DYNAMIC SERVICE CHAINING

WITH

DYSCO

forcing packets through "middleboxes" for security, optimizing performance, enhancing reachability, etc.

Pamela Zave

AT&T Labs—Research

Ronaldo A. Ferreira

UFMS, Brazil

Xuan Kelvin Zou

Google

Masaharu Morimoto

NEC Corporation of America

Jennifer Rexford

Princeton University

SERVICE CHAINING: THE STATE OF THE ART

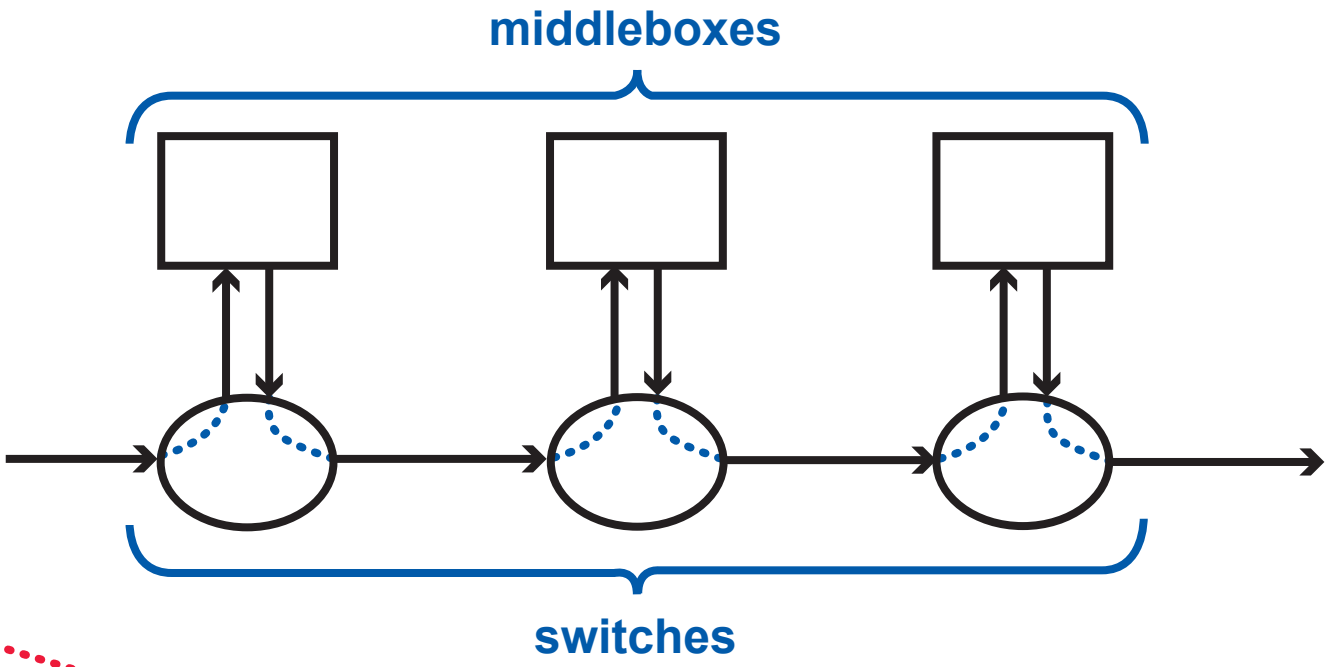
in a data center, each dotted line is a forwarding rule for directing packets of a flow through this service chain

the number of rules is on the order of (10K•chain length•2)

central controller must respond (in real time) with rule updates for failures, traffic fluctuations, resource scaling

What if a box changes packet headers, so packets coming out of the box do not match the forwarding rules?

What if a box classifies packets, to send some on a different service chain?



