ICT-317756

TRILOGY2

Trilogy2: Building the Liquid Net

Specific Targeted Research Project

FP7 ICT Objective 1.1 The Network of the Future

D2.4 Advanced Tools for Controlling Liquidity

Due date of deliverable: 31 December 2014

Actual submission date: January 15, 2015

Start date of project	1 January 2013
Duration	36 months
Lead contractor for this deliverable	Universidad Carlos III de Madrid
Version	v6.0, January 15th, 2015
Confidentiality status	Public

Abstract

Trilogy 2 sets out to create the liquid net acrosss three dimensions: cross-resource, cross-layer and cross-provider. Cross-resource liquidity is all about sharing transport, processing and storage resources between users, cross-provider means allowing liquid resources to be shared among providers and cross layer is about mecahnisms for liquidty between the different layers of the network. This deliverable describes tools that allow end users and operators to exercise control over the liquid net. They include tools to enable resource sharing, mechanisms for resource monitoring and security approaches for MPTCP. Key to controlling the liquid network is the recognition that users and operators will be in a tussle over resources, so we include tools that allow senders to test the feedback they get from receivers as well as tools to ensure accurate information exchange between end-users and operators and among operators.

Target Audience

Ultimately the target audience of this deliverable is the community of operators and end-users that will make use of the liquid network. More immediately this deliverable is targeted at engineers who define the structure of ICT systems, and those who define the standards and frameworks that are necessary for these ICT systems to interwork across the industry.

Impressum

Full project title	TRILOGY2: Building the Liquid Net
Title of the workpackage	D.2.4 Advanced Tools For Controlling Liquidity
Editor	Toby Moncaster, UCam
Project Co-ordinator	Marcelo Bagnulo Braun, UC3M
Copyright notice	© 2015 Participants in project TRILOGY2

Executive Summary

Trilogy 2 sets out to create the liquid net acrosss three dimensions: cross-resource, cross-layer and cross-provider. Cross-resource liquidity is all about sharing transport, processing and storage resources between users, cross-provider means allowing liquid resources to be shared among providers and cross layer is about mecahnisms for liquidty between the different layers of the network. In order to achieve this Trilogy 2 has produced a number of tools to create and control this liquidity. This deliverable explores some of the advanced tools for the control of liquidity cross-resource and cross-provider. One of the major aims of these tools is to reflect what the DoW calls the "tussle between interests that sometimes align and sometimes conflict".
Broadly we can describe the tools in terms of the three functions they enable:

- Tools for controlling the sharing of resources between users. These are fundamental to the liquid network and include tools that are designed to create and leverage a market in resource liquidity.
- Tools for securing information exchanged across the liquid network. This is important in a network where trust cannot be taken for granted and where conflicting interests may result in non-optimal sharing of resources.
- Tools for monitoring resource usage. Without the ability to accurately monitor resource usage it will never be possible to make efficient use of the liquid network.

Central to all of these is the need for accurate and timely information exchange and so we have included discussion of tools that allow both end users and network operators to verify the accuracy of information relating to the state of resources in the liquid network. We have chosen to divide our description of the tools into those primarily intended for the end-user and those that give control to operators.
Chapter 2 describes the tools that to provide end users with control over their use of the resources. MPTLS and SMTCP are two different approaches to securing MPTCP that make different assumptions about likely usage scenarios. Secure Multipath TCP (SMTCP) uses tcpcrypt to encrypt both the data and signalling in MPTCP. As the signalling is protected there is no need for SMTCP to include the existing MPTCP security and HMAC mechanisms. SMTCP is still vulnerable to a man-in-the-middle attack during intitial connection negotiation but is secure against other forms of attack. Multipath TLS (MPTLS) integrates MPTCP with TLS. It changes the MPTCP semantics from an ordered bytestream to a reliable message service with each message equating to one TLS block. Each message is sent in its entirety along a single MPTCP sub-path. MPTLS uses the encrypt-then-mac technique. Encrypted TLS blocks are passed to MPTCP as messages. Each message is then MACed. The TCP receiver test is a simple mechanism to allow senders to test the accuracy of the feedback they are sent by their receivers. This is important in a network where users are in a constant tussle with each other and may not be able to trust the feedback they see. Inner Space is an extension to the TCP protocol designed to overcome the singificant limitations of the TCP option space (both in terms of the length of options and in terms of which options can traverse middleboxes). By providing an independent

channel within the TCP data Inner Space allows new end-to-end and end-to-middle signalling channels. This in turn acts as an enabler for many novel transport mechanisms including simplifying the design of the secure versions of MPTCP listed above.

Chapter 3 presents tools that allow operators to control and monitor their resources. FUBAR allows operators to make traffic scheduling decisions based on maximising the actual utility of end-users. Flows are shifted according to two measures of utility: bandwidth and latency. FUBAR offers significant utility improvements over traditional shortest path routing. Congestion Exposure or ConEx provides mechanisms to allow operators to monitor the state of the network and to enforce resource sharing. These rely on a new approach for improved ECN feedback called Accurate-ECN. ConEx includes the mechanisms for users to expose congestion and for operators to check the accuracy of the declared congestion. Finally the Federated Market provides the necessary tools to allow compute and storage resources to be freely traded across the global network as a liquid commodity. This directly realises the aim of a truly liquid network and OnApp's Cloud.net already makes use of the technology.

List of Authors

Authors	Toby Moncaster, John Thomson, Bob Briscoe, Olivier Bonaventure, Nikola Gvozdiev, Mark Handley, Marcelo Bagnulo Braun
Participants	UCam, OnApp, BT, UCL-BE, UCL-UK, UC3M
Work Package	D.2.4 - Advanced Tools For Controlling Liquidity
Security	PUBLIC (PU)
Nature	R
Version	v6.0
Total number of pages	81

Contents

List of Figures

List of Tables

1 Introduction

Liquidity is about sharing transport, processing and storage resources between users. Trilogy 2 has worked on the design of tools for creating liquidity (WP1) and tools for controlling liquidity (WP2). This deliverable explores some of the advanced tools for the control of liquidity. These tools cover information security, resource sharing, resource control and monitoring. Central to all of these is the need for accurate and timely information exchange so we have included discussion of tools that allow end users and network operators to verify the accuracy of information relating to the state of resources in the liquid network.

We have chosen to split these tools between ones principally intended to provide end user control and ones that allow operators to exercise control. In the former we include tools for improving the security of MPTCP, a simple approach to allow senders to test whehter their receivers are sending accurate feedback and Inner Space, a novel technique to significnantly extend the TCP option space. In the latter we include FUBAR, which allows operators to make routing decisions based on flow utility, tools relating to policing congestion and ConEx and the Federated Market which provides a mechanism to allow compute and storage resources to be traded between operators.

MPTCP is a key tool for providing liquidity across the network, however it is known to be potentially vulnerable to man in the middle and other attacks. MPTLS and SMTCP are two different approaches to securing MPTCP making different assumptions about the likely usage scenarios. They both build on the existing MPTCP protocol but MPTLS changes it from a stream protocol to a reliable multipath message protocol, using TLS to encrypt the data and MPTCP to MAC and transmit each TLS chunk on an independent sub stream. SMTCP keeps the same semantics as MPTCP but uses tcpcrypt for securing both the data and signalling. This reduces the vulnerability significantly, but still leaves it open to man in the middle attacks during initial connection negotiation.

The TCP receiver test is a simple mechanism to allow senders to test the accuracy of the feedback they are sent by their receivers. This feedback is used both to estimate the RTT (and hence the effective bandwidth) and to notify the sender of any congestion. TCPs feedback mechanism is built on trust which was a good model in the early days of the Internet, but simple self interest on the part of the receiver means that this model breaks down in a network where users are in a constant tussle with each other. The tests work by imitating re-ordering and congestion and checking to see if the receiver accurately responds to this.

Inner Space is a novel approach designed to overcome the difficulty with extending the transport layer. The transport layer has become hard to extend, for two main reasons:

- Many middleboxes only forward the two original transport protocols (TCP and UDP) and sometimes only the one dominant application layer protocol (HTTP). To make matters worse, they often only forward TCP headers that conform to the stereotype they expect;

- The TCP option space was limited to 40 B when TCP was first defined [54].

Inner Space overcomes this by embedding additional control and option data within the data portion of packets that *look* like TCP packets to any middlebox. Inner Space removes the limitations of option length making it much easier to deploy mechanisms such as tcpcrypt. It also opens up opportunities to deploy entirely new transport protocols and to ensure they will pass middleboxes unhindered. Inner Space also enables end-to-middle communications—a potentially important aspect for the liquid net where negotiating with middleboxes may become a key part of increasing resource liquidity.

FUBAR allows operators to make traffic scheduling decisions based on maximising the actual utility of end-user's flows. It does this by estimating their utility functions along two axes—latency and bandwidth. It then runs a flow balancing algorithm that aims to maximise the utility across all flows while avoiding becoming trapped in local maxima. FUBAR performs significantly better than traditional shortest path routing and is computationally tractable.

Congestion Exposure or ConEx provides mechanisms to allow operators to monitor the state of the network and to enforce resource sharing. ConEx was a product of Trilogy 1 and most of the work on the protocol was completed outside Trilogy 2 . In this deliverable we report on two new approaches for congestion policing (including new ways to ensure congestion feedback is trustworthy). as well as Accurate ECN—a way to allow precise feedback of ECN markings by encoding counters within the existing ECN feedback mechanism in TCP.

Finally the Federated Market provides all the tools to allow compute resources to be traded across the global network as a liquid commodity. Such trading is a vital element in the liquid net and we show how to enforce trust and security in such a model. We also discuss how such flexibility can be used in real world scenarios.

2 User Control

In this section we explore some of the tools that will help users to exert control over their use of the liquid network. If end-users are to make full use of the liquid network then they must trust that their data is secure. We present two possible approaches for improving the security of MPTCP. Another key aspect is trust between end-points. We present a simple set of tests that a sender can use to verify that their receiver is sending accurate feedback about the state of the connection. Finally the ability to signal end-to-end across the network is currently limited due to restrictions on option space within the standard TCP header. This has a direct impact on protocols such as MPTCP that rely in additional in-band signalling for control. We describe a new approach that embeds additional options within the data rather than the header.

2.1 Securing TCP/MPTCP

Multipath TCP (MPTCP) is a mechanism to split a single TCP connection across multiple sub-paths. This allows better resource pooling and improved resilience. MPTCP was one of the major deliverables to come out of Trilogy 1. Having been standardised it is now seeing increasing real-world adoption. While it is rapidly maturing as a standard, MPTCP is known to be vulnerable to certain security threats [4]. Within the context of Trilogy 2 we have investigated two contrasting approaches to securing MPTCP. These make different assumptions and are suited to different use cases.

Secure Multipath TCP (SMTCP) uses tcpcrypt to encrypt both the data and signalling in MPTCP. As the signalling is protected there is no need for SMTCP to include the existing MPTCP security and HMAC mechanisms. SMTCP is still vulnerable to a man-in-the-middle attack during intitial connection negotiation but is secure against other forms of attack.

Multipath TLS (MPTLS) integrates MPTCP with TLS. It changes the MPTCP semantics from an ordered bytestream to a reliable message service with each message equating to one TLS block. Each message is sent in its entirety along a single MPTCP sub-path. MPTLS uses the encrypt-then-mac technique. Encrypted TLS blocks are passed to MPTCP as messages. Each message is then MACed. This is motivated by two reasons. First, Multipath TCP already verifies the received data by using its optional DSS checksum. We replace this checksum with a cryptographic MAC authentication that has strong security properties. Second, if Multipath TCP receives a TLS record with an invalid MAC it can simply discard the data and wait for its retransmission or perhaps terminate the affected TCP subflow and create a new one to retransmit the data.

2.2 Securing MPTCP With tcpcrypt

Multi-path TCP (MPTCP) [28] defines extensions to TCP that allow transmitting data over multiple paths in a single TCP connection. This is achieved by opening multiple subflows within the same TCP connection. Each subflow is associated with a different address/port pair. As currently defined, MPTCP provides basic security for the signaling used to establish the subflows. A threat analysis for MPTCP is presented in [3] and a residual threat analysis is presented in [4] and in Deliverable D1.1. From these analyses we can see that

MPTCP as currently defined is vulnerable to attackers that can eavesdrop the initial connection establishment exchange and also to attackers that can intercept any subflow establishment exchange. In addition, MPTCP does not provide any protection to the data stream (other than splitting the data stream over multiple paths), as this was a non goal of the MPTCP design. In Deliverable DD1.1 we concluded that if a more secure version of MPTCP should be pursued, the path to follow would be to protect the data stream rather than trying to provide additional security to the signaling. The reader is referred to the aforementioned reference for additional insight why this is the case. The goal of this document is provide initial considerations about how to provide enhanced security to MPTCP by securing the data stream.

In this section, we analyze the use of tcpcrypt [7] to secure MPTCP. Tcpcrypt defines extensions to opportunistically encrypt the data stream of a TCP connection. By using tcpcrypt in MPTCP, we would be able to provide enhanced security for MPTCP. We note however, that the resulting solution would still be vulnerable to Man-in-the-Middle attacks during the initial key negotiation. However, the attacker in this case must be active and must remain located along the path during the whole lifetime of the connection.

We call this combination of tcpcrypt and MPTCP 'SMTCP' (Secure Multipath TCP). This provides stronger security than MPTCP, both for the signaling and for the data. Since all the MPTCP signaling will be protected by tcpcrypt (i.e. encrypted and its integrity protected) there is no need for the existing MPTCP security mechanisms. This means that there is no need to negotiate an MPTCP key and that the HMAC protection provided by the MPTCP protocol is not needed (except for backward compatibility). All the protection will be achieved with the tcpcrypt extensions.

2.2.1 Initial SMTCP connection

Suppose both A and B are SMTCP capable. Suppose A has both $IPA1$ and $IPA2$. Suppose A initiates an SMTCP connection with B. The exchange would be as follows:

- $A \rightarrow B$: `SYN + MP_CAPABLE` (including A's key (ka) and with the `C` bit set) + `CRYPT/Hello` This contains 15 bytes of options (the motivation for including both the `MP_CAPABLE` and the `CRYPT` option is for backward compatibility. `SYN` packets usually also carry several other options: MSS (4 bytes), SACK (2 bytes) and Window-Scale (3 bytes). Negotiating timestamps would be 10 more bytes. TCP's option space has a maximum length of 40 bytes, so all the above options can fit.

- $B \rightarrow A$: `SYN/ACK + MP_CAPABLE` (including B's key (kb) and with the `C` bit set) + `CRYPT/PKCONF` (with pub-cipher-list) Assuming we have 2 algorithms in the list, this is 10 bytes, making a total of 22 bytes. This packet also usually carries the same options as the SYN packet, so in this case not all the additional options would fit.

- $A \rightarrow B$: `INIT` (3 bytes of options) plus crypto data in the payload (The `MP_CAPABLE` option is needed since the keys are generated by tcpcrypt).

- $B \rightarrow A$: `INIT` (3 bytes of options) plus crypto data in the payload.

During the 3-way handshake MPTCP generates the following values:

- The **key**: a session key to protect the signaling (one for each side of the connection)
- The **Token**: used as connection identifier (one for each side)
- The **IDSN**: Initial Data Sequence Number (one for each side)

The idea would be to derive them from the tcpcrypt values.

- The keys: tcpcrypt generates 4 keys, kec, kac, kes and kas. These will be used to secure MPTCP, as discussed later on. For the purposes of MPTCP signaling, the keys that will be used are the authentication keys, so the keys for MPTCP are kac and kas.
- The tokens: tcpcrypt generates a Session ID, which is the full length of a hash output. The key point is that tcpcrypt generates a single `SID` for both endpoints, while MPTCP generates one per endpoint. In addition, MPTCP needed to generate a pair of IDSNs.

We could then generate the MPTCP values out of the tcpcrypt values as follows

- $KeyA = kac$
- $KeyB = kas$
- $TokenA = 32$ most significant bits of $(hash(ka))$
- $TokenB = 32$ most significant bits of $(hash(kb))$
- $IDSNA = 64$ least significant bits of $(hash(kac + SID))$
- $IDSNB = 64$ least significant bits of $(hash(kas + SID))$

2.2.2 Adding new subflows

The question is whether to treat a new subflow as a new tcpcrypt connection or not, the implication being that a new tcpcrypt connection uses a different shared secret and hence different keys (even though no new public key operations are needed). Probably this is not the way to go. All the flows of a MPTCP connection should be part of the same tcpcrypt session.

So, one simple way of doing this would be to simply use the existent MPTCP exchange to add a new subflow with the MPTCP security measures. This implies sending an `MP_JOIN` containing the receiver's token and a random number which will be responded with another `MP_JOIN` and the final `JOIN` message. The tcpcrypt keys are used instead of the regular MPTCP keys.

A alternative approach would be to drop completely the MPTCP security mechanisms and use the tcpcrypt MAC option to secure the MPTCP signaling. This implies that the tcpcrypt MAC option would need also to protect the MPTCP `MP_JOIN` option

2.2.3 Exchanging data

Once the keys and the other values have been negotiated, data can flow. All data in the MPTCP connection will be encrypted with tcpcrypt keys and its integrity protected using the tcpcrypt MAC option. This adds 22 bytes of options (assuming a 160 bit long hash).

Now, MPTCP includes the DSS option in order to synchronize the data sequence number with the sequence numbers of the subflows. The `DSS` option max length is 28 bytes. The results it that the `MAC` option plus the `DSS` option are 50 bytes, which is a problem. The good news is that the `DSS` does not need to be sent in every segment and that 28 bytes is the maximum length.

Currently the `DSS` option includes information both about `DSN` mapping to subflow sequence number and data ack. In order to limit the size of the option, one option is to prevent both the `Data Ack` and `DSN` subflow sequence numbers mappings being sent in the same option. This would mean that when `Data Acks` are sent, the `DSS` option has a maximum of 12 bytes and when `DSN` to subflow sequence number mappings are sent, the max length is 20 bytes. This is still 2 bytes too long.

There are two ways we can shrink this. One option is to prevent the use of `Checksum` when tcpcrypt is used. `Checksum` is optional, so this could be done. Moreover, it makes sense to do this, because all the information protected in this checksum is protected by the tcpcrypt `MAC` option. This means that the `DSS` option is now 18 bytes, which with the tcpcrypt `MAC` option will make 40 bytes of tcp options. The other possible way to shrink this is to use 4 byte sequence numbers rather than the 8 byte ones. This would reduce the `DSS` option to 14 bytes for the `DSN` to subflow sequence number mapping and to 8 bytes in the case of `Data ACKs`.

The `DSS` option will be sent in the clear i.e. not encrypted by tcpcrypt. The `MAC` option must cover the `DSS` option. This implies that we need to add the `DSS` option to the `MAC` data structure.

2.2.4 Backward compatibility

There will be the following 5 types of node:

- MPTCP nodes: supports MPTCP as defined in [28] but does not support tcpcrypt
- tcpcrypt nodes: supports tcpcrypt but does not support MPTCP
- SMTCP capable nodes: support MPTCP and tcpcrypt and the use of tcpcrypt to secure MPTCP
- legacy nodes: support neither tcpcrypt nor MPTCP
- MPTCP/tcpcrypt nodes: supports both tcpcrypt and MPTCP but does not support the use of tcpcrypt to protect MPTCP

The expected behaviour is as following:

(i) SMTCP contacts an SMTCP node, SMTCP should be used

(ii) SMTCP contacts an MPTCP node, MPTCP should be used

(iii) SMTCP contacts a tcpcrypt node, tcpcrypt should be used

(iv) SMTCP contacts an MPTCP/tcpcrypt node, MPTCP and tcpcrypt are used in a non integrated fashion

(v) SMTCP contacts a legacy node, TCP should be used.

In order to achieve, we use the following approach. In the `SYN` of the initial 3-way handshake, both the `CRYPT/Hello` option (3 bytes) plus the `MP_CAPABLE` option including the initiator's key (12 bytes) should be sent. This supports ii), iii) and v) i.e. the receiver can discard either of the two options or both of them resulting in each of the mentioned cases.

In order to support case i) (and to distinguish it from case iv), we need to signal it in an explicit way. The easiest way is to use one of the flags `C` to `H` in the `MP_CAPABLE` message. Let's assume it is the `C(rypt)` flag. If the `C` flag is set and the `CRYPT/Hello` option is present, this means SMTCP (i.e. use tcpcrypt to protect MPTCP signaling and data).

2.2.5 Remaining Issues

One main challenge in order to use tcpcrypt to secure MPTCP is the limited TCP option space. There is little room for TCP options and this approach would consume most of it, which would prevent the use of other options like SACK. In order to deal with this issue, we are proposing a general framework to extend the TCP option space that would solve the particular problem of carrying cryptographic material for securing TCP streams, as well as the much wider problem of limited TCP option space. This is developed further in section 2.5.

A second issue to consider is how this would work with TCP Segmentation Offload (TSO). TSO allows end-hosts to offload the relatively computationally expensive task of segmenting the TCP stream to dedicated hardware on the NIC. This is important to maintain performance with the increasing line speeds available. Currently MPTCP is compatible with TSO and it is important that SMTCP is also compatible.

2.3 Integrating MPTCP With TLS

Transport Layer Security (TLS) [26] is an application layer protocol that allows to encrypt and authenticate the data exchanged between applications running on different hosts. Various documents have analysed the security of the TLS protocol from different viewpoints [60]. Over the years, various extensions to TLS have been developed and deployed. Besides attacks on the cryptographic algorithms or their implementations, TLS is also vulnerable to some forms of attacks that affect the underlying TCP protocol [54].

TCP, by default, does not include any cryptographic technique to authenticate/encrypt data. Three types of solutions have been proposed to improve the security of TCP. A first approach is to tune the TCP stack to prevent some packet injection attacks. Examples of this approach may be found in [29] or [56]. Another approach is to add authentication to the TCP protocol. The TCP MD5 option defined in [35] was the first

example of such an approach. The TCP-AO option defined in [63] and [46] is a more recent example. These two solutions were designed to protect long-lived TCP connections such as BGP sessions from packet injection attacks. They assume that a secret is shared among the communicating hosts. A third approach is to extend TCP to include the cryptographic techniques directly inside the TCP stack [7].

Multipath TCP [28] allows several interfaces (or paths) to transmit the packets that belong to a single connection. It achieves this by managing several TCP connections (called subflows) for each Multipath TCP session. With Multipath TCP, the number of subflows associated with a given session may change dynamically. This ability to adapt to changes in the underlying TCP subflows can also be used to improve the reaction to various packet injection attacks. This means Multipath TCP can cope with various types of attacks and errors that affect the TCP stack and cannot easily be recovered at the application layer without implementing a session layer protocol.

