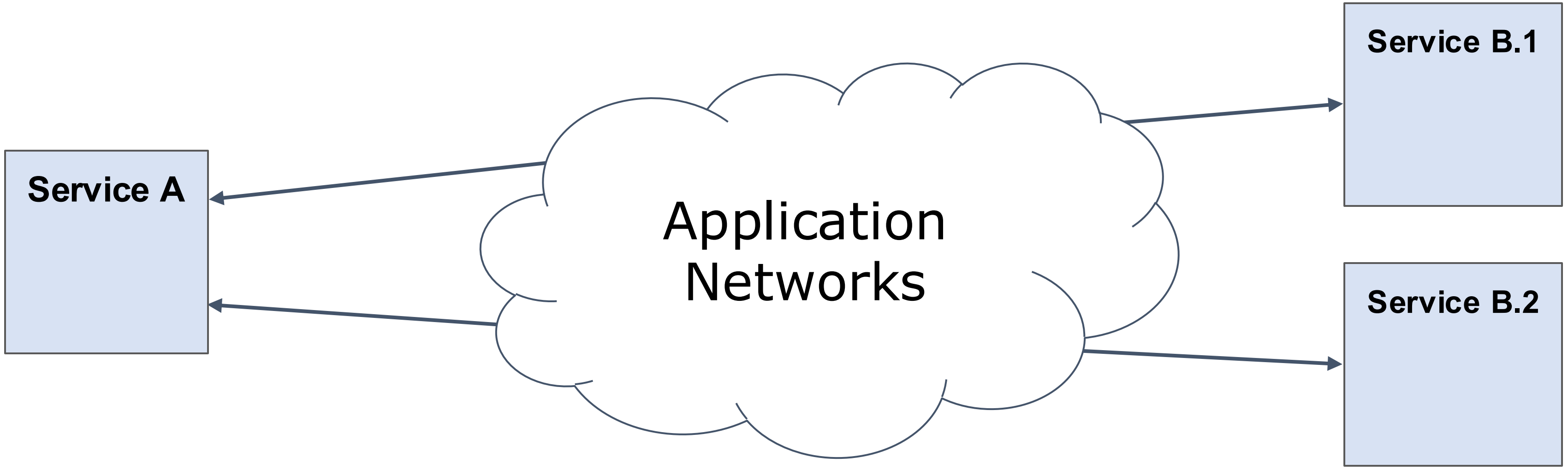
High-level Programming of Application Networks

Xiangfeng Zhu, Yuyao Wang, Banruo Liu, Yongtong Wu, Nikola Bojanic, Jingrong Chen, Gilbert Bernstein, Arvind Krishnamurthy, Sam Kumar, Ratul Mahajan, Danyang Zhuo

The Rise of Application Networks



Traffic Management

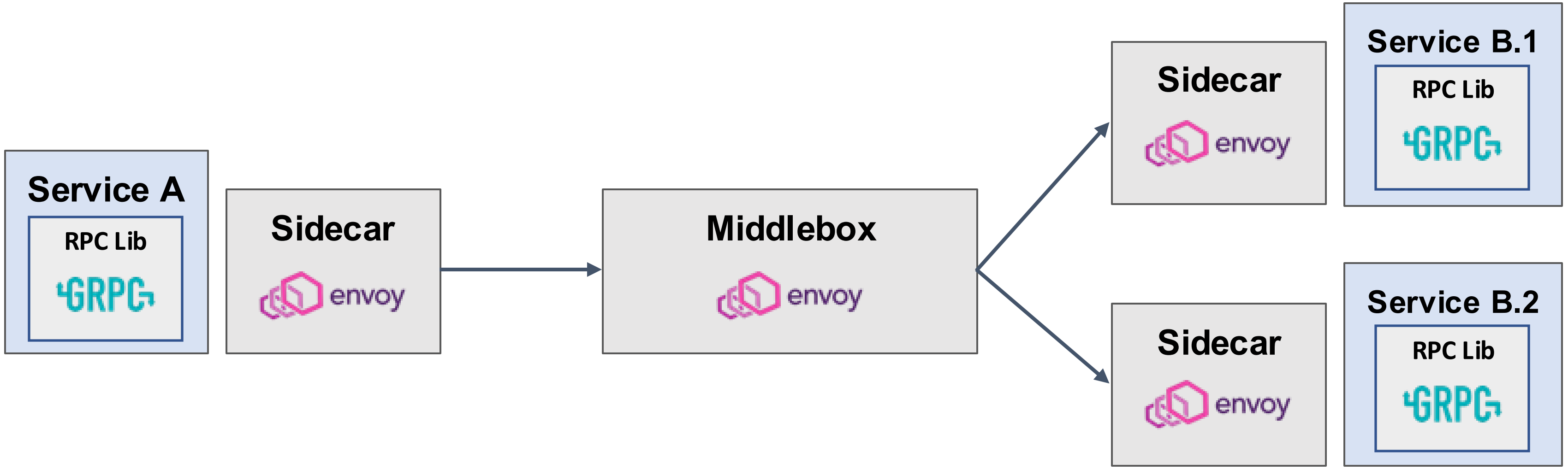
Policy Enforcement

Security

Observability

Application Network Functions (ANFs)

