Using Strongly Typed Networking to Architect for Tussle

Chitra Muthukrishnan[†], Vern Paxson[‡,¶], Mark Allman[‡], Aditya Akella[†]
[†]University of Wisconsin-Madison, Madison, WI, USA
[‡]International Computer Science Institute, Berkeley, CA, USA
[¶]University of California-Berkeley, Berkeley, CA, USA
{chitra,akella}@cs.wisc.edu, {vern,mallman}@icir.org

ABSTRACT

Today's networks discriminate towards or against traffic for a wide range of reasons, and in response end users and their applications increasingly attempt to evade monitoring and control, resulting in an ongoing *tussle* whose roots run deep. In this work we explore an architectural paradigm that can accommodate such tussles in a systematic and transparent fashion. The key idea at the core of our design is *strongly typed networking*: the notion that application messages contain type information that fully describes the content being transferred. Our framework allows for *transparency* between parties which then leads to *dialog* and *choice* for both users and service providers. While in the early stages, we provide a possible framework for directly addressing the tussle between end users and "the network" without resorting to an ever-increasing degree of obfuscation and inference.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: Network Architecture and Design

General Terms

Design

1. INTRODUCTION

Modern networks no longer follow the Internet's original design tenet of remaining agnostic to the nature of the traffic they carry. Today's networks discriminate towards or against traffic for a variety of reasons, including resisting attacks, imposing acceptable-use policies, permitting law enforcement monitoring, preventing users from consuming too many resources, tuning performance, and blocking users from using competing services. Some decry these realities as an unfortunate state of affairs, and many end users desire freer access than such discrimination imposes, but surely these practices reflect a reality that is here to stay: they are rooted in compelling business, economic, and governmental concerns. Accordingly, they constitute a *tussle space* that we must determine how to accommodate rather than resist [4].

The means used to identify traffic for purposes of discrimination and control are often crude and hidden, leaving even expert users with no understanding of why a particular activity fails or behaves differently than expected. In turn, users—and their applications—often try to evade these discrimination mechanisms. These tensions have lead to an arms race resulting in tangled layers of protocols and encodings, and network elements making sometimes heavy-handed policy enforcement decisions that aim to at best "mostly work right". This morass of obfuscated protocols then leads to a system that is both brittle and difficult to debug. We view this state of affairs as untenable, given its woeful trajectory towards evermore entanglement and misdecision as the arms race progresses.

Some would argue that we need to return to a "neutral network" whereby differential treatment of traffic is not permitted, but rather ISPs must treat all traffic equally regardless of type. However, relying on such an approach is problematic for a number of reasons: (i) not all middleboxes make distinctions between traffic, rather some attempt to aid all traffic (e.g., proxy caches), (ii) any legal notion of "network neutrality" is unlikely to be universally binding and (iii) use of middleboxes in private networks will still fall outside the bounds of anything developed for society at large. Therefore we do not view a non-technical network neutrality approach to be universally viable, and hold out little hope that the fundamental tension will go away. Thus, alternatively, we must ask: *How can we architect our way out of this mess?*

In this paper we explore a possible architectural framework for doing so. We root our framework in the pursuit of two core goals. First, the architecture must provide *transparency* to (i) end systems (users) in terms of what restrictions and transformations the network imposes on their traffic, and (ii) network control elements (NEs) in terms of visibility into the semantics of the traffic they carry. Second, the architecture must facilitate *choice* for end systems

Hotnets '10, October 20–21, 2010, Monterey, CA, USA.

with regard to which aspects of their communication the network may inspect or modify, and which aspects remain inviolable and perhaps wholly private. The notion of choice includes the opportunity when possible to select among multiple paths offering differing degrees of network control vs. cost. Such choice is vital in order to allow *tussles* to play out in a productive fashion. (Note that "paths" here refers to routing along either network-layer forwarding or higher-level overlays.)

