FLICK: Developing and Running Application-Specific Network Services

Presenter: Richard G. Clegg, Imperial College
Imperial College: Abdul Alim, Luo Mai, Lukas Rupprecht, Eric Seckler, Paolo Costa, Peter Pietzuch, Alexander L. Wolf
Cambridge: Nik Sultana, Jon Crowcroft, Anil Madhavapeddy, Andrew W. Moore, Richard Mortier
Nottingham: Masoud Koleini, Carlos Oviedo, Derek McAuley
Kent: Matteo Migliavacca
Packet processing vs application-specific middlebox

Client

Packet processing
(process, packet):
dest = hash (packet.srcIP + packet.srcport)
forward (packet, dest);
Header data only used.
Packets have fixed format.

Application-specific
(memcached router)
(process, key_val_pair):
dest = hash (key_val_pair.key);
forward (key_val_pair, dest);
Applications have different data formats (e.g. key-value pairs, HTTP request/reply).
TCP flow not packets.
One packet != one data item.
Packet processing vs application-specific middlebox

Packet processing (ECMP loadbalancer)

process(packet):
   dest = hash(packet.srcIP + packet.srcport)
   forward(packet, dest);

Header data only used. Packets have fixed format. Basic data unit is packet.

Application-specific (memcached router)

process(key_val_pair):
   dest = hash(key_val_pair.key)
   forward(key_val_pair, dest);

Applications have different data formats (e.g. key-value pairs, HTTP request/reply). TCP flow not packets. One packet != one data item.
Packet processing vs application-specific middlebox

Packet processing
(ECMP loadbalancer)

process(packet):
    dest = hash(packet.srcIP + packet.srcport)
    forward(packet, dest);

- Header data only used.
- Packets have fixed format.
- Basic data unit is packet.

Application-specific
(memcached router)

process(key_val_pair):
    dest = hash(key_val_pair.key)
    forward(key_val_pair, dest);

Applications have different data formats (e.g., key-value pairs, HTTP request/reply).
TCP flow not packets.
One packet != one data item.
Packet processing vs application-specific middlebox

Packet processing (ECMP loadbalancer)

process(packet):
  dest = hash(packet.srcIP + packet.srcport)
  forward(packet, dest);

- Header data only used.
- Packets have fixed format.
- Basic data unit is packet.

Application-specific (memcached router)

process(key_val_pair):
  dest = hash(key_val_pair.key);
  forward(key_val_pair, dest);
Packet processing vs application-specific middlebox

Packet processing (ECMP loadbalancer)

process(packet):
  dest = hash(packet.srcIP + packet.srcport)
  forward(packet, dest);

- Header data only used.
- Packets have fixed format.
- Basic data unit is packet.

Application-specific (memcached router)

process(key_val_pair):
  dest = hash(key_val_pair.key)
  forward(key_val_pair, dest);

- Applications have different data formats (e.g. key-value pairs, HTTP request/reply).
- TCP flow not packets.
- One packet != one data item.
Problem: The application-specific middlebox

Figures from: Making Middleboxes Someone Elses Problem, Sherry et al. SIGCOMM 2012

Richard G. Clegg
FLICK: Application-specific network services
USENIX ATC
Problem: The application-specific middlebox

Figures from: Making Middleboxes Someone Elses Problem, Sherry et al. SIGCOMM 2012
Problem: The application-specific middlebox

NetAgg: Using Middleboxes for On-path Aggregation
CoNEXT 2014

Figures from: Making Middleboxes Someone Elses Problem, Sherry et al. SIGCOMM 2012
Problem: The application-specific middlebox

BlindBox: Deep Packet Inspection over Encrypted Traffic
**SIGCOMM 2015**

NetAgg: Using Middleboxes for On-path Aggregation
**CoNEXT 2014**

Figures from: Making Middleboxes Someone Elses Problem, Sherry et al. SIGCOMM 2012
Problem: The application-specific middlebox

- BlindBox: Deep Packet Inspection over Encrypted Traffic
  SIGCOMM 2015

- Yoda: A Highly Available Layer-7 Load Balancer
  EuroSys 2016

- NetAgg: Using Middleboxes for On-path Aggregation
  CoNEXT 2014

Figures from: Making Middleboxes Someone Else's Problem, Sherry et al. SIGCOMM 2012
Problem: The application-specific middlebox