Current Approach: Service Meshes



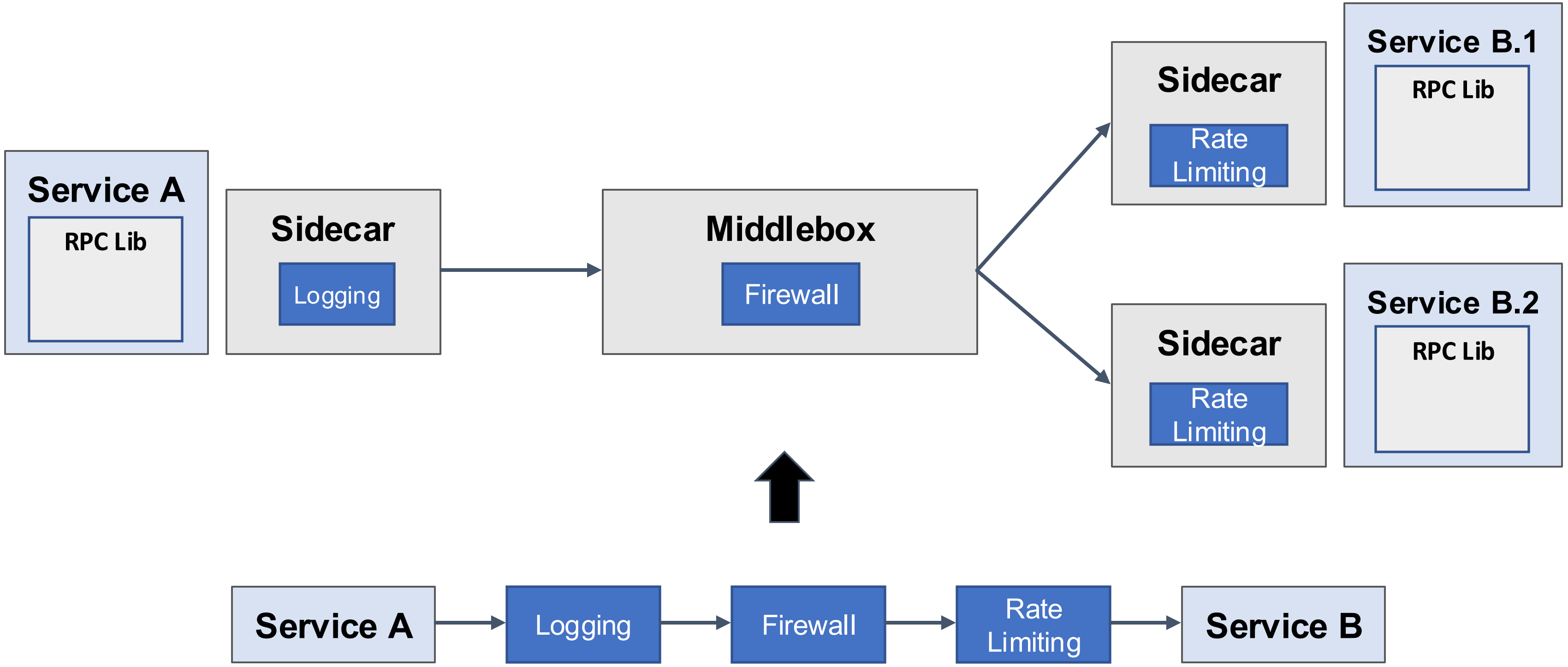
Istio

Google Cloud Anthos

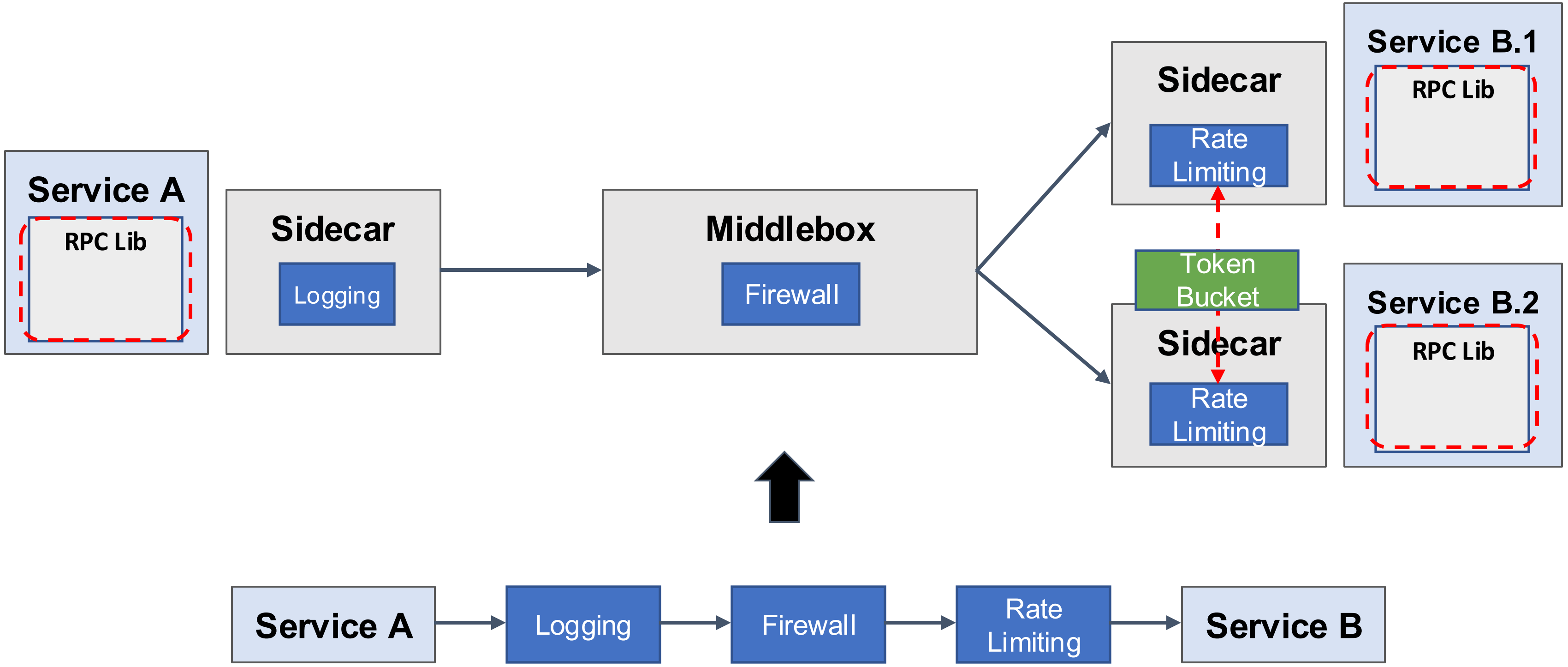
HashiCorp Consul

VMware Tanzu

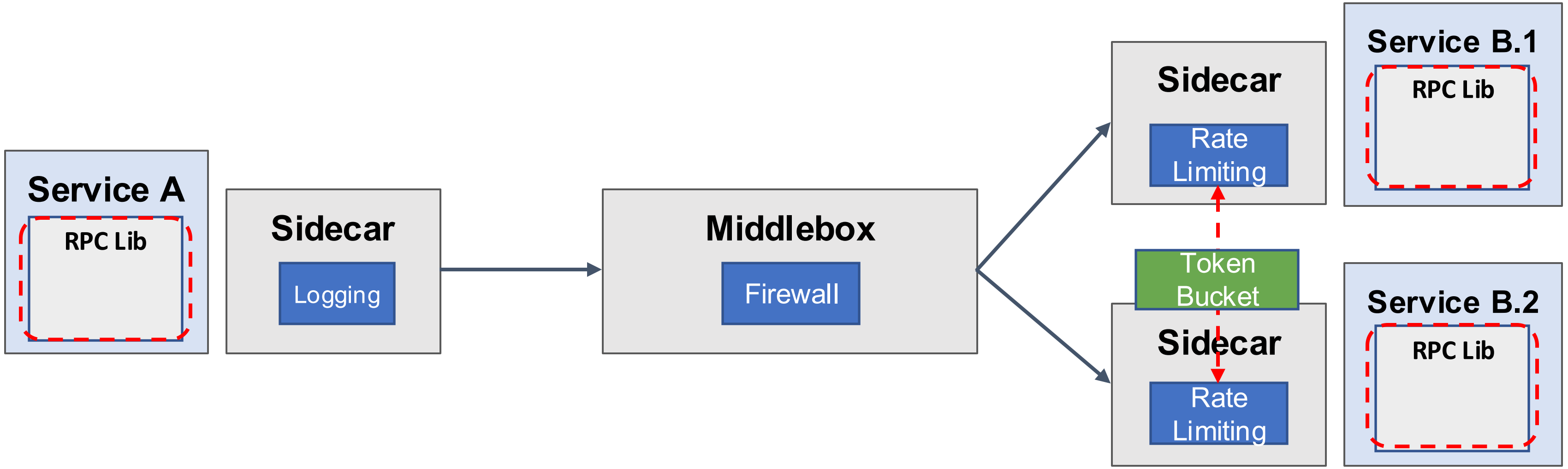
Challenge 1: High Developer Burden



Challenge 2: High Performance Overhead

