Service-Centric End-to-End Abstractions in Next-Generation Networks

Tilman Wolf
Department of Electrical and Computer Engineering
University of Massachusetts Amherst
wolf@ecs.umass.edu

***Abstract*— Next-generation network architectures will be governed by the need for flexibility. Heterogeneous end-systems, novel communication abstractions, and security and manageability challenges will require networks to provide a broad range of services that go beyond the simple store-and-forward capabilities of today's Internet. This paper introduces new abstractions for information transfer and data services in the network, which overcome the constraints of the end-to-end argument that has dominated current network designs. The explicit separation of communication and processing allows the composition of a variety of information transfer patterns that expand the capabilities of the network to meet its next-generation challenges. We discuss implementation issues that arise in this network architecture and present several examples of how applications can utilize the proposed abstractions. This present a first step towards a unified view of the convergence of networking, processing, and their distributed applications.**

I. Introduction

Computer networks continue to expand in terms of complexity, size, their applications, and the diversity of systems connected through them. As we have the opportunity to consider novel network architectures, it is important to explore the impact and tradeoffs of different end-to-end communication abstractions. This topic is especially important in light of the expansion of networks to incorporate sensors, mobile wireless devices, and other end-systems that are considerably different from traditional hosts and servers.

The end-to-end argument that has dominated the design of the current Internet poses a considerable limitation on the flexibility of the network architecture to adapt to new requirements. Assuming a network that simply forwards packets between end-systems that implement more complex functions is a too narrow view that is no longer suitable. Even the current Internet provides a number of features (or "services") that go beyond forwarding, and this shift towards more functionality will continue. Next-generation networks will have to offer a much broader range of such services, and the application layer abstractions that we have developed for the current Internet (e.g., TCP sockets) will no longer be sufficient. We propose a novel way of providing suitable application layer abstractions for communication that meet the needs of next-generation network architectures. This is a particularly important aspect of the network architecture as it is exposed to all applications and determines what kind of functionality can be provided by the network.

Our key idea is to base communication abstractions on the transfer of *information* rather than the process of sending *data*, because the fundamental task of a network is to transfer information form one system to another. In the end, it is data bits that are being sent on the links, but by focusing too quickly on this process a lot of opportunity for new networking paradigms gets lost. By providing some clues on the semantics of the information that is to be transferred, the network can provide more advanced services than blindly moving bits. For example, information may be encoded differently, stored for later reuse, etc. The process of handling data representation, transfer, and storage of information is considered a *service*. These service can be used explicitly or implicitly in order to address practical data transfer problems. Examples for such services range from simple reliability and congestion control (currently implemented in TCP) to content caching (in web proxies) and transcoding. By focusing on information transfer and allowing services to implement different mechanisms for encoding, transferring, and processing data, we can not only replicate the functionality of the current Internet but also explore novel communication abstractions that address the problems of next-generation distributed computing platforms.

In this paper, we propose an Information Transfer and Data Service (ITDS) architecture that utilizes router-based processing functionality to implement data services on network nodes. We describe our vision of the operation of such an architecture and how current and future network applications can be implemented. We also explore the design trade-offs and theoretical challenges are encountered. To tackle these problems, we propose an approach to handling service abstraction and to mapping service tasks to resources. In general, this paper focuses more on weighing different design alternative than providing specific results of one particular proof-of-concept implementation. We believe that such general considerations of how to handle end-to-end abstractions in the next generation network are more timely than already committing to one particular solution.

We discuss related work in Section II. The ITDS architecture is described in more detail in Section III, and examples for application use are shown in Section IV. Considerations for a possible implementation are discussed in Section V-C. Section VI summarizes and concludes this paper.

II. Related Work

Several shortcomings of the design of today's Internet are expressing themselves in the lack of flexibility, security, and manageability. When the Internet was originally designed in the 1970's, the goal was to have a simple, packet-switched communication infrastructure, which connects a large number of networks with gateways or routers [1]. The network itself was kept relatively simple and provided just basic communication between the end-systems. This led to networking protocols, where most of the complexity is implemented on the end-systems (e.g., retransmission of lost packets, congestion control based on end-system observations).

The design of the current Internet has been revisited and proposals for new Internet architectures have been considered for some time [2], [3], [4]. The move of the Internet towards a commercial enterprise has changed the underlying assumption of trust and economic incentives [5].

