Architecting for Diversity at the Edge: Supporting Rich Network Services Over an Unbundled Transport

Submitted in partial fulfillment of the requirements for
the degree of
Doctor of Philisophy
in
Department of Electrical and Computer Engineering

Fahad R. Dogar

BSc (Hons), Computer Science, LUMS, Pakistan
MS., Electrical and Computer Engineering, Carnegie Mellon University

Carnegie Mellon University

May, 2012

Keywords: Architecture, Transport, Services

Abstract

The end-to-end nature of today's application and transport protocols is increasingly being questioned by the growing heterogeneity of networks and devices, and the emergence of rich in-network services. As a result, performance of end-to-end protocols is often poor and many in-network optimizations are hard to deploy because they do not fit into today's Internet architecture. This thesis takes a clean slate approach towards better accommodating diversity in the Internet architecture. We propose two architectural concepts: i) today's end-to-end transport is unbundled such that network specific functions, like congestion control, are implemented on a *per-segment* basis, where a segment spans a part of the end-to-end path that can be considered homogeneous (e.g., wired Internet or an access network) and ii) we expose applications' data units and their naming to certain elements within the network, thereby facilitating the deployment of various data oriented and higher level services inside the network. We have designed a network architecture, Tapa, that systematically combines these concepts into a coherent architecture. Tapa uses end-to-end protocols that transfer application data units (ADU) over segments; these segments appear as traditional Internet-like "links" to the end-to-end protocols and can use Internet-style protocols (e.g., TCP/IP-like protocols in the backbone) or custom solutions at the edges. We demonstrate the effectiveness of Tapa by showing how it can support: i) various wireless and mobility optimizations, ii) an in-network energy saving service that can provide up to 2-5x improvement in battery life of mobile devices, iii) improved content distribution in online social networks, and iv) higher level services with new application semantics.

Acknowledgments

This is the moment I have been dreaming of since the start of my PhD. Writing acknowledgements is like taking a walk down the memory lane knowing that whatever happened at that time happened for the good. I cannot thank God enough for enabling me to reach this point. I am also grateful to HEC and NSF for funding my graduate studies and research. It is often quoted that PhD is like a roller-coaster ride, with its own highs and lows. In this note, I would like to acknowledge those special people in my life who ensured that I was always at a high by shielding me from the lows.

I was fortunate to have Peter Steenkiste as my advisor. Peter is professional, wise and understanding – qualities that make him a pleasure to work with. He was an ideal advisor for me as he gave me complete freedom to pursue my interests and yet was always available to give his invaluable advice and help. Peter took me under his wings when I was beginning to question CMU's decision of giving me admission and was instrumental in keeping me positive throughout my stay.He also made sure that I focused on the important things during my PhD, something that enabled me to graduate on time. Thank you, Peter!

My sincere thanks also to Srini Seshan, Ratul Mahajan and Dina Papagiannaki for agreeing to sit on my thesis committee and for providing timely feedback even though I always bugged them at the very last moment. Srini's insights regarding several aspects of my thesis, Ratul's critical feedback and pertinent questions, and Dina's focus on deployment aspects of my work as well as her collaboration on the Catnap project, helped my thesis in various ways.

I also had the privilege to work and interact with some great people at CMU. Satya always inspired me with his work-ethics, focus and clarity in thinking and presentation. Dave Andersen's passion for research and approach towards problem solving was equally inspiring. He also took time to give useful feedback on practice talks and several drafts of my papers. Amar Phanishayee and Olatunji Ruwase were a pleasure to interact with, both as collaborators and as friends. Amar was always around to talk about research and life in general. I also thoroughly enjoyed working with the XIA team during my final year.

Several folks at CMU, including my office-mates and Tuesday seminar colleagues, provided great company as well as technical feedback. This included Vijay Vasudevan, Vyas Sekar, Watanee Viriyasitavat, Iulian Moraru, Jiri Simsa, Dongsu Han, Niraj Tolia, Jeff Pang, Kaushik Lakshminarayanan, Dan Wendlandt, George Nychis, Xiaohui Wang, Amy Lu, Xi Liu, Wolfgang Richter, Athula Balachandranan, Umar Javed, Niraj Tolia, Himabindu Pucha, Bin Fan, and Hyeontaek Lim to name a few. Last but not the least, CMU has a great administrative support system; Angela Miller epitomizes this – she was amazingly kind and helpful.

I was also extremely fortunate to find great friends outside school. Ihsan Qazi, Asim Jamshed, Syed Ghaus Ali, Ali Arshad, Sara Tahir, Mir Hamza Mahmood, Ammar Baray, Usman Khan, Rutu Bole, Baber Farooq, Madiha Farooq, Ahsan Latif, Agha Ali and Rameez Mustafa provided me a life outside school, which was critical in keeping me sane. I must say – you guys rock!

My friendship with Ihsan dates back to our undergraduate days, but the time we spent together in Pittsburgh further strengthened our bond. We spent four memorable years together in Pittsburgh, three of which were as apartment mates. I will always cherish the fond memories of our long discussions on religion, sports, relationships, research – in short, everything and anything going on in our lives. He was with me through thick and thin, through paper rejections and swine flu, and through birthdays and paper acceptance celebrations. And how can I forget our joint cooking and movie sessions, frantic cleaning of apartment before inspections, playing cricket inside as well as outside our apartment, desperate sprints to catch a missing bus, and so on. Ihsan, thank you for all this and more importantly for being a great friend.

Special thanks also to Ali Bhai for his quality time, Sara for always cheering us up with her lively plans, and Hamza, Asim and Ali Arshad for being always ready to have fun.

I must also give credit to those people without whom I could not enter a university like CMU. My friends and teachers at LUMS always encouraged me to pursue graduate studies. They showed confidence in me when I had little self-belief. My friends from LUMS days – Nazim Ashraf, Sana Khawaja, Qurrat-ul-Ain Saeed, and Tariq Mahmood – provided crucial support at different stages of my life. My undergraduate teachers provided me a good foundation that helped me during my PhD. Dr. Tariq Jadoon invigorated my interest in computer science research while Dr. Zartash Uzmi provided me with the opportunities and guidance to undertake research while I was an undergraduate student. I have no doubt, that without Dr Zartash's mentoring I would have never made it to CMU. These professors have been role models for me, both as mentors and human beings. I will always remain indebted to them and to all the other teachers who have taught me throughout my academic

life.

This thesis would not have been possible without the unconditional love and support of my family. My sister, Saima, was the main reason I didn't get as homesick as I initially expected to be. She was always there to talk to me, to boost my confidence and to offer words of encouragement. Rain or shine, she would always talk to me, constantly reminding me that even though I was physically far away from my family, yet, in spirit, I was very close to them. I am also grateful to my brothers and bhabis for taking good care of my parents while I was away and for always encouraging me and taking pride in my accomplishments. My sincere thanks also to my wife Isha, her family, and all my relatives who prayed for me and encouraged me. Isha was instrumental in motivating me to defend my thesis early and then to submit my final draft. Sometimes I wish she was with me during my graduate studies in Pittsburgh but then I wonder whether I would have been able to do justice to both research and her. My heart also goes out for my late grand-father, Haji Noor Karim, whose words of wisdom and yearning for education were always a source of inspiration for me. He would've surely been a proud man today!

Finally, I will make an attempt to thank two people without whom I would have been nowhere. Words cannot do justice to my feelings of gratitude towards my parents. Tears fill my eyes as I remember all the sacrifices my parents made for my upbringing. Education had always been a top priority for them and they did everything that was possible to provide good, quality education to me. I still remember how my mother would patiently convince me every morning to go to school during my primary school days. I still remember the night when my father took me to the market and got me a new pair of school shoes even though he had come back late from work. Their sacrifices continued during my PhD too. Staying far away from her children is always tough for a mother and it was no different for my mom. She put up a brave show just so that I could get the best education without worrying about her. Her constant prayers for my success and her unfaltering confidence in me was a great source of comfort for me. Likewise, my dad's desire to see me do well in life and his brave fight against Parkinson's, were a great source of inspiration for me. As a small token of appreciation, I dedicate this thesis to my dear parents.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Today's Internet is based on the "smart hosts, dumb network" principle: the network provides a simple best effort data delivery service while all the intelligence is placed at end-points, as part of the transport and application layers [44]. Two recent trends, however, increasingly challenge this principle:

1. Heterogeneous Networks and Devices: Unlike the original Internet, the networks that make up the Internet now are very diverse, ranging from very high speed optical backbones, to low speed, unreliable wireless networks that have very different properties from wired links (e.g., higher error rates, variable bandwidth and delay), causing problems for end-to-end protocols such as TCP [29, 56]. Similarly, unlike traditional end-points, many modern devices, such as sensors and mobile nodes, are highly resource constrained and often require customized protocols (or even new protocol stacks) [80]. Dealing with heterogeneity of devices and networks affects how we distribute functions across devices. For example, when applying the end-to-end argument [102], if the end-to-end path comprises of networks with very different properties then we can no longer view the communication subsystem as a *single homogeneous* module.

2. Rich Network Services Today's network needs to provide a wide variety of services, including *data oriented services* [52, 72], such as application independent caching, content retrieval from multiple sources, etc, and *higher level services*, such as transcoding and virus scanning proxies [15, 101]. Unfortunately, inserting such services in an end-to-end path is hard in today's Internet, primarily because of two reasons: i) TCP's end-to-end semantics

do not accommodate the role of an intermediary, and ii) applications' data units and their naming is not exposed to the network. As a consequence, services tend to be implemented in an application specific manner, creating an entry barrier for new applications as they have to re-implement support for common network services (e.g., caching, support for mobility, etc). For example, consider a new application protocol that allows P2P-style file sharing between hosts. If this protocol wants to benefit from locality of requests and use in-network caching, it will need to provide this support even if the network already supports HTTP caching. This is because these two application protocols will have their own naming and customized way of requesting data, thereby requiring an application specific network service.

1.2 Limitations of Existing Solutions

There is a growing realization that addressing the above challenges require more support from the network and purely end-to-end solutions are insufficient. However, so far, our response to the above trends is to implement ad-hoc solutions, such as splitting transport connections or using transparent proxies and various other application middleboxes [15, 29, 39, 73]. Such temporary fixes have several limitations. The main problem is that these points solutions hinder long term innovation as they make the Internet *brittle and complex* [46, 60]. The added complexity is visible on multiple fronts. Higher level services often need to re-implement a significant amount of underlying functionality due to lack of architectural support. For example, a virus scanning service will need to re-implement transfer optimizations, such as using multiple interfaces or retrieving data from multiple sources in parallel. Similarly, as mentioned earlier, new applications also need to provide support for common in-network services, such as mobility and caching, which creates an entry barrier. Another dimension is that these solutions often introduce new failure modes as they are forced to work transparently with existing applications [29]. As a result, it is difficult for developers to write robust applications that can work in presence of unexpected failures caused by these ad-hoc solutions.

Apart from the additional complexity, another problem with these application specific approaches is that they often fail to match the performance of a system-wide architectural solution. For example, application independent caching can provide better results compared to application specific approaches [94, 113]. Similarly, an integrated approach towards congestion control can perform better compared to implementing congestion control on a per application/flow basis [30]. Energy efficiency is another area where a system-wide approach, instead of application specific approaches, is more beneficial because, for example, a device can only be put into deep sleep mode if no application is using it at that time [20, 55, 109].

1.3 Key Architectural Concepts

As the diversity in today's Internet is only going to increase in future, there is a need to address these problems in a systematic way, instead of using ad-hoc solutions. So in this thesis, we ask the following question: how can we provide *architectural* support to accommodate heterogeneous networks and rich network services? To this end, we take a clean-slate approach towards designing a new architecture that addresses these problems from the ground up. As a first step, we revisit the question of how we should distribute functions between end-points and the network. This exercise leads us to two key concepts that form the basis of this thesis.

1. Unbundling Transport The growing diversity of networks and devices means that it is difficult to have an end-to-end solution that works well under *all* scenarios. So instead of having a "one size fits all" solution, as is the case with TCP in today's Internet, we argue for customized solutions that are tailored for the underlying network. This requires unbundling today's transport layer in both the vertical (across layers) and horizontal (across the network topology) dimensions of the system in a way that network specific functions, like congestion control, can possibly be implemented with the help of the network while functions that ensure correct application semantics can still be implemented on an end-to-end basis.

2. Exposing Applications' Data Units to the Network The need to have a wide variety of in-network services, most of which operate on data, means that it can be beneficial to expose applications' data units and their naming to certain elements within the network. This implies that applications follow certain conventions while naming their data (e.g., self-certifying names [113]) and the network understands these conventions and can undertake different operations (e.g., lookup/routing) without requiring application specific help. This enables the deployment of various data oriented and higher level *services* without requiring application specific help (e.g., application independent caching). Moreover, it can also potentially improve the network's ability to manage its resources (e.g., congestion control, admission control and related tasks).

1.4 Tapa

Tapa synthesizes the above concepts into a coherent architecture. It turns *segments*, which correspond to a portion of an end-to-end path that is homogeneous (e.g., "wired Internet", a private network owned by an enterprise, a wireless mesh network, etc) into Internet-

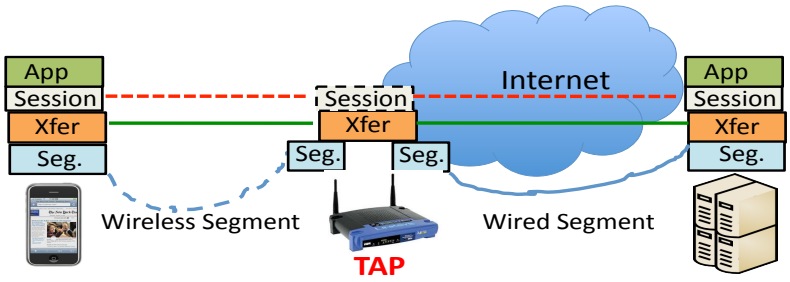


Figure 1.1: Tapa Overview.

like "links"; these segments can use Internet-style protocols (e.g., TCP/IP-like protocols in the backbone) or custom solutions at the edges. Each segment provides best effort data delivery service to the upper layer – functions that may be required to provide this service (e.g., routing, error control, congestion control, etc) are internal to the segment and hidden from higher layers. Segments correspond to our first concept as they are an outcome of unbundling today's transport in both the *vertical* (across layers) and *horizontal* (across the network topology) dimensions of the system.

Figure 1.1 shows a typical end-to-end data transfer in Tapa consisting of two segments (wired and wireless). On top of segments are end-to-end layers (transfer and session), which are based on application data units (ADUs) [45]; these protocols focus on higher level challenges related to applications and in-network services.

The *transfer* layer supports end-to-end data transfers overs multiple segments, similar to how IP supports connectivity in today's Internet. The transfer service runs on both the end-points and network elements, called Transfer Access Points (TAPs), that interconnect segments. TAPs provide the glue required to combine multiple segments (e.g., buffer space) and also support various types of services. For example, at the transfer layer, TAPs can support data oriented services, like application independent caching and opportunistic content retrieval, similar to how DOT [113] supports such functionality at end-points.

The *session* layer implements specific application semantics over the transfer layer. The presence of the segment and transfer layers makes the session layer lightweight, isolating it from the details of the underlying network technologies as well as specific data retrieval mechanisms. As a result, it is easier to implement session protocols with diverse semantics

required by different types of applications (e.g., fully reliable, streaming applications, etc.) It also becomes easier to insert services into the end-to-end path, while maintaining specific semantics between the end-points and (possibly third party) network services.

We have designed a session layer that supports new semantics involving in-network services for four functions that traditionally involve the end-points only: confidentiality, reliability, ordering, and data integrity. We show how real applications can use various types of semantics for these functions based on their requirements. For example, a social network application can explicitly allow an intermediary service to decrypt and cache a photo, but may want end-to-end encryption for sending its username/password information. This is an example of *controlled transparency* [47], as applications can explicitly allow an intermediary to read certain encrypted data while choosing end-to-end encryption for data that requires complete confidentiality. Another example of an in-network service that changes the traditional end-to-end semantics of reliability is *delegation*. Using this service, applications can off-load reliable data transfer to the TAP while still ensuring correct application behavior.

Finally, we design an *API* that allows applications to make full use of Tapa, a key aspect of which is choosing appropriate communication semantics between the end-points and in-network services. The API provides a high level abstraction for *data oriented* applications to retrieve and publish content using a simple `put/get` interface. The API operates at the granularity of ADUs, which is a natural unit of decision making for applications. So applications can specify their high level requirements for retrieving and publishing ADUs rather than using a low level API based on byte streams or datagrams (e.g., socket API). This enables applications to undertake common tasks, such as data retrieval, without worrying about the underlying protocols or mechanisms that Tapa may use. At the same time, the API also provides applications enough *control* to change the *semantics* of the ADU communication, if they desire so. For example, applications can opt for different communication semantics, which may involve the role of an in-network service or use of a specific transfer mechanism/protocol.

1.5 Benefits under Diverse Scenarios

In this thesis we consider a broad range of scenarios to demonstrate the effectiveness of Tapa. We present detailed case studies in later chapters, which show that Tapa can easily support mobile and wireless optimizations (Swift, Chapter 3), an in-network energy saving service (Catnap, Chapter 4), and personalized content distribution in social networks (Vigilante, Chapter 5). Here we give a brief overview of the problem that each of these case studies addresses and the key benefits that we attain through the use of various Tapa features.

1.5.1 Supporting Wireless and Mobile Users

The original Internet was designed with static users and wired networks in mind, so there is no architectural support for wireless and mobile users in today's Internet [90]. As a result, performance is often poor and many wireless and mobile optimizations are hard to deploy because they do not cleanly fit into the design of today's Internet architecture [29, 31].

Tapa can help mobile and wireless users in three broad ways. First, it can improve data transfer performance for mobile clients through caching and intelligent use of segments. As an example, a mobile user can easily switch to a different TAP, which may use a different wireless technology, and still continue to download the ADUs from the old TAP. Second, Tapa's use of ADUs and segments simplify the use of wireless optimizations that allow multiplexing of multiple interfaces or access points (TAPs in this case). Such optimizations are not only difficult to implement in today's Internet, but are also tied to a specific transport protocol (e.g., TCP) or specific underlying technology (e.g., WiFi) [68]. Third, a wireless device can use a variety of protocols at the segment layer without worrying about the issues of the wired segment. This enables the use of protocols that are best suited for the particular technology/environment. For example, if we have a wireless mesh network, we can use a segment protocol like HOP [81], which relies on per-hop mechanisms for reliability and rate control while still using TCP like protocols on the wired segment.

1.5.2 Energy Efficient Data Transfers

Energy efficiency has always been critical for mobile hosts, but has recently received attention in the context of wired hosts as well [20, 22]. Today's transport (i.e., TCP) requires strict synchronization between end-points leaving little room to put end-devices into sleep mode. As a result, end devices have to remain up for the whole duration of transfers and most sleep modes (e.g., Suspend-to-RAM) are only used when no application is conducting any data transfer [55].

Tapa has several ingredients that allow more energy efficient data transfers compared to today's Internet. Specifically, we have designed Catnap, an in-network traffic shaping service that leverages key features of Tapa to provide significant energy savings to end devices. Catnaps allows mobile devices to go into sleep mode *during* data transfers by intelligently shaping when data is sent on different segments. It targets settings where the wireless segment is much faster compared to the wired segment; in such scenarios, Catnap allows the wireless segment to remain inactive for most of the time while still ensuring that the transfer finishes on time. During the time the wireless segment is inactive, the mobile device can enter into various sleep modes (e.g., 802.11 Power Save Mode, Suspend-to-RAM mode,

etc), thereby providing significant energy savings to the mobile client. Catnap can provide up to 2-5x battery life improvement for real mobile devices under certain conditions [55].

1.5.3 Improved Content Distribution in Online Social Networks

Online Social Networks (OSN) have gained unprecedented popularity in recent times and involve billions of users who generate and consume a huge amount of content. Supporting content distribution in such environments is challenging: traditional centralized solutions do not scale well, resulting in a high cost solution that still gives poor performance.

In order to address this problem, we use TAPs located at users' homes to store and distribute social networking content within a social community. This is feasible because OSN users have limited social circles and predictable content access patterns, making it possible to use techniques such as on-demand content retrieval or pre-fetching content based on OSN information. We have designed a system, Vigilante, that leverages Tapa to enable the above techniques, thereby improving the performance and scalability of content distribution in OSNs. Our evaluation on PlanetLab shows that Vigilante can outperform even the best case performance achieved using traditional solutions like CDNs.

1.6 Scope

This thesis focuses only on *edge* diversity, which is typically present in most Internet access scenarios today due to the use of various mobile and wireless devices at the edges. In such scenarios, there could be a wide range of devices (e.g., laptops, smart-phones, sensors that may use various access technologies (e.g., WiFi, bluetooth, cellular) under diverse access scenarios (e.g., community mesh network, vehicular Internet access, etc). As shown in this thesis, effectively dealing with this diversity requires efficient solutions to deal with disruptions, which are caused by mobility or poor channel conditions, as well as exploiting the inherent opportunities present in these settings, such as opportunistically using multiple interfaces to aggregate bandwidth or to leverage the broadcast nature of the wireless medium [52].

In terms of services, this thesis focuses on in-network services that are loosely supported at the transport or higher layers of today's protocol stack. Example of such services in today's Internet include transport connection splitting, application layer proxying, caching, transcoding services, virus and worm scanners, etc. The case studies presented in this thesis focus on a subset of these services. This subset is referred to as *data oriented services*, which are services that typically operate on chunks of data (ADUs) and do not require application

specific help i.e., they are fairly broad in terms of their applicability and can operate using some generic *hints* provided by the application. Examples of such data oriented service include caching and traffic shaping. Finally, services that are typically supported below the transport layer (e.g., NAT) do not explicitly benefit from the solutions presented in this thesis and are out-of-scope.

1.7 Thesis Statement

This thesis states that:

We can accommodate the growing diversity at the edge by systematically combining two concepts: i) unbundling today's transport in both the vertical and horizontal dimensions of the system and as a result implementing most of the traditional transport functions on a per-segment basis rather than on an end-to-end basis, and ii) exposing applications's data units and their naming to certain elements within the network.

In order to demonstrate the feasibility of the above statement, we show how a systematic architecture based on the above concepts simplifies the use of various mobile and wireless optimizations, results in significant energy savings for end-devices, improves the scalability of online social networks, and provides explicit support for new in-network services that change the end-to-end communication semantics.

1.8 Contributions

This thesis makes contributions in three broad categories:

Concepts/Ideas

- We introduce two concepts: unbundling of transport in both the horizontal and vertical dimensions of the system and exposing ADUs and their naming to certain elements within the network. Our main contribution is synthesizing these concepts into a coherent architecture (i.e., Tapa). In addition, we also propose a new API for API, which is simpler and allows greater control over semantics compared to the socket API or other APIs proposed for data oriented architectures [113].

- We propose the idea of exploiting bandwidth discrepancy between wired and wireless links to allow energy savings to mobile devices. We also propose and synthesize specific concepts that help in realizing this idea in a practical, real-world system.

- We propose the idea of forming a personal CDN amongst TAPs of users to store and distribute online social networking content. We also enhance the Tapa design to show how this idea can be realized in a practical system.

- We introduce session services, like controlled transparency and delegation, that are useful for performance and policy reasons. We show how these services can be easily supported in Tapa without requiring any application specific help at the TAPs and without changing the API exposed to applications for controlling the semantics of these services.

Software Artifacts

- We have built software prototypes of all the systems proposed in this thesis. These prototypes are open source and run on Unix (and its variants). They have been tested and evaluated on multiple hardware types and experimental environments.

Insights based on Prototype Evaluation and Real World Measurements

- We evaluate the Swift prototype on real world and emulator based testbeds. Our evaluation attests to the benefits of Swift for mobile and wireless users under a wide variety of real world scenarios.

- We evaluate the Catnap prototype with multiple mobile devices, under a wide range of traffic scenarios. Our evaluation shows that we can get significant battery life improvements for real mobile devices. Our measurements also shed insights into the problems with existing available sleep modes in typical home access scenarios; these measurement results constitute a useful contribution independent of Catnap.

- We conduct a comprehensive measurements study of Facebook, which sheds unique insights into the content distribution strategies used by Facebook, as well as highlight the limitations of using these strategies for large scale content distribution in OSNs.

