The expressive power of the Internet design

David D. Clark

Version 5.0 of April 27, 2009[1]

Introduction

The present Internet is not defined in terms of its semantics, at least at the packet level. The loose packet carriage model of "what comes out is what went in" is intentionally almost semantics-free. The packets just carry bytes. Packet boundaries can have some limited semantics, but not much. The original design presumed some constraints on the semantics of packet headers, such as global addresses, but the progress of time has violated these and the Internet keeps working. TCP does impose some modest semantic constraints, but of course TCP is optional, and not a mandatory part of the architecture.

What defines the Internet, and the range of behavior that is available in the Internet, is the expressive power of the packet header, which has more to do with its format than any semantics. Most fields (e.g. packet length) are unremarkable, some (like the TOS bits) have been redefined several times in the history of the Internet, some (like the options) have atrophied, and some (most obviously the IP addresses) have had a most interesting history in which the only constants are that they are 32 bit fields, that whatever value they have at each end must remain constant for the life of a TCP connection (because of the pseudo-header) and that at any locale in the network, they must provide the basis for some router action. They can be rewritten (as in NAT), turned into logical addresses (as in multicast or anycast), and they can be microcoded in a number of ways to capture address hierarchy (net/rest, A/B/C, CIDR). All that really matters is that they are 32 bits long, and that at any point, they must have at least local meaning to a forwarding process.

The evolution in thinking with respect to IP addresses is worth some study. The initial idea that addresses were drawn from a single global address space and mapped uniquely to physical ports on physical machines turned out not to be a necessary constraint, but just a simple model to get started. We were initially fearful that if we deviated from this definition, the coherence of the network would fall apart, and we would not be able to ensure that the Internet was correctly connected, or debug it when it was not. Indeed, these fears are somewhat real, and it is possible today to "mess with" addresses in such a way that things stop working. But mostly, the Internet continues to work, even with NAT boxes, VPNs and private address spaces, because the consequences of messing with

[1] The research reported here and the preparation of this document was supported by the Office of Naval Research under contract N00014-08-1-0898, and by the National Science Foundation under agreement 0836555. The opinions contained are those of the author, and do not reflect the opinions of the supporting agencies.

addresses are restricted to regions within which there is agreement to assign a common meaning to those addresses. Those self-consistent regions need not be global.

These regions can be created within the Internet in a number of ways. Routers may be coupled by routing protocols running in the background, or by manual configuration. To the extent that we have a region of the network in which we deploy a consistent set of routing protocols, if we were to rewrite the address of a packet in the middle of this region there is no telling in general what might happen. We assumed that to prevent packets from looping, we had to add protection against an inconsistent routing computation (the TTL). We assumed, as a simplifying assumption, that we did not need to reason about what would happen if the address changed. But this line of reasoning (and indeed it is a useful and simple line of reasoning) only suggests that the scope of an unchanging address should more or less match the scope of a dynamic routing protocol. And indeed, even this need not be a rigid constraint.

In the limit, each "region" could just be two routers, the sender and the receiver for each hop along the path of the packet. (This would somewhat resemble a scheme based on label rewriting.) Regions this small would be hard to manage without some sort of overarching framework for state management (and would have other drawbacks as I discuss later), but a single global region—the starting point for the Internet design—has also proven to have complexities. In practice, the operational Internet has gravitated to regions that represent some sort of rough balance among the issues that arise from big and small regions.

The observation I make here, of course, is that the packet header is the fixed point in this process. The limits on this sort of evolution or "gravitation" are what can be expressed in the header, not any assertions about the semantics of addresses.

Per-hop behaviors

We can generalize from this discussion of addressing and ask more abstractly about the local behavior of routers and the resulting overall function. In fact, the network is built up of somewhat independent routers. What we care about is that the local behaviors (the "per-hop behaviors", or PHBs) of these routers compose to achieve the desired results end-to-end. If the packets get delivered (which is really the only thing that defines today's properly operating Internet, except in the context of defense against attack), then it does not matter how the PHBs were coordinated, if at all. If the packets do *not* get delivered, then debugging may be more or less a nightmare, depending on the tools for coordination and analysis, but this is a separate issue (see below).

Today, a router has a rather simple set of behaviors. Ignoring QOS and source-routes for the moment, a router either picks (one or more) outgoing paths on which to forward a packet, or drops it. The router can have as much state as inventive people define for it—static and dynamic forwarding tables, complex routing protocols, and static tables that define unacceptable addresses (e. g. so-called Martian and bogon packets). The router can also rewrite the packet at will, subject to the pseudo-header constraint. But looking to the future, not all elements in the network need be "routers". Elements, once they receive a packet, can perform any PHB that does not cause the end-to-end behavior to fail. So when we consider PHBs as the building block of network function, we should be careful not to limit ourselves to a model where the only PHBs are "routing" or "forwarding".

