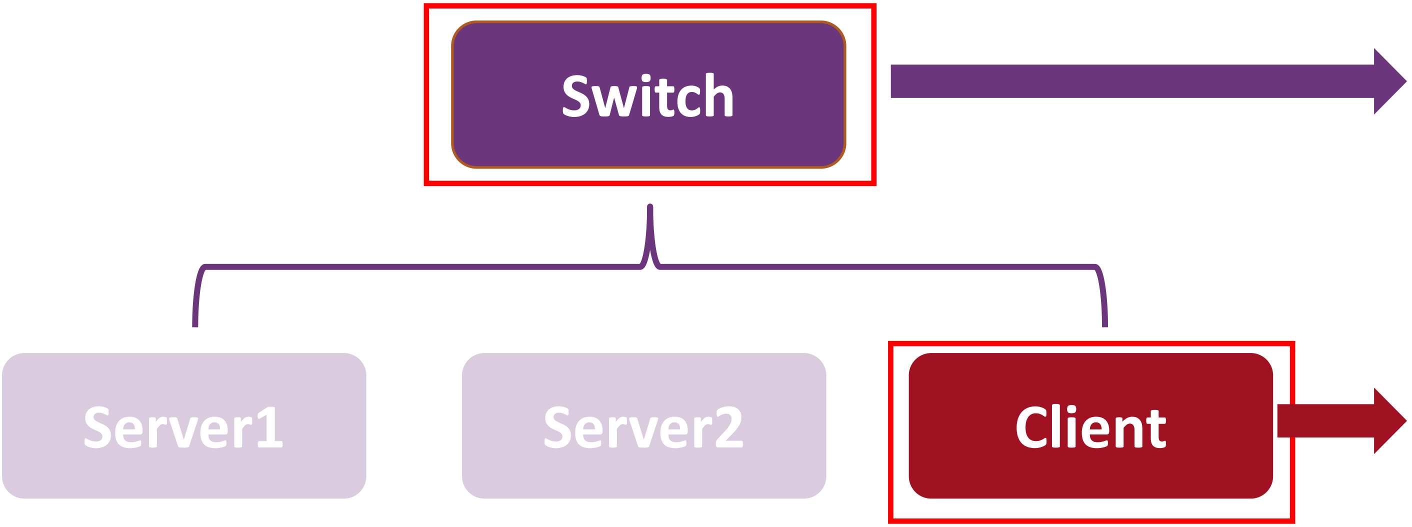



```
/* P4_14 Program */
action a_get_tmp() {
  subtract(tmp, smac, dmac);
}
action a_com_zero() {
  // do something
}
table get_tmp {
  actions { a_get_tmp; }
}
table com_zero {
  reads {tmp: exact}
  actions { a_com_zero; }
}
control com_smac_dmac {
  apply(get_tmp);
  apply(com_zero);
}
```

```
if (smac == dmac){
    // do something
}
```

Can We Provide a Computation-centric Programming Model to Include INC

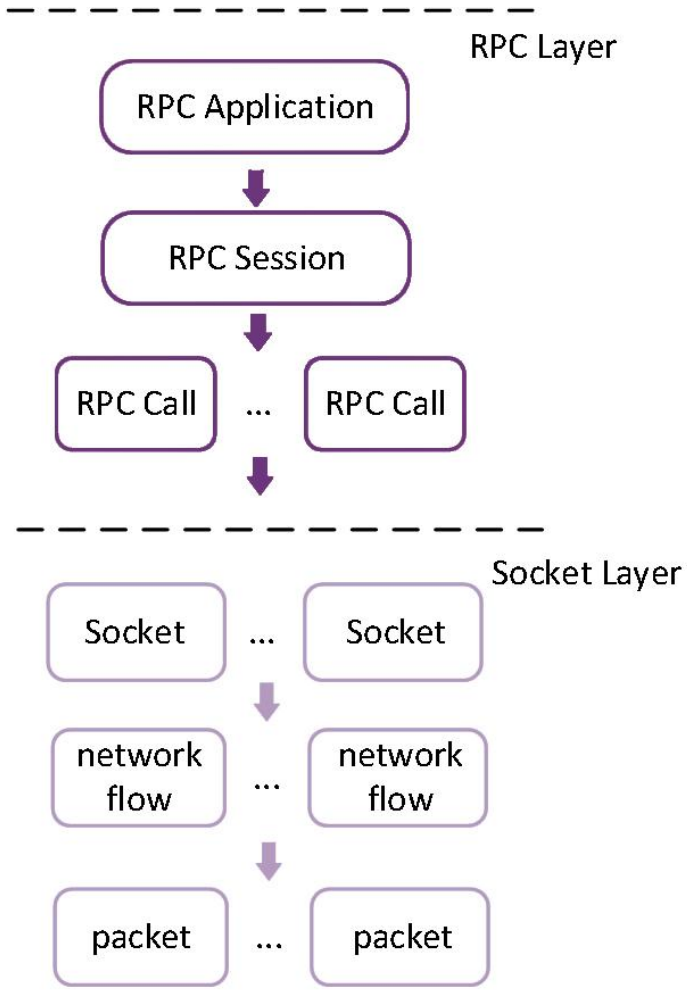
- P4 language is network-centric and focus on communication
- Users only take care of computation
- RPC adapts INC functions better than other models (e.g., MPI)




```
/* P4_14 Program */
action a_get_tmp() {
  subtract(tmp, smac, dmac);
}
action a_com_zero() {
```

```
void PushPull(double* data, int length) {
  NewGrad request;
  AgtrGrad reply;
  ClientContext context;
  request.mutable_tensor()->mutable_data()
        ->Add(data, data+length);
  Status status = stub_
        ->Update(&context, request, &reply);
  memcpy(data, reply.tensor().data(),
        length * sizeof(double))
  train(data);
}
```