- We conduct an evaluation of Vigilante on the PlanetLab testbed using hundreds of nodes. We show that Vigilante can outperform even the best case performance achieved using Facebook's content distribution mechanisms.

1.9 Road Map

The road map for the rest of the thesis is as follows:

- In Chapter 2, we describe Tapa in detail; this includes the intuition behind the key concepts that form the basis of Tapa, and how Tapa combines these concepts into a coherent architecture.
- In Chapter 3, we present the design, implementation, and evaluation of Swift, a Tapa prototype that is specifically designed for wireless and mobile users.
- In Chapter 4, we present the design, implementation, and evaluation of Catnap, highlighting how key concepts of Tapa work together to provide energy savings to mobile clients.
- In Chapter 5, we present how Vigilante helps in scaling content distribution for online social networks, using key concepts of Tapa. This chapter also presents a detailed measurement study on content distribution in Facebook.
- In Chapter 6, we describe how Tapa can support diverse communication semantics through a suitable session layer and API. We also present a case study that illustrates how a social networking application can benefit from these rich semantics.
- Chapter 7 concludes this thesis by summarizing the key contributions and pointing out directions for future work.

Chapter 2

Tapa

Tapa is a clean slate network architecture that addresses the problem of accommodating the increasing diversity in today's Internet: diversity of heterogeneous networks and devices, and rich in-network services that are needed to support the trend towards content and service centric networking. In this chapter, we describe Tapa in detail, including the key concepts that form the basis of this architecture as well as how these concepts are combined to form a complete architecture. We also provide an overview of Tapa's API and discuss how Tapa supports services at different levels of the system, giving a precursor to the three case studies that follow this chapter. We present the detailed design of Tapa's API and a session layer that supports rich semantics between end-points and services after the case studies (Chapter 6). Finally, in this chapter, we also discuss a broad range of research work that is most relevant to Tapa.

2.1 Architectural Concepts

In the original Internet, most of the intelligence was built into the end points, as part of application and transport protocols. We revisit this question of how functions should be distributed between end-points and the network and propose two key changes, both of which involve a more explicit role for the network.

2.1.1 Unbundling Today's Transport

Today's transport protocol (i.e., TCP) is responsible for many functions, which are implemented at end-points. This design decision was rational given the *homogeneous* nature of the links in the original Internet. As shown in Figure 2.1, those links provided fairly reasonable reliability and a best effort service to the end-to-end transport. As a result, the

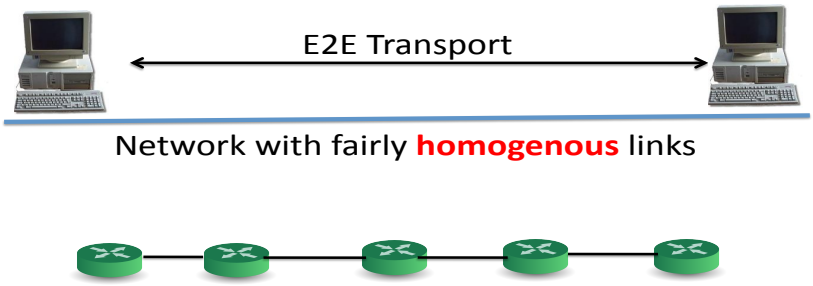


Figure 2.1: Original Internet. End-to-End Transport over Homogeneous Links

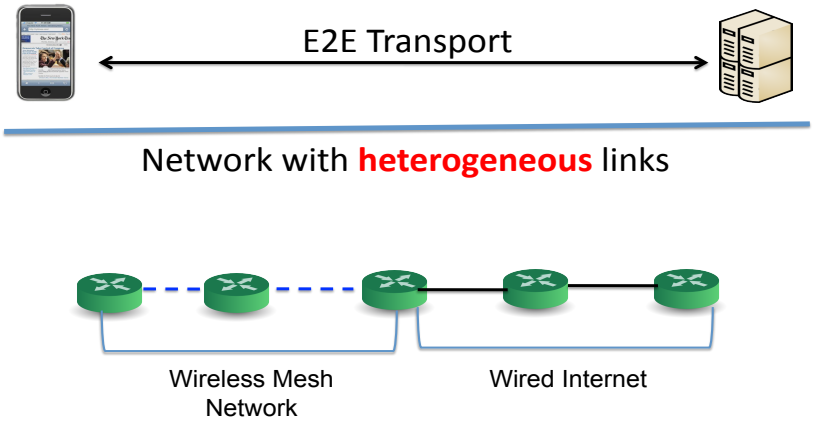


Figure 2.2: Today's Internet - End-to-End Transport over Heterogeneous Links

end-to-end transport could focus on implementing specific semantics required by the applications (e.g., full reliability, ordering, etc). Subsequently, congestion control was made part of TCP, rather than as a separate function supported at the network level. This decision was made for convenience reasons rather than based on any sound reasoning [93]. So TCP, today, implements congestion control and various other functions on an end-to-end basis.

Compared to the original Internet, typical end-to-end paths in today's Internet are very heterogeneous. As shown in Figure 2.2, the path could consist of some links that are part of a wireless mesh network while other links could be more traditional wired links. Today's end-to-end transport protocols have to operate over such heterogeneous links. This has three broad implications.

First, designing a "one-size fits all" solution is very difficult given the increasing heterogeneity of networks. As a result, we are forced to use solutions like TCP, which works reasonably well for the wired Internet but performs poorly in other network (e.g., wireless networks [29, 31], data center networks [21], etc) Second, there is no role for in-network services in today's transport protocols as all transport functions, ranging from connection establishment to connection termination, involve the end-points only. So tasks such as interception, redirection, etc, which are important for in-network services like caching, turn out to be difficult to support in today's Internet because TCP has no support for intermediaries. Third, application semantics, which include tasks that must be implemented with the help of end-points [102] (e.g., error control) are conflated with other functions like congestion control. For example, TCP uses acknowledgements for both congestion control and reliability, thereby making it difficult to implement other semantics (e.g., partial reliability) that may be required by applications.

We propose unbundling today's transport so that we can better accommodate the needs of heterogeneous networks and in-network services. The unbundling process has both vertical (across layers) and horizontal (across network topology) dimensions. We propose to *raise the level of abstraction that the network provides to the higher layers (e.g., transport) of the system*, since this may make it easier to "hide" diversity at the lower layers. So other than the functions that must be implemented with the help of end-points [102] (i.e., specific application semantics) all other functions are pushed down, affecting modularity of the system in the vertical direction.

As a second part of unbundling, we propose *decoupling network regions with very different properties*, such that properties of one region do not affect the other. For example, adequate in-network buffering can hide the losses that may be experienced in one region from the other region. Decoupling facilitates the deployment of customized solutions for each region as solutions designed for one region need not worry about the properties of other regions. Decoupling affects modularity of the system in the horizontal direction as it also involves the network in supporting some functions. Involving the network facilitates the deployment of in-network services as certain functions (e.g., connection management) can explicitly involve intermediaries. Moreover, nodes that implement decoupling can be used to insert additional functions (e.g., data oriented and higher level services) inside the network, thereby making use of advances in technology (e.g., cheap storage).

Figure 2.3 shows the impact of this unbundling of transport in both the vertical and horizontal dimensions. As depicted in the figure, raising the level of abstraction and decoupling regions with different properties can make the regions fairly "homogeneous". This can simplify the deployment of end-to-end solutions over homogeneous networks. This is rem-

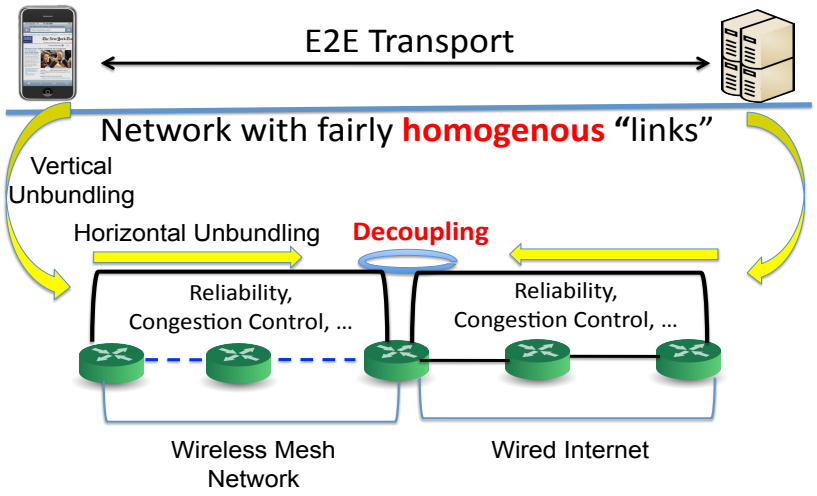


Figure 2.3: Unbundling transport in both the vertical and horizontal dimensions.

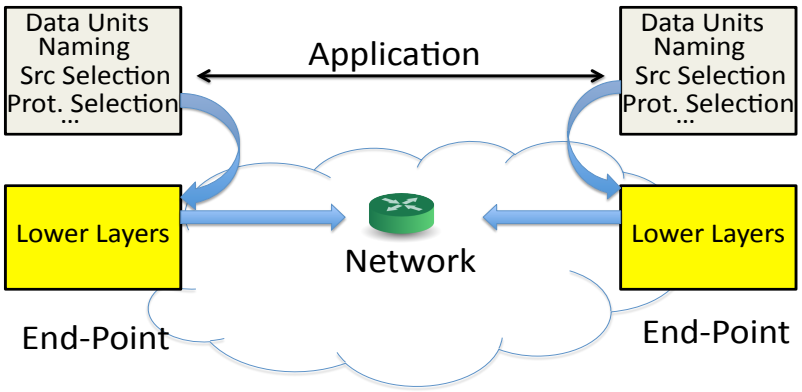


Figure 2.4: How application functions are exposed to the network in both vertical and horizontal directions.

iniscent of the original Internet where a light-weight transport (minus congestion control) worked over fairly homogeneous links.

2.1.2 Exposing Applications' Data Units to the Network

In today's Internet, applications typically need to perform several functions, including dealing with data and its naming, as well as communication related tasks, such as selecting the other end-point (e.g., server) and transport protocol (e.g., TCP/UDP). All these functions were important for applications that required *host-based* communication; their primary goal was communicating with another host and retrieving data was often a secondary/optional task for them.

Today, most applications are *data oriented* in nature, so they are interested in getting data irrespective of which host is used to retrieve it. Satisfying these requirements often entails using a variety of optimizations, such as opportunistically fetching content from different sources or caching data in the network [50, 52, 65]. Unfortunately, because information about the data units and its naming is restricted to the applications, these optimizations are often implemented in an application specific manner, thereby hindering widespread benefits of using data oriented services.

We propose exposing applications' data units to the network, which has implications on both the vertical (across layers) and horizontal (across the network topology) dimensions of the system. As shown in Figure 2.4, exposing it to the lower layers (below the application) of the end system affects the vertical dimension while exposing this information to certain elements within the network brings a horizontal angle. The vertical change allows *lower layers* of end-hosts to use various application independent data retrieval optimizations, similar to DOT [113] while the horizontal element allows the *network* to use various in-network services without requiring application specific help.

At a higher level, this change reflects the application's intent to allow lower layers of the end-point, as well as network services to perform certain actions on the data. For example, applications may not specify the other end-point, allowing the network to opportunistically fetch data from any source. Similarly, instead of the application specifying the transport protocol, the lower layers may make their own decision of which protocol or interface to use based on a variety of criteria e.g., expected performance or cost of using a link.

2.2 Tapa Design Overview

Tapa synthesizes the above concepts into a coherent architecture. As shown in Figure 2.5, it introduces the concept of *segments*, where a segment corresponds to a portion of an end-to-end path that is homogeneous (e.g., "wired Internet", a private network owned by an enterprise, a wireless mesh network, etc). Each segment provides best effort data delivery

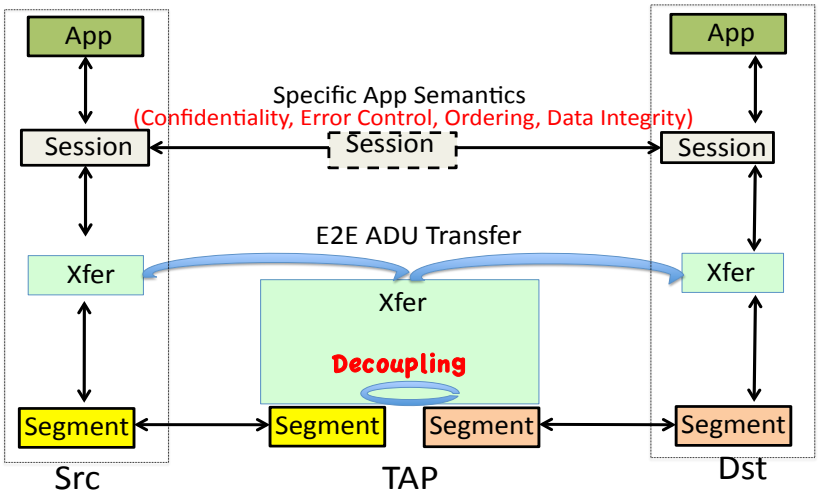


Figure 2.5: Tapa Overview.

service to the upper layer – functions that may be required to provide this service (e.g., routing, error control, congestion control, etc) are internal to the segment and hidden from higher layers. So segments can be viewed as Internet-like "links" and can use Internet-style protocols (e.g., TCP/IP-like protocols in the backbone) or custom solutions at the edges.

On top of segments is a transfer layer that inter connects segments, delivering application data units (ADUs) [45] from one end-point to the other. The transfer layer supports various services that run at the TAP – the most basic of these services is a *decoupling* service that isolates one segment from the other by providing adequate buffering at the TAP. As we discuss §2.4, we can also deploy various data oriented and higher level services on top of this basic decoupling service – for example, application independent caching and traffic shaping are two such data oriented services that can be supported.

On top of the transfer layer is the *session* layer, which implements application semantics between the two end-points. Optionally, it can also include an intermediary service, thereby providing new communication semantics that are not available in today's Internet. Tapa's session layer provides support for traditional functions like reliability, confidentiality, data integrity and ordering, but can include the role of intermediaries for each of these functions. So we can have session services, such as *delegation* or *controlled transparency* [47], that can change the semantics of the end-to-end communication. As we explain §2.4, these session services can also be used as a building block for higher level services, such as transcoding.

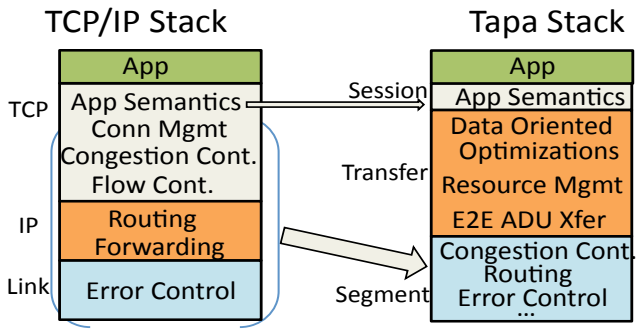


Figure 2.6: Comparison of protocol stacks.

Function	Description
Session Layer Interface for Applications	
`get(ADU, session + xfer semantics)`	Call used to pull an ADU.
`put(ADU, session + xfer semantics)`	Call used to push/publish an ADU.
Transfer Layer Interface for Session/Higher Layers	
`get(ADU, xfer semantics)`	Call used to pull an ADU.
`put(ADU, xfer semantics)`	Call used to push/publish an ADU.
`register(ID, handler)`	services/apps registering with the xfer layer to receive ADUs destined to them.

Table 2.1: Interfaces used at different levels of the Tapa system.

Figure 2.6 captures the changes introduced by Tapa by showing a comparison between today's TCP/IP stack and Tapa's protocol stack. Tapa's segment layer inherits most of the TCP/IP functionality, but implements it on a per-segment basis rather than on an end-to-end basis. Similar to TCP, Tapa's session layer implements application semantics, although these semantics are much richer compared to TCP's end-to-end semantics. Finally, Tapa's transfer layer supports data oriented functionality, which is not present in today's TCP/IP stack (although transfer services like DOT [113] support some of these functions).

We now provide an overview of the Tapa API and a discussion on the various services that run on TAPs. This is followed by a detailed description of the three Tapa layers.

2.3 API Overview

Tapa has several ingredients that make it necessary to have a new API that can allow applications to make full use its functionality. In this section, we give a high level overview of the APIs used at different levels of the Tapa system. We revisit the API design in

Chapter 6 and provide a more specific version of this API along with several examples to illustrate the use of the API.

Tapa's API allows two high level operations on ADUs: `put` and `get`. The `put` call allows *pushing* an ADU to a host or service while the *get* call allows retrieving an ADU based on the ADU identifier. As we elaborate in Chapter 6, there are certain default semantics associated with these operations, which reflect the common use cases for these calls. Moreover, as shown in Table 2.1, the interface also allows a higher layer to control the semantics of these two operations i.e., change the default semantics. This depends on the specific layer in question and the service it provides to higher layers of the system. For example, the transfer layer provides best effort ADU delivery and as a default, attempts to transfer ADUs as fast as possible. If the higher layers want a "slow" transfer, for ADUs that are not needed immediately, they can control the semantics through the API. Similarly, it is the session layer that provides a fully reliable service as a default, so if applications want a best effort service, they can change the semantics through the API. Finally, applications or services can also *register* their interest to the transfer layer for receiving ADUs. They specify their `id` as well as a handler (e.g., callback) to the transfer layer. This is the standard interface used by the higher level services that run on TAPs for receiving ADUs that are destined to them.

2.4 Services on TAPs

One of the primary goals of Tapa is to enable support for diverse in-network services. This thesis focuses on services that are typically supported at transport or higher layers of today's TCP/IP stack. As case studies, we show how Tapa facilitates a diverse range of data oriented services (e.g., application independent caching, traffic shaping, etc). These services build on top of the basic decoupling service that is provided by the TAP's transfer layer. As shown in Figure 2.7, other services logically sit at different levels of the Tapa system, depending on their semantics. Broadly, this depends on the service semantics and its positioning with respect to the Tapa APIs used at different levels of the system. If the service impacts the default semantics of a layer's API then it is logically at that layer (or at a higher layer) of the system. We explain this with the help of some examples.

Tapa's transfer layer provides best effort ADU delivery and its default semantics are transferring ADUs as fast as possible. So timing of ADU transfers is an issue that is the responsibility of the transfer layer. As a result, a traffic shaping service on a TAP, which may *increase* the transfer time of ADUs, is logically a transfer service because it changes the default semantics associated with ADU transfers. We can also imagine similar transfer

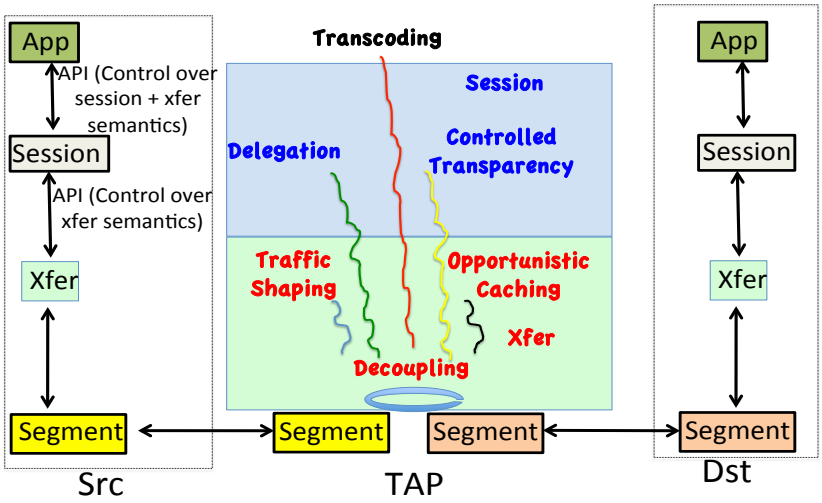


Figure 2.7: Different types of services in Tapa.

services that may conduct "slow" or DTN style transfers for ADUs that are not required immediately. Another service that logically sits at the transfer level is opportunistic caching because it improves the performance provided by the transfer layer's best effort service; such an opportunistic caching service at the TAPs can be transparent to end-points or can be explicitly invoked by the transfer service of an end-point, as we do in Swift (Chapter 3), to help mobile users.

We can categorize other services based on the same rule. For the session layer, any in-network service that impacts the session semantics is logically at the session (or higher level) of the system. For example, Tapa's session layer provides support for reliability, confidentiality, data integrity, and ordering. It also associates some default semantics with these functions – for example, end-points are responsible for reliability, so a sender only discards the data when it is acknowledged by the receiver. Similarly, it provides end-to-end data integrity checks and no confidentiality as its default semantics. Any in-network service that impacts the semantics associated with *these four functions* is logically a session (or higher) level service in Tapa. For example, a delegation service changes the default reliability semantics, as the service takes over the responsibility of reliably sending ADU to the receiver, so the sender can potentially discard data after it has been acknowledged by the delegation service. Similarly, a controlled transparency service provides selective confidentiality, such that ADU is visible to the in-network service and the two end-points,

but is encrypted for any other element on the end-to-end path. We provide a detailed design of a session protocol that accommodates such session services in Chapter 6.

The above view of looking at services at different levels of the Tapa system also suggests that we can build higher level services by using lower level services as building blocks. For example, a transcoding service can be built on top of the session services, like controlled transparency or service that verifies the integrity of the data, and as a result it need not worry about implementing these lower level semantics. Instead it can focus on higher level functionality, which is not covered by the lower level services. Similarly, session level services can in turn use transfer services as basic building blocks. For example, delegation, which is a session level service, can be built on top of opportunistic caching (a transfer service) or it can have its storage/reliability functionality. In our prototype, session level services, like delegation and controlled transparency, build on top of caching, so they need not re-implement the logic for storing ADUs.

2.5 Segment Layer

The segment layer is responsible for transferring data across a segment, e.g., client - TAP or TAP - server, providing a best effort service to the higher layers. The internal details of how this is achieved is left up to the segment layer.

Segments can choose the data granularity they use internally (e.g., frames, bytes, etc.) allowing them to optimize communication as appropriate. Segment endpoints do not move, so they *can* use network-specific locators as addresses. For example, in the wired Internet IP addresses based on CIDR may provide the necessary scalability, while MAC addresses may be more appropriate in wireless networks.

Segments must implement error, flow, and congestion control as needed. Tapa's transfer layer expects segments to be *reasonably* reliable, but segments can use very different ways for achieving that, e.g., TCP style retransmissions and ACKs over wired segments versus in-network coding or opportunistic forwarding and block acknowledgements on a wireless segment [38, 81]. It is similarly well known that congestion control mechanisms need to be tailored for different types of networks [29, 81]. Some segments may need specialized routing protocols for delivering data over multiple hops within a segment (e.g., wired Internet or mesh network). Some segments (single hop, point to point) may not require routing or congestion control. The important point to note is that all these functions are implemented *inside* a segment and hidden from upper layers.

Figure 2.8 shows some possible ways to implement these functions for wired and wireless segments. On the wired side, we can use variants of existing protocol stacks. For example,

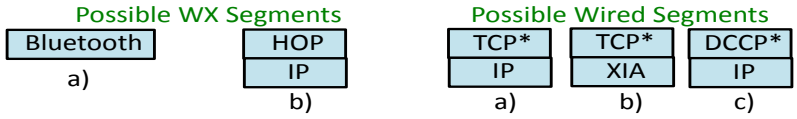


Figure 2.8: Possible Segment Protocols for Wireless and Wired Segments

existing transport protocols (e.g., TCP, DCCP [71]) can be used with minor modifications to satisfy the "best effort" delivery requirement of a Tapa segment. These modified protocols can work over different network layers, such as IP or XIP [23][1].

On the wireless side, we can use protocols that are customized for the particular technology or environment. For example, if we have a wireless mesh network, we can use a segment protocol like HOP [81], which relies on per-hop mechanisms for reliability and rate control. Segments can also bypass IP and traditional link layers, using options as diverse as blue-tooth and wireless USB. This is a direct benefit of raising the level of abstraction – not all networks and devices need to use TCP/IP in order to be part of the Internet. In Tapa, mobile clients can run light-weight customized protocol stacks for the segment technologies they use to communicate with a TAP, without worrying about heavy-weight functions that are required to communicate over the Internet (e.g., TCP style congestion control). In Chapter 3 (Swift), we present several segment protocols that illustrate this flexibility for wireless users.

