1 Customizable In-Network Services

Tilman Wolf
University of Massachusetts Amherst

One of the key characteristics of the next-generation Internet architecture is its ability to adapt to novel protocols and communication paradigms. This adaptability can be achieved through custom processing functionality inside the network. In this chapter, we discuss the design of a network service architecture that can provide custom in-network processing.

1.1 Background

Support for innovation is an essential aspect of the next-generation Internet architecture. With the growing diversity of systems connected to the Internet (e.g., cell phones, sensors, etc.) and the adoption of new communication paradigms (e.g., content distribution, peer-to-peer, etc.), it is essential that not only existing data communication protocols are supported but that emerging protocols can be deployed, too.

1.1.1 Internet Architecture

The existing Internet architecture is based on the layered protocol stack, where application and transport layer protocols processing occurs on end-systems and physical, link, and network layer processing occurs inside the network. This design has been very successful in limiting the complexity of operations that need to be performed by network routers. In turn, modern routers can support link speeds to tens of Gigabits per second and aggregate bandwidths of Terabits per second.

However, the existing Internet architecture also poses limitations on deploying functionality that does not adhere to the layered protocol stack model. In particular, functionality that crosses protocol layers cannot be accommodated without violating the principles of the Internet architecture. But in practice, many such extensions to existing protocols are necessary. Examples include network address translation (where transport layer port numbers are modified by a network layer device), intrusion detection (where packets are dropped in a network layer device based on data in the packet payload), etc.

To avoid this tension in the next-generation Internet, it is necessary to include deployment of new functionality as an essential aspect of the network architecture.

1.1.2 Next-Generation Internet

The main requirement for a next-generation Internet is to provide data communication among existing and emerging networked devices. In this context, existing protocols as well as new communication paradigms need to be supported. Since it is unknown what kind of devices and communication abstractions may be developed in the future, it is essential that a next-generation network architecture provide some level of extensibility.

When considering extensibility, it is important to focus on the data plane of networks (i.e., the data path in routers). The control plane implements control operations that are necessary to manage network state, set up connections, and handle errors. But the data plane is where traffic is handled in the network. To deploy new protocol functionality into the network, it is necessary to modify the way traffic is handled in the data plane.

Extensions in the data plane have been explored in related research and differ in generality and complexity. Some extensions simply allow selection from a set of different functions. Others permit general-purpose programming of new data path operations. What is common among all solutions is the need for custom processing features in the data path of the routers that implement these extensions.

1.1.3 Data Path Programmability

The implementation of data communication protocols is achieved by performing processing steps on network traffic as it traverses a network node. The specific implementation of this processing on a particular system or device can vary from ASIC-based hardware implementation to programmable logic and software on general-purpose processors. In the existing Internet, ASIC-based implementations are common for high-performance routers. This approach is possible since the protocol operations that need to be implemented do not change over time. (RFC 1812, which defines the requirements for routers that implement IP version 4, has been around since 1995.)

In next-generation networks, where new functionality needs to be deployed after routers have already been deployed, it is necessary to include software-programmable devices in the data path. By changing the software that performs protocol processing, new protocol features can be deployed. Thus, programmability is no longer limited to end-systems, but gets pushed into the data path of networks.

1.1.4 Technical Challenges

Programmability in the data path of routers does not only affect the way traffic is processed, but also places new demands on the control infrastructure and thus on the network architecture. The available programmability needs to be managed and controlled during the operation of the network. There are a number of technical challenges that arise in this context:

- Programmable router systems design: Programmable packet processing platforms are necessary to implement custom packet processing. The design and implementation of such systems require high-performance processing platforms that support high-speed I/O and efficient protocol processors to sustain high-bandwidth networking. Secure execution of code, system-level runtime resource management, and suitable programming interfaces need to be developed.
- Control of custom functions: The functionality that is implemented on routers needs to be controlled, as different connections may require different functions. This control may require traffic classification, custom routing, and network-level resource management.
- Deployment of new functions: Custom functions that are developed need to be deployed onto router systems. Code development environments need to be provided. The deployment process can range from manual installation to per-flow code distribution. Trust and security issues need to be resolved as multiple parties participate in code creation, distribution, and execution.

Some of these problems have been addressed in prior and related research.

1.1.5 In-Network Processing Solutions

Several solutions to providing in-network processing infrastructure and control have been proposed and developed. Most notably, active networks provide per-connection and even per-packet configurability by carrying custom processing code in each packet [1]. Several active network platforms were developed [2, 3] differing in the level of programmability, the execution environment for active code, and hardware platform on which they were built. Per-packet programmability as proposed in active networks is very difficult to control. In practical networks, such openness is difficult to align with service providers' need for robust and predictable network behavior and performance. Also, while active networks provide the most complete programming abstraction (i.e., general-purpose programmability), the burden of developing suitable code for particular connection is pushed to the application developer.

A less general, but more manageable way of exposing processing capabilities in the network is with programmable routers. While these systems also provide general-purpose programmability, their control interface differs considerably: system administrators (i.e., the network service provider) may install any

set of packet processing functions, but users are limited to selecting from this set (rather than providing their own functionality) [4, 5].