What if the flow path changes, but packets of existing sessions must still be sent to the original boxes? ("session affinity")

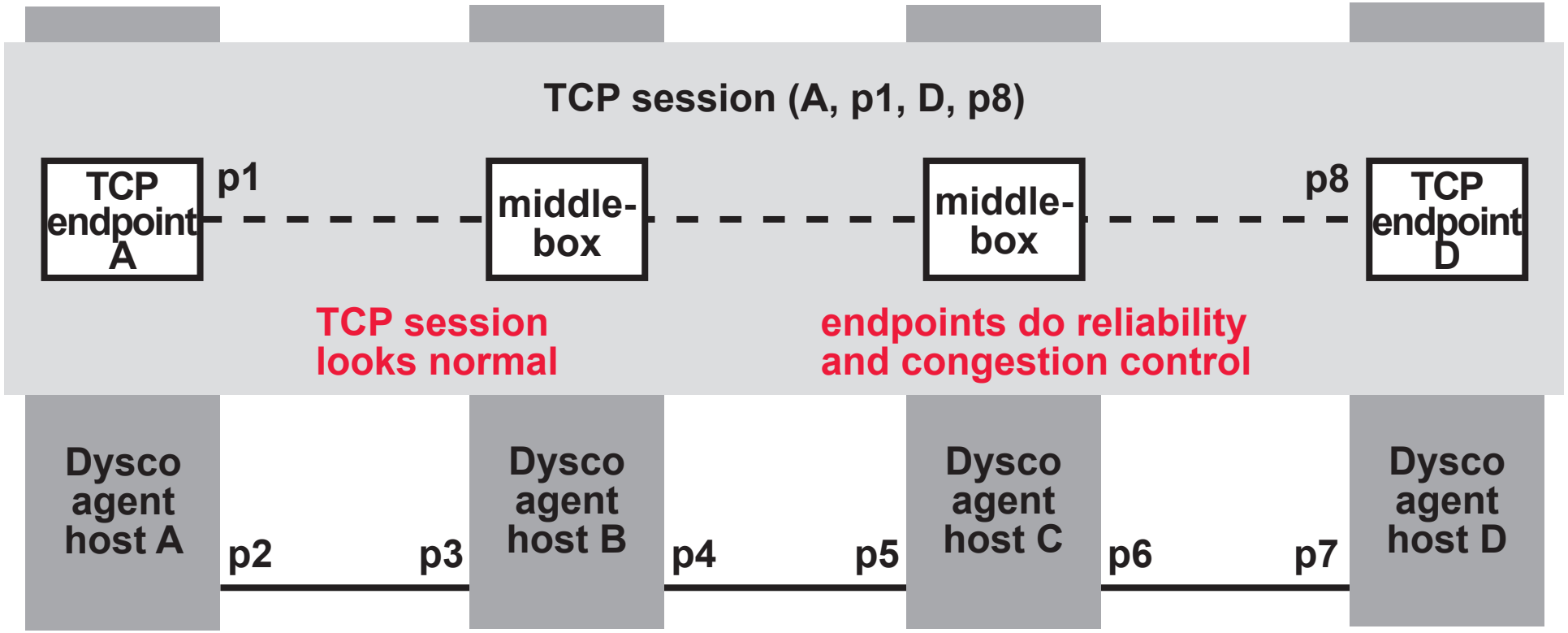
there has been progress on some of these problems,