In this section, we propose a high level design for Multipath TLS (MPTLS). Additional details may be found in [8]. Multipath TLS integrates Multipath TCP and TLS together to provide enhanced security for the applications. This integration can be beneficial for security sensitive applications that already rely on TLS. A similar approach could be defined for other application layer protocols such as SSH. It could also address the requirements of the TCP extension being developed within the TCPINC working group.

2.3.1 Attacks Considered

Any security protocol must be designed to protect against a given set of attacks. For this work, we consider three different types of attackers:

- a completely off-path attacker that cannot capture any of the packets exchanged by the communicating hosts but is able to inject spoofed packets. We call this attacker the off-path attacker.
- an attacker that sits on (at least one of) the paths used to exchange packets between the communicating hosts at the beginning of the connection but later moves away from this path. We assume that this attacker is able to capture the packets exchanged and send spoofed packets but cannot modify the packets sent by the communicating hosts. We call this attack the partially on-path attacker.
- an attacker that is always on the path between the communicating hosts. This attacker is able to capture the packets exchanged by the communicating hosts and modify them. We call this attacker the on-path attacker. This attacker could also be a middlebox that sits on one of the paths used by the communicating hosts.

The off-path attacker is the simplest type of attacker in our taxonomy. This attacker can inject spoofed packets inside existing TCP connections. To inject a packet so that it is accepted inside an existing TCP connection, the attacker needs to guess the IP addresses of the communicating hosts, the port numbers (one is usually well-known) and sequence numbers and acknowledgements that fit inside the receive window. Several techniques have been defined [30],[45], [56], [5] to cope with such attacks and some have been implemented [31]. The

off-path attacker can try to inject either packets containing data or control packets (i.e. packets carrying the `RST` or the `FIN` flags). For both control and data packet injection attacks, a successful attack results in the termination of the affected TCP connection.

Multipath TCP is by design less vulnerable to such attacks than regular TCP since it uses 64 bit data sequence numbers. It should be noted that the utilisation of TLS on a TCP connection does not improve its reaction against this form of attack. If the underlying TCP connection is reset due to an off-path attack, the TLS session is reset as well. The standard solution to cope with these off-path attacks is to authenticate the packets that are exchanged at either the TCP or the IP layer. The TCP-MD5 option, defined in [35], allows authentication of the packets exchanged by the communicating hosts. This prevents the packet injection attacks discussed above since the receiving host can verify the validity of all received packets and easily reject those sent by the attacker. The recently proposed TCP-AO option [63] generalises this technique by allowing different types of hash functions and supporting key rollover techniques. Unfortunately, both TCP-MD5 and TCP-AO suffer from an important drawback. They assume that the communicating hosts have a shared secret. This shared secret is required because both techniques aim at authenticating all packets including the initial packets of the three-way handshake. While TCP-MD5 and TCP-AO can be used to secure the long-lived TCP connections used by BGP sessions between Internet routers, they cannot be easily used to secure regular Internet traffic. Furthermore, combining TCP-AO with Multipath TCP would consume a large fraction of the TCP option space in the packets and would prevent the use of other important TCP extensions such as TCP-SACK. For these reasons, TCP-AO [63] is not a suitable solution.

The second type of attacker that we consider is the on-path attacker. This is the most powerful attacker. TCP by itself is not protected against such attackers that can modify the packets exchanged on a TCP connection. Some of the deployed middleboxes operate like on-path attackers since they modify the contents of TCP packets. Typical examples are NAT devices that change IP addresses and port numbers. Application Level Gateways running on NATs and TCP normalisers that re-segment TCP packets [41] are other examples. Furthermore, studies have also shown that some middleboxes may generate packets to terminate TCP connections for various reasons [27]. We need to distinguish two types of attacks from these on-path attackers of middleboxes:

- attacks that modify the packet payload without terminating the connection
- attacks where the middlebox terminates the affected TCP connection

The first type of attack is transparent for TCP. TCP does not detect the attack since the checksum has been modified by the attacker and delivers the modified payload. Multipath TCP, thanks to its DSS checksum, can detect a payload modification performed by a middlebox that understand TCP but not Multipath TCP. In this case, it falls back to regular TCP to preserve the connectivity between the communicating hosts. It should be noted that it would be possible to design a middlebox that modifies the payload of Multipath TCP packets in a way that cannot be detected by Multipath TCP (e.g. if the middlebox updates the DSS checksum after

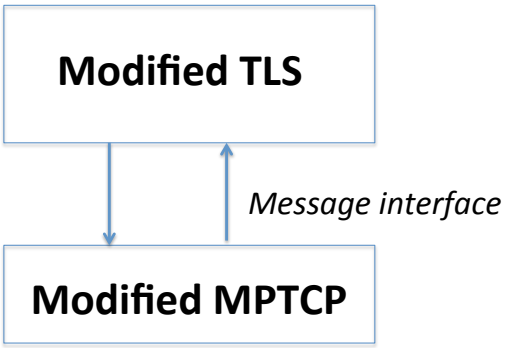


Figure 2.1: MPTLS architecture

having modified the payload). If the middlebox decides to close the connection by using regular TCP RST or FIN flags, then Multipath TCP can react by establishing a new subflow (possibly via another path) to preserve the connection in spite of the attack. The ability to manage several subflows is an important technique that allows Multipath TCP to react to attacks. While this reaction is effective against currently deployed TCP middleboxes, it is possible to design Multipath TCP aware middleboxes that inject specific Multipath TCP packets (e.g. by using the FAST_CLOSE option whose security relies on the keys exchanged during the initial handshake) to actively terminate Multipath TCP connections.

On the other hand, TLS uses cryptographic techniques to secure keys that allow authentication and encryption of the records exchanged over the (Multipath) TCP connection. With appropriate cryptographic techniques and a Public Key INfrastructure that prevents MITM attacks, it is possible to negotiate keys through an on-path attacker without enabling this attacker to derive the security keys. These security keys can then be used to authenticate and encrypt the records that are exchanged. Unfortunately, while the current TLS specification and implementations verify the authenticity of the received records from the derived secret keys, they react to an authentication failure by releasing the underlying TCP connection and alerting the application. This implies that an on-path attack will result in a denial of service attack for TLS applications.

2.3.2 High-level architecture of MPTLS

At a high level, MPTLS integrates TLS and Multipath TCP as shown in figure 2.1. The application interacts with the TLS implementation through the existing API without any change. This ensures backward compatibility with existing applications.

MPTLS modifies the TLS sublayer and keep in this layer the following functions that are already part TLS :

- secure handshake and key negotiation
- transmission and reception of the TLS records
- encryption/decryption of the TLS records

The TLS techniques for authenticating TLS records are moved to the modified Multipath TCP sublayer. For this, we leverage the recently proposed encrypt-then-mac technique [33]. This is motivated by two

reasons. First, Multipath TCP already verifies the received data by using its optional DSS checksum [28]. We replace this checksum with a cryptographic MAC authentication that has strong security properties. Second, if Multipath TCP receives a TLS record with an invalid MAC it can simply discard the data and wait for its retransmission or perhaps terminate the affected TCP subflow and create a new one to retransmit the data. This is much better from a security viewpoint than the current approach used by TLS that terminates the TLS session as soon as a record with an invalid MAC has been received.

To enable Multipath TCP to correctly compute the MAC of each TLS record, we modify the interface between TLS and Multipath TCP. While regular TCP provides a bytestream service to TLS, our modified Multipath TCP provides a message mode service. With this service, Multipath TCP transports a sequence of TLS records. All these records are delivered in sequence (possibly after some retransmissions by the underlying layer) to the TLS sublayer at the receiver. Each TLS record is sent as a single message by Multipath TCP. More precisely, the following workflow is used:

- the application generates a block of up to 2^{14} bytes of plain text
- the TLS layer encrypts the plain text and adds the TLS record header
- the TLS layer passes the TLS record to Multipath TCP as a message
- Multipath TCP maps the entire TLS record via one DSS option onto a single subflow, computes the MAC and adds the DSN option
- the corresponding packets are reliably transported through the network and delivered to the Multipath TCP layer at the destination
- Once all packets have been correctly received at the destination, the MAC is verified. If the verification succeeds, the TLS record (without the authenticated MAC) is passed to the TLS layer that extracts the record header and decrypts the ciphertext to retrieve the plaintext and pass it to the application.

As explained earlier, TLS negotiates the keys that are used to encrypt and authenticate the TLS records. Multipath TCP also needs keys to authenticate the establishment of subflows. Instead of exchanging the Multipath TCP keys in clear as defined in [28], we leverage the technique proposed in [58] that allows to derive additional keys from the MasterSecret negotiated during the secure TLS key exchange. The required keys are then passed to Multipath TCP as proposed in [52] to secure the establishment of new subflows and also authenticate the transported TLS records. A drawback of this approach is that the keys required to authenticate the establishment of subflows are only available at the end of the TLS key exchange. Thus, this key exchange can only be performed over the initial subflows.

2.3.3 Modifications to Multipath TCP

The high level solution described above requires some modifications to Multipath TCP to support the integration of TLS with Multipath TCP:

- Multipath TCP must be modified to support a message-mode service (limited to messages of up to 2^{16} bytes) instead of the default bytestream service
- Multipath TCP must be able to utilize the keys generated by TLS to authenticate the messages through a MAC algorithm and the establishment of new subflows
- optionally an improved API to enable a better exchange of information between TLS and Multipath TCP

[28] uses the three way handshake to negotiate the utilisation of Multipath TCP through the utilisation of the `MP_CAPABLE` option and also to exchange the keys that are used to both identify the Multipath TCP connection and authenticate the additional subflows. Since in MPTLS the keys will be provided by TLS, there is no need to exchange keys during the three way handshake. However, the three-way handshake is also used to exchange the tokens that identify the Multipath TCP connection on each host and the initial data sequence numbers in both directions. To identify the Multipath TCP connection, we place the Sender's token in the MP_CAPABLE option of the SYN and SYN+ACK segments. This variant of the MP_CAPABLE option is shorter than the MP_CAPABLE option defined in [28]. The two could be distinguished by relying on the length of the MP_CAPABLE option or based on the version number or one of the bits of the MP_CAPABLE option.

The second important modification to Multipath TCP is to replace its bytestream service by a message-mode service that is targeted to the needs of the modified TLS sublayer. To distribute the data over different subflows, Multipath TCP relies on the Data Sequence Signal option [28].

This option allows to map Length bytes from the bytestream of the Multipath TCP connection starting at data sequence number 'DSN' to the subflow sequence number 'SSN'. Since the length field is encoded as a two bytes long unsigned integer, it can be used to map up to 2^{16} bytes. This is larger than the maximum size of the TLS plaintext encoded inside a record (2^{14} bytes) [26]. Even with expansion, a TLS ciphertext will never become larger than 2^{16} bytes. We cover each TLS record with a single mapping. This implies that a single TLS record can only be mapped onto one subflow. If the TLS session is interactive, then short TLS records will be used anyway. If the TLS session transports a large amount of data, sending entire records over each subflow should not impact the performance. We note that some deployments of TLS already dynamically adapt the length of the TLS records to the activity of the session [32].

The original DSS option defined in [28] maps data from the bytestream to the subflow sequence numbers. Its main fields are :

- The Data Sequence Number of the first mapped byte in the bytestream
- The Subflow Sequence Number
- The Length of the mapping

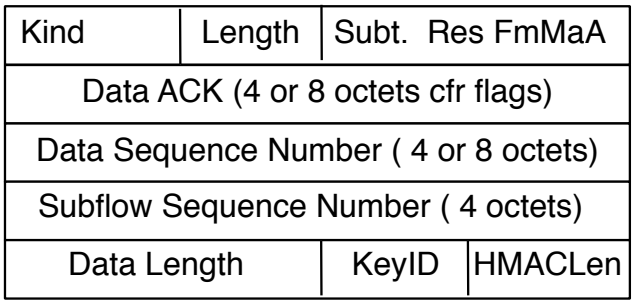


Figure 2.2: Modified MPTCP DSS option

- The checksum that is computed over the data and a pseudo-header to detect middlebox interference

This option can also contain an optional Data acknowledgement and a few flags that indicate whether 32 or 64 bits Data sequence/acknowledgement numbers are used [28].

We modify the semantics of the DSS option as follows. First, the Data-level length becomes the size of the TLS record that is transported. Given the constraints imposed by [26], this record will always be shorter than 2^{16} bytes. Second, the DSS checksum is disabled and replaced by a MAC.

TLS supports a range of authentication techniques that can be negotiated during the TLS handshake. In this first version of the document, we assume that TLS has negotiated a keyed HMAC to authenticate the TLS record. Subsequent versions of this document will analyse other record authentication methods such as [49]. There are several possible design options to transport the computed HMAC.

At a high level, our objective is to transport the computed HMAC together with the mapped data. As explained earlier, this data will be transported on the same TCP subflow given that the TLS record is covered by one DSS mapping. Since the TLS keys can change during the lifetime of a TLS sessions, MPTLS needs to indicate which key has been used to compute an HMAC. This problem can be solved by associating a key identifier to the keys that are passed by TLS to Multipath TCP and place this key identifier inside the DSS option. Figure 2.2 provides the structure of the DSS option used by MPTLS.

The semantics of the last two fields of this modified DSS option are:

- 'MAC Length' is an 8 bits unsigned integer that specifies the length of the HMAC that follows the DSS option, i.e. the HMAC starts at subflow sequence number 'SSN' and ends at 'SSN+MACLength-1'. A length of 0 indicates that no HMAC has been computed for the TLS record.

- 'Key Identifier' is an 8 bits unsigned integer that indicates the key that was used to compute the attached HMAC. When a TLS session starts, the key identifier is set to zero. It is incremented by one after each renegotiation of the keys. Using key identifiers allows an MPTLS implementation to verify the validate of TLS records that have been generated by using different keys.

The last point to be discussed is the actual data that is covered by the HMAC. MPTLS uses the HMAC to authenticate both the TLS record, i.e. the payload mapped by the DSS, and the DSS option itself. This implies that any modification to one of the fields of the DSS option or to the data part would be detected

by the HMAC. It should be noted that in contrast with TCP-AO, MPTLS does not protect the TCP header fields. This choice is motivated by the fact that various middleboxes modify IP and TCP fields, notably the IP addresses, the TCP port numbers, the sequence and acknowledgement numbers. Including these fields in the MPTLS HMAC would prevent any communication through such middleboxes.

With MPTLS, data can only be acknowledged at the Data-Sequence level (i.e. through the Data ack field of the DSS option) once it has been received in sequence and the HMAC that authenticates the received TLS record has been validated. If the HMAC authentication fails for one received TLS record, then the corresponding subflow should be terminated with a reason code [9] that indicates an authentication failure. The TLS record will then be retransmitted over another available subflow or a new subflow will be established to transmit this record. This mode of operation allows Multipath TCP to cope with various types of packet injection attacks without breaking the connection and affecting the TLS layer or the application.

2.3.4 Required modifications to TLS

Multipath TCP can be completely transparent to the application since it provides the same socket interface as regular TCP. On the other hand, TLS is a record oriented protocol. Data is encoded in records that are reliably exchanged over the underlying TCP connection. TLS defines two types of records:

- the control records that are used to exchange control information such as the secure handshake messages that negotiate the cryptographic parameters and keys
- the data records that transport encrypted and authenticated data

These two types of records have a variable length and are encoded by using a TLV format specified in [26]. The TLS protocol defines several types of data records depending on the type of encryption scheme that is used. Current TLS implementations apply a MAC-then-Encrypt approach to transmit data [26]. This implies that each data record, is first authenticated, e.g. by using a negotiated HMAC algorithm, then the authenticated record is encrypted. Cryptographers have argued about several security problems with this approach and there are ongoing discussions about the use of Encrypt-then-MAC for the data record [33]. This technique has better security properties and so we build upon it to integrate TLS and Multipath TCP together.

At a high level, an MPTLS connection starts like a regular Multipath TCP connection. The Multipath TCP connection starts with a three-way handshake using the MP_CAPABLE option to negotiate the utilisation of Multipath TCP. Once the three-way handshake has finished, the bytestream is established and TLS can start its key negotiation. All the crypto mechanisms defined in [26] can be used to negotiate the crypto parameters and keys. In contrast with TCP-AO, this crypto negotiation is performed over an unprotected bytestream as when TLS is used over single-path TCP. In particular, no HMAC is included in the DSS option defined in the previous section.

TLS uses the client_write_MAC_key and server_write_MAC_key to authenticate the data records. We build upon the encrypt-then-mac principle [33] and place the encryption/decryption function in the TLS layer and

the authentication function inside Multipath TCP. As in AEAD algorithms [49], we assume that two different keys are used for the encryption and the authentication. The encryption keys are generated and stored in the TLS layer. The authentication keys are generated in the TLS layer by using the PRF described in [26] or through the procedure defined in [58].
[33] defines a TLS record as being composed of:

```
struct {
          ContentType type;
          ProtocolVersion version;
          uint16 length;
          GenericBlockCipher fragment;
          opaque MAC;
        } TLSCiphertext;
```

Although the length field of this record is encoded as a 16 bits integer, TLS limits the record size to 2^{14} bytes at most. Since MPTLS performs the authentication outside of the TLS record, we need to remove the opaque MAC from the structure of the TLS record. With regular TCP, this record format enables the receiver to retrieve the record boundaries and extract it from the bytestream. In MPTLS, this feature is not required since it provides a message mode service that delivers entire TLS records. For this reason, we also remove the length information from the TLS record. Note that this implies that it will be impossible for MPTLS to fallback to regular TCP if middlebox interference is detected. The modified TLS record becomes:

```
struct {
          ContentType type;
          ProtocolVersion version;
          GenericBlockCipher fragment;
        } MPTCP_TLSCiphertext;
```

MPTLS needs to derive more keys than when TLS is used over regular TCP. Once the TLS handshake has succeeded, the crypto parameters and keys are known by the two communicating hosts. In TLS [26], six keys are derived from the Master key. We leverage [58] to derive two additional keys:

- client_write_MPTCP_key[SecurityParameters.mac_key_length]
- server_write_MPTCP_key[SecurityParameters.mac_key_length]

The *write_MAC_key and *_write_MPTCP_key keys are derived by TLS and immediately passed to the Multipath TCP sublayer by using a similar technique as [52]. The *write_MAC_key keys are used to authenticate the TLS records while the *_write_MPTCP_key keys are used to authenticate the establishment of subflows.

2.4 Testing TCP Receivers

Many of the tools for monitoring and controlling resource liquidity rely on TCP-like feedback mechanisms to pass information about the state of the resources back to the sender. The potential problem with this is that it relies on the honesty and accuracy of the receiver for sending this information. In many cases the tussle in the network means that the receiver has a self-interest in mis-declaring this. For instance by under-declaring congestion or by sending acknowledgements before data has actually arrived a receiver may hope to receive a higher throughput by subverting the congestion control algorithm at the sender.

When any network resource (e.g. a link) becomes congested, the congestion control protocol [2] within TCP/IP expects all receivers to correctly feed back congestion information and it expects each sender to respond by backing off its rate in response to this information. This relies on the voluntary compliance of all senders and all receivers. Over recent years the Internet has become increasingly adversarial. Self-interested or malicious parties may produce non-compliant protocol implementations if it is to their advantage, or to the disadvantage of their chosen victims. Enforcing congestion control when trust can not be taken for granted is extremely hard within the current Internet architecture.

Simple attacks have been published showing that TCP receivers can manipulate feedback to fool TCP senders into massively exceeding the compliant rate [59]. Such receivers might want to make senders unwittingly launch a denial of service attack on other flows sharing part of the path between them [62]. But a more likely motivation is simple self-interest—a receiver can improve its own download speed with the sender acting as an unwitting accomplice. Savage [59] quotes results that show this attack can reduce the time taken to download an HTTP file over a real network by half, even with a relatively cautious optimistic acknowledgemnt strategy.

Senders do not have to be motivated solely by "the common good" to deploy these changes. It is directly in their own interest for senders serving multiple receivers (e.g. large file servers and certain file-sharing peers) to detect non-compliant receivers. A large server relies in part on network congestion feedback to efficiently apportion its own resources between receivers. If such a large server devotes an excessive fraction of its own resources to non-compliant receivers, it may well hit its own resource limits and have to starve other half-connections even if their network path has spare capacity.

The proposed tests do not require the receiver to have deployed any new or optional protocol features, as any misbehaving receiver could simply circumvent the test by claiming it did not support the optional feature. Instead, the sender emulates network re-ordering, network loss and ECN Congestion Experienced (`CE`) marks to test that the receiver reacts as it should according to the TCP protocol.