In the context of next-generation network architecture, programmability in the data (and control) plane appears in network virtualization [6]. To allow the co-existence of multiple networks with different data path functionality, link and router resources can be virtualized and partitioned among multiple virtual networks. Protocol processing for each virtual slice is implemented on a programmable packet processing system.

The technology used in router systems to provide programmability can range from a single-core general-purpose processor to embedded multi-core network processors [7] and programmable logic devices [8, 9]. Studies of processing workloads on programmable network devices have shown that difference to conventional workstation processing are significant and warrant specialized processing architectures [10, 11]. The main concern with any such router system is the need for scalability to support complex processing at high data rates [12].

One of the key challenges for existing solutions is how to provide suitable abstractions for packet processing as part of the network architecture. On end-systems, per-flow configurability of protocol stacks has been proposed as a key element of next-generation networks [13, 14]. For in-network processing, our work proposes the use of network services as a key element in the network architecture.

1.2 Network Services

To provide a balance between generality and manageability, it is important to design the right level of abstractions to access programmability and customization. We discuss how network services provide an abstraction that supports powerful extensibility to the network core while permitting tractable approaches to connection configuration, routing, and runtime resource management.

1.2.1 Concepts

The concept of a "network service" is used to represent fundamental processing operations on network traffic. A network service can represent any type of processing that operates on a traffic stream. Note that the term "service" has been used broadly in the networking domain and often refers to computational features provided on end-system (e.g., on a server). In our context, network service refers to data path operations that are performed on routers in the network. Examples of network services include very fundamental protocol operations as they can be found in TCP (e.g., reliability, flow control) and security protocols (e.g., privacy, integrity, authentication) as well as advanced functionality (e.g., payload transcoding for video distribution on cell phones).

When a connection is established, the sender and receiver can determine a "sequence of services" that is instantiated for this particular communication.

The dynamic composition of sequences of services provides a custom network configuration for connections.

We envision that network services are well-known and agreed-upon functions that are standardized across the Internet (or at least across some of the Internet service providers). New network services could be introduced via a process similar to how protocols are standardized by the Internet Engineering Task Force (IETF). Thus, it is expected that the number of network services that a connection can choose from is in the order of tens, possibly hundreds. The network service architecture does not assume that each application introduces its own service (as it was envisioned in active networks), and therefore a very large number of deployed network services is unlikely. Even with a limited number of network services, the number of possible combinations (i.e., the number of possible sequences of services) is very large. For example, just 10 distinct network services and an average of 4 services per connection lead to thousands of possible service sequences. While not all combinations are feasible or desirable, this estimation still shows that a high level of customization can be achieved while limiting the specific data path processing functions to a manageable number.

To further illustrate the concept of network services, consider the following examples:

1. Legacy TCP: Conventional Transmission Control Protocol (TCP) functionality can be composed from a set of network services, including: reliability (which implements segmentation, retransmission on packet loss, and reassembly), flow control (which throttles sending rate based on available receive buffer size), and congestion control (which throttles sending rate based on observed packet losses). Network service abstractions support modifications to legacy TCP in a straightforward manner. For example, when a connection wants to use a rate-based congestion control algorithm, it simply instantiates the rate-based congestion control network service (rather than the loss-based congestion control service).
2. Forward Error Correction: A connection that traverses links with high bit-error rates may instantiate a forward error correction (FEC) network service. Similar to reliability and flow control, this functionality consists of a pair of network services (the step that adds FEC and the step that checks and removes FEC). This service could be requested explicitly by the end-systems that initiate the connection or it could be added opportunistically by the network infrastructure when encountering lossy links during routing.
3. Multicast and video transcoding: A more complex connection setup example is video distribution (e.g., IPTV) to a set of heterogeneous clients. The transmitting end-system can specify that a multicast network service be used to reach a set of receiving end-systems. In addition, a video transcoding network service can be used to change the video format and encoding. Such a network service is useful when the receiving end-system (e.g., a cell phone) cannot handle the data rate (e.g., due to low-bandwidth wireless links or due