Tussle

One of the distinctive features of networks and distributed systems is that they are composed of actors whose interests are not aligned. Sometimes one of the actors is a clear "bad guy": e.g. someone wants to infiltrate my computer against my wishes. This tension leads to devices such as firewalls, which are an example of a PHB that is not simple routing, but forwarding or dropping based on the content of the packet. Firewalls are an attempt by the receiver to overrule the intentions of the sender: a PHB that the receiver wants executed on the packet, but the sender does not.

Sometimes the issues are not black and white, but more nuanced I want a private conversation, law enforcement wants to be able to intercept any conversation with proper authorization. I want to send a file privately, copyrights holders want to detect if I am serving up infringing material. To the extent these tussles are played out "in the net" (as opposed to end the end-nodes or the courts), they will be balanced through the relative powers of the different actors to exploit the expressive power of the network. So our discussion of expressive power, and the tools that implement it, will be strongly shaped by the reality of tussle.

What can (and cannot) we do?

Is there a formalism?

Computer scientists are accustomed to thinking about the implications of semantics: what are the limitations of some semantic construct. We are less accustomed (and less equipped with tools) to think about the expressive power of a packet header—what functions are consistent with some format. It is sort of like asking what ideas can be expressed in sentences of the form "subject, verb, object". The question seems ill-defined and unbounded. Even harder is to catalog what *cannot* be expressed. But this question is the one that actually captures the limits of what the Internet can and cannot do. So we should try to think about how to think about it.

If (in general) a network element can be programmed to do "anything" as its PHB, then the resulting overall function is the result of the execution of these PHBs in some order, where the order is defined by the routing of the packet among these devices. Of course, since the devices themselves can define the routing, the resulting expressive power (the computational power, if you will) is presumably rather complex.

This view of packet processing has not been seriously explored (with the exception of some of the Active Network research), because in the Internet of today, the overall function we want to achieve is very simple—the delivery of the packet. If that is the desired overall function, there is not much demand for the complex concatenation of arbitrary PHBs within the network. But as we think about wanting to do more complex things as a packet moves from source to destination (many having to do with security), the range of interesting PHBs will grow. So it is worth some consideration of what factors define or limit the expressive power of a network.

A framework for PHB execution

In this section, I pose a three-dimensional model that describes the landscape of PHB execution: delivery, alignment of interests and parameterization.

Delivery

The first dimension of the model is to ask *why* or *how* the packet arrives at the element that implements the PHB. There is a simple model that covers most of the circumstances that cause a packet to arrive at an element in the network. I will propose three cases, which I call *intentional, contingent,* and *topological.*

Intentional: In this case, the packet arrives at the element because it was specifically sent there. For example, with source routes, the route is a series of addresses, each of which directs the packet to the next such router. As another example, a packet arrives at a NAT box because it was intentionally sent there.

Contingent: In this case, the packet may or may not arrive at a given element, but if it happens to arrive, then the PHB will occur. This is the basic mode of datagram routing. There are no pre-established paths from source to destination (which would be examples of intentional delivery). Each router computes routes to all known destinations, and if a packet happens to show up, the router forwards it.

Topological: In this case, there is nothing in the packet that causes it to arrive at a particular device, but instead the topology of the network (physical or logical) is constrained to insure that the packet does arrive there. Firewalls are a good example of topological delivery. The sender (assuming he is malicious) has no interest in intentionally sending his attack packet to a firewall. He would prefer to route around it if he could. The receiver wants some assurance that the firewall will be in the path. The receiver will normally not be satisfied with contingent protection. So the remaining tool available is to constrain the connectivity or routing graph so that the only path (or paths) to the receiver pass through the firewall.

Alignment of interests

The second dimension of the model is to capture the relationship between the sender of the packet and the owner of the element that implements the PHB. This dimension directly captures the nature of tussle. Again, I will propose three cases: *aligned, adverse* and *coerced.*

Aligned: In this case, the interests of the sender and the element match. Simple routing, multicast, etc., usually falls in this obvious class. The sender sent the packet, the router forwards it, and this is what both parties expected.

Adverse: In this case, the PHB performs a function that the sender does not want. A firewall is a good example here, as would be other sorts of content filtering, deep packet inspection, logging and so on.

Coerced: This can be seen as a special case of an *adverse* relationship in which the sender is required to conform with the requirements of some sort of PHB (or even agree to intentionally send the packet to the location of the PHB), even though the interests of the sender and the owner of the PHB are adverse. In this case, we can expect the sender to cheat or lie (in terms of what values are in the packet) if it is possible.

