Public Review for

Towards Slack-Aware Networking

Fahad Dogar

Some of today's communication patterns, such as those that occur between machines, do not necessarily require low latency nor fine-grained synchronicity between senders and receivers. From this observation, the author proposes a *"Slack Aware"* networking architecture where applications can express their latency and synchronicity requirements (or lack of). By enabling the network to delay some communications, it could be possible to optimise its behaviour, e.g. by batching communications from different applications together.
The reviewers debated over this paper. On one hand, they liked the proposed idea if it remains preliminary and has not been evaluated in details. On the other hand, some elements of the proposed idea reminded the reviewers of older architectures such the email system or UUCP and newer solutions such as ICN or NDN. This is not suprising as many proposed architectures reuse ideas from earlier ones. In the end, the reviewers agreed that even if the paper did not contain a detailed evaluation and did not address all issues, it was worthwhile to document it for the community to let other researchers to further explore this idea.

Public review written by
Olivier Bonaventure
UCLouvain

Towards Slack-Aware Networking

Fahad R. Dogar
Tufts University, USA
fahad@cs.tufts.edu

ABSTRACT

We are moving towards an Internet where most of the packets may be consumed by *machines* – set-top-boxes or smartphone apps prefetching content, Internet of Things (IoT) devices uploading their data to the cloud, or data centers doing geo-distributed replication. We observe that such machine centric communication can afford to have *slack* built into it: every packet can be marked as to when it will be consumed in future. Slack could be anywhere from seconds to hours or even days. In this paper, we make a case for slack-aware networking by illustrating slack opportunities that arise for a wide range of applications as they interact with the cloud and its pricing models (e.g., spot pricing). We also sketch the design of *SlackStack*, a network stack with explicit support for slack at multiple levels of the stack, from a slack-based interface to slack-aware optimizations at the transport and network layers.

CCS Concepts

• **Networks** → Network Architectures;

Keywords

architecture; slack; delay-tolerant; cloud-pricing; IoT

1. INTRODUCTION

Today's Internet is centered around *human-centric* communication: there is a waiting user behind every communication (e.g., web request) and the network tries to deliver the packets as soon as possible. In future, it is possible that most of the packets will be consumed by *machines* – set-top-boxes or smart-phone apps pre-fetching content, Internet of Things (IoT) devices uploading their data to the cloud, or data centers (DC) doing geo-distributed replication.

We posit that such machine centric communication can afford to have *slack* built into it: every packet can be marked as to *when* it will be consumed in future. Slack could be anywhere from seconds to hours or even days. For example, a social network application prefetching photos may have a slack of seconds or minutes; an IoT device uploading data to the cloud may have a slack of few minutes to hours; and a set-top box prefetching videos for offline viewing may even have a slack of few days.

Information about slack allows the network to do *slack-aware networking*: the network can reduce cost of transferring data by scheduling data transmissions at non-peak times [23, 29], or by compressing, batching, and multicasting data, as packets need not be sent immediately. The network can also use different power saving modes, with energy savings even for slack as small as tens of millisconds or seconds [16, 31]. From a user's perspective, slack-aware networking can reduce cost of network transfers, which facilitates aggressive prefetching of content, thus leading to potential improvement in performance and availability.

While past work has shown the benefits of slack-aware networking in limited settings [29,31,37], what makes slack-aware networking particularly attractive now is the emergence of the *cloud*. Most popular applications have a footprint in the cloud, so we can realize benefits of slack-aware networking without requiring any explicit support from ISPs. More importantly, the cloud offers a range of performance vs. cost options, which are amenable to slack-aware networking. For example, cloud usage cost could vary 2-37x across *time* (e.g., due to variation in spot pricing) and *space* (e.g., due to variable costs across DC locations) [1, 3]. The spatial slack is *synergistic* with temporal slack: the temporal slack gives us (spatial) flexibility in choosing which cloud to use as an end-point or for in-network services (e.g., a far-away but cheaper DC) while the new spatial options can decouple end-points through use of in-network storage, which could in turn create new opportunities for temporal slack (e.g., transferring data during off-peak times).