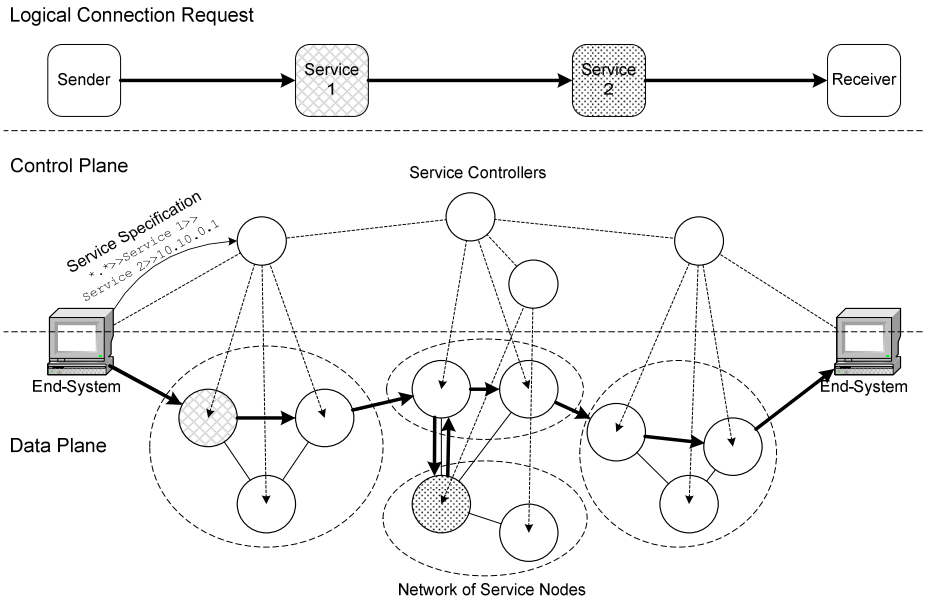


Figure 1.1 Network Service Architecture.

to limited processing capacity). In this scenario, network services are used to perform processing from the network layer to the application layer.

Note that the end-systems that set up a connection using network services do not specify where in the network a particular service is performed. It is up to the network to determine the most suitable placement of the network service processing. Leaving the decision on where and how to instantiate service to the network allows the infrastructure to consider the load on the network, policies, etc., when placing services. End-system applications are not (and should not have to be) aware of the state of the infrastructure and thus could not make the best placement and routing decisions. In some cases, constraints can be specified on the placement of network services (e.g., security-related network services should be instantiated within the trusted local network).

1.2.2 System Architecture

An outline of the network service architecture is shown in Figure 1.1. There are three major aspects that we discuss in more detail: control plane, data plane, and the interface used by end-systems.

The control plane of the network service architecture determines the fundamental structure and operation. Following the structure of the current Internet, which consists of a set of federated networks (i.e., autonomous systems (AS)), our architecture also groups the network into networks that can be managed autonomously. When exchanging control information (e.g., connection setup,

routing information) each AS can make its own local decisions while interacting with other AS globally. In each AS, there is (at least) one node that manages control plane interactions. This "service controller" performs routing and placement computations, and instantiates services for connections that traverse the AS.

In the data plane, "service nodes" implement network service processing on traffic that traverses the network. During connection setup, service controllers determine which nodes perform what service and how traffic is routed between these nodes. Any configuration information that is necessary for performing a network service (e.g., parameters, encryption keys) are provided by the service controller.

The end-system API is used by applications that communicate via the network. Using this interface, communication is set up (similar to how sockets are used in current operating systems) and the desired sequence of services is specified. When initiating a connection setup, the end-system communicates with its local service controller. This service controller propagates the setup request through the network and informs the end-system when all services (and the connections between them) have been set up.

There are several assumptions that are made in this architecture. These are:

- The sequence of services specified by a connection is fixed for the duration of the connection. If a different service sequence is necessary, a new connection needs to be established.
- The underlying infrastructure provides basic addressing, forwarding, etc. There is ongoing research on how to improve these aspects of the next-generation Internet, which is beyond the scope of our work. Progress in this domain can be incorporated in the network service architecture we describe here.
- Routes in the network are fixed once a connection is set up. This can be achieved by tunneling or by using a network infrastructure that inherently allows control of per-flow routes (e.g., PoMo [15], OpenFlow [16]).

Given the fundamental concept of network services and the overarching system architecture, there are a number of important technical problems:

- End-system interface and service specification: The interface used by applications on the end-systems using the network service architecture needs to be sufficiently expressive to allow the specification of an arbitrary sequence of services without introducing too much complexity.
- Routing and service placement: During connection setup, the network needs to determine where network service processing should take place and on what route traffic traverses the network. With constraints on service availability, processing capacity, and link bandwidth, this routing and placement problem is considerable more complex than conventional shortest-path routing.

- Runtime resource management on service nodes: The workload of service nodes is highly dynamic because it is not known a priori what network service processing is used by a connection. Thus, processing resources allocated to particular network services may need to be adjusted dynamically over time. This resource management is particularly challenging on high-performance packet processing platforms that use multi-core processors.

We address solutions to these problems in the following sections. It is important to note that even though the solutions are specific to our network service architecture, similar problems can be solved in other systems that employ in-network processing.

1.3 End-System Interface and Service Specification

When using the network for communication, an end-system application needs to specify which network services should be instantiated. We describe how a "service pipeline" can be used to describe these services and how it can be composed and verified. The service pipeline has been described in our prior work [17]. Our recent work has extended the pipeline concept and integrated it into a socket interface [18]. Automated composition and pipeline verification is described in [19].

1.3.1 Service Pipeline

A connection setup in a network with services is conceptually similar to the process in the current Internet. The main difference is that the set of parameters provided to the operating system not only includes the destination and socket type, but also needs to specify the network services. Since we use a sequence of services, we can provide this information in form of a service pipeline.

The service pipeline is conceptually similar to the pipeline concept in UNIX, where the output of one command can be used as the input of another command by concatenating operations with a '|' symbol. For network services, we use the same concatenation operation (with different syntax) to indicate that the output of one service becomes the input of another. For each service, parameters can be specified. When streams split (e.g., multicast), parentheses can be used to serialize the resulting tree.