Challenges in RPC-baed INC Programming



- Interface INC functions
- Support concurrent apps
- High-level data types
- Organize messages
- Reliable computation
- Switch memory management
- Reliable transmission
- Flow control

This Talk

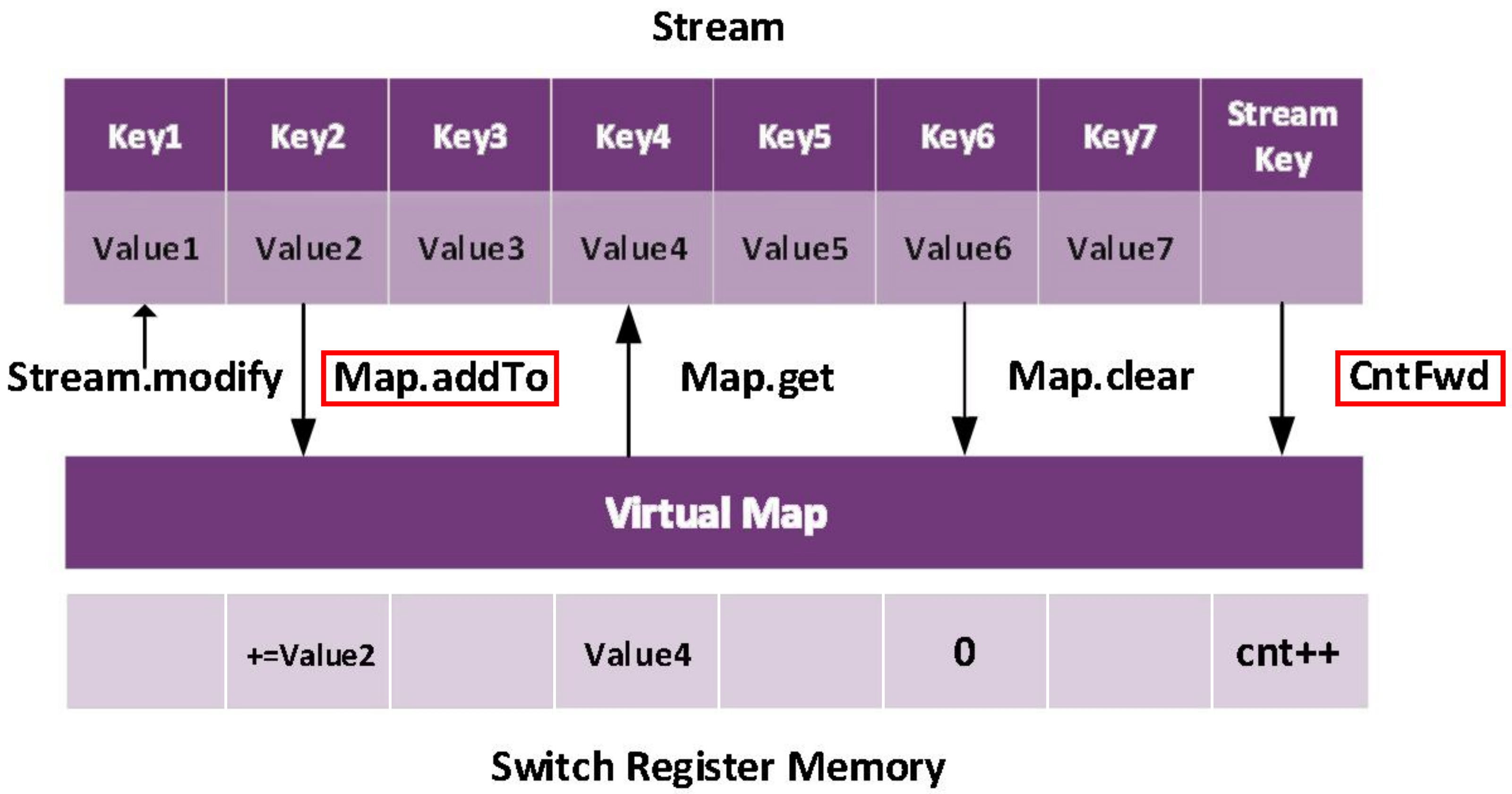
in the paper

Switch Program is Complex, but We Can Provide High-level Primitives

- We identify a minimum set of primitives to compose INC applications, named reliable INC primitives (RIPs)
- We hope to use the description of INC primitives (Netfilter) to replace switch programs