2.6 Transfer Layer

2.6.1 Overview

The transfer layer is the "inter-networking" layer of Tapa and its role is somewhat similar to that of IP in today's Internet: providing a best effort data delivery service over multi-hop paths, but there are several differences in both the control and data plane. For example, Internet routing needs to establish routes in large scale but fairly stable networks; in contrast, the Tapa transfer layer establishes short (e.g., two-segment) paths but paths can be very volatile due to the dynamics of the access network (e.g., mobility and wireless dynamics). The introduction of the transfer layer is in part to allow for a separation of concerns. The segment layer can focus on the challenges associated with specific networks (e.g., scalability within core Internet) while the transfer layer can focus on dealing with

[1]We discuss the implications of using Tapa over content centric architectures like XIA in §2.8

higher level challenges (e.g. selection of segments based on content or service availability, or mobility).

Tapa's transfer layer is based on *application data units (ADUs) [45]*, so unlike IP, which delivers datagrams, the transfer layer is responsible for delivering ADUs between two end-points. Also, unlike IP, which supports a push based model for sending data, the transfer layer supports both *pull* and *push* models, thereby catering to both data oriented applications as well as interactive applications. The transfer layer also supports various in-network *services*, that run at the TAPs and may also involve the end-points. Finally, the transfer layer also undertakes *resource management* to ensure that network and TAP resources are used judiciously.

2.6.2 Application Data Units

ADUs offer a middle ground between existing applications that generally use containers (e.g., file names) and IP, which uses datagrams. In Tapa, ADUs can be defined in a flexible manner based on the requirements of the application. For example, an ADU could correspond to a whole file (e.g., Catnap[55]), to chunks within a file (e.g., DOT[113]), or an MPEG frame in a video transfer.

Associated with an ADU is its identifier, which is useful for retrieving an ADU based on its name. The identifier could be based on any naming convention, such as content hash [72, 113], hierarchical addressing based on URIs [65], etc. In our specific design, we use content hashes to identify an ADU but Tapa can use other options for naming, or can even use more than one option, as long as these options are standardized i.e., broad rules regarding length of the identifier, how to specify different naming types, etc are pre-determined.

We use content based hashing (CBH) because it offers several benefits for data oriented applications. CBH is a known technique that has been leveraged in several prior systems, including DOT [113], DONA [72], and many P2P systems [48]. CBH requires the use of a well known hash function, such as SHA1, to generate a globally unique identifier for a given content. Anyone who has the identifier can retrieve the content, even from an untrusted/arbitrary host, and still verify that it received the correct content (by comparing the hash of the received content with the requested identifier). This is useful for data oriented applications that care about receiving the correct data irrespective of which host is actually used to retrieve it.

In addition to the identifier, we also associate *hints* with an ADU, which broadly relate to any piece of information that the application can provide to the lower layers/network to

facilitate the transfer of the given ADU. In this thesis, we have used the size of the ADU and possible sources from which the ADU can be retrieved (e.g., a nearby cache or a peer) as hints in different contexts, but other information such as the delivery requirements of an ADU can also be considered.

As we will see throughout this thesis, ADUs play a central role in Tapa as they enable support for various data oriented and higher level services (e.g., application independent caching), in addition to facilitating improved resource management within the network (i.e., congestion control and related tasks). They also form the basis of a new API that applications use to retrieve and publish data.

2.6.3 Basic Operations

The transfer layer supports two modes for data transfer: i) pull mode, which retrieves the ADU corresponding to an ADU identifier and is therefore suitable for data oriented applications, and ii) push mode, which sends an ADU to a given destination and is suitable for interactive applications or any non-data oriented task that require sending message to a specific destination.

In the **push** mode, the transfer layer of the source receives the ADU and the identifier corresponding to the destination through a `put` interface, which is similar to the API used in prior content addressable and data oriented systems [99, 111, 113, 119]. It has the option of creating a direct segment with the other end-point, or it can leverage TAPs, in which case, it must be able to discover TAPs that may be useful in assisting with the communication.

We can consider a variety of mechanisms for TAP discovery, including help from lower layers (e.g., similar to AP discovery in WiFi), via service discovery protocols [12], resolution services [116], or the application providing this information. Swift (Chapter 3), which is designed for wireless and mobile users, leverage lower layer information for TAP discovery while Vigilante (Chapter 5) uses information from the application regarding the social network for a user to discover other TAPs. Through this TAP discovery process, the transfer layer collects information about the TAP's identifier and the services it offers (e.g., type of segments, support for caching, etc).

Tapa requires that the identifiers for both end-points and TAP(s) be globally unique. This is different from IP which uses topological addresses that refer to the location of the host. We use identifiers because separating identifiers from locators is important for mobility, which is one of the main considerations for Tapa.

An implication of using identifiers at the transfer layer but allowing the segment layer to use locators is that the segment layer must be able to translate these identifiers into locators

that can be used to establish the segments; locators may only have meaning locally within a network, i.e. they do not have to be globally unique. In our prototype, we use host-names as identifiers for convenience, but self-certifying identifiers would be more appropriate for an actual deployment [27]. The identifiers are mapped to locators using either a naming service service (e.g., DNS for wired Internet segments), or based on a locally maintained mapping to internal Addresses (e.g., MAC addresses for a bluetooth based segment).

Once the transfer layer has all the identifiers, it is in a position to establish an end-to-end path between the client and the server passing through the TAP. In the typical scenario, the data plane of the transfer layer is relatively straightforward: TAPs read data from one-segment and write to the other, ensuring that adequate buffering is provided; end-points (and optionally TAPs too) need to reassemble ADUs and deliver to higher layers.

The **pull** mode is similar to the push mode in most aspects, such as how segments are established or identifiers are resolved, etc, but there are some differences in both the API as well as the operations. First, higher layers use a `get` interface to specify the identifier corresponding to the ADU that they want to retrieve. The ADU may also have hints regarding nodes who may have the ADU, but the transfer layer is not obligated to follow them. In the pull mode, the transfer layer can act in an opportunistic manner. So it can serve the ADU from its local cache or forward the ADU request to some other node. Once the ADU request reaches a node that has the ADU then the response is sent back to the source, similar to how an ADU is sent in the push mode.

In both the pull and push modes, the selection of TAPs and segments is done by the transfer layer and is usually transparent to the application. However, in some cases, the application may want more control over the decisions that are normally left up to the transfer layer. For example, an application may want to use a specific TAP or a segment, similar to how applications today control the transport protocol that they want to use. Even though we don't expect this to be the normal case, there should still be provision in the API to allow applications such control. In Chapter 6 (Semantics), we discuss how Tapa's API is able to provide such low level control to the applications.

2.6.4 Transfer Services

TAPs can offer various services that can supplement the basic operations of the transfer layer. We present some examples of such services as case studies in later chapters. Here we just provide a brief overview of how different kinds of services can be supported at the transfer layer.

As shown earlier in Figure 2.7, the most basic service is a *decoupling* service that isolates

one segment from the other by providing adequate buffering at the TAP. An elaborate form of this decoupling service can also provide support for disconnections by storing ADUs in a persistent storage while mobile clients are disconnected (Swift - Chapter 3). We can also imagine a service that controls the timing of ADU transmissions on different segments to provide energy savings (Catnap - Chapter 4).

2.6.5 Resource Management

Although congestion control is implemented inside segments, the transfer layer also needs to ensure that TAP buffers do not overflow. We include this broadly under *resource management*, which includes congestion control for legitimate sources as well as protection against malicious sources (e.g., DoS attacks).

Several features of the Tapa architecture provide support for a a flexible and holistic approach towards resource management, enabling TAPs to consider a variety of solutions, including congestion avoidance and control. For example, Tapa's use of ADU hints, which include length of the ADU, provides information to the TAPs about future traffic load, which can be used to do admission control. TAPs strategic location at network edges means that they can also coordinate with end-points to install filters against potential DDoS attacks, thereby acting as a first line of defense [83]. The limited number of segments in an end-to-end path means that it is easier for TAPs to coordinate amongst each other in order to avoid congested segments. It also simplifies the use of many existing techniques, such as hop-by-hop flow control [78] or end-to-end congestion control based on feedback from the network [69, 96]. In this thesis, we mostly consider two segment paths and therefore use hop-by-hop flow control based on back-pressure [78] in our prototype implementation. However, future work should also explore the other options discussed above as they offer a promising ground to address a number of important resource management challenges in a holistic manner.

Finally, note that many of the above solutions are only applicable because we expect TAPs to be mostly used at the network edges, so they don't face the scalability challenges typically faced by a core router. However, it is still important that segment and transfer protocols are *light-weight*, in terms of the amount of state maintained at the TAPs, and *secure*, so they do not create new vulnerabilities.

2.7 Session Layer

Similar to IP in today's Internet, the segment and transfer layers offer a best effort service to the transfer and session layers, respectively. This means that they deliver data with high probability, but delivery is not guaranteed. The motivation for the best effort nature of the segment and transfer layers is the same as for IP [102], i.e., full network-level reliability is expensive and not always needed. Therefore, Tapa's session layer still provides reliability and other application semantics (e.g., ordering, confidentiality, data integrity, etc) to the end-points, albeit over a very short path consisting of a small number of segments. Important thing to note is that we can easily implement support for different semantics (e.g., partial reliability or out-of-order delivery) as the session layer need not re-implement network specific functions, like congestion control, which are coupled with application semantics in TCP.

Another important consideration is that TAPs are visible to the end-points, opening the door for richer semantics that explicitly capture the role of services running on the TAP. At the session level, we can support services such as delegation and controlled transparency, which are useful on their own but also can be used as building blocks for higher level services, such as transcoding or virus scanning (as shown earlier in Figure 2.5). Note that these session services are themselves implemented on top of transfer services, like application independent caching or simple decoupling, showing how Tapa is able to systematically support a wide range of services that build on each other. In Chapter 6 (Semantics), we discuss the session services in detail and present the design of a session layer that accommodates the various services mentioned above.

2.8 Related Work

Tapa's design is inspired by a large body of work, including the design of the original Internet, split connection approaches, overlay networks, and proposals that deal with visible middleboxes [29, 44, 53, 102, 110]. We give a brief overview of the most relevant proposals, focusing on the key differences with Tapa.

Split Connection Approaches: A common theme in supporting wireless users with poor links or limited capability is to add customized support at the wireless middlebox (I-TCP [29], PEP [39] etc). Like Tapa, these approaches are also able to decouple different segments, although Tapa provides a broader form of decoupling with the use of large buffers in the form of caching/persistent storage. The fundamental difference, however, arises be-

cause the split connection approaches try to make the splitting transparent to applications, as well as to end-point transport protocols, resulting in several problems.

A major problem with these approaches is that the end-points no longer communicate directly with each other, so we can no longer support end-to-end semantics that are required for the correct working of the application. One such example is data integrity, which is only supported on a per-segment basis in the split connection approaches through the use of checksums. This does not address any errors that may occur at the TAP and end-points will not be able to detect such problems. This is a classical example of the end-to-end arguments which argue that functions such as data integrity must be implemented end-to-end, if any errors during the transmission have to be detected properly. Similar semantics issues arise for other functions as well if we use split connection approaches. For example, for reliability, an acknowledgement from an intermediary may be seen as one from the other end-point causing the sender to discard data even though the other end-point may not have received the data.

Another set of problems arise due to the "hard state" maintained at the intermediary in these split connection approaches in the form of transport connection state for each of the connections that the intermediary maintains with the end-points. This results in new failure modes because if the intermediary fails then the end-to-end connection also breaks. Similarly, mobility also becomes a challenge as this state needs to be transferred to the new intermediary if one of the end-points move. Finally, as this state is hidden, it often interacts poorly with new techniques that may be added at end-points (e.g., multiplexing multiple interfaces for an end-to-end transfer).

Tapa does not have any of these problems as TAPs/intermediaries are explicitly visible to end-points. So TAPs do not break the end-to-end semantics, their failures can be handled, and mobility is easy to support as it does not require any transfer of state between TAPs.

Overlays: Overlays have been used to support diverse functions and optimizations, such as path redundancy, throughput optimization, multi-path transfers, and content sharing [28]. Tapa can be viewed as an overlay, although it has very unique characteristics. For example, unlike traditional overlays, Tapa's topology is highly constrained, so we do not need a routing protocol. Wireless segments can be short-lived, which can result in very dynamic topology. Segments in Tapa mostly line up with network boundaries and as a result, Tapa segments may use very different technologies, e.g., IP on the wired segment and custom protocols on the wireless segment. Finally, the role of the the two end-points (e.g., mobile client and a fixed host) is very asymmetric. All the differences require new mechanisms not found in other overlays.

Disruption Tolerant Networks (DTN): DTNs [58] are also specialized overlays that are designed to handle arbitrary disruptions and delays. In contrast, Tapa's focus is only on dealing with disruptions at the edges, which are typical in most wireless and mobile access scenarios. This difference in focus results in several differences between Tapa and DTNs. For example, DTNs do not support end-to-end semantics and only support delegation, which is an optional session service in Tapa. Moreover, DTNs employ push-based routing which results in poor performance under good connectivity scenarios whereas Tapa can employ a pull mode and make use of various data oriented optimizations.

Transfer Services : Tapa's transfer service leverages several concepts used in DOT [113], although we focus on the challenges and opportunities specific to wireless and mobile users. Tapa leverages the concept of TAPs and brings a strong spatial aspect to data transfers which is essential for wireless and mobility optimizations. Also, in order to provide rich transport semantics, our transfer service is implemented *below* end-to-end transport whereas DOT works on top of existing transport layer protocols.

Recently, Popa et al [92] propose the use of HTTP as the narrow waist of the Internet, so it can be viewed as a transfer service. Like Tapa, HTTP can also accommodates higher level middleboxes (i.e., proxies) that can implement data plane optimizations like Catnap. However, due to inherent limitations of HTTP (i.e., naming, rigid semantics, etc), it is difficult to exploit content centric and multi-path optimizations offered by Swift and Vigilante. Moreover, we also define the roles of layers below and above the transfer service i.e., segment and transport layers, which play a key role in the Tapa architecture.

Data Oriented Architectures The move from host based applications to data oriented applications has been recognized by the research community and there have been several proposals for data oriented network architectures (e.g., CCN [65], XIA [23], DONA [72], etc). These architectures enable routing on content identifiers rather than host identifiers, which is similar in spirit to Tapa's pull mode. If Tapa is implemented over such data oriented architectures then Tapa's transfer layer need not support such "routing" based on ADU identifiers as the underlying network will support content lookup in the most optimized manner. This will simplify Tapa's transfer layer and will also improve the performance of retrieving content.

Tapa most naturally fits over architectures like XIA which provides support for host and service based communication in addition to data oriented communication. This is important because Tapa also has a push mode where an ADU needs to be sent to a specific destination. Tapa's push mode can make use of XIA's support for host based communication

in order to enable communication between two end-points. Moreover, Tapa can also leverage XIA's support for services, simplifying several tasks associated with services (e.g., discovery, resolution, etc) that are required in Tapa. Using XIA, Tapa can directly route on service identifiers, instead of requiring to know a specific host on which the service is running. This enables late binding to service names, avoids an additional level of indirection and also supports service migration. Moreover, XIA's intrinsic security addresses a number of security challenges, such as TAP authentication. So if Tapa is used over XIA, the end-points can be assured that they are communicating with the right TAP or service.

Visible middleboxes: The use of TAPs in Tapa is inspired by a large body of work related to making middleboxes, such as firewalls and NATs, *visible* to the end points – through appropriate routing, addressing, and signalling mechanisms [61, 116]. Our work shares the concern of earlier work that hidden middleboxes can be a source of problems, although we focus on using middleboxes to specifically optimize end-to-end transfers. As a result, we are mainly concerned about "flow middleboxes" that carry transport state (e.g., proxies or other middleboxes that use different transport regimes for an end-to-end connection). This is different from proposals like NUTSS [61] and DOA [116] that deal with network-level middleboxes i.e., ensuring that middleboxes become part of routing and addressing and can therefore process packets (e.g., NATs, firewalls, etc).

The work that is most relevant is the proposal by Ford and Iyengar [60] who break-up the transport layer to accommodate flow middleboxes in the end-to-end path. Tapa is a complete architecture which provides a broader form of decoupling of segments, allowing use of non-IP protocols within a segment. Tapa also uses the concept of ADUs, and provide support for reduced synchronization, mobility, and delegation.

Chapter 3

Supporting Mobile and Wireless Users

The original Internet comprised of fixed hosts that mostly used wired networks to access the Internet [79]. Today, most of the users access the Internet through some wireless device (e.g., laptops, PDAs, smart-phones,etc), and Internet access in a mobile environment has become quite common. The lack of architectural support for wireless and mobile users in today's Internet is performance limiting and also makes it hard to build new applications and services that can push the limits for mobile and wireless users.

One of the main goals of Tapa is to provide explicit support for mobile and wireless access. Tapa addresses the key challenges involved in wireless and mobile access, and also leverages the opportunities that are present in these settings. In order to highlight these benefits, this chapter presents Swift, a prototype design and implementation of Tapa, which is specifically tailored for mobile and wireless users. We first provide a motivation behind Swift by describing the key problems that wireless and mobile users face in today's Internet. This is followed by an overview of how Tapa can benefit wireless and mobile users. In §3.3 and §3.4, we present the design and implementation of Swift, respectively. We present the results of a comprehensive evaluation of Swift in §3.5. Finally, we present related work and summarize.

3.1 Motivation

Wireless networks and mobile access present their own set of challenges and opportunities [29, 31, 38, 68, 70]. Unfortunately, in today's Internet, it is hard to overcome these challenges and leverage the available opportunities. In this section, we separate out the

key challenges and opportunities with regards to wireless networks and mobile scenarios, respectively.

3.1.1 Wireless in today's Internet

Wireless networks are quite diverse in terms of their characteristics compared to traditional wired networks [29, 31]. Some of the important differences in characteristics that have impact on the performance of transport protocols include variation in bandwidth, loss rate and latency. For example, some wireless networks have very low bandwidth (e.g., Zigbee [18]), on the order of few kbps, while other wireless technologies (e.g., 60 GHz band [18]) can offer up to gbps in bandwidth. Similar variations exist in latency as typical delays in 3G networks are much higher compared to 802.11 based wireless networks. Moreover, as wireless is a shared medium, there could be significant variations in bandwidth, latency and other factors over time for the *same* wireless technology.

It is well known that TCP does not perform well in many of the above scenarios, such as those involving high bandwidth delay-product networks, or networks with large variations in bandwidth or latency [69, 96]. Such performance degradation is especially more pronounced in some wireless scenarios, such as wireless mesh networks [37, 97], vehicular networks [63], etc. As a result, new protocols have been proposed as an alternative to TCP, which are designed for specific networks or environments (e.g., mesh networks [81], vehicular environments [56], etc). However, in most scenarios, wireless networks are used at the *access only* and a typical end-to-end scenario involves the wired Internet as well. This makes it challenging to design an end-to-end protocol that works well for *both* the wireless network as well as the wired Internet. As a result, even though TCP does not offer optimal performance in many wireless networks, it is still used as an end-to-end transport protocol in many scenarios.

In addition to these challenges, there are opportunities that are typically present in most wireless access scenarios, but are difficult to leverage in today's Internet. One such opportunity is multiplexing multiple access points to aggregate uplink bandwidth. This is especially useful in residential scenarios where the wireless link offers much higher bandwidth compared to the wired link. This combined with the observation that wired links in a home's neighborhood are usually idle, makes it attractive to aggregate the uplink bandwidth of residential APs [67, 68]. Another similar optimization involved multiplexing multiple interfaces (e.g., 3G, WiFi, Bluetooth) to improve transfer throughput or to gain other benefits (e.g., improved energy efficiency) [89, 114]. These optimizations are hard to implement in today's Internet because of the end-to-end nature of today's transport and

once implemented are usually tied to a specific transport protocol (e.g., TCP) or specific underlying technology (e.g., WiFi) [68], making it hard to introduce new protocols in future.

3.1.2 Mobility in today's Internet

In today's Internet, performance under mobile scenarios is often much poorer compared to stationary scenarios. This is due to a combination of lack of support within the network for mobility as well as the use of end-to-end protocols that are not suited for mobility (i.e., TCP). For example, the network still lacks any support for vertical hand-offs, which results in disconnections during typical mobile scenarios. Similarly, there is no in-network support for disconnections, which is typically provided through the use of extra storage at network elements. At the end-points, the transport protocol (i.e., TCP) is tied to a specific location (i.e.g, IP address), which makes it difficult to handle mobility in the middle of transfers. Moreover, TCP is sender driven, which is undesirable as typical scenarios today have the mobile host as a client/receiver, so it is in a better position to know about its changing environment and therefore should be in-charge of data transfers.

In addition to the challenges, mobile data transfer scenarios also have opportunities that are difficult to leverage in today's Internet. As discussed earlier, most applications today are interested in getting data irrespective of which node is used to retrieve it. In a mobile scenario, the device typically comes in the range of several other devices/nodes that can potentially act as sources of data. Unfortunately, such opportunities are lost as data retrieval in today's Internet is tied to a specific host, so even if the mobile client moves, it continues to download the data from the original source, even if that source is no longer the optimal one.

3.2 Benefits of Tapa

Tapa can help a mobile and wireless client in three broad ways. First, a wireless device can use a variety of options as segment protocols for the wireless segment without worrying about making them work for the entire end-to-end path (including the wired segment). For example, if we have a wireless mesh network, we can use a segment protocol like HOP [81], which relies on per-hop mechanisms for reliability and rate control. Segments can also bypass IP and traditional link layers, using options as diverse as blue-tooth and wireless USB. This is a direct benefit of raising the level of abstraction – not all networks and devices need to use TCP/IP in order to be part of the Internet. In Tapa, mobile clients can run light-weight customized protocol stacks for the segment technologies they use to

communicate with a TAP, without worrying about heavy-weight functions that are required to communicate over the Internet (e.g., TCP style congestion control).

Second, the combination of a higher level API, which is not tied to a specific protocol or technology, and its use of ADUs and segments, makes it easier for Tapa to leverage wireless optimizations that involve multiplexing of multiple interfaces or access points (TAPs in this case). In the common case, application just specify the ADU(s) they want to transfer and it is up to the lower layers to select the appropriate segment(s). As a result, multiplexing of segments is naturally supported; the transfer layer decides an appropriate segment for *each* ADU; so for the same application, it can fetch some ADUs using one particular segment while other ADUs can be retrieved using a different segment.

Third, the transfer layer can improve data transfer performance for mobile clients through caching and intelligent use of segments. For example, imagine a mobile user who is downloading a large file from a slow server. Initially she is using her home TAP but in the middle of the download she moves to Starbucks. Her home TAP will continue to download ADUs from the server and cache them in its storage; after she moves to Starbucks, the transfer layer will re-establish a segment to her home TAP via the Starbucks TAP and retrieve the missing ADU (by re-issuing requests for the missing ADU ids). This will be a much faster option compared to going all the way to the slow server to retrieve the missing ADUs.

3.3 System Design

We have designed and implemented, Swift, a prototype of Tapa that realizes the above mentioned benefits for mobile and wireless users. Our design focuses on implementing the key features of Swift and how they interact with each other rather than optimizing individual modules. The major component of our prototype is a service that runs on the mobile client and logically sits at the transfer level. This transfer service makes use of an in-network caching service and uses various options that are tailored for specific wireless environments as its segment protocols. Swift can use a thin session layer that has the minimum default functionality needed to support end-to-end transfers in Tapa. Similarly, Swift uses the Tapa API earlier presented in §2.3 with the default session and transfer semantics.

As the transfer layer implements most of the functionality required to support Swift, we focus on its data and control planes and finally briefly discuss the role of the session layer.

3.3.1 Transfer Layer - Data Plane

The data plane of Swift supports the basic functionality required in Tapa. In this section, we explain those functions again with a mention of how they are helpful in a wireless and mobile scenario. We describe key features of the data plane, focusing on transfers from the server to the client; the reverse path is similar.