Currently the sender has no means to verify that a receiver is correctly sending this feedback according to the protocol. A receiver that is non-compliant has the potential to disrupt a sender's resource allocation, increasing its transmission rate on that connection which in turn could adversely affect the network itself. This section describes a two stage test process that can be used to identify whether a TCP receiver is non-compliant [51]. The tests enshrine the principle that one shouldn't attribute to malice that which may be

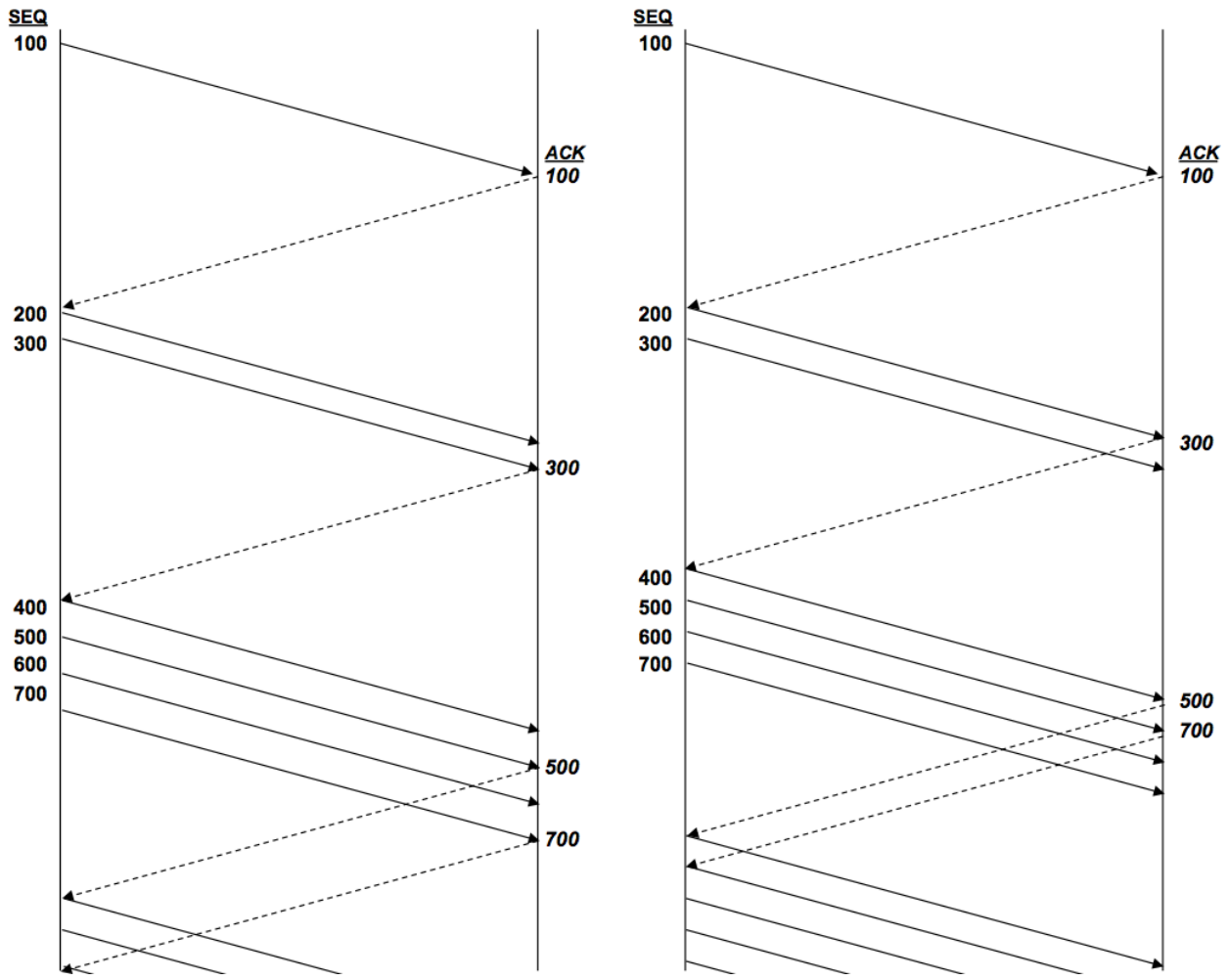


Figure 2.3: The flow on the left acknowledges data when it is received. The flow on the right uses optimistic acknowledgement to reduce the apparent RTT.

accidental. The first test causes minimum impact to the receiver but raises a suspicion of non-compliance. The second test can then be used to verify that the receiver is non-compliant. Finally there is a discussion of possible approaches for testing the accuracy and timeliness of ECN feedback. Importantly this specification does not modify the core TCP protocol - the tests can either be implemented as a test suite or as a stand-alone test through a simple modification to the sender implementation.

2.4.1 Requirements for a Robust Solution

Since the above problems come about through the inherent behaviour of the TCP protocol, there is no gain in introducing a new protocol as misbehaving receivers can claim to only support the old protocol. The best approach is to provide a mechanism within the existing protocol to test whether a receiver is compliant. The following requirements should be met by any such test in TCP and are likely to be applicable for similar tests in other transport protocols:

(i) The compliance test must not adversely affect the existing congestion control and avoidance algorithms since one of the primary aims of any compliance test is to reinforce the integrity of congestion control.

(ii) Any test should utilise existing features of the TCP protocol. If it can be implemented without altering the existing protocol then implementation and deployment are easier.

(iii) The receiver should not play an active role in the process. It is much more secure to have a check for compliance that only requires the receiver to behave as it should anyway.

(iv) It should not require the use of any negotiable TCP options. Since the use of such options is by definition optional, any misbehaving receiver could just choose not to use the appropriate option.

(v) If this is a periodic test, the receiver must not be aware that it is being tested for compliance. If a misbehaving receiver can tell that it is being tested (by identifying the pattern of testing) it can choose to respond compliantly only while it is being tested. If the test is always performed this clearly doesn't apply.

(vi) If the sender actively sanctions any non-compliance it identifies, it should be certain of the receiver's non-compliance before taking action against it. Any false positives might lead to inefficient use of network resources and could damage end-user confidence in the network.

(vii) The testing should not significantly reduce the performance of an innocent receiver.

2.4.2 Overview of the Tests

The precise details of the tests can be found in the relevant Internet Draft [51] which also includes discussion of potential negative impacts of the test (for instance on TCP's RTT estimator). The following subsections provide a summary of the tests.

2.4.2.1 The Probabilistic Test

The first requirement for a sender is to decide when to test a receiver. The protocol document doesn't specify when the test should be performed but does give some guidance. Like Sherwood's skipped segment solution [62], the proposed solution depends on the strict requirement that all TCP receivers have to send a duplicate acknowledgement as soon as they receive an out-of-order segment. This acknowledges that some data has been received, however the acknowledgement is for the last in order segment that was received (hence duplicating an acknowledgment already made). SACK extends this behaviour to allow the sender to infer exactly which segments are missing. This leads to a simple statement: if a receiver is behaving compliantly it must respond to an out-of-order packet by generating a duplicate acknowledgement.

Following from the above statement, a sender can test the compliance of a given receiver by simply delaying transmission of a segment by several places. A compliant receiver will respond to this by generating a number of duplicate acknowledgements. The sender would strongly suspect a receiver of non-compliance if it received no duplicate acknowledgements as a result of the test. A misbehaving receiver can only conceal its actions by waiting until the delayed segment arrives and then generating an appropriate stream of duplicate acknowledgements to appear to be honest. This removes any benefits it may be gaining from cheating because it will significantly increase the RTT observed by the sender.

The actual mechanism for conducting the test is extremely simple. Having decided to conduct a test the sender selects a segment, N. It then chooses a displacement, D (in segments) for this segment where strictly $2 < D < K - 2$ (where K is the current window size). In practice only low values of D should be chosen to conceal the test among the background reordering and limit the chance of masking congestion. Ideally D should be 6 or less. D has to be greater than 2 to allow for the standard fast retransmit threshold of 3 duplicate acknowledgements. If K is less than 5, the sender should arguably not perform any compliance testing. This is because when the window is so small then non-compliance is not such a significant issue. The exception to this might be when this test is being used for testing new implementations.

To conduct the probabilistic test the sender transmits segments $N + 1, N + 2$, etc. instead of segment N. Once it has transmitted $N + D$ it can transmit segment N. The sender needs to record the sequence number, N as well as the displacement, D.

According to data reported by Piratla [53], as many as 15% of segments in the Internet arrive out of order though this claim may not be accurate. Whatever the actual degree of re-ordering, receivers always expect occasional losses of packets which they cannot distinguish from re-ordering without waiting for the re-ordered packet to arrive. Consequently a misbehaving receiver is unsure how to react to any out-of-order packets it receives. It should be noted that the natural reordering may reduce the displacement deliberately introduced by the test so the sender should conduct the test more than once. If a receiver appears to fail this test then the deterministic test should be used instead.

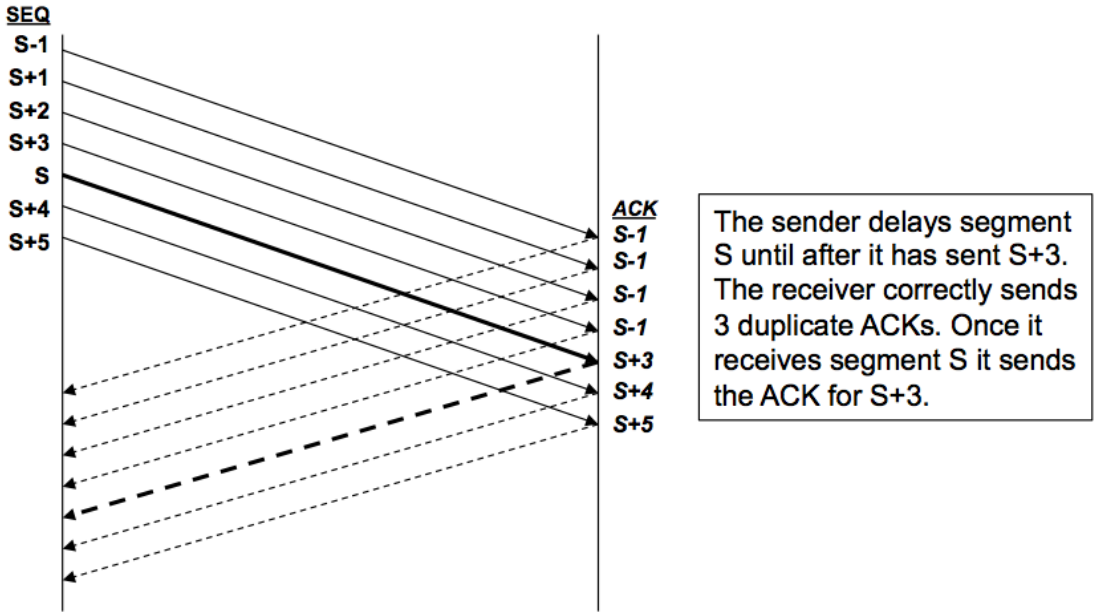


Figure 2.4: A compliant receiver reacting to the probabilistic test

2.4.2.2 The Deterministic Test

In order to perform the deterministic test the sender again needs to choose a segment, M to use for testing. This time the sender holds back the segment until the receiver indicates that it is missing. Once the receiver sends a duplicate acknowledgement for segment $M-1$ then the sender transmits segment M. In the meantime data transmission should proceed as usual. If SACK is not in use, this test clearly increases the delay in reporting of genuine segment losses by up to a RTT. This is because it is only once segment M reaches the receiver that it will be able to acknowledge the later loss. Therefore, unless SACK is in use, the sender has to pessimistically perform a congestion response following the arrival of 3 duplicate acknowledgements for segment $M-1$ as mandated in [2].

If the sender sees an acknowledgement for any segment greater than $M-1$ then it is clear that the receiver is sending inaccurate feedback. The exact response of a sender in this case is not specified, but typical responses might include terminating the connection or adding the receivers IP address to a blacklist.

2.4.2.3 Testing Correct ECN Feedback

Identifying whether a receiver is suppressing ECN `CE` marks is slightly harder. ECN [57] mandates that receipt of a `CE` (congestion experienced) mark should toggle the receiver into sending `ECE` (echo congestion experienced) flag in the TCP header until it sees a `CWR` (congestion window reduced) TCP header flag from the sender. This means that setting a `CE` mark on a packet risks preventing further `CE` marks from being reported for a full RTT. One way to reduce this risk is to start setting `CWR` flags pre-emptively a delay of a few segments.

Accurate ECN (Forward Ref) allow for accurate feedback of `CE` marks. Thus for connections that are known to have negotiated use of one of these you can simply test a connection by setting a `CE` mark on a packet and

checking to see if the cumulative ACK covering that packet echoes the mark. At that point the sender should set CWR but should not congestion respond.

2.5 Inner Space: Extensibility of Transport Control

2.5.1 Transport Extensibility: The Problem

The transport layer has become hard to extend, for two main reasons:

(i) Many middleboxes only forward the two original transport protocols (TCP and UDP) and sometimes only the one dominant application layer protocol (HTTP). To make matters worse, they often only forward TCP headers that conform to the stereotype they expect;

(ii) The TCP option space was limited to 40 B when TCP was first defined [54].

In 2011, Honda *et al* [36] documented TCP extensibility tests on a broad but small set of paths (in the first Trilogy project) and found that there were few if any middlebox traversal problems over residential access networks, but the chance of a new option traversing other types of access was terrible.

To better visualise the problem, we have prepared Figure 2.5 from the dataset made publicly available by the authors of [36]. Too much faith should not be placed in the precise percentages, given the number of paths tested (shown in parentheses) was small. Nonetheless, the figure shows cellular was especially bad (stripping options on 40% of paths for port 80 and 20% for other ports), but WiFi hotspots, enterprise, and university networks were nearly as bad (typically, about 18% of paths blocked new extensions). Also, the top chart shows that some paths block traffic to unassigned port numbers, which are meant to be used for new protocols as they are developed. Shamefully, Universities, not cellular, are the worst culprit for this behaviour.

The purpose of the Trilogy 2 project is to make liquid resource control feasible. This is why we are extending the capabilities of the Internet's transport protocols. And it is why we are espousing network functions virtualisation (NFV) technology, to make it easy to scale and flex network functions (primarily middleboxes) using virtualisation technology. And it is why we are developing technologies to communicate market information between customer end-systems and the infrastructure resources they need (compute, bandwidth and storage).

However, none of this will be deployable if we cannot clear the logjam that is effectively making any extension to communication protocols infeasible. We need to be able to extend both end-to-end and end-to-middle communications channels.

Inner Space overcomes this by embedding additional control and option data within the data portion of packets that *look* like TCP packets to any middlebox. Inner Space removes the limitations of option length making it much easier to deploy mechanisms such as tcpcrypt. It also opens up opportunities to deploy entirely new transport protocols and to ensure they will pass middleboxes unhindered. Inner Space also enables end-to-middle communications—a potentially important aspect for the liquid net where negotiating with middleboxes may become a key part of increasing resource liquidity.

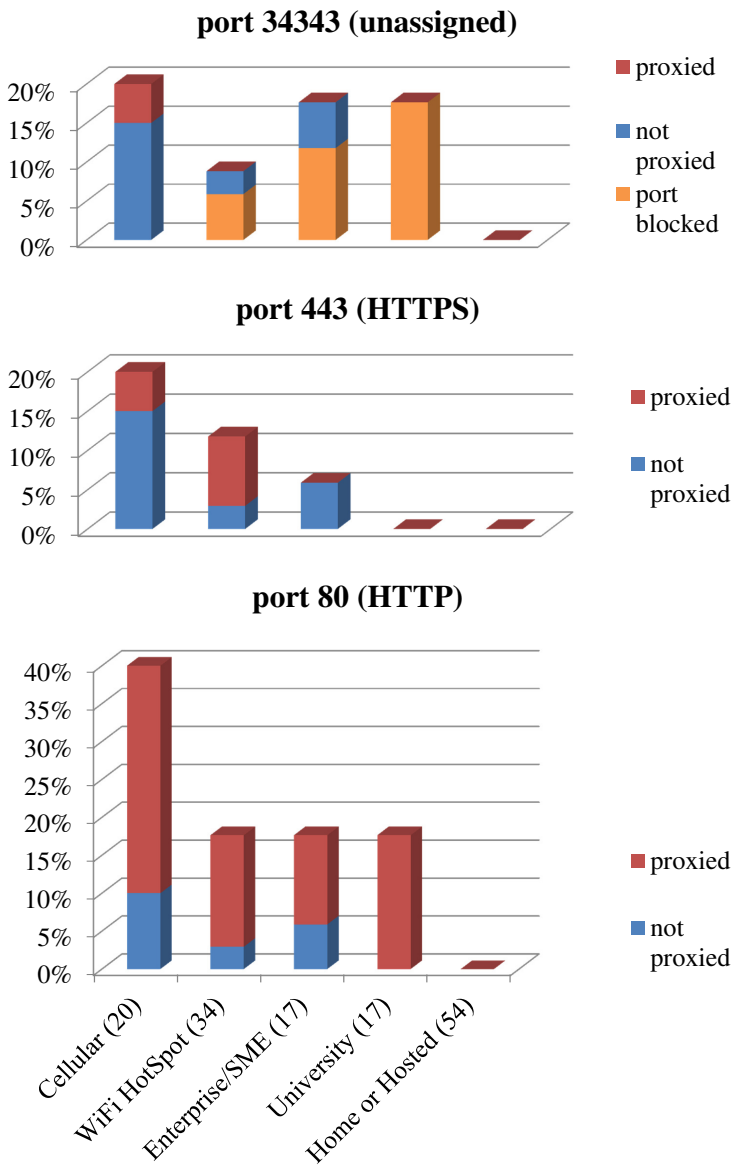


Figure 2.5: Prevalence of an Unknown TCP Option being Stripped from the Initial Segment (nn): No. of paths tested to ea. port

The original Trilogy work on bandwidth resource pooling using MPTCP had to go to extraordinary lengths to traverse middleboxes, prompting paper titles such as “Is it still possible to extend TCP?” [36] then “How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP” [55].

The latter paper metaphorically raised an inscription at the entrance to the field of transport protocol extension, carved with the words “Abandon hope, all ye who enter here”. Up to that point, hope was still held that commands to control MPTCP could be deployed within the TCP Data, thus circumventing middlebox interference. However, Raiciu *et al* [55] proved that one critical and prevalent MPTCP command (the Data ACK) would have to be placed as a TCP option outside the TCP Data, otherwise flow-control deadlock could ensue. This single command was enough to make the whole of MPTCP susceptible to the levels of option stripping seen in Figure 2.5, whether the other MPTCP commands were hidden away or not.

The Inner Space protocol (see 2.5.4) *should* solve this deadlock problem. However, where middlebox traversal is concerned, there will always be surprises—it is essential to test a large number of paths.

Whereas middlebox traversal is a general problem for transport extensibility, our second problem is specific solely to TCP: exhaustion of the space assigned for new control options. Nonetheless, the only packets that can reliably traverse all middleboxes have to look exactly like TCP. Therefore, we use a TCP-look-alike protocol, which leads us to have to solve the shortage of TCP option space.

Since MPTCP was developed, other extensions to TCP have been proposed, such as TCP Fast Open (to remove the round of handshaking latency from resumed connections) and candidate protocols (tcpcrypt and the TLS option) to provide privacy from large government agencies by using opportunistic encryption. Most of the recent extensions need relatively large amounts of space to exchange cryptographic material (cookies, keys, etc.).

These, when combined with existing commonly used options and/or with MPTCP, already exhaust the available 40 B of option space in TCP. Indeed, tcpcrypt alone exhausts the available space and currently makes special arrangements to place initial key exchange control options where TCP normally locates the payload.

The two words ‘Inner Space’ are appropriate as a name for the solution; ‘Inner’ because it encapsulates (tunnels) options within the TCP Data and ‘Space’ because the space for TCP options within the TCP Data becomes virtually unlimited–constrained only by the maximum segment size.

2.5.2 Inner Space: Status

The full specification of the Inner Space protocol is publicly available from the Trilogy Web site [**?**], but not yet published. For this deliverable it has been conveniently integrated into one document, but it is unlikely to be published as a single Internet Draft, because some of the modules are of more general utility. It is structured in modular sections and it is planned to submit them to the IETF as a number of smaller drafts, each potentially with a different set of co-authors.

A cut-down mode of the protocol (the ‘Default’ mode) was submitted to the IETF [13] and presented at the Nov’14 meeting. It was presented twice in different contexts:

(i) In a special session of the TCP maintenance and minor modifications (tcpm) WG as a candidate solution to the problem of TCP option space exhaustion, with useful middlebox traversal and latency features [12];

(ii) In the TCP increased security (tcpinc) WG, as a way to make opportunistic encryption deployable without being susceptible to downgrade attacks by middleboxes, and without adding latency [14].

The aspect of Inner Space that solves the flow-control deadlock problem had not been fully written up at that time. It is now fully specified as a mode of the Inner Space protocol, which we will call 'TCPbis' [**?**].

The Inner Space idea has appeared rather late in the Trilogy 2 project. However progress has been fairly rapid. The initial idea focused solely on extending option space on the opening segments of a TCP connection. It appeared on the first day of the summer IETF (21-Jul-14). By the November IETF meeting, it had developed through 5 published drafts.

Since then, about a dozen engineers have collected together to develop these ideas. Half a dozen plan to work on middlebox traversal testing (some is already in progress and most is planned for Q1 of 2015). Two or three expect to implement it in Linux, including upstreaming.

Most are currently working within two EU FP7 projects (Trilogy 2 and RITE). EU-funded work on the aspects of the protocol that cut latency will be reported via the RITE project. Work on the base protocol will be reported through Trilogy 2. One participant from NetApp is expecting to work on it in the EU CYCLOPS project, and some in the RITE project are hoping to continue the work in the EU NEAT project when RITE finishes (Nov'15). At present only one participant is based outside the EU—the originator of the Minion protocol who now works for Google.

2.5.3 Inner Space: Implications

2.5.3.1 End-to-Middle Communications Channel

Once endpoints have an end-to-end control channel within the TCP Data, they can use authentication or even encryption to stop middleboxes interfering with it. Then given middleboxes already interfere with Outer TCP Options, these Outer Options in the main header can serve a new purpose as a channel for end-system TCP stacks to interact with middleboxes, but only if they choose to.

The Echo Cookie TCP option is a speculative idea being explored to migrate connection state from one set of middleboxes to another if the path changes, but without any explicit coordination between the middleboxes on the two paths (this is currently only a speculative idea). The Echo Cookie option was introduced in conjunction with the Inner Space protocol but has already been split out for separate publication [18] because it is more generically useful. The initial idea is that one host (say A), or a middlebox, would encode its connection state in a cookie and send it in an Outer TCP Option to the other host B. B would automatically echo the cookie back to A. On the way, any middlebox(es) on the new path could be populated with the old path's connection state. If the endpoint was migrating, it could co-ordinate migration of connection state with

migration of application-layer state using a similar parallel cookie mechanism at the application layer. This might become a standardised facility, but a proprietary scheme could also be arranged in a private network.

2.5.3.2 Deployable New Transport Services

The TCPbis mode of Inner Space [**?**] adds the ability to include messages within TCP's datastream that can be delivered out of order. That is, they can be delivered immediately they arrive (rather than in the order sent, which leads to head-of-line blocking whenever a message is delayed or lost). Out-of-order delivery is the key to solving the flow-control deadlock problem.

This is not a new idea; it builds on TCP Minion [38]. However critically, unlike Minion, it includes the control channels that can be used to negotiate new behaviours, which makes it incrementally deployable for existing applications and transports.

The ability to choose whether messages are delivered in order transforms the Inner Space protocol into a container for any transport service. On the wire it appears identical to TCP, so it should traverse the large majority of middleboxes. But it can adopt the intended behaviour of other transport protocols, by tunnelling control messages within the TCP Data.

Table 2.1 lays out the 3×3 matrix of possible transport services that can be provided to applications once messages can be delivered with a choice of ordering. The payload is the 'message' in the vertical dimension, and control options are the 'message' in the horizontal.