After briefly describing the problem space's basic constraints in §2, the remainder of this paper develops the elements of our framework. We emphasize that our goal in doing so is to conduct a *thought experiment* regarding the viability of a coherent architecture for end-system/NE tussles. We explicitly do not in this work address issues of performance or overhead, and for purposes of clarity we employ an approach based on the use of XML that clearly in its present form incurs significant inefficiencies. Our focus remains strongly on exploring what sort of conceptual complexity appears required to architect our way out of this mess. We in fact do not get all that far towards addressing the many issues that arise; our overarching goal is instead to spur further exploration by the research community of this difficult problem space. We leave to future efforts the major problems of turning concepts such as those we sketch into an effective, acceptably efficient working system. Finally, when reflecting on the difficult remaining issues, we wish the reader to consider: if we do not develop something like this approach, then where will the future take us?

2. PROBLEM SPACE CONSTRAINTS

Before proceeding we note two fundamental constraints regarding mediation of communication between a sender S and a receiver R that traverses some number of network control elements, $\text{NE}_{1...n}$. While we cannot ultimately avoid these constraints, our framework is designed to address them to the degree possible.

First, any NE can simply thwart communication by not forwarding traffic. While this would seem to give the NEs the upper hand in determining policy, our architectural framework calls for offering end hosts both *transparency* in terms of why particular communication is being blocked and *choice* in terms of the path to use (and, hence, the policies to accept).

Second, S and R can collude to transmit disallowed messages. Our framework explicitly includes the concept of verification of messages to help combat this problem. While this may curtail casual establishment of covert channels and may limit their effective bandwidth, in the limit it is essentially impossible to prevent steganography and covert channels from being established between willing partners. We note this limit is present in today's traffic, as well. Furthermore, pushing disallowed traffic towards steganography increases the costs on the malicious actors, and is likely the best ultimate outcome we can hope for.

```
<http,reply>
  <status> 200 OK </status>
  <set-cookie>abc</set-cookie>
  <content,exe>
  <data>...</data>
  </exe,content>
</reply,http>
```

Figure 1: Unencrypted HTTP message

3. ARCHITECTURAL COMPONENTS

We next turn our attention to the high-level concepts of the envisioned architectural framework. These will be discussed in more detail in subsequent sections.

Typing: At the heart of our architectural approach is a notion we term *strongly typed networking*. In programming languages, the notion of typing provides semantic context for what are otherwise simply bits in memory cells. Similarly, for networking we envision ubiquitous use of typing as transforming the current strongly layered model: rather than a transport protocol simply serving to transfer opaque binary payloads, what it carries includes exposed type information that governs the semantics of how the receiver of the data will interpret it. These types are both (i) *extensive*, ranging from atomic values (IP addresses and TCP ports) to higher-level constructs (URLs, command names, status codes) to aggregated objects (MIME, HTTP request/replies, SMTP dialog) and (ii) *exhaustive*, i.e., everything is typed without exception. Typing addresses the *transparency* goal described in §1. Figure 1 shows an an HTTP message marked up in XML and annotated with type information.

Dialog: The goals of transparency and choice and the notion of typing facilitate dialog: the ability for end systems to engage with NEs in order to explicitly agree upon what types of visibility and control may be applied to the given traffic, and what facets remain out of bounds (unexamined or unperturbed by the network).

A dialog begins with the end-systems selecting the degree to which they are willing to expose an application's semantics to NEs within the network. These semantics might include names and types of data items to be transferred, identifiers to associate with activity, explicit indications of what portions of the session will be hidden via encryption, perhaps a pledge to use only a recognized subset of the application protocol's full functionality,[1] and so on. NEs inspect the end system's description of these semantics to determine whether the information provides the NE with enough context to correctly carry out its function. If the NE finds the information provided by the application insufficient, it can *reject* the session. It does so in an explicit fashion, by informing the application what additional information/permissions it would require to accept the session.