but with so many of them

(5 already, 3 to go),

maybe it's time to consider a true end-to-end approach

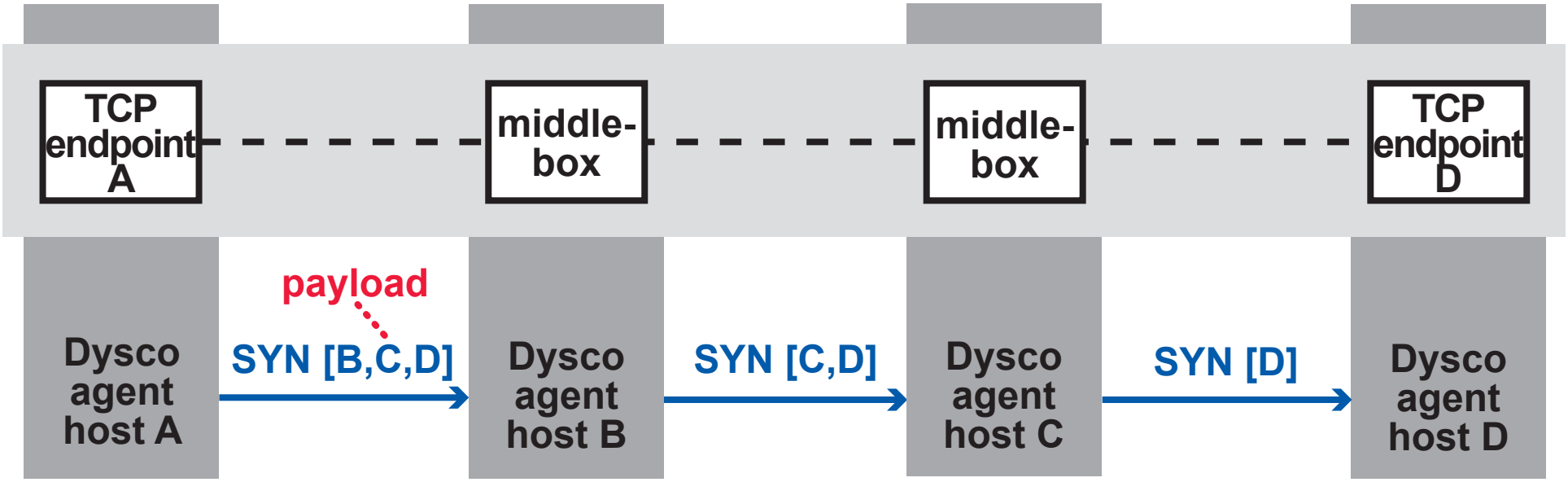
DYSCO IS A SESSION PROTOCOL FOR SERVICE CHAINING