Transfer An ADU transfer typically touches the data planes of three nodes: server, TAP, and the client, and uses two segment layers: one connecting the server to a TAP and one connecting the TAP to the client. The data plane at the server creates the ADUs and delivers them to the TAP. In the simplest case, the data plane on the TAP forwards the ADUs between the two segments, which could simply entail reading from one segment and writing to another or maintaining a shared queue between the two segments. Adequate buffering is important for proper decoupling of segments; this is even more important in a wireless context where the properties of segments could be drastically different, thereby requiring more buffer space than usual. An important point to note is that the additional state at the TAP is treated as *soft* state, so even if a TAP fails, it is still possible to recover data in an end-to-end fashion. This is important because mobility may require a mobile client to change TAPs and having hard state at the old TAP would make such a move difficult.

The data plane on the client is responsible for handing the ADUs to the session layer which in turn presents them to the application according to the required semantics (e.g., ordering). The data plane also provides feedback about the transfer progress to the transfer control plane which uses this information for error recovery, e.g. after a TAP failure or change of TAPs due to mobility.

Naming As discussed earlier, we use self-certifying names for ADU which are based on their content hashes. It helps in "content-centric networking" where ADUs can be retrieved from many different sources in an opportunistic manner; this cab be very useful in a mobile environment where a mobile device can have opportunistic contacts with various devices [105].

Transfer Modes Swift supports both the pull and push modes for ADU transfer. The pull mode is especially useful in a mobile environment as it allows the mobile client to be in-charge of the transfer decisions. Push mode is required to bootstrap things or to support applications that are not suitable for the pull mode of data transfer (e.g., interactive applications).

3.3.2 Transfer Layer - Control Plane

As mentioned earlier, Tapa's pull mode allows the client to be in-charge of the transfer as it is in the best position to know about its changing conditions and to make decisions accordingly. At a high level, the control plane of the transfer layer is responsible for the following tasks: collecting information about connectivity options in the wireless network(s), managing segments for use by the data plane, and error recovery.

Collecting Network Information

We support three tasks that are most relevant in the context of wireless and mobile users.

TAP discovery involves identifying TAPs that can be used for data transfer. This process can involve discovering TAPs that can *currently* communicate with the mobile-host or TAPs that may be encountered in *future*. In the first case, information from lower layers can be used (e.g., beacons) while in the second case, user input regarding its mobility plans can be combined with publicly available hotspot locators or previous history to know the likely TAPs that may be encountered in future. The outcome of the discovery function is a list of candidate TAPs, plus a mechanism to communicate with them, either directly or through some other TAP.

Probing involves finding out the path properties between the mobile host and the TAP. The goal is to anticipate the expected performance while communicating with the TAP. This process can be passive (e.g., using signal strength information), active (e.g., typical bandwidth estimation techniques), or based on historical experiences (e.g., performance received during past encounters with the TAP). Passive techniques have lower overhead, but active probing may need to be invoked on demand to get more accurate information.

Data and service discovery involves getting information about the data that is present in a TAP's storage and other services it may offer. This function could be very light-weight or more sophisticated depending on the requirements. A light-weight procedure can involve the TAP providing hints regarding caching functionality and possibly nature of contents. For example, total size of the storage, proportion of different categories of data (entertainment, news etc.) can be useful for the mobile host. In contrast, a more heavy-weight process may involve actual exchange of ADU ids (complete list or a bloom filter).

Managing Segments

Upon receiving a request to transfer an ADU, and assuming no segments are available, the control plane needs to establish a segment to transfer the ADU. Subsequently, it will also

monitor progress and TAP availability to adjust segments as needed.

Initial Segment Selection: The initial decision is based on the following information: ADU requirements, spatial resources available (TAPs and services they provide), and the connectivity to the available TAPs. For example, the properties of a 3G interface in terms of throughput, latency, and reliability, and cost, are very different from WiFi. In the general case, the decision outcome comprises a suitable TAP and interface.

Authentication: Authentication may be required to establish a segment. Once a TAP is discovered, the control plane can initiate an authentication phase which can authenticate one or both the parties to each other. Moreover, the authentication credentials can also be used in an enterprise setting for handoffs. This function will have to be expanded in future implementations of Swift, e.g. by adding support for common forms of access control found in hot spots and campus deployments.

Monitoring and Adaptation: There are two aspects to monitoring. First, the data plane reports the status of ADU transfers, providing information about future transfer needs. Second, the control plane collects information about the status of existing segments (e.g. based on keep-alives, signal strength, or observed throughput) and opportunities for new TAPs and segments (e.g. through the background TAP discovery progress). Based on the monitoring, the control plane decides whether it needs to take any action, such as switching to a new TAP. In case of mobility, this may be *required* while in other scenarios it may be beneficial for performance reasons. A segment is set up with the new TAP and request for missing ADUs (as indicated by the data plane) are sent over this segment to the old TAP, or to the server if needed. Another case, namely TAP failure, is discussed below.

During a transfer, is is possible that the control plane detects another segment that can be used in parallel with the existing segment. Examples of such scenarios include residential wireless scenarios where the up-link bandwidth of APs can be aggregated to improve throughput. Another similar opportunity is the simultaneous use of multiple interfaces, which are typically present in today's mobile devices. If such an opportunity exists, the control plane will establish another segment and will coordinate with the data plane on how to "stripe" the ADUs over the available segments.[1]

[1] Our system evenly distributes the ADU requests if multiple TAPs with similar performance are available.

Error Recovery

The transfer layer can encounter errors of different degrees. The loss of segments due to mobility or changes in the wireless network should be handled as a routine event, as described above. A more extreme version of this event is the failure of a TAP. It differs in two ways. First, there is no warning that the segment will disappear, so it is more difficult to transition to a new segment, thereby increasing the likelihood of disruption in connectivity. Second, the state on the TAP is lost, but as the transfer layer is based on soft state, the main impact is that any ADUs stored on the TAP will have to be retrieved from the source.

Recovery of ADUs that are lost due to TAP failure or handoffs involves both the control and data plane. The data plane reports the missing ADU ids – the decision of how and when to undertake the recovery is based on application requirements and spatial conditions. For example, if the application has latency requirements then the timeout value used to detect loss would be small. The spatial conditions help determine the cause (e,g., TAP failure or hand-off) and provide information about other alternate options. Based on these factors, the server may be contacted again through a new TAP (if the old one failed) or through the old TAP again. In contrast, if the user moved, it *can* ask the new TAP to get the ADUs from the previous TAP.

3.3.3 Session Layer

Swift's end-to-end session layer is thin and implements two functions: *reliability* and *ordering*. Reliability semantics are provided in the form of end-to-end acknowledgements that can acknowledge a single ADU or multiple ADUs. Of course, these acknowledgements are also considered an ADU and are therefore transferred in the same way. In rare cases where the transfer layer is unable to recover lost ADUs (e.g., ids of the missing ADUs are not known or repeated TAP failures) then it is left to the end-to-end session layer of the sender to timeout and initiate a resend. These timeouts can be coarse grained as the underlying transfer and segment service provide reliability in most cases. Similarly, many applications also require ordering of data (ADUs), so we provide this support. Note that the session layer may temporarily store the ADUs in a persistent storage if there is significant reordering and it is short of buffers.

3.4 Implementation

We have implemented a prototype of Swift in C++ for a LINUX environment. It operates as a user level server with one binary that supports three different modes: mobile-host, TAP,

and server. The prototype implementation is multi-threaded and consists of approximately 5000 lines of code. The communication with applications is through UNIX sockets while the communication between threads uses shared memory, with locks used for synchronization.

The common functionality shared by all three modes includes most of the transfer layer data plane functionality. This includes support for caching ADUs, looking them up or transferring them to another node. The server includes functionality for creation of ADUs (other nodes can also support this functionality). This support is provided to enable legacy applications to create ADUs and therefore, is provided as an application library. We currently only support ADU creation for static files. Applications specify the file and ADU size, and fixed sized ADUs are generated from the file. We use `SHA1` of the ADU contents as the ADU id. All ADUs are stored as separate files on disk with their id as the name of the file. There is a meta-file that contains information about the mapping of a file to the ADUs with their appropriate location in the file. On receiving a request, this meta-file is consulted to get information about the ADU ids corresponding to a file. Finally, ADUs can be created a-priori or on demand (i.e., when a request arrives).

The unique functionality in the mobile-host includes customized support for TAP discovery in a WiFi environment. There are two ways of discovering TAPs: i) users can specify a file containing information about TAPs and ii) live discovery of TAPs. For live discovery, we initially used the `iwconf` tools to get information about neighboring WiFi nodes. However, this tool gave us timing granularity of seconds which was not sufficient for quick switch-overs. Therefore, we have built a live discovery tool that allows us to do fine-grained TAP discovery.

The live discovery tool sniffs packet in monitor mode using `libpcap` [13]. It parses the 802.11 header for the MAC address and *received signal strength indication* (RSSI) [11] information, and primarily looks for three things: presence of a new TAP, departure of a TAP, and change in RSSI behavior of a node. A new TAP is discovered if you hear a new node transmitting. We assume a global database file that has a mapping from MAC addresses to IP addresses and use this to establish a segment with the newly discovered node. For the departure of a TAP, we use keep alive messages — the absence of which upto a time limit is used as an indication that the node has disappeared. Finally, we use an exponential weighted average of the RSSI values to smooth out fluctuations that may be present in the readings.

Other Implementation Strategies: Our prototype expects all devices (wired hosts, wireless hosts, and of course TAPs) to adopt to the Tapa requirements, thereby leveraging all the benefits offered by Swift. It is possible to build a version of Swift that does not require

changes on the wired host. One option is to have the mobile host delegate all responsibility for communications to the TAP, effectively allowing the TAP to masquerade as the mobile host. This would result in significant loss of functionality (e.g., no handoff during a session) and will also have many of the drawbacks associated with traditional hidden middleboxes (new failure models, e.g. failure of a TAP), but it may be useful for compatibility with legacy devices. Alternatively, the TAP could function as a traditional layer 2 device (e.g. WiFi access point) for sessions that involve legacy devices. Pretty much all Swift benefits would be lost, but it would result in more traditional failure semantics.

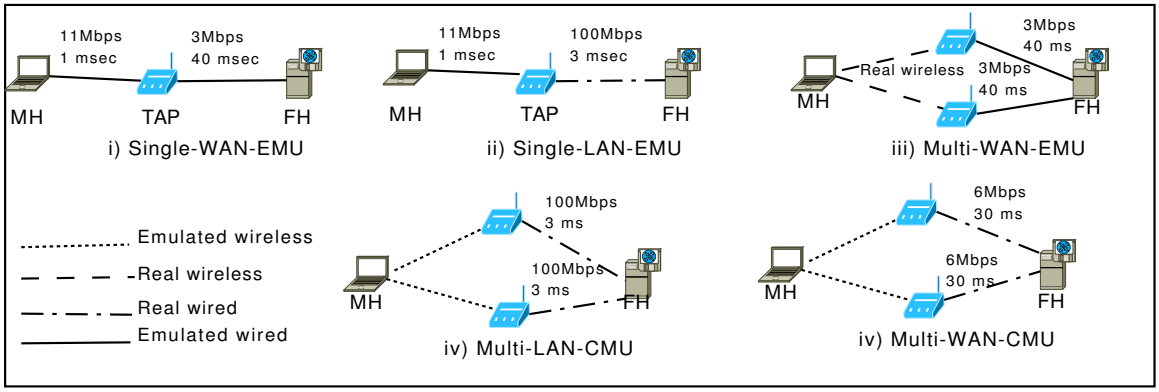


Figure 3.1: Various Topologies used in the Evaluation. Ones with Emu suffix correspond to Emulab testbed while those with CMU correspond to CMU wireless emulator. Wired links are emulated.

3.5 Evaluation

The goal of our evaluation is to show the benefits of Swift under diverse scenarios. Our evaluation can be divided into two parts. First, we show how Tapa can support diverse segment protocols and various mobility and wireless related optimizations. We demonstrate functionality as well as quantify the effort in adding that functionality (lines of code and man hours required). In the second part of the evaluation, we deal with micro-benchmarks of scenarios where it may hurt to use Tapa. Specifically, we evaluate the overhead of using Tapa and how it undertakes recovery in case of TAP failures.

Experimental Setup: We have used both a real world and an emulation-based testbed for our evaluation. (Figure 3.1 shows the various topologies used in the experiments). Emulab's Wireless Testbed [2] offers indoor desktop nodes equipped with WiFi support, thereby providing a realistic environment to conduct typical 802.11 experiments. For experiments, involving mobility, we used the CMU Wireless Emulator [1]. The emulator also offers real laptops equipped with WiFi and bluetooth support. However, the wireless signal passes

through an FPGA based emulator rather than going on the ether.

We used 802.11 b mode with default values. We used ad-hoc mode to avoid the switching overhead associated with managed mode. This is a practical alternative to the various solutions that allow fast switching between APs and recent studies have used a similar approach for such experiments [32]. Unless otherwise noted, we conduct five runs for each experiment and report the average (deviation is within 10% of the mean). We varied the ADU size from 64kB to 1MB.

3.5.1 Benefits for Wireless and Mobile Users

Supporting Diverse Segment Protocols

A key goal of our prototype was to enable decoupling of wireless and wired segments so that diverse segment layers can be used on the wireless side. We now discuss our experiences in adding different segment protocols to Tapa.

TCP: We added support for TCP as a segment on both the wired and wireless side for two reasons. First, in some network scenarios (e.g., wired), it provides many of the features that we expect from a segment protocol. Second, as its use is well understood, it allowed us a convenient way to implement a working segment protocol, thereby simplifying the implementation and debugging of the prototype.

HOP: HOP is a possible replacement for TCP in multi-hop mesh networks and environments involving mobility and disruption [81]. It runs between a client and a mesh gateway and expects some kind of decoupling at the gateway, so that a TCP-like protocol can work on the wired side for end-to-end transfers. We added a light weight stub (*50 LOC*) that removed the differences between the HOP API and the interface that Tapa expects segment layers to implement. Overall, it took roughly *20 man hours* to fully add support for HOP as a segment layer for Tapa. As HOP is designed for bulk data transfers, its performance in an ADU request/response settings is sub-optimal, so we added some custom support (like batching of packets at the TAP) to improve HOP's performance.

Blast : We have specifically designed a protocol for 802.11 based single-hop wireless networks where TCP features, like congestion control and ACK based reliability, are often an over-kill. Blast is built on top of UDP – it offers no congestion control, but provides light-weight reliability (in the forms of NACKs) and flow control. The protocol was developed independently and later integrated with Tapa as a segment protocol, requiring

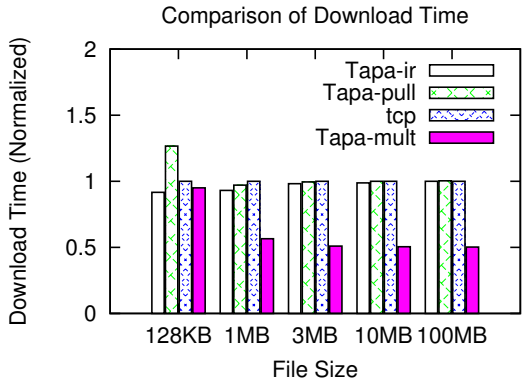


Figure 3.2: Aggregating uplink bandwidth of multiple TAPs.

approximately *10 man hours* for the integration effort.

Bluetooth: We also added support for Bluetooth RFCOMM transfer mode as a Tapa segment protocol. As this mode bypasses IP, it is an attractive option for small devices with limited capabilities who want to communicate over the Internet (through a TAP). The API exposed by the bluetooth library uses the socket API (unlike HOP). As a result it was straightforward to incorporate bluetooth as a segment protocol in Tapa, requiring approximately *5 man hours* and *20 LOC* for this task.

Performance: Table 3.1 shows the performance of these different segment layers (with tcp on the wired side) in an end-to-end transfer. TCP, HOP, and Blast used WiFi on the Emulab testbed while the bluetooth (BT) experiment was on the emulator. The results show expected performance under the given conditions.

	TCP	HOP	Blast	BT
Xput	5.95Mbps	6.4Mbps	6.6Mbps	600kbps

Table 3.1: Download of a 10MB file with different segment protocols. TCP is used on the wired segment. End-to-end TCP throughput i.e., without Tapa, is roughly 5.9Mbps.

Optimizations

Tapa was designed with the goal of simplifying the use of various optimizations that are applicable in the context of mobile wireless users. Our prototype supports several such optimizations. Implementing the optimizations was simple, with each optimization requiring at most *100 LOC*. In our experiments we focus on the optimization of using multiple segments, as this single optimization shows the underlying flexibility that Tapa offers. For

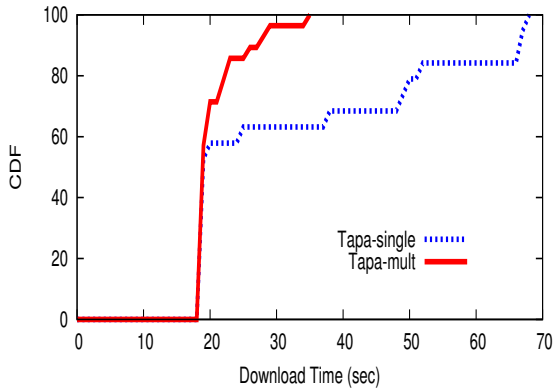


Figure 3.3: File Download Times in a Vehicular Communication Scenario. Use of multiple segments can mask disruptions that are common in such environments.

example, this optimization can be used with any arbitrary protocol (e.g., one segment uses HOP while the other uses TCP), any underlying technology (e.g., one bluetooth based segment and another 802.11 based segment), and to have segments with different ISPs/service providers. We have used this optimization in the following ways: to aggregate AP uplink bandwidth, for efficient hand-offs, to mask failures and for multiplexing multiple interfaces. We present results corresponding to two different scenarios: i) aggregating uplink bandwidth of multiple APs, ii) masking disruptions in a vehicular scenario.

Aggregating AP uplink bandwidth: This optimization is well understood but difficult to implement in today's Internet [67, 68]. In Tapa, supporting and benefiting from this optimization is quite natural, which we show through a simple experiment. For this set of experiment we use the `Multi-WAN-EMU` topology which uses two TAPs. We compare the normalized download time (with respect to `tcp`) for `Tapa-ir`, `Tapa-pull`, `tcp`, and `Tapa-mult`. `Tapa-ir` refers to the Tapa mode where the response is immediately sent as a *single* ADU; `Tapa-pull` refers to first getting the ids and then making request for those ids; and `Tapa-mult` represents the case where we make use of both TAPs. As Figure 3.2 shows, `Tapa-mult` provides benefits even for a small file size of 128kB which only contained two chunks of 64kB each. Previous approaches like FatVAP [68] aggregate bandwidth at a TCP stream level and therefore cannot exploit this opportunity at such a fine granularity.

Masking Disruptions in a Vehicular Scenario: We consider communication between a vehicle and multiple TAPs and how use of multiple segments can mask short disruptions that are difficult to handle if we use a single segment for end-to-end communication. We

use link level traces from a real world testbed at Microsoft's campus in Redmond [14]. We pick the two APs in the MS testbed with the best connectivity with the van and emulate their wireless channels using the emulator (`Multi-WAN-EMU` topology). The van downloads a 10MB file from a server located over the Internet; the RTT between the server and APs is 60ms. We compare the performance of `Tapa-mult` (i.e., Tapa with multiple segments) with `Tapa-single` (i.e., Tapa with a single segment). For each scenario, 20 requests are made and start times for the requests are chosen uniformly at random.

Figure 3.3 shows the cdf of the download times achieved in the two scenarios. Approximately 50% of the transfers do not experience any difference: these transfers are made during periods of good connectivity where there is no need to switch to the alternate segment. However, transfers that occur during bad periods experience severe performance degradation in case of `Tapa-single` whereas with `Tapa-mult` switching to the alternate segment (if it is available at that time) significantly mitigates the performance degradation.

3.5.2 Overheads: Micro-benchmarks

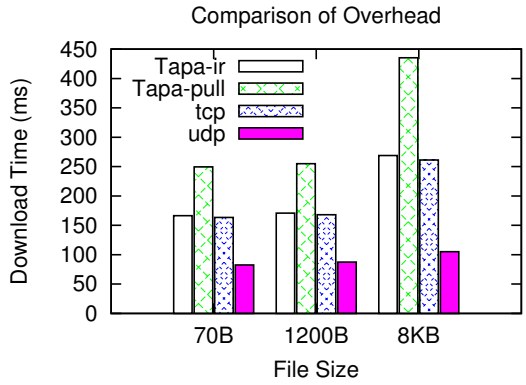
Tapa offers several optimizations but not every application can benefit from them, so it is important to consider scenarios where it may hurt to use Tapa. We therefore conduct micro evaluation to evaluate the overhead of using Tapa under a scenario where no optimization is used.

Performance Overhead

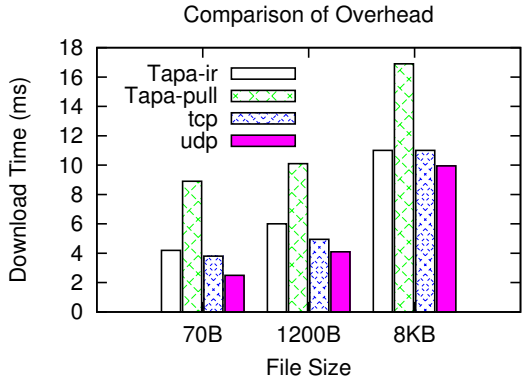
We consider the `Single-WAN-EMU` and `Single-LAN-EMU` topologies for this set of experiments. We use TCP on both the wireless and wired segments. Also, even though Tapa allows reuse of segments which can eliminate connection set up delay, but we disable this option for these experiments. This ensures a fair comparison with standard end-to-end tcp.

We compare the time required to complete a short request-response exchange in four different scenarios: `Tapa-ir` which refers to the mode where we push ADUs, `Tapa-pull` where we pull an ADU by first retrieving its id and then retrieving the data, `tcp` and `udp`. Figure 3.4(a) shows the results in a WAN setting with an RTT of `80 ms`. `tcp` and `Tapa-ir` perform the same whereas `Tapa-pull` requires more time because of the extra RTT involved in making a request for individual ADUs. Figure 3.4(b) shows that even in a LAN setting Tapa does not introduce any noticeable overhead and performs similar to the underlying segment protocols that it uses (i.e., TCP).

The above results for messages as small as 70 bytes show that the extra overhead introduced by Tapa in the form of ADUs and TAP is negligible. It also shows that `Tapa-ir` is a



(a) WAN



(b) LAN

Figure 3.4:]
Measuring Tapa's Overhead: Tapa adds negligible overhead in both LAN and WAN scenarios. For transfers larger than 1MB (not shown) all scenarios have similar performance.

useful mode that can be used by interactive applications with short messages. This analysis suggests that over stable wireless links, Tapa will perform as well as today's protocol stack for a wide range of applications.

Reliability

As Tapa has to deal with ADU recovery in case of *TAP failures*, we want to measure the overhead of this process. We use the LAN topology with tcp on both segments, and consider a scenario where the TAP fails after 15 sec and loses all its state. It is up again after 5 sec and can start serving the clients again. This pattern is repeated and clients continue to make requests at uniformly random times for a 10MB file. We compare end-to-end tcp

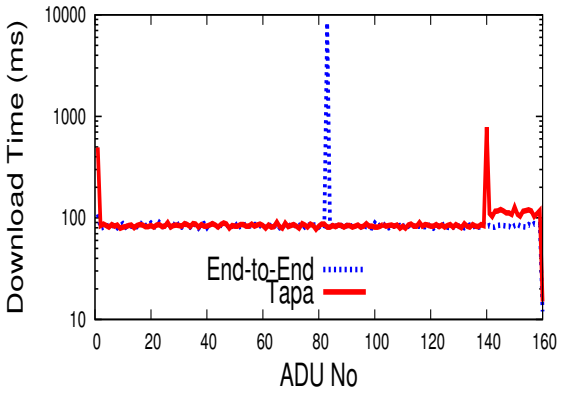


Figure 3.5: Case of TAP failure. Tapa recovers efficiently from TAP failures. Note log scale.

which naturally recovers from such AP failures, with Tapa, where a TAP failure requires recovering the ADU that was being transmitted and all the ADUs that were yet to be transferred.

