Minion:
An All-Terrain Packet Packhorse
to Jump-Start Stalled Internet Transports

Jana Iyengar*, Bryan Ford+
Dishant Ailawadi+, Syed Obaid Amin*,
Michael F. Nowlan+, Nabin Tiwari*, Jeffrey Wise*

*Franklin and Marshall College   +Yale University
Transports come and transports go ...

- **SCTP**
  - multistreaming, message boundaries, multihoming, partial reliability, unordered delivery
  - RFCs 4960, 3257, 3309, 3436, 3554, 3758, 3883 …
  - NAT behavior: draft-stewart-behave-sctpnat

- **DCCP**
  - Unreliable, congestion-controlled, datagram service
  - RFCs 4336, 4340, 4341, 4342, 5238, 5634, …
  - NAT behavior: RFC 5597
but the Internet remains loyal!

- TCP and/or UDP get through all middleboxes
  - UDP does not get through all middleboxes, but TCP does
  - (see paper for more on why UDP is insufficient)

- Other transports do not get through
  - SCTP and DCCP not supported by middleboxes
  - Practically impossible to get support for any new transport
How deep does this loyalty run?

- **Network Address Translators (NATs)**
  - Cheap and ubiquitous, entrenched in the network
- **Firewalls**
  - Rules based on TCP/UDP port numbers; possibly DPI
- **Performance Enhancing Proxies (PEPs)**
  - Transparently used for improving TCP performance
A taxonomy of transport functions

Functional Components in Transport Layer

- Application Layer
- (optional) Isolation Layer (security)
- Semantic Layer (services offered to app: reliability, ordering, etc.)
- Flow Layer (congestion control)
- Endpoint Layer (endpoint identification: port numbers)
- Network Layer

Middleboxes in the network and transport functions on which they interpose

- NATs, Firewalls
- Performance Enhancing Proxies (PEPs)
- Traffic Normalizers
- Corporate Firewalls
Deployment Impossibility-Cycles

Middlebox support for new transport

Apps using new transport

Market pressure through user demand

Performant implementations for popular OSes
What have we done so far?

- “NATs are evil. We won't care about them.”
- “It will all change with IPv6.”
- “Don't design around middleboxes, that will only encourage them!”
- “Alright, we'll specify how middleboxes ought to behave with different protocols. But they still have to behave.”
- “Why build a new transport?? It won't get deployed anyways.”

*Kübler-Ross model: Five stages of grief*
The final stage: Acceptance

- Design assumptions for new transport services:
  - New transport services should *require* modifications to *only* endhosts
  - Middleboxes are here to stay

- Consequences:
  - New end-to-end services *should not require* changes to middleboxes.
  - New end-to-end services must use protocols that appear as legacy protocols on the wire.

- Eg: MPTCP
The Minion Suite

- Uses legacy protocols ...
  - TCP, UDP, SSL

- ... as a substrate ...
  - turn legacy protocols into minions that offer an unordered datagram service

- ... for building new services that apps want
  - multistreaming, message boundaries, unordered delivery, optional congestion control
  - (working on: stream-level receiver-side flow control, multihoming and multipath, partial reliability)
What's in the Minion Suite?

- Reduce legacy protocols to *endpoint-* and *flow-layer* minions on which middleboxes can interpose.
- Build more sophisticated services on top of minions.

**Diagram:**
- Application Layer
  - Semantic Layer (services to app)
  - Isolation Layer (security)
- Flow Layer (congestion control)
- Endpoint Layer (endpoint identification)
- Network Layer

**Protocol Minions:**
- TCP minion
- UDP minion
- SSL minion
- DTLS (optional)
- Semantic SCTP, Semantic DCCP, Semantic SST, or any other new transport deployed with an application

**Transport Protocols:**
- IPv4, IPv6
- TCP
- UDP
- DTLS
- SSL
- TCP (sans CC)
- SSL (sans CC)
TCP Minion

- Retain TCP protocol semantics on the wire
  - Connection-oriented → setup/teardown preserved
  - Fully reliable → retransmissions
  - Byte-stream → re-segmentation in network tolerated

- Provide datagram service to app/semantic layer
  - embed upper layer messages in byte-stream
  - extract and deliver messages at receiver from byte-stream without regard to order (COBS encoding)
  - (cannot forgo TCP retransmissions → reliable datagram service)
TCP minion in operation

At app sender
- m3
- m2
- m1

App messages
- m3
- m2
- m1

At TCP-minion sender
- m3'
- m2'
- m1'

Encoded app msgs
- m3'
- m2'
- m1'

On the wire
TCP segments
- TCP segment 2
  - m3' m2_2
  - m3 m2_1 m1

TCP segment 1
- m2_1 m1

At TCP-minion receiver
- m3'
- m2'
- m1'

Encoded msgs extracted from received TCP segments
- m3'
- m2_2

Decoded app msgs
- m3
COBS encoding

- Size-preserving encoding that eliminates all occurrences of delimiter byte
  - Max overhead of 0.4% (5 bytes for 1250-byte msg)
  - Delimiter byte then inserted between messages
  - Receiver extracts messages, decodes, delivers up
- We make one modification
  - We insert delimiter byte both before *and* after msg
    - Increases max overhead to 0.8%
  - To deal with common cases for apps
    - App sends only one message (eg: HTTP GET req)
    - Each app msg gets encap'd in its own TCP segment
App messages with TCP (TLV encoding) vs. TCP-minion

![Graph showing comparison between TCP and TCP-minion]

- App Message Sequence Number (1195-byte msgs)
- Time received at app (seconds)
Stacking new services

- Semantic SCTP:
  - message boundaries, multistreaming, unordered delivery, multihoming, multipath, (partial reliability)

- Semantic DCCP:
  - TCP-minion service is exactly the same as DCCP with TCP-like congestion control (CCID-2)
  - negotiate CC on top of TCP-minion, and change CC algo used in kernel during runtime

- Semantic SST:
  - receiver-side per-stream flow control
  - stream prioritization
SSL Minion

- SSL-minion protects end-to-end signaling and data,
  - appears as SSL on the wire, and
  - provides a reliable datagram service
- App messages are broken into SSL records at sender, and authentication code (MAC) is appended
- Receiver uses SSL’s basic record header as a “weak” recognizer of a record delimiter
  - record authentication successful → record delimiter accurate!
UDP Minion

• Provides UDP encap of new transport
  – Similar to “GUT” proposal
  – Importantly, contains accurate app endpoint information: UDP source/dest port numbers are the ports that apps are bound to.
Our implementation of the minions

- Some inside Linux kernel, the rest in userspace libraries
- Added SO_UNORDERED sockopt to SOCK_STREAM
  - subsequent read()s results in a contiguous byteblock being returned, without regard to order
  - TCP sequence number returned with byteblock
  - This minor change is the only one required in-kernel
- Userspace library for rest of TCP- and SSL-minion
  - reassembles byteblocks, extracts message, decodes, and delivers up
  - can ship as part of apps
In Conclusion

- TCP, SSL, UDP work on the Internet
  - mature, performant implementations
  - workhorses of the Internet

- We can implement new services by modifying ends and retaining on-the-wire protocols
  - Most mods deployable with apps
  - Turn workhorses into packhorses!