service chain is set up as part of the TCP SYN handshake

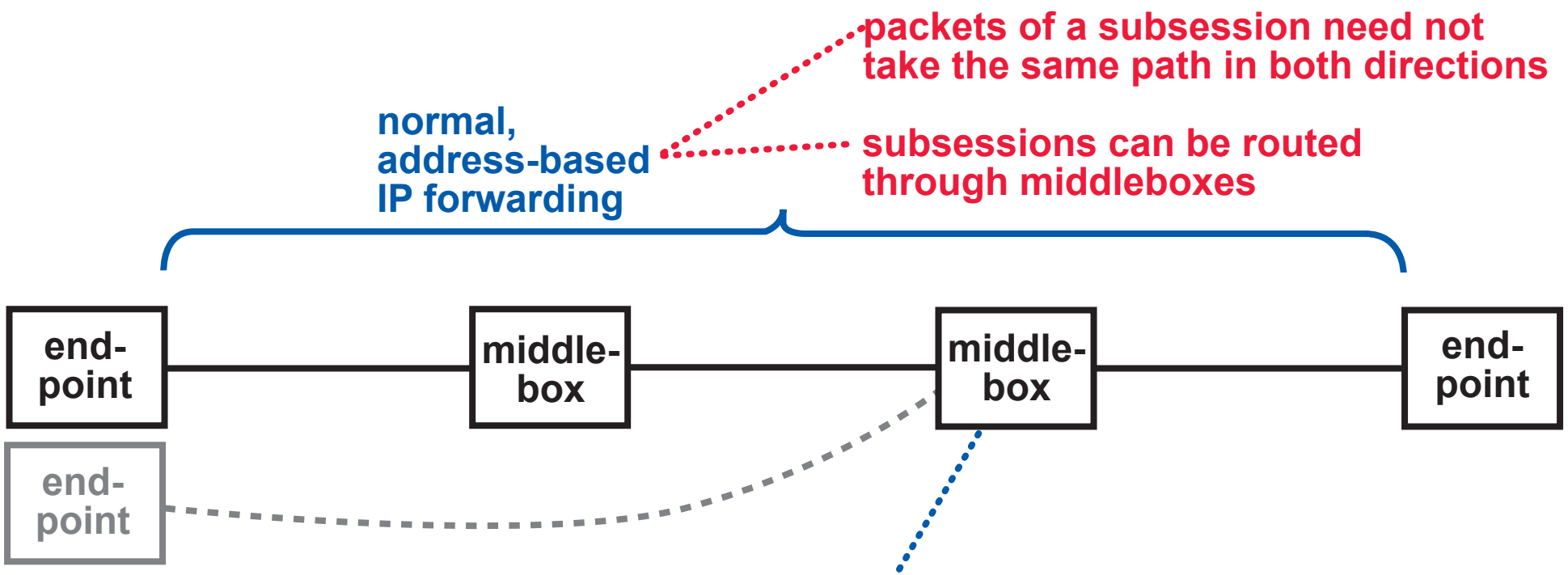
packets in the network are rewritten to have the 5-tuples of their subsessions