We emphasize that the nature of such a dialog already occurs today, but much more crudely. To first order, today's version is that the application advertises what it is doing

[1] E.g., avoiding the CONNECT method in an HTTP session.

via a TCP or UDP port number, and perhaps via the initial payloads it sends. If the network dislikes the offered information, it declines to forward the traffic; perhaps completing the "dialog" by returning an ICMP or TCP RST packet [5]. As sketched in §2 this basic balance of power will not change—the network will *always* have the upper hand in that, if dissatisfied, it can block communication. However, if visibility continues to decline, increasingly operators will find themselves forced to impose draconian restrictions on allowed traffic that will exacerbate collateral damage.

Choice: In response to a rejected session the application decides whether it is willing to provide the additional information. If not, the application can attempt to seek an alternate path that imposes less burdensome requirements, if available. If no alternative path is available and the application (user) is unwilling to yield the required information or control, then communication will not occur—but with the cause of the failure in full view.

Verification: By the notion of *strongly* typed networking, we mean that NEs can have confidence that the payloads they analyze will indeed be interpreted according to their purported types. As discussed in §2, ultimate confidence cannot be determined in the limit. However, our architectural framework includes a verification mechanism in the form of *attesters* that can verify a payload item's type. These could be located in the end host (as trusted modules verified by a TPM) or trusted third-party services that are not part of the communication.

4. INITIAL DESIGN

We now turn our attention from architectural concepts to a sketch of how the framework may work in practice (in high-level terms). We stress—as noted in §1—that our goal in this paper is to simply explore the type of architecture required to address the current tensions in the network and not to present a finished technical design.

4.1 User Choice Via Encryption

A key aspect of our framework is to provide end-hosts with choices with regard to various aspects of their communication that NEs can inspect or modify versus those that are private. In our framework, we use selective encryption as a way of enabling this choice.

Specifically, the sender provides a given decryption key to a given NE if it is willing to allow that NE to inspect the corresponding element. The sender then encrypts different portions of its message with the appropriate keys. We also include integrity checks tied to keys; these may be the same as the encryption keys, or different. Possession of an integrity key allows an NE to modify the corresponding element; possession of only the encryption key allows it to inspect, but not to modify (as the integrity check would then fail at the receiver).

We now illustrate the important steps in selective encryption, and highlight underlying design issues, using simple examples.

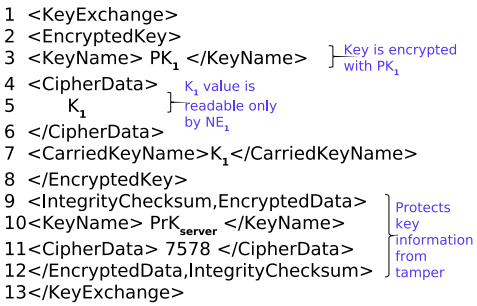

Figure 2: Exchange of symmetric key K_1 with NE_1

Key Establishment. After the sender chooses a path, it exchanges the requisite keys with NEs along the path by sending separate key exchange messages, one per NE. Using a separate key exchange message, the recipient is also provided with a master set of keys that include the keys for NEs and a recipient-specific key. We note that keys once established could be applied to all flows between a sender and a recipient as long as the path between them remains stable (we discuss path stability further in §6).

Figure 2 shows an example key exchange message. The sender uses the public key of NE_1 (PK_1) to encrypt the NE's symmetric key K_1 (lines 2-8). The message also includes "key names" (lines 3 and 10), used by the sender to indicate parts of its messages that an NE could access. Thus, in this example whomever owns the key names PK_1 can decrypt and read CipherData containing the encrypted value of K_1. Note that the fields in this key exchange message are protected from tampering by on-path NEs with an integrity checksum computed using the private key of the server, (PrK_{server}) (lines 9-12).

Messages. Once keys are established, a sender encrypts each message with the appropriate keys. In Figure 3, we illustrate this for an HTTP reply message exchanged between the server and the client for the example shown in Figure 4. (For ease of exposition, the example described is simplistic and not as exhaustive as it would necessarily be.)