Figure 3.5 shows the download time for each of the 160 ADUs present in the file (note log scale). The spikes show the increased time that is required to download the affected ADU – during whose transfer the TAP failed. Note that as requests are made at random times we pick two typical runs (one each for Tapa and end-to-end tcp) that depicts how the recovery process works in both scenarios. In Tapa, the client discovers that the TAP is down (using absence of hello messages) and establishes another segment with an alternate TAP or waits for the old TAP to appear again. As the transfer service knows the ids of the missing ADU, it sends a new request to the server using the new TAP. As the graph shows, Tapa can recover efficiently from TAP failures.

3.6 Related Work

Problems caused by wireless and mobility are well known and there is a large body of work that addresses these concerns. Most of these proposals center around making things work with existing protocols and without making any modifications to the applications. Our approach is clean slate, which provides a cleaner solution, although its deployment may be more difficult compared to many prior proposals. We discuss some representative proposals that address various problems in this domain and compare how Swift addresses similar problems in a much simpler and elegant manner.

Wireless and the Internet A common theme in supporting wireless users with poor links or limited capability is to add customized support at the wireless middlebox (I-TCP [29], PEP [39] etc). However, as discussed earlier, due to the end-to-end nature of today's Internet, these proposals break end-to-end semantics, creating new problems due to "hard state" at the middlebox (e.g., new failure modes, mobility across different networks, etc.)

Other research has looked at utilizing multiple APs (e.g., FatVAP [68]) or multiple interfaces [114]. Not only are these optimizations hard to implement, but their efficacy is limited by the rigidity of the Internet architecture, so they tend to be specific to a transport protocol or the underlying link layer technology. For example, FatVAP is specific to 802.11, can only deal with aggregation of bandwidth on an inter-TCP stream level, and does not consider application level requirements. As a result, we often need to re-implement these optimizations if we change the access scenario, protocol or application requirements. As we show in Swift, the architectural support in Tapa simplifies the deployment of these optimizations, so they can be used with any application or segment protocols.

Mobility and the Internet: Several architectural proposals address the issue of mobility at the network layer. This includes proposals like Mobile IP and i3 [110], which are network layer proposals and still require suitable support on top of them for reliable, end-to-end data delivery. For example, *i3* decouples the act of sending and receiving a *packet*, providing network level support for mobility [110]. Therefore, it expects a single end-to-end transport connection on top of the i3 infrastructure. Tapa focuses on decoupling an end-to-end *flow*, using customized protocols on each segment. More recently, the MobilityFirst project [6] leverages several concepts (e.g., in-network storage, blocks/ADUs, segments, etc) of Tapa/Swift to support mobility as a norm rather than an exception. Tapa can be used on top of this architecture: pushing some of the mobility related optimizations inside the architecture, instead of having them at the transfer layer, can improve performance and simplify the role of Tapa's transfer layer.

TCP-Migrate [107] allows a TCP connection to remain active even if the mobile host moves across different subnets. Their approach is end-to-end and does not require any support from the network. In contrast, Tapa relies on caching within the network (i.e, at TAPs) to reduce performance disruptions caused by mobility. We believe that as mobility becomes a norm rather than an exception, support from the network will become absolutely necessary. Haggle addresses the problem of content discovery in a mobile environment [105] advocating an opportunistic approach to content retrieval. This is similar to Tapa's approach of empowering the mobile host to make data transfer decisions based on

the availability of resources.

3.7 Summary

In this chapter, we presented Swift, a prototype of Tapa, which is specifically designed for wireless and mobile users. We showed how Swift addresses the key challenges faced in a mobile, wireless environment, in addition to leveraging the opportunities that are available in these settings. Through a detailed design, implementation and evaluation, we showed that Swift can provide benefits for wireless and mobile users at three broad levels: i) it can accommodate diverse wireless protocols as segment protocols within an end-to-end path, ii) it can easily use optimizations such as multiplexing multiple segments to improve throughput and mask disruptions, and iii) it can facilitate data transfers in mobile scenarios by making use of caching at TAPs.

Chapter 4

Traffic Shaping Service for Energy Savings

In this chapter, we show how key concepts of this thesis can be applied to improve the energy efficiency of data transfers. We present an in-network service, Catnap, which can provide up to 2-5x improvement in battery life of mobile devices. Catnap builds on top of two key concepts that are central to Tapa: i) decoupling of wired and wireless segments, and ii) making the network aware of ADUs. In this chapter, we provide the motivation behind Catnap, its detailed design, implementation, and evaluation, and case studies on how existing applications, like web and email, can use and benefit from Catnap.

4.1 Overview

Energy efficiency has always been a critical goal for mobile devices as improved battery lifetime can enhance user experience and productivity [86, 89]. A well-known strategy for saving power is to sleep during idle times. Prior work in this context exploits idleness at various levels, such as sleeping during user think time [40] or when TCP is in slow start [75]. Recent work also explores the possibility of entering low power consuming states in the middle of data transfers [82]. However, deeper sleep modes, such as 802.11 PSM or Suspend-to-RAM (S3), are assumed to be unusable *during* data transfers when applications are constantly sending data, as entering into these sleep states degrades application performance due to the overhead of using these sleep modes [25, 75].

In this chapter, we present Catnap, an in-network service that saves energy by combining tiny gaps between packets into meaningful sleep intervals, allowing mobile clients to sleep *during* data transfers. These tiny gaps arise when a high bandwidth access link is

App. Type	Examples	Catnap Strategy (Target Device)	Expected Benefits	Remarks
Interactive	VoIP	None	None	Device and NIC remain up to maintain user experience (UE)
Short Web Transfers (<100kB)	`facebook` `google`	Batch Mode (Small Devices: Smart Phones, Tablets, etc.)	30% NIC sleep time (Section 4.6)	- Embedded objects rendered together - Batching may require app. specific support at the TAP (e.g., java scripts)
Medium Sized Transfers (128kB to 5MB)	`you-tube` `flickr` images mp3 songs	Normal Mode (Small Devices)	up to 70% NIC sleep time (Section 4.5.4)	No impact on UE
Large Transfers (>5MB)	maps & movies software updates ISR [74]	1. Normal Mode (Small Devices) 2. S3 Mode (All Devices)	1. >70% NIC sleep time 2. 40% device sleep time for a 10MB transfer (Section 4.5.4)	1. No impact on UE 2. User permission to enter S3 mode is desirable (Section 4.7)

Table 4.1: Energy Savings with Catnap for different types of applications.

bottlenecked by some slow link on the path to the Internet. A typical example is a home wireless network where 802.11 offers high data rates (54Mbps for 802.11g and 600Mbps for 802.11n) but the uplink to the Internet (Cable, DSL) offers much lower bandwidth, typically in the range of 1-6 Mbps [51]. Catnap is targeted towards data oriented applications (e.g., web, ftp, etc) that consume data in *blocks* i.e., their *Application Data Units* (ADU) [45] size is reaonsably large. For such applications, combining small idle periods and hence delaying individual packets is acceptable if the ADU is delivered on time.

Catnap builds on three concepts. First, it *decouples* the wired segment from the wireless segment, thereby allowing the wireless segment to remain inactive even when the wired segment is actively transferring data [29, 58]. The TAP batches packets when the mobile device is sleeping and sends them in a burst when the device wakes up. Second, it uses an ADU (e.g., web object, telnet command, P2P chunk, etc.) as the unit of transfer. This allows the TAP to remain *application independent* and yet be aware of *application requirements*, thereby ensuring that delay sensitive packets (e.g., telnet) are differentiated from delay tolerant packets (e.g., web, ftp). Third, the TAP uses bandwidth estimation techniques [68] to calculate the available bandwidth on the wired and wireless links. The bandwidth estimates determine the amount of batching required for on-time ADU delivery. Recall that the first two concepts (decoupling of segments and ADUs) are naturally supported in Tapa while the third concept, bandwidth estimation, can be viewed as one of the tasks that TAPs need to undertake as part of resource management.

We have implemented a prototype of Catnap as a transfer service that runs on the TAPs.

The service uses OS buffers to decouple wired and wireless segments, but also supplements these buffers with extra storage capacity to ensure that data continues to flow on the wired segment even if the client is in sleep mode for arbitrary long intervals of time. Catnap further implements a novel *scheduler* that schedules ADU transmissions on the wireless segment, ensuring maximum sleep time for the clients with little or no impact on the end-to-end transfer time. The scheduler dynamically *reschedules* transfers if conditions change on the wired or wireless segments, or new requests arrive at the TAP. The scheduler can further operate in two modes: i) one where no increase in end-to-end transfer completion time is allowed, and ii) a second one, that we call *batch* mode. In this latter case, multiple small objects are batched together *at the TAP* to form one large object, allowing additional energy savings at the cost of increased delay for individual objects.

4.1.1 Energy Savings with Catnap

Table 4.1 gives an overview of Catnap's benefits for various applications, also highlighting some of the possible issues with using Catnap. As we can see from the table, for transfer sizes larger than 128kB, Catnap can put the NIC to sleep for almost 70% of the total transfer time without any impact on user experience. For transfers larger than 10MB, S3 mode provides significant system wide energy savings. However, its use is limited to scenarios where the user is not involved in other tasks. Our evaluation further demonstrates the ability of Catnap to benefit existing application protocols, such as HTTP and IMAP, without requiring modifications to servers or clients. Performance and energy efficiency are sometimes at odds in these application scenarios that often involve transfer of very small blocks of data (e.g., short web transfers). We demonstrate using the Catnap *batch* mode that one can manage such a trade-off for very small transfers by batching multiple objects at the TAP.

4.1.2 Contributions

The key **contributions** of this chapter are:

- We design and implement Catnap which exploits the bandwidth discrepancy between wireless and wired links to improve energy consumption of mobile devices. The key design components of Catnap include decoupling of wired and wireless segments and a scheduler that operates on ADUs and attempts to maximize client sleep time without increasing overall ADU transfer times.

- We conduct a detailed evaluation of Catnap using real hardware, under realistic network conditions. We show how Catnap can improve battery life of tablet PCs and laptops by enabling various sleep modes. We also consider case studies of how existing application protocols, like HTTP and IMAP, can use and benefit from Catnap.

Our results clearly attest to the promise of Catnap to increase mobile client battery lifetime at no to little performance impact.

4.1.3 Chapter Organization

The rest of the chapter is organized as follows. In §4.2, we motivate the need for Catnap. We present the design and implementation of Catnap in §4.3 and §4.4. §4.5 presents the detailed evaluation of Catnap. In Section 4.6, we present case studies that show how two legacy application protocols (IMAP and HTTP) can use and benefit from Catnap. We discuss the usability aspects of Catnap in §4.7. Finally, we discuss work that is most relevant to Catnap.

4.2 Motivation

We first motivate the need for a system wide view of energy management with a focus on why existing sleep modes are often not very useful in practice. We then discuss an opportunity for energy savings and how we can exploit it to enable various sleep modes without degrading application performance.

4.2.1 Sleep Modes – Potential Benefits and Limitations

Many devices today offer a range of sleep modes that allow sub-components of the system to be placed in various sleep states. The benefits of putting specific sub-systems into sleep mode depend on the contribution of each sub-system to the overall energy consumption. For example, 802.11 consumes less than 15% of the total power on a laptop but can consume more than 60% of the total power on a small hand-held device [89]. As a result, the benefits of 802.11 power saving mode (PSM) depend heavily on the device in use. On the other extreme are deep sleep modes like S3 (Suspend-to-RAM) that benefit both small as well as large devices, since they shut-off most of the device circuitry, thereby hardly consuming any power at all. There are also other sleep modes that are becominge increasingly more popular, such as a "reading mode" that keeps the display on but puts most of the other sub-systems into sleep mode, or a "music mode" that can be used if the user is only listening

Suspend-to-RAM (S3)	**802.11 PSM**
10 sec	Beacon Interval ($\approx$ 100ms)

Table 4.2: Overhead of using different sleep modes.

	ping	**ssh**	**web-transfer**	**ftp**
Total Time (sec)	10	20	5	100
Sleep Time (sec)	7	12	0	0

Table 4.3: Transfer and NIC sleep times for different applications in a typical home wireless scenario. Applications that constantly send data (web and file transfer) do not allow any NIC sleep time.

to music and not doing any other task. As network activity is often an important part of the user's interaction with the device, managing it in an intelligent manner can be critical for increasing the battery life of mobile devices.

Limitations of Power Saving Modes: In today's Internet, it is difficult to utilize the available sleep modes *during* data transfers because of their associated overheads. Table 4.2 lists the overhead of entering and leaving the sleep state when using Suspend-to-RAM(S3) and 802.11 PSM modes. For the S3 mode, the associated overhead approaches 10 seconds [20], and is clearly unsuitable for TCP based applications. Similarly, for 802.11 PSM, the NIC wakes-up at the granularity of beacon intervals (100ms); as packets can be delayed by this interval, performance of applications like telnet, VoIP, distributed file access, or short web transfers can be severely degraded if the NIC goes into sleep mode when these applications are in use [25, 75]. Therefore, there is general consensus that the NIC *should not* enter PSM mode as soon as it encounters an idle period – rather it should only enter sleep mode when this will not impact application performance [25, 75, 82].

Making the PSM implementation aware of application latency requirements is problematic for various reasons. First, it hinders application innovation as policies for new applications have to be incorporated in the PSM. Second, a single application, like web, could support streaming, instant messaging, and file transfers, that have very different interactivity requirements. Therefore, most 802.11 cards use a conservative policy of entering PSM only when there is no network activity for a certain time (generally on the order of 100ms). This ensures that cards do not sleep in the middle of an application session, even for applications that do not send packets continuously (e.g., VoIP).

We conducted an experiment to verify this for a 802.11g based network with a cable modem providing the uplink connection. As shown in Table 4.3, applications like `ping` and `ssh` that have idle periods, either because of inherent `1sec` gaps between successive packets or due to user think time, can indeed allow the NIC to sleep throughout their execution.

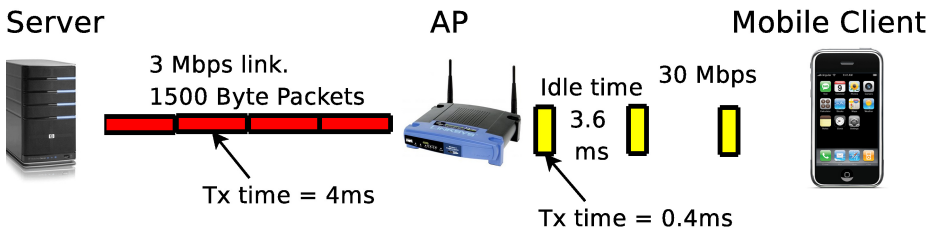


Figure 4.1: Data transfer in a typical home wireless scenario. Idle periods on the wireless segment are individually too short and remain un-utilized.

On the other hand, data intensive applications, like file or web transfer, tend to send data as fast as possible and thus will hardly lead to any significant idle periods that could allow the NIC to enter sleep mode. As these data intensive applications become increasingly more popular, we need to find avenues where we can save power even while using these applications.

4.2.2 Opportunity – High Bandwidth Wireless Interfaces

Figure 4.1 depicts one such opportunity that can allow energy savings *during* the use of data oriented applications. It shows a typical 802.11g based home wireless network that connects to the Internet through a DSL connection. The server is constantly sending packets to the access point (AP) but packets sent on the wireless channel are roughly evenly spaced by the packet transmission time on the bottleneck link (DSL). For the example scenario, this idle time is less than 4 millisecond. As discussed and shown earlier, 802.11 PSM does not exploit these small idle opportunities and the NIC remains up for the whole transfer duration. Of course, using S3 mode in this scenario is impossible as TCP will likely time-out during the time the client is in S3 state.

Discussion: It is difficult to exploit the above opportunity in today's Internet. The key reasons are as follows:

- It is difficult for the network/middlebox to differentiate between packets that are needed immediately by the client from packets that can be potentially delayed. For example, a telnet packet should be delivered immediately while a packet that is part of an HTTP response message can be delayed until the whole message is received. This implies that applications should be able to provide a *workload hint* to the network/TAP indicating when the data is consumed by the application.

- The current TCP based communication model assumes strict synchronization between end-hosts. Not only the two end-points need to be present at all times but they also

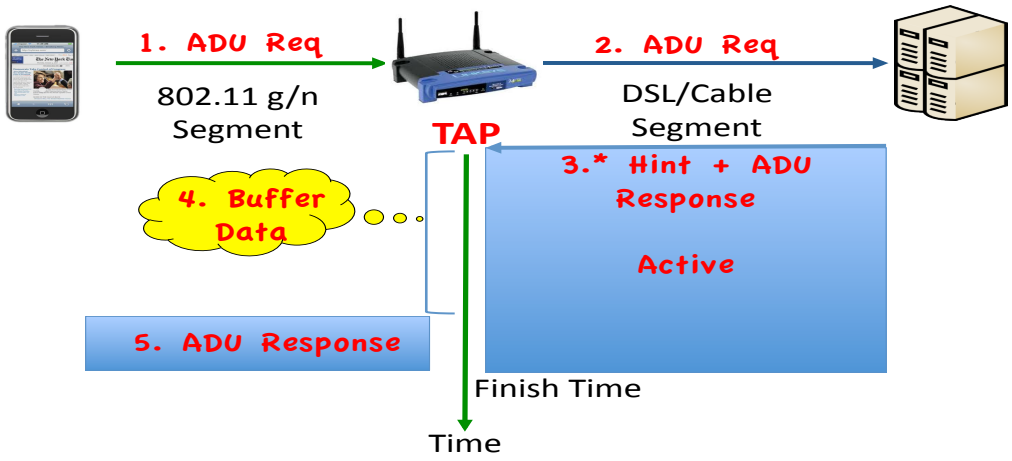


Figure 4.2: Basic overview of Catnap.

need to send regular messages (e.g., ACKs) in order to sustain communication. This limits the opportunities to sleep for a mobile client, as going into sleep mode could result in degradation in performance or even loss of connection. The system should be able to support *relaxed syncronization* between end-points, allowing TAPs to *shape traffic* in a way that allows maximum sleep time for the mobile device.

4.3 Catnap: Design

Catnap addresses the above two problems, thereby exploiting the opportunity created by bandwidth discrepancy between the wired and wireless segments. We first provide a high level overview of Catnap followed by details on its main components.

4.3.1 Overview

Figure 4.2 provides the high level overview of Catnap by showing the key steps involved in a typical data transfer. The client makes a request for an ADU to the Catnap service which forwards the request to the destination server over a separate segment (TAP - server). The server sends the response to the Catnap service which schedules the best time to initiate the wireless transmission to the client, so as to minimize the time the client is awake.

The timing of the transmission is based on what we term as *workload hints*. *Workload hints* expose information on the ADU boundaries in the provided server response and the willingness of the client to suffer a slight increase in overall transfer completion time for additional energy savings. The former can be provided by the server itself or through an application specific proxy on the TAP. We go through the *workload hints* in more detail later in this section.

The scheduler uses the *workload hints* along with information about the wired and wireless bandwidth to calculate the total transfer time and the length of the time slot needed to transmit the block of data on the wireless side. In the general case, the wireless time slot is scheduled as late as possible while ensuring that transfer times do not increase. Of course, the network conditions can change or new requests overlapping with the previous time slot can arrive, so the scheduler may need to update its scheduling decision as new information becomes available. The Catnap scheduler can further operate in what we call the *batch* mode. If the client is tolerant to a slight increase in the transfer completion time, then the scheduler can concatenate multiple small slots into bigger ones, thus allowing for greater energy savings. The *workload hint* will convey the willingness of the client to employ the *batch* mode.

Finally, while we focus on Catnap's benefits in download scenarios, the case of uploading data from the mobile device to the Internet is even simpler and does not require the full Catnap functionality. In an upload scenario, the device bursts the data to the TAP who buffers it and sends it at the bottleneck rate to the destination. The Catnap scheduler does not need to be involved in this case.

4.3.2 Example Scenarios

We discuss two example scenarios in which Catnap can provide energy savings: i) small transfers where the NIC can be put into sleep mode, ii) large downloads where the whole device can be put into sleep using the S3 mode. In both examples, we do back-of-the-envelop calculations for the potential energy savings under the network configuration shown in Figure 4.1. We only consider the transmission delay for our calculations and ignore the impact of RTT or protocol behavior in our analysis.

NIC energy savings for small transfers (< 1MB): Examples of such small transfers include many web-pages, high resolution images, and initial buffering requirements of video streaming (e.g., `you-tube`). For a 128kB transfer, the total time required to complete the transfer is 350ms while it takes only 35ms to complete the transfer on the wireless link. This means that the remaining time (315ms) can be used by the mobile device to sleep. As

we show in Section 4.5, this time is long enough for the wireless NIC to enter sleep mode and then return without any change to the PSM implementation. As a result, the user does not experience any change in response times.

S3 sleep mode for large transfers (> 5 MB): Examples of large transfers include: downloading a song or a video, downloading high resolution maps, file synchronization between devices, or downloading a virtual machine image to resume a suspended session (e.g., ISR [74]). If we assume a 10MB transfer, the total transfer time is 27 seconds. The wireless transmission takes only 2.7 seconds. The mobile device can thus sleep for the remaining time (24.3 seconds). A duration of tens of seconds is long enough to enter and exit S3 mode. We show in Section 4.5 how S3 mode can provide significant energy savings for large file transfers.

The important thing to note here is that we can only use S3 mode if the user is not doing any other task while the download is taking place. For example, the user starts the download and leaves the device to grab coffee, or the user stops at a gas station and starts downloading mp3 songs while she refuels her car. In such cases, as the user is not actively using the device, the system can easily use the S3 mode. In some scenarios, the user may be so short of battery that she is willing to stop other tasks in order to get energy savings from the S3 mode. We discuss the usability aspects of the S3 mode later in Section 4.7.

4.3.3 Decoupling

Catnap relies on the *decoupling* between the wired and wireless segments provided by Tapa. For complete decoupling, we use in-network persistent storage in addition to OS buffers. This is important because in some cases we want the Catnap service to operate in a cut-through way i.e., to forward data as soon as it receives it, which does not require significant buffering. However, in some cases the TAP may need to buffer a large amount of data before sending it to the client. This case is similar to the DTN-style store and forward approach that requires extra storage to supplement the OS buffers.

Deployment Scenarios: As data oriented applications get more popular, the storage requirements of a Catnap service will increase. Many modern APs already support the addition of USB sticks that can provide this extra storage for Catnap [5]. In certain deployments (e.g., enterprise settings) that use thin APs and a centralized controller, we expect the Catnap functionality and the storage to be co-located with the centralized controller. This provides an easy way to deploy Catnap without modifying all the APs in an enterprise. The main point is that the Catnap service should be location at a point that can decouple the end-to-end path into two segments: a bottleneck segment on the one side and a high

bandwidth segment on the other side. So the Catnap service need not always be physically co-located with the wireless AP. This implementation strategy also naturally handles mobility *within* a subnet, as data is buffered at the central controller rather than at individual APs. However, the scheduling decision may need to be updated after a hand-off, as the wireless bandwidth available to the new AP may significantly differ from the bandwidth available to the old AP.

4.3.4 Workload Hints

Catnap relies on the concept of ADU to identify when a certain data block will be used by an application [45]. The concept of ADU is well known in the networking community as it is a natural unit of data transfer for applications. Different applications use different ADUs – for example, P2P applications can define a *chunk* as an ADU because the download and upload decisions in P2P applications are made at the granularity of a chunk [48]. Similarly, a file transfer application can define the whole file as an ADU because a partial file is generally useless for the users. In some contexts, defining an ADU may vary depending on the scenario. For example, a web server can define all objects embedded within a web-page as one ADU if all the objects are locally stored or as separate ADUs if these objects need to be fetched from different servers. ADUs also have other advantages – for example, they are a natural data unit for caching at the middlebox

While applications naturally operate on ADUs, data transfer is based on byte streams (TCP based sockets) rather than ADUs. In order for Catnap to estimate the oncoming workload, we need to identify the boundary between ADUs. Such information can be provided by the application or through an application specific proxy on the TAP. Note that most applications have an implicit notion of ADUs embedded in their byte stream. For example, as we discuss in Section 4.6, both HTTP and IMAP implicitly define their responses as ADUs – they also have headers that include the length of the ADUs in them. Such information could be easily extracted by an application specific proxy running on the TAP.