Over time, several additions have been proposed and implemented in the Internet, because the initial design did not consider them. Since the Internet does not support the dynamic deployment of new protocols, these features needed to be added as special processing functions to routers. The following list highlights a few, which are characterized by the need of support by the network (i.e., they cannot be implemented on end-systems only):

- Random Early Detection (RED [6]) is a queue management scheme for routers to fairly drop packets from rogue TCP flows. This is can be implemented in a very simple fashion, but it constitutes a type of processing on a router. Many current routers implement some form of RED, but only few use it in practice.
- Network Address Translators (NAT [7]) are another common component in IP networks. A NAT allows multiple hosts in a subnet to use a single globally unique IP address. IP packets passing between the subnet and the Internet are modified by the NAT. This reduces the number of IP addresses used by a subnet and thereby extends the time before the IP address space is exhausted.
- Firewalls [8] are a standard security component of most networks. Packets are filtered depending on rules defined by the network administrator. This enables the blocking of network traffic that could compromise the security of hosts on the network (e.g., port scanning). Firewall rules can be numerous and complex, which requires significant computational power on the firewall to keep up with typical access link speeds.
- Intrusion Detection Systems (e.g., snort [9]) check packet headers and content against signatures of well-known attacks. Traffic that is considered malicious can be blocked dynamically.
- Web Switching [10] is a method for distributing a web server over several physical machines while presenting a single font-end to the outside. Web switches parse HTTP requests in packets and determine the appropriate server to which to forward the request. Since the HTTP request is sent only after the TCP connection is established, the web switch also has to splice the TCP connection between client and back-end server.

In practice, these changes either have been slowly added to all routers (e.g., RED) or they were implemented on servers that are connected to routers (e.g., intrusion detection). These specialized solutions are working reasonably well in practice, but also cause the Internet architecture to diverge more and more from its original design as more and more services require processing inside the network.

One important observation that can be derived is that the *functionality* of the network plays a key role in today's Internet. It is not bandwidth anymore that dominates the considerations for network architectures and their implementation. Instead, it is becoming increasingly important to provide customers with features, such as security and manageability. An increasing number of specialized services will be required in order to support the growing number of heterogeneous end-systems and network technologies. To provide an infrastructure for such services, it has been proposed to equip routers with general-purpose processing capabilities in the data path. Services can then be implemented in software and be created, deployed, and used dynamically. In the most general form, processing has been proposed in active networks [11], which have exhibited considerable challenges in manageability and security. A more moderate approach of data-path processing is found in form of programmable routers, where the system administrator (instead of the end-system application/user) may deploy new functions that are performed on the router [12]. An example for utilizing processing capabilities to deploy new protocols is shown in [13].

From a technology point of view, the trend towards the need for more services is being met by recent progress in embedded processing systems. "Network processors" (NPs) are multi-processor systems-on-a-chip that are optimized for high-performance data I/O and processing. Using multiple embedded RISC processor cores and specialized co-processor for hardware acceleration, NPs can process network traffic in software at Gigabit per second speeds. Thus, NPs provide an ideal platforms for implementing network services ranging from simple forwarding to complex payload processing. At this point there are a number of companies producing a variety of network processors such as the Intel IXP family [14].

III. Information Transfer and Data Services

We propose a novel end-to-end abstraction that can utilize the capabilities of next-generation network and make them accessible to end-systems. We allow the processing of data as it traverses the network and thereby remove the constraints set in place in current network architectures.

A. End-to-End Information Transfer

With the design of any abstraction, there is a trade-off between simplicity and generality. Ideally, an application layer abstraction should be simple to use for a straightforward information transfer. At the same time, it should also allow for

complex transactions that consider realistic system constraints. The key questions are:

- How much general-purpose processing and storage capabilities can and should be placed in to the network infrastructure?
- How much of that functionality should be exposed to the end-system?
- How can the semantics of an information exchange be specified to utilize the network's capabilities to its fullest?

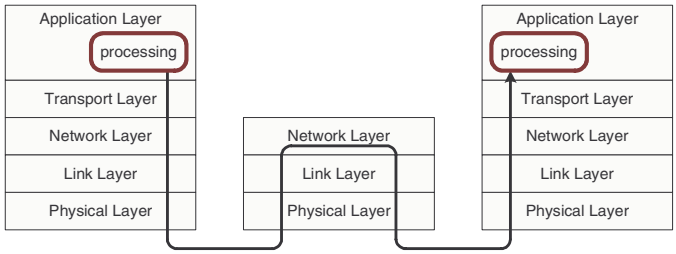
1) Information Transfer vs. Data Transfer: The key idea for our proposed end-to-end abstraction is to focus on the basic purpose of a network: *information transfer*. The encoding of information into data is a secondary step, and there are numerous ways of doing that. In any case, it is the information that is relevant to the end-system. For example, consider refreshing a web page that has not changed since the last retrieval. It does not matter how the information ("the web page has not changed") is conveyed to an end-system. There are many ways of how the corresponding data transfer can be implemented: In the simplest case, the data is downloaded again and displayed. Alternatively, the server might just notify the client that the page has not changed, thus saving an unnecessary data transfer of the body of the page. It is also possible that a proxy interacts with this exchange.