Unfortunately, today's TCP/IP network stack is ill-suited to exploit these opportunities. From a low level application-network interface (socket API) to a synchronous communication model (TCP) and use of locations as addresses (IP), today's TCP/IP stack lacks support to fully exploit the spatial and temporal slack opportunities. On the other extreme are disruption tolerant networks (DTNs) [18] that deal with *arbitrary* disruptions: they do not optimize for slack in "well-connected" networks like the Internet. We, therefore, call for a new network stack with explicit support for slack, a network stack that allows applications to seamlessly move from an interactive mode (no slack) to regimes where slack could be anywhere from milliseconds to hours or days.

As a first step in this direction, we propose *SlackStack*, a network stack with support for slack at multiple levels of the system – from the application level, in the form of new APIs, to slack-aware mechanisms at the transport and network layers. *SlackStack* exposes *slack* information – both temporal and spatial – throughout the network stack, and uses the *mailbox* abstraction to allow producers and consumers to effectively control the slack optimizations as per their requirements. The idea of a mailbox is inspired by the postal system – like physical mailboxes, a *SlackStack* mailbox decouples the producer and consumer using in-network

storage. This decoupling occurs at multiple levels and target different goals: privacy and insertion of services at the end-to-end (transport) level vs. higher throughput and lower energy consumption at the per-hop (router) level. These differences lead to different storage implementations for the mailboxes: globally visible, cloud-based mailboxes at the transport level, and temporary memory regions at the per-hop (router) level.

In the remainder of this paper, we elaborate on our case for slack-aware networking (§2), present the sketch of *SlackStack* and how it leads to rich opportunities at multiple levels of the network stack (§3), and, finally, how our proposal builds upon, and relates to, a large body of prior work (§4).

2. A CASE FOR SLACK-AWARE NETWORKING

Slack-Aware Applications. We discuss four broad classes of applications that can benefit from slack-aware networking.

1. *Video-on-Demand (VoD)* can consume more than 50% of Internet bandwidth at peak times [4]; most of this traffic is not for live content, so it can potentially benefit from slack-aware networking. Many services already offer *offline* viewing, with some providing users with explicit control over *when* they want to watch a video (e.g., PlayLater [7]) while others try to actively prefetch videos that the user is likely to watch in future [2]. The slack time of VoD could thus vary from zero (real-time) to hours or even days.
2. *Social networking* applications like Facebook have a huge user base. We posit that several features of these applications (e.g., photo sharing) can benefit from slack-awareness in two ways. First, when a content is published, the user may be offline (e.g., she may be sleeping), so the content could be transferred in a slack-aware fashion to her device. Second, even if the user is online, adding a little slack (e.g., on the order of few seconds) to when the data is made visible to the user may not be noticeable.
3. *IoT applications* can upload huge amounts of data to the cloud and their slack requirements could vary from real-time to minutes or hours. For example, home monitoring cameras (e.g., Nest [6]) stream data to the cloud for archival storage but also allow real-time monitoring. For such applications, the slack value could vary depending on whether the user is actively watching the video feed, or any activity in the video has been detected or not.[1]
4. *Inter-Data Center Replication.* For performance, availability, and fault-tolerance, many cloud applications replicate data across multiple DCs [34]. The slack value for replicated traffic could vary significantly, depending on the application requirements (e.g., whether the replication is on the critical path of user requests or not).

These slack-aware applications also rely heavily on the the cloud – they either run inside the cloud or can gain additional benefits with the help of cloud services [21, 22].

Cloud Pricing Models. We analyze the cost offerings of the three major cloud providers (Amazon, Microsoft, and Google) and observe significant price variation in both the spatial (across DCs or resources) and temporal dimensions. For example, for different DC locations in Amazon, the compute and network costs could be 1.6x to 2.7x higher compared to the locations in US while the regular instances could be up to 37x more expensive compared to the cheapest spot price instance of the same type. Microsoft and Google have similar price differentials: Microsoft's compute and network costs could vary by 1.5x to 2x respectively across DC locations while Google's compute and network costs could vary by 1.4x and 1.9x, respectively. While Microsoft doesn't offer any spot pricing, Google offers *preemptable* virtual machines (VM) that offer up to 70% discount over normal VMs [5].

Potential Gains. Based on the above four applications, we pick four specific scenarios to highlight potential benefits of slack-aware networking when it is used in today's cloud-based world. We broadly put all these scenarios under machine-centric communication as data is retrieved by machines or smartphone "apps" working on behalf of users.