DYSCO AGENTS HAVE AUTONOMOUS POWER



- any agent can cache policies (abstract or concrete service chains) obtained from a policy server
- session affinity comes for free
- most middleboxes run unmodified—Dysco is transparent to them
- Dysco agents maintain the mapping between TCP 5-tuples and subsessions, so a middlebox that modifies the 5-tuple is no problem
- with an API, a middlebox can classify SYN packets and tell the Dysco agent where to send them next
- there is little more state than what was already present in stateful endpoints and middleboxes

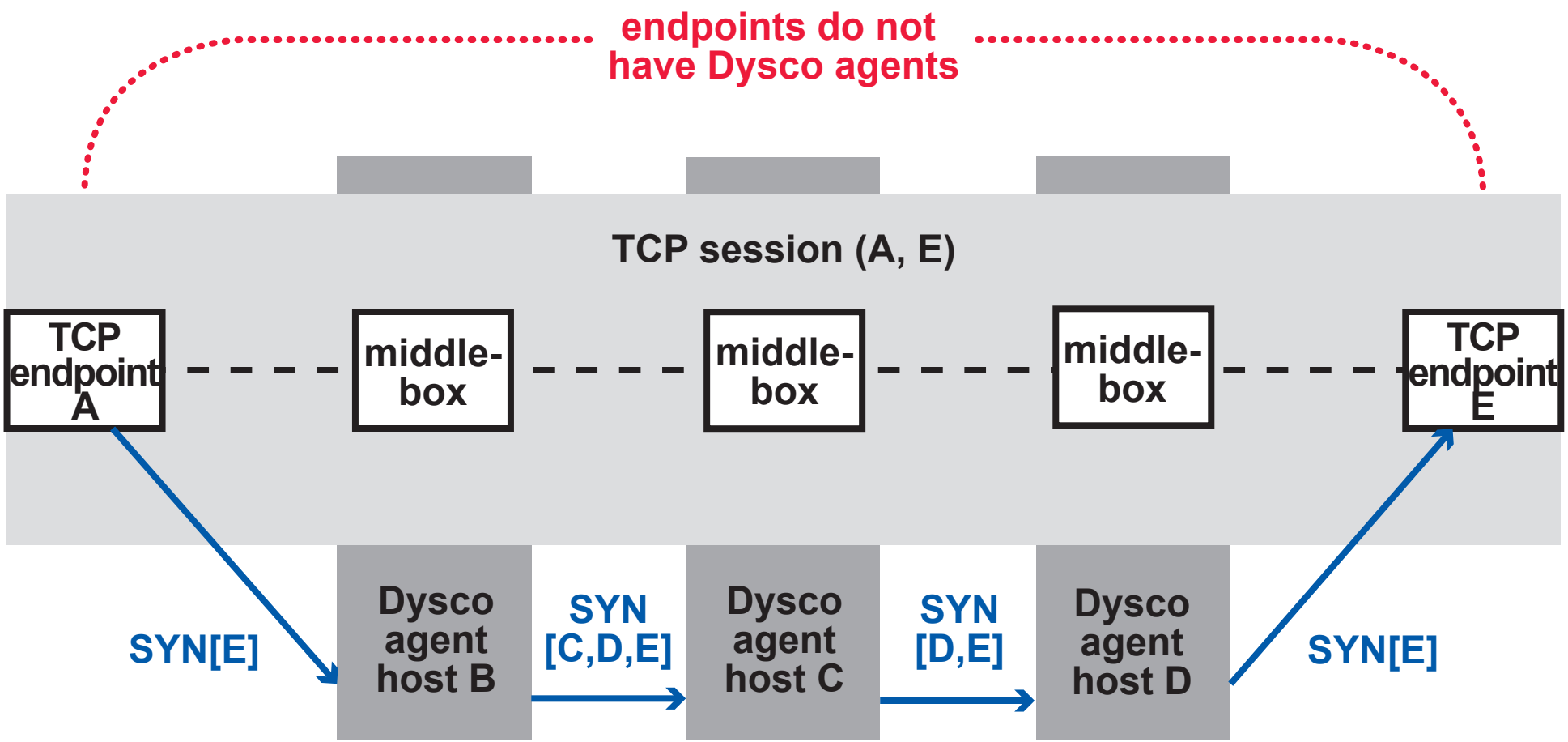
DYSCO SERVICE CHAINING IS INDEPENDENT OF ROUTING



FURTHER MOTIVATIONS

- if an endpoint is multihomed (Multipath TCP), this approach makes it possible to merge all streams at one middlebox
- if a middlebox is outsourced to another cloud, this approach works across domain boundaries