In this message, lines 3 through 21 are encrypted using the symmetry key of NE_1 (key K_1). However, a portion of the message, between lines 6 and 13, is only visible to the recipient and not to NE_1. This is encrypted using the recipient's public key (PK_{client}). In effect, the NE only has visibility into lines 3 through 5 and 14 through 21 of the reply message, i.e. into the executable component of the message. The checksum in line 11, protected using the server's private key, facilitates detection of any unauthorized modifications by the NE to the earlier portion of the message.

Access to the context. In many cases, the specific action taken by NEs on data of certain types depends for its correctness (and to thwart evasion) on the higher-level context within which the data type occurs. For example, an NE might have different heuristics to decide if an executable is harmful or not depending on whether it was sent as a HTTP

```
1  <http,EncryptedData>
2  <KeyName>K1</KeyName>
3  <CipherData,reply>
4     <EncryptedData>                                  Readable by NE1 & client
5     <KeyName>PKclient</KeyName>
6     <CipherData>
7        <status> 200 OK </status>                     Readable only by client
8        <set-cookie>...</set-cookie>
9        <MessageIntegrity,EncryptedData>              Integrity protects status & cookie fields
10       <KeyName> PRKserver </KeyName>
11       <CipherData> 1456 </CipherData>
12       </EncryptedData,MessageIntegrity>
13    </CipherData,EncryptedData>
14    <content,exe>                                    Readable by NE1 & client
15    <data> </data>
16    <MessageIntegrity,EncryptedData>                 Updated by NE1 after deleting <data> field
17    <KeyName>K1</KeyName>
18    <CipherData>2654</CipherData>
19    </EncryptedData,MessageIntegrity>
20    </exe,content>
21 </reply,CipherData>
22 </EncryptedData,http>
```

Figure 3: Encrypted HTTP reply via NE$_1$

reply or as an email attachment. If just the executable data is exposed to the NE, it would lack knowledge about this high-level context needed to operate on it. To facilitate using context in policies, whenever an NE requires access to a field, our framework also provides the NE access to the (potentially nested) parent fields. For instance, within an HTTP reply an NE given access to some file would also be given access to the fact that it was coming as part of a reply being sent over HTTP (see example reply in §4.2).

Permission to modify. Note that in this example, the NE also possesses an integrity key (line 17), which in this case is the same as the symmetric key K_1. Hence, the sender has permitted the NE to modify the executable component. If the NE does modify the executable content, it updates the integrity checksum in line 18 so that the recipient can validate the modification.

When NEs have the ability to modify message fields, then the sender and the receiver may end up with different notions of what content actually reached the receiver. It is important in such situations that the sender synchronize with the receiver to understand what data eventually arrived. The synchronization could be handled at the application level, using, for instance, queries about the presence of a certain type of field, or the hash or size of content elements.

4.2 Communication Overview

Figure 4 shows a small sample network and labels the various facets of instantiating communication.

1. *Route Discovery*: A sender that wishes to communicate with a destination first must discover the available routes, which could be network-level, overlay or multi-homed paths. This process also yields the public keys of all the involved network elements. While route discovery is out of scope for this paper we believe there are methods that can

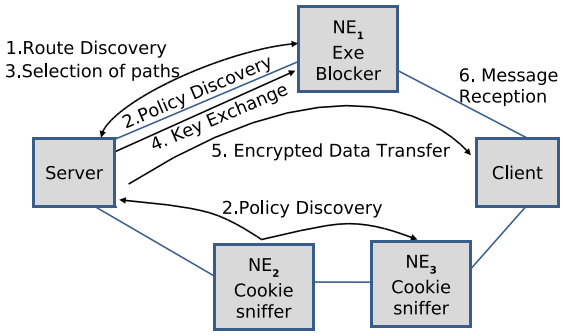

Figure 4: Sample HTTP communication (chooses path via NE$_1$)

largely support our framework (e.g., NIRA [12] and Pathlet routing [8]).