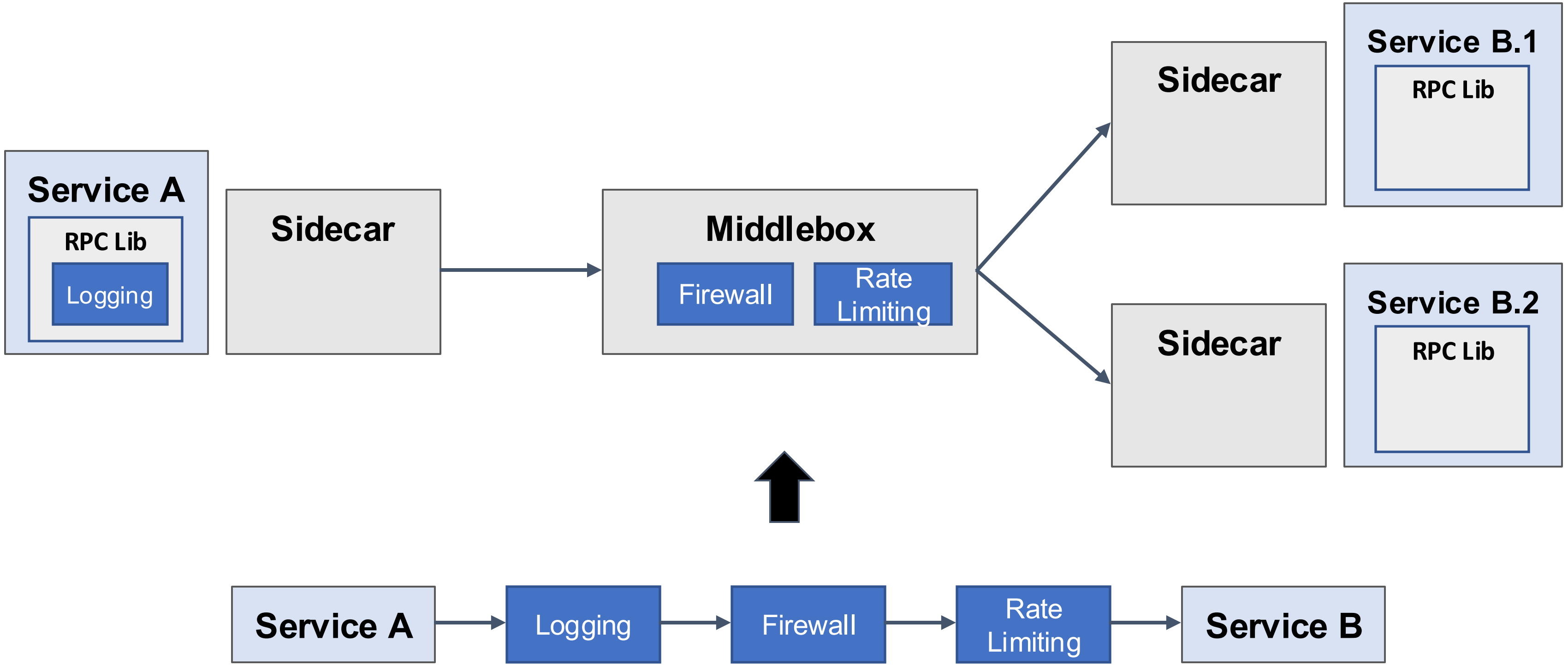


Challenge 2: High Performance Overhead



Service mesh can increase latency and CPU usage by 2-7X

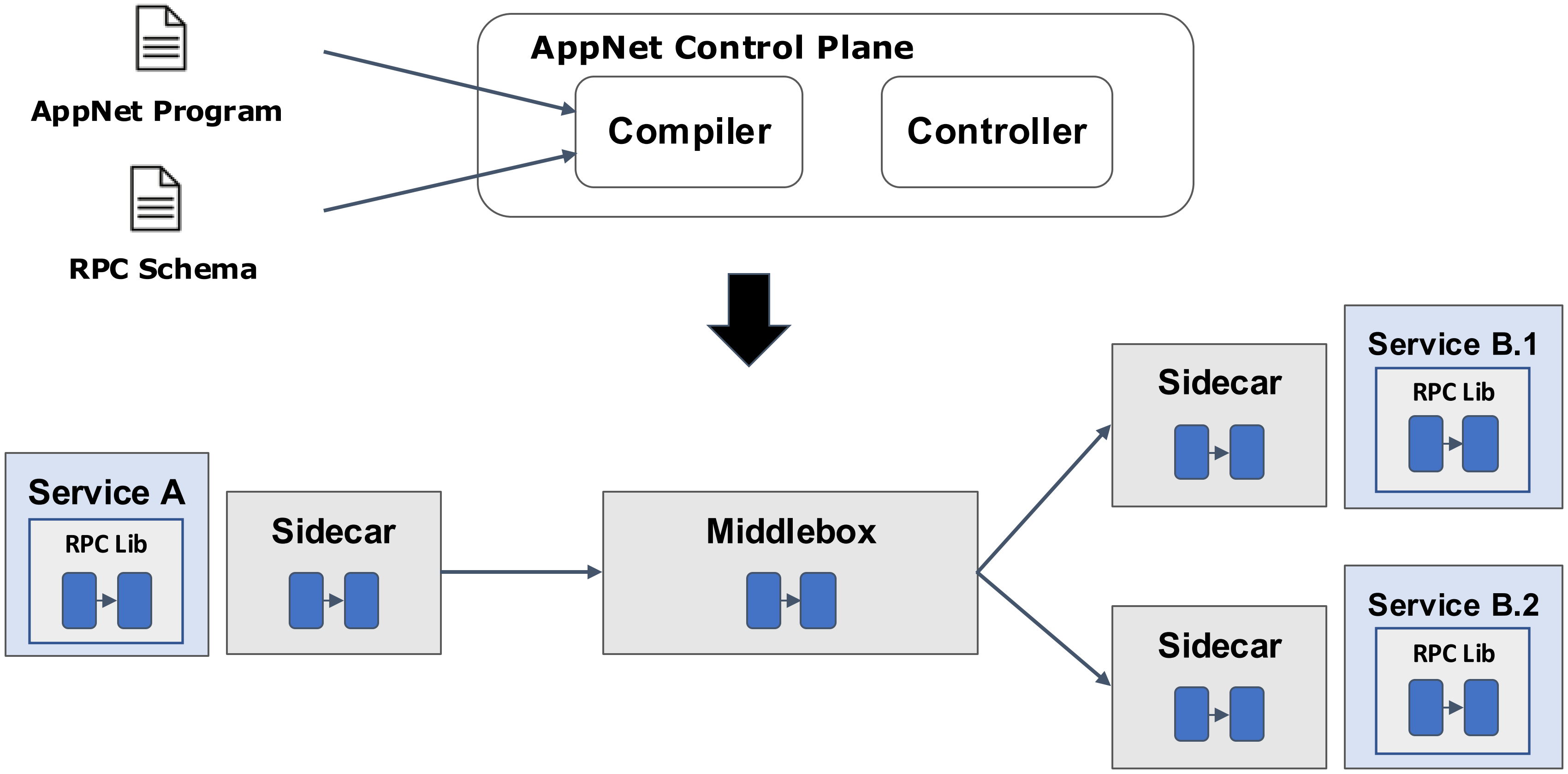
Challenge 2: High Performance Overhead



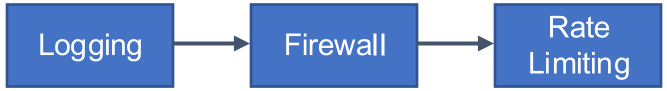
Goal

Make application networks **easy to build** and **highly performant**

AppNet: Decouples Specification from Implementation



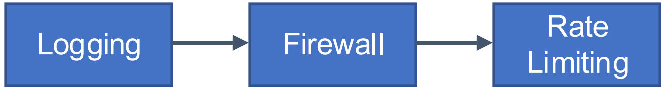
AppNet Abstractions



- RPC Processing as a chain of elements

```
log()——>firewall()——>rate_limiting()
```