Payload	Control Options		
	Ordered	Unordered	Both
Ordered	Default	(TCP)	TCPbis
Unordered		(UDP)	UDPbis
Both		(SCTP)	TCP2

Table 2.1: Possible Transport Service Modes Enabled by a Choice of Message Ordering

Each cell of the matrix contains the transport protocol behaviour that the Inner Space Protocol could emulate, while still traversing middleboxes as well as vanilla TCP. The Default behaviour is that provided by the simplest variant of Inner Space, as specified in the previously published draft [13]. The three main existing transport protocols (TCP, UDP and SCTP) appear in the middle column, but they are enclosed in parentheses because it would probably be pointless to implement solely unordered control options, given TCP's default semantics make ordered options available naturally.

The interesting modes are those in the right-most column, which have been given the interim names 'TCPbis', 'UDPbis' and 'TCP2'. TCPbis is the mode specified in [**?**] and outlined here. Traditionally TCP control options have been delivered out of order; outside the sequence space of the TCP Data. The 'bis' signifies that Inner Space adds a choice of ordered control options to TCP And, of course, it adds middlebox traversal and no arbitrary limit on the space for new options. Similarly UDPbis adds a reliable, ordered control channel to UDP.

The cell labelled TCP2 probably has the most potential. It has not (yet) been specified, but the foundations

for it are already in place in the TCPbis mode [**?**]. A separate document to specify the TCP2 mode is planned. It has been deliberately named TCP2 as a parody of HTTP2, because it provides the same transport services (see below).

In TCP2 mode, a single TCP connection could deliver data in multiple independent streams without the overhead of multiple connections. If one stream is waiting for a loss to be recovered, this would still allow other streams to make progress, this avoiding head-of-line blocking. This is the service that Stream Control Transport Protocol (SCTP) is designed to provide, but it cannot be deployed over the public Internet because many middleboxes only forward packets to ports that had well-known uses when the middlebox was implemented. Web applications really need a transport that provides multiplexed stream service within one connection (which is why they open multiple TCP connections). In the absence of any prospect of SCTP or an equivalent service becoming deployable at the transport layer, HTTP has been extended to provide multiplexed streaming at the application layer (HTTP2 [6]).

Nearly all the additions in HTTP2 are transport capabilities that would ideally have been provided by transport layer extension, had the transport layer not been logjammed by middleboxes. The Inner Space protocol actually adds these new transport capabilities in effectively the same place as HTTP2 (within the TCP Data), but it structures them separately from the application layer protocol, so that they are not exclusive to HTTP.

Inner Space is also structured so that data transformations such as compression or encryption can easily be introduced and controlled by TCP options (see later), as a generic facility available to any application layer protocol. Initiated with the help of Trilogy 2 participants, the IETF's TrAnsPort Services (TAPS) working group is now chartered [37] to map out all the transport services required by applications. Inner Space could provide a protocol for applications to negotiate which of these transport services each end wants and understands.

2.5.4 Inner Space: Approach

Vanilla TCP is generally acceptable to all middleboxes, therefore the Inner Space protocol uses vanilla TCP as its wire protocol. However, Inner Space introduces its own extensibility mechanism, given i) Outer TCP Options are so often interfered with by middleboxes; and ii) the maximum space assigned to them is so small. The Inner Space approach will be described using:

- Two motivating subsections:
  - explaining the root-cause of these two problems, which clarifies why other attempts to solve the problem have not worked;
  - outlining the strategy for dealing with middlebox interference once and for all.
- Followed by two subsections outlining the technology:
  - how the Inner Space protocol tunnels control channels within the TCP Data;

– and how it bootstraps these inner channels without altering the outer TCP header at all, so there is no risk of such an alteration being 'normalised'.

2.5.4.1 Root-Cause of the TCP Extensibility Logjam

In the TCP header there has always been a 4-bit field (the Data Offset) that defines where the header ends and the payload starts, in 4-octet words. The maximum size of the header is therefore $(2^4 - 1) \times 4\,\mathrm{B} = 60\,\mathrm{B}$ (which includes 20 B for the base header). Therefore, there are only 40 B available for options.

Numerous attempts have been made to redefine the size or the scale of the Data Offset field. However, they all risk delivering corrupt data to the application. This is because middleboxes will always exist that use the original unmodified meaning of the Data Offset, and therefore assume everything beyond this point is the payload. If the sender has placed control options beyond this point, the middlebox will treat them as data. Many such middleboxes resegment the TCP datastream (using their incorrect understanding of the Data Offset). They will therefore shift any control options that appear to be data into different positions within each segment, resulting in corruption of user-data.

Other common behaviours can lead to similar corruption. A middlebox can 'normalise' the TCP header into the stereotype it considers a TCP header should conform to. Then any newly defined options or flags that redefine or scale the Data Offset would be stripped, shifting the header/payload boundary backwards, but leaving any control options unmoved—stranded in what now appears to be the payload.

Even if the two end systems test for all possible middlebox behaviours before using extended control space, the path can change, e.g. due to mobility events, introducing corruption part-way through a connection. Also, it is hard to test for resegmentation because it can occur only occasionally and non-deterministically.

2.5.4.2 Middleboxes: Dominate then Cooperate

The strategic aim is eventually to cooperate with middleboxes, but only once the protocol has established its new channel for e2e control which it can protect from any middlebox interference using message authentication and/or opportunistic encryption (both of which need only ephemeral keys and do not require any identity or key management).

It would not be feasible to protect the existing control channel (Outer TCP Options) with message authentication or encryption, because too many current middleboxes would still alter or strip the Outer Options. This would invalidate authentication, requiring the receiver to have to ignore or discard a significant proportion of segments.

Thus if the endpoints introduced Outer Option authentication today, they would just cause their own working service to fail. Whereas, if endpoints authenticated the new inner channel from day one, if any middlebox tried to tamper with the new channel, the middlebox would have caused the working service to fail. No operator would deploy such a middlebox.

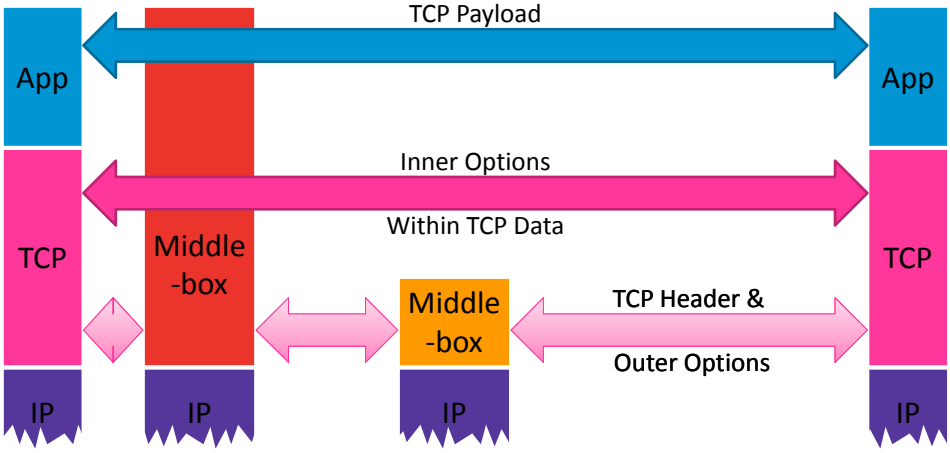


Figure 2.6: Inner Space Encapsulation Model

2.5.4.3 Tunnelling Control Channels within TCP Data

The Inner Space protocol ensures new TCP capabilities can traverse most middleboxes by tunnelling TCP options within the TCP Data as 'Inner Options' (Figure 2.6).

The extensibility facility provided by traditional TCP options (now termed 'Outer Options') is still available, but it is assumed that the end-to-end path might be split into a number of hops by middleboxes (Figure 2.6) and Outer Options can only be guaranteed to survive over one such hop. Therefore, these Outer Options are set aside for communication with middleboxes, because they can be left visible (or encrypted using a key shared with the middlebox) even if the Inner Options are encrypted end-to-end.

The problem of traversal of Outer Options from one segment to the next then transforms into a problem that the middlebox industry has to deal with, i.e. the externality (failure to take account of extensibility) is internalised (turned into a problem for those who are causing the problem).

The Inner Space protocol primarily addresses middleboxes that intervene at the transport layer (the one on the right in Figure 2.6), but it also includes an optional facility to ensure traversal of middleboxes that intervene right up to the application layer (the one on the left), e.g. Web filters and similar forms of deep packet inspection (DPI). For details of the latter, see [**?**, Appendix C.3].

The Inner Space protocol is merely a framing protocol; deliberately very limited in scope. It solely:

- identifies where each frame starts
- provides frame size information
- defines the semantics of different types of frame

The first two are defined by fields of the protocol itself, while the last (the type of each frame) is defined by the mode the connection is in (Default, TCPbis, TCP2, etc.). The semantics of the various frames used in each mode are defined in the specification of that mode. The connection mode is part of a connection's state, that can be set by an Inner TCP Option (called the ModeSwitch Option).

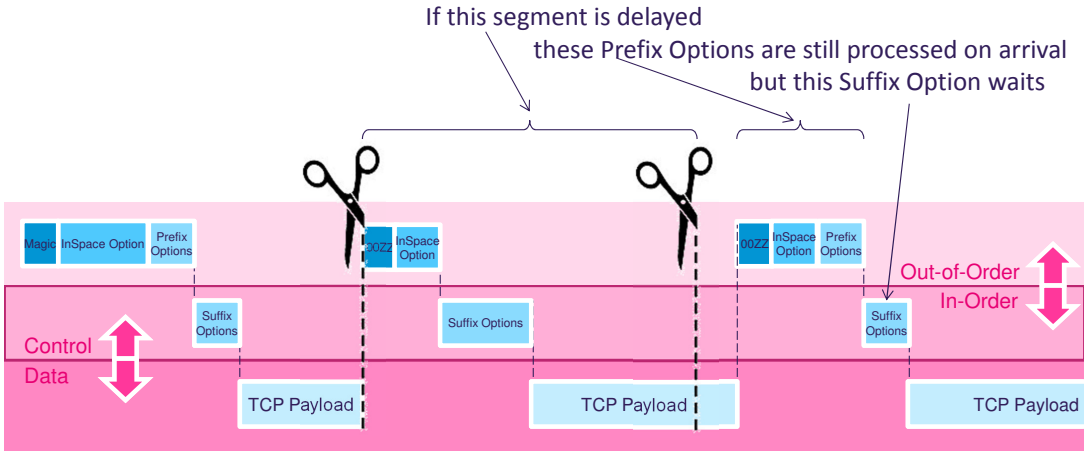


Figure 2.7: Control Channels within a 'TCPbis' mode Datastream

With just this minimal framing information, the TCP receiver can reconstruct the Inner Options sent by the sender, even if a middlebox resegments the datastream and even if it strips 'Outer Options' from the TCP header because it does not recognise them.

Figure 2.7 represents the three different types of channel of the TCPbis mode, by visually spreading the single data stream over three layers, from top to bottom:

- An out-of-order control channel (fire-and-forget TCP options);
- An in-order control channel (flow-controlled TCP options);
- An in-order data channel (the TCP datastream).

In reality, of course, all three layers are transmitted as one contiguous datastream. The control channel headers are added by the sending TCP/Inner-Space stack and processed then removed by the receiving TCP/Inner-Space stack.

In the out-of-order control channel, the darker blue headers represent markers that indicate where the sender started each sent segment (labelled '00ZZ'). Then, if the stream suffers resegmentation (represented by the scissors in Figure 2.7), the receiver can still find the original segment boundaries. The marking mechanism will be outlined in § 2.5.5.

In the 'Default' mode (which is not shown in the Figure) only in-order control options are needed, so segment markers are not needed. This is because each segment states its own size, so the receiver can work along the stream calculating where the next segment starts one segment at a time.

If out-of-order delivery is needed (e.g. for TCPbis mode), the protocol is still confined solely to framing, but each frame header consists of a little more than just frame sizes (see § 2.5.5). This is why the Default mode (with only in-order delivery) has been submitted and presented to the IETF first [13].

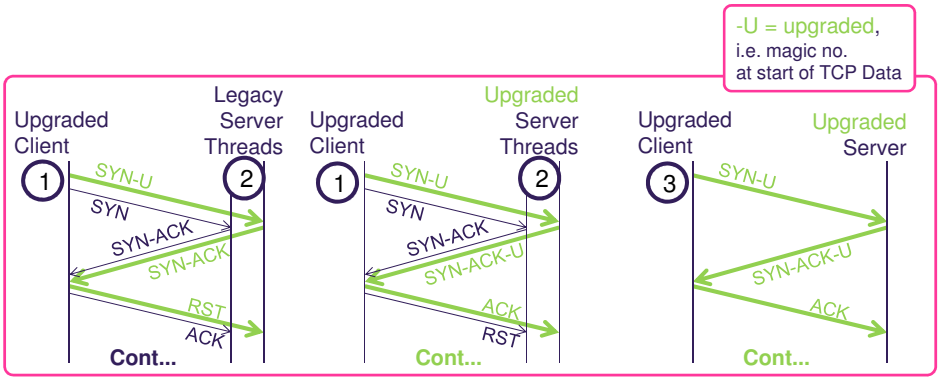


Figure 2.8: Dual Handshake and Migration to Single Handshake

2.5.4.4 Bootstrapping Inner Options without Outer Options

At the very start of the datastream the first marker in either direction is special, as illustrated by the 'Magic' label in Figure 2.7. The presence of a standardised magic number on an initial SYN or SYN/ACK tells the receiving TCP/Inner-Space stack that this connection comes from a sender that understands the Inner Space protocol. If the magic number is not present, the TCP/Inner-Space stack just responds with a normal TCP segment.

This approach is not completely safe, because arbitrary payload data might just happen to start with the standardised magic number (see § 2.5.8). Nonetheless, it allows the two end-systems to negotiate support for Inner Space using a TCP header that is completely indistinguishable from a vanilla TCP header. This avoids the need for some new option or flag in the main TCP header, which would be vulnerable to stripping by a 'normalising' middlebox, which in turn would lead the receiver to deliver corrupt user-data to the application.

Although a TCP stack that has been upgraded to understand the Inner Space protocol will recognise the magic number, one that has not will not. If a TCP client knows from previous experience that a server supports Inner Space, it can simply start using the Inner Space magic number on the very first SYN.

In the unlikely but possible event that the server at that address no longer supports Inner Space, it will respond without a magic number. Fortunately, a TCP server does not deliver payload to the application until the handshake has completed, so the client can abort the handshake with a RST and start another with an 'Ordinary' SYN segment.

If a TCP client has no prior knowledge of whether a particular TCP server supports Inner Space or not, it can avoid any latency either way by using dual handshakes in parallel, as illustrated in Figure 2.8. One indicates support for Inner Space while the other doesn't. If the server responds to the former with a magic number, the client aborts the latter handshake. If the server responds to the former without a magic number, the client aborts the former.

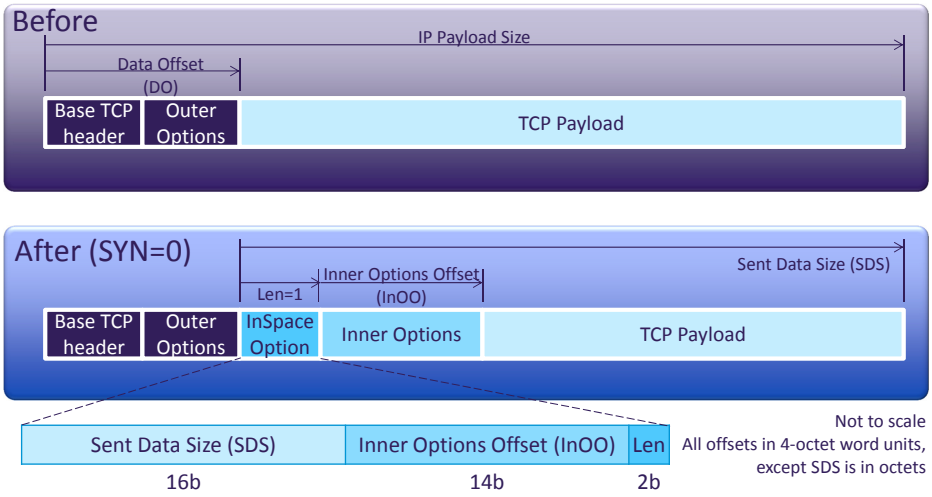


Figure 2.9: Inner Space TCP segment structure (Default Mode, SYN=0)

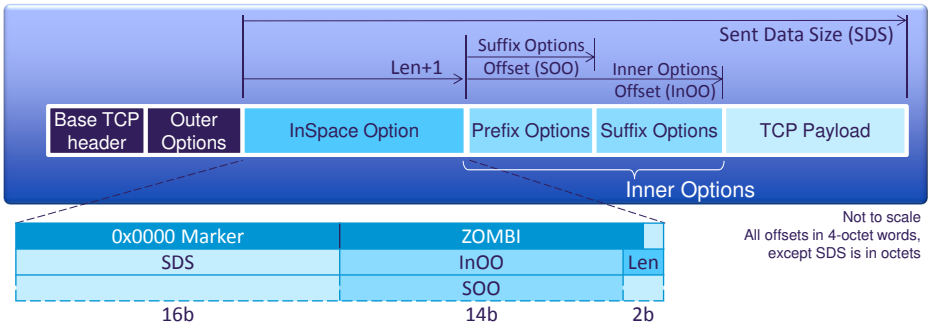


Figure 2.10: Choice of In-Order and Out-of-Order TCP Control Options (TCPbis Mode, SYN=0)

2.5.5 Inner Space Protocol: Format and Semantics

Figure 2.9 shows the structure of a Default Mode Inner Space segment in comparison to an ordinary (non-Inner-Space) TCP segment, shown at the top of the figure. The segment starts with an 'InSpace Option' that is merely a simple framing header to indicate the size of i) itself; ii) the space for Inner TCP Options; and iii) the TCP Data as it was when sent. This simple framing header adds an incredible amount of extensibility to TCP, merely by creating a robust spacious control channel.

Nonetheless for full extensibility, as we have already explained, certain ACK-related control options have to be delivered out of order; to avoid flow-control deadlock. This requires 'TCPbis' mode to be turned on.

Figure 2.10 illustrates the structure of a TCPbis segment. The spaces for the two control channels ('out-of-order' and 'in-order') that were illustrated in Figure 2.7 are visible within the TCP Data as 'Prefix Options' and 'Suffix Options'. This requires one more field in the InSpace option (the Suffix Options Offset or SOO) to specify the size of the second control channel.

The 'Prefix Options' and 'Suffix Options' fields are merely spaces that can contain existing or new TCP options (or they may be empty). TCPbis mode defines the order in which the receiver processes any TCP

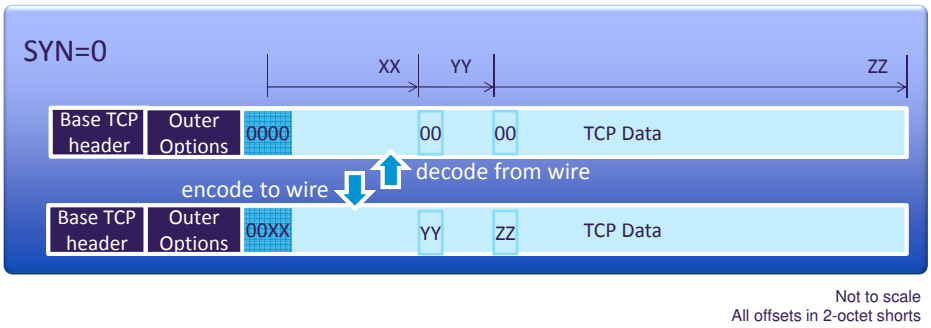


Figure 2.11: Zero Overhead Message Boundary Insertion (ZOMBI) Encoding

options within these fields as follows:

(i) Prefix Options: on arrival (out-of-order);

(ii) Outer Options: on arrival (out-of-order);

(iii) Suffix Options: processed with the datastream (in-order).

The TCPbis mode spec. [**?**] includes details of other additional steps in the processing order, in order to allow for possible transformations on the TCP Data, e.g. encryption or compression. Even though no current TCP options transform the data, one is in the process of definition (tcpinc), and http2 requires encryption and compression. The processing order has to be specified carefully because Inner TCP Options might control these transformations (e.g. a re-key command), but the Inner Options might themselves be transformed because they are within the TCP Data.

As has already been explained, processing Inner Options in order is straightforward. However, processing them out of order requires a marker so that the receiver can find the start of each segment even after a gap and resegmentation. The field labelled 'Marker' in Figure 2.10 serves this function. It has to be aligned on a 4-octet word boundary relative to the start of the stream, which is straightforward (see [**?**] for details). The marker is always 16 bits filled with zeros.

However, 16 zeros are only useful to mark the start of segment if there are no other occurrences of 16 zeros aligned on a 4-octet boundary within the segment. In TCPbis mode, the sender ensures this is the case by using a simple encoding to replace any other zeros, as illustrated in Figure 2.11. It starts by placing a pair of zero octets straight after the marker (the field labelled 'ZOMBI' in Figure 2.10). Then it replaces them and any other pairs of zero octets with the offset (in 2-octet units) to the next pair of zero octets, or the end of the segment if there are no more. This guarantees there will be no other zeros, because an offset can never itself be zero.

We have called this the 'ZOMBI' encoding, standing for Zero Overhead Message Boundary Insertion. Its overall overhead is 2 B, not zero, but the name means there is zero additional overhead once the message boundary has been inserted. The encoding is also very fast to decode at the receiver.

The ZOMBI encoding is an improvement over the Consistent Overhead Byte Stuffing encoding (COBS [24]) used in Minion. COBS uses a single zero byte as the marker, but there is not always enough space for an offset in a single byte. Therefore, occasionally COBS has to insert a few extra stepping-stone bytes (using another special value) to reach the next occurrence of zero. Because its output is generally a little larger than its input, COBS has to write an intermediate copy of the segment to memory. In contrast, ZOMBI is a zero-copy encoding, because it never expands the input (zero-overhead).

To guarantee the zero-overhead property, ZOMBI relies on the size of the marker being large enough for the maximum offset needed. In the case of TCP segments this is a little less than 2^16, because that is the maximum size of an IP packet (v4 or v6). For jumbograms, ZOMBI simply uses a larger, 4 B, marker (see [**?**, Appendix C.2]).

The first segment in each direction does not need a marker and it does not need to be ZOMBI encoded, because TCP always has to deliver it before subsequent segments can be transmitted.

2.5.6 Inner Space Protocol: Tricky Bits

2.5.6.1 Sequence Space and Flow-Control Coverage

Once out-of-order messages are included within the TCP datastream, decisions have to be made as to whether such messages should be included in TCP's sequence space, and in TCP's flow-control calculations. Ideally they would not be. However, compromises have had to be made because otherwise certain middleboxes would intervene.