The *workload hints* are included in the ADU header and comprises of three fields: `(ID, Length, Mode)`. The `Length` field is the most important as it allows the Catnap service to identify ADU boundaries, enabling the service to know when the ADU will be "consumed" by the application. The `Mode` field indicates the client's willingness to use *batch* mode. In our current design, knowledge on `Mode` is sent from the client to the TAP and played back by the server. The default mode is not to use batching. Finally, the `ID` field is optional and is used to identify and cache an ADU.

4.3.5 Scheduler

The scheduler is the heart of Catnap as it determines how much sleep time is possible for a given transfer. The goal is to maximize the sleep time without increasing the total transfer time of requests. This requires *precision* in scheduling and the ability to dynamically *reschedule* requests if the conditions change. The rescheduling of requests also ensures fairness amongst requests as old requests can be multiplexed with a new request that arrives later in future.[1]

As we do not want to increase the transfer time, we rule out the simple store and forward technique where the TAP receives the whole ADU and *then* transfers it to the mobile client. As this strategy is agnostic of network conditions, it could add significant delay if the wireless medium is temporarily congested. In some cases, however, the client may be willing to tolerate additional delay if this improves the battery life. The user would, however, want control over this option, so it can be used in a judicious manner. Based on these requirements, the scheduler provides two modes: in the *normal* mode it allows maximum sleep time without increasing the transfer time, while in the *batch* mode it provides savings on top of the normal mode by batching (and thus delaying) data. We will first focus on the normal mode and then discuss how we provide additional support in the form of batching.

How much can we sleep? There are *four* variables that determine the amount of sleep time that is possible for a given transfer: i) the size of the transfer, ii) the cost (in terms of time) of using a sleep mode, iii) the available bandwidth on the wired segment, and iv) the available bandwidth on the wireless segment.

The first two variables are easier to determine. The hints provide the Catnap service with the size of the transfer. Similarly, the cost of entering a sleep mode is generally fixed for a certain sleep mode and device. The last two factors i.e., available bandwidth on the wired and wireless segments, are more difficult to determine. Given these four variables, the sleep time for a given transfer is given by:

$$T_{Sleep} = \frac{Size_{ADU}}{BW_{Wired}} - (\frac{Size_{ADU}}{BW_{Wireless}} + Cost_{SleepMode}) \tag{4.1}$$

The first term is the time required to complete the transfer on the wired segment. If the wired segment is the bottleneck then this time is also the total transfer time. We can only sleep if this time is greater than the sum of the transfer time on the wireless segment and

[1]For fairness amongst different stations (different TAPs or legacy 802.11 stations), we rely on the fairness provided by 802.11.

the cost of using the sleep mode. The equation provides some intuition for Catnap. For example, sleep mode is only possible when the wireless bandwidth is *more* than the wired bandwidth. Similarly, because of the fixed cost of using a sleep mode we need a minimum size for the ADU to get benefits from sleeping. As we increase the ADU size this cost is amortized and we can get more time to sleep. The last observation provides the motivation behind Catnap's batch mode.

Challenges: Leveraging the above sleep time benefits involves several challenges. First, calculating the available bandwidth is difficult, especially for the wireless medium, which is a shared resource. Conditions on the server side can also change due to extra load that may impact the finish time of transfers. A second, but related, challenge is that transfer requests come in an *online* manner – the TAP does not have a-priori information about future requests. So Catnap's policy of waiting and batching data may prove sub-optimal as new requests may arrive during the batching period. In summary, the Catnap scheduler has to account for unpredictability in network conditions as well as the online nature of the requests arriving at the TAP.

1. Estimate **Capacity** of Wired Link
2. Estimate **Available BW** for this transfer
3. Calculate **finish time (FT)** of transfer
4. Estimate **Wireless Capacity**
5. Calculate **Virtual Slot Time (VST)**
6. Schedule Transfer at (FT − VST)

Shift or **Merge** Slots, if required

7. Periodically check for **Rescheduling**

Figure 4.3: Basic steps of the scheduling algorithm.

Algorithm

Figure 4.3 lists the basic steps of Catnap's scheduling algorithm. It accounts for conditions on both the wired and wireless medium in deciding when to schedule a transfer. Furthermore, it reschedules requests based on changes in network conditions (both wired and wireless) or the arrival of new requests.

The two key features of the scheduling algorithm are: i) it mostly relies on information that is *already present* with the Catnap service to estimate the finish time of the transfer, as well as the slot time on the wireless segment and ii) it avoids *multiplexing* time slots for different clients on the wireless side, as this results in extending the slot time for all the clients that are multiplexed.[2]

[2]As an example, consider two clients with a transfer duration of 3ms each. Multiplexing them together

We now describe the scheduling steps in detail. Recall that the scheduling decision is made when the Catnap service receives the response that has the workload hint about the length of the transfer. With this information, the scheduler performs the following steps:

Step 1 – Estimate wired capacity. On the wired side, the bottleneck could be the access link, a core Internet link, or the server. Earlier studies have shown that broadband access links are often the bottleneck in the end-to-end path [51]. Therefore, as our main target is home wireless scenarios, it is reasonable to start-off with the assumption that the access link is the bottleneck in the wired path. We can use a fixed value if the access capacity remains constant (e.g., DSL) or we can periodically use an existing bandwidth estimation tool to estimate the capacity of the access link [66]. To account for the case that the bottleneck is elsewhere, e.g., on the server, the Catnap service continuously monitors the throughput that is actually achieved on the server-TAP path.[3] If the observed throughput differs from the initial bandwidth estimate, the transfer is re-scheduled (Step 7) using the observed throughput as the new estimate.

Step 2 – Estimate Available BW for the transfer. When estimating the available bandwidth for a transfer, we need to account for the fact that multiple transfers may be using the access link at the same time. This is necessary when we use the access link bandwidth as an initial estimate of available bandwidth. To this end, the Catnap service identifies all active transfers and splits the access bandwidth among these transfers in different proportions. As we use TCP based segment, we divide the bandwidth in an RTT-proportional way amongst the ongoing transfers. Note that bandwidth estimates based on observed throughput automatically account for the sharing of the access link, so if this estimate is incorrect, it will be updated based on actual observed performance (Step 7).

Step 3 – Calculate transfer finish time. Recall that the scheduler already knows the length of the transfer (`Length`). Step 2 also provides information about the available bandwidth for the transfer on the wired side. So combining these two pieces of information allows the scheduler to determine the finish time (FT) of the transfer. This time is important since it constitutes the scheduler's deadline for the client transfer.

Step 4 – Estimate wireless capacity. Based on steps 1-3 we know how long the transfer will take on the wired server-TAP segment. We also need a similar estimate for the wireless segment and estimating the available wireless capacity is a first step in this direction. Note that the conditions may change, but we use the *current* available capacity as a guide to what we will likely experience in the future. As wireless is a shared resource and

means that both of them are up for approximately 6ms whereas without multiplexing they only need to be up for 3ms.

[3] We ignore this step for short transfers due to the effects of TCP slow start.

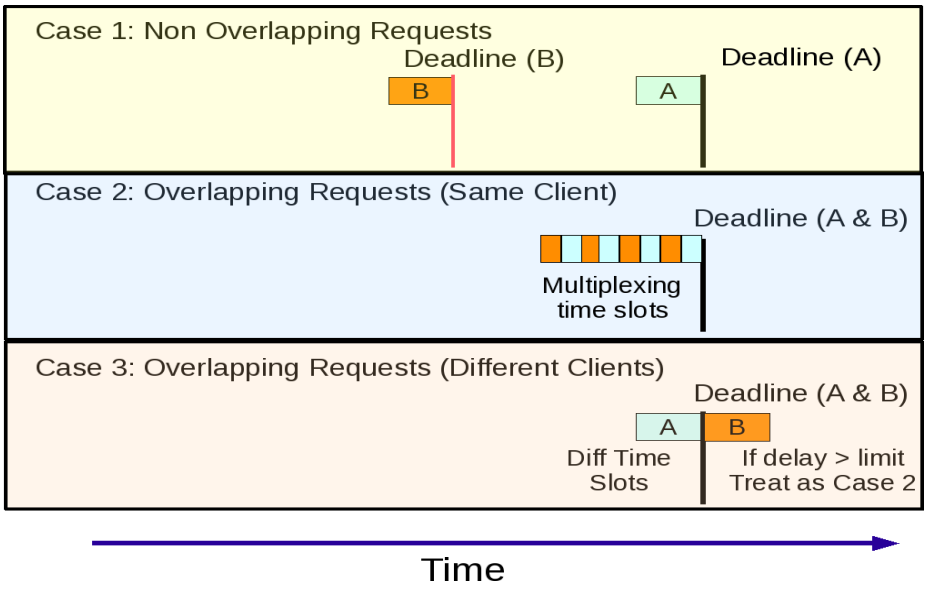


Figure 4.4: Different cases of virtual time slot scheduling.

the TAP can potentially overhear other transmissions in its neighbourhood, our prototype uses the medium busy/idle time as the basis for wireless capacity estimation. We provide more details on our specific implementation in Section 4.4.2.

Step 5 – Calculate Virtual Slot time. Estimating the wireless capacity allows the TAP to know how much channel access it will get on the wireless medium (Step 4). We then combine this information with the bit-rate used between the client and the TAP to estimate the expected throughput between the TAP and the client. Finally, based on the expected throughput and the size of the transfer, we know the time required to burst the whole transfer on the wireless segment. We call this the virtual slot time (VST). Note that we also add a small padding time to each virtual time slot as a safety measure to account for unexpected changes to the network conditions.

Step 6 – Schedule the Virtual Slot. Ideally, we want to schedule the virtual time slot as late as possible so that its end time aligns exactly with the finish time of the request. If there is no other transfer scheduled at that time then we can simply schedule the virtual time slot at this position (Figure 4.4: Case 1).

However, there will be cases when multiple transfers contend for wireless transmission time, e.g., a new transfer overlaps with a previously scheduled transfer. In such cases, if the new and old slots belong to the *same* client then we simply multiplex them together and make a bigger time slot (Case 2). The case where the slots partially overlap is also handled naturally in this case.

In contrast, if the two slots belong to different clients, then multiplexing them together is not the best strategy as it increases the up time for *both* clients. So if the time slots correspond to different clients then we should schedule them at non-overlapping times. However, non-overlapping time slots mean that at least one of the time slots is scheduled *after* its original finish time (Case 3). We can pick any order for scheduling the requests (A-B or B-A), if the slots perfectly overlap. If the slots partially overlap then we first schedule the transmission that has an earlier deadline.

We adopt this strategy of non-overlapping time slots if the increase in delay is less than a certain threshold, otherwise we multiplex the two time slots. This threshold can be tuned to achieve greater energy savings at the cost of some increase in delay.[4] For example, a very small value of the threshold can be used if the user is not willing to tolerate any additional delay and a higher value of the threshold may be fine if the user is running short of battery. We discuss possible ways to capture such user preferences in Section 4.7.

A final point is that multiplexing two slots increases the slot length and the new, bigger time slot may start overlapping with some other slot, so this decision needs to be made recursively. In practice, implementations can put a limit to the number of slots that can be *disturbed*. If this limit is exceeded, then a slot is scheduled even if the scheduled completion time exceeds the finish time of the request.

Step 7 – Reschedule. As network conditions can change or our initial estimates can prove wrong, we need to periodically re-evaluate our scheduling decisions. That's what we aim to achieve through the monitoring of real time system performance. We monitor conditions on both the wired and wireless segments and readjust our scheduling decision if the available bandwidth on either of these segments change. §4.4.2 provides details on our specific implementation for monitoring wired and wireless conditions.

Note that despite our best efforts our transfer finish times are not strict guarantees but educated guesses based on *current* conditions. As we show in our evaluation, Catnap is able to finish transfers on-time or with minimal delay in a variety of test scenarios. There could be other reasons why rescheduling may be required (e.g., arrival of new requests). As we allow new requests to be multiplexed with previously scheduled slots, there is no issue of *un-fairness* caused due to earlier scheduling decisions.

Batch Mode

Figure 4.5 lists the basic steps performed by the batching algorithm. These steps are performed for each client that wants batched transfers. In step 1, we calculate the batch

[4]Our current implementation uses a fixed value for this threshold.

1. Calculate batch size B
2. Estimate MAX_WAIT_THRESHOLD
3. Batch data until batch size >= B OR
   No packet for MAX_WAIT_THRESHOLD
4. Burst data

Figure 4.5: Basic steps performed in the batch mode.

size that can ensure a certain sleep interval for the device, computed through Equation 4.1. Of course, our target sleep time also depends on the delay that can be tolerated by the user. For example, if the user is short of battery she may be willing to tolerate more delay, while in other scenarios she may not tolerate any additional delay. In the current design, the user gives a binary indication i.e., whether it wants to use the batch mode or not. If batch mode is desired, then the Catnap service uses a fixed pre-determined threshold for the maximum additional delay that a transfer can bear (set to a fixed value of 2 sec in the current implementation).

In step 2 we consider situations where we may never get enough data for batching. To address these cases, we estimate MAX_WAIT_THRESHOLD, which is the maximum interval of time we should wait for new data to arrive for a certain batch. If we do not get any packet for this long, we send the partial batch to the client. In our current implementation, we set this threshold to twice the RTT between the TAP and server – as no packet during this duration is a good indicator that no more data will arrive in the near future. The above two steps determine how long we batch data for a given client.

4.4 Implementation

We have implemented a prototype of the Catnap service in C for the Linux environment. It works as a user level daemon and implements the core design components of Catnap while using simple and convenient alternatives for modules that are not central to the working of Catnap. We give a brief overview of the important parts of the implementation.

4.4.1 Sleep Modes

Our implementation supports the use of two types of sleep modes: i) 802.11 PSM and ii) S3 mode. We enabled the PSM mode on the client device – as a result it goes into sleep state whenever it encounters an idle period longer than a pre-determined duration. The minimum value of this duration varies for different NICs and was generally found to be between 30ms to 250ms for many popular wireless NICs. In the sleep mode, the NIC wakes

up at every beacon interval to check for incoming traffic. In summary, we used the standard PSM implementations supported by the NICs, without any Catnap specific modifications.[5]

Like the 802.11 PSM, we used standard mechanism of enabling S3 mode in client devices. In our current implementation, the client device goes into S3 state without notifying the user. However, in practice, we expect a proper user interface that can involve the user in this decision. In order to wake-up a sleeping client device, the TAP sends a Wake-On-LAN [16] magic packet to the client. Many recent wireless NICs do support the wireless WOL feature [17], but we did not have access to this hardware, so we used the wired ethernet interface for the magic packet. With the wireless WOL support, mobile devices will use their wireless interface in PSM mode to listen for magic packets from the Catnap service i.e., the client wireless NIC will wake up at every beacon interval to check for any magic packet from the TAP.

4.4.2 Scheduler

The centralized scheduler within the Catnap service is responsible for all scheduling decisions in both directions. We have implemented three types of schedulers: i) normal scheduler, which is not Catnap enabled, ii) Catnap-normal mode, and iii) Catnap-batch mode. On the wired side we use the normal scheduler while on the wireless side we use one of the two Catnap enabled modes. This implementation strategy allows us to use the same Catnap service at the client side as well. For example, we could have the client NIC sleep even longer if the client itself batched its own traffic towards the server.

Wired Bandwidth Estimation Module: This module assumes that the access link bandwidth is known and fixed (e.g., DSL). Future implementations can be extended to periodically use existing bandwidth estimation tools to estimate the access link bandwidth [66]. Once the transfer starts, this module keeps track of the throughput of the transfer – this allows identification of cases where the access link may not be the bottleneck (e.g., server being bottlenecked). This process is only applied to long transfers ($>$ 1 sec) because their throughput information is likely to be more accurate and also because larger transfers have more impact on other ongoing transfers.

Wireless Bandwidth Estimation Module: Our wireless bandwidth estimation method is inspired by prior work that utilizes medium busy time information, along with the bit-rate used between the client and AP, to estimate the available capacity [62, 85]. The product of these two terms i.e., (bit-rate * medium free time), is often a good estimate of the capacity, as it captures the channel conditions as well as the overall load on the network.

[5] How to optimize the PSM implementation for Catnap is an interesting future research direction.

For the medium free time, we use a passive approach of overhearing data transmissions in monitor mode and calculating the medium busy time based on a moving 1 sec window. In this window, we calculate the sum of the bytes corresponding to the overheard frames and the transmission time of these frames (based on the bit-rate of the transmission). Note that we need to ignore the time that the TAP uses the medium for its *own* transmissions. The medium free time information is combined with the bit-rate information[6] – that can be retrieved from most 802.11 drivers – to get the available capacity.

Note that the above mechanism relies on looking at a window of time to decide on the available bandwidth. Naturally, this does not account for cases where an unexpected burst of traffic from another station overlaps with a scheduled time slot. The underlying fairness provided by 802.11 helps in reducing the negative impact of such cross traffic. In addition, we inflate the virtual time slot by 10% as padding time to deal with such unexpected cross traffic. In our evaluation, we show that this approach works well under a variety of cross traffic scenarios.

4.5 Evaluation

The goal of our evaluation is to quantify the energy savings made possible by Catnap for different scenarios. Our evaluation can be divided into two parts. First, we consider several micro-benchmark scenarios related to Catnap's scheduling. In the second half of the evaluation, we evaluate the energy savings possible through Catnap on actual devices and for a variety of scenarios. Our experiments use a simple custom request-response application, which allows us to measure the benefits of Catnap without worrying about the complexities of different application protocols. Section 4.6 presents two case studies that show how Catnap can be leveraged by legacy application protocols: HTTP and IMAP.

4.5.1 Evaluation Summary

The key points of our evaluation are as follows:

- **Scheduling:** We show that the scheduler accurately estimates conditions on the wired and wireless segments, thereby ensuring maximum sleep time to clients with no added delay in transfer times. (Section 4.5.3)
- **Rescheduling:** We show how Catnap can efficiently address the challenge of rescheduling transfers. We consider several cases where rescheduling is required: i) server get-

[6]The actual achievable throughput is lower because of some fixed protocol overhead of 802.11 and TCP.

Figure 4.6: Network configuration used in the experiments.

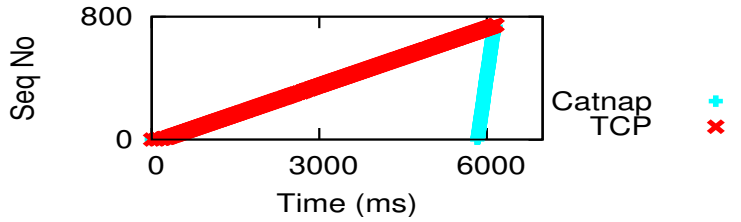
ting bottlenecked, ii) presence of wireless cross traffic, and iii) arrival of new requests at the TAP. (Section 4.5.3)

- **Energy Benefits:** Through experiments, we quantify the sleep time for both the wireless card and the device itself and how it translates into battery life improvement for real devices. Table 5.1 presents the summary of the evaluation. The benefits with Catnap get more pronounced as we increase the size of transfers. Specifically, Catnap allows the NIC to sleep for almost 70% of the time for a 5MB transfer while the device can sleep for around 40% of the time for a 10MB transfer. These energy savings translate into up to 2-5x battery life improvement for Nokia N810 and IBM Thinkpad T60. (Section 4.5.4)

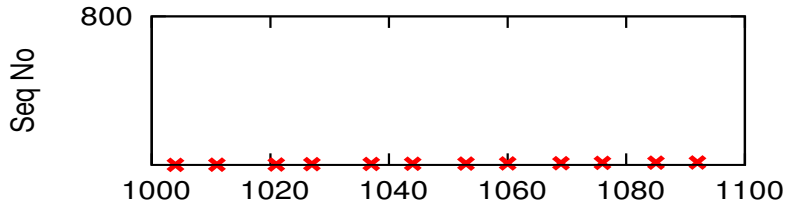
4.5.2 Setup

Figure 4.6 shows the network configuration used in the experiments; it represents a typical residential wireless access scenario. The wired WAN connection is emulated using the traffic control module of Linux. On the wireless side two different configurations are used. We used the Emulab wireless testbed for some of the experiments while for other experiments we used an actual AP connected to a Catnap enabled laptop via a 100Mbps connection. The main reason for using the Emulab testbed was the convenience of introducing cross traffic on the wireless segment (as it provides multiple wireless nodes), so we can evaluate the scheduler under different conditions.

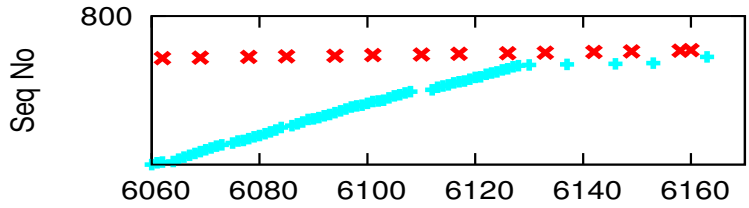
We used three client devices in our experiments. The first one is a Nokia N810 Internet Tablet which runs a light-weight Linux distribution. It has a 400Mhz processor with 128MB DDR RAM and also supports a 802.11g wireless interface with PSM capability. The tablet was running the default background processes during our experiments. The second device is a standard IBM thinkpad T60 laptop with support for 802.11 PSM and Ethernet Wake-



(a) Transfer Progress from start to end.



(b) Zooming into 1000-1100 ms period.



(c) Zooming into last 100 ms of transfer.

Figure 4.7: Single Transfer. Finish time is the same in both cases (a). Middle graph zooms into a middle period while the bottom graph zooms into the last 100ms of the transfer.

on-LAN. Finally, some of the experiments also involve an Emulab desktop node that is equipped with Wifi.

Our evaluation compares different modes of Catnap with standard end-to-end TCP-based transfers (TCP)[7]. Also, unless otherwise stated we present the mean of five runs, and the min and max values are within 5% of the mean.

4.5.3 Scheduler - Microbenchmarks

We conduct an evaluation of the efficacy of the scheduler using simple and realistic scenarios that a Catnap service will likely face. We first establish the accuracy of the scheduler and then show how it adapts to different situations by rescheduling transfers. All the results

[7]In our network configuration, the performance of end-to-end TCP and of using separate TCP sessions over the wired and wireless paths was similar.

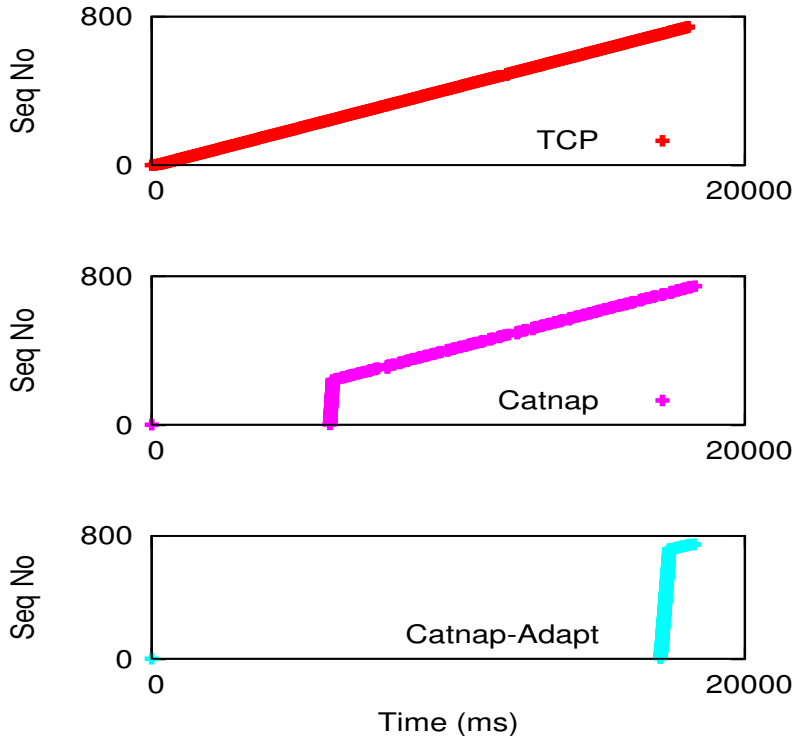


Figure 4.8: Reacting to a server being bottlenecked (500Kbps whereas access link is 1.5Mbps). Middle graph shows Catnap without adaptation – less batching due to estimation for earlier transfer completion. Last graph shows Catnap with adaptation.

in this section show the sequence number of packets received by the client as a function of time.