BlindBox: Deep Packet Inspection over Encrypted Traffic
SIGCOMM 2015

NetAgg: Using Middleboxes for On-path Aggregation
CoNEXT 2014

Yoda: A Highly Available Layer-7 Load Balancer
EuroSys 2016

Introducing mcrouter: A memcached protocol router
Facebook blog

Figures from: Making Middleboxes Someone Elses Problem, Sherry et al. SIGCOMM 2012
Creating new application-specific middlebox

Expressiveness

Ease of implementation
Creating new application-specific middlebox

Ease of implementation

Expressiveness

C/C++ code from scratch
Creating new application-specific middlebox

Expressiveness

Ease of implementation

C/C++ code from scratch

ClickOS existing modules
Creating new application-specific middlebox

Ease of implementation

Expressiveness

C/C++ code from scratch

ClickOS new modules

ClickOS existing modules
Creating new application-specific middlebox

Expressiveness

Ease of implementation

C/C++ code from scratch

ClickOS new modules

Software Defined Networking

ClickOS existing modules

Richard G. Clegg
FLICK: Application-specific network services
USENIX ATC 3 / 23
FLICK for the datacentre
FLICK for the datacentre

- WAN opt
- IP firewall
- HTTP loadbalancer
- HTTP loadbalancer
- mcrouter
- spam filter
- mcrouter
- mcrouter
- mcrouter
FLICK for the datacentre

FLICK

IP firewall

FLICK

FLICK

FLICK

FLICK

FLICK

FLICK

FLICK

FLICK

FLICK

FLICK

FLICK

FLICK

FLICK
General system for application-specific middleboxes?

**Challenge 1: Ease-of-use**

Rapidly express many middlebox functions.
System created in hours not weeks/months.
General system for application-specific middleboxes?

**Challenge 1: Ease-of-use**
Rapidly express many middlebox functions. System created in hours not weeks/months.

**Challenge 2: Performance**
Generality must not have large performance penalty. Performance similar to specially written system.
Challenge 1: Ease-of-use
Rapidly express many middlebox functions.
System created in hours not weeks/months.

Challenge 2: Performance
Generality must not have large performance penalty.
Performance similar to specially written system.

Challenge 3: Safety/Isolation
Middleboxes should be “safe” in resource usage.
Applications on same machine share resources well.
FLICK overview

**FLICK programs**

```
// type kv: record
//   key : string
//   value : string

// function hadoop: ([(kv->-) mappers, -/kv reducer]):
//   if !all_ready(mappers):
//     let result = foldl on mappers
//       ordering elem el, e2 by elem.key as e_key:
//       let v = combine(x1.val, x2.val)
//       kv(e_key, v)
//     result -> reducer
//   fun combine(v1: string, v2: string) -> (string):
```

**Flick programs**

Domain specific language (DSL) for application-specific middleboxes.
Tens of lines of code not tens of thousands
Flick task graphs

Break work into independently schedulable units (tasks). Join tasks by channels into task graphs.
FLICK overview

Flick platform
The running implementation. Integrates the compiled C++ from DSL. Handles network connections, worker threads and scheduling tasks.
FLICK – the language

Richard G. Clegg
FLICK: Application-specific network services
USENIX ATC 7 / 23
FLICK (language) – features

type cmd: record
  key : string

proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
  | backends => client
  | client => target_backend(backends)

fun target_backend: ([/-cmd] backends, req:cmd) -> ()
  let target = hash(req.key) mod len(backends)
  req => backends[target]
**FLICK (language) – features**

```ocaml
type cmd: record
   key : string

proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
   | backends => client
   | client => target_backend(backends)

fun target_backend: ([cmd] backends, req: cmd) -> ()
   let target = hash(req.key) mod len(backends)
   req => backends[target]
```

- Process as basic unit of code expresses flow of typed data.
- Control structures restricted. Bounded loops and hence execution time.
- Strongly typed for safety.
**FLICK (language) – processing data (memcached)**

```
type cmd: record
  key : string

proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
  | backends => client
  | client => target_backend(backends)

fun target_backend: ([/-cmd] backends, req: cmd) -> ()
  let target = hash(req.key) mod len(backends)
  req => backends[target]
```
FLICK (language) – processing data (memcached)