2. *Policy Discovery*: After discovering candidate routes a sender must discover the policies of the NEs along these paths. To do so the sender transmits a query that includes the kinds of messages that will be sent, the fields the application wishes to keep private and the fields the application is willing to expose, as well as annotations indicating which fields the application will allow to be modified or deleted by the NEs. Each NE then inspects the query and decides whether the given restrictions are acceptable and informs the sender of desired changes. For instance, in an HTTP transaction the application may be willing to expose cookies while one of the NEs may require access to executable content. The negotiation can span multiple rounds as necessary. The policies are also encrypted with the appropriate keys to ensure the policies are only exposed to the end points and particular NEs implementing the given policy.

An alternate style would operate instead in an implicit fashion, with no separate discovery phase. Here, the sender simply goes ahead and transmits its messages protected however it wishes. Intervening NEs that require greater visibility or control over the communication drop the messages and return control notifications to the sender highlighting what would have to change in the protections the sender has employed, similar to IP's MTU Discovery mechanism. This approach has the advantage of simplicity and robustness in the presence of routing changes. Further, it is useful for caching and re-using policies that are not likely to have changed along a particular path between two end hosts—thus avoiding the time- and resource-consuming discovery phase. However, such implicit discovery could also lead to inefficiencies that arise when an end host discovers their operations are not permitted mid-stream (e.g., because the end host then decides to change paths and so effort has been wasted when compared to up-front negotiation).

3. *Selection of Paths*: Once the source determines the policies in place on each path, it chooses a path based on the detected policies and any of the other path characteristics available (e.g., delay and loss).

4. *Key Exchange*: Once a path is chosen the sender exchanges symmetric keys with the NEs that have played a part (passive or active) in the transaction. More than one key per NE may be necessary as the NE may have access to inspect some fields, while being allowed to modify others. Further, each key is shared with the receiver as the receiver will ultimately be responsible for validating and decrypting the message for the application.

5. *Encrypted Data Transfer*: The sender then encrypts messages according to the negotiated policy and source routes them along the chosen path. The NEs decrypt and inspect or modify the portions of the messages that have been so negotiated. Whenever an NE modifies a field it appends an integrity checksum to prevent other NEs from rewriting that field. These checksums are also used by the receiver to identify the NE that modified a field.

6. *Message Reception*: The receiver decodes the message fields with the corresponding keys.

5. VERIFICATION

Verification is fundamental to any system that has competing stakeholders, and therefore is an important aspect of our design: Our framework not only helps parties agree on policies and communicate according to them, but also provides confidence to stakeholders that either communication did proceed as agreed to, or there is a way to detect that there was a violation.

First, as discussed in §4, applications ensure NEs read and modify only authorized fields within a message by appropriate use of encryption. Similarly, NEs can be certain that they have access to the portions of a message that the negotiated policy requires. In the encryption mechanism described in §4.1, the sender could hide a field that an NE wishes to inspect by encrypting it with a key the NE does not possess, thereby evading the NE's policies. The NE in turn could drop the traffic as being outside the negotiated policy.

As discussed in §2, the sender and receiver can collude to evade the NEs' policies. While in the limit it is nearly impossible to thwart such efforts while still providing a communications channel, there are steps an NE can take to mitigate the risk. In particular we describe two kinds of *attesters* to aid NEs in enforcing policy.

End-Host Attesters: The first category of attester is run on the end hosts themselves. In this case, a known attester is bootstrapped and validated by the host's TPM module. The attester appends attestations of type information to the messages for the NEs to consult.

Third-Party Attesters: These entities are independent of the communication flow and can be used to verify a particular message is of the claimed type. As part of an NE's policy an end host may have to route traffic through such an attester and give the attester access to the given message. This may be more palatable to the end hosts (and users) than providing an ISP—which naturally already has a broad view of their traffic—access to the messages. A neutral third-party can give the NE confidence in the message type (although, obviously not a complete guarantee).