In the proposed ITDS architecture, end-system applications specify the information transfer that is desired and the network (in cooperation with the end-systems) determines the appropriate handling of data. There are three major aspects that need to be considered in this process:

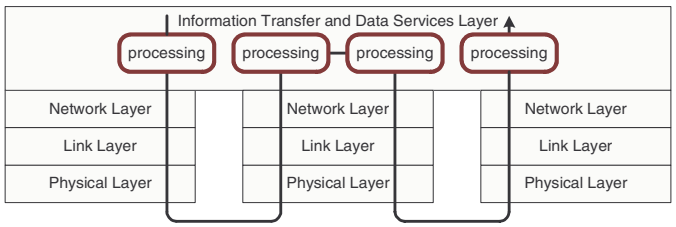
- Data Encoding and Semantics: The information that needs to be transmitted needs to be encoded into data. In many cases, this is already given, for example in a file transfer.
- Data Transmission: The transfer of data between end-systems and network systems corresponds to the traditional functionality of a network.
- Data Processing: In order to provide advanced services, data needs to be processed. This includes modification of the transferred data and processing for the purpose of control.

The last point – having processing as a first-class networking function – is the key difference to the existing Internet architecture. Instead of limiting processing to end-systems, as it has been done in the current layered Internet architecture, we propose to permit processing throughout the network. We call these processing tasks *data services* and their purpose is to implement the handling of data that is necessary to achieve the desired functionality and performance properties.

The difference between the layered Internet architecture and the Information Transfer and Data Services abstraction is contrasted in Figure 1. Instead of forcing the transport and application layers to be limited to the end-systems, processing can take place throughout the network. Transport layer functionality is seen as part of the application layer and thus not considered a separate layer. The network, link, and physical



(a) Layered Internet Architecture



(b) Information Transfer and Data Services Architecture

Fig. 1. Layered Internet Architecture Compared to Information Transfer and Data Services Architecture.

layers are shown unchanged as this work does not address these aspects of network architecture. The proposed ITDS abstraction can interface with any network-level architecture that provides system-to-system connectivity.

2) Data Semantics: By putting the encoding of information under control of the network, we allow the system to maintain semantic information on the bits that are being transmitted. The network can therefore modify the data to more efficiently transfer the information. In the example of a web caching, a proxy could intercept web requests and respond with a local copy of a document. As a result the same *information* is transferred without involving an end-to-end *data* transfer. This process would not require any configuration on the end-system since it could be done transparently by the network.

It would not be feasible to assume that the semantics of all information transfers on the Internet could be formalized concisely. Instead, it is sufficient to describe the main characteristics of the information exchange. An initial and probably incomplete list of possible characteristics is:

- Streaming vs. Random Access: determines if the information transfer is of a typical request/response type (i.e., random access) or a stream of information that is sent continuously to the receiver.
- Point-to-Point vs. Multi-Point: determines if the information is distributed to only one end-system or to multiple.
- Interactive vs. "Canned": determines if information is generated interactively (i.e., "on-the-fly") or if it already exists and just needs to be distributed.
- Bandwidth-Sensitive vs. Delay-Sensitive: determines the performance demands of different information transfers.

These characteristics can be combined to represent a broad set of information exchanges. For example, an typical `ssh` session would be random access, point-to-point, interactive, and delay

sensitive; a video broadcast would be streaming, multi-point, and either canned and bandwidth sensitive (e.g., movie) or interactive and delay-sensitive (e.g., video conference).

B. Data Services

The data service aspect of the ITDS architecture is the key to providing flexible handling of data streams in the network. A data service can receive, store, process, and transmit data that is transmitted over the network. Data services leverage the semantic information about a data stream to implement different functionalities. Examples for data services are:

- Reliability: buffering and acknowledgement-based retransmission of lost data as it is done in TCP.
- Privacy: encryption and decryption services between end-systems (similar to SSL) or subnets (similar to VPN).
- Congestion Control: limiting of data transfer rate based on the state of the (sub-)network.
- Caching: storing of information for certain application-layer protocols and making it accessible to other connections.
- Security: firewalling, intrusion detection, payload scanning and other mechanism to identify and mitigate attacks.
- Quality of Service: prioritized forwarding of data based on service requirements.
- Multicast: duplication and forwarding of data along multiple links and local retransmission for scalable, reliable multicast.
- Payload Transcoding: depending on the semantics of the transferred information, this service can adapt the content that is transferred. For example, large images from web pages can be downsampled for display on a cell phone. This service is very specialized and highly dependent on the transferred information and its coding in data.