Primitive	Args	Semantics
Map.addTo	stream	map[stream.key]+= stream.value
Map .get	stream	stream.value = map[stream.key]
Map.clear	empty	map[stream.key] = 0
Stream.modify	op,para	stream.value = op(stream.value, para)
CntFwd	key,th,tgt	cnt[key]++; if cnt[key] == th then forward(tgt) else drop

We Implement RIPs Using Host and Switch Memory



NetRPC Programming Examples: Very Similar to gRPC

```
import "netrpc.proto"
message NewGrad {
  netrpc.FPArray tensor = 1;
}
message AgtrGrad {
  netrpc.FPArray tensor = 1;
}
service Training {
  rpc Update(NewGrad) returns (AgtrGrad)
        {} filter "agtr.nf"
}
```

Protobuf

```
{ //agtr.nf
  "AppName": "DT-1",
  "Precision": 8,
  "get": "AgtrGrad.tensor",
  "addTo": "NewGrad.tensor",
  "clear": "copy",
  "modify": "nop",
  "CntFwd": {
    "to": "ALL",
    "threshold": 2,
    "key": "ClientID",
  },
}
```

Netfilter

INC-enabled data types

Indicating NetFilter file name

```
shared_ptr<Channel> channel = CreateCustomChannel(server_ip,
    InsecureChannelCredentials());
unique_ptr<Stub> stub_(NewStub(channel));
void PushPull(double* data, int length) {
  NewGrad request;
  AgtrGrad reply;
  ClientContext context;
  request.mutable_tensor()->mutable_data()
        ->Add(data, data+length);
  Status status = stub_
        ->Update(&context, request, &reply);
  memcpy(data, reply.tensor().data(),
        length * sizeof(double))
  train(data);
}
```