Another technique that an NE can employ is a form of "normalization" to try to remove any non-crucial parts of a transmission that may represent a covert channel. For instance, scrubbing comments out of HTML markup or converting simple images from one common format supported by all browsers (e.g., jpeg) to another (e.g., png). This requires the NE to negotiate permission to change the content during the dialog phase. But, it could be a useful technique in mitigating the covert channel problem.

6. ADDITIONAL CONSIDERATIONS

In this section we briefly touch on possible extensions to our framework.

Routing Changes. One of the reasons the current Internet architecture calls for a "dumb middle" is so that routing changes within the network—e.g., caused by a network failure—do not hinder communication. Therefore, one practical issue that our framework must address concerns routing changes. As the network alters its forwarding to route around an outage, traffic from active flows will begin to appear at NEs that have no prior context for the communication and thus may lack sufficient information to soundly analyze the flow beginning mid-stream. Such NEs may therefore not forward the traffic.

To avoid the brittleness of routing changes, we might consider developing notions of *type-safe handoffs*, i.e., mechanisms by which an existing flow can present to a new NE information regarding what has previously transpired on the flow along its old route, such that the new NE can determine whether it has sufficient information to safely accept the flow. Perhaps we could realize such a mechanism by NEs periodically issuing certificates documenting the progress of ongoing flows. In the presence of a routing change, the end system would roll back communication to the point covered by the most recent set of certificates and then proceed forward from there, after priming the NEs along the new path with the information from the certificates. Alternatively, if we used an implicit-style approach for policy negotiation as briefly sketched above, that might suffice to accommodate routing changes too.

Transport Properties. A different issue concerns the fact that NEs might base their control decisions not only on an understanding about the semantics of communication, but also its transport properties, such as its rate of transfer, fanout, or aggregate volume (across a number of flows). Our initial framework does not provide a ready means for NEs to express such constraints (nor for end systems to hide such information if they choose), because such properties are dynamic, structural, and contextual, rather than rooted in the *content* of communication. We view accommodating transparency and dialog for these sorts of NE decision-criteria as an important problem to address when architecting for tussle, and while our type-based approach is not an ideal match

the foundation provided by the system of exchanging structured information with NEs may be useful in grappling with these issues in the future.

Cooperating Actors. While we have focused in this work heavily on the issues that arise when users and NEs view one another as potential adversaries, our principles of facilitating transparent dialog and imposing a discipline of strongly typed communication can also open up new possibilities among cooperating entities. Unambiguous knowledge of the semantics of messages can enable NEs to perform operations on behalf of end systems in order to reap performance gains or realize network-based services. As a simple example, consider a transcoding NE that knows how to rewrite images in different formats to smaller representations suitable for display on mobile devices or to conserve bandwidth over low-speed links. With strongly typed networking, the NE can correctly locate and transform *any* instance of a rewritable image, whether located in a Web page, a chat session, a file transfer, or some new application with unknown broader semantics. End systems transferring items they do not wish transformed can simply deny the NE the ability to modify the item.

7. RELATED WORK

Some of the key design goals underlying our framework have been previously explored in other contexts. Several architectural components for interposing middleboxes atop network communication in a ground-up, rather than an add-on, fashion have been explored (e.g., [11, 9]). Additionally, several recent studies have also explored how to improve the transparency of network communication. For instance, X-Trace [7] and Packet Obituaries [1] call for the network to inform hosts about certain properties, while labeling [3] calls for end-hosts to inform network intermediaries about properties of the content they are transferring. Finally, the literature is filled with one-off negotiation mechanisms for various purposes (e.g., option negotiation in TCP, content type negotiation in HTTP [6], reconciling objectives of peer ISPs [10], etc.). Our work differs from these previous efforts in two key respects. First, while previous work has considered traversal, transparency and negotiation in isolation and for specific purposes, our work combines these themes in a general and coherent fashion. Second, much of the prior work assumes a cooperative setting whereby end hosts and network elements are working together to facilitate communication. In contrast, we architect for an adversarial setting wherein end-hosts desire greater freedom and privacy in communication, while network entities desire a high degree of visibility and control.