Prefetching Photos. Alice opens the photo album of her friend. As she views the first photo, her social networking app decides to prefetch the subsequent photos, requesting them with a slack of few seconds. Normally, the network downloads the photos from the nearest DC, but for photos with slack, it can download them from a far-away but cheaper DC. If most of the requests could be served from the cheaper DC, it could significantly reduce the cost for the application provider.

Home Video Surveillance. For this scenario, slack-aware networking can help in three ways. First, slack-aware scheduling could allow video surveillance to *co-exist* well with other home traffic (e.g., web browsing). Second, it can reduce the *infrastructure requirement* of processing and storing the video feeds inside the cloud: data with slack can be batched and processed using a slower VM, in an energy efficient manner, and using a cheaper (but slower) storage option. Third, user app can decide to consume a *subset* of the total data, by specifying a time range or other criterion to filter, thereby reducing its download traffic with the help of the network.

Overnight Backup. Consider a bulk overnight backup transfer from a DC in Asia to a DC in US. Slack-aware networking can reduce cost of transferring data by exploiting lower prices that are available during the slack interval – either because of spot-pricing or 95th percentile pricing used by ISPs [3, 29]. To transfer data, we can wait for a time when the cost is lower for *both* DCs or in some cases it may make sense to even use an intermediate DC (e.g., a DC in Europe) as a store-and-forward point.

Slack-aware VoD. Consider an app that shows the user recommendation on new movies along with multiple options to download – options with slack being cheaper compared to the option of downloading immediately. Suppose the user decides to watch the movie after few hours, but during this time she visits her friend and decides to watch the movie at her friend's home. As the data may be in "transit", the slack-aware network can provide user with the updated cost to reroute the transfer to her friend's mailbox, which will likely be lower than downloading the movie twice. While this scenario can also reap the benefits of the previous scenarios, it highlights: i) network impacting user behavior by giving slack-based options and ii) user changing her transfer options based on new preferences – a flexibility lacking in

[1]These devices typically run computer vision algorithms on the device (often supplemented by additional computation in the cloud) to detect activities in the scene.

today's Internet because data is transferred *immediately*.

Requirements. The above scenarios help us identify four key application requirements for slack-aware networking.

1. *Network should expose slack trade-offs to the user/application.* Exposing the different slack options and the cost associated with them may impact application/user choices because there is an inherent flexibility in many of these applications (e.g., VoD scenario). This has implication on the human-machine interface as well as the application-network interface.
2. *Supporting a wide range of slack times.* The amount of slackness applications can tolerate may vary significantly, both across applications, as well as within an application. Applications should, therefore, have fine-grained control over setting a slack value for their network transfers. Similarly, the network stack should have mechanisms to deal with a wide range of slack values, exploiting opportunities created by slack in the *time* dimension (i.e., *when* transfers happen) and the *space* dimension (i.e., *which* DC is used as an end-point or an intermediary).
3. *Slack times may change during transfers.* This has implication on both the interface as well as the underlying slack-aware mechanisms. The network should have controls for changing the slack value dynamically based on changed circumstances.
4. *Respecting Existing and New Tussles.* Supporting the above slack optimizations will bring forth existing tussles as well as create new ones [11]. For example, it may entail requiring information that the end-points may not be willing to share with each other or with the network. For users, this could be their location or the devices they are going to use during the slack time while for the application providers it could be the spatial options for data delivery, such as the intermediate DC locations and their costs. The stack should provide suitable controls and trade-offs that could gracefully handle these tussles.

3. SLACKSTACK

3.1 Overview

We present the Slack-Aware Network Stack (*SlackStack*), which combines the concept of slack with well-known architectural building blocks to enable a wide range of slack-based optimizations. In *SlackStack*, the entire stack is aware of the slack options, both temporal and spatial, and this information is pushed down from the application to lower layers, in both the *vertical* (across layers) and *horizontal* (across roles) directions.

Mailboxes. Inspired by the postal system, *SlackStack* uses mailboxes which leverage in-network storage to decouple the producer and consumer. They support two operations: *push*, which enables the producer to control slack-based optimizations, from the source of the data to the mailbox, and a *pull* operation which puts the consumer in-charge of the optimizations, from the mailbox to the consumer device. Most existing architectures either support a pull mode (e.g., CCN [18]) or a push mode (e.g., DTN [18]), but *SlackStack* uses both in order to provide mechanisms to deal with the increased "tussles" expected in a slack-based Internet.