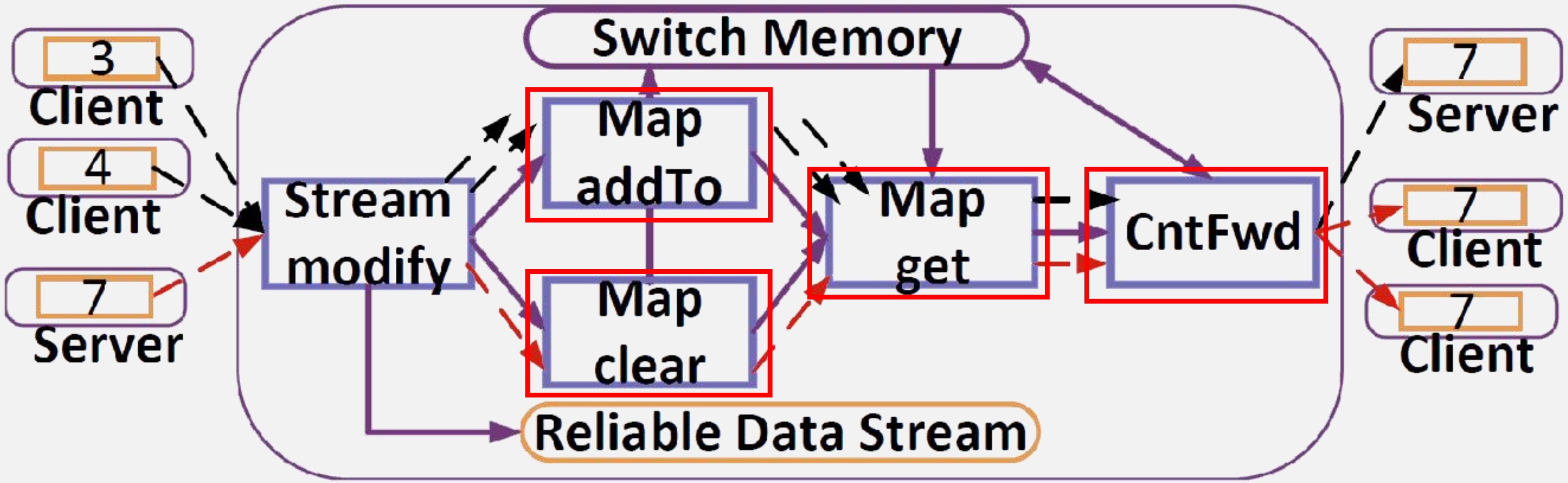
RPC

Quantization factor

Reliable INC primitives

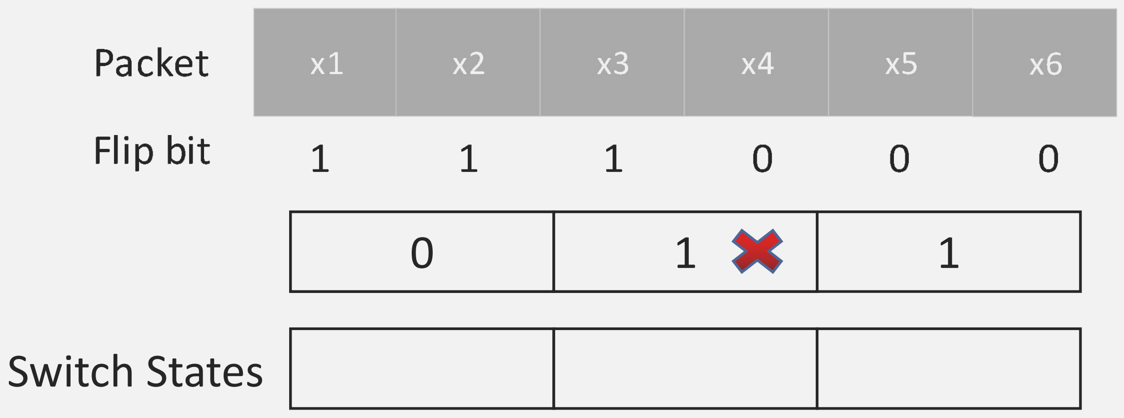
Support Concurrent INC Applications in One Switch

- We implement RIPs on the programmable switch to support multiple applications concurrently:



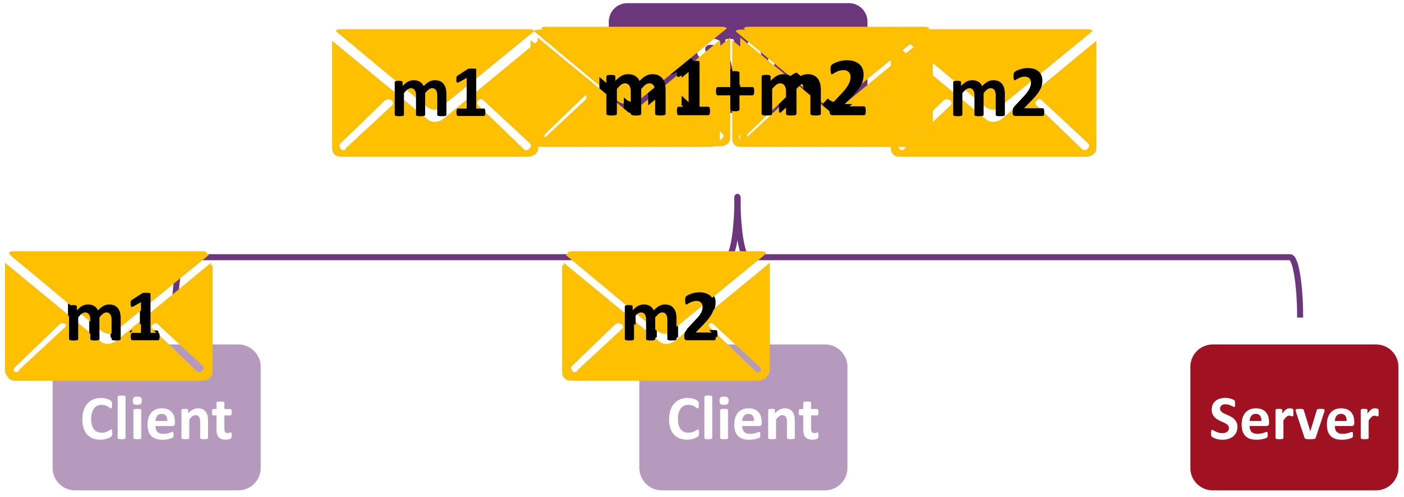
Reliable INC Requires Memory-Efficient Idempotence

- RPC calls should always succeed eventually, so RIPs should be same
- INC requires idempotence in addition
  a. Sockets only guarantee at -least-once packet transmission
  b. However, repetive accumulation on the switch causes incorrect result
  c. Normal path of some INC applications do not involve servers (on-switch reliability)
- We need to detect resent packets with limited switch memory



Reliable INC Requires Fallback to Fit RPC Calls

- INC can fail due to insufficient switch memory, computation overflow, etc.
- We implement all RIPs on the hosts. When INC fails, the RPC server can complete computation instead



Utilizing Switch Memory Efficiently Guarantees INC Benefits

- Sufficient switch memory makes INC full effect
- We need a management scheme to utilize switch resource efficiently
- We address switch memory in a key-value level by clients