Elements of a service specification are:

- Source/sink: Source and sink are represented by a sequence of IP address and port number separated by a ':' (e.g., `192.168.1.1:80`). The source may leave the IP address and/or port unspecified (i.e., `*:*`).
- Network service: The service is specified by its name. If configuration parameters are necessary, they are provided as a sequence in parentheses after the

name (e.g., `compression(LZ)` specifies a compression service that uses the Lempel-Ziv algorithm).
- Concatenation: The concatenation of source, network service(s), and sink is indicated by a '`>>`' symbol.

The service specifications for the three examples given in Section 1.2.1 are:

1. Legacy TCP: `*:*>>reliability_tx(local)>>flowcontrol_tx(local)>>congestioncontrol_tx(local)>>congestioncontrol_rx(remote)>>flowcontrol_rx(remote)>>reliability_rx(remote)>>192.168.1.1:80`
   The three key features of TCP (reliability, flow control, and congestion control), which are provided as separate services, need to be instantiated individually. Each consists of a receive and a transmit portion. The `local` and `remote` parameters indicate constraints on the placement of these services.
2. Forward Error Correction: `*:*>>[FEC_tx>>FEC_rc]>>192.168.1.1:80`
   Forward error correction is similar to the services in TCP. The brackets indicate that it is an optional service.
3. Multicast and video transcoding: `*:*>>multicast(192.168.1.1:5000, video_transcode(1080p,H.264)>>192.168.2.17:5000)`
   The multicast service specifies multiple receivers. Along the path to each receiver different services can be instantiated (e.g., video transcoding).

Service pipelines provide a general and extensible method for specifying network services.

1.3.2 Service Composition

Clearly, it is possible to specify service combinations that are semantically incorrect and cannot be implemented correctly by the network. This problem leads to two questions: (1) how can the system verify that a service specification is semantically correct and (2) how can the system automatically compose correct specifications (given some connection requirements)? The issue of composition of services has been studied in related work for end-system protocol stacks [20] as well as in our prior work on in-network service composition [19].

To verify if a service specification is valid, the semantic description of a service needs to be extended. For a service to operate correctly, the input traffic needs to meet certain characteristics (e.g., contain necessary headers, contain payload that is encoded in a certain way). These characteristics can be expressed as preconditions for that service. The processing that is performed by a service may change the semantics of the input. Some characteristics may change (e.g., set of headers, type of payload), but others may remain unchanged (e.g., delay-sensitive nature of traffic). The combination of input characteristics and modifications performed by the service determines output characteristics. By propagating these characteristics through the service sequence and by verifying that all preconditions are met for all services, the correctness of a service sequence can be verified. The

semantics of a service can be expressed using a variety of languages (e.g., web ontology language (OWL)). The verification operation can be performed by the service controller before setting up a connection.

A more difficult scenario is the automated composition of a service sequence. An application may specify the input characteristics and desired output characteristics of traffic. Based on the formal description of service semantics, a service controller can use AI planning to find a sequence of services that "connects" the input requirements to the output requirements [19]. This feature is particular important when multiple parties contribute to the service sequence (e.g., an ISP may add monitoring or intrusion detection services to a service sequence). In such a case, the originating end-system cannot predict all possible services and create a complete service sequence. Instead, additional services are included during connection setup.

Once a correct and complete service sequence is available, the services need to be instantiated within the network.

1.4 Routing and Service Placement

There are a number of different approaches to determining a suitable routing and placement for a given sequence of services. In our prior work, we have explored how to solve this problem given complete information on a centralized node [21] as well as in a distributed setting [22]. We have also compared the relative performance of these approaches [23]. We review some of these results in this section.

1.4.1 Problem Statement

The service placement problem is stated as follows (from [23]): The network is represented by a weighted graph, $G = (V, E)$, where nodes V correspond to routers and end systems and edges E correspond to links. A node v_i is labeled with the set of services that it can perform, $u_i = \{S_k | \text{service } S_k \text{ is available on } v_i\}$, the processing cost $c_{i,k}$ (e.g., processing delay) of each service, and the node's total available processing capacity p_i. An edge $e_{i,j}$ that connects node v_i and v_j is labeled with a weight $d_{i,j}$ that represents the link delay (e.g., communication cost) and a capacity $l_{i,j}$ that represents the available link bandwidth. A connection request is represented as $R = (v_s, v_t, b, (S_{k_1}, \ldots, S_{k_m}))$, where v_s is the source node, v_t is the destination node, b is the requested connection bandwidth (assumed to be constant bit rate), and $(S_{k_1}, \ldots, S_{k_m})$ is an ordered list of services that are required for this connection. For simplicity, we assume that the processing requirements for a connection are directly proportional to the requested bandwidth b. For service S_k, a complexity metric $z_{i,k}$ defines the amount of computation that is required on node v_i for processing each byte transmitted.