- if traffic is encrypted (mcTLS), this approach makes it possible for Dysco agents to exchange keys

INCREMENTAL AND SECURE DEPLOYMENT



service chain starts wherever SYN is routed to a host with an agent

service chain extends only to trusted hosts

another service chain can start among another trust group

within a trust group, signaling can be protected by standard techniques (nonces, encryption)

DYSCO IN THE DATA CENTER

POLICY SERVERS

- decide which session goes through which service chain
- do nothing to enforce these decisions

DYSCO AGENTS IN HOSTS

- cache decisions
- do all the work of enforcing them

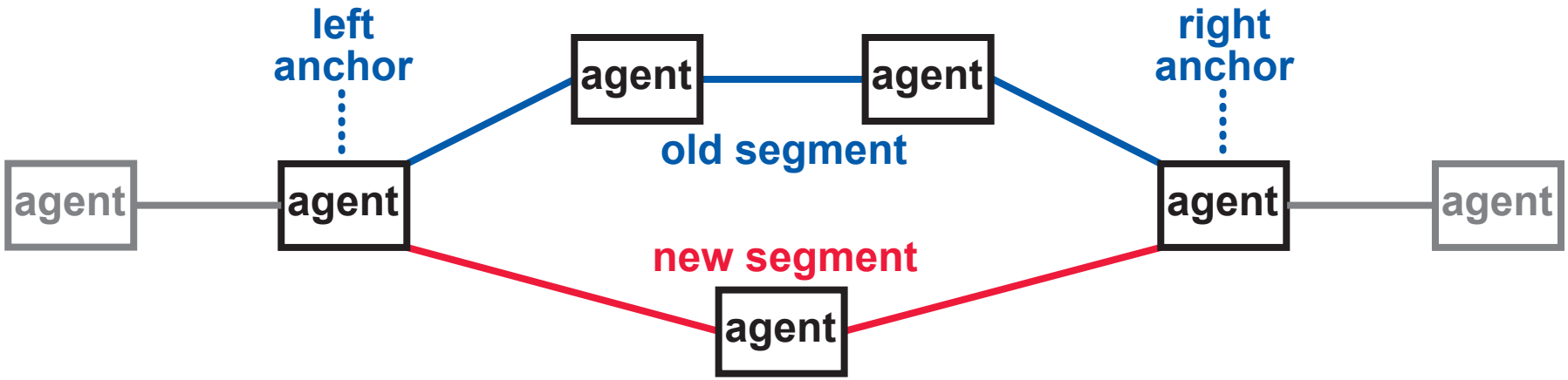
policy servers have an additional degree of freedom, because they can trigger DYNAMIC RECONFIGURATION of the service chain for an ongoing session!