Mailboxes have different realizations at different levels of the stack based on the relevant concerns: privacy and insertion of services at the end-to-end level vs. higher throughput and lower energy consumption at the per-hop (router) level. These differences lead to different storage abstractions for the mailboxes: at the end-to-end level, we have globally visible, mailboxes with persistent storage that can be hosted in the cloud (or cloudlets [32]) and are named using self-certifying identifiers (MID) [9,20]; at the router level, we can have temporary *memory regions*, where neighboring routers in a cluster can directly write packets to, without involving the host processor (e.g., using RDMA [17,26]).

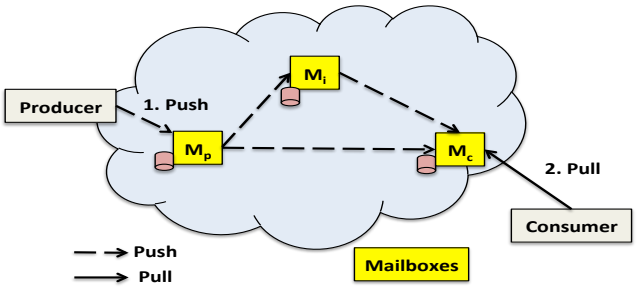

Figure 1: *SlackStack* example showing a consumer getting data from a publisher through mailboxes.

Building Blocks. *SlackStack* leverages three other important building blocks. First, TCP's synchronous communication model and end-to-end semantics are limiting, so *SlackStack* requires a hop-by-hop transport (e.g., Tapa [15]) which decouples the end-points using in-network storage, while also supporting flexible application semantics (e.g., end-to-end vs. per-hop acknowledgements). Second, *SlackStack* uses a pub/sub API [19] between the application and the network, based on `push()` and `pull()`. Third, *SlackStack* uses application data unit (ADU) [10] as the data granularity – ADU could be defined in a flexible manner by the application – as an application chunk or the entire file (e.g., photo). The use of ADUs and a pub/sub API provides a much higher level of abstraction compared to today's socket-based API, which simplifies the use of slack-aware optimizations.

End-to-End Transfer. Figure 1 shows an example of how data is transferred between the producer and the consumer using one or more mailboxes. In this example, the consumer mailbox is hosted in a cloud close to the user; data is pushed to this mailbox by the producer, and later pulled by the consumer. We present *SlackStack* as an overlay on top of today's Internet, but most of the optimizations apply even if *SlackStack* is deployed as a native architecture.

Push Phase. The producer application uses a `push()` call to write data to its own mailbox. Transport figures out which consumer mailbox(es) the data needs to go based on prior registrations. The MID of the consumer mailbox is resolved to a network address using a key-value based, global name resolution service (e.g., Auspices [33]). Given the destination, slack value, and information about the network graph (in-network storage, costs, etc), the *routing module* computes a route (i.e., which intermediate mailboxes should be used) and schedule (i.e., when should they transfer data).

Subsequent mailboxes may update the schedule depending on new information (e.g., changes in MID resolution or network costs).

On the *forwarding plane*, *SlackStack* uses hop-by-hop reliable transfers to push the data to the mailbox. While we focus on a store-and-forward transfer model in this paper, *SlackStack* can also support a cut-through transfer mode, in order to speed up transfers at the cost of limiting the use of in-network services that operate on complete ADUs (e.g., virus scanners). Our software router based forwarding will allow routers (and middleboxes) in a cluster/DC to write directly to specific memory regions of their neighbors based on the slack-value of the data – using help from neighbor's dedicated packet reception processor or through interconnects like RDMA [26]. Data is finally stored in the consumer mailbox and a delivery notification is sent to the consumer transport.

Pull Phase and Advance Mailbox Services. Consumer makes a `pull()` call to retrieve data from its mailbox. The pull phase puts the onus of getting the data on the consumer (similar to CCN [25]); it could get the data at any time, on any device, using a transport mechanism that is optimized for the particular scenario. Consumer may want to retrieve different data compared to what the producer originally sent, as user preferences may have changed during the slack time. *SlackStack* supports a *query* interface with the `pull()` mode that allows filtering of ADUs based on a query, and allows in-network services (e.g., transcoding) to be added *on-the-fly* in the forwarding path through dynamic MID resolution. The resolution service returns a *stack* of MIDs to support indirection (similar to i3 [35]) but the resolution can be updated at every hop (similar to DTN [18]). This id resolution can also be context specific (e.g., different return value based on who is resolving) [33].