There are a number of other services that could be conceived: multi-path routing, forward-error-correction coding, multi-destination content-distribution using erasure codes, etc. One important observation is that two services can provide the same functionality, but are implemented differently. For example, congestion control can be AIMD-based as in TCP or can use explicit congestion notification (ECN).

1) Computation on Routers: Data services on network nodes require the availability of general-purpose processing capabilities on these systems. The main challenge when designing such a system is to manage the different data services and the state that is associated with different streams of data. In principle, there are three steps that need to be performed on a ITDS-enabled router system to support basic data service functionality:

- Identification of Processing Requirements: Based on the control information that has been provided to the ITDS-enabled router, it can be determined what data services a particular connection (or individual packet) requires. In order to maintain this information, a packet classification or flow identification mechanism is necessary.
- Access to Processing Context: In order to provide processing tasks that perform computation across multiple packets, it is necessary to maintain processing state for each connection. This context needs to be made available whenever processing is continued.
- Resource Management: Both processing time and memory space are resources that need to be tightly controlled. Since services are not installed dynamically by end-systems (as it was proposed for active networks), it is much easier to control processing tasks. In terms of processing performance, it is difficult to compare different system implementations [15]. Enforcing resource reservations for processing is further complicated by the fact that computation that is terminated due to excessive resource usage may leave the system (or at least the data service for one particular information transfer) in an inconsistent state.

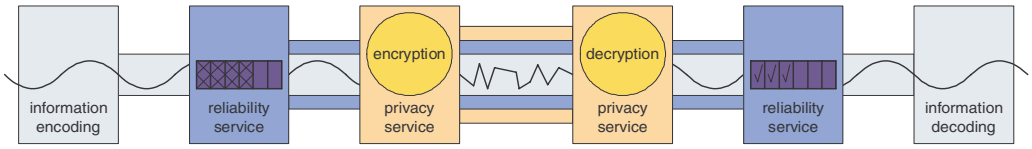
We note that processing on routers is provided for efficient, high-performance, secure, manageable end-to-end communication. Processing is not provided for the sake of having processing resources available remotely. Clearly, it is possible to provide processing resources as infrastructure components, but that issue goes beyond the scope of this paper.

2) Storage: Together with computation, it can be considered to provide storage as a feature for future networks. From a systems point of view, this is a much further reaching proposition that simply providing processing capabilities. However, it also permits the consideration of novel end-to-end networking paradigms. For example, instead of simply transferring information between end-systems, networks could be used for information storage and retrieval. Similar to peer-to-peer overlay networks, which are currently implemented on end-systems to exchange content, an ITDS-enabled network could be used to store data on routers and or dedicated storage systems within the network infrastructure. The basic operations to access this information could be *put* and *get*.

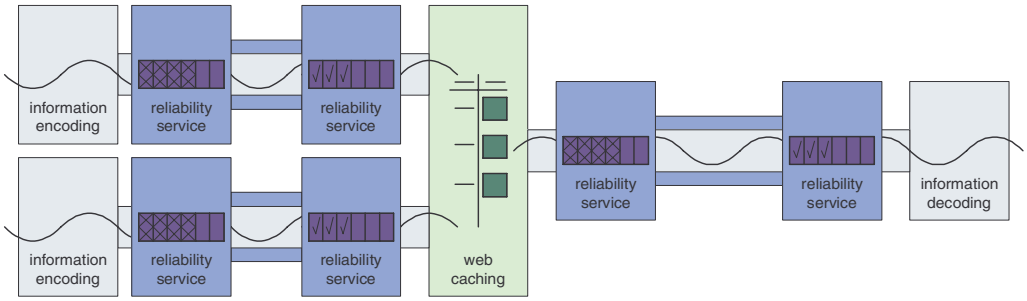
IV. Examples

To illustrate the ITDS concepts, we discuss a few examples of data services that implement different types of end-system-to-end-system information transfers:

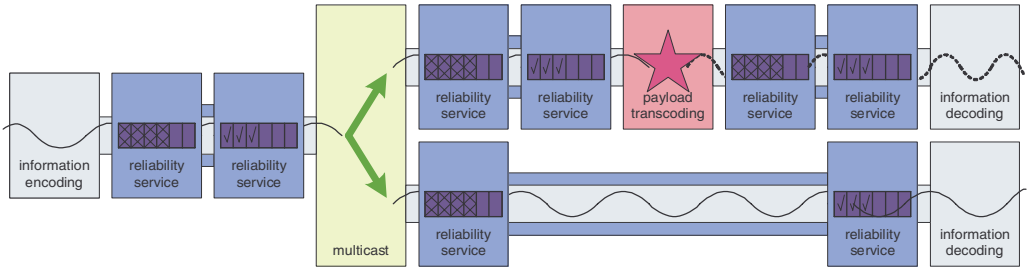
- Example I: Reliable and Private Communication. This type of communication can be composed from two data services that implement reliability and privacy. The combined data service can then be applied to any type of point-to-point information transfer.
- Example II: Web Caching. This type of communication can be achieved by combining a caching service with a reliability service. Multiple end-systems can use the same caching service and thus achieve the desired caching functionality. This service can be applied to a conventional point-to-point information transfer with a semantics that is understood by the caching service.
- Example III: News Distribution. This type of communication uses multicast service (and reliability) service to reach a large number of end-systems. It may use



(a) Example I: Reliable and Private Point-to-Point Communication



(b) Example II: Web Caching



(c) Example III: Content Distribution and Transcoding

Fig. 2. Examples of ITDS Scenarios.

a "push"-paradigm rather than having the end-systems request content individually. In addition, various nodes can employ content transcoding operations to adapt the presentation of the transferred information.

Figure 2 illustrates the semantics of the information flow for these three examples. The nesting of colored boxes indicate the layers of semantics that are added to an information transfer (e.g., indicating that a particular data stream is an encrypted representation of the information).

Being aware of the semantics of information exchanges allows the network to provide data services that further improve the end-to-end communication. These services can be initiated automatically by the network in case the end-system application does not explicitly specify them. For example, an end-system might not have specified congestion control, but the network enforces it to ensure fair sharing of resources. Intrusion detection is another example where the network or the administrator can add services that are orthogonal to what the end-user application requests.

V. ITDS Implementation

To implement the ITDS functionalities described in the previous section, we need to address a number of fundamental algorithmic and systems questions that need to be addressed by any network implementation that supports services:

- How can the use of services be specified?
- How can service tasks be mapped to network nodes?
- How can services be controlled by the network and by the end-system application?

We discuss several issues related to the design a practical environment to specify, compose, map, and control data services.

A. Service Creation

1) Service Specification: In order to fully utilize the capabilities of ITDS abstractions, it is necessary to provide a set of suitable programming abstractions. The abstractions need to be simple enough to setup a straightforward communication channel quickly, yet rich enough to accommodate more complex configurations. We envision the programming interface to be comparable to that used by BSD sockets, except that a larger number of parameters need to be passed to the system. During a setup of a socket, the end-system application needs to specify to the ITDS system the structure of the information flow between end-systems and the data services that should be invoked.

There are many possible approaches to how one can specify this information in an easy to read and parse format. We

propose to use an abstraction that is similar to that of UNIX "pipes" and has already been explored in the context of programmable networks [16]. This concept can be extended further include a variety of annotations that are necessary in the context of the ITDS architecture.

The idea of the "annotated pipe" approach is to view communication as a transfer of information from one end-system to another, where processing is performed on intermediate nodes. The simplest case would be the transmission of information from one node to another. The syntax for such a transfer could be specified in many ways, and finding a suitable notation is part of ongoing and future work. However, to illustrate the concept, we make use of UNIX-like syntax:

```
{N1 > N2}
```

could be used as a specification for a simple data transfer between two nodes. This transfer could be further annotated to utilize more advanced features. For example,

```
{N1 | TCP_TX > N2 | TCP_RX}
```

could specify the use of TCP features on the transmission. In this case, the vertical bar (|) indicates that data is to be passed through a processing stage on that particular node. A further extension would be an additional transcoding service at an arbitrary node (*) in the network:

```
{N1 | TCP_TX > * | TCP_RX |
TRANSCODE | TCP_TX > N2 | TCP_RX}.
```

Similarly, this notation can be extended for other cases.