Value Stream

1	2	3	4	5	6	7	Stream Key
Value1	Value2	Value3	Value4	Value5	Value6	Value7	

Pool-based Streaming

Value5	Value2	Value3	Value4
Value5	Value2	Value3	Value4

Utilizing Switch Memory Efficiently Guarantees INC Benefits

Key-value Stream

1	2	3	4	5	6	7	Stream Key
Value1	Value2	Value3	Value4	Value5	Value6	Value7	1

On-switch Cache

Value1	Value2	Value3	Value4
Key1	Key2	Key3	Key4
Value1	Value2	Value8	Value4

Server

Key-value Stream

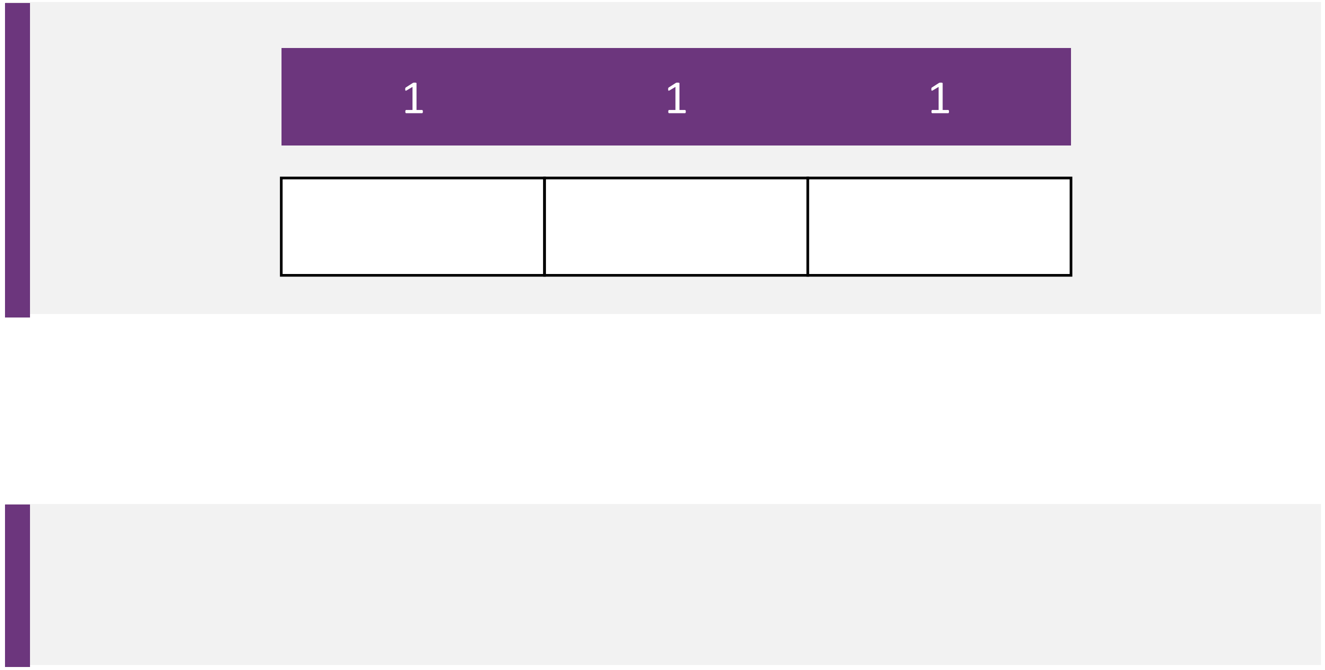
1	2	3	4	5	6	7	Stream Key
Value1	Value2	Value3	Value4	Value5	Value6	Value7	2

On-Host Addressing Requires Handling Client Crash

- NetRPC relies on hosts to manage switch memory correctly
- Memory leak happens when the client crashes and loses states
- We apply a two-phase timeout to recycle valuable switch memory