We now discuss the key challenges and opportunities at different levels of *SlackStack*.

3.2 Slack-Aware Interface

Our pub/sub API provides a high level abstraction for data transfers. The API should also support a number of other requirements: it should be *bidirectional*, allowing not only the applications to set the slack value but also enabling the network to provide a set of slack vs. cost tradeoffs to the application (similar to the VoD scenario). It should also support *adaptability* as slack preferences may change, and should also provide suitable *controls* for specifying slack, privacy, and data sharing requirements.

We propose the use of a directed-acyclic graph (DAG) to express the communication requirements between the application and the network. A DAG can capture the spatial (cloud replicas) and temporal (pricing options) dimensions of slack-based communication. End-points and mailboxes can be represented as nodes in the DAG while edges will correspond to the flow of information. For each operation, the application can specify exact mailbox to be used or can let the network choose the specific replica while still meeting the slack requirements. The application also specifies the slack requirement, replication semantics (e.g., strict vs. eventual consistency), and privacy requirements that should be met.

By leveraging slack and consistency requirements, *SlackStack* can decide which mailbox(es) to use while minimizing the cost. For example, it could decide to write to the mailbox in the cheapest DC and return to the application, if eventual consistency is required. Similarly, applications can specify privacy/security requirements, including permissions for in-network services to operate on encrypted ADUs [14, 15, 30].

While prior proposals (e.g., XIA [8, 20], RANS [24], etc) also use a DAG, the main challenge in *SlackStack* will be to support a simple, yet expressive interface, that can be used for specifying mailboxes and their semantics, along with slack and privacy/sharing requirements.

A DAG will also be used in the opposite direction, from the network to the application, to convey different transfer options to the application. One possible way to expose these options is using the concept of a *marketplace* [39], where slack options will be advertised as different services, and represented in the DAG with their appropriate cost and performance. The application may need to provide some hints with regards to acceptable slack range or budget, so the network can focus on providing only feasible options. After getting the options, the application can choose its desired option, and use the marketplace for payment and accountability [39].

3.3 Slack-Aware Transport Services

The *basic* service provided by *SlackStack* transport is reliable data delivery to a mailbox (push mode) and from a mailbox to the consumer (pull mode), using a reliable hop-by-hop transport protocol (e.g., [12, 13, 15, 18]). For the mailbox, we need to investigate how we can use different cloud storage options that have different cost vs. performance trade-offs. For example, Amazon's S3 can be half the price of EBS if the reads and writes are done on larger blocks, but has an order of magnitude higher latency, which means that data should be aggregated and data transfers should be carefully "staged" to meet the slack requirements.

Dynamic Insertion of Transport Services. The potentially large slack time for data transfer means that the requirements of the transfer may change, thus requiring *on-the-fly* transport services. To meet the requirements of such dynamic scenarios, we consider two complementary techniques: a query interface for mailboxes, and insertion of transport services (e.g., transcoding) while data is in-flight.

The query interface extends the `pull()` mechanism, enabling consumers to query the mailbox and retrieve a subset of ADUs, or even modified ADUs, based on the requirements. We can consider a simple SQL like interface, with initial support for `SELECT`, `FROM`, and `WHERE` clauses. Consumers will be able to search for specific ADUs that meet certain criterion (e.g., produced in the last 1 min), retrieve certain fields from structured ADUs (e.g., IoT data), and also search across shared mailboxes of other users (e.g., search for an ADU in a friend's shared mailbox).

To support insertion of services in a dynamic fashion, we can combine two well known concepts: stack of identifiers to effectively chain multiple services [35, 38] and *late binding* [18] to ensure that the service identifiers are resolved repeatedly to identify any updates. So the MID will be resolved to a stack of identifiers corresponding to the various in-network services that need to be used (at that time). We can potentially associate a lease (timeout value) with this mapping so the network makes sure to resolve the MID again after the lease expires. Further, this name resolution could also be *context* dependent – for example, return a dif-

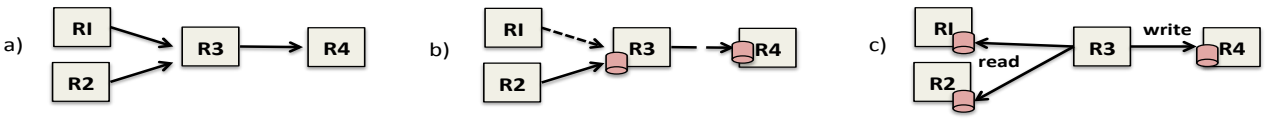

Figure 2: Energy Saving Example. All nodes need to synchronize in a), but use of storage in b) decouples them. Use of RDMA in c) allows R3 to do all the work via remote reads and write.

ferent stack based on who is querying (trusted vs. untrusted entities) or at what time the query is being made (depending on how much slack is left). The above proposal synthesizes known concepts, but moves the complexity to the resolution service.