AppNet Abstractions



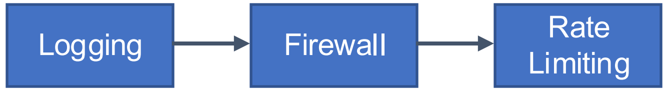
- RPC Processing as a chain of elements

- Generalized match-action rules over RPC field and state

```
req(rpc):
    username = get(rpc, 'username')          // Get username from RPC

    match get(firewall_rules, username):     // Get the permission from firewall_rules table
        'allowed' =>
            send(rpc)
        'denied' =>
            send(err('firewall'))
        None =>
            send(err('firewall'))
```

AppNet Abstractions



- RPC Processing as a chain of elements
- Generalized match-action rules over RPC field and state
- Shared state with configurable consistency level

See Paper for AppNet Grammar

$$
\begin{aligned}
\mathit{Chain} &::= \left[\begin{array}{l}
\mathbf{\texttt{client}}\text{: } \mathit{Element}^* \\
\mathbf{\texttt{any}}\text{: } \mathit{Element}^* \\
\mathbf{\texttt{server}}\text{: } \mathit{Element}^* \\
\mathbf{\texttt{pair}}\text{: } (\mathit{Element}, \mathit{Element})^* \\
[\mathbf{\texttt{weak}}]
\end{array}\right. \\
\mathit{Element} &::= \left[\begin{array}{l}
\mathbf{\texttt{state}}\text{: } \mathit{Decl}^* \\
\mathbf{\texttt{init}}(\mathit{Var}^*)\text{: } \mathit{Assign}^* \\
\mathbf{\texttt{req}}(\mathit{Var})\text{: } \mathit{Action}^* \ [\mathit{MatchAction}] \\
\mathbf{\texttt{resp}}(\mathit{Var})\text{: } \mathit{Action}^* \ [\mathit{MatchAction}]
\end{array}\right. \\
\mathit{Decl} &::= \mathit{Var} \ [\mathbf{\texttt{shared}} \ [\mathbf{\texttt{weak}} \ [\mathbf{\texttt{sum}}]]] \\
\mathit{MatchAction} &::= \mathbf{\texttt{match}}(\mathit{Expr}) \ \mathit{Case}^+ \ [\texttt{'*'} \Rightarrow \mathit{Action}^+] \\
\mathit{Case} &::= \mathit{Literal} \Rightarrow \mathit{Action}^+ \\
\mathit{Action} &::= \mathit{Assign} \mid \mathit{Send} \mid \mathit{Foreach} \mid \mathit{Return} \\
\mathit{Assign} &::= \mathit{Var} = \mathit{Expr} \mid \mathbf{\texttt{set}}(\mathit{Var}, \mathit{Expr}^+, \mathit{Expr}) \\
\mathit{Send} &::= \mathbf{\texttt{send}}(\mathit{Message}, \mathit{Channel})
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Foreach} &::= \mathbf{\texttt{foreach}}(\mathit{Var}, \mathit{LambdaFunc}) \\
\mathit{Return} &::= \mathbf{\texttt{return}} \ [\mathit{Expr}] \\
\mathit{Message} &::= \mathit{Var} \mid \texttt{'error'} \\
\mathit{Channel} &::= \mathbf{\texttt{down}} \mid \mathbf{\texttt{up}} \mid \mathit{Var} \\
\mathit{Expr} &::= \mathit{Literal} \mid \mathit{Var} \mid \mathbf{\texttt{get}}(\mathit{Var}, \mathit{Expr}^+ [, \mathit{LambdaFunc}]) \\
&\qquad \mid \text{BuiltinFunc}(\mathit{Expr}^*) \\
\mathit{LambdaFunc} &::= \mathbf{\texttt{lambda}}(\mathit{Var}+) \Rightarrow \mathit{Action}^* \ [\mathit{MatchAction}] \\
\mathit{Var} &\in (\text{set of variable names}) \\
\mathit{Literal} &\in (\text{literal values, e.g. } 0.1, 42, \mathbf{true})
\end{aligned}
$$

AppNet Compiler

- Goal: Find a high-performance configuration while preserving semantics
  - Platform (gRPC, Envoy, …)
  - Location (caller, callee, middlebox)
  - Execution Order