Scheduler Accuracy

In this experiment the network conditions are stable and there is only one transfer. The main goal is to test the accuracy of the wireless bandwidth estimation module. Figure 4.7(a) shows that transfer times are the same for both Catnap and TCP. Figure 4.7(b) zooms into the middle period of the transfer. We see that with TCP each packet arrives after a gap of a few milliseconds while Catnap is batching packets during this time. Figure 4.7(c) zooms into the last 100ms of the transfer. We see that Catnap is bursting packets to the client. The tail at the end shows Catnap's conservative approach that slightly inflates each virtual time slot to ensure that Catnap does not exceed the transfer deadline.

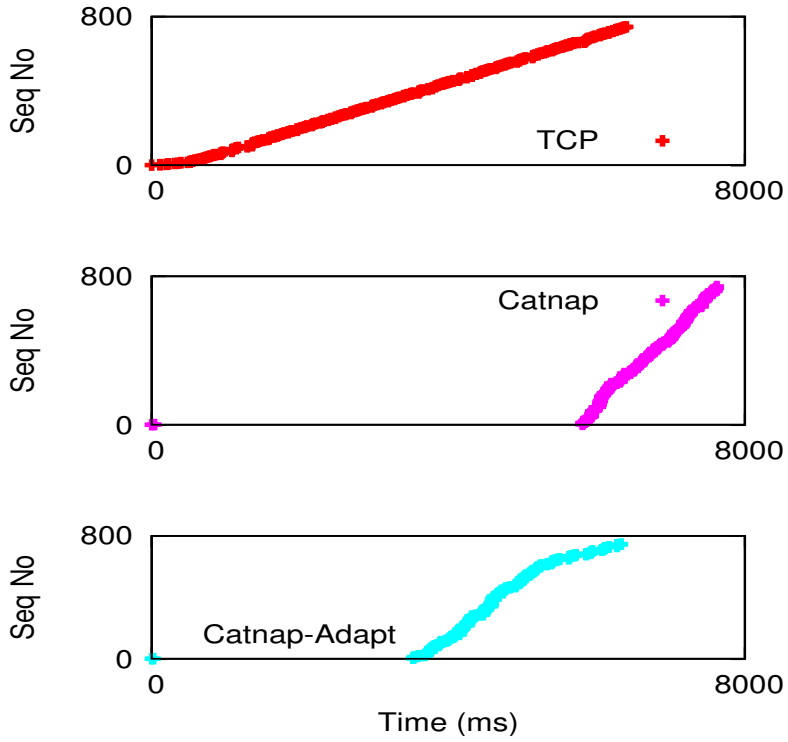


Figure 4.9: Reacting to TCP wireless cross traffic. Top graph shows TCP while middle one shows Catnap without adaptation. It anticipates more bw on the wireless side and therefore exceeds the transfer time while Catnap with adaptation is able to adjust (bottom graph).

Adaptation and Rescheduling

We now consider several scenarios where adaptation is required.

Adapting to changes in server bandwidth: We now consider the case where the server becomes the bottleneck soon after the start of the transfer. In the experiment, the server gets bottlenecked at 500Kbps. Figure 4.8(b) shows the case where the Catnap service does not adjust to the lower server throughput and continues to consider the bottleneck bandwidth to be 1.5Mbps. As a result, it sleeps less because it expects the transfer to end soon. In contrast, adjusting to the lower server bandwidth allows approximately 200% more sleep time to the client (Figure 4.8(c)). Note that the adaptation is accurate enough to ensure that the transfer time does not increase. Finally, because we constantly monitor the progress of the transfer, we can adjust to the conditions at any point during the transfer.

Adapting to wireless cross traffic: We now consider different cases where we explicitly induce cross traffic on the wireless side. In the first experiment, we start a long-lived TCP flow (cross traffic) during the batching phase of the Catnap transfer. In this scenario,

the wireless medium is heavily congested as the TCP background traffic uses a wireless bit rate of 1Mbps. Figure 4.9(c) shows that Catnap adapts to the cross traffic by rescheduling the time slot at an earlier time. Without such adaptation (Figure 4.9(b)), the transfer time is increased by around 15%.

We also conduct two experiments where the behavior of the cross traffic is less predictable i.e., ON-OFF behavior. In the *web-like* scenario, there is an OFF-period of 2 seconds after an ON period during which there are 5 transfers each of size 100kB. In the *Catnap-like* scenario, the cross traffic starts at the exact time as the Catnap service has scheduled the virtual time slot – in this case it is too late for Catnap to react. Both the above scenarios are more challenging cases because bandwidth estimation is less likely to work in scenarios where the cross traffic is bursty. We conduct 10 runs for these experiments, with each run starting at a random time.

	Mean	Max
Web-like Traffic	$< 2\%$ (120ms)	5% (300ms)
Catnap-Like Traffic	8% (480ms)	18% ($\approx$1sec)

Table 4.4: Mean and Max increase in transfer times under two different background traffic scenarios. With web-traffic, there is hardly any impact on Catnap transfers. If another Catnap-like TAP starts a burst in the background then transfer is delayed by 8% on average.

Table 4.4 summarizes the results for the above two scenarios. In the web-like scenario, there is hardly any impact on transfer times because of the light load of the cross traffic. As our bandwidth estimation considers the bytes transferred in a window of time, the OFF periods dominate the ON periods and therefore there is little impact on the available capacity. The worst case occurs when the Catnap time slot overlaps with the start of the ON period. However, as the results indicate, even the maximum increase in delay is just 5%. In contrast, the Catnap-like cross traffic can increase the transfer time by up to 18%. Again, the average increase is less (8%) because the extra padding with time slots allows us some leeway to deal with such unexpected cross traffic. These results show that, while we cannot provide a hard guarantee that the transfer time will not increase, we can minimize this chance and the impact is generally limited.

Adapting to multiple requests: We now consider scenarios where the Catnap service has to deal with multiple requests. We report the experimental results of the more challenging scenario where the virtual time slots of a future request overlaps with a previously scheduled transfer. Figure 4.10 shows the results of two overlapping requests. TCP multiplexes the two requests naturally whereas Catnap can handle them in two different ways (see Figure 4.4). Figure 4.10(b) shows the case where the requests are from the same client

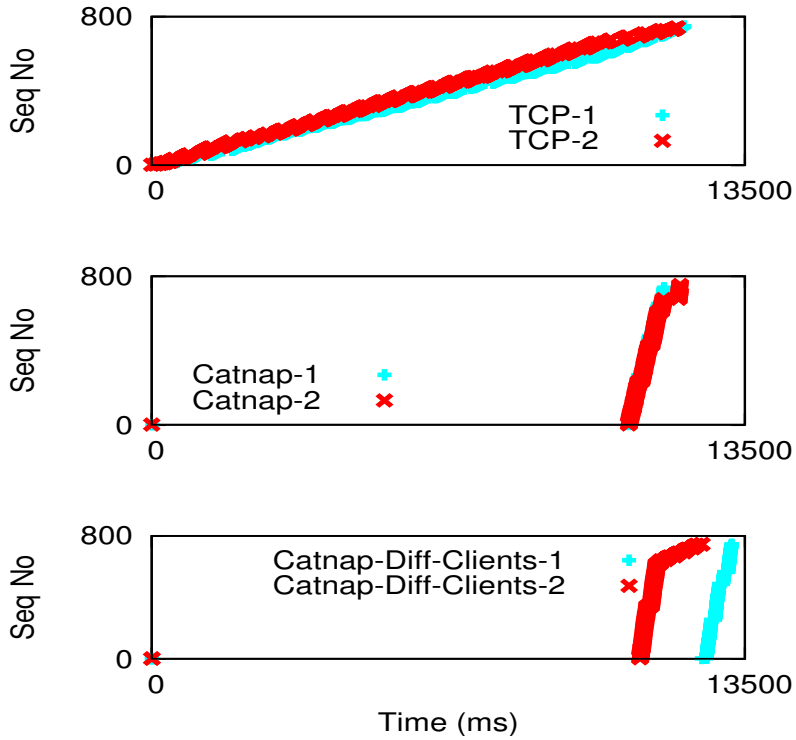


Figure 4.10: Two Overlapping Transfers. The middle graph shows multiplexing of time slots in Catnap if both the requests are from the same client (both transfers overlap so only one is visible) The bottom graph shows how requests from different clients will be allotted non-overlapping time slots in Catnap.

and are combined in a bigger time slot, while ensuring that transfer times do not increase. In contrast, Figure 4.10(c) shows the case where the requests are from different clients. In this case, we assign non-overlapping slots to both clients, which reduces the up time of *both* clients. However, it results in increased transfer delay for the second transfer – if this delay is not acceptable then we multiplex the two requests (similar to Figure 4.10(b)).

Discussion

The above experiments show that Catnap's bandwidth estimation is accurate and allows the scheduler to adapt to changes in network conditions as well as handle multiple requests that may overlap with each other. The results also show that variations on the wireless side are less critical as long as the available bandwidth on the wireless side is significantly higher than the bottleneck link bandwidth.

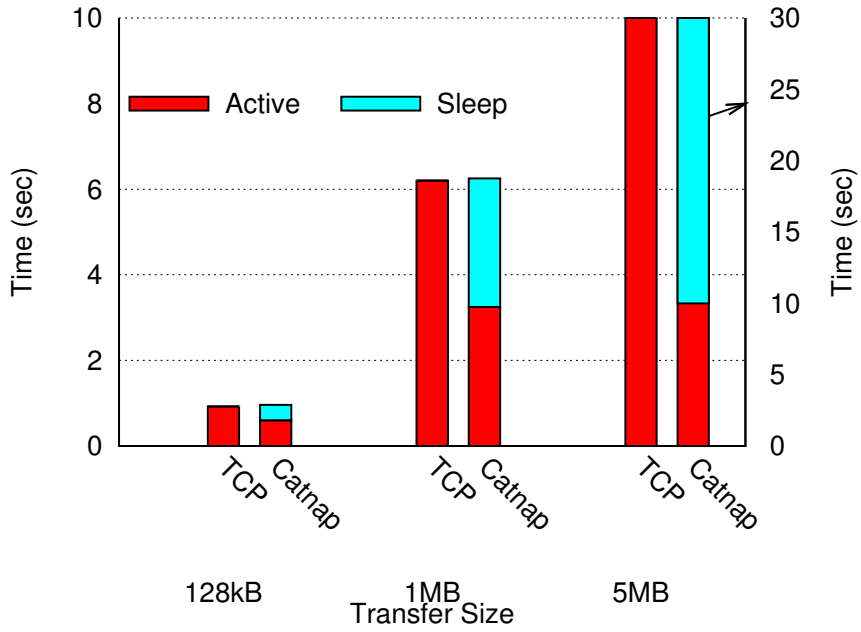


Figure 4.11: NIC Sleep time for Nokia N810. Catnap allows even 1 second long transfers to sleep, while leading to the same overall transfer completion time across all tested transfer sizes.

4.5.4 Energy Savings

We conducted several experiments to characterize the potential energy benefits of using Catnap. We have quantified both 802.11 PSM savings as well as savings using S3 mode. For the N810 platform we focus on PSM savings as network activity is the major power consumer and S3 mode does not offer much additional savings. In contrast, for T60 we focus only on S3 mode as PSM alone does not offer any worthwhile power savings.

NIC PSM Energy Savings

First, we want to evaluate how well Catnap performs in terms of energy benefits by allowing the NIC to sleep in-between transfers. For the experiments, we used the deepest PSM mode available in N810, which offers the greatest energy savings and enters sleep mode if there is an idle period of 200ms (we call this `psm_cost`).

How long can the NIC sleep. We evaluated the transfer time and the proportion of transfer time spent sleeping for TCP and Catnap. Figure 4.11 shows this comparison for different transfer sizes. Catnap amortizes the `psm_cost` over the total transfer time and therefore performs better as the transfer duration increases. As we can see from the figure, the transfer time does not increase with Catnap and yet it is able to sleep for around 30%

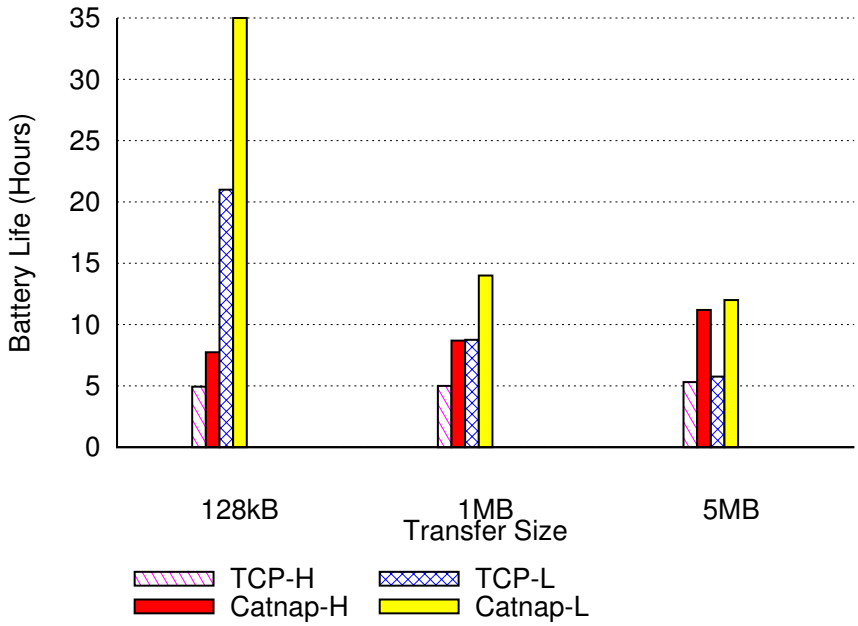


Figure 4.12: Battery life improvement for N810 due to NIC savings under heavy (H) and light (L) workloads.

of the time for a 128kB transfer, which is roughly the size of many popular web-pages. Furthermore, it can sleep for around 50% of the time for a 1MB transfer, which is roughly the pre-buffering requirement of a `you-tube` video, and around 70% of the time for a 5MB transfer, which is the size of a typical mp3 song.

Does increased NIC sleep time improve battery life? Here we want to quantify how increased NIC sleep times with Catnap translate into battery life improvement for the N810. This experiment uses two different workloads: i) heavy load (H) where transfers are made continuously, and ii) light load with an idle time of 5 seconds between transfers. We evaluate how long the battery lasts by fully charging it before the experiment and letting the experiment continue until the battery power completely drains out. The N810 automatically switches off the display back-light if there is no human activity, so all reported results are under this setting.

As Figure 4.12 shows, greater sleep opportunities in Catnap translate into significant battery life improvement over TCP. Our results confirm the N810 specifications that the device consumes negligible power in idle state. As a result, we notice that NIC power savings are beneficial for both heavy and light workloads, with up to 2x improvement in battery life.

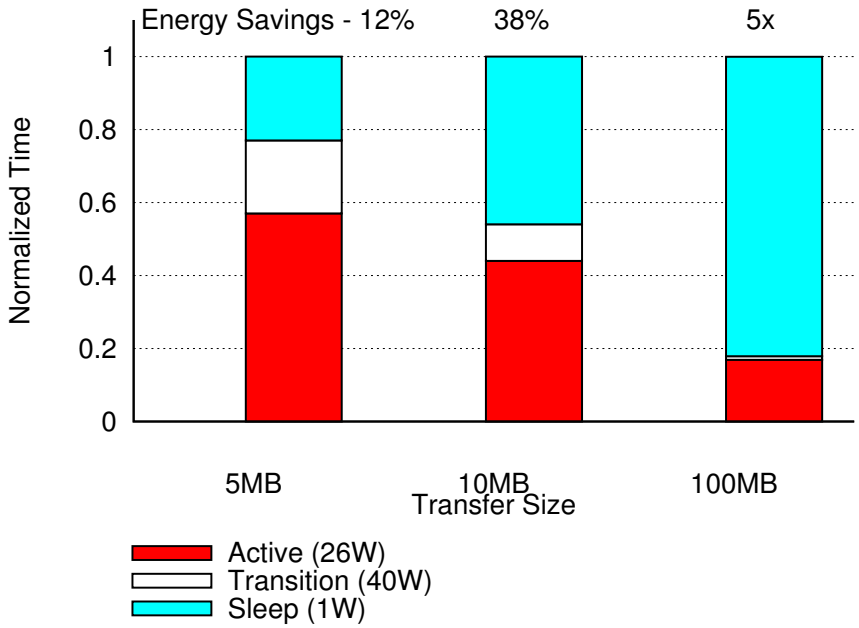


Figure 4.13: Catnap's transfer time and energy costs using S3 mode (normalized wrt TCP) for T60. TCP based transfers remain in active state for the whole duration.

Deep Sleep Mode (S3) Energy Savings

We wanted to evaluate the energy benefits of using Catnap with S3 mode for a laptop. As we noted earlier, S3 mode takes roughly 10 seconds to suspend and resume back, and is therefore only suitable for longer transfers. Future reduction to the S3 mode overhead could possibly make it useful even for smaller transfers. Note that Catnap just creates more opportunities to use S3 mode, so issues such as how to deal with incoming traffic when the client device is in S3 mode are orthogonal and can be addressed using existing mechanisms [20].

For this experiment, the client makes a request and then enters S3 mode. The Catnap service continues with the transfer, caches the data in its storage, and at the appropriate time wakes up the client by sending a wake-on-lan magic packet. The client wakes up and downloads the cached data from the TAP. Figure 4.13 shows the normalized transfer times with respect to baseline TCP transfers for different file sizes. It also shows the proportion of time Catnap spends in each of the three states: Active, Transition, and Sleep, in addition to the power consumed in these three states. Note that TCP transfers remain in active state for the whole duration but Catnap allows significant sleep time without increasing the total time to complete a transfer. More specifically, even for a 5MB transfer that only lasts

for 30 seconds, Catnap is able to spend around 20% of the time in S3 mode while for larger transfers (100MB) the proportion of sleep time increases to around 85%, which translates into 5x energy savings. If files are constantly downloaded, these energy savings correspond to up to 5x improvement in battery life.

4.6 Case Studies

The goal of the case studies is to gain some understanding of how different applications can benefit from Catnap. We pick two popular application protocols: IMAP and HTTP. Our goal is to make these protocols work with Catnap without modifying standard clients or servers. Furthermore, we are interested in evaluating the energy benefits when these applications use Catnap. Specifically, we focus on the challenging cases of short web transfers and small email message exchanges, which require the use of Catnap's batch mode in order to get energy benefits.

	Catnap-batch	TCP
Start-up Phase ($\approx$ 3 sec)	Same	Same
Download Phase (3.6 sec)	650 ms	3.6 sec

Table 4.5: Comparison of NIC up time for an email session involving 20 messages of 20kB each. In the start-up phase, both TCP and Catnap-batch perform the same but in the download phase, Catnap-batch results in around 80% less NIC up time.

.

4.6.1 IMAP

We enabled the use of standard IMAP clients and servers with Catnap. This required the use of a modified IMAP proxy at the TAP that parsed the responses from the IMAP server and added the length information as a hint for the Catnap service. The modification process was simple and required approximately 50 additional lines of code. All the features that were supported by the IMAP proxy worked fine with Catnap.

We observed that the IMAP behavior in practice is more complex than a simple request-response protocol. Broadly, there are four steps that a typical client follows in downloading email messages from the server: authentication, checking inbox status, downloading message headers, and downloading message bodies. We refer to the first three steps as the *start-up* phase and the last step as the *download* phase. Most clients only download the body when it is required, which means that for every message there is a separate request to download

it. However, this option can be changed by setting the client to work in *offline* mode where the client downloads all unread messages.

The start-up phase involves the exchange of short control messages, which are not large enough to get energy benefits. In the download phase, if the client is downloading large messages (e.g., attachments) then the cost of the start-up phase gets amortized and we observe energy benefits that are similar to our results in the previous section. However, most messages that do not have attachments are smaller than 100kB, which makes it difficult to get any energy benefits if we treat each message as a separate ADU. Here, we focus on how we can get energy benefits for small messages by batching them using Catnap's batch mode.

We conduct a simple experiment where we consider a traveller who has multiple unread messages to download (e.g., a professor checking her e-mail at the airport on her way to a conference). We consider the case when the email account has 20 unread messages of 20Kb each. This is a typical size of a call for paper announcement or an email belonging to a discussion thread on a mailing list. We compare the NIC up-time for Catnap's batch mode with that of using TCP. Table 4.5 shows that in the start-up phase, the up time is the same for both the schemes because there is not enough data for the batch mode to get any real benefit. However, once the download phase starts, the NIC is only active for 650ms with Catnap's batch mode whereas it is up for 3.6 seconds with TCP. This shows that if there are more messages – even if they are individually small – then the batch mode can combine them to get energy benefits.

4.6.2 HTTP

Catnap with web browsers and servers: As a first step we focused on using Catnap with standard web browsers and servers. We installed a modified squid proxy on the TAP – the Catnap service interacted with the squid proxy rather than with servers. The modification to the squid proxy involved adding the Catnap hint (i.e., transfer length) that is used by the Catnap proxy for scheduling. Adding this hint was simple as HTTP responses from servers already contain the length of the responses in the HTTP header.[8] The whole modification process involved the addition of 20 lines of code. We verified that the client browser could view different kinds of web-pages (youtube, cnn, static and dynamic pages, pages with SSL, etc) while using Catnap. Note that we could make the same changes to the web server, eliminating the need for the web proxy.

Why smaller web pages may not benefit. Typical HTTP protocol interactions

[8]Even dynamic content is sent as a chunk in HTTP and the response contains the length of the chunk.

	Total Size	Max Size Object
www.google.com	36kB	15kB
www.yahoo.com	540kB	48kB
www.amazon.com	580kB	60kB
www.nytimes.com	860kB	260kB
www.cnn.com	920kB	150kB

Table 4.6: Comparison of different popular web-pages in terms of total size and the size of the largest embedded object within that page. Individual objects are generally too small (median object size of these web-pages is also less than 50kB) to get any energy benefits but the total size for most pages is large enough to get energy benefits from Catnap.

between the browser and server are much more complex than a simple request-response exchange. A webpage has many embedded objects and they are usually fetched using separate requests over different TCP connections. We observed that for large transfers the cost of these multiple requests gets amortized and the energy savings were similar to the simple request-response protocol. However, for normal web browsing, energy benefits with Catnap were minimal. Table 4.6 provides an insight into the reasoning for this behavior. While most web-pages as a whole are big enough to benefit from Catnap, individual objects are usually smaller than 100kB, which makes it difficult to get any energy benefits. This indicates that we should treat the web page as one large block of data rather than separate small objects.

Catnap Batch mode: We wanted to evaluate the effectiveness of Catnap's batching mode for web-pages that have multiple embedded objects but no scripts. In such cases, the browser first makes a request for the main page and after it gets the response, it makes parallel requests for all the objects embedded within the page.

	Catnap-batch	TCP
Download Time	1.5 sec	1 sec
NIC up time	650 ms	1 sec

Table 4.7: Comparison of download time and NIC up time for a small web-page download. Catnap batch mode results in 35% less NIC up time compared to TCP.

.

We conduct a simple experiment by selecting a web-page with the above characteristics (www.cs.cmu.edu/~bmm). The objects within the page are individually all smaller than 100kB but their combined size is close to 150kB, which is large enough to get energy savings. Table 4.7 shows the download time and the NIC up time for Catnap-batch mode and TCP. Note that Catnap's normal mode performs the same in this case as TCP because

the objects are individually too small to get any energy benefits. Without Catnap it takes 1 sec to download the web-page and the NIC is active for the whole duration. With the use of Catnap-batching, the download time increases to 1.5 sec, however, the NIC is active for only 650ms, which shows a reduction of 35% in up-time for the NIC. In case of N810, which hardly consumes any power when the NIC is sleeping, the NIC savings translate into equivalent system level savings. However, because we are increasing the total time of transfers, devices that consume significant power even when their NIC is sleeping may not get similar benefits from the batch mode.