2) Service Composition: The data services available for information transfer can be composed to meet the need of a particular information transfer. When specifying a communication interactions with annotated pipes, the ITDS systems need to be able to verify that a particular request yields a feasible information exchange. Thus, it is necessary to associate a very simple notion of semantics with the information as it is being transferred and processed. The end-to-end communication can be augmented by predicates that match the concepts described above in terms of information transfer (e.g., point-to-point, streaming, etc.) and data services (e.g., encrypted, etc.). When an end-system application requests a set of services, the system can apply a few simple checks to ensure that the requested communication is semantically correct. For example, privacy only makes sense as a service if it is used with two subservices: encryption and decryption. Similarly, reliability has two parts that are necessary for a correct usage (i.e., all `TCP_TX` uses need to be matched with `TCP_RX`). Finally, there are required and optional services. The system may decide to add services that are required by policy (e.g., intrusion detection and congestion control) or it may add services that improve performance (e.g., caching).

Another important consideration is the scope of a service (e.g., local, subnet, or global). The scope defines where certain services are available and can be performed. For example, VPN encryption should be done within the local scope (i.e. the LAN), before data is transferred on the WAN.

Some services need to be deployed throughout a portion of a network (e.g., QoS or congestion control), but we consider that different network service providers could use different techniques on different networks. By allowing the composition of different service along the path from end-system to end-system, we can achieve global inter-operation between data services that are implemented differently.

3) Service Control: Many of the data services described above require specific configuration information. For example, privacy requires a set of keys (or at least a specification for the mechanism of key exchange). It is assumed that during the setup of a communication, the end-system can provide this information. In some cases, it may be desirable for the network to check or change certain parameters. For examples, service limit the amount of processing or memory resources the can be allocated. These programming abstractions will need to be refined as ITDS abstractions are implemented, but they provide an initial starting point for exploration.

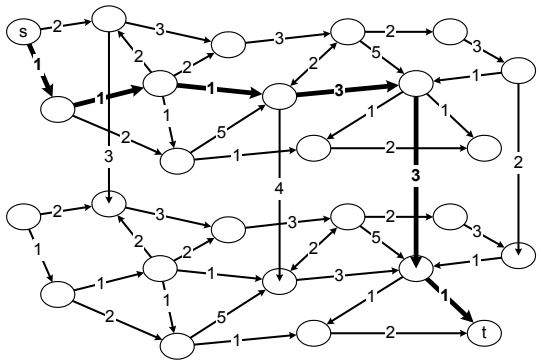
B. Service Mapping and Placement

Another important topic is how to map processing requirements to processing resources inside the network. Given a specification for an information transfer in form of an annotated pipe, the network needs to determine where resources are available and which mapping is most suitable. In a connection-oriented implementation, this may be done once at connection setup. In a connection-less implementation, this may be done per-packet.

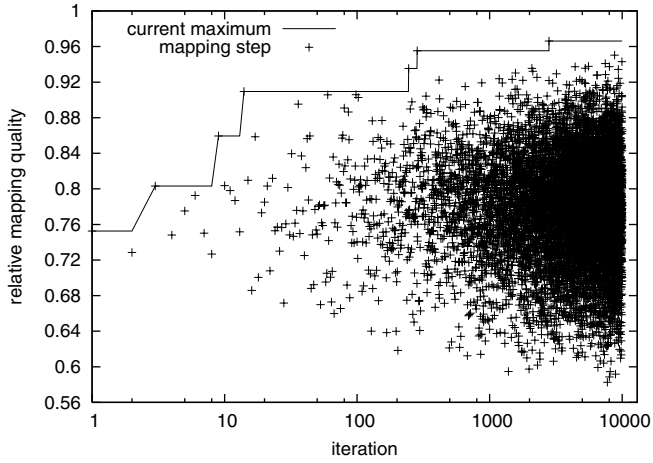
The problem of mapping processing tasks to multiple available processing resources is NP-complete [17]. In prior work, we have developed several heuristic approaches to this problem in different contexts. The layered graph approach shown in Figure 3(a) constrains the mapping problem in such a form that it can be reduced to a shortest path problem [18]. Each layer presents a different stage of the processing that needs to be performed on the data stream and vertical connections present potential processing nodes. Using a single cost metric, the shortest path can be found in the layered graph and then mapped back to a single layer. A solution it not guaranteed when considering resource constraints [19].

Another heuristic for tackling this mapping problem is to randomly place processing tasks on nodes in the network. This approach has been used successfully to assign processing tasks on multiprocessor NP systems [20]. The performance of such an approach can suitable when very little processing time is available. Figure 3(b) shows the progression of mapping quality over an increasing number of randomized placement attempts. Even when using only a few attempts (e.g., 100), good performance (e.g., better than 90% of the maximum performance) can be achieved. The particular performance numbers depend on the details of the problem.