Out-of-order TCP Options (fire-and-forget) are included in the TCP sequence space, but the sender immediately removes them from its retransmit buffer, considering them to have been acknowledged implicitly. The receiver acknowledges the sequence space consumed by fire-and-forget options, but not unless it is also acknowledging other sequence space.

Fire-and-forget options are exempted from flow-control calculations at both ends of each half-connection. However, it is likely to be necessary for the two ends to conspire to inflate the window value they communicate in the main TCP header, again in order to keep certain middleboxes happy.

The specifics are not discussed further in this overview, but can be referred to in the TCPbis mode specification [**?**].

2.5.6.2 Segments with No Payload

One of the fundamental rules in the design of TCP is that acknowledgements (ACKs) are never ack'd if they contain no data themselves (so-called Pure ACKs). Inner Space introduces the possibility that segments that carry no payload might still carry fire-and-forget control options that consume sequence space (which we term 'Impure ACKs'). This raises the prospect of a never-ending sequence of ACKs ack'ing ACKs (an ACK storm).

The Inner Space specification includes a simple rule to avoid ACK storms: the receiver "MUST NOT consume further sequence space solely to acknowledge impure ACKs". Nonetheless, the receiver is allowed

to acknowledge fire-and-forget sequence space with a pure ACK under certain conditions, just to keep a middlebox quiet.

2.5.6.3 When to Stop Digging a Hole

At this point one might start to become concerned that a fairly straightforward and principled initial design will become increasingly compromised by the need to play along with all the weird behaviours that middleboxes might choose to adopt. Nonetheless, the rules it has been necessary to adopt so far have been fairly straightforward (although admittedly pragmatic rather than principled).

However, only implementation experience will determine how significant these compromises are. Altho the new rules have been easy to say, it is likely not to be straightforward to implement them—they will involve introducing exceptions to some very fundamental assumptions in the original design of TCP. Such compromises in turn might make it harder to introduce further fundamental changes to TCP in future.

2.5.7 Inner Space: Interaction with Existing & Proposed Internet Mechanisms

2.5.7.1 Interaction with Existing End-Systems

With the introduction of out-of-order delivery to the Inner Space protocol, it can now carry all existing and proposed TCP options within the TCP Data. Thus, in theory, Inner Space can shelter any existing or new TCP option from middlebox interference, at least as deep as the transport layer and possibly deeper.

Inner Space also saves each TCP option from consuming a handshaking round to check whether the other end and the path support a particular TCP option. Thus, many options including newer ones like MPTCP or tcpinc can save a round-trip (or more) of latency [14]. Inner Space is also compatible with the new TCP Fast Open option, that removes the TCP's handshake latency when a client opens a connection to a server that it has recently connected to.

This could make MPTCP applicable to a range of new applications where resilience can be provided by duplication or coding over multiple paths for short flows (mice), whereas at the moment MPTCP is only really applicable for elephants.

In a similar vein, few if any users are likely to enable opportunistic encryption on all their traffic if it adds a round trip to the handshake of all their short flows (e.g. Web). Therefore by removing this handshake latency, Inner Space can be considered to make increased TCP security (tcpinc) feasible.

The TCPbis mode spec [**?**] gives specific details of the compatibility between Inner Space and known TCP options, as well as extensive discussion on ensuring the socket API remains backward compatible.

2.5.7.2 Interaction with Existing Middleboxes

In general, it is hoped and expected that Inner Space will traverse most middleboxes. However, some hurriedly configured firewalls are known to block data in the initial SYN segment, categorising it as the signature of an attack. Data-in-SYN should actually only be considered as *part* of the signature of a flooding attack if correlated with other signs. It is hoped that such sloppy categorisation will be updated over the coming months as TCP Fast Open (TFO) is deployed (the first widespread use of data-in-SYN for a couple of

decades). However, it can be predicted that many of these poorly configured firewalls will never be reconfigured. Therefore it will be necessary to develop another way to bootstrap Inner Space, probably by relaxing the latency requirement; a connection that takes a while to work is still better than one that fails quickly.
During TFO deployment, a bug in the Linux netfilter conntrack module was identified that black-holes all attempts to send a SYN segment for a flow if the first attempt contained data but was lost. The conntrack module is generally used when Linux is in forwarding mode (i.e. in a network device). Therefore, although the bug has been fixed, but it will persist in many middleboxes. Therefore, the fall-back strategy mentioned above will have to work-round this bug as well.

2.5.8 Inner Space: Benefits and Drawbacks

The benefits of Inner Space can be summarised as:

- (expected) minimal middlebox traversal problems;
- incremental deployment from legacy servers;
- zero handshaking delay;
- a choice of in-order and out-of-order delivery;
- no arbitrary limit on available space for control options.

The drawbacks fall into four categories:

- Architectural compromises (see "When to Stop Digging a Hole" in § 2.5.6 above);
- Newly introduced security concerns.
- Overheads;
- Non-determinism (the statistical nature of the magic number approach);

The Security Considerations section of the Inner Space TCPbis mode spec. [**?**] discusses three potential new security concerns, all of which have been addressed by alterations to the design. Of course more concerns may arise later.
The TCPbis mode spec [**?**] includes a section giving formulae for all the various overheads that Inner Space will introduce. The formulae are dependent on variables representing factors unknown at this stage, such as the proportion of connections that will use the Inner Space protocol. A set of example values for all the variables is proposed in that section, in order to give a feel for the likely size of the various overheads.
In summary overheads are caused by two main factors: i) the additional bytes and processing for the In-Space framing header; and ii) the additional connections whenever the dual handshake is used. The former consumes a very low percentage of extra bandwidth (probably $< 0.1\%$) but processing overhead will be unknown until an implementation is built and tested. The dual handshake is however more demanding: it might

lead to a 5–10% increase in connection rates and perhaps a 1–5% increase in connection state on servers. Nonetheless the dual handshake is likely to cause negligible increase in network traffic (sub-1%) and it adds zero latency. [**?**] should be referred to for details.

The magic number approach introduces a probability of at worst 1 in 2^{-66} that payload data in an ordinary connection will be mistaken for a signal that Inner-Space is supported. This is equivalent to roughly 1 connection collision somewhere in the world every 40 years. It should be noted though that it is currently very unusual for there to be any data in the initial segment in either direction. If there is none, then the probability of a collision is zero. And the risk of a collision the other way round is zero, i.e. an Inner Space connection will never be confused for an ordinary connection.

3 Operator Control

Unless operators feel able to exert a degree of control, there is a real risk that the liquid network will be unable to reach its full potential. Equally it is important that such control should not in fact prevent the adoption of the liquid net. Currently the Internet is locked in a perpetual arms race with users wanting to retain control over their traffic and operators wanting to protect their network from unfair use and external threats. A full discussion of this tussle is presented in Deliverable D2.3. This chapter explores the tools that Trilogy 2 have developed to enable operator's to exert such control.

One key to maximising resource liquidity in the network is the ability to migrate flows onto paths that offer them the greatest utility. Currently flows are routed based on shortest path metrics which may not equate to the lowest latency or most predictable bandwidth. FUBAR is a system designed to address this.

In order to make informed decisions about the state of the network both for scheduling, routing and security it's important to know the state of resources throughout the network. Congestion Policing and Congestion Exposure (ConEx) was originally introduced as part of Trilogy 1 (supporting the architectural principle of Information Exposure). Here we present the tools that control and enforce resource liquidity within a pool. They make use of congestion as a market metric; either local congestion information as an initial deployment step or distributed information gathered by congestion feedback and re-feedback (ConEx). We also provide an update on the standardisation status of ConEx in the IETF.

Finally the ultimate goal of the liquid network is full liquidity across all resource types. An important aspect to that will be the ability to trade resources globally. The Federated Market is a tool that enables compute resources to be traded like any other market commodity thus creating both the demand and supply for compute liquidity.

3.1 Flow Utility Based Routing

In this section we present an ongoing effort to produce a system that reduces congestion and maximizes the utility of the entire network by installing new routes and changing the traffic load on existing ones. Flow Utility Based Routing (FUBAR) [**?**] works offline to periodically adjust the distribution of traffic on paths. It requires neither changes to end hosts nor precise prior knowledge of the traffic matrix.

3.1.1 Motivation

What should be the role of the routing system in a large ISP or enterprise network? Traditionally, it was all about ensuring connectivity, with cost metrics used as a crude way to spread load. Equal-cost multipath routing [23] added limited capability to load-balance, and MPLS-TE [25] provided additional ability to tweak traffic patterns to mitigate congestion. More recently, OpenFlow and Software Defined Networking [50] have provided a much more flexible data plane. But how should a routing system resolve applications' competing, qualitatively different demands?

Large networks with sufficiently many users tend to have relatively stable traffic matrices—demand between

each entry point and exit point does not vary very quickly. At the same time, the demand from each traffic aggregate is elastic, but not infinitely so. Streaming video, for example, can switch between different data rates, but once it reaches sufficient quality, bitrate becomes bounded. Even file transfer, which ought in principle to be infinitely elastic, is generally not in reality, due to access link capacity, receive window bounds and I/O limitations. In fact much web traffic is latency bounded, as it finishes while still in slowstart. The job of the routing system should be to satisfy all traffic aggregates, subject to there actually being enough capacity available in the network. But bandwidth isn't the only factor to be considered. Real-time flows have much firmer latency requirements—there's no point in giving arbitrary bandwidth to videoconferencing traffic if doing so involves routing it over a high-delay path. The routing system should instead try to maximize the utility of the traffic flowing over it, where utility depends on both delay and throughput. This is a very different problem than that solved by traditional routing protocols.

Based on information from switches, FUBAR measures the network's traffic matrix and determines how to route the aggregates that comprise that matrix so as to maximize utility for the network's users. Not all traffic is treated equally; real-time traffic must be routed over lower-delay paths, and even bulk aggregates are routed over the lowest-delay path if there is sufficient capacity. If there is not sufficient capacity, FUBAR splits an aggregate, progressively moving traffic onto less preferred paths until there is no persistent congestion in the network, or no further gains in utility can be achieved.

Characterizing the goal of routing in this way makes it into a difficult problem. The pieces of the puzzle are:

- How to predict the utility of an aggregate, given a traffic class and a certain amount of bandwidth and delay?
- How to predict how multiple aggregates will share a link?
- Given these, how to choose a set of possible paths for each aggregate, including paths that avoid congested hot-spots?
- Finally, how to split aggregates between multiple paths in such a way that overall utility is maximized?

If we can satisfy all of these goals, we will also achieve a network with no persistent congestion given sufficient provisioning, or one in which there are no severely congested hotspots if the network is under-provisioned.

To achieve these goals, we make a number of gross simplifications. We start with a very crude utility model and fill in the model's parameters based on measurements of how the traffic actually responds. We classify traffic with crude heuristics supplemented by operator knowledge when that is available. We also use a fairly simplistic model of how traffic aggregates share a congested link. We believe these rough approximations capture the essence of how real traffic behaves. Crucially, they allow us to reason about and choose between options for routing, and require little or no operator input. Current routing systems do not even take such

limited information into account; if they perform well, it is more due to luck and manual tuning by operators who rarely know the real demands of the traffic they are engineering.

Given these models, when the network is congested, we must generate paths that provide alternatives to the default low-delay paths. Then we must determine the optimal mapping of sub-aggregates to paths. This problem is NP-hard, but given the limitations of the models, there is little point in being too concerned with always finding an optimal solution. We care only about finding a good solution, and FUBAR performs sufficiently well in this respect to provide very substantial gains in utility over conventional shortest-path routing. Moreover, by alleviating congestion, FUBAR avoids building long queues in the network, even when operating at high network utilization.

3.1.2 System Design

The four components needed to implement FUBAR are:

- A way to measure the traffic matrix of a network.
- A method to approximate the utility of an individual flow.
- A method to estimate the bandwidth a flow will achieve given a defined path through the network in the presence of all the other flows in the network.
- A method to derive policy compliant paths for each flow, including the lowest delay path and others that avoid hotspots.
- An optimization algorithm to find the best way to split each aggregate of flows between the multiple policy compliant paths so as to maximize the overall utility of the network.

We will address each of these in turn.

3.1.2.1 Traffic Matrix

Traditionally, measuring the traffic matrix was difficult because routers did not keep sufficient counters. In an SDN network, the SDN controller is involved in setting up flows, so measuring the traffic matrix is much simpler.

FUBAR needs periodic per-aggregate bandwidth measurements and approximate flow counts for each aggregate. We believe that in real-world deployments FUBAR will be separate from the SDN controller and the measurements required will be taken hierarchically.

3.1.2.2 Utility

To capture how useful a flow is to the user application we borrow Shenker's concept of *utility* [61], but extend it to be a function of both bandwidth and delay. In FUBAR each flow is associated with a *utility function* which provides a mapping from bandwidth and delay to a single unitless real number in the range $[0 - 1]$. A utility close to 0 means that the flow does not accomplish any useful work. This may be because it currently does not have enough bandwidth or because the path that it takes through the network has too high a delay.

Conversely, a utility close to 1 means that providing additional bandwidth or reducing the flow's delay are not likely to render it any more useful to the application.

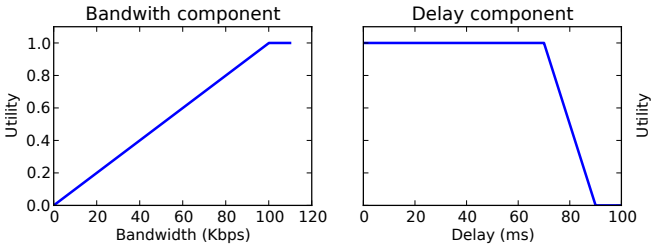


Figure 3.1: Real-time flow utility function

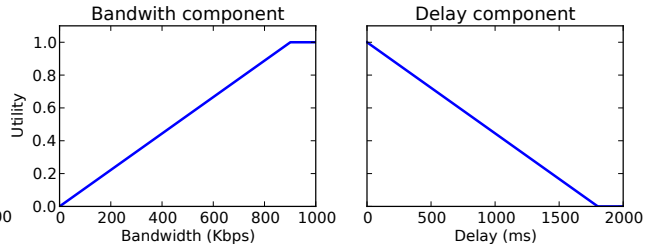


Figure 3.2: Bulk transfer flow utility function

The function itself defines a surface whose shape is very application dependent. Our utility metric consists of a bandwidth component and a delay component that are multiplied together to form the final utility. Examples of the two components can be seen in Figures 3.1 and 3.2.

For most applications a default delay curve can be used that slowly decays to zero as delay increases to a few seconds. Interactive applications tend to be more delay sensitive, so the operator can specify a non-default delay curve for flows to a certain port or from a particular server.

Our goal is to minimize the need for operator knowledge, so we rely on continuous traffic measurements to scale the bandwidth component as needed. We can infer the inflection point of the bandwidth curve when an aggregate is using an uncongested path and fails to utilize it. Our solution will also work with any non-linear increasing function for either bandwidth or delay, but for simplicity we chose those shapes because they are defined by the fewest points.

Figure 3.1 shows an example utility function of an interactive flow. At $0\,kbps$ the flow gets no bandwidth and is therefore not useful—its utility is 0. The utility grows as it gets more bandwidth and maxes out at $100\,kbps$, after which even if the flow is given more capacity the application is unable to make use of it. It can also be seen that this flow is delay sensitive—if the delay it experiences grows above $90\,ms$ the overall utility will drop to 0 (the two components are multiplied).

Figure 3.2 shows a bulk data transfer flow that can tolerate relatively large variations in delay, but requires more bandwidth. Conventional wisdom is that a single TCP flow with enough data to send is never satisfied—it will consume the entire capacity of the network and its bandwidth "peak" will be the bandwidth of the bottleneck link in its path. In real networks most TCP flows do not behave this way, especially in the backbone. The bandwidth demands of large transfers tend to be limited by the application itself (*e.g.,* the data rate of a compressed YouTube or Netflix video). Small transfers, on the other hand, are so small that they barely exit slow start. Even software downloads tend to be constrained in customers' access networks rather than in the backbone. When we look at multiplexed aggregates our preliminary measurements show that it is almost always possible to define an upper bound on the bandwidth requirement at any instant.

3.1.2.3 Traffic Model

For any particular allocation of flows to paths, we need to know how the flows will occupy the network so that we can estimate their bandwidth and hence their utility.

If there is at least one congested link we are faced with predicting how its capacity will be split among flows that cross it. Each congested link truncates the demands of flows that traverse it, so affects the distribution of flows on other congested links.

To estimate the traffic distribution we use a TCP-like traffic flow model. The model assumes a congested flow's throughput is inversely proportional to its RTT, and that congestion control algorithms in use are "compatible" in the sense that they co-exist reasonably gracefully with TCP traffic.

We imagine the network as a series of empty pipes. We fill them by having each flow grow at a rate inversely proportional to its RTT. A flow can stop growing either because it satisfies its demand (obtained from the peak of the bandwidth component of the utility function) or because there is no more room to grow because a link along its path has become congested. In practice we don't deal with individual flows, but with *bundles* of flows that share the same entry point, exit point, traffic class, and path through the network. The algorithm proceeds in steps, congesting a link or satisfying a bundle at each step until each bundle is either congested or has its demands met.

This flow model is simple enough to run quickly, yet sufficient to provide us with a back-of-the-envelope estimate of what bandwidth each flow can expect to get given a path assignment. We use this flow model as the building block of our optimization algorithm outlined in the next section.

3.1.2.4 Choosing Paths

We want to be able to take an aggregate of flows that share a source, destination and traffic class, and to split this into bundles of flows that are routed over different paths through the network. This will allow us to progressively offload an aggregate from its lowest delay path onto higher delay but less congested paths, so long as doing so increases utility.

The default path for an aggregate is easy to find—it's simply the lowest delay path, as if that is uncongested it will give the best utility. However, as that path becomes congested, we want to consider offloading part of the aggregate onto other paths. An optimal algorithm would need to consider all the possible policy-compliant paths between that source and destination. This is clearly computationally infeasible, so we need to wisely choose paths to add as alternatives.

Unfortunately, the choice of a good alternative path depends not only on the topology but also on which links are congested, as we want to offload traffic onto uncongested paths. This leads to a catch 22: we cannot choose paths without knowing congestion, but we can't know congestion levels without choosing paths for the traffic to use.

Our solution to this dilemma is to take an iterative approach. We start with only the lowest delay path in the path set for an aggregate and run the traffic model. This will predict where congestion will occur. If there is

no congestion, we're done.

If not, we add new paths to the path set for any aggregate that experiences congestion. The algorithm queries a path generator to find three alternative different policy-compliant paths not currently in the path set for that aggregate:

(i) A global path: the lowest delay path that avoids all congested links, regardless of whether they are currently used by this aggregate.

(ii) A local path: the lowest delay path that avoids all congested links that are being used by the congested aggregate.

(iii) A link-local path: the lowest delay path that simply avoids the most congested link used by the aggregate.

The global path, if one exists, is the alternative giving maximum additional incremental capacity as it is uncongested, but it may have high delay. The link-local path is the lowest delay alternative, but may already be congested. The local path offers a good middle ground. We tried different approaches and found this particular choice of three paths to be the best tradeoff between speed and solution quality.

The algorithm iterates, finding the best way to split each aggregate across the paths in the path set, as described in the next section, then adding more paths to the path set of any aggregates still congested, and so on, until either no improvement is found or all aggregates are uncongested. In our experiments we usually find that three or four iterations are sufficient, so we end up with approximately ten to fifteen paths in the path set for each aggregate.

3.1.3 Flow Allocation

Given a network topology and a set of aggregate flows, plus for each aggregate, a utility function and a path set, we've finally reached the heart of the problem: how to split each aggregate across paths so as to maximize overall utility. An optimal solution is NP-hard, but given all the other approximations we have made to get this far, optimality is not of great importance. What we want is simply a good solution.

For this we use a polynomial-time heuristic. The main intuition is that at first flows are allocated on low delay paths and are gradually moved to higher delay paths until congestion disappears. The algorithm proceeds in steps, increasing utility at each step until no progress can be made. This greedy approach guarantees termination, but it can converge to a local optimum. We will come back later to what we do when it does this.

Initially all flows for each aggregate are allocated to the shortest delay path. The traffic model is evaluated at this point and if there is no congestion the solution is optimal. In this case all flows have their bandwidth demand satisfied and are on the shortest policy-compliant delay path, which results in maximal attainable overall utility.

If there is congestion the algorithm will try to alleviate it by moving some flows of each aggregate away. It looks at congested links one at a time, starting with links where a change is likely to result in the best utility

gain—the ones that are the most oversubscribed.

Focusing on a single congested link, the algorithm goes over all aggregates crossing that link; for each it estimates the gain of moving some or all of the flows away. For a given flow path there are two main choices to be made—how many flows to evict and where to move the flows to.

The number of flows to move (N) depends on how large the aggregate is in terms of traffic volume. Small aggregates are moved in their entirety because they are unlikely to have a big impact on the final solution. For large aggregates there is a tradeoff between speed and utility—the more flows are moved at a time the faster the algorithm will converge, but the lower the overall utility of the solution will be.

Flows will have to be moved to an alternate path that excludes the congested link. As described in Section 3.1.2.4, the algorithm obtains three alternative paths.

At each step all three new alternatives are tested for each aggregate that crosses the congested link. When an aggregate is tested, N flows are removed from the original path and added to the alternative. The traffic model is evaluated to estimate the utility of the new solution. The best move is committed. At the end of the step, if no move improves utility, no progress can be made and the algorithm terminates.

Escaping local optima

The flow allocation problem is not convex and it is possible for the greedy algorithm above to get stuck in a local optimum. In this case no progress can be made, even though there is still congestion in the network. There is a simple tweak that can help us escape—when the algorithm gets stuck we can try to move larger and larger numbers of flows. In particular while we are in a local optimum we can tweak the number of flows to move (N) to progressively shift more and more flows. This will help us explore the state space more aggressively. This algorithm is motivated by simulated annealing [40], but we have found it gives similar results in a much shorter time than a naive simulated annealing solution.

It is also possible that even though there is still congestion in the network we have reached the global optimum. In this case the algorithm will give up after having tried to move even large aggregates in their entirety and failing to improve overall utility.

3.1.4 Preliminary Evaluation

To better understand FUBAR's potential, we set out to answer the following questions:

- Is it computationally tractable for real-world deployments, at least for an offline system?
- How good are the routing solutions it finds?
- Does it manage to alleviate congestion?
- Is it able to prioritize some flows, while still retaining good overall performance?
- How do the utility functions affect performance?