Phase-1 Timeout

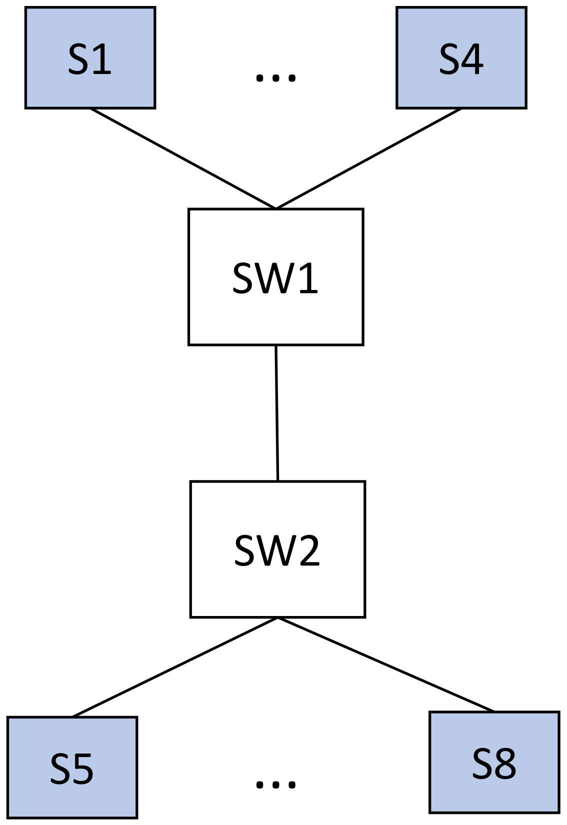
Phase-2 Timeout



Switch

Server

NetRPC Evaluation: Setup

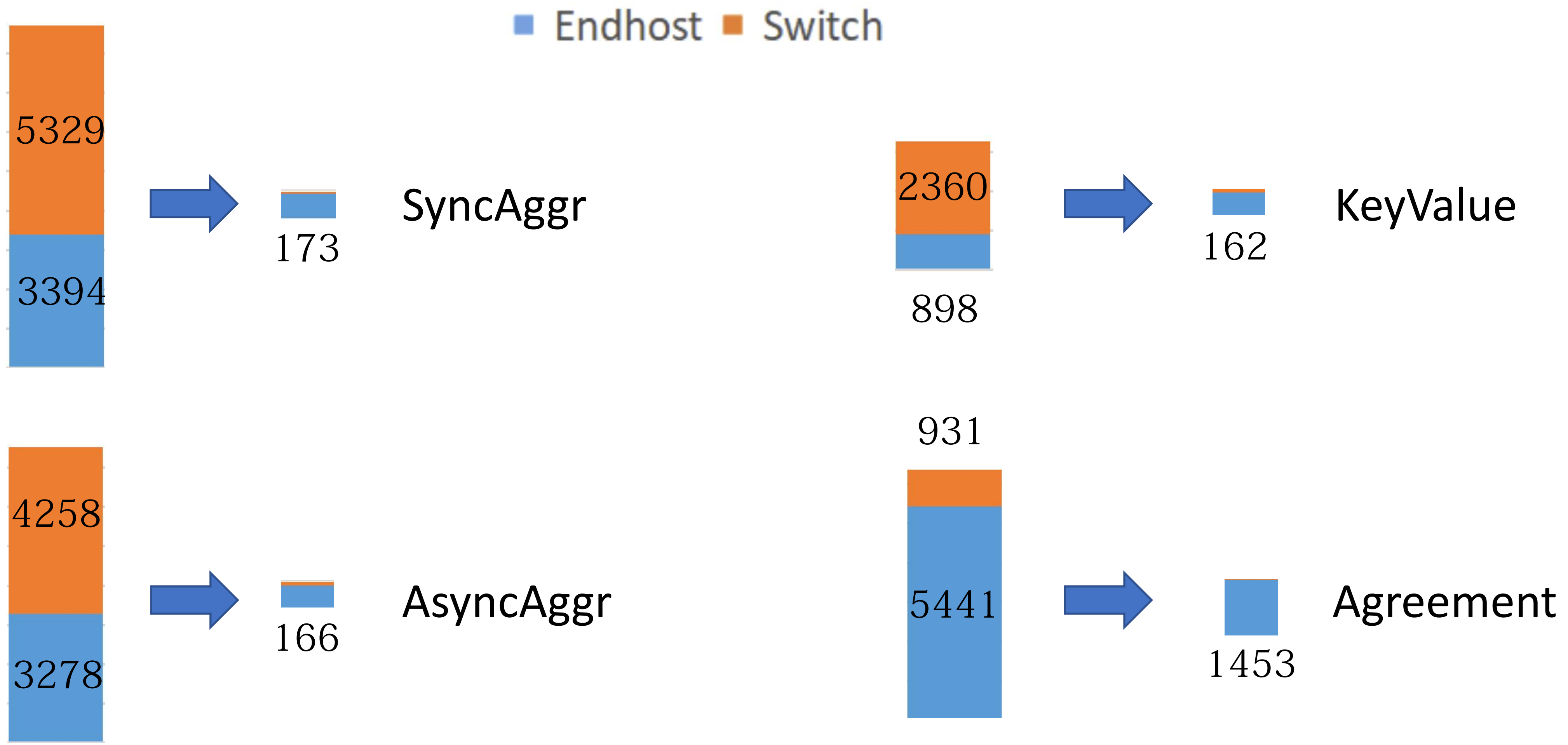


Type	Applications and Existing Systems
SyncAgtr	Distributed ML training (ATP, SHARP, SwitchML)
AsyncAgtr	MapReduce (ASK, NetAccel, Cheetah)
KeyValue	Cache (NetCache, DistCache), Monitoring (ElasticSketch)
Agreement	Synchronization (P4xos, NetChain, NetLock)

- Can NetRPC simplify INC programming?
- How does the NetRPC system perform?
- Can NetRPC support concurrent application?
- Can NetRPC guarantee relaibility?