In passing, it is worth asking why I looked at the relation between the sender and the PHB, but not the relation of the receiver and the PHB. This is because sending packets is asymmetric in an obvious way—only the sender can directly control the sending of the packet. There are special cases that can be explored where the receiver can exercise some control (e.g. in cases of multi-homing or anycast, which I consider later). If the interests of the sender and receiver are aligned, then if there is an adverse PHB in the path, it must be there because of some third party (e.g an ISP or a government authority, etc.). If the interests of the sender and the receiver are adverse (e.g. the sender is an attacker), then the most common case is that the PHB in question is aligned with the receiver (e.g. a firewall).

Parameterization

The third dimension of the model is that the packet triggers the execution of a PHB, and thus the data in the packet is in some sense the *input* to that PHB, like arguments to a subroutine. The values in the packet are the input parameters to the PHB, and if the packet is modified, this is similar to the rewriting of variables in the invocation of a subroutine. (In other words, to use the vocabulary of programming language, the packet invokes the PHB by reference rather than by value.) The element that executes the PHB can have lots of persistent state (which can be modified as a result of the PHB), and can have distributed or "more global" state if suitable signaling and control protocols are devised.

In this context, I will again offer two cases, although these more define ends of a spectrum than distinct modes: *explicit* and *implicit.*

Explicit: While the PHB can in principle look at any data fields in the packet, in common cases there will be specific fields set aside in the header as input to specific PHBs. This is the common case for packet forwarding: since packet forwarding is the basic operation of neworking, there is an explicit address field used as input to the forwarding lookup. The Internet (sometimes) supports QoS, so there is an explicit field in the packet that is the input argument to the QoS algorithm.

Implicit: In other cases, there is no specific field used as input to the PHB: the PHB looks at fields intended for other purposes. Firewalls block packets based on port numbers, some ISPs assign QoS based on port numbers, packets are sometimes routed based on port numbers (e.g. when Web queries are deflected to a cache or an outgoing

SMTP connection is deflected to a local mail server.) If the PHBs have state, they can also base their actions on implicit information such as the arrival rate of packets.

This model suggests that there is some rough analogy between the expressive power of a network and a programming language of some sort, where the "computation" is a series of subroutine executions, driven by the input parameters carried by the packet, and where the order of execution is defined by the routing protocols, together with the expressive power of the packet to carry the addresses that drive the forwarding. Of course, the addition of tussle and nodes that are hostile in intent with respect to the sender adds a twist that one does not find in programming languages, and in fact this "twist" may be one of the most important aspects of what the network actually "computes". So the power of an analogy to a programming language remains to be explored.

Pruning the space of options—tussle analysis.

What I just described is a 3x3x2 design space. But in fact it is less complex than that. The starting dimension that helps to sort out this space is the one of alignment of interests.

Aligned: If the sender *wants* the PHB to be executed, then *intentional* delivery and *explicit* arguments make sense. *Contingent* delivery may be suitable in some cases (e.g. the basic forwarding function), but *explicit* arguments (e.g. the address field) still make sense.

Adverse: If the sender does *not* want the PHB to be executed, then he cannot be expected to provide any explicit arguments to the PHB, so the design must be based on *implicit* approaches. Nor can the PHB count on *intentional* delivery, so *contingent* or *topological* delivery must be used.

Coerced: In this case, the PHB may be able to demand *explicit* arguments, but the sender can be expected to lie if possible. So there must be some sort of other (usually implicit) inputs that can serve to police the explicit inputs.

So the common design options will be:

- Aligned:
  - Explicit
    - Intentional or contingent
  - Implicit (less common)
    - Contingent or topological
- Adverse:
  - Implicit
    - Contingent or topological
- Coerced:

Explicit
    Topological
Implicit
    Contingent or topological

Implementation considerations

Implicit arguments can be expensive. In the worst case (deep packet inspection), the PHB may process the entire contents of the packet as input to its operation. Clearly, this is not as efficient as a pre-designed action where the PHB picks a preset field (e.g. an address field) and uses this for a table lookup. So implicit arguments must be used sparingly.

Prior work

Role-based Architecture

The proposal for a Role-based Architecture (RBA) (Braden, Faber et al. 2003) is perhaps the closest example of an architecture that captures the idea of general PHBs and the expressive power of a packet header. In this proposal, PHBs are called *roles,* and the data that is input to each node is called the Role-specific Header, or RSH. The packet header is described as a *heap* of RSH's. RSH's are an example of *explicit* arguments. The proposal discusses both intentional and contingent routing, where the intentional addressing would be based either on the ID of a role, or the ID of a role at a specific node. The paper does not delve into tussle to any degree, or work through the case of roles that are adverse to the interest of the sender, so there is not much attention to implicit arguments or to topological delivery. However, the idea of a network as a sequence of computations performed on a packet based on explicit input arguments is the core concept of role-based architecture.