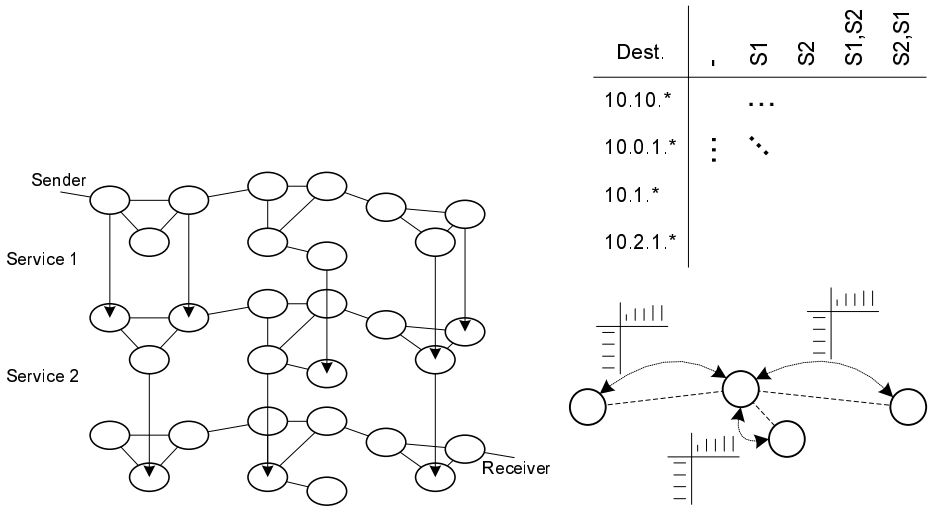
Given a network G and a request R, we need to find a path for the connection such that the source and the destination are connected and all required services can be processed along the path. The path is defined as $P = (E^P, M^P)$ with a sequence of edges, E^P, and services mapped to processing nodes, M^P: $P = ((e_{i_1,i_2}, \ldots, e_{i_{n-1},i_n}), (S_{k_1} \rightarrow v_{j_1}, \ldots, S_{k_m} \rightarrow v_{j_m}))$, where $v_{i_1} = v_s$, $v_{i_n} = v_t$, $\{v_{j_1}, \ldots, v_{j_m}\} \subset \{v_{i_1}, \ldots, v_{i_n}\}$ and nodes $\{v_{j_1}, \ldots, v_{j_m}\}$ are traversed in sequence along the path. The path P is valid if (1) all edges have sufficient link capacity (i.e., $\forall e_{x,y} \in E^P, l_{x,y} \geq (b \cdot t)$, assuming link $e_{x,y}$ appears t times in E^P), and (2) all service nodes have sufficient processing capacity (i.e., $\forall S_{k_x} \rightarrow v_{j_x} \in M^P, p_{j_x} \geq \sum_{y|S_{k_y} \rightarrow v_{j_x} \in M^P} b \cdot z_{j_x,k_y}$).

To determine the quality of a path, we define the total cost $C(P)$ of accommodating connection request R as the sum of communication cost and processing cost: $C(P) = \left(\sum_{x=1}^{n-1} d_{i_x,i_{x+1}}\right) + \left(\sum_{\{(j_x,k_x)|S_{k_x} \rightarrow v_{j_x} \in M^P\}} c_{j_x,k_x}\right)$. In many cases, it is desirable to find the optimal connection setup. This optimality can be viewed (1) as finding the optimal (i.e., least-cost) allocation of a single connection request or (2) as finding the optimal allocation of multiple connection requests. In the latter case, the optimization metric can be the overall least cost for all connections or the best system utilization, etc. We focus on the former case of a single connection request.

It was shown in [21] that finding a solution to a connection request in a capacity-constrained network can be reduced to the traveling salesman problem, which is known to be NP-complete. Therefore, we limit our discussion to the routing and placement problem without constraints. Using heuristics, the solutions can be extended to consider capacity constraints.

1.4.2 Centralized Routing and Placement

The centralized routing and placement solution was first described in [21]. The idea is to represent both communication and processing in a single graph and to use conventional shortest-path routing to determine an optimal placement. This solution requires that a single cost metric is used for both communication and processing cost. To account for processing, the network graph is replicated for each of the m processing steps in the connection request (as illustrated in Figure 1.2(a)). Thus, a total of $m + 1$ network graphs ("layers") exist. The top graph, layer 0, represents communication that is performed before the first network service is performed. The bottom graph, layer m, represents communication after all processing steps have been completed. To traverse from one layer to another, vertical edges between layers are added. These edges can only connect the same nodes in neighboring layers. The existence of a (directed) vertical edge indicates that the necessary service processing step to reach the next layer is available on that node. The cost of traversing that edge is the cost of processing on that node.



(a) Centralized Routing on Service Nodes.

(b) Distributed Routing on Service Controllers.

Figure 1.2 Routing and Placement in Network Service Architecture.

Routing is achieved by finding the least cost path in the layered graph between the source node on layer 0 and the destination node on layer m. This path is projected back into a single layer with vertical edges indicating placement for network service processing.

The algorithm is guaranteed to find the optimal path for a network without capacity constraints. The computational cost is that of running Dijkstra's shortest path algorithm on the layered graph. Since the layered graph is $m+1$ times the size of the original network graph, the complexity is $\mathcal{O}(|m||E| + |m||V| + |m||V| \log(|m||V|))$.