WHY?

INSERT	. . . a packet scrubber when intrusion detection raises an alarm
	. . . a video transcoder during periods of network congestion
DELETE	. . . a load balancer after the server has been chosen
	. . . a caching proxy if the content is non-cacheable
REPLACE	. . . a middlebox that needs maintenance
	. . . a middlebox that has become a hairpin after endpoint mobility

HERE IS WHERE WE SHOW OUR TECHNICAL WIZARDRY:

PROTOCOL FOR DYNAMIC RECONFIGURATION OF A SERVICE CHAIN



GENERAL

- handles concurrent attempts to reconfigure a session
- handles failure to create a new segment
- can delete middleboxes even if they added/ removed data (altering sequence numbers)
- handles race conditions

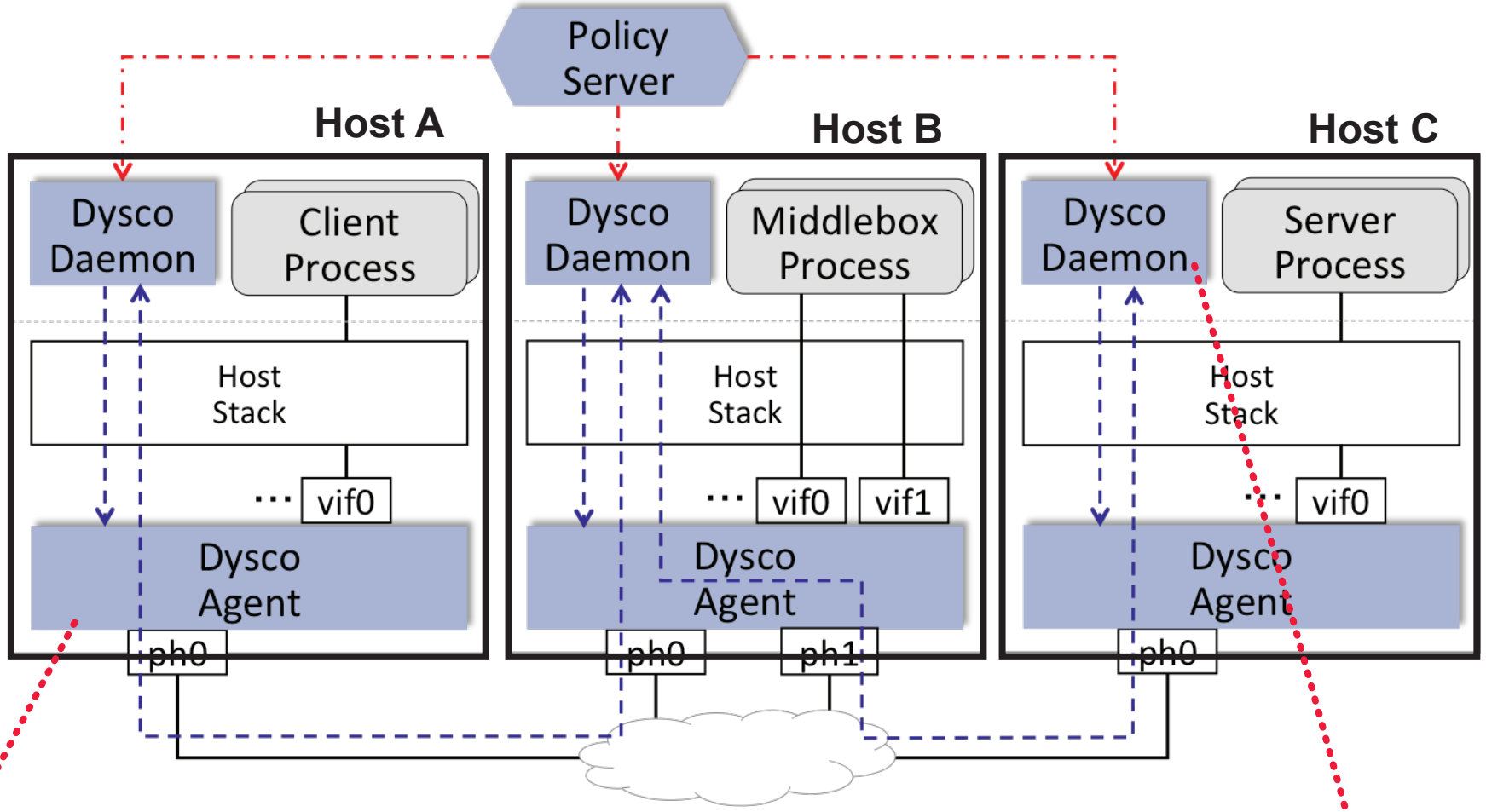
EFFICIENT

- no packet buffering required (unless old box state will migrate to new segment)
- TCP bytes acknowledged on the same path on which they were sent

VERIFIED CORRECT

- used the model-checker Spin
- no data loss, deadlocks, undefined cases, unnecessary failures, inconsistent sequence numbers, dirty terminations

PROTOTYPE IMPLEMENTATION



Agent is a Linux kernel module (could also use DPDK)

intercepts packets in the device driver

can be used with Docker, Mininet

computes incremental checksums

Daemon runs in user space

implements reconfiguration protocol

communicates with the Policy Server

DYSCO DEGRADES PERFORMANCE VERY LITTLE

SESSION INITIATION

- session initiation with 4 middleboxes
- worst case: checksum computation not offloaded to NIC
- average Dysco delay .094 ms

TCP GOODPUT

- 1000 sessions going through the same middlebox (link is saturated)
- worst-case Dysco penalty is 1.5%