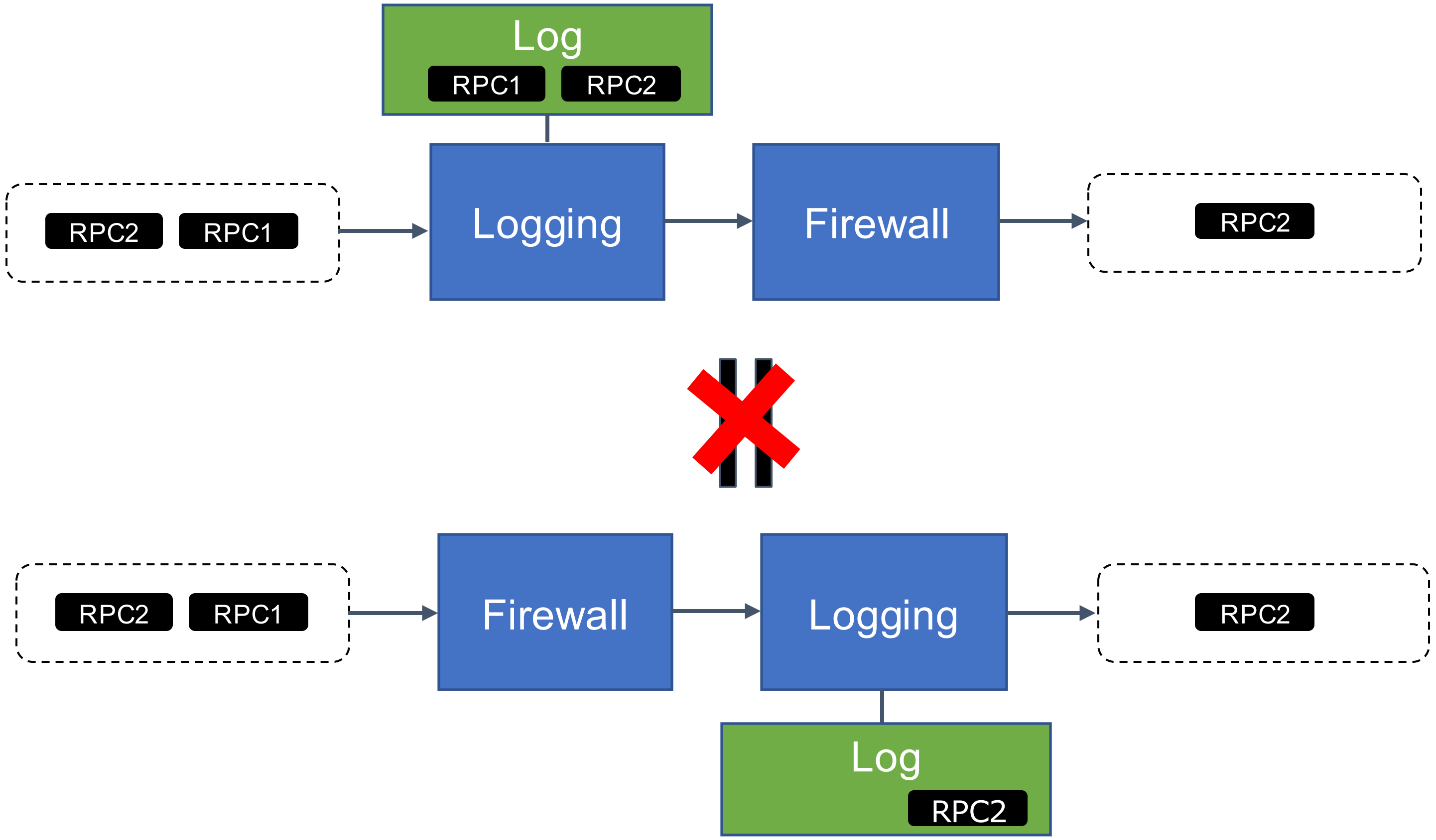
AppNet Compiler

- Goal: Find a high-performance configuration while preserving semantics

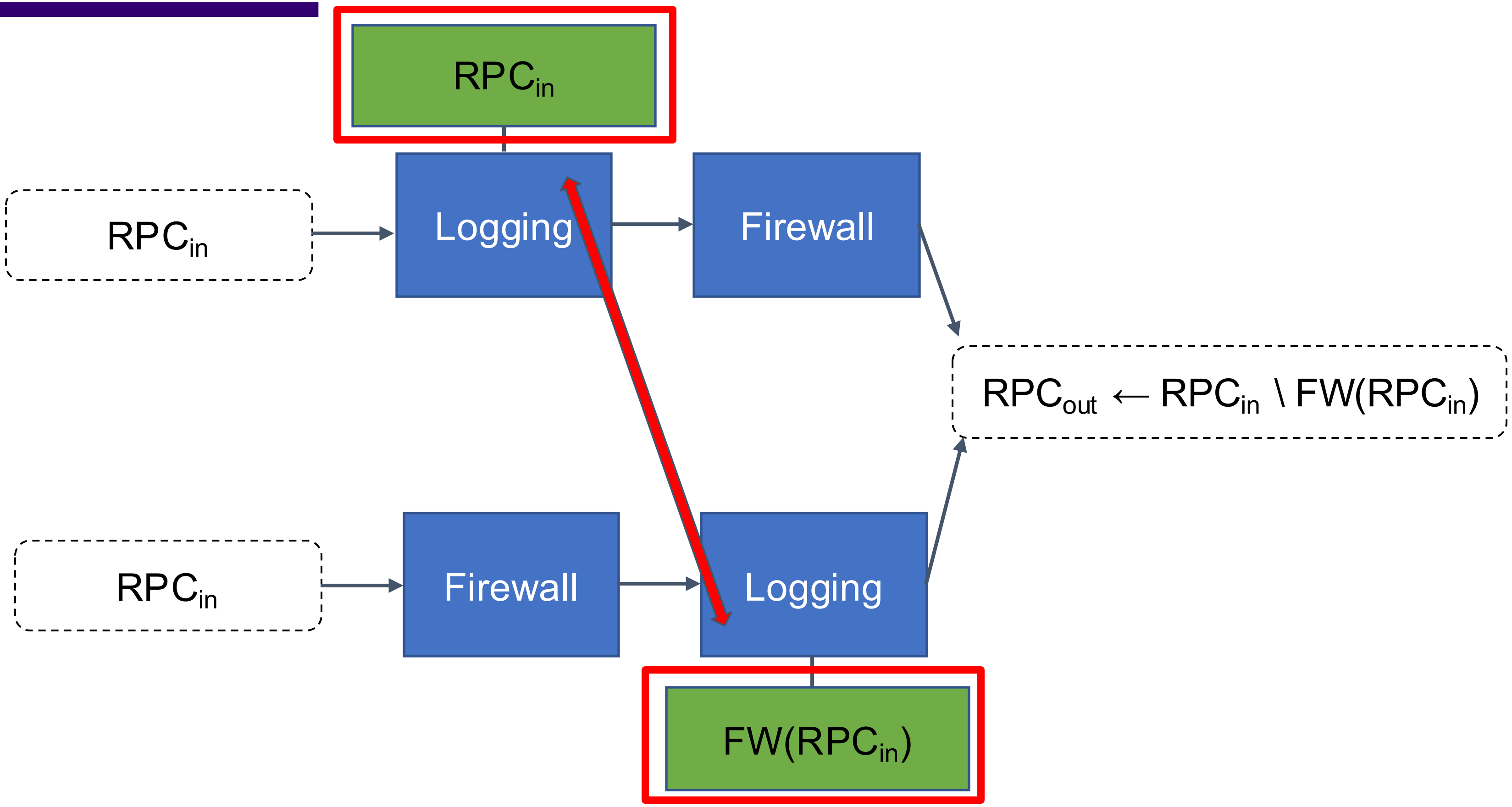
Challenges

- Preserve semantic equivalence
  - Some ANFs are stateful
  - Reordering or relocating ANFs may change behavior
- Huge search space
  - Many platform + location + order permutations