1.4.3 Distributed Routing and Placement

One of the drawbacks of the centralized layered graph solution is the need for complete knowledge of all network links. In an Internet-scale deployment it is unrealistic to assume that such information is available. Thus, we also present a distributed approach, where information can be aggregated and nodes have a limited "view" of the network. This algorithm has been described in [22].

The distributed routing and placement algorithm uses a dynamic programming approach similar to distance vector routing [24]. Let $c_v^{k_1,\ldots,k_m}(t)$ denote the cost of the shortest path from node v to node t where the requested services $S_{k_1}, \ldots, S_{k_m}$ are performed along the way. For shortest path computation (i.e., no services), we use the notation $c_v^-(t)$. Thus, a node v can determine the least-cost path by considering to process i $(0 \leq i \leq m)$ services and forwarding the request to any

neighboring node n_v ($n_v \in \{x \in V | e_{v,x} \in E\}$):

$$c_v^{k_1,\ldots,k_m}(t) = \min_{0 \le i \le m} \left(\sum_{l=1}^{i} c_v^{k_l}(v) + \min_{n_v} \left(c_v^-(n_v) + c_{n_v}^{k_{i+1},\ldots,k_m}(t) \right) \right).$$

The argument i on the right side determines how many of the m services that need to be performed should be processed on node v. Note that if $i = 0$, no service is processed, i.e., $\sum_{l=1}^{i} c_v^{k_l}(v) = 0$. If $i = m$, all the services are processed on node v, i.e., $c_{n_v}^{k_{i+1},\ldots,k_m}(t) = c_{n_v}^-(t)$. The argument n_v determines to which neighbor of v the remaining request should be sent.

To acquire the necessary cost information, nodes exchange a "service matrix" with their neighbors (as illustrated in Figure 1.2(b)). This matrix contains costs for all destinations and all possible service combinations. Since the number of service combinations can be very large, a heuristic solution has been developed that only uses cost information for each individual service. This approach is discussed in detail in [22].

1.5 Runtime Resource Management

The network service architecture presents a highly dynamic environment for the processing system on which services are implemented. Each connection may request a different service sequence, which can lead to variable demand for any particular service. This type of workload is very different from conventional IP forwarding, where each packet requires practically the same processing steps. While operating systems can provide a layer of abstraction between hardware resources and dynamic processing workloads, they are too heavy-weight for embedded packet processors that need to handle traffic at Gigabit per second data rates. Instead, a runtime system that is specialized for dealing with network service tasks can be developed. Of particular concern is to handle processing workloads on multi-core packet processing systems. We discuss the task allocation system developed in our prior work [25].

1.5.1 Workload and System Model

The workload of a router system that implements packet forwarding for the current Internet or service processing for next-generation networks can be represented by a task graph. This task graph is a directed acyclic graph of processing steps with directed edges indicating processing dependencies. Packet processing occurs along one path through this graph for any given packet. Different packets may traverse different paths. An example of such a graph representation of packet processing is the Click modular router abstraction [26].

As discussed above, changes in traffic may cause more or less utilization along any particular path in the graph and thus more or less utilization for any partic-

ular processing step. To determine the processing requirements at runtime, it is necessary to do runtime profiling and track (at least) the following information:

- Processing requirements for each task.
- Frequency of task usage.

In our runtime system prototype, we represent the processing requirement as a random variable S_i, which reflects the processing time distribution of task t_i. For any given packet, the task service time is s_i. The frequency of usage is represented by the task utilization $u(t_i)$, which denotes the fraction of traffic traversing task t_i.

Based on this profiling information, the runtime system determines how to allocate resources to tasks.

1.5.2 Resource Management Problem

The formal problem statement for runtime management of multi-core service processors is as follows (from [25]): Assume we are given the task graph of all subtasks in all applications by T task nodes $t_1, \ldots, t_T$ and directed edges $e_{i,j}$ that represent processing dependencies between tasks t_i and t_j. For each task, t_i, its utilization $u(t_i)$ and its service time S_i are given. Also assume that we represent a packet processing system by N processors with M processing resources on each (i.e., each processor can accommodate M tasks and the entire system can accommodate $N \cdot M$ tasks). The goal is to determine a mapping m that assigns each of the T tasks to one of N processors: $m : \{t_1, \ldots, t_T\} \rightarrow [1, N]$. This mapping needs to consider the constraint of resource limitations: $\forall j, 1 \leq j \leq N : |\{t_i | m(t_i) = j\}| \leq M$.

The quality of the resource allocation (i.e., mapping) can be measured by different metrics (e.g., system utilization, power consumption, packet processing delay, etc.). Our focus is to obtain a balanced load across processing components, which provides the basis for achieving high system throughput.

1.5.3 Task Duplication

One of the challenges in runtime management is the significant differences in processing requirements between different tasks. Some tasks are highly utilized and complex and thus require much more processing resources than tasks that are simple and rarely used. Also, high-end packet processing systems may have more processor cores and threads than there are tasks.