Structure allows work to break into smaller task units

```plaintext

<table>
<thead>
<tr>
<th>type</th>
<th>key : string</th>
</tr>
</thead>
</table>

proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
  | backends => client
  | client => target_backend(backends)

fun target_backend: ([cmd] backends, req:cmd) -> ()
  let target = hash(req.key) mod len(backends)
  req => backends[target]
```

Richard G. Clegg  
FLICK: Application-specific network services
FLICK (language) – processing data (memcached)

```
type cmd : record
  key : string
  ...

proc Memcached : (cmd / cmd client, [cmd / cmd] backends)
  | backends => client
  | cmd => ...

fun target_backend : ([/-cmd] backends, req : cmd) -> ()
  let target = hash(req.key) mod len(backends)
  req => backends[target]
```

Structure allows work to break into smaller task units

Convenient abstractions for middlebox
**FLICK (language) – processing data (memcached)**

```
type cmd: record
  key : string

proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
  | backends => client
  | client => target_backend(backends)

fun target_backend: ([--;/cmd] backends, req:cmd) -> ()
  let target = hash(req.key) mod len(backends)
  req => backends[target]
```
type cmd: record
key : string

proc Memcached: (cmd/cmd client, [cmd/cmd] backends)
| backends => client
| client => target_backend(backends)

fun target_backend: ([-/cmd] backends, req:cmd) -> ()
  let target = hash(req.key) mod len(backends)
  req => backends[target]

Process: entry point defines how channels connect
FLICK (language) – processing data (memcached)

\[
\text{type cmd: record} \\
\qquad \text{key : string}
\]

\[
\text{proc Memcached: (cmd/cmd client, [cmd/cmd] backends)} \\
\qquad | \text{backends => client} \\
\qquad | \text{client => target_backend(backends)}
\]

\[
\text{fun target_backend: ([-/cmd] backends, req:cmd) -> ()} \\
\qquad \text{let target = hash(req.key) mod len(backends)} \\
\qquad \text{req => backends[target]}
\]
type cmd = unit {
    %byteorder = big;
    magic_code : uint8;
    opcode : uint8;
    key_len : uint16;
    extras_len : uint8;
    status_or_v_bucket : uint16;
    total_len : uint32;
    opaque : uint32;
    cas : uint64;
    extras : bytes & length = self.extras_len;
    key : string & length = self.key_len;
    value : bytes & length = self.value_len;
};
Based on Spicy/binpac++ [IMC2006]
FLICK (language) – parsing data (memcached)

Based on Spicy/binpac++ [IMC2006]

Developer can quickly parse even complex formats like HTTP

```plaintext
type cmd = unit {
  % byteorder = big;
  magic_code : uint8;
  opcode : uint8;
  key_len : uint16;
  extras_len : uint8;
  # anon field - future use
  status_or_v_bucket : uint16;
  total_len : uint32;
  opaque : uint32;
  cas : uint64;
  extras : bytes & length = self.extras_len;
  key : string & length = self.key_len;
  value : bytes & length = self.value_len;
};
```
type cmd = unit {
  % byteorder = big;
  magic_code : uint8;
  opcode : uint8;
  key_len : uint16;
  extras_len : uint8;
  % anon field - future use
  status_or_v_bucket : uint16;
  total_len : uint32;
  opaque : uint32;
  cas : uint64;
  extras : bytes & length = self.extras_len;
  key : string & length = self.key_len;
  value : bytes & length = self.value_len;
};

Based on Spicy/binpac++ [IMC2006]

Developer can quickly parse even complex formats like HTTP

Compiles to efficient C++. Only extracts fields used in processing
type cmd = unit {
  %byteorder = big;
  magic_code : uint8;
  opcode : uint8;
  key_len : uint16;
  extras_len : uint8;
  extras_len : uint8;  # anon field - future use
  status_or_v_bucket : uint16;
  total_len : uint32;
  opaque : uint32;
  cas : uint64;
  extras : bytes & length = self.extras_len;
  key : string & length = self.key_len;
  value : bytes & length = self.value_len;
};
type cmd = unit {
  %byteorder = big;
  magic_code : uint8;
  opcode : uint8;
  key_len : uint16;
  extras_len : uint8;
  : uint8;  # anon field - future use
  status_or_v_bucket : uint16;
  total_len : uint32;
  opaque : uint32;
  cas : uint64;
  extras : bytes & length = self.extras_len;
  key : string & length = self.key_len;
  value : bytes & length = self.value_len;
};
FLICK – the task graph