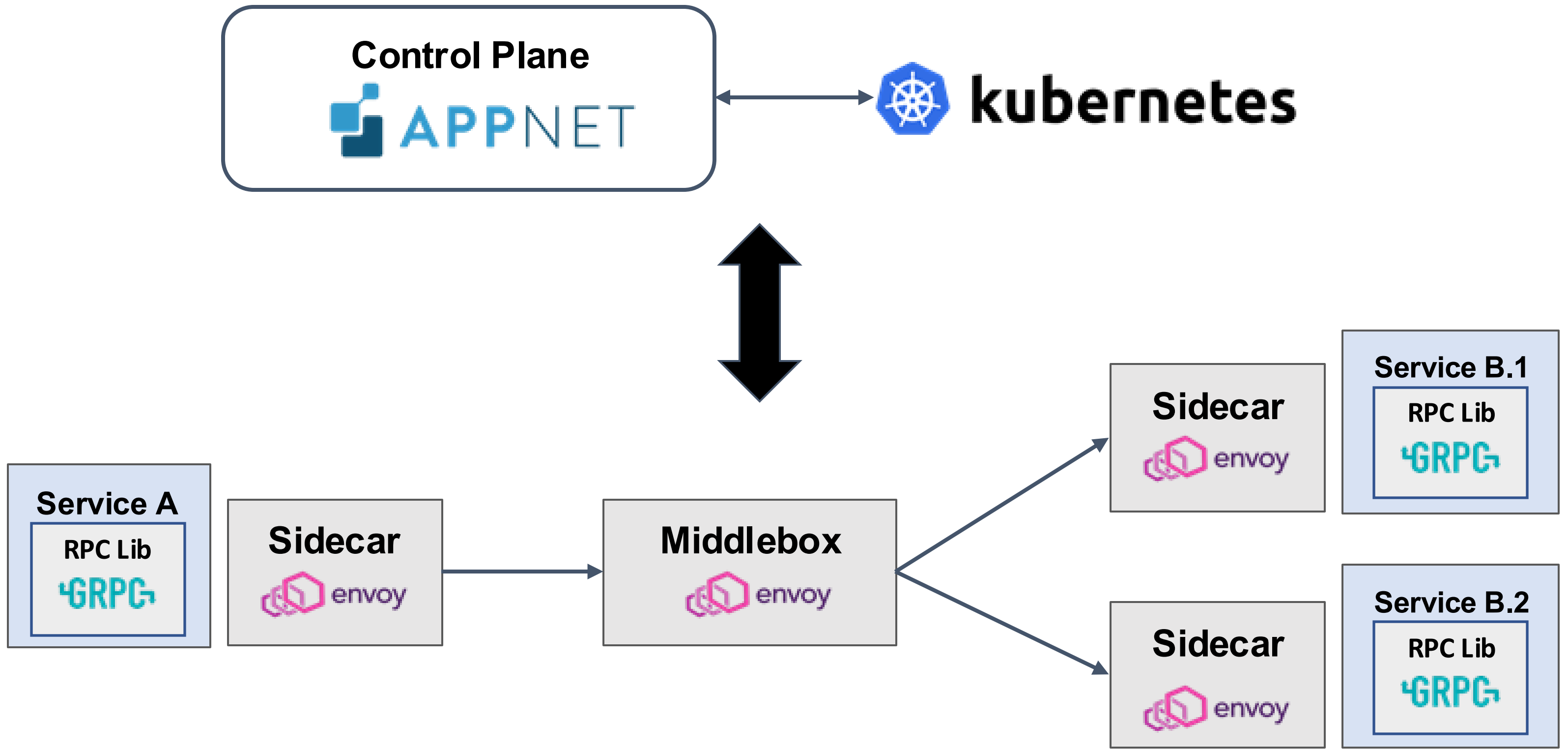
Example: Semantic Inequivalence



Equivalence Checking: Symbolic Execution



Implementation



Evaluation Questions

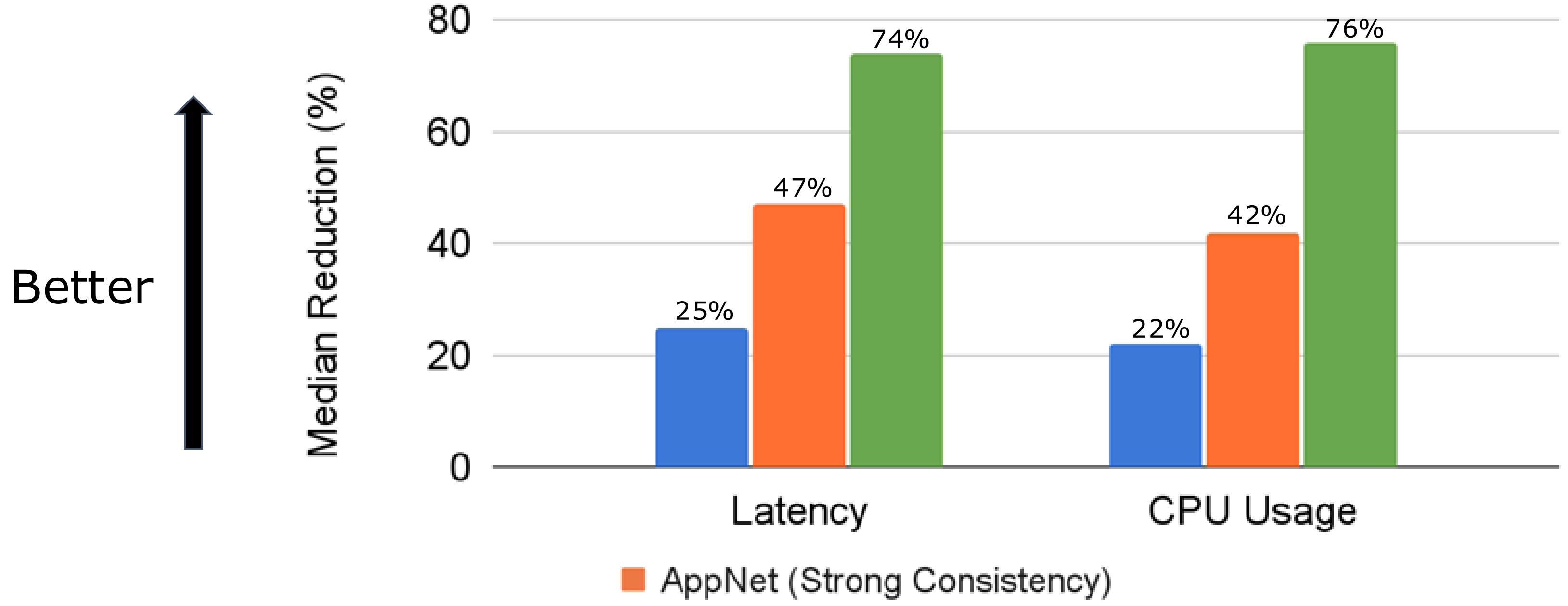
- **Expressiveness**
  - Can AppNet easily express common ANFs?
- **Performance**
  - Can AppNet reduce overhead and improve application performance?

AppNet Simplifies ANF Development

- 12 common ANFs can be implemented in 7-28 LoC
- Meta’s ServiceRouter and Google’s Prequal in < 100 LoC