NetRPC Greatly Reduces User Code Complexity

- NetRPC reduces lines of code of INC application by up to 97%



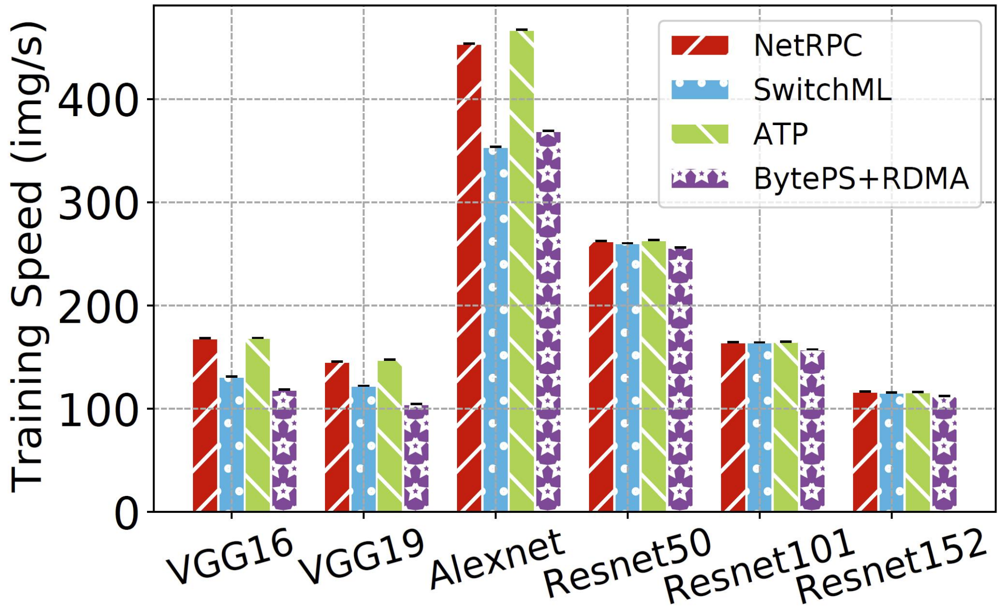
NetRPC Achieves Similar Performance to Handcrafted Code

- NetRPC achieves similar performance (≥90%) to baselines even after programming simplification

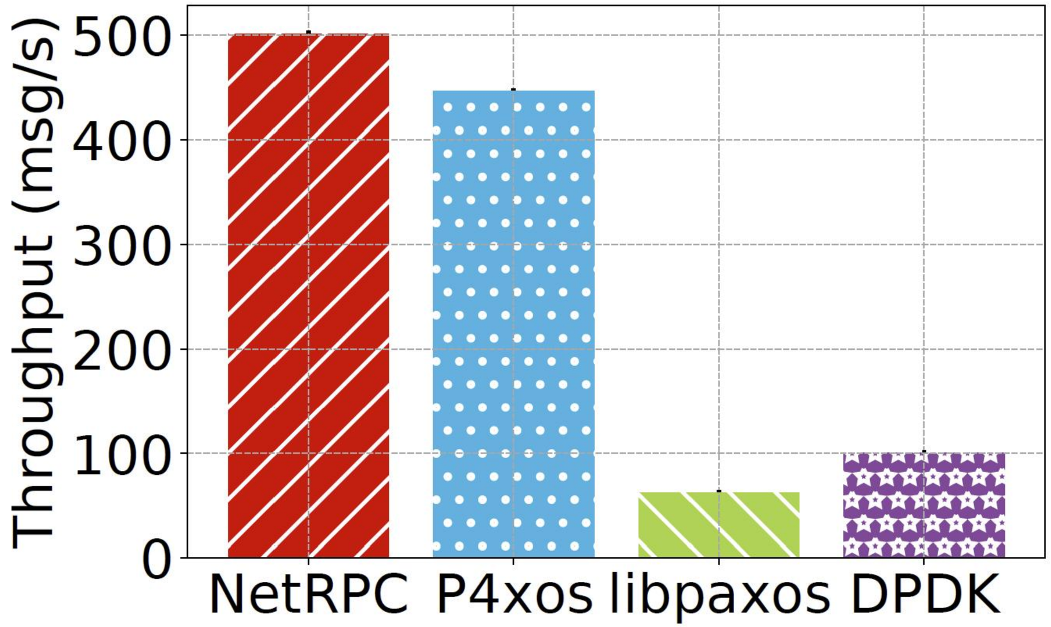
Metrics	NetRPC	Prior Arts	DPDK
SyncAgtr Goodput(Gbps)	50.55	46.44(ATP)	40.11
AsyncAgtr Goodput(Gbps)	72.31	73.96(ASK)	45.88
Voting Delay(μ s)	20	22(P4xos)	92
Monitor Delay(ms)	3.52	3.26(ElasticSketch)	4.05