Embedded Scripts and Application Specific Techniques: In practice many web-pages have scripts that further complicate the HTTP protocol as browsers make further requests only after executing the script. The script execution can itself generate requests for new objects, making it difficult to batch large amount of data. Researchers have proposed *application specific* approaches that can turn the HTTP protocol into a single request-response exchange [106]. The general approach is to have the proxy respond with all the embedded objects (including embedded scripts), in response to a request for a web-page. Using such techniques at the TAP can extract maximum benefits out of Catnap as they turn the complex protocol interaction between the client and server into a simple protocol where the client makes a request for the main page and receives all the relevant objects in response. This could further improve the energy savings with Catnap. For example, for the above web-page experiment, the NIC up-time can be reduced to 450 ms if we use application specific support at the TAP.

4.7 User Considerations

Increased Delay: Most of our evaluation focuses on the case where Catnap does not add delay to transfers. However, increases in delay are possible, for example, when using Catnap's batch mode or when there is a sudden surge in load on the wireless network. Our results show that the additional delay is small – of course, the users can even avoid this delay if they do not want the extra energy savings. So there is clearly a trade-off that users need to consider while deciding on whether they want to use the batch mode or not.

Interface: Besides possible changes in delay, Catnap may also affect the interface of some applications. The most important interfacing issue is how to expose the use of S3 mode to the user. Most devices already have sensors to detect whether a user is currently using the device or not. We can leverage this support even though the user needs to be ultimately involved in the decision to enter the S3 state. This could be implemented in the same way as automatic reboots are implemented for system updates i.e., allowing the user

to explicitly cancel the action within a certain time limit.

In addition to the S3 mode, there could be other cases where the interface to the application needs to be changed. For example, for large downloads, users expect a status bar that shows the steady progress of the transfer. With Catnap the transfer stalls at the start and finishes with a rapid burst at the end. Despite such a change, the interface could certainly expose the predicted completion time as estimated by the scheduler, thus making the user experience more predictable.

Control: Note that many mobile devices already provide users with power-save options (e.g., dim screen, reduce processor frequency) that affect the user experience. Catnap naturally fits this model. The user interface for setting Catnap preferences (e.g., use of batch mode, tolerance to extra delay, use of S3 mode) could be integrated with existing power save applications. This will raise interesting questions of how users address the trade-off between improved battery life and possible increase in transfer time.

4.8 Related Work

The problem of improving energy efficiency of mobile devices has received considerable attention in the research community. Here, we discuss the work that is most relevant to Catnap.

Exploiting Idle Opportunities: There have been several proposals to exploit idle opportunities that are *already present* at different levels. Most of these proposals are orthogonal to Catnap as they target a different application or network scenario. For example, there are proposals that target idle time between web requests or during TCP slow start [75]. Similarly, some proposals focus on interactive applications [25, 86], or consider power savings for sending data over a GPRS link [106]. Catnap targets scenarios where applications are constantly sending data but there are idle opportunities due to the network characteristics i.e., bandwidth discrepancy between wired and wireless segments.

A recent proposal by Liu and Zhong [82] is most relevant to Catnap as it considers similar network settings. It exploits advances in modern 802.11 transceivers to enter low overhead power saving states during the small idle intervals between packets. Catnap takes a different approach: it *combines* these small gaps between packets into larger sleep intervals for applications that consume data in blocks – as a result, Catnap can exploit deeper sleep modes such as 802.11 PSM and S3 mode. We believe that future mobile devices will use both these techniques: for data oriented applications, aggressive sleep approaches, like Catnap, are more suitable, while for interactive applications, exploiting low power saving states becomes important.

Traffic Shaping: There is a significant body of work that uses traffic shaping in order to *create* more sleep opportunities for both the wireless NIC and the system [42, 57, 88, 91]. Most of these schemes shape traffic at the end-points, in an application specific manner [42, 91]. Nedveschi et al.[88] also consider traffic shaping within the network core, but at the expense of added delay for all application types. Unlike these proposals, Catnap shapes the traffic considering the bandwidth discrepancy between the wired and wireless segments. As a result, the traffic shaping can only be done at the TAP, rather than the end-points. Furthermore, Catnaps takes into account the challenges of scheduling and rescheduling transfers on the wireless segment – such challenges are not considered by existing proxy based solutions [100]. Finally, as Catnap uses workload hints, it can use sleep modes for long duration of time without affecting user experience.

Deep Sleep Modes: The S3 mode for Catnap is similar to the use of deep sleep modes in Somniloquoy [20] and Turducken [109]. The common theme is that the main system sleeps while another device keeps up with the data transfer. In case of Catnap, this device is the TAP while the other two systems rely on additional low power hardware that is co-located with the mobile device. Avoiding the introduction of additional hardware on the mobile device, however, comes with its own challenges since we now have to intelligently schedule transfers on the wireless segment. Another difference is that the aforementioned prior systems use a pure store and forward approach which increases the duration of a transfer. In contrast, Catnap uses a cut through approach where the transfers over the wired and wireless segments are overlapped in order to keep the transfer time the same. However, this requires workload hints from applications, which is not required in the other systems.

Hints and Batch Mode: Many design decisions made in Catnap are inspired by previous work in the area of energy management for mobile devices. Anand et al. [25] use application *hints* to determine the intent of applications while using the NIC, which helps in keeping the NIC active for interactive applications. However, for data oriented applications that constantly send data, their approach always keeps the NIC active. Catnap also uses hints from applications but the hint relates to the length of the transfer. Similarly, Catnap's batch mode is based on the principle of trading application performance for improved battery life – this principle is well-understood, for example, in the context of lowering the fidelity of multimedia information to improve energy consumption [59]. Catnap applies this principle in the context of increasing the transfer delay to get energy benefits. More recently, TailEnder explores the use of batching multiple requests at the client to improve energy efficiency of outgoing transmissions [33]. In contrast, Catnap's batching mechanism has broader applicability as it can be used at both the TAP and the client, thereby allowing

batching of both incoming and outgoing data at the client.

Discussion: In summary, Catnap's uniqueness comes from the following: i) unlike most work on energy management, Catnap focuses on periods of peak network activity when applications are constantly sending data, ii) it leverages a range of sleep modes (NIC PSM, S3) rather than focusing on a specific one, and iii) it uses ADUs for application independent and yet application aware energy savings. Catnap therefore occupies a unique space in the energy saving realm: it provides an alternate view of how data transfers should be conducted over the Internet to maximize power savings for mobile clients.

4.9 Summary

This chapter presented the design, implementation, and evaluation of Catnap, an in-network service that leverages key concepts of Tapa to provide up to 2-5x battery life improvement for mobile devices. Through an extensive evaluation, we showed how benefits of Catnap increase as the transfer size increases. As the use of data oriented applications over mobile devices increases, we believe that services like Catnap will become more important.

Chapter 5

Scaling Online Social Networks

Online Social Networks (OSN) have grown at an unprecedented scale in recent times. The huge amount of content generated and consumed in these networks create new challenges for content distribution. Based on a measurement study of Facebook, we highlight and quantify the limitations of existing web based content distribution mechanisms. In order to address the challenges of content distribution in OSNs, we make a case for using TAPs to store and distribute social networking content within a user's social community. We also design, implement, and evaluate Vigilante, a content distribution service that uses key concepts of Tapa to improve the performance and scalability of distributing content in OSNs.

5.1 Overview

OSNs have become ubiquitous. Facebook (FB) alone has over 500 million active users who share over 30 billion pieces of content (photos, links etc) every month [4]. Supporting this huge scale is a major challenge for OSN providers and despite significant investments in network and data center infrastructure, ensuring a high quality user experience still remains challenging. One of the key reasons is that OSNs fundamentally differ from the traditional web – both in terms of how content is generated and how it is accessed. In OSNs, content is generated at the edges and access patterns are highly localized within a typically small social community. As a result, we need to develop new techniques to optimize content publishing and retrieval in OSNs, since solutions developed for the web do not suffice.

In this chapter, we *quantify* the challenges in distributing content for large scale OSNs and *address* them with the help of Tapa. In order to *quantify* the limitations of traditional web based content distribution solutions, we conduct a comprehensive client side measurement study of (FB). Our study reveals that handling user generated content, whether it is

of a dynamic nature (e.g., new feeds) or static nature (e.g., photos), is the key challenge for OSN providers like FB. We show that even with the extensive use of caching within data centers and distributed caching using traditional CDNs such as Akamai, it is difficult to consistently provide good performance for services like photo sharing. Performance bottlenecks were observed both in uploading and downloading of photos on FB. For example, our results show as much as a tenfold increase in response times for photos that are not cached at the nearest CDN.

In order to *address* the problem of scaling OSNs to billions of users, we propose the use of TAPs (e.g., home AP, media center, etc) for storing and distributing online social networking content. So a user's TAP stores the content that is relevant for the user and also distributes it to her friends, if they require so. Such content centric networking amongst TAPs and end-points is feasible because *social relations* determine the content access patterns in an OSN. So a user's content retrieval is dictated by her *limited* social circle and *predictable* access patterns. For example, Alice will be interested in viewing photos uploaded by her close friends or the links that they post. These social relationships allow the TAP to identify and store a large amount of relevant content, which results in fast response times during content retrieval. This is true both for *on-demand* fetching, e.g., by retrieving content from a nearby TAP rather than a remote central server, as well as through *pre-fetching*, e.g. by proactively retrieving content from the TAPs of friends.

We show how Tapa facilitates the use of TAPs for storing and distributing OSN content. It provides support for *publishing*, *retrieving*, and *caching* content in an application independent manner. Social networking applications can *aid* these content centric functions of Tapa by providing *hints* or by pushing content in advance to suitable TAPs for performance or availability reasons.

Hints provide information to the TAP on the likely nodes who may have the required ADU in their cache. Hints can be generated and managed in a variety of ways. The simplest solution is to have application generated hints stored on a centralized server. For example, when Bob uploads content, it can include a hint that he is keeping a copy on his TAP. Similarly, after downloading Bob's photos, Alice can store a copy on her TAP and notify the server. Alternatively, hints can also be managed in an application independent manner by Tapa, for example by having TAPs exchange information about the content (i.e., the self-certifying ADU identifiers) they are caching.

Tapa can also easily support a variety of roles for the TAP in the context of OSNs. In the simplest case, the TAP just maintains a cache of content stored at the server, as described above. However, it is possible to make the TAP the primary source of content, with the server simply keeping a backup copy. It is also possible to control when data is uploaded

Type	Category	Generated By	Static/Dynamic	Scale	Strategy
Chat Messages	C1	Users	Dynamic	Huge	Handled by FB Chat Servers located on US West Coast
Message Feeds	C1	Users	Dynamic	Huge	Served through 2/3 FB sites in US (east and west coast)
Profile Data	C1	Users	Fairly Static	Medium	Same as above – CDNs are not used as profile view is customized for each user
Advertisement Images	C2	FB	Static	Small /Medium	Akamai CDNs are used to distribute content –Heavy Replication
Style Sheets, Javascripts	C2	FB	Static	Small	CDNs – Heavy Replication
Profile Photos	C3	Users	Fairly Static	Medium	CDNs – Medium/Low Replication
Photo Albums	C3	Users	Fairly Static	Huge	CDNs – Low Replication

Table 5.1: Different types of content on FB.

to the server, for example to reduce server peak loads. The precise role of the TAP can be controlled by the application by associating specific semantics with the transfer, using a suitable API.

Finally, we have built Vigilante, a service that leverages Tapa to support OSN content distribution. We have also built a photo distribution application that allows users to publish and retrieve photos within a social circle. We have evaluated Vigilante on the PlanetLab testbed under various workload and access scenarios. We have also quantified the benefits of, and trade-offs between, the different content distribution strategies and show how Vigilante can achieve high performance, scalability, and fault tolerance under various scenarios.

We make three main contributions in this chapter:

- **Measurement Study.** Our study includes a *synthesis* of information that is already known about FB and new insights that quantify the performance of their photo service. (Section 5.2)

- **A case for using TAPs as personal CDNs:** We establish the feasibility of using TAPs to store and distribute OSN content and also provide an analysis of different design options in this regard. (Section 5.3)

- **Vigilante:** We present the design, implementation and evaluation of Vigilante – a service that leverages Tapa for content distribution in OSNs. (Sections 5.4, 5.5 and 5.6)

Type/Category	Measurement Site	RTT (ms)
C1: FB chat (single site - west coast)	US-West	2.5
	US-East	70
	Europe	167
	Asia	222
	Australia	171
C1: user FB front page (two sites - east coast and west coast)	US-West	1.3
	US-East	7.5
	Europe	108
	Asia	222
	Australia	171
C2: static.ak.fbcdn.net (nearest Akamai server for javascripts etc.)	US-West	3
	US-East	0.5
	Europe	12.6
	Asia	21
	Australia	12.5
C3: photos-*.ak.fbcdn.net (nearest Akamai Server for photo albums)	US-West	3
	US-East	12.3
	Europe	35
	Asia	21
	Australia	12.5

Table 5.2: Observed RTT for different content servers.

5.2 Content Distribution in OSNs: A Measurement Study of Facebook

In this section we use Facebook as a case study to better understand the challenges of content distribution in large OSNs. We adopt a top-down approach, starting with a categorization of different types of content and then focusing on the performance of their photo service. The measurements and analysis conducted in this section was done in April 2010.

5.2.1 Categorization of Content

We consider a user's FB home-page and parse out information about different types of content. This helps us in categorizing content and identifying a typical server that serves that content. We subsequently identify the location of the server by measuring the RTT and path from different vantage points. These vantage points are PlanetLab nodes that are located all across the world. This provides us with valuable information about how the particular content is being served and the nearest location of the data source.

Table 5.1 lists the findings[1]. For each content type, we mention the key features as-

[1]We do not list the details of third-party applications or external links that are not served by FB. Nazir

sociated with it along with FB's content distribution strategy. The key features include how a particular content type is generated (user vs. FB), how quickly it changes (static vs. dynamic), and the scale at which it is generated. It is not surprising to find that these factors play a big role in determining the appropriate strategy. Broadly, there are three main categories:

C1 - User Generated Dynamic Content: This includes dynamic content (e.g., chat messages, status updates) as well as content that is highly customized for different users (e.g., profile views). This type of content is difficult to serve through CDNs as the content is dynamic in nature, can be updated at short time scales, and may require fine-grained access control. We observe that FB uses its two data centers in US to handle most of the content in this category. This is verified by FB's own engineering blog that indicates that the west coast site is the primary one and the east coast data center supports a fully replica of the primary [3]. We observe that the FB chat is handled by a single site on the west coast.

As the above content is served by one or more sites in US, the location of the client plays the most important role in determining the delay. This is clearly visible for client delays to the chat servers which are located at a single site. As shown in Table 5.2[2], clients on the west coast experiences a less than 5ms RTT while clients in Asia and Australia experience delays greater than 150ms.

In contrast, user's main FB page – which shows status updates and user profiles – is handled by two data centers. This improves the delays experienced by clients on the east coast as well as in Europe. Specifically, the east coast user observes an RTT of less than 10ms because of the presence of a data center on the east coast. In contrast, users in Asia and Australia do not gain, as suggested by their RTT to the server which is greater than 150ms.

C2 - FB Generated Static Content: This category includes content such as: ads[3], javascripts, style sheets, logos. The common feature is that FB is in control of when this data is generated, so they can make use of CDN services to distribute it in an efficient way. Most of this type of data (e.g., FB logo, website-layout, etc) is fairly static in nature and changes on the order of few months. Their overall size is small and almost all users require this data, which makes it ideal for distributing it through CDNs. We observed that FB uses Akamai to distribute content belonging to this category.

Table 5.2 shows the delay experienced by users in accessing this type of data. Unlike the

et al. cover issues related to third-party applications in detail [87].

[2]We report the average of 1000 pings. Min and Max were within 10% of the average.

[3]Even though ads are customized for each user but their generation is under the control of FB. All ads are replicated at CDNs – users are given the ad URLs that are relevant to them

previous category where data was served by FB servers in US, this data is served by nearby Akamai servers. As a result, delay is fairly low even for clients in Asia and Australia. This is the well-expected benefits of using a CDN like Akamai which has presence all over the world.

C3 - User Generated Static Content: This includes profile pictures and photo albums of users. There are three key features of this data-type that makes it different from traditional static content (e.g., content in the previous category). First, it is generated by users – they decide when to change their profile picture as well as when to upload photo albums. As a result, FB does not have control over the generation of this type of data. Second, the amount of data in this category is huge, especially for photo albums. Third, unlike the content in previous category which is needed by most users, the content access in this category is user dependent.

The content in this category is also different from other user generated content (C1) because photo albums are write-once and users do not usually change their profile photos at short time scales. These reasons appear to play a key role in FB's strategy of using Akamai CDNs to distribute this content.

Unlike the previous category where data is heavily replicated, we observed that there were relatively fewer Akamai CDN sites that served photo requests. This is indicated by the results in Table 5.2 which shows that for some sites (US East Coast and Europe) the nearest photo CDN had greater RTT compared to the nearest CDN that served other static content. However, for some sites (e.g., Asia and Australia), the same CDN served both types of content and the RTTs were the same. We observed that most of these cases were outside Europe and US where Akamai may have fewer CDN sites.

In order to verify the above observations on a larger scale, we used a publicly available ping tool[4] to measure the RTT to nearby C2 and C3 servers from 40 different vantage points. The results show that a significant fraction of the vantage points (>30%) observe a difference of at least 30ms in RTT between the nearest C2 and C3 CDN servers i.e., the RTT to the nearest C2 server was at least 30ms less than the RTT to the nearest photo CDN server. Around 20% of the points observed similar RTT for both C2 and C3 servers.

Finally, we also observed that in some cases different servers were used for profile and photo album pictures. Again, servers for profile pictures were likely to be closer (greater replication) compared to photo album servers, as the scale of profile pictures is much smaller compared to photo album pictures.

Discussion: The above discussion highlights three important points. First, static content generated by FB is the easiest to handle (C2). They use existing CDNs to heavily

[4] just-ping.com

replicate this data and are able to serve it to users with low delay. Second, dynamic content is difficult to handle as it has to be served through a limited number of sites (C1). This makes it difficult to reduce delay for users who are far away from those sites. Third, dealing with user generated static content is also a problem. Even though FB uses Akamai to serve this type of content, but the sheer size of their photo requirements makes it challenging to use standard CDN replication techniques.

5.2.2 Measuring Photo CDN Service

In this section we evaluate the performance of FB's photo CDN service. We focus on this because the sheer volume of data in this category makes it challenging to effectively use traditional CDN optimizations [35]. Moreover, our personal experiences and anecdotal evidence indicates that user experience while viewing photo albums on FB is often poor. We first explain the unique measurement strategy that we adopted and then focus on our evaluation that highlights the key factors that influence performance. This includes the availability of a photo at the nearest CDN, and the RTT between the client and the nearest CDN as well as the original root photo servers.

Measurement Methodology:

We explain how we conduct controlled experiments and measure latency for photo requests.

Controlled Experiments on FB: We adopt a novel strategy to measure the effectiveness of CDNs in distributing photos to the clients. We leverage the fact that in this case users *themselves* generate the content that is being served. Moreover, most OSNs allow privacy features that can be used to control the access rights to a particular data item. For example, a FB user can upload a photo album and restrict its viewing rights such that only her friends can view the album, or no one is allowed to view the album. We leverage these features to control *when* data is made available and *how* it is accessed.

Our measurement strategy is a departure from previous studies that focused on CDNs that serve provider generated content (e.g., content generated by cnn, nytimes etc.) The main constraint in these scenarios is that clients do not have control over *when* data is generated and *who* accesses it. This restricted earlier studies to focus on certain properties of CDNs that were independent of the content they served (e.g., DNS delay, load on server) [76, 112].

Measuring Latency: As we point out earlier, latency is the key metric for FB data access. Most data items within FB are represented as independent HTTP objects – client needs to make separate HTTP requests in order to fetch them. It is easy to measure the

total response time (TRT) of a request, as it is the time between when an HTTP request was generated and when the complete HTTP response was received.

There are several factors that contribute towards TRT. We focus on two factors: RTT and Server Delay, as they play an important role for small TCP transfers. Moreover, content providers have more control over reducing these types of delays. We use the `ping` tool to measure the RTT between client and server. For the server delay, we measure the the time between the following two packets sent by the server: i) TCP ACK packet for the HTTP request sent by the client, and ii) the first packet of the HTTP response.[5]

Impact of Cache Misses - Single Site

As a first step we focus on the performance results from a single site as it helps in understanding the key factors that influence performance.

Experiment Setup: We upload 1000 copies of the same image on a FB user account. The image once stored is 80kB in size. The album viewing rights are restricted, so it is not visible to anyone. This provides us with complete control over how many times a particular photo is accessed and the locations from which the requests are made. We parse out the URLs corresponding to the images in this album – they are of the form: `photos-*.ak.fbcdn.net/hphotos-ak-ash1/hs436.ash1/24068_392106023813_503448813_3786155_5239605_n.jpg`. The * corresponds to eight different servers used by FB (a to h) and images appear to be randomly assigned to any one of these servers. Finally, anyone who knows the URL can fetch the image.

We pick a client node in Utah. The RTT between the client and nearest photo CDN is around 40ms. The client uses the `wget` HTTP client to retrieve a photo. For each photo URL we make three back-to-back requests i.e., a subsequent request is made only after the previous request is finished. We verified that later access times (fourth request onwards) were similar to the third request if requests are made within a short time-span.[6] There is a gap of 3 minutes after which the client moves to the next image. The whole experiments requires approximately 1 day to finish. We record all HTTP packet exchanges using `tcpdump` and later process the logs to calculate different statistics for each request.

Results: Figure 5.1 shows the response times for a sample of 20 photos to illustrate the key insight. As we can see, the response times of the first photo request is higher compared to second and third requests. *This is because if the image is fetched for the first time, it is*

[5] Of course, the ACK can be piggy-backed on the HTTP response message – in this case server delay is zero.

[6] Even though photos remain in cache for a variable amount of time but we are almost guaranteed to find them in the cache if a repeat request is made within a few minutes.

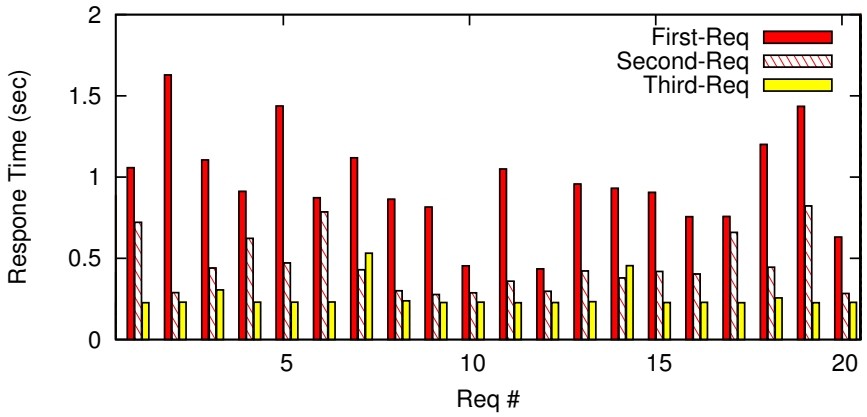


Figure 5.1: Response times for photo requests.

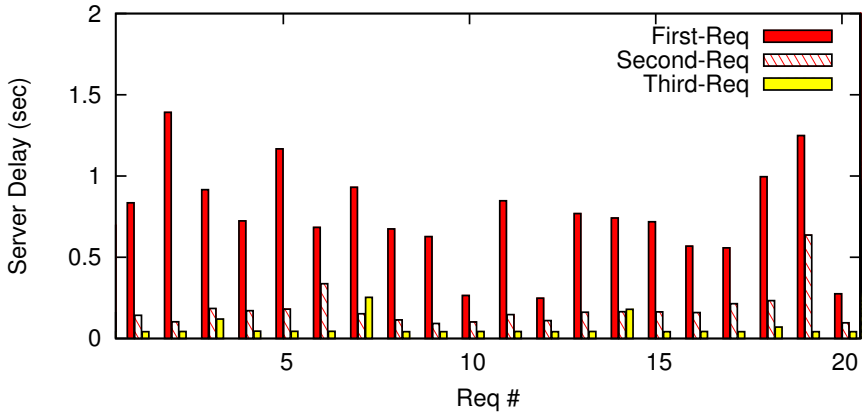


Figure 5.2: Server delay for photo requests

	First-Req	**Second-Req**	**Third-Req**
Mean	1sec	