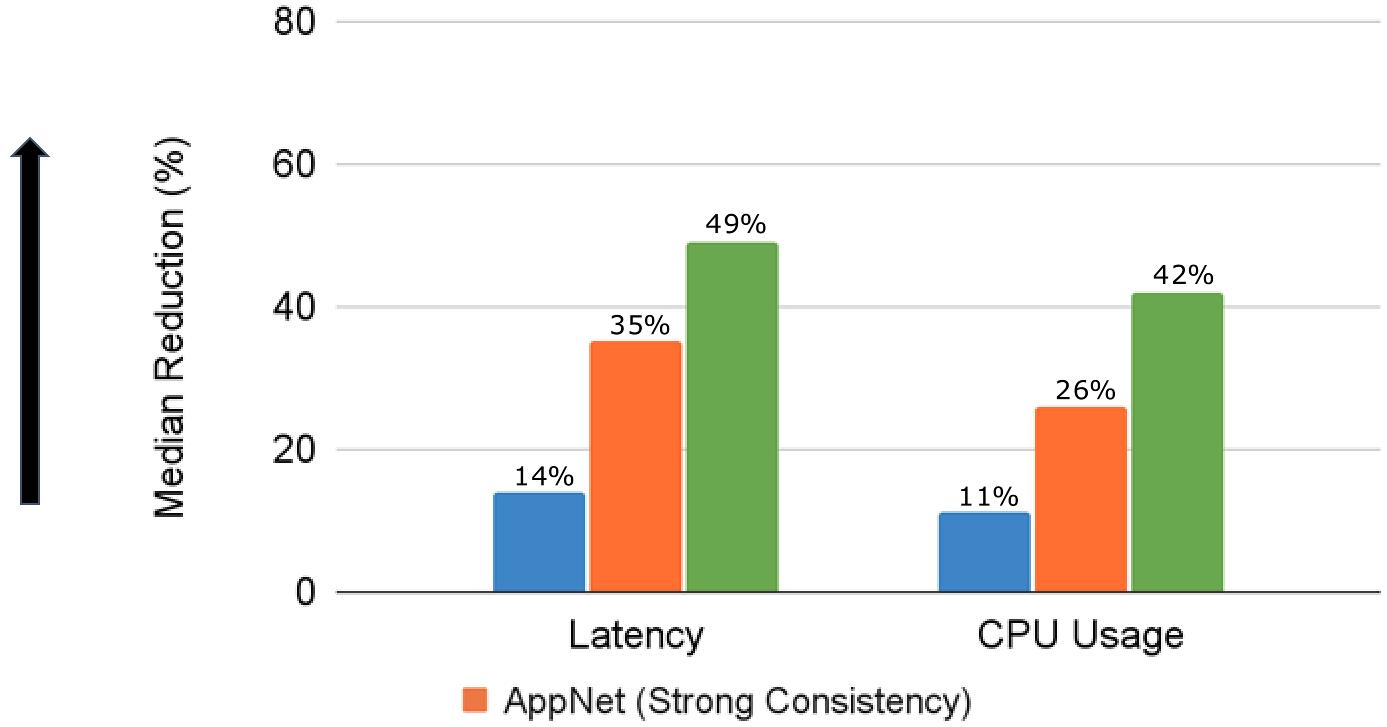
Reduce LoC by 5–60× compared to manual implementation

AppNet Reduces RPC Processing Overhead



AppNet Improves Application Performance

Better



APPNET

- Application networks today are hard to use and have poor performance
- AppNet **decouples specification from implementation**
  - Auto-generates efficient implementations across platforms
  - Optimizes performance based on platform and user policy

https://github.com/appnet-org/appnet

https://appnet.wiki/