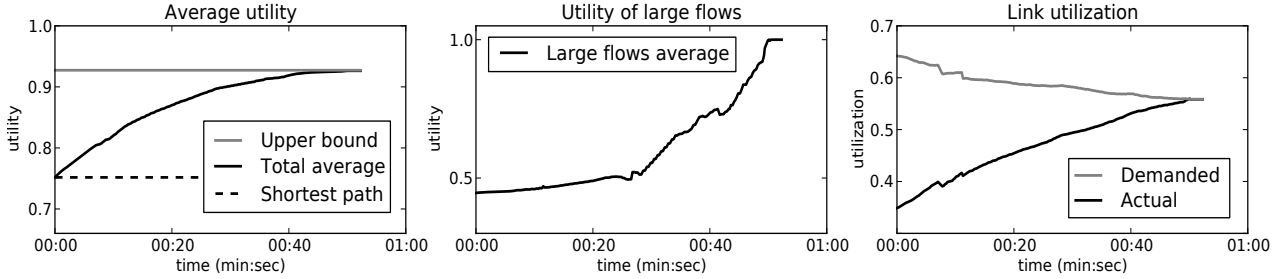


Figure 3.3: A run in the provisioned case

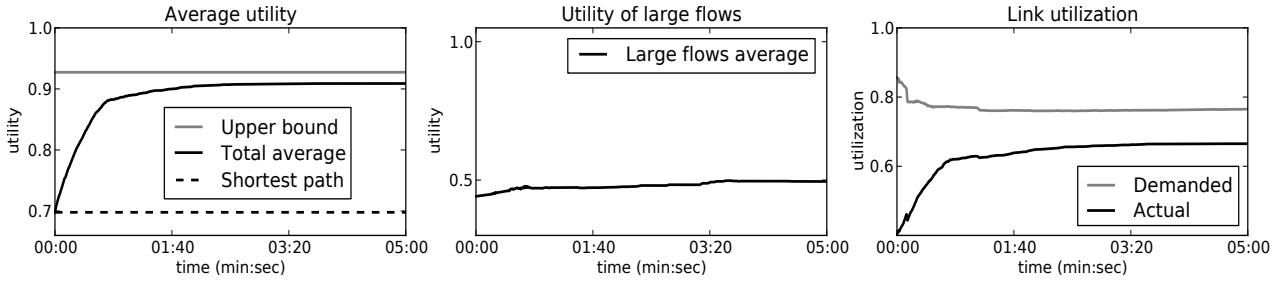


Figure 3.4: A run in the underprovisioned case

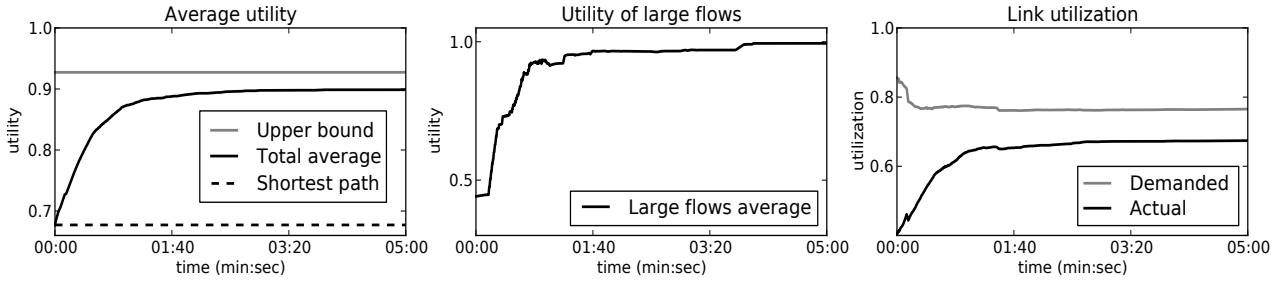


Figure 3.5: A run in the underprovisioned case with large flows prioritized

To answer those questions we run FUBAR using Hurricane Electric's core topology [34]. This contains 31 POP nodes and 56 inter-POP links. For each of all 961 aggregates we randomly pick either a real-time utility function or a bulk-transfer one. To reflect real-world traffic we also add a 2% probability of there being a large aggregate using a file transfer utility function with a higher max bandwidth (1 or 2 Mbps). Delay utility functions are as shown in Figures 3.1 and 3.2. We examine two main cases:

Provisioned: each link of the topology has a capacity of 100 Mbps. This is enough capacity to make it possible to alleviate congestion, but not enough capacity for every flow to be satisfied on its shortest path

Underprovisioned: each link of the topology has a capacity of 75 Mbps. In this case there is not enough capacity to completely eliminate congestion.

Figures 3.3 and 3.4 present a single run of FUBAR in each of these two cases running on a 1.3 GHz Core i5. The current implementation is single-threaded and implemented in Java. The graphs show how the algorithm progressively optimizes the computed solution in real-time. Utility values are predicted utilities based on the traffic model described in 3.1.2.3.

The leftmost graph shows how utility evolves over time. The "total average" is the overall utility of the network—the average of utilities of all aggregates, weighted by number of flows in the aggregate. The other two lines are for reference. To produce the "upper bound" curve we isolate an aggregate by removing all other aggregates from the network and determine what the single aggregate's utility would be if there were no other traffic. We repeat this for each aggregate and then take the mean. The "shortest path" line shows what utility would be if all the traffic takes its shortest path through the network.

The second graph on each row shows the utility of large flows, as these are harder to provide for and put disproportionate strain on the network.

In the right graphs we show link utilization. The "actual utilization" curve represents total used capacity divided by total network capacity[1]. The "demanded utilization" curve shows total demand[2] divided by total network capacity. Demanded utilization decreases as the optimization runs because total network capacity increases as more links are brought into play. If the two curves meet, demand has been satisfied.

Running time. The running times of both cases are within the five minute limit for an offline system. In the provisioned case FUBAR finds a solution in less than a minute, whereas it takes about five minutes to reach an optimum when underprovisioned. In the latter case, the algorithm spreads out traffic, lightly congesting more and more links, taking longer to test moves over each one of them. Eventually there isn't a move that can improve utility, and the algorithm terminates.

Solution quality. In both cases FUBAR improves significantly on traditional shortest-path routing. This is expected as it starts by putting all flows on the shortest paths and improves from there on. Thus shortest-path routing is a lower bound for total utility. In the provisioned case FUBAR closely approaches the upper bound. In the underprovisioned case, although FUBAR improves utility by over 30%, the upper bound is unreachable.

It is also interesting to observe the utility of aggregates with large flows. By default all flows are treated equally regardless of their volume. It is much easier to initially gain utility by moving the small flows as provisioning for them results in a quick utility boost. In Figure 3.3, only once the majority of small flows have been optimized do chunks of large flows start being moved; eventually all flows are optimized.

The observation that large flows are hard to optimize is also evident from Figure 3.4 where large flows are sacrificed to better accommodate smaller ones when resources are scarce. Thus the final utility of large flows is significantly lower than the global one.

Avoiding congestion. Minimizing congestion not only helps applications, but it also makes the network more predictable, as queue sizes are minimized. If there is enough capacity available we manage to eliminate congestion as is evident from the link utilization graph in Figure 3.3. When the two curves meet the capacity that is consumed by flows equals the demand of those flows and the network is not congested.

[1] total network capacity is the sum of capacities across all links *that are used.*

[2] this is what the flows would have liked to get, again on links that are used

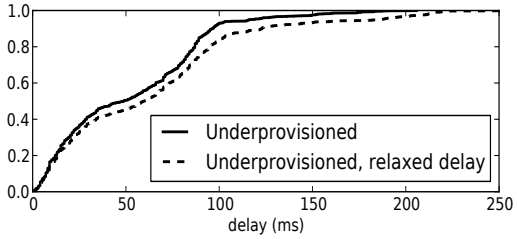


Figure 3.6: Increase in delay as result of relaxing delay restrictions

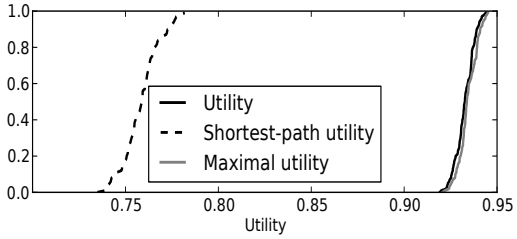


Figure 3.7: A CDF of 100 runs of the provisioned case

In the underprovisioned case link utilization improves by 80%, but in the end there is still congestion in the network (as evident from the gap). It would be possible to drive utilization even higher by saturating the network with large flows, starving smaller ones, but instead we try to improve utility for all participants.

Prioritization. Figure 3.5 shows a single run of the underprovisioned case when priority is given to large flows by increasing their weighting when computing the network utility. Now the utility of the large flows grows faster and is able to reach its peak. This comes with a slight increase of link usage, since larger flows occupy more network resources. The increase is not more because the number of small aggregates greatly outnumbers the number of large aggregates.

Another effect is the slower growth of overall utility because prioritizing large flows does initially hurt the numerous smaller ones. In the end, overall utility has not changed a great deal—the 1% reduction in small flow utility is offset by the gain in utility of larger ones.

Delay changes. To see the effect of the delay part of the utility function, we ran the underprovisioned case with small flows using double the delay parameter from Figures 3.1 and 3.2. Overall the results look very similar to the Figure 3.4 curve, though utility and utilization increase a little as longer paths are no longer penalized. Overall run time is also increased slightly since the algorithm takes more time to explore now-viable alternatives.

It is also interesting to see how the overall delay distribution changes. In Figure 3.6 we present CDFs of the delays of all flows in the network in the original and the relaxed delay case. It is evident that even though overall utility has increased, paths have lengthened, with a median delay increase of about $10\,ms$ and tail-end increase of about $50\,ms$. For real-time traffic, every millisecond matters; the ability to trade utilization for delay by tuning a single parameter is a very useful one for network operators to have at their disposal.

Repeatability. The results so far have only shown experiments with a single run of the algorithm. Are the results stable across different traffic matrices? We ran 100 passes of the provisioned case with the same topology, but with different random seeds for choosing the traffic matrices. We present the results in Figure 3.7 as a CDF. We show curves for FUBAR's utility in comparison to shortest-path and the upper-bound maximal utility. In all runs FUBAR's performance closely approached the theoretical limit, as in Figure 3.3.

3.2 Congestion as a Market Metric; Policing, Exposure (ConEx) and Feedback

The first Trilogy project explained that, when resources are pooled, there is no need to enforce sharing of the pool except at any instant when there is contention for the resource. Building on Kelly [39] the first Trilogy project proposed that it would therefore simplify resource sharing if a sufficiently robust congestion metric were available such that it could be used to enforce resource sharing. The congestion metric had to be fine-grained enough to measure instantaneous congestion, so the term 'congestion' does not necessarily signify a pathological condition unless it becomes persistent. The idea was not to *require* operators to share resources based on a congestion metric, but for such sharing to at least be straightforward. Then, as the market in resources tends to commoditise due to competition, more operators will tend to use the congestion metric, because it represents the underlying cost of the pooled resource.

Prior to the first Trilogy project, congestion-based resource control had always been framed in terms of dynamic congestion pricing. We introduced the idea of policing network traffic based on an explicit congestion metric tagged to the traffic with provable integrity properties, in a system called re-ECN [19]. The main problem with a congestion metric in a packet network is that it appears at the egress of the network, not at the ingress where a policer needs the metric to be visible so that it can prevent traffic entering the network if it has exceeded its congestion allowance. We solved this by the operator giving the sender the incentive to reinsert congestion feedback, then measuring the difference between this 're-feedback' signal and the pre-existing standardised explicit congestion notification (ECN) signal. Policing using congestion in this way has the same properties as dynamic pricing, but the dynamics of the price are hidden from the tenant (customer) within the policer. So tenants and network operators only see a predictable constant usage charge, and predictable, low congestion, low delay service.

The four subsections below report on work in the Trilogy 2 project that has built on the above foundations:

Congestion Policing (§ 3.2.1): We have documented *why* congestion policing is a better solution than traditional scheduling approaches.

Congestion Policers (§ 3.2.2): We have worked backward from the original ideal design for the whole distributed system to an approach that does not require any system-wide changes. As an initial deployment step, a 'bottleneck congestion policer' can be deployed as a drop-in replacement for a weighted round robin (or weighted fair queuing) scheduler.

Congestion Exposure (ConEx § 3.2.3): We have worked through the IETF with others to standardise ConEx. Major changes to the assumptions underlying the original re-ECN design have been made, in particular using loss as well as ECN as a congestion signal.

Congestion Feedback (AccECN § 3.2.4): It was always recognised that the congestion feedback information used in some common Internet transport protocols (particularly TCP) were insufficiently accurate for ConEx to use. Perversely, TCP's loss feedback is sufficiently accurate, but its ECN feedback is not. We have proposed a general-purpose ECN feedback mechanism to replace that of TCP, which fixes the accuracy problems but also fixes a flaw in the design of Data Centre TCP.

3.2.1 Congestion Policing

We have written up a full explanation of why congestion policing isolates the network performance of different parties who are using a network, and why it is more useful than other forms of performance isolation such as peak and committed information rate policing, weighted round-robin or weighted fair-queueing.

The write-up was originally submitted to the IETF ConEx WG as an individual submission [16], which means it has not been adopted onto the IETF's standardisation agenda. The draft is about congestion policing in general, whether or not it uses ConEx signals. Therefore, now that the basic ConEx mechanism drafts have been completed, and the ConEx working group is being closed (see § 3.2.3) this draft will probably be re-targeted at another venue, probably the IETF's transport area working group that has started working on congestion-based traffic management using tunnels (see § 3.2.2).

The main benefit of congestion policing is that it isolates performance even when the load applied by everyone is highly variable, where variation may be over time, or by spreading traffic across different paths (the hose model). Unlike other isolation techniques, congestion policing is an edge mechanism that works over a pool of links across a whole network, not only at one link. If the load from everyone happens to be constant, and there is a single bottleneck, congestion policing gives a nearly identical outcome to weighted round-robin or weighted fair-queuing at that bottleneck (see Figure 3.8c)). But otherwise, congestion policing is a far more general solution.

The key to the solution is the use of congestion-bit-rate rather than bit-rate as the policing metric. *How* this works is very simple and quick to describe (§ 3.2.2).

However, it is much more difficult to understand *why* this approach provides performance isolation. In particular, why it provides performance isolation across a network of links, even though there is apparently no isolation mechanism in each link. The draft [16] builds up an intuition for why the approach works, and why other approaches fall down in different ways. It is written in the context of a network of multi-tenant data centres, which reflects the scenario we are focusing on in the Trilogy 2 project. And all the numerical examples are based on a data centre scenario. Nonetheless, congestion policing can be applied to any network deployment scenario, whether a mobile access network, a broadband access network. The data centre scenario is particularly relevant because carrier networks are increasingly being built using the same virtu-

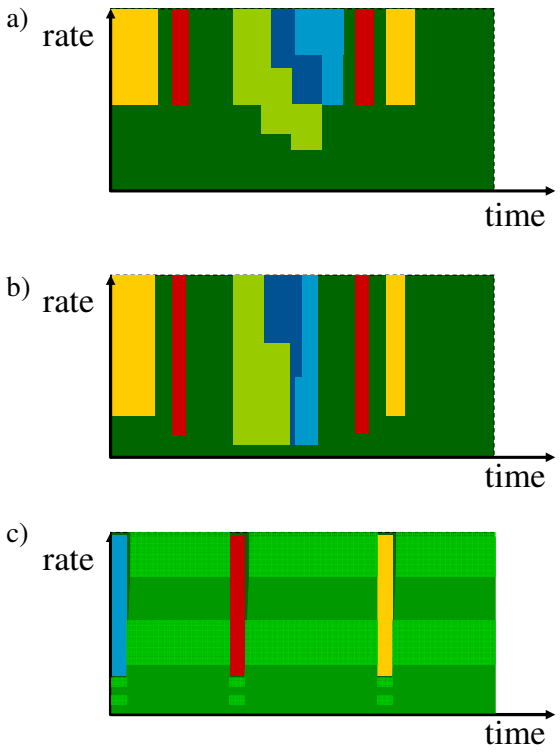


Figure 3.8: a) Weighted Round Robin (WRR) scheduler for comparison; b) Congestion Policing: Intended Performance Isolation Outcome; c) A congestion policer tends to to WRR scheduler for persistent loads.

alisation technology as multi-tenance (cloud) data centres. Also the data centre scenario can be extended to include the internals of the I/O subsystem within a server, allowing general-purpose network technology to be used for storage I/O or general data networking, while isolating each from the other for each tenant.
The bullets below provide a summary of the explanation in Section 3 of the draft [16], which builds from the simple case of long-running flows through a single link up to a full meshed network with on-off flows of different sizes and different behaviours:

- Starting with the simple case of long-running flows focused on a single bottleneck link, tenants get weighted shares of the link, much like weighted round robin, but with no mechanism in any of the links (see Figure 3.8c)). This is because losses (or ECN marks) are random, so if one tenant sends twice as much bit-rate it will suffer twice as many lost bits (or ECN-marked bits). So, at least for constant long-running flows, regulating congestion-bits gives the same outcome as regulating bits;

- In the more realistic case where flows are not all long-running but a mix of short to very long, it is explained that bit-rate is not a sufficient metric for isolating performance; how *often* a tenant is sending (or not sending) is the significant factor for performance isolation (see Figure 3.8a)), not whether bit-rate is shared equally whenever a source happens to be sending;

- Although it might seem that data volume would be a good measure of how often a tenant is sending,

we then show why it is not. For instance, a tenant can send a large volume of data but hardly affect the performance of others – by being more responsive to congestion (see Figure 3.8b)). Using congestion-volume (congestion-bit-rate over time) in a policer encourages large data senders to be more responsive (to yield), giving other tenants much higher performance while hardly affecting their own performance. Whereas, using straight volume as an allocation metric provides no distinction between high volume sources that yield and high volume sources that do not yield (the widespread behaviour today);

- We then show that a policer based on the congestion-bit-rate metric works across a network of links treating it as a pool of capacity, whereas other approaches treat each link independently, which is why the proposed approach requires none of the configuration complexity on switches that is involved in other approaches.

- We also show that a congestion policer can be arranged to limit bursts of congestion from sources that focus traffic onto a single link, even where one source may consist of a large aggregate of sources.

- We show that a congestion policer rewards traffic that shifts to less congested paths (e.g. multipath TCP or virtual machine motion).

- We show that congestion policing works on the pool of links, irrespective of whether individual links have significantly different capacities.

- We show that a congestion policer allows a wide variety of responses to congestion (e.g. New Reno TCP, Cubic TCP, Compound TCP, Data Centre TCP and even unresponsive UDP traffic), while still encouraging and enforcing a sufficient response to congestion from all sources taken together, so that the performance of each application is sufficiently isolated from others.

3.2.2 Congestion Policers

ConEx depends on deployment of modified senders, then individual networks can use the congestion signals they provide to give them better service. In the original Trilogy project we recognised that it is always hard to get a system deployed if it requires multiple parties to deploy different parts. We analysed the deployment incentives and found that there could be useful incentives for senders to make the first deploy step, however an end-system developer would have to be convinced that a network would use the signals if it made the first move.

In Trilogy 2 we have been investigating a different approach where the network operator unilaterally deploys the whole policing system, at least initially. Figure 3.9 shows an evolution to the full ConEx scenario (Figure 3.9c)) starting from an initial deployment using a bottleneck congestion policer (Figure 3.9a)). Each approach solves trustworthiness of the congestion signals used by the congestion policer in a different way. The first two (a),(b) ensure trustworthy signals by keeping full control of the whole congestion signalling and policing mechanism. Whereas in the last case (c) the tenant generates the ConEx signals, but an audit

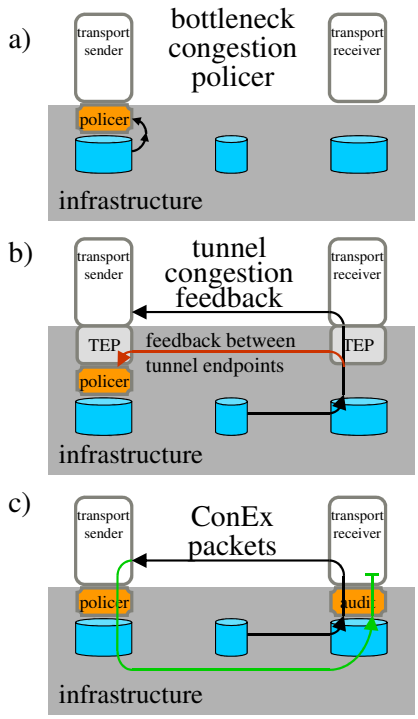


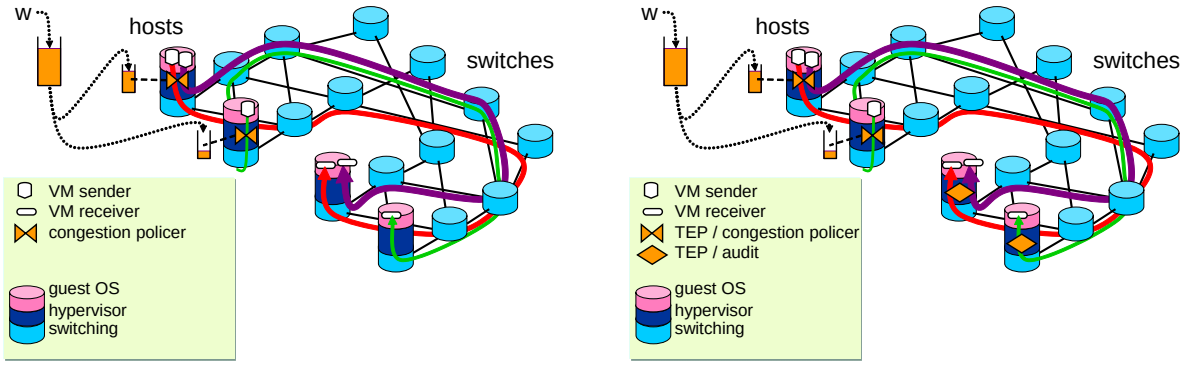
Figure 3.9: Trust Models for Path Congestion Feedback

mechanism at the network egress ensures that there is never any benefit in cheating. This audit system was a focus of the original Trilogy project and is not described further here.