FLICK programs

FLICK task graphs

FLICK platform

compiled

loaded & executed

cooperative scheduling

thread

thread

thread

channels
FLICK – the task graph

Separate input, processing and output tasks enable parallelism
FLICK – the task graph

Input TCP flow

Client

Server 1

Server 2
FLICK – the task graph

Client

Server 1

Server 2

Input task deserialises data
FLICK – the task graph

Processing task forwards by key

Client

Server 1

Server 2
FLICK – the task graph

Client

Server 1

Server 2

Output task serialises data
FLICK – the task graph

Client

Server 1

Server 2

Server sends reply
FLICK – the task graph

Input task deserialises data

Client

Server 1

Server 2
For memcached router each client has its own task graph.
FLICK – the task graph

- For memcached router each client has its own task graph.
- Different types of task graph – some have data parallelism.
- Data and task parallelism.
FLICK – the platform

FLICK programs

FLICK task graphs

FLICK platform

compiled

loaded & executed

cooporative scheduling

thread

thread

thread

channels
FLICK – the platform

Graph Dispatcher

Existing graph

New graph

Graph pool

Application Dispatcher

Scheduler

Task Queue

Work Threads

Richard G. Clegg

FLICK: Application-specific network services
FLICK – the platform

Efficiently handle the TCP connection set up and tear down

Graph Dispatcher

Graph pool

Application Dispatcher

Scheduler

Work Threads
FLICK – the platform

Efficiently handle the TCP connection set up and tear down

Manage memory allocation smartly (reduce dynamic allocation)

Application Dispatcher

Graph Dispatcher

Scheduler

Work Threads

Richard G. Clegg

FLICK: Application-specific network services

USENIX ATC
FLICK – the platform

- Efficiently handle the TCP connection setup and tear down
- Manage memory allocation smartly (reduce dynamic allocation)
- Schedule tasks fairly between applications (safety/isolation)

Richard G. Clegg

FLICK: Application-specific network services

USENIX ATC 15 / 23
FLICK – the platform

Application Dispatcher
FLICK – the platform
FLICK – the platform

Application Dispatcher

DPDK/mTCP (userland TCP)  reduce kernel calls
FLICK – the platform

Choose application to connect to

Application Dispatcher
FLICK – the platform

Existing graph

Graph Dispatcher

Graph pool

Application Dispatcher

Richard G. Clegg

FLICK: Application-specific network services

USENIX ATC
FLICK – the platform

Connect to existing graph

Graph Dispatcher

Application Dispatcher

Existing graph

Graph pool
FLICK – the platform

Graph Dispatcher

Application Dispatcher

Existing graph

New graph from pool

New graph

Graph pool

Richard G. Clegg  FLICK: Application-specific network services  USENIX ATC
FLICK – the platform

Graph Dispatcher

Existing graph

New graph

Graph pool

Application Dispatcher

Scheduler

Task Queue

Work Threads

Richard G. Clegg

FLICK: Application-specific network services
FLICK – the platform

Application Dispatcher

Graph Dispatcher

Graph pool

Existing graph

Tasks with data added to queue

Scheduler

Task Queue

Work Threads

Richard G. Clegg

FLICK: Application-specific network services
FLICK – the platform

Existing graph

New graph

Graph pool

Application Dispatcher

Scheduler

Task Queue

Workers pick tasks from queue

Work Threads

Richard G. Clegg

FLICK: Application-specific network services

USENIX ATC
FLICK – the platform

- Graph Dispatcher
  - Graph pool
  - Application Dispatcher
  - Scheduler
    - Task Queue
      - Tasks have limited time on a thread
    - Work Threads
FLICK – the platform

- Existing graph
- New graph
- Graph pool
- Scheduler
- Task Queue
- Work Threads