SERVER REQUESTS PER SECOND

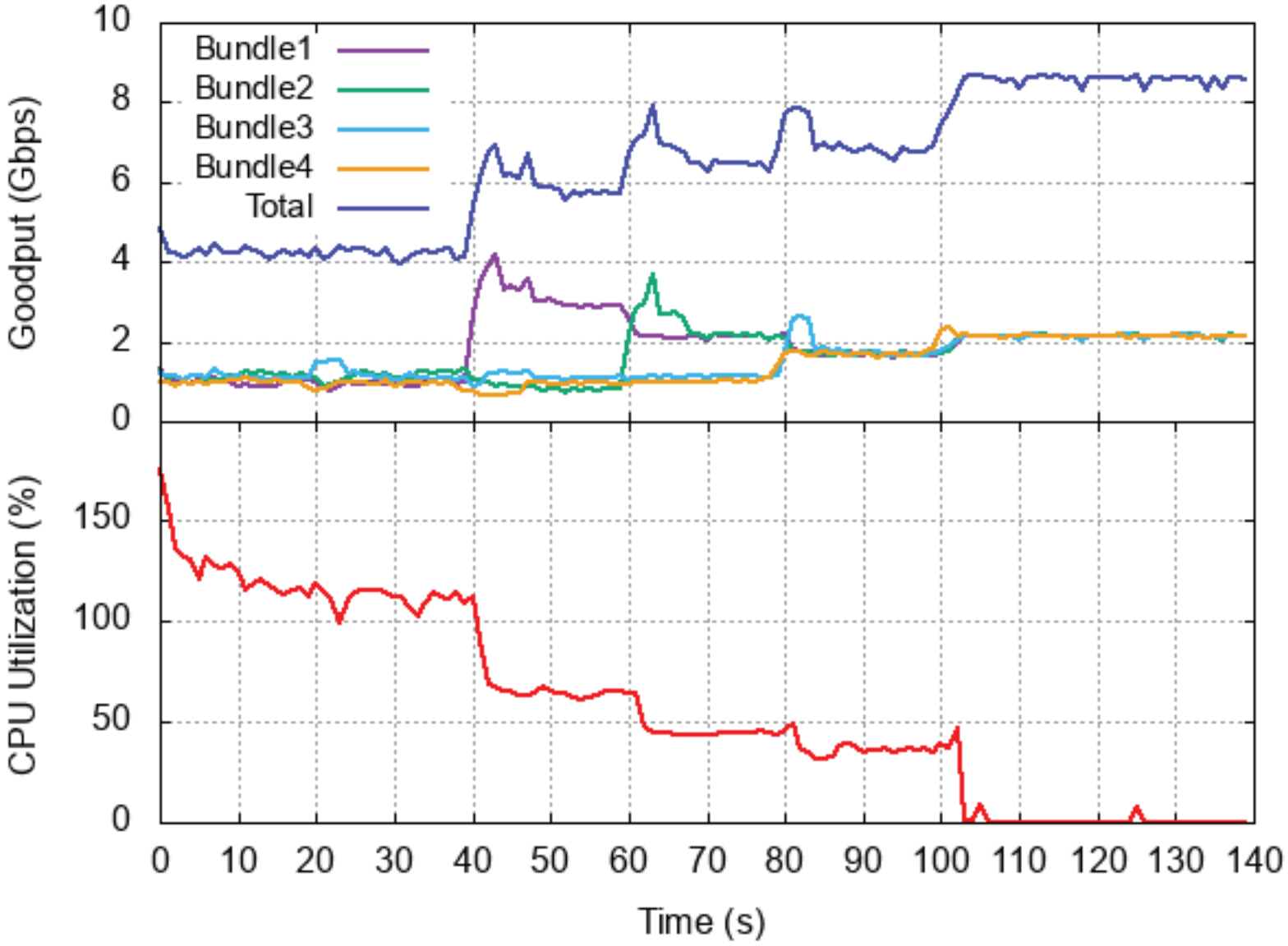
- we use NGINX HTTP server
- load is approximately 300,000 requests per second
- 4 middleboxes between the client and server
- worst-case Dysco penalty is 1.8% fewer requests per second

RECONFIGURATION IMPROVES PERFORMANCE

600 TCP sessions, each going through a proxy

at intervals, we trigger removal of the proxy from 1/4 of the sessions

80% of reconfigura-tions take less than 2 ms, none more than 4 ms



after all removals, CPU utilization at the proxy drops to zero, GOODPUT DOUBLES

CONCLUSIONS

LIMITATIONS

Reconfiguration protocol uses assumptions that do not always hold across domain boundaries . . .

. . . mainly because our TCP sessions need metadata and control signaling.

we are not alone!

e.g., Multipath TCP has similar problems

This is a problem crying for a general solution.

DYSCO IMPROVES ON . . .

. . . previous work that uses a session protocol or encapsulation for service chaining:

DOA
Connection Acrobatics
NSH

DYSCO APPEARS COMPATIBLE WITH . . .

. . . TCP-oriented protocols for . . .

. . . middleboxes that decrypt (mcTLS)

. . . multihoming (Multipath TCP)

. . . mobility (ECCP, TCP Migrate, msocket)

FUTURE DIRECTIONS

Integrate these efforts, using the findings of each to improve the others.

Compare directly with fine-grained forwarding rules as an approach to service chaining.