3.4 Slack-Aware Forwarding

Slack-aware packet forwarding can leverage the well-understood benefits of batching and can use slack-awareness while scheduling packets. Here, we highlight the potential for energy savings because of slack. Most prior proposals on energy savings are limited by the constraint that both end-points of a link need to be active simultaneously in order to transmit a packet [31]. Fig. 2a shows a simple example involving four nodes: routers R1 and R1 want to send packets to R4 through R3. In this scenario, all four nodes need to be up *at the same time* and synchronize their communication (shown by having the same type of arrow), in order to potentially use any sleep modes for energy saving. As noted by Nedveshchi et al. [31], such synchronizing becomes complex as the size of the network grows, and as we consider realistic traffic patterns.

Using slack information, we can *decouple* the sender and receiver with the help of in-network storage: a sender can write to the memory of its neighbor who can process the data at a later time (when it wakes up). Fig. 2b shows how the transfers can happen at different times (indicated by different types of arrows). All nodes need not be up at the same time – each node can have its own schedule for sleeping/waking up as long as it obeys the slack requirement of the packets. The example assumes that it is possible to write directly to the memory of your neighbor. We can consider two options in this regard, both targeted towards edge clusters or small scale DCs, which use software middleboxes and routers. First, we can dedicate a core for packet processing which receives incoming packets and stores them in suitable memory regions; other processors wake up based on their schedule and process data from their memory. The second option is based on RDMA-based interconnects which allow direct read and write to remote memory without involving the remote processor [17, 26].

Fig. 2c shows how RDMA can be leveraged for additional energy savings. The router in the middle (R3) can allow everyone else to sleep: it can do a remote read from the memories of the senders (without involving their processors) and then subsequently do a remote write to the memory of the destination. So any router that wakes-up ensures that it does both a read and a write, allowing every two (out of three) routers to potentially sleep all the time.

3.5 Slack-Aware Routing

We consider the slack-aware routing problem in the context of a cloud provider transferring data from one DC to another. We can formulate this as a least-cost routing problem on a graph where a vertex represents some resource (e.g., storage, compute) and the edges carry the delay and cost associated with transferring data from one resource to the other.

A first challenge is capturing the *temporal* aspect of the problem. We need to capture the flexibility offered by slack: data could wait at a resource and only move when the cost is lower. Similar to NetStitcher [29], we can do a time expansion of the cost/delay graph, although they only consider a limited number of time periods (peak vs. offpeak) which allows them to compute the optimal solution even though this is an NP-hard problem. In our case, we will need to use heuristics such as: pruning infeasible slack options or removing those nodes with a small change in price, to reduce the size of the graph and possibly solve the problem in a multistage fashion.

A second issue is putting suitable cost/delay weights for future times because this information may not be known, especially if a spot pricing model is being used for different cloud resources. To address this challenge, we can use *forecasts* based on applying standard statistical techniques on historical data, and report confidence intervals, in order to capture the uncertainty in our forecasts.

4. OUR WORK IN CONTEXT

Table 1 shows how the building blocks used in *SlackStack* are inspired by prior proposals, including DTNs [18], content and data-centric architectures (e.g., CCN [25, 28], DOT [36]), pub-sub systems (e.g., PURSUIT [19]), future Internet architectures (e.g., XIA [20]), indirection-based proposals [35, 38] (e.g., i3 [35]), slack-based scheduling (e.g., NetStitcher [27], Calendaring [27]) and segment-based transports (e.g., Tapa [15]). *SlackStack* synthesizes these concepts and uses them in a novel way for slack-aware networking. For example, *SlackStack* uses both push and pull modes while existing systems mostly use either a push mode (e.g., DTN [18]) or a pull mode (e.g., CCN [25]), but not both. Similarly, while DTN and CCN make use of in-network storage, *SlackStack* uses it in a novel way: in-network storage can be addressed (using MIDs) and services can be added on-the-fly while data is in transit. In addition, the two new concepts in *SlackStack*, which most prior proposals lack, are: i) use of slack as an *opportunity* rather than a constraint (as in DTNs), so if the network is connected, DTNs will *immediately* deliver the data, anticipating disconnections in the future, while *SlackStack* may potentially wait to optimize for slack, and ii) consideration of cost vs. performance trade-offs present in cloud settings, which open unique slack opportunities in both time and space. These differences lead to opportunities and challenges that form the basis of the rich research program that we presented in this paper.