Faster than Handcrafted Code in End-to-end Application

- NetRPC achieves even better training throughput than ATP (≥97%)
- NetRPC brings 12% higher throughput than P4xos



Distributed Training



Agreement

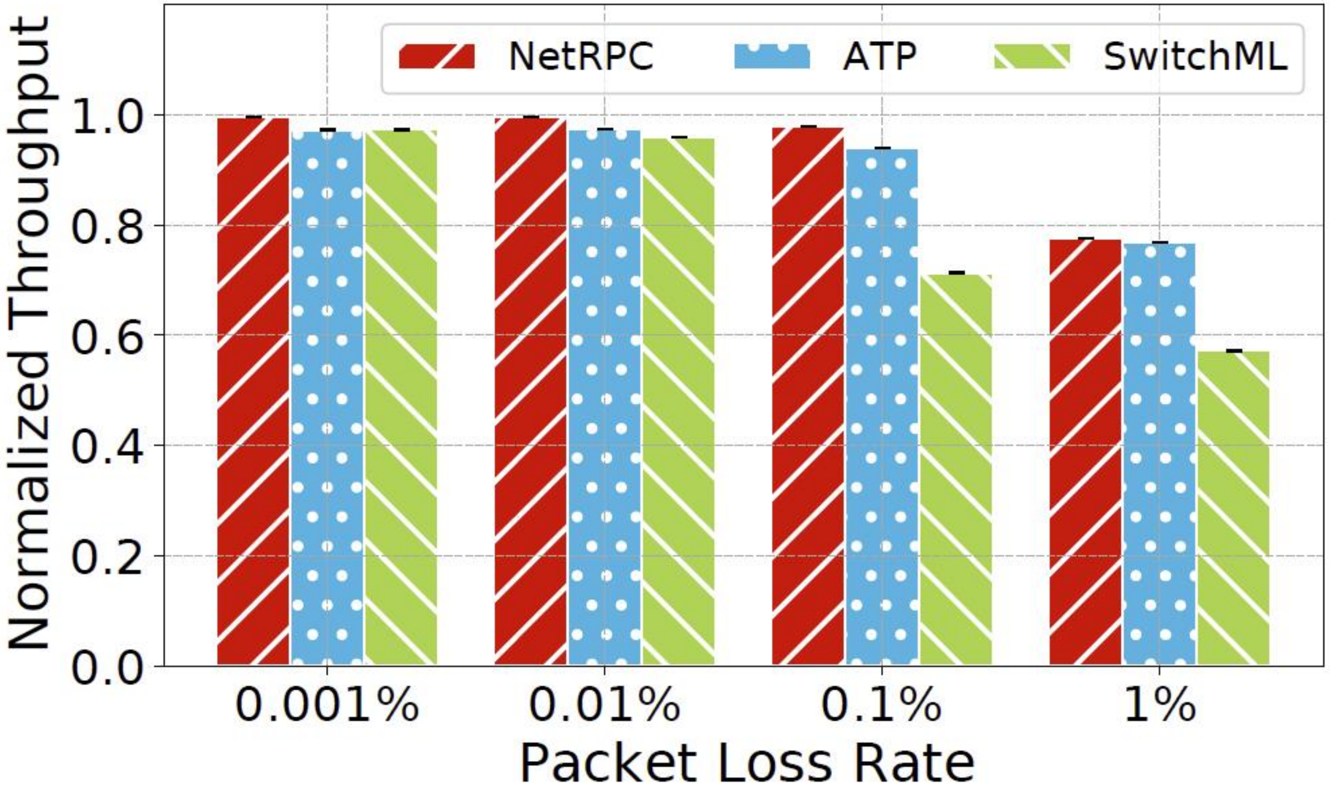
NetRPC Supports Multiple Concurrent Applications

- NetRPC can support concurrent INC applications with different types and different numbers

Metrics	1APP	4APP	4APP×5
Sync Goodput(Gbps)	50.55	24.88	24.84
Async Goodput(Gbps)	72.31	36.01	36.6
Goodput Sum(Gbps)	N/A	60.89	61.44
KeyValue Delay(ms)	3.52	3.56	3.85
AgreementDelay(μ s)	20	21	24

NetRPC is Reliable under Packet Loss

- NetRPC shows less performance degradation than prior arts with various packet loss rate.



Conclusion

- **NetRPC:**

  The first framework that integrates INC into the familiar RPC programming model

- **Contribution：**

  Make INC development easyer and offer similar or better performance boosts than handcrafted systems

- **Future work:**

  Explore scheduling policies and scale NetRPC to more complex topologies

Thanks!