In addition, labeling has previously been suggested as a way of informing NEs about communication properties (e.g., generically in [3] and for specific tasks such as the DiffServ code points [2]). Our architectural framework is similar to these previous efforts in that there is explicit meta-information shared between components of the system. Our contribution is to identify types as a specific instance of labeling that is both (i) more general than task-specific labels for particular applications and (ii) more specific than an arbitrary labeling scheme, so that general policies and verification across protocols and services become tractable. We view this intermediary point in the design space as providing two significant benefits: *enforceability*, thus being suitable for adversarial settings, and *semantic clarity* (i.e., an NE knows how to correctly process a given label), advantageous in both adversarial and non-adversarial settings.

8. SUMMARY

In this paper we have outlined an architectural framework that uses a notion of strongly typed protocols to design for the ongoing tussle between users and networks. Rather than accept the evolutionary and ad-hoc approach of users increasing the layering and obfuscation of their communications and operators trying to cope by designing crude methods for dealing with traffic that appears to violate policies, we step back and pose the question "can we architect our way out of this mess?" Our initial exploration is merely meant to provide a basis for thinking about how to construct a network dialog to allow various actors in the system the ability to reason about and synchronize policies for a particular communication stream. The system discussed in this paper is a conceptual framework that is far from a concrete and efficient reality in many ways. While we do not wish to downplay the importance of such issues, our goal is first to grapple with the overall complexity of devising an architecture to directly deal with the real and frequent tussles that arise in networks everyday.

Acknowledgments — This work has benefited from discussions with Somesh Jha, Kevin Fall, Eric Rescorla and Scott Shenker. This work is funded in part by NSF grants CNS-0626889, CNS-0722035, CNS-0831535, CNS-0831780 and CNS-0905134.

9. REFERENCES

[1] K. Argyraki, P. Maniatis, D. Cheriton, and S. Shenker. Providing packet obituaries. In *ACM SIGCOMM HotNets*, 2004.
[2] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. An Architecture for Differentiated Service, Dec. 1998. RFC 2475.
[3] M. S. Blumenthal and D. D. Clark. Rethinking the design of the internet: the end-to-end arguments vs. the brave new world. *ACM Trans. Internet Technol.*, 1(1):70–109, 2001.
[4] D. D. Clark, J. Wroclawski, K. R. Sollins, and R. Braden. Tussle in cyberspace: defi ning tomorrow's internet. In *ACM SIGCOMM*, 2002.
[5] M. Dischinger, A. Mislove, A. Haeberlen, and K. P. Gummadi. Detecting BitTorrent Blocking. In *Internet Measurement Conference*, 2008.
[6] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1, Jan. 1997. RFC 2068.
[7] R. Fonseca, G. Porter, R. H. Katz, S. Shenker, and I. Stoica. X-trace: A pervasive network tracing framework. In *NSDI*, 2007.
[8] P. B. Godfrey, I. Ganichev, S. Shenker, and I. Stoica. Pathlet Routing. In *ACM SIGCOMM*, Aug. 2009.
[9] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. *SIGCOMM Comput. Commun. Rev.*, 37(4):181–192, 2007.
[10] R. Mahajan, D. Wetherall, and T. Anderson. Negotiation-based routing between neighboring ISPs. In *Proc. NSDI'05*.
[11] M. Walfi sh, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *Proc. USENIX OSDI*, San Francisco, CA, December 2004.
[12] X. Yang. NIRA: a new Internet routing architecture. In *Proc. FDNA '03*. ACM.