	DTN	i3	CCN	XIA	Pub-Sub (PURSUIT)	NetSticher	Calendaring	DOT	Tapa	*SlackStack*
In-network Storage	X		X	X	Limited	X		X	Limited	X
Push Mode	X	X		X	X	X	X	X	X	X
Pull Mode			X					X	X	X
Indirection		X		X	X	X			X	X
Slack	X					X	Limited			X
Cloud Tradeoffs										X

Table 1: Comparison of *SlackStack* with other relevant proposals.

5. CONCLUSION

We argue that slack-based optimizations will become increasingly more important in future. Our proposal, *SlackStack*, should be viewed as a first step towards understanding how the network stack can be revamped to support slack-aware optimizations. We believe that *SlackStack* can significantly benefit and evolve from a broad range of community feedback and involvement as it extensively uses well-known building blocks, both from seminal proposals as well as recent efforts on future Internet architectures, and raises questions that span the *entire* network stack.

Acknowledgement

For their useful input on *SlackStack*, the author would like to thank John Byers, Ihsan A. Qazi, Peter Steenkiste, HotNets 2017 reviewers, members of the NaT Lab at Tufts, and the CCR reviewers.

6. REFERENCES

[1] Amazon AWS. http://aws.amazon.com.
[2] Amazon Fire. http://www.amazon.com/fire.
[3] Amazon Spot Instances. https://aws.amazon.com/ec2/spot/.
[4] Demand for Netflix, Amazon and other streaming video continues to grow. http://www.usatoday.com/.
[5] Google Preemptable VMs. https://cloud.google.com/preemptible-vms/.
[6] Nest. https://nest.com/.
[7] PlayLater. https://www.playon.tv/playlater/.
[8] Anand, A., Dogar, F., Han, D., Li, B., Lim, H., Machado, M., Wu, W., Akella, A., Andersen, D. G., Byers, J. W., et al. Xia: An architecture for an evolvable and trustworthy internet. In *Proc. ACM HotNets* (2011).
[9] Andersen, D. G., Balakrishnan, H., Feamster, N., Koponen, T., Moon, D., and Shenker, S. Accountable Internet Protocol (AIP). In *SIGCOMM* (2008).
[10] Clark, D. D., and Tennenhouse, D. L. Architectural considerations for a new generation of protocols. In *Proc. ACM SIGOCMM* (1990).
[11] Clark, D. D., Wroclawski, J., Sollins, K. R., and Braden, R. Tussle in cyberspace: defining tomorrow's internet. In *ACM SIGCOMM CCR* (2002), pp. 347–356.
[12] Dogar, F. R., Phanishayee, A., Pucha, H., Ruwase, O., and Andersen, D. G. Ditto: a system for opportunistic caching in multi-hop wireless networks. In *ACM MobiCom* (2008).
[13] Dogar, F. R., and Steenkiste, P. M2: Using Visible Middleboxes to Serve Pro-active Mobile-Hosts. In *ACM SIGCOMM MobiArch '08* (New York, NY, USA, 2008), ACM, pp. 85–90.
[14] Dogar, F. R., and Steenkiste, P. Segment based internetworking to accommodate diversity at the edge. *Technical Report - CMU-CS-10-104* (2010).
[15] Dogar, F. R., and Steenkiste, P. Architecting for Edge Diversity: Supporting Rich Services Over an Unbundled Transport. In *Proc. ACM CoNext* (2012).
[16] Dogar, F. R., Steenkiste, P., and Papagiannaki, K. Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *Proc. ACM MobiSys* (2010).
[17] Dragojević, A., Narayanan, D., Castro, M., and Hodson, O. Farm: fast remote memory. In *Proc. Usenix NSDI* (2014).
[18] Fall, K. A delay-tolerant network architecture for challenged internets. In *Proc. ACM SIGCOMM* (2003), ACM.