The bottleneck congestion policer (BCP) is applicable where the network operator has arranged the primary bottlenecks in the access around the edges of the network, which is a common approach in many network designs, e.g. broadband, cellular, and some data centres. The tunnel congestion feedback approach (Figure 3.9b)) is an alternative starting point with all the policing mechanisms still within the control of the network operator (the grey infrastructure box). Tunnel congestion feedback would be a useful starting point if the network design is not primarily bottlenecked at the edges. Even if the operator starts with the edge bottleneck design, it could progress to tunnel congestion feedback to deal with occasional congestion in its core.

Whereas Figure 3.9 shows one path through the network, Figure 3.10 attempts to show a whole network of users, in order to emphasise where there is contention due to paths crossing. It compares a network arbitrated by bottleneck congestion policers (Figure 3.10a) with a network protected by ConEx policers and audit functions (Figure 3.10b).

We have fully written up the ConEx and tunnel congestion feedback approaches in an IETF Internet Draft [22]. Again, this is being migrated from the ConEx working group to the IETF's transport area, given it is more generally applicable than just ConEx.



(a) Locally Bottlenecked by Design (b) Fully Distributed (Edge and Core) Queue Control

Figure 3.10: Evolution of Congestion Policing Architecture

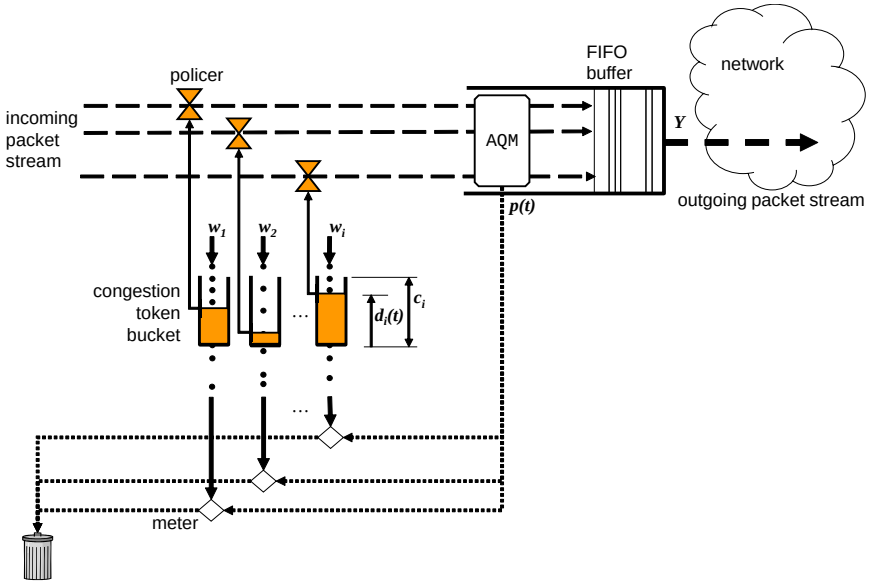


Figure 3.11: Schematic of Bottleneck Congestion Policer

3.2.2.1 Bottleneck Congestion Policer (BCP)

Unlike ConEx or Tunnel Congestion Feedback, there is no need to standardise a bottleneck congestion policer design; it is purely an algorithm that can be used as a drop-in replacement to improve on existing schedulers, such as weighted round robin (WRR) or weighted fair queuing (WFQ). Outside the IETF, we have been investigating one particular form of bottleneck (non-ConEx) congestion policer [15] and Wagner (outside the Trilogy 2 project) has proposed two alternative designs of bottleneck congestion policer and compared all three using simulation [15, 65]. However, unfortunately, the comparison was made on performance for an individual user, rather than isolation between users.

Our preferred BCP design is shown schematically in Figure 3.11. It is extremely simple, using only a single FIFO queue containing an active queue management (AQM) algorithm (any algorithm would suffice, e.g. RED, adaptive RED, PIE, etc.). The 'congestion allowance' of each user i is represented by the token fill-rates w_i. And the state of the congestion 'account' of each user is represented by the fill level of their bucket $d_i(t)$. If a tenant's bucket is empty, the policing function discards that tenant's packets until the bucket is no

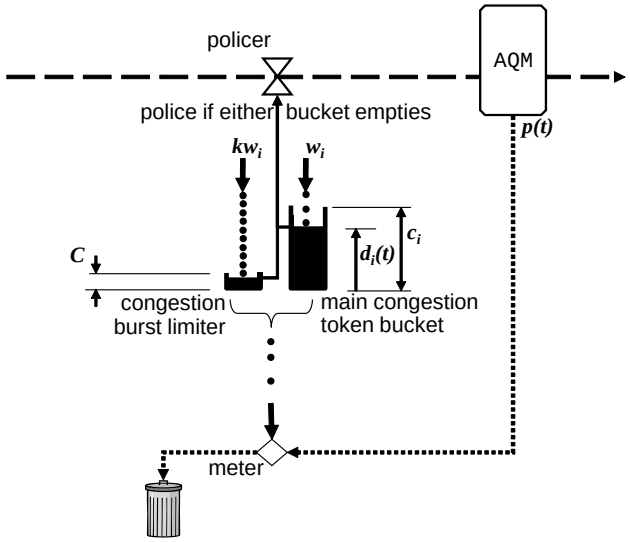


Figure 3.12: Schematic of Dual Token Bucket for Bottleneck Congestion Policer

longer empty.

Conceptually, if the AQM drops a particular tenant's packet, the size of that packet is deducted from that tenant's bucket level. When the AQM drops packets it does so randomly. Therefore, at a particular level of congestion, the proportion of all the drops that one tenant attracts will be proportionate to their share of the load. The probability that the AQM drops packets rises with queue depth. Therefore, the deeper the queue at the time a tenant sends a packet the more likely their packet will be dropped and therefore subtracted from their bucket. Therefore, the buckets that empty fastest will belong to those tenants who send most traffic at the most congested times. Conversely, a tenant can prevent their bucket from emptying when there is no congestion—even a large amount of traffic at uncongested times will not reduce that tenant's bucket depth.

Each bucket in Figure 3.11 is actually two buckets both of which have to be non-empty to allow that user's packets to flow (Figure fig:conex-bcp-dual-tb). The primary bucket can be very deep (minutes or hours of usage at normal congestion levels). The secondary bucket must be very shallow (e.g. one or two packet's worth of congestion), but it is filled at a much faster rate than the primary bucket (e.g. $k = 10$. The secondary bucket ensures that there is no benefit if a tenant saves up congestion allowance then uses it all in one burst. In particular, it regulates the transient congestion that a tenant can cause by starting up a number of flows all at once.

Any tenant that loads the system beyond their own allowance suffers focused drops from their own policing function, and therefore cannot send more traffic into the shared queue. The sum of all the secondary congestion allowances limits the total possible drop rate from the shared queue, and therefore places a hard limit on the maximum queue depth. The sum of all the primary congestion allowances limits the long-run average queue depth.

Pseudocode for a BCP is given below. Variable names not defined below are as defined above.

1: **for** $each\ pkt$ **do**

```
2:    i = classifyUser(pkt)
3:    i.d1+ = i.w * (t(now) − i.t)                    ▷ fill
4:    i.d2+ = k * i.w * (t(now) − i.t)                ▷ fill
5:    i.t = t(now)
6:    ▷ p: AQM drop prob, s: size of pkt
7:    i.d1− = s * p                                   ▷ drain
8:    i.d2− = s * p                                   ▷ drain
9:    if i.d1 < 0 || i.d2 < 0 then
10:       drop(pkt)                                   ▷ police
11:   end if
12: end for
```

It can be seen that the policer is extremely simple, being implemented in just nine lines (and there are obvious optimisations possible in the duplicated fill and drain lines). This intrinsic simplicity can be attributed to using the correct metric. There follows an explanation of the pseudocode:

Fill: After classifying the packet to match it to a tenant ID, a standard token fill algorithm is used to update both the tenant's buckets, taking the product of the fill rate and the time since this tenant last sent a packet;

Drain: The (identical) lines that drain the two buckets are the most subtle aspect of the algorithm. The algorithm does not actually subtract whole dropped packets from the bucket of the tenant that sent each dropped packet. Instead, it continuously deducts a proportion of each packet from the bucket of the tenant sending each packet. The proportion is determined by the drop probability variable p taken from within the AQM algorithm.

Police: If either of the tenant's buckets is empty after filling and draining, the final line drops the arriving packet.

Without the smooth draining function, the policing function would be rather severe. Once a tenant hit the bottom of the bucket, it would drop a sequence of packets, then none. With smoothed draining, the policer is gentle enough to smoothly take over the role of congestion control from a source that is persistently unresponsive to congestion.

A more sophisticated two-tier policer (per tenant, then per-flow) could be used that focused drop on those flows from the tenant that were causing most congestion. However, it should be rare for a policer to actually have to police a tenant, therefore it seems misplaced to add complexity just to punish a transgressing tenant in a considerate way. Therefore we believe the simple policer shown will be sufficient—if it does have to act, it treats all of a tenant's traffic equally. Under normal conditions a tenant's own congestion control on each flow will keep their bucket from emptying, avoiding ever having to experience policing.

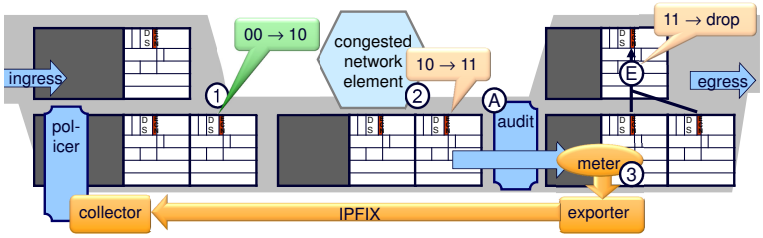


Figure 3.13: Congestion Feedback over a Tunnel

The firm that supplies BT with traffic policing equipment has implemented the BCP in software (outside the Trilogy 2 project). They compared the BCP with the Fair Queuing (FQ) and Stochastic Fair Blue (SFB) algorithms. They initially compared them based on Jain's Fairness Index using just long-running persistent traffic from all users. At low traffic rates (e.g. 200Mb/s), all the approaches gave sufficiently fair results (BCP was best, but both BCP and FQ gave a JFI of over 95%, while SFB was more variable but always over 85%). However, at higher data rates (e.g. 4Gb/s) the fairness index of FQ and SFB collapsed to respectively 65% and 60%, while the BCP continued to give a consistent JFI of over 95%. The BCP continued to perform at this level of fairness at the greatest load the testbed could apply, 51Gb/s shared between sustained traffic from 20 million tenants.

Of course, the BCP is designed to maintain proper 'cost fairness' between tenants when they apply loads with significantly different profiles over time (e.g. bursty, versus smooth). However, even in the rudimentary scenario of persistent non-bursty traffic load, the BCP clearly leaves other designs standing, due to its far simpler, single queue design.

As outlined in the introductory explanation, the bottleneck congestion policer is only applicable in a network designed to keep most congestion at a bottleneck at the edge of the network, where state for mapping interface identifiers to tenant identities is already in place. If congestion is more evenly spread throughout the network core, or if there is potential for persistent congestion at the egress of the network, then tunnel congestion feedback will be needed.

3.2.2.2 Tunnel Congestion Feedback

We have fully documented the mechanism for congestion policing based on tunnel congestion feedback (and for ConEx-based policing) in [22]. Figure 3.13 summarises the idea. For a pure tunnel congestion feedback solution, there would be no audit function (A). It is included so that the diagram can also be used to explain the hybrid tunnel/ConEx solution later.

The diagram shows packets being encapsulated (tunnelled) at the ingress to the network on the left, and decapsulated at the egress on the right. In all multi-tenant data centres, some form of encapsulation is used anyway to provide virtual private network addressing. For congestion feedback tunnelling we require IP-in-IP tunnelling (there can be a shim layer between the two), but other tunnelling protocols could be adapted to provide similar capabilities (see "Guidelines for Adding Congestion Notification to Protocols that Encapsulate

IP" [20]).

The idea is to enable ECN on all switches (2) within the network and enable ECN (1) for all traffic across the tunnel, even if the endpoints are not ECN-capable. Then any congestion within the network is signalled as an ECN mark to the egress of the tunnel. This is because ECN is easy to meter at the tunnel egress, whereas drops are much harder.

However, if either of the endpoints is not ECN-capable, they will not understand ECN marks. So the tunnel egress converts ECN marks into drops, but only if the e2e (inner) packet is not ECN-cpaable. It does not need to do anything special to achieve this, it merely has to exploit standard ECN decapsulation behaviour [10], which requires it to drop any packet that arrives with an ECN mark in the outer, when the inner is not ECN-capable. On the other hand, if the inner is ECN-capable, the decapsulator propagates the ECN mark to the receiver by marking the inner, and it does not drop the packet.

This arrangement effectively defers all dropping from the core to the tunnel egress using just standard tunnel components. Then the a simple meter (3) at the egress monitors ECN marks, and feeds back this information to the congestion policer at the tunnel ingress. IPFIX is appropriate to use as the feedback protocol. The only non-standard behaviour is to turn on the ECN capability in the outer of every packet at the ingress (1).

Our draft [22] proposes that the egress meter ignores any traffic that the sender has marked as ConEx-capable. This is because, for ConEx traffic, the ingress policer can use the ConEx signals in the packet directly. As mentioned above, in this hybrid design, ConEx and non-ConEx traffic each use different trust mechanisms to ensure the ingress policer has trustworthy congestion information:

- for non-ConEx traffic, the operator's tunnel feedback provides trusted congestion information;
- for ConEx traffic, the ConEx signals provide the trusted congestion information, and the audit function ensures that trust is warranted.

The tunnel congestion feedback mechanism has been 'reinvented' by Wei *et al.* [66] and the IETF's transport area working group chairs have recently encouraged them to work with us on a joint draft.

As outlined in the introductory explanation, tunnel congestion feedback is useful when congestion is not mainly confined to the ingress edge. Nonetheless, if ConEx is deployed in sources, the congestion information it provides is preferable over tunnel congestion feedback, because:

- the signalling work is done by the endpoints, which already have to feedback congestion information (Figure 3.9b) shows the unnecessarily duplicated feedback flows);
- ConEx congestion signals give the conservative expectation of congestion as calculated by the endpoints, whereas tunnel feedback is delayed by at least one round trip time. The endpoint is in the best position to know when it is sending a burst that might cause congestion, and the ConEx audit framework ensures the source is given sufficient incentive to reveal this risk honestly.

Thus in the ConEx case, the policer can work on information that represents the source's conservative expectation of the *risk* of congestion, and it should therefore generally prevent one tenant's congestion from affecting others, even if caused by incast.

3.2.3 Congestion Exposure (ConEx)

As well as developing the above 'non-ConEx' solutions we have continued to complete the informational and experimental specifications necessary to document ConEx itself. Documenting ConEx has the obvious purpose of specifying a standard protocol so that implementations will interoperate. But it also defines a benchmark against which alternatives such as those introduced above can be measured.

Beyond the original RFC describing ConEx Concepts and Use-Cases [11], we have not been directly involved in co-authoring ConEx drafts other than the primary documentation of an abstraction of the mechanism [48], which will be briefly outlined below.

We have helped review the other main ConEx mechanism drafts, all of which have now completed the working group phase and are moving on to the Area Director and Steering Group review phase:

- "IPv6 Destination Option for ConEx" [42], which specifies the ConEx wire protocol for IPv6;
- "TCP modifications for Congestion Exposure" [43], which chooses TCP as an example of how to generate ConEx signals, given TCP's role as the most prevalent and important transport protocol;
- "ConEx Crediting and Auditing" [64], which complements our draft on Congestion Policers to Specify Credit and to Define an Example Audit function.

Given the closure of the ConEx working group, the authors of the last draft (Credit and Audit) are being encouraged to take it through to RFC as an Area-Director-sponsored draft.

Our Abstract Mechanism draft ([48] co-authored with Matt Mathis of Google), is passing through the steering group review phase at present. The primary purpose of the Abstract Mechanism is to provide a roster of the parts of the system that have to be defined in the concrete specifications, and to explain the concepts and how all the parts work together (i.e. the architecture). As well as the various device functions (senders, receivers, policers, auditors), a section is devoted to requirements for the wire protocol.

The mechanism has been defined in abstract terms as well as specific concrete specifications, because it was known in advance that compromises would have to be made in any concrete specification (primarily because of the constraints imposed by the existing IP protocol). Therefore, it was considered important to state the goals and ideals, so that any set of compromises would be understood with eyes open. And if that set of compromises proved unsuccessful, the original goals would still be documented in case it was decided to work out a new set of compromises.

Re-ECN [19] represented the first attempt to decide on a good set of compromises, and the first set of ConEx specifications itemised above represents the second attempt:

- The main problem with Re-ECN [19] is that the ECN field in the IP header is used to encode the signals, so it is only possible on ECN-capable packets. Therefore congestion exposure is not possible if the receiver is not ECN-capable. However, using the ECN field does theoretically make ConEx propagate correctly over tunnels.

- The ConEx wire protocol [42] does not depend on both ends supporting ECN. However its main problems are:

  – it is for IPv6 only (an individual submission has been proposed to make extra space in the IPv4 header for ConEx and other extensions [17], but it would only be usable in private networks because the IETF is becoming resistant to improving IPv4 any more);

  – ConEx signals are not visible in the outer IP header of a packet if it is tunnelled using IP-in-IP. Therefore a ConEx device has to look for an inner IP header to find the ConEx information.

3.2.4 Congestion Feedback (AccECN)

The standard (New Reno) variant of the TCP algorithm only responds once per round trip to any amount of congestion signals. Most TCP variants until recently (Cubic, Compound, etc) did likewise. This behaviour stems from the typical drop-tail behaviour of buffers when TCP was invented, which manifests as a burst of correlated losses every time there is a congestion event. Unfortunately, TCP's original congestion feedback mechanism was designed on the basis that any more than one congestion signal per round trip would be ignored, rather than feeding back the full information, in case the sender might want to use it in future.

That future scenario is indeed upon us. Both Data Centre TCP [1] and ConEx senders need to know the extent of congestion during a round trip, not just whether at least one congestion event occurred. TCP Selective Acknowledgements (SACK [47]) do feed back fine-grained loss information to the sender and most TCP endpoints now support SACK. However, perversely, the addition of ECN to TCP [57] did not following this lead, and still feeds back no more than one mark per round trip.

DCTCP uses ECN, so to feed back fine-grained congestion information, a DCTCP receiver uses the same 'Echo Congestion Experienced' flag as defined for standard ECN-TCP [57] but it defines very different semantics. Unfortunately, the DCTCP feedback scheme is flawed—if ACKs are lost it becomes highly ambiguous, which is unsafe for a congestion avoidance protocol.

Therefore, we have persuaded the IETF that an extension to TCP is necessary, for a TCP receiver to be able to provide accurate feedback of the signals received in each arriving ECN field. We convinced the IETF's TCP maintenance working group to charter an informational RFC giving the motivation and requirements for this extension. Following numerous revisions, this requirements draft has now completed the working group phase and is working through the steering group review phase on its way to becoming an RFC [44].

We have also prepared a candidate solution to this problem (we have actually proposed a few alternative solutions in appendices and converged on a preferred one in the body of the draft [21]). The draft is still

and individual submission, with no formal status at the IETF. However, it is the only candidate to address the requirements, so once the requirements have been published as an RFC, we are hopeful this draft will be adopted onto the IETF's experimental standardisation track.

Work on these two drafts is primarily undertaken in and reported via the Reducing Internet Transport Latency (RITE) EU FP7 project. However, the Trilogy project is also acknowledged given the motivation for the work includes the congestion exposure (ConEx) protocol.

Our candidate solution (called Accurate ECN or AccECN for short [21]) is rather more complex than we would have liked, in order to fit into the limited space in the TCP header without using TCP options, given such a large proportion of Internet paths block or strip unrecognised TCP options (see § 2.5.1). Given we have recently developed what we believe is a general solution to middlebox traversal and space limitations for TCP options (see Inner Space in §2.5), we have placed our AccECN work on hold while we establish whether the Inner Space solution is viable, and whether the IETF will adopt it. If so, a more straightforward AccECN solution will be possible.

3.3 The Federated Market

One of the key aims of WP2 is to solve the issue of controlling pooled resources in a manner that "reflects a tussle between interests that sometimes align and sometimes conflict."[3] One of the key models investigated is that of a Federated Market for resource sharing. This differs from the centralised model where all the resources are owned by a single party such as storage and network resources provided by the Akamai CDN platform or compute resources from a public cloud provider such as Amazon or Rackspace. This model is sustainable and profitable for those with a large enough infrastructure to gain market traction. However for companies and enterprises that don't have access to such resources, or who have spare resources they would like to monetise, the Federated model may be a better approach.

The Federated Market (described in Deliverable D1.3) offers a system that allows server resources to be traded on the open market. It offers resource owners the chance to monetise under-utilised resources and offers buyers the chance to create flexible virtual Clouds that can be scaled to meet their exact needs. By making resources more liquid and easier to migrate between owners, the centralised system moves towards commodity or utility computing that is common in many other markets where resources are traded.

Other approaches exist but the Federated Market model is particularly powerful and has proved successful in the telephony and Internet service industries (see Figure 3.14). The implementation of the Testbed platform and subsequent use of the software platform in the real world has uncovered some key properties of a generalised market system for trading of resources. In the rest of this section we will detail some of the benefits that will encourage adoption of such a model and highlight some of the management properties that allow for this tussle to be resolved.

[3] See WP2 description in DoW.

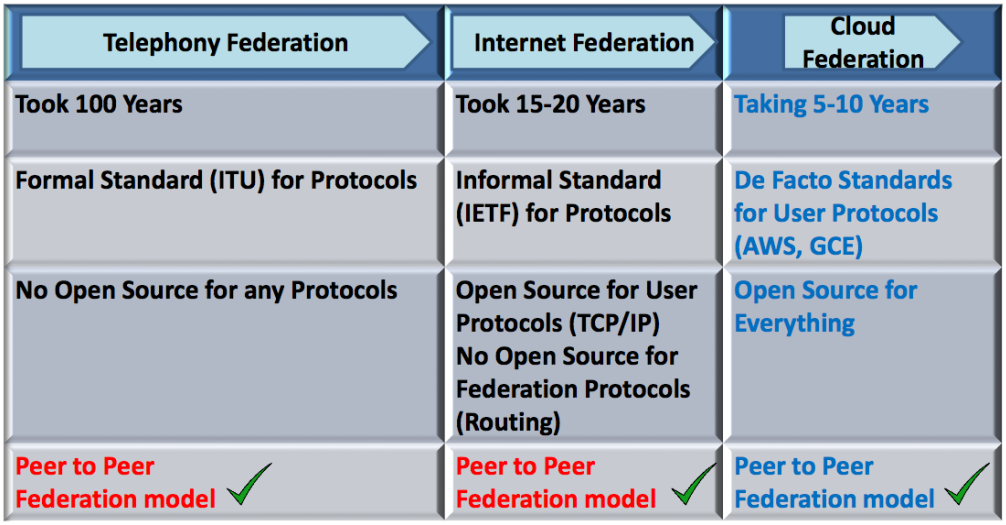


Figure 3.14: Federation is a proven strategy[4]

3.3.1 Description of the Market

The OnApp Federated Market, often just referred to as the Market, is a tool that has been developed in the scope of WP1 "Creating Liquidity" to allow the buying and selling of compute resources across cloud platform providers. The Market allows cloud owners to sell under-utilised resources to buyers that are willing to pay for them. More technical details about the architecture and platform can be seen in Deliverables Deliverable D1.1 and Deliverable D1.3. This section describes the management of that liquidity as opposed to the tools that are used to create the management infrastructure.