To address this problem, we have developed a technique called "task duplication" that exploits the packet-level parallelism inherent to the networking domain. Task duplication provides a straightforward way to distributing processing tasks onto multiple processing resources. For the discussion, we assume processing is stateless between packets. If stateful processing is performed, the

runtime system can ensure that packets of the same flow are sent to the same instance of the processing task.

Task duplication creates additional instances of tasks with high work requirements. The amount of work, w_i a task performs is the product of the processing requirements for a single packet and the frequency with which the task is invoked: $w_i = u(t_i) \cdot E[S_i]$. This amount of work can be reduced if the number of task instances is increased. If a task is duplicated such that there are d_i instances and traffic is spread evenly among these instances, then the amount of utilization for each instance decreases to $u(t_i)/d_i$. Thus, the effective amount of work per instance is $w_i' = \frac{u(t_i)}{d_i} \cdot E[S_i]$. Therefore, a more balanced workload can be obtained by greedily duplicating the task with the highest amount of work until all $M \cdot N$ resources are filled with tasks. This also allows the use of all resources if there are fewer tasks than resources.

Note that the work equation also shows that the differences in the amount of work per task are not only due to the inherent nature of the task (i.e., the expected service time $E[S_i]$), but also due to the dynamic nature of the network (i.e., the current utilization of the task $u(t_i)$). Thus, the imbalance between tasks cannot be removed by achieving a better (i.e., more balanced) offline partitioning, and there is always need to adapt to current conditions at runtime.

1.5.4 Task Mapping

Once the tasks and their duplicates are available, the mapping of tasks to processors needs to be determined. There are numerous different approaches to placing tasks. When using tasks with vast differences in the amount of work that they need to perform, then a mapping algorithm needs to take care in co-locating complex tasks with simple tasks. If too many complex tasks are placed on a single processor, then that system resource becomes a bottleneck and the overall system performance suffers. Solving this type of packing problem is NP-complete [27].

The benefit of having performed task duplication is that most tasks require nearly equal amounts of work. Thus, the mapping algorithm can place any combination of these tasks onto a processor without the need for considering difference in processing work. Instead, secondary metrics (e.g., locality of communication) can be considered to make mapping decisions. We have shown that a depth-first search to maximize communication locality is an effective mapping algorithm. Our prototype runtime system that uses duplication and this mapping strategy shows an improvement in throughput performance over a system with conventional symmetric multiprocessing (SMP) scheduling provided by an operating system [25]. More recent work considers not only processing resource allocation, but also memory management [28]. In particular, the partitioning of data structures among multiple physical memories with different space and performance characteristics is an important issue. Static partitioning used in tradi-

tional packet processing systems is not sufficient for the same reasons that static processing allocations cannot adapt to changing networking conditions.

Overall, runtime management of processing resources is an important aspect of packet processing platforms in next-generation networks – especially as the complexity and diversity of such services continues to increase.

1.6 Summary

The functionality provided by the networking infrastructure in the next-generation Internet architecture encompasses not only forwarding, but also more advanced protocol and payload processing. A key challenge is find suitable abstractions that allow end-systems to utilize such functionality, while maintaining manageability and controllability from the perspective of service providers. We presented an overview of a network service architecture that uses network services as fundamental processing steps. The sequence of services that is instantiated for each connection can be customized to meet the end-system application's needs. We discussed how service specifications can be used to express these custom processing needs and how they can be translated into a constrained mapping problem. Routing in networks that support services is a problem that needs to consider communication and processing costs. We presented two solutions, one centralized and one distributed, to address the routing problem. We also presented how runtime management on packet processing systems can ensure an effective utilization of system resources.

The use of network service abstractions to describe in-network processing service can be used beyond the work presented here. For example, when developing virtualized network infrastructure, network service specifications can be used to describe data path requirements for virtual slices.

In summary, in-network processing services are an integral part of the next-generation Internet infrastructure. The work we presented here can provide one way of making such functionality possible.

References

[1] Tennenhouse DL, Wetherall DJ. Towards an Active Network Architecture. ACM SIGCOMM Computer Communication Review. 1996 Apr;26(2):5–18.

[2] Tennenhouse DL, Smith JM, Sincoskie WD, Wetherall DJ, Minden GJ. A Survey of Active Network Research. IEEE Communications Magazine. 1997 Jan;35(1):80–86.

[3] Campbell AT, De Meer HG, Kounavis ME, Miki K, Vincente JB, Villela D. A Survey of Programmable Networks. ACM SIGCOMM Computer Communication Review. 1999 Apr;29(2):7–23.

[4] Wolf T. Design and Performance of Scalable High-Performance Programmable Routers. Department of Computer Science, Washington University. St. Louis, MO; 2002.

[5] Ruf L, Farkas K, Hug H, Plattner B. Network Services on Service Extensible Routers. In: Proc. of Seventh Annual International Working Conference on Active Networking (IWAN 2005). Sophia Antipolis, France; 2005.

[6] Anderson T, Peterson L, Shenker S, Turner J. Overcoming the Internet Impasse through Virtualization. Computer. 2005 Apr;38(4):34–41.