[19] Fotiou, N., Trossen, D., and Polyzos, G. C. Illustrating a publish-subscribe internet architecture. *Telecommunication Systems 51*, 4 (2012), 233–245.
[20] Han, D., Anand, A., Dogar, F. R., Li, B., Lim, H., Machado, M., Mukundan, A., Wu, W., Akella, A., Andersen, D. G., et al. XIA: Efficient Support for Evolvable Internetworking. In *Proc. Usenix NSDI* (2012).
[21] Haq, O., and Dogar, F. R. Leveraging the Power of the Cloud for Reliable Wide Area Communication. In *Proc. ACM Hotnets* (2015).
[22] Haq, O., Raja, M., and Dogar, F. R. Measuring and improving the reliability of wide-area cloud paths. In *Proc. WWW* (2017).
[23] Hong, C.-Y., Kandula, S., Mahajan, R., Zhang, M., Gill, V., Nanduri, M., and Wattenhofer, R. Achieving high utilization with software-driven WAN. In *Proc. SIGCOMM* (2013).
[24] Iftikhar, A. M., Dogar, F., and Qazi, I. A. Towards a redundancy-aware network stack for data centers. In *Proc. ACM HotNets* (2016).
[25] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L. Networking named content. In *CoNEXT '09* (New York, NY, USA, 2009), ACM, pp. 1–12.
[26] Kalia, A., Kaminsky, M., and Andersen, D. G. Using rdma efficiently for key-value services. In *Proc. ACM SIGCOMM* (2014).
[27] Kandula, S., Menache, I., Schwartz, R., and Babbula, S. R. Calendaring for wide area networks. *ACM SIGCOMM Computer Communication Review 44*, 4 (2015), 515–526.
[28] Khan, A. Z., Baqai, S., and Dogar, F. R. QoS aware Path Selection in Content Centric Networks. In *Proc. IEEE ICC* (2012).
[29] Laoutaris, N., Sirivianos, M., Yang, X., and Rodriguez, P. Inter-datacenter bulk transfers with netstitcher. In *Proc. ACM SIGCOMM* (2011).
[30] Naylor, D., Schomp, K., Varvello, M., Leontiadis, I., Blackburn, J., López, D. R., Papagiannaki, K., Rodriguez Rodriguez, P., and Steenkiste, P. Multi-context tls (mctls): Enabling secure in-network functionality in tls. In *Proc. ACM SIGCOMM* (2015).
[31] Nedevschi, S., Popa, L., Iannaccone, G., Ratnasamy, S., and Wetherall, D. Reducing network energy consumption via sleeping and rate-adaptation. In *Proc. Usenix NSDI* (2008).
[32] Satyanarayanan, M., Bahl, P., Caceres, R., and Davies, N. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE 8*, 4 (2009), 14–23.
[33] Sharma, A., Tie, X., Uppal, H., Venkataramani, A., Westbrook, D., and Yadav, A. A global name service for a highly mobile internetwork. In *Proc. ACM SIGCOMM* (2014).
[34] Sovran, Y., Power, R., Aguilera, M. K., and Li, J. Transactional storage for geo-replicated systems. In *Proc. ACM SOSP* (2011).
[35] Stoica, I., Adkins, D., Zhuang, S., Shenker, S., and Surana, S. Internet indirection infrastructure. In *Proc. SIGCOMM* (2002).
[36] Tolia, N., Kaminsky, M., Andersen, D. G., and Patil, S. An architecture for internet data transfer. In *NSDI '06*.
[37] Venkataramani, A., Kokku, R., and Dahlin, M. Tcp nice: A mechanism for background transfers. *ACM SIGOPS Operating Systems Review 36*, SI (2002), 329–343.
[38] Walfish, M., Stribling, J., Krohn, M., Balakrishnan, H., Morris, R., and Shenker, S. Middleboxes no longer considered harmful. *OSDI* (2004), 215–230.
[39] Wolf, T., Griffioen, J., Calvert, K. L., Dutta, R., Rouskas, G. N., Baldin, I., and Nagurney, A. Choicenet: toward an economy plane for the internet. *ACM SIGCOMM Computer Communication Review 44*, 3 (2014), 58–65.