One of the key aspects that allows the platform to work is the use of a Federated system. This allows each cloud provider to maintain ownership and control of resources but allow certain operations to be performed by registered agents on local systems by remote users. In the Federated Market, each cloud provider can assign a 'zone' of resources. This includes a set of servers with an associated billing plan, set of restrictions/constraints, VM templates (see Figure 3.15) and physical resources set up in a particular configuration. In the current implementation this configuration cannot be changed by the remote user and is static once assigned to the Federation.

3.3.1.1 Actors and Roles in the system

There are three roles within the system:

The Seller (synonymous with supplier) has resources that they want to make available to other users through the platform.

The Buyer acquires resources from one or more sellers that it selects.

The Platform (the Market Engine) refers to the software that handles the communication between seller and buyer through receiving and sending messages that are then interpreted and stored appropriately.

[4]Image from `http://www.intercloudtestbed.org/uploads/2/1/3/9/21396364/intercloud_master_design_v5.pdf`

The interactions between these different actors are described in Deliverable D1.3.

Figure 3.15: One hypevisor can support multiple VMs with different templates

3.3.2 Ownership and Accountability

There are many legal implications and ramifications of running services on remote infrastructure[5]. Privacy policies and Terms of Service will determine what workloads and content can be stored and executed on the hardware. Currently normal practice is for a service provider that is hosting content to specify what is allowed and what is not allowed and if there is an issue raised, ask the user to remove the content, delete the content itself and/or terminate the service. The exact policy varies between hosting providers and may also be subject to the laws and constraints of where the hardware and content is physically located. Many providers will also pass on the liability of hosting any unsanctioned content to the users. In the Market the server/Datacenter owner that provides resources is ultimately responsible for the infrastructure that they own and so a similar system of rules and constraints would be needed.

There may also be cases where the content itself is legal but the actual activities being executed on the infrastructure are damaging or illegal. For instance the nodes may be being used as part of a Distributed Denial of Service (DDoS) attack or may host spam advertising or email botnets. Anyone investigating such a case will normally start by contacting the registrar of the IP address(es) involved who will then forward it to the provider. The provider will then determine a suitable course of action depending on the policies and

[5]The Cloud Legal project is a good source of information - `www.cloudlegal.ccls.qmul.ac.uk`

contracts involved. A more difficult type of workload to police is that of sustained resource (ab)use that has a detrimental affect to other users in the system. Internet Service Providers (ISPs) who offer unlimited network resources, often have a Fair Usage Policy (FUP) that states whether some activity is deemed as excessive and allows for limiting of services after a certain threshold. Similar policies are likely to make there way into other resource providers to allow for sustained fair usage for all customers.

In the Market, resource sellers are required to give at least a months notice if they are intending to remove physical resources from the Market so that warnings can be sent to end-user customers and sufficient time provided for migrating content. This is enforced as a contractual obligation between resource providers and the Market provider and ensures a certain level of stability for the customers. This in turn is seen as an incentive to customers who will have a chance to migrate to another provider. Service Level Agreements (SLAs) exist between companies to enforce a given level of service which if not maintained would be lead to an obligation to pay back credits/money or some other pre-agreed services. While it is hard to force a provider to stay online the risk of fines and legal redress should help and this is a risk that affects all users of Cloud platforms where they don't own the hardware.

3.3.2.1 Trust in the platform

Trust is an important element that is associated with the contracts and agreements set up between different stakeholders in the Market. Stakeholders must trust others in the system in order to fulfil the service obligations. There are various levels of trust throughout the Market system as summarised below.

The Buyer trusts that:

- The Seller's resources are as described and not over-provisioned, access to the resources exceeds some minimum level, resources will be secured according to a Security policy and only accessible via authorised agents and an appropriate service level is maintained as per the SLA.
- The Platform is impartial and offers a fair comparison of resources on offer with a transparent search and filtering system, has made suitable pre-checks to ensure the Sellers are valid, resources identified on the platform are consistent with what the seller is advertising and that resources are clear and unambiguous.

The Seller trusts that:

- The Buyer will honour payments in a timely manner, honours contractual obligations including but not limited to SLA, policies and TOC, acts within the laws of the country/jurisdiction hosting the content and/or services and does not abuse resources or cause the service of others to perform in a degraded manner
- The Platform will accurately represent the resources that are available and any additional services that are included, will fairly list resources and has the ability to easily search and filter results.

The Platform trusts that:

- The Buyer will make the payments and honour the terms set out by the Seller, will not degrade the performance of the platform and acts in good faith while using the platform.
- The Seller will capture and bill for the use of resources accurately, transparently and fairly.

As described before there is a minimum time for sellers to take down resources that it has added to the Market. This helps to ensure that buyers and end-users of the system have time to react to changes to the system.

3.3.3 Security model

As described in Deliverable D1.3, security models designed for a single cloud have to be adapted when multiple clouds are connected together. Agents that act on behalf of a remote system will be installed on local hardware and will have permission to perform certain activities such as the creation and deletion of virtual machines. Currently the Market interface creates a user when joining or publishing a Federated Market zone. This user has a privileged role set assigned to it which makes it a new attack vector. Thus any API commands that come through this user must be validated as coming from an authenticated user. Also the agent is a secured user with anspecial role set within the management user interface that can only be assigned by an administrator. Only zones that are published to the Federated Market will be made visible to other users of the Market. Currently white/black-listing support is not implemented but it is envisaged that the Market will filter out some results that are inappropriate such that some buyers will not see some sellers. This may be because some providers want to only make resources accessible to a sub-set of the market, or some buyers may have restrictions on where they can purchase resources from. A good example of this is EU Data Protection laws that place restrictions on where Personal Data may be stored and processed.

3.3.4 Resource Pricing and the Market

Successful markets exist in many domains ranging from the travel industry to utility companies. Within Europe two of the best models for our Federated Market are the telecommunications industry (both fixed and mobile) and the Internet Service Provider industry. Below we list some of the most important market functions as they relate to the Federated Market.

- Accessibility to the market. Some markets are only accessible to a set of entities. If this set comprises only of companies in the same enterprise then it is considered a closed market. If an entity can join the market after certain pre-conditions are met then it is described as an open market. The Federated Market is designed as an open market.

- Centralised or distributed. A market can be centralised with all resource trading performed at a centralised point. Alternatively it can be distrbuted with multiple points of access to the market. Currently the Federated Market is centralised but the intention is to make it distributed.

- Contractual obligations. There may exist several contracts established across the platform that detail the roles and responsibilities for entities involved. Service Level Agreements (SLAs) can be used to specify the services expected. Within the Federated Market this includes the obligation to give 4 weeks notice of turning off a given server as well as any SLAs that may be agreed between Sellers and Buyers.

- Regulators. Markets may require regulation either by an established group of users of the platform, by an entity established solely for this purpose or by a government regulator. Within the Federated Market model the Provider acts as a regulator.

- Competition for resources. Resources may be competed for based on several models that are associated with the pricing model. Competition in the market can help dictate fair prices at a level that the buyers are willing to spend. If there is no competition then the prices may be artificially high or low, controlled by the dominant operator. One of the aims behind the Federated Market is to encourage greater competition by making it easy for operators to trade under-utilised resources.

By providing the infrastructure needed to create a marketplace, the Federated Market should lead to the creation of a competitive market in cloud computing. Already Cloud.net has customers in 16 locations ranging from Atlanta to Zurich.

3.3.4.1 Resource Flexibility

One of the key benefits the Federated Market gives is avoiding vendor lock-in by allowing Buyers to create clouds hosted in a number of different zones. This also makes it extremely easy to scale out and to create clouds in the location where you want them. The concept of standard VM templates further simplifies this process.

Over-subscription can be used to increase the overall usage of the platform by increasing the number of resource users that are assigned to resources on the basis that the workloads will not all be running at full utilisation all the time (statistical multiplexing). Some resources are more suitable to over-provisioning than others. Memory ballooning allows Hypervisor platforms to give a brief increase in RAM but the administrator and/or platform has to be careful to ensure that if there are several VMs on the platform with ballooning enabled their memory regions cannot possibly overlap. For storage, not having access to expected resources is usually less critical than not being able to request and assign RAM that may block a particular process. The requirements of the workloads though ultimately decide which resources are critical and cannot be shared.

Operating System type workloads that have end-users running different tasks leads to burstier demand making them well suited to resource sharing, but if too much over-provisioning is used this leads to resource congestion and contention with the potential to impact all users. In a truly liquid system, the price of resources will change based on supply and demand and if these values are accessible and known, users can move workloads accordingly. This is already the case in closed-system providers such as Amazon that offer spot-price instances. This model of resource sharing sets a price for users of the market. The highest price

in the auction will then win the resource at a price just above that which the next user is willing to pay. With this model there is no guarantee that the workloads will ever be run but can be used to get cheaper resources at times when there is less demand.

4 Conclusions

This document described the advanced tools for controlling liquidity that the Trilogy 2 consortium have been working on. These tools cover aspects of resource sharing, resource monitoring and security (including trust). These tools enable operators to trade their resources on a global Federated Market. They give network operators the ability to route traffic flows to maximise utility, freeing up resources in the resource pool. ConEx closes the information gap between end-users and the network and provides a mechanism to monitor and police the use of the network in a fair and transparent fashion. MPTCP lies at the heart of transport liqidity, but is known to suffer from security issues. SMTCP and MPTLS provide two different mechanisms to leverage MPTCP in a secure fashion. Finally any use of a shared resource pool relies on trust between all parties. The TCP receiver tests described here provide a simple way for senders to test receivers that may be competing for a greter share of resources and the ConEx mechanisms allow operators to be sure that both senders and receivers are being accurate in their reporting of the state of the network.

Bibliography

[1] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitu Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data Center TCP (DCTCP). *Proc. ACM SIGCOMM'10, Computer Communications Review*, 40(4):63–74, October 2010.

[2] M. Allman, V. Paxson, and E. Blanton. TCP Congestion Control. RFC 5681, September 2009.

[3] M. Bagnulo. Threat Analysis for TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6181, March 2011.

[4] Marcelo Bagnulo, Christoph Paasch, Fernando Gont, Olivier Bonaventure, and Costin Raiciu. Analysis of MPTCP residual threats and possible fixes. Internet-Draft draft-ietf-mptcp-attacks-02, IETF Secretariat, July 2014. IESG Evaluation.

[5] S. Bellovin. Security problems in the TCP/IP protocol suite. *SIGCOMM Comput. Commun. Rev.*, 19(2):32–48, April 1989.

[6] Mike Belshe, Roberto Peon, and Martin Thomson. Hypertext Transfer Protocol version 2. Internet-Draft draft-ietf-httpbis-http2-16, IETF Secretariat, November 2014. AD Evaluation.

[7] Andrea Bittau, Dan Boneh, Mike Hamburg, Mark Handley, David Mazieres, and Quinn Slack. Cryptographic protection of TCP Streams (tcpcrypt). Internet-Draft draft-bittau-tcp-crypt-04, IETF Secretariat, February 2014.

[8] Olivier Bonaventure. MPTLS : Making TLS and Multipath TCP stronger together. Internet-Draft draft-bonaventure-mptcp-tls-00, IETF Secretariat, October 2014. I-D Exists.

[9] Olivier Bonaventure, Christoph Paasch, and Gregory Detal. Processing of RST segments by Multipath TCP. Internet-Draft draft-bonaventure-mptcp-rst-00, IETF Secretariat, July 2014. I-D Exists.

[10] B. Briscoe. Tunnelling of Explicit Congestion Notification. RFC 6040, November 2010.

[11] B. Briscoe, R. Woundy, and A. Cooper. Congestion Exposure (ConEx) Concepts and Use Cases. RFC 6789, December 2012.

[12] Bob Briscoe. Inner Space. Presentation in IETF Proceedings, URL: `http://www.ietf.org/proceedings/91/slides/slides-91-tcpm-6.pdf`, November 2014.

[13] Bob Briscoe. Inner Space for TCP Options. Internet-Draft draft-briscoe-tcpm-inner-space-01, IETF Secretariat, October 2014. I-D Exists.

[14] Bob Briscoe. Inner Space for tcpinc. Presentation in IETF Proceedings, URL: `http://www.ietf.org/proceedings/91/slides/slides-91-tcpinc-1.pdf`, November 2014.

[15] Bob Briscoe. Network Performance Isolation in Data Centres using Congestion Policing. Presentation in IETF Proceedings, URL: http://www.ietf.org/proceedings/91/slides/slides-91-dclcrg-1.ppt, November 2014.

[16] Bob Briscoe. Network Performance Isolation using Congestion Policing. Internet-Draft draft-briscoe-conex-policing-01, IETF Secretariat, February 2014.

[17] Bob Briscoe. Reusing the IPv4 Identification Field in Atomic Packets. Internet-Draft draft-briscoe-intarea-ipv4-id-reuse-04, IETF Secretariat, February 2014.

[18] Bob Briscoe. The Echo Cookie TCP Option. Internet-Draft draft-briscoe-tcpm-echo-cookie-00, IETF Secretariat, October 2014. I-D Exists.

[19] Bob Briscoe, Arnaud Jacquet, Toby Moncaster, and Alan Smith. Re-ECN: Adding Accountability for Causing Congestion to TCP/IP. Internet-Draft draft-briscoe-conex-re-ecn-tcp-04, IETF Secretariat, July 2014. I-D Exists.

[20] Bob Briscoe, John Kaippallimalil, and Patricia Thaler. Guidelines for Adding Congestion Notification to Protocols that Encapsulate IP. Internet-Draft draft-ietf-tsvwg-ecn-encap-guidelines-01, IETF Secretariat, November 2014. I-D Exists.

[21] Bob Briscoe, Richard Scheffenegger, and Mirja Kuehlewind. More Accurate ECN Feedback in TCP. Internet-Draft draft-kuehlewind-tcpm-accurate-ecn-03, IETF Secretariat, July 2014. I-D Exists.

[22] Bob Briscoe and Murari Sridharan. Network Performance Isolation in Data Centres using Congestion Policing. Internet-Draft draft-briscoe-conex-data-centre-02, IETF Secretariat, February 2014.

[23] C. Hopps. Analysis of an Equal-Cost Multi-Path Algorithm. *RFC 2992*, 2000.

[24] Stuart Cheshire and Mary Baker. Consistent Overhead Byte Stuffing. *Proc. ACM SIGCOMM'97, Computer Communications Review*, 27(4):209–220, October 1997.

[25] D. Awduche and J. Malcolm. Requirements for Traffic Engineering Over MPLS. *RFC 2702*, 2009.

[26] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246, August 2008.

[27] S. Floyd. Inappropriate TCP Resets Considered Harmful. RFC 3360, August 2002.

[28] A. Ford, C. Raiciu, M. Handley, and O. Bonaventure. TCP Extensions for Multipath Operation with Multiple Addresses. RFC 6824, January 2013.

[29] F. Gont. ICMP Attacks against TCP. RFC 5927, July 2010.

[30] F. Gont and S. Bellovin. Defending against Sequence Number Attacks. RFC 6528, February 2012.

[31] Fernando Gont. Survey of Security Hardening Methods for Transmission Control Protocol (TCP) Implementations. Internet-Draft draft-ietf-tcpm-tcp-security-03, IETF Secretariat, March 2012.

[32] I. Grigorik. *High Performance Browser Networking*. O'Reilly, 2013. `http://chimera.labs.oreilly.com/books/1230000000545`.

[33] P. Gutmann. Encrypt-then-MAC for Transport Layer Security (TLS) and Datagram Transport Layer Security (DTLS). RFC 7366, September 2014.

[34] Hurricane electric IP transit network https://www.he.net.

[35] A. Heffernan. Protection of BGP Sessions via the TCP MD5 Signature Option. RFC 2385, August 1998.

[36] Michio Honda, Yoshifumi Nishida, Costin Raiciu, Adam Greenhalgh, Mark Handley, and Hideyuki Tokuda. Is it Still Possible to Extend TCP?. In *Proc. ACM Internet Measurement Conference (IMC'11)*, pages 181–192, November 2011.

[37] IESG. Transport Services (TAPS) Charter. Working group charter, Internet Engineering Task Force, 2014.

[38] Janardhan Iyengar, Bryan Ford, Dishant Ailawadi, Syed Obaid Amin, Michael Nowlan, Nabin Tiwari, and Jeffrey Wise. Minionan All-Terrain Packet Packhorse to Jump-Start Stalled Internet Transports. In *Proc. Int'l Wkshp on Protocols for Future, Large-scale & Diverse Network Transports*, November 2010.

[39] Frank Kelly and Thomas Voice. Stability of end-to-end algorithms for joint routing and rate control. *Computer Communications Review*, 35(2):5–12, April 2005.

[40] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *SCIENCE*, 220(4598):671–680, 1983.

[41] C. Kreibich, M. Handley, and V. Paxson. Network intrusion detection evasion, traffic normalization, and end-to-end protocol semantics. In *Proc. USENIX Security Symposium*, 2001.

[42] Suresh Krishnan, Mirja Kuehlewind, and Carlos Ucendo. IPv6 Destination Option for ConEx. Internet-Draft draft-ietf-conex-destopt-08, IETF Secretariat, November 2014. I-D Exists.

[43] Mirja Kuehlewind and Richard Scheffenegger. TCP modifications for Congestion Exposure. Internet-Draft draft-ietf-conex-tcp-modifications-06, IETF Secretariat, November 2014. I-D Exists.

[44] Mirja Kuehlewind, Richard Scheffenegger, and Bob Briscoe. Problem Statement and Requirements for a More Accurate ECN Feedback. Internet-Draft draft-ietf-tcpm-accecn-reqs-07, IETF Secretariat, July 2014. Waiting for Writeup.

[45] M. Larsen and F. Gont. Recommendations for Transport-Protocol Port Randomization. RFC 6056, January 2011.

[46] G. Lebovitz and E. Rescorla. Cryptographic Algorithms for the TCP Authentication Option (TCP-AO). RFC 5926, June 2010.

[47] M. Mathis, J. Mahdavi, S. Floyd, and A. Romanow. TCP Selective Acknowledgment Options. RFC 2018, October 1996.

[48] Matt Mathis and Bob Briscoe. Congestion Exposure (ConEx) Concepts, Abstract Mechanism and Requirements. Internet-Draft draft-ietf-conex-abstract-mech-13, IETF Secretariat, October 2014. Approved-announcement to be sent::Revised I-D Needed.

[49] D. McGrew. An Interface and Algorithms for Authenticated Encryption. RFC 5116, January 2008.

[50] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[51] Toby Moncaster, Bob Briscoe, and Arnaud Jacquet. A TCP Test to Allow Senders to Identify Receiver Non-Compliance. Internet-Draft draft-moncaster-tcpm-rcv-cheat-03, IETF Secretariat, July 2014. I-D Exists.

[52] Christoph Paasch and Olivier Bonaventure. Securing the MultiPath TCP handshake with external keys. Internet-Draft draft-paasch-mptcp-ssl-00, IETF Secretariat, October 2012.

[53] Nischal M Piratla, Anura P Jayasumana, and Tarun Banka. On reorder density and its application to characterization of packet reordering. In *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, pages 156–165. IEEE, 2005.

[54] J. Postel. Transmission Control Protocol. RFC 793, September 1981.

[55] Costin Raiciu, Christophe Paasch, Sebastien Barre, Alan Ford, Michio Honda, Fabien Duchene, Olivier Bonaventure, and Mark Handley. How Hard Can It Be? Designing and Implementing a Deployable Multipath TCP. In *Proc. USENIX Symposium on Networked Systems Design and Implementation (NSDI'12)*, April 2012.

[56] A. Ramaiah, R. Stewart, and M. Dalal. Improving TCP's Robustness to Blind In-Window Attacks. RFC 5961, August 2010.

[57] K. Ramakrishnan, S. Floyd, and D. Black. The Addition of Explicit Congestion Notification (ECN) to IP. RFC 3168, September 2001.

[58] E. Rescorla. Keying Material Exporters for Transport Layer Security (TLS). RFC 5705, March 2010.

[59] Stefan Savage, Neal Cardwell, David Wetherall, and Tom Anderson. Tcp congestion control with a misbehaving receiver. *ACM SIGCOMM Computer Communication Review*, 29(5):71–78, 1999.

[60] Yaron Sheffer, Ralph Holz, and Peter Saint-Andre. Summarizing Current Attacks on TLS and DTLS. Internet-Draft draft-sheffer-uta-tls-attacks-00, IETF Secretariat, February 2014. Replaced by draft-ietf-uta-tls-attacks.

[61] S. Shenker. Fundamental design issues for the future internet. *IEEE JSAC*, 13(7):1176–1188, September 1995.

[62] Rob Sherwood, Bobby Bhattacharjee, and Ryan Braud. Misbehaving tcp receivers can cause internet-wide congestion collapse. In *Proceedings of the 12th ACM conference on Computer and communications security*, pages 383–392. ACM, 2005.

[63] J. Touch, A. Mankin, and R. Bonica. The TCP Authentication Option. RFC 5925, June 2010.

[64] David Wagner and Mirja Kuehlewind. Auditing of Congestion Exposure (ConEx) signals. Internet-Draft draft-wagner-conex-audit-01, IETF Secretariat, February 2014.

[65] David P. Wagner. Congestion Policing Queues - A New Approach To Managing Bandwidth Sharing At Bottlenecks. In *Proc. 10th Int'l Conf. on Network and Service Management, CNSM*, pages 1–6, November 2014.

[66] Xinpeng Wei, Lei Zhu, and Lingli Deng. Tunnel Congestion Feedback. Internet-Draft draft-wei-tsvwg-tunnel-congestion-feedback-03, IETF Secretariat, October 2014. I-D Exists.