[7] Wolf T. Challenges and Applications for Network-Processor-Based Programmable Routers. In: Proc. of IEEE Sarnoff Symposium. Princeton, NJ; 2006.

[8] Hadzic I, Marcus WS, Smith JM. On-the-fly Programmable Hardware for Networks. In: Proc. of IEEE Globecom 98. Syndey, Australia; 1998.

[9] Taylor DE, Turner JS, Lockwood JW, Horta EL. Dynamic Hardware Plugins: Exploiting Reconfigurable Hardware for High-Performance Programmable Routers. Computer Networks. 2002 Feb;38(3):295–310.

[10] Crowley P, Fiuczynski ME, Baer JL, Bershad BN. Workloads for Programmable Network Interfaces. In: IEEE Second Annual Workshop on Workload Characterization. Austin, TX; 1999.

[11] Wolf T, Franklin MA. CommBench – A Telecommunications Benchmark for Network Processors. In: Proc. of IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). Austin, TX; 2000. p. 154–162.

[12] Wolf T, Turner JS. Design Issues for High-Performance Active Routers. IEEE Journal on Selected Areas of Communication. 2001 Mar;19(3):404–409.

[13] Dutta R, Rouskas GN, Baldine I, Bragg A, Stevenson D. The SILO Architecture for Services Integration, controL, and Optimization for the Future Internet. In: Proc. of IEEE International Conference on Communications (ICC). Glasgow, Scotland; 2007. p. 1899–1904.

[14] Baldine I, Vellala M, Wang A, Rouskas G, Dutta R, Stevenson D. A Unified Software Architecture to Enable Cross-Layer Design in the Future Internet. In: Proc. of Sixteenth IEEE International Conference on Computer Communications and Networks (ICCCN).

Honolulu, HI; 2007.

[15] Calvert KL, Griffioen J, Poutievski L. Separating Routing and Forwarding: A Clean-Slate Network Layer Design. In: Proc. of Fourth International Conference on Broadband Communications, Networks, and Systems (BROADNETS). Raleigh, NC; 2007. p. 261–270.

[16] McKeown N, Anderson T, Balakrishnan H, Parulkar G, Peterson L, Rexford J, et al. OpenFlow: enabling innovation in campus networks. SIGCOMM Computer Communication Review. 2008 Apr;38(2):69–74.

[17] Keller R, Ramamirtham J, Wolf T, Plattner B. Active Pipes: Program Composition for Programmable Networks. In: Proc. of the 2001 IEEE Conference on Military Communications (MILCOM). McLean, VA; 2001. p. 962–966.

[18] Shanbhag S, Wolf T. Implementation of End-to-End Abstractions in a Network Service Architecture. In: Proc. of Fourth Conference on emerging Networking EXperiments and Technologies (CoNEXT). Madrid, Spain; 2008.

[19] Shanbhag S, Huang X, Proddatoori S, Wolf T. Automated Service Composition in Next-Generation Networks. In: Proc. of The International Workshop on Next Generation Network Architecture (NGNA) held in conjunction with The IEEE 29th International Conference on Distributed Computing Systems (ICDCS). Montreal, Canada; 2009. .

[20] Vellala M, Wang A, Rouskas GN, Dutta R, Baldine I, Stevenson D. A Composition Algorithm for the SILO Cross-Layer Optimization Service Architecture. In: Proc. of the Advanced Networks and Telecommunications Systems Conference (ANTS). Mumbai, India; 2007.

[21] Choi SY, Turner JS, Wolf T. Configuring Sessions in Programmable Networks. In: Proc. of the Twentieth IEEE Conference on Computer Communications (INFOCOM). Anchorage, AK; 2001. p. 60–66.

[22] Huang X, Ganapathy S, Wolf T. A Scalable Distributed Routing Protocol for Networks with Data-Path Services. In: Proc. of 16th IEEE International Conference on Network Protocols (ICNP). Orlando, FL; 2008.

[23] Huang X, Ganapathy S, Wolf T. Evaluating Algorithms for Composable Service Placement in Computer Networks. In: Proc. of IEEE International Conference on Communications (ICC). Dresden, Germany; 2009.

[24] Bellman R. On a Routing Problem. Quarterly of Applied Mathematics. 1958 Jan;16(1):87–90.

[25] Wu Q, Wolf T. On Runtime Management in Multi-Core Packet Processing Systems. In: Proc. of ACM/IEEE Symposium on Architectures for Networking and Communication Systems (ANCS). San Jose, CA; 2008.

[26] Kohler E, Morris R, Chen B, Jannotti J, Kaashoek MF. The Click Modular Router. ACM Transactions on Computer Systems. 2000 Aug;18(3):263–297.

[27] Johnson DS, Demers AJ, Ullman JD, Garey MR, Graham RL. Worst-Case Performance Bounds for Simple One-Dimensional Packing Algorithms. SIAM Journal on Computing. 1974 Dec;3(4):299–325.

[28] Wu Q, Wolf T. Runtime Resource Allocation in Multi-Core Packet Processing Systems. In: Proc. of IEEE Workshop on High Performance Switching and Routing (HPSR). Paris, France; 2009.