We can extend these heuristics to apply to our ITDS architecture. The annotations in the pipe abstraction require additional constraints that are not yet considered. Also, scalability becomes an important issue when considering mapping problems that span the entire Internet. By hierarchically partitioning the network, it may be possible to solve mapping problems on a local and global level.



(a) Layered Graph



(b) Randomized Mapping

Fig. 3. Mapping Techniques. The layered graph approach constrains the mapping problem to a shortest path problem. The randomized mapping approach places tasks randomly and obtains incrementally better solutions over time.

C. Implementation Scenarios

Depending on the capabilities of the underlying network, different implementations of ITDS may be different. For example, we have been careful not to specify if an ITDS system is based on connection-oriented or connection-less data exchange. Either is possible with different tradeoffs. Similarly, state management, processing hardware, and service placement (edge vs. core) can be decided.

1) Connection-Oriented vs. Connectionless Communication: In addition to providing data services, the network needs to provide a mechanism for directing traffic towards particular nodes. Some services require that all packets traverse the same data service node in order to achieve consistent behavior.

- A connection-oriented implementation of ITDS would make it possible to ensure that all packets are routed to the same data services, which may be required for consistent processing. It also allows tight control over resource usage of different connections.
- A connection-less implementation would match more closely with the architecture of the current Internet. It would require less overhead on connection setup and teardown and would require fewer resources on data service nodes. In today's Internet architecture, source routing or tunneling could be employed if consistent routing was required.

2) State Management: Both connection and processing state need to be maintained at some level by the network to support data services. One important question that needs to be considered for ITDS is what kind of guarantees are necessary. One can envision a system that guarantees all state, i.e., is entirely "hard-state." While this will make processing easiest, it may also be most difficult to implement. In contrast, a soft-state approach may not provide sufficient guarantees, but is clearly simpler to implement. There are a variety of intermediate solutions, which have been discussed in the context of signaling protocols [21]. It is also important to note that the issue of state management ties in with the issue of connection-oriented and connection-less communication. The hard-state approach is difficult to maintain when communication is connection-less and path changes can occur.

3) Processing Hardware: In the simplest case, a workstation can be used as a processing systems through which data is forwarded. Such a configuration usually does not scale to very high data rates or complex processing tasks. An alternative that is optimized for high-performance processing of network traffic is the use of network processors. These embedded multiprocessors employ parallel processor cores to achieve Gigabit per second link speeds for header-processing applications and tens to hundreds of Megabit per second link speeds for payload-processing applications [22]. A third choice is to use programmable logic devices (e.g., FPGAs) to implement some of packet forwarding and processing tasks [23].

We envision that the ITDS infrastructure will follow the route of network-processor-based programmable router systems, which is the most scalable approach to providing the necessary processing power for high-speed data service. However, the use of a multiprocessor NP systems makes the development of a prototype ITDS also considerably more challenging. Some of the operating system issues that arise in this context have been addressed in [24].

4) Edge vs. Core: Another interesting questions in the context of ITDS is how "deep" inside the network data services can be offered. Currently, there is a trend towards deploying a lot of functionality at the edges (e.g., firewalls, intrusion detection systems, wireless gateways). Similarly, large server farms often offload networking-related processing tasks (e.g., TCP termination, SSL processing) to front-end systems.

With increasing link speeds towards the core of the network, the available processing resources on ITDS router and the complexity of the requires services determine where the border for possible data services is. Having a "natural" constraint on where processing is feasible will help in managing the complexity of the service mapping problem described above, since a considerable chunk of the network would not support data services.

The design choices for these implementation scenarios have an impact on the performance of the ITDS system and its operation in detail, but are independent of the overall conceptual abstractions of information transfer and data services. Therefore, it is possible to envision interoperable instances of ITDS architectures that are based on different implementation details.

VI. Summary and Conclusion

This paper introduces a new abstraction for information transfer and data services in the network, which addresses the challenge of overcoming the constraints of an network design that is based on the end-to-end argument. The explicit separation of communication and processing allows the composition of a variety of information transfer patterns that expand the capabilities of the network beyond what is possible in the current Internet. This present a first step towards a unified view of the convergence of networking and processing and its distributed applications.

References

[1] D. D. Clark, "The design philosophy of the DARPA internet protocols," in *Proc. of ACM SIGCOMM 88*, Stanford, CA, Aug. 1988, pp. 106–114.

[2] D. D. Clark and D. L. Tennenhouse, "Architectural considerations for a new generation of protocols," in *Proc. of ACM SIGCOMM 90*, Philadelphia, PA, Sept. 1990, pp. 200–208.