Case studies: what has and has not worked

Mobility

Host (and network) mobility is a well-known and well-studied problem. In today's Internet, dealing with mobility is complicated by the fact the IP address is used both for forwarding and for end-node identity. Separating these two concepts into two different data fields in the packet will allow the location field (e.g. that data that is input to the forwarding RBA) to be changed as the mobile host moves. This division does not solve the two resulting problems: how to keep the location information up to date, and how to make sure the identity information is not forged. Linking identity to location provided a weak form of security: if two machines have successfully exchanged packets, the location is sufficiently unforgable that it can stand as a weak identifier. But by separating the two problems, they can each be resolved separately, and managed differently in different situations as circumstances require.

NAT boxes

NAT boxes are a wonderful example of how one can disrupt two of the most fundamental assumptions of the original Internet and still have enough functions mostly work that we accept the compromise. The assumptions of the original Internet were that there was a single, global address space, and there was no per-flow state in forwarding elements. NAT boxes, of course, have per-flow state, and lacking a protocol to set up and maintain soft state, they depend on a "trick": they use the first outgoing packet to set up the state, which then persists to allow incoming packets to be forwarded.

This "trick" does not allow state to be set up for services that are "behind" the firewall, so that the first packet can be an incoming one. To allow the forwarding of incoming packets, today one must resort to manual setup of the state. However, it is easy to imagine that if there were some scheme to allow end-nodes to set up state along the path to them, it would be straight-forward to manage the full range of forwarding tasks done by NAT boxes.

Firewalls

Firewalls, as I described above, are an example of a PHB that is hostile to the interests of the sender (the potential attacker) and thus must depend on implicit information. The firewall has the poorly-defined task of trying to distinguish "good" from "bad" behavior, based on whatever hints can be gleaned from the packets. Normally, all a firewall can do today is a very crude set of discriminations, blocking traffic on certain well-known ports and perhaps certain addresses. The roughness of the discrimination is not necessarily a consequence of the details of the current Internet, but perhaps the intrinsic limits of making subtle discriminations based only on implicit fields in the packets.

This outcome is not necessarily a bad thing. Sometimes users want the blocking to succeed (when they are being attacked) and sometimes they want it to fail (when some third party such as a conservative government is trying to block their access to other sites on the Internet). If we decide to make the job of the firewall easier, we should consider whose interests we have served.

Tunnels

Tunnels, or packet encapsulation, is often thought of as a way to control the routing of a packet, but more generally it is a way to interpose an explicit element in the path toward a destination. The encapsulated packet is the explicit information used as input to the end-point of the tunnel. Sometimes the starting point of the tunnel is contingent or topological; some times it is coincident with the sender. For example, TOR[2] can be seen as an example of nested tunnels, each with explicit information as input to the PHB at each TOR forwarder.

[2] See www.torproject.org

Indirection schemes and credentials in packets

Recently, there have been a number of proposals (Andersen 2003; Stoica, Adkins et al. 2004; Yang, Wetherall et al. 2005) to improve the security of services on the network by interposing some sort of checkpoint or intermediate relay in the path from the client to the server. This relay can permit or deny access based on the rights of the client, or perhaps rate-limit or otherwise constrain clients as appropriate. These devices depend, in general, both on state stored in the relay and additional information in the packets. Since there are no fields to carry this information today, such schemes are required to build on such fields that already exist (implicit inputs, to use the term above), although this is a clear example where explicit fields would help, or else the schemes depend on some form of clever encapsulation.

IP options

The original IP specification contained the concept of IP options—fields that could be defined and used as needed to carry additional information in the IP header. Over time, the IP option mechanism has fallen into disuse, and it is worth considering why. It might seem that the IP option would fall into the *explicit* class of input, but the processing of the field was still inefficient, because the forwarding device had to parse all of the options to see if one of then represented an input it was to use for its PHB. The router had to process a variable-length field of unpredictable contents, and this was too much computation for a high-performance packet forwarding path; it was almost like deep packet inspection.

So the lesson is that any suggestion for augmenting packets with additional fields that can be used for explicit inputs to PHBs needs to be very sensitive to performance and efficiency issues.

Learning from experience—what could we do better next time?

Addressing

It is generally recognized that the current approach of using the IP address both as a locator and as an identifier is incorrect. There should be two fields, or perhaps three, each serving a distinct purpose.

Locator: This field is used as input to the forwarding PHB of a router. It may be rewritten (as in a NAT device), highly dynamic (in the case of a mobile device) and so on.
End point Identifier (EID): This field is used by each end of the connection to identify itself to the other end(s). There are in general three issues with such a field: how to make sure a malicious sender cannot forge a false identifier, how each end associated meaning with this field (is there some sort of initial exchange of credentials associates with the EID, or do high-level protocols associate some meaning with it once the connection is in place), and third, should elements other than the end-nodes (e.g. PHBs in the network) be allowed to see and exploit this value?

In-network identifier (INID): if the decision is taken that the EID is private to the end-nodes of a connection, then there *may* be need for some other identifier that can be seen and used by PHBs in the path from the sender to the receivers. This possibility raises many sub-questions in turn.

So while the general idea of the locator-identity split is well understood, there is no clear agreement on how to design the system that would result. This would be a challenge question for a future Internet.

Packet size

In the early days of the Internet, it was necessary to exploit communication channels that had been designed for other purposes. One consequence of this fact was that some technologies could not handle "normal size" packets, and packets had to be fragmented into smaller parts to be carried over these channels. Since in general it is less efficient to handle smaller packets, there was pressure to make packets larger in the normal case. This tension led to various mechanisms such as Maximum Transmission Unit (MTU) discovery protocols.

As the Internet has grown up and become the dominant mode of data transport, the need to deal with "unusual" communication channels has faded. Today, network technology is designed to support the Internet. So it would seem that we should be able to take a more simple approach to packet size, which is just to set a standard number. However, since encapsulation and other sort of header extensions (present and future) will cause the packet header to grow and shrink, a future Internet must still think about how to deal with packet size.

Signaling and state setup

In the original Internet, the designers avoided any hint of a signaling protocol or setting up per-flow state in the routers. There were several reasons for this preference. One was simplicity—if we could do without we would avoid yet another thing that could go wrong. In particular, once per-flow state is instantiated in a router, then it has to be managed. When should it be deleted? What happens if the router crashes? The simplicity of the stateless model makes it easier to reason about resilience and robust operation.

Another reason is overhead. It seems a waste to go the overhead of setting up state for an exchange that may involve only one packet. Much better to have a system in which the sender can "just send it". But if this works for one packet, why not for all the packets?

However, the fact that PHBs include more than simple forwarding change this equation. Per-flow state might only be needed in specific elements to deal with special cases. Second, we are now dealing with per-flow state (e.g. in NAT boxes) whether we design for it or not. And some emerging ideas such as indirection schemes depend on per-flow state. So it seems worth revisiting this design decision.

Design options for packet headers

If there seems to be some value (some increase in function or generality) from the ability to provide richer input data to a PHB, it is worth at least briefly speculating on how this might be done.

Blank "scratch-pad" in the packet

A simple idea is to leave a fixed, blank area in the packet header, to be used creatively from time to time. One need only look at all the creative ideas for reuse of the *fragment offset* field to appreciate just how powerful a little extra space can be. To avoid the issues that arose with the IP option field, the expectation for this field should be that a contingent element would not normally look at it. Only elements that have the specific requirement for an input value would parse the field. This might most easily be implemented as a rule that says only the intentional recipient of a packet will examine the scratch-pad area.

The drawback of this scheme is that there might be more than one PHB along the path from the sender to the receiver, so there might be a conflict as to how the scratch-pad should be used. So we might consider a more complex scheme.

Push-down stack model

A more complex model for explicit data in packets is a pushdown stack of records of explicit data, carried as part of the packet header. In this model, the packet is explicitly directed by the sender to the first element that should perform a PHB using data from the packet. That element (conceptually) pops the first record off of the stack of explicit information and uses it as input to the PHB. Then, using either stored PHB state or information in the record that was just popped off the stack, it identifies the next element to which the packet should go. This PHB can push a new record onto the stack, or leave the one provided by the original sender, based on the definition of the intended function.

Issues of performance would suggest that the design would not literally pop a record off a stack (thus shortening the packet and requiring that all the bytes be copied.) A scheme involving offset pointers could be devised that would achieve the desired function.

One way to describe the problem with the IP option was that it was conceived more in the spirit of contingent execution rather then intentional execution. The sender sends the packet addressed to the destination, and routers along the path can look at the options to see what they are supposed to do with it. In the context of aligned interests and per-flow state, we can see a movement toward intentional delivery of packets to nodes with specific PHBs. The push-down stack model (and the more simple scratch-pad model) are more attuned to the intentional delivery model.

This sort of mechanism seems to build on the rough analogy between PHB sequencing and some sort of programming language. And packet encapsulation is a rough version of a push-down mechanism, in which the whole header is "pushed" onto the stack by the encapsulating header.

A heap

The proposal for RBA (see above) includes the idea of the packet header as a *heap* of role-specific headers, or RSH's. The implication of the *heap* is that the roles are not always executed in a pre-determined order, so the idea of push and pop is too constraining.

Per-flow state

The discussion to this point has more or less described a pure datagram scheme, in which each packet is treated in isolation. Per-flow state in the router can enrich the range of PHBs that can be invented, by linking the treatment of different packets in a sequence.

State initiation bit

If we are prepared to consider per-flow state as part of the design, we need to consider whether the protocols should include a standard way to establish and maintain this state. The original preference in the Internet design was to avoid an independent control plane as a mandatory component of the network. (Of course, there is no way to prevent parties from attaching controllers to the network if they choose to.) The original design preference was to carry control information (to the extent that it existed at all) using fields in the data packets, which flowed along the data forwarding path. It is possible to imagine a similar scheme as a standard means for an end-node to establish and maintain per-flow state in intermediate elements.

Such an idea would enrich the expressive power of the packet header by building the idea of state establishment into the design, which would link the treatment of a succession of packets.

Without claiming that all the details are worked out, one can imagine that just as TCP has a state-establishment phase and a connected phase, protocols that establish state in intermediate elements could follow the same pattern. A bit in the header (similar to SYN) could signal that the packet contains state-establishment information. This packet might require more processing overhead (and thus represents a vector for DDoS attacks), but in normal circumstances would only be sent at the initiation of a connection. Once the state is established, some much more efficient explicit indication in the packet could link subsequent packets to that stored state. The two sorts of packets could have different formats.

Maintaining state in intermediate elements

Assuming that the state is soft-state (a choice that could be debated), the protocol should include a means to reinstate the soft state if it is lost. One could imagine a new sort of ICMP message signaling that some expected state is missing. To recover from this, the sender would have to transition back from a fully "connected" mode into a state-setup mode. One could imagine that the sender could re-establish the state in two ways. First, it could do so "from scratch" by sending whatever initial information was used. Second, the intermediate node that holds the state could send back to the source a bundle (perhaps encrypted) that could be used to re-establish the state efficiently. This could be re-sent from the source on demand.

Such a scheme might make sense in the special case of intentionally sending a packet to an anycast address. In this case, the sender is making a deliberate decision to send to this intermediate entity, but the actual physical machine implementing the service might change. In this case, it might be necessary to reestablish some state in that box.

In-network state associated with receivers

The discussion above covered the case of a sender establishing state along a path as part of session initiation. But an equally common case is state set up along a path that arises from the receiver rather than the sender. Setting up and maintaining this state is actually the trickier part of the scheme.

As an illustration of the problems, consider the case discussed above, where, as a part of protecting the receiver from attack, connection validation is outsourced to a set of indirection elements. Since a sender (either legitimate or malicious) may connect to any one of these (perhaps using an anycast address), every one of these elements must have available the information necessary to validate all acceptable senders, or else there must be an authentication protocol for those devices to send off credentials to a back-end service. At a minimum, the protection devices need to be able to find this service.

In practice, this pattern sounds more like hard state, somewhat manually set up and torn down, rather than dynamic soft state.

In other cases, soft state may make more sense. A transient service behind a "firewall of the future" may want to open an incoming port (assuming that a future network has ports, of course), and this may best be done as a dynamic setup of soft state. In this case, mechanisms will need to be provided to make sure the state is still in place, even though the receiver is not necessarily sending any data packets.

Security of architectures with rich expressive power

Protecting the explicit data

Once we introduce the concept of explicit data carried in the packet and used as input to various PHBs in the communication path, we have to ask about the security implications of this data. The classic triple of "confidentiality, integrity, availability" is a useful place to start. Another summary is "that which is not encrypted can be seen, that which is not signed can be changed".

Using the proposal for a push-down stack of records of explicit data for different PHBs, we can see a number of issues. The problem of gross corruption of the header is perhaps beyond the scope of this note—if an element is that malicious, the outcome is the same as a failure to forward, which is a more general problem. The more interesting question is spying on the information, or more purposeful modification of information on the stack, to somehow break a PHB further along the path. To prevent this, in the extreme, each record on the pushdown stack could be encrypted using a public key of the element in question. This implies considerable processing overhead, and some way to get the right public key reliably. The overall complexity is somewhat similar to a client using the TOR system, so we do have evidence that users are willing to tolerate this overhead.

However, in some cases, it may be desirable for one PHB to modify the input data for a subsequent PHB, so the approach taken to secure the data should not be inflexible.

Protecting the intermediate elements from attack

Once we recognize the existence of intermediate elements (and their PHB) as a part of the architecture, we have to take methodical steps to deal with attacks on these devices themselves.

These devices are perhaps (often?) simpler than general purpose operating systems, and it may be possible to engineer them to a higher standard of resistance to penetration attacks. To the extent that these are "first line" elements—exposed to the full open Internet as interfaces to resources behind them, it will be necessary to engineer them to a high standard of penetration resistance.

More generally, we have to ask about DDoS attacks against these devices. For example, if indirection services are deployed to protect servers from DDoS attacks, those services themselves will be attacked if this provides a way to disrupt access to the service itself. So the ability to protect "first-line" elements from DDoS attacks is a general problem the architecture should solve.

Having said that, there need not be only one way to control these attacks. The point of this focus on syntax is to allow diversity in the underlying semantics, including how forwarding is done and protection is implemented. But one general idea that seems to have merit is to give such elements *anycast* addresses. If the elements are replicated and have only anycast addresses, then any DDoS attack will be diffused across the replicated elements. Services requiring a higher degree of protection can subscribe to indirection services with a higher degree of replication.

There will no doubt be many consequences of replicating an element that serves to protect servers from unwelcome users. Most obviously, if the element depends on soft state, then if a legitimate user is moved from one replica of the element to another, the soft state will have to be reconstituted. This requirement is one of the motivations behind the earlier discussion of maintaining soft state.

Protecting legitimate communications from attack.

As we add capabilities for the sender to add more expressive explicit data to the packet header, the possibility arises that third parties "in the network", with interests adverse to both the sender and receiver, and with topological control, will be able to use the available mechanisms to coerce explicit information from senders as a condition of usage. For example, a conservative government might demand that some explicit identifying information be added to the packet as a condition of making a connection out of the country. Today, there is no practical way to demand that, exactly because the packet header is not expressive enough. As we make the header more expressive, we have to consider how we have shifted the balance of power among the various actors. This consideration fits into the larger discussion of tussle below.

Reasoning about generality

The various examples discussed earlier suggest that intermediate PHBs are used in two common ways. One is to achieve better control over forwarding and routing—e.g. mobility and tunnels. The other use is to provide various sorts of protections for a receiver that does not want to be fully attached to the open Internet and fully vulnerable to arbitrary attack. In the first case, we can assume that the interests of the parties are aligned. By allowing users to exploit more flexible options for forwarding, it would seem that such services enhance generality. In the second case, we can assume (since the interests of the parties are adverse) that the forwarding will be constrained *topologically,* and the relationship is the one I called *coerced*—the sender, to reach the receiver, is required to pass through some sort of first-line PHB, and is required to present some sort of explicit information. A successful design will insure that this requirement is not too much of a burden for a valid user, but is sufficiently robust that a malicious user is deflected. Of course, the details of this will depend on the specific circumstances of the final receiver—whether it is prepared in principle to talk to anyone or interested only in a known group of clients.

Such restrictions are in principle the opposite of generality—they intentionally preclude certain classes of things from happening. However, looking only at the level of syntax and header construction, as we have done here, it seems unlikely that we can provide rules to tell good from bad behavior. Such restrictions are the opposite of availability as well; they deliberately make it impossible to do something. Since the high-level goal of availability is to "route around" flaws, topological restrictions must be balanced against the resulting reduction in diversity.

Tussle and regions

Consider the example discussed above of a firewall, put in place by the receiver to block attacks by the sender. In this adverse circumstance, the receiver must depend on implicit arguments and topological routing. For this to work, the region of the network within which the receiver is located must provide enough control over topology (connectivity and routing) to ensure that the firewall is in the path of the packets.

To generalize, what this illustrates is that different actors within the network (the sender, the receiver, the ISPs, other third party participants) will have the right to control certain parts of the network (or the expectation that certain parts will be operated consistent with their requirements), and within each such region of the network, the expressive power of the parts found there (the PHBs and the routing) will be used to further the intentions of that actor.

The factor that will determine the outcome of the tussle (e.g. the balance of power) is not the PHBs (which, as we noted, can be more or less anything), but the information in the packet that can serve as the input to the PHB, and the order of processing of the packet.

The order of processing arises from the natural nature of packet forwarding: the packet originates in the region of the sender (who thus gets first crack at any desired PHBs), then enters into the global network, and finally enters into the region of the receiver and the PHBs found there. The information that is in the packet at each stage is a consequence of this ordering. For example, the sender can include data in a packet that is used by the PHBs in the region of the sender and then stripped out so that the other regions cannot see it. While the packet is in the global "middle" region, some or most of the packet can be encrypted to prevent it being examined, and so on.

But as I have noted, PHBs can do more or less "anything" that can be derived from the information in the packet, and the routing is under the control of each of these regions. The fixed point in this design is the packet header itself. So when we think about putting more or less expressive power into the header (e.g. a more or less expressive format), we should consider whether the different options shift the balance of power in ways that match our preferences.

Debugging

All mechanisms fail. Complex mechanisms fail complexly. If we design a network that permits all sorts of complex routing options and invocation options for PHBs, the potential for failure will certainly go up. Tools to debug and recover from such failures will be critical if we are to meet goals of availability and usability.

PHBs that are contingent are the hardest to debug, since the sender did not invoke them intentionally. The idea of invoking explicit mechanisms to try to diagnose a failure in a box the sender did not even know about is troubling. This fact suggests that when effective diagnosis is desired, the design should prefer intentional invocation of PHBs.

If the interests of all parties are aligned, it would make sense that the tools for debugging would be effective and useful. However, if the interests of the parties are adverse, the situation becomes more complex. If for example, an attacker is being thwarted by a firewall, it may be in the interest of the firewall to prevent any sort of debugging or diagnosis of the failure. The goal (from the point of view of the defender) is to keep the attacker as much as possible in the dark as to what is happening, so as to prevent the attacker from sharpening his tools of attack. So while tools and approaches for debugging and diagnosis must be a part of any mechanisms to provide expressive power for a FI, tussle issues must be taken into account in their design.

(Certain classes of failure are easy to debug, even for contingent PHBs. Fail-stop events that cause the element not to function at all can be isolated and "routed around" just like any other router failure. "Fail-go" events do not require diagnosis. It is the partial or Byzantine failures of a contingent PHB that may cause diagnosis problems for the sender.

It is for this sort of reason that intentional invocation of PHBs is to be preferred unless the goal of the PHB is to confound the sender.)

Evolvability

In this context, the term *evolvability* refers to the ability of the network architecture to survive over time and evolve to meet changing needs while still maintaining its core coherence. The history of the Internet provides some informative case studies.

In the early days, the designers of the Internet thought that the concept of a single global address space was part of the Internet architecture, and we bemoan the emergence of NAT devices, VPNs etc, as an erosion of the architectural coherence of the Internet. To some extent this is true; NAT makes the deployment of passive services behind the NAT barrier more complex, and leads to such inelegancies as STUN. On the other hand, it is also clear that in the large, the Internet has survived the emergence of NAT, and perhaps global addresses did not need to be such a central assumption of the presumed architecture.

Perhaps less mourned but more relevant is the atrophy of IP options. IP options were hard to process in the fast path of routers, and were deprecated in practice to the point where they are essentially gone. IP options were developed to allow for future evolution of the architecture, and they could have provided a substantial degree of expressive power. They vanished. One could speculate about the implications of this fact:

- This degree of expressive power is not in fact necessary, and made the network over-general.
- IP options were not well designed, and required much more processing than a better-designed option.
- The loss of IP options represents an un-orchestrated decision to favor short-term cost reduction over future evolvability.

Whatever the mix of actual reasons is, one can learn two lessons from the above.

First, avoid mechanisms that are costly to maintain when they are not needed. For example, if there are fields in packets that are used to carry “extra” input values to PHBs, design them so that only the device that actually implements the PHB has to parse those fields or otherwise pay any attention to them. If the packet is intentionally addressed to the device, then the processing rule is clear: if the packet is not for you, don’t look at the extra fields.

Second, any mechanism added to a packet header should have at least one important use from the beginning, to make sure that the implementation of the mechanism remains current. If designers propose something intended to facilitate evolution, but cannot think of a single use for it when it is proposed, perhaps it is overkill.

Finally, the addition of tools to promote evolvability may shift the tussle balance, so enthusiasm for rich expressive power may need to be tempered by a realistic assessment of which actors can exploit that power. Indeed it would seem that the goal of evolution over time is inseparable from the goal of operating in different ways in different regions of the network at the same time, in response to different perceived requirements within those regions.

Conclusions

Making design choices about the potential expressive power of an FI seems to call for a tradeoff between evolvability and flexibility on the one hand, simplicity and understandablity on the second hand, and tussle balance on the third hand. However, there is no reason to think that this tradeoff is fundamental. Creative thinking might lead to alternative ways of defining packets and routing such that we gain in all three dimensions. To explore this space, it may be helpful to ask ourselves challenge questions of the sort that a clean slate thought process invites, such as why do packets have to have addresses in them, or why do we need routing protocols?

Acknowledgement

This version of the paper greatly benefits from comments provided by John Wroclawski to an earlier version.

Andersen, D. G. (2003). Mayday: distributed filtering for internet services. Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems - Volume 4. Seattle, WA, USENIX Association.

Braden, R., T. Faber, et al. (2003). "From protocol stack to protocol heap: Role-based architecture." SIGCOMM Comput. Commun. Rev. **33**(1): 17-22.

Stoica, I., D. Adkins, et al. (2004). "Internet indirection infrastructure." IEEE/ACM Trans. Netw. **12**(2): 205-218.

Yang, X., D. Wetherall, et al. (2005). A DoS-limiting network architecture. Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications. Philadelphis, Pennsylvania, USA, ACM Press.