Richard G. Clegg
Evaluation – latency/throughput (loadbalancer)

Clients send HTTP requests up to ten backends. Persistent TCP connections to/from loadbalancer. Vary number of clients measure latency and throughput. DPDK/mTCP used to reduce kernel calls in connections.
Clients send HTTP requests up to ten backends.
Persistent TCP connections to/from loadbalancer.
Vary number of clients measure latency and throughput.
DPDK/mTCP used to reduce kernel calls in connections.
Evaluation – latency (loadbalancer)

Lower is better
Evaluation – latency (loadbalancer)

![Graph showing latency vs concurrent clients for FLICK, Apache, and Nginx]

- **Mean latency (ms)**
- **Concurrent clients**

**FLICK**

**Apache**

**Nginx**

---

Richard G. Clegg

FLICK: Application-specific network services

USENIX ATC
Evaluation – latency (loadbalancer)

Richard G. Clegg

FLICK: Application-specific network services

USENIX ATC
Evaluation – throughput (loadbalancer)

Concurrent clients vs. Thousand reqs/s

Higher is better

Richard G. Clegg

FLICK: Application-specific network services

USENIX ATC
Evaluation – throughput (loadbalancer)

Richard G. Clegg  FLICK: Application-specific network services  USENIX ATC
Evaluation – throughput (loadbalancer)

![Graph showing throughput for different load balancers and software packages: FLICK mTCP, FLICK, Apache, Nginx.](image)

The graph illustrates the throughput (in thousand requests per second) for different concurrent client loads. The x-axis represents the number of concurrent clients, ranging from 100 to 1600, and the y-axis represents the throughput in thousand requests per second, ranging from 0 to 300 throttle.

Richard G. Clegg

FLICK: Application-specific network services

USENIX ATC
Evaluation – scalability with cores

This middlebox merges data in big data systems. Binary merge tree takes advantage of data parallelism. See "NetAgg: Using Middleboxes for Application-specific On-path Aggregation in Data Centres" [CoNext 2014].
Evaluation – scalability with cores

- This middlebox merges data in big data systems.
- Binary merge tree takes advantage of data parallelism.
- See “NetAgg: Using Middleboxes for Application-specific On-path Aggregation in Data Centres” [CoNext 2014].
Evaluation – scalability with cores

Test scaling. Measure throughput as number of cores increases.

Three data sets each one billion words. 8, 12 and 16 character words.

Richard G. Clegg
FLICK: Application-specific network services
USENIX ATC
Evaluation – scalability with cores

- Test scaling. Measure throughput as number of cores increases.
- Three data sets each one billion words. 8, 12 and 16 character words.
- Merge eight streams – measure throughput of output stream.
Evaluation – scalability with cores

Maximum throughput

Higher is better

Richard G. Clegg
FLICK: Application-specific network services
USENIX ATC
Evaluation – scalability with cores

Throughput Mb/s

16 char words

Maximum throughput

CPU cores

Richard G. Clegg

FLICK: Application-specific network services

USENIX ATC
Evaluation – scalability with cores

Maximum throughput

Throughput Mb/s

CPU cores

Richard G. Clegg

FLICK: Application-specific network services

USENIX ATC
Conclusions

Application-specific services

- Application-specific middleboxes are here to stay.
- Packet processing systems not suitable for these.
Conclusions

Application-specific services
- Application-specific middleboxes are here to stay.
- Packet processing systems not suitable for these.

The FLICK system
- FLICK domain-specific language – “safe by design”.
- Task graph abstraction gives task and data parallelism.
- Performance of FLICK comparable to specialist system.
Conclusions

Application-specific services

- Application-specific middleboxes are here to stay.
- Packet processing systems not suitable for these.

The FLICK system

- FLICK domain-specific language – “safe by design”.
- Task graph abstraction gives task and data parallelism.
- Performance of FLICK comparable to specialist system.

Thank you – questions?

Richard G. Clegg
richard.clegg@imperial.ac.uk
Comparison with Moxi (also supports multi-core + binary protocol).
Set up 128 clients making multiple requests.
Latency reduction shown.
FLICK throughput with mTCP 198,000 reqs/sec.
Moxi throughput 82,000 reqs/sec