[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 Transactions on Internet Technology*, vol. 1, no. 1, pp. 70–109, 2001.

[4] D. Clark, K. Sollins, J. Wroclawski, D. Katabi, J. Kulik, X. Yang, B. Braden, T. Faber, A. Falk, V. Pingali, M. Handley, and N. Chiappa, "New Arch: future generation internet architecture," Tech. Rep., Dec. 2003.

[5] S. Shenker, D. Clark, D. Estrin, and S. Herzog, "Pricing in computer networks: reshaping the research agenda," *SIGCOMM Computer Communication Review*, vol. 26, no. 2, pp. 19–43, 1996.

[6] S. Floyd and V. Jacobson, "Random early detection (RED) gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, Aug. 1993.

[7] K. B. Egevang and P. Francis, "The IP network address translator (NAT)," Network Working Group, RFC 1631, May 1994.

[8] J. C. Mogul, "Simple and flexible datagram access controls for UNIX-based gateways," in *USENIX Conference Proceedings*, Baltimore, MD, June 1989, pp. 203–221.

[9] *The Open Source Network Intrusion Detection System*, Snort, 2004, http://www.snort.org.

[10] G. Apostolopoulos, D. Aubespin, V. Peris, P. Pradhan, and D. Saha, "Design, implementation and performance of a content-based switch," in *Proc. of IEEE INFOCOM 2000*, Tel Aviv, Israel, Mar. 2000, pp. 1117–1126.

[11] D. L. Tennenhouse, J. M. Smith, W. D. Sincoskie, D. J. Wetherall, and G. J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, Jan. 1997.

[12] T. Wolf, "Challenges and applications for network-processor-based programmable routers," in *Proc. of IEEE Sarnoff Symposium*, Princeton, NJ, Mar. 2006.

[13] S. Pandey, A. Somani, and A. Tyagi, "A reliable protocol for processing within IP-routed networks," in *Proc. of Eleventh International Conference on Computer Communications and Networks (ICCCN)*, Miami, FL, Oct. 2002, pp. 84–89.

[14] *Intel Second Generation Network Processor*, Intel Corporation, 2005, http://www.intel.com/design/network/products/npfamily/.

[15] V. Galtier, K. L. Mills, Y. Carlinet, S. Leigh, and A. Rukhin, "Expressing meaningful processing requirements among heterogeneous nodes in an active network," in *Proc. of the Second International Workshop on Software and Performance*, Ottawa, Canada, Sept. 2000, pp. 20–28.

[16] R. Keller, J. Ramamirtham, and T. Wolf, "Active pipes: Program composition for programmable networks," in *Proc. of the 2001 IEEE Conference on Military Communications (MILCOM)*, McLean, VA, Oct. 2001, pp. 962–966.

[17] B. A. Malloy, E. L. Lloyd, and M. L. Souffa, "Scheduling DAG's for asynchronous multiprocessor execution," *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, no. 5, pp. 498–508, May 1994.

[18] S. Y. Choi, J. S. Turner, and T. Wolf, "Configuring sessions in programmable networks," in *Proc. of the Twentieth IEEE Conference on Computer Communications (INFOCOM)*, Anchorage, AK, Apr. 2001, pp. 60–66.

[19] S. Y. Choi and J. S. Turner, "Configuring sessions in programmable networks with capacity constraints," in *Proc. of IEEE International Conference on Communications (ICC)*, Anchorage, AK, May 2003.

[20] N. Weng and T. Wolf, "Profiling and mapping of parallel workloads on network processors," in *Proc. of The 20th Annual ACM Symposium on Applied Computing (SAC)*, Santa Fe, NM, Mar. 2005, pp. 890–896.

[21] P. Ji, Z. Ge, J. Kurose, and D. Towsley, "A comparison of hard-state and soft-state signaling protocols," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, Karlsruhe, Germany, Aug. 2003, pp. 251–262.

[22] T. Wolf and M. A. Franklin, "CommBench – a telecommunications benchmark for network processors," in *Proc. of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Austin, TX, Apr. 2000, pp. 154–162.

[23] D. E. Taylor, J. S. Turner, J. W. Lockwood, and E. L. Horta, "Dynamic hardware plugins: Exploiting reconfigurable hardware for high-performance programmable routers," *Computer Networks*, vol. 38, no. 3, pp. 295–310, Feb. 2002.

[24] T. Wolf, N. Weng, and C.-H. Tai, "Design considerations for network processor operating systems," in *Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS)*, Princeton, NJ, Oct. 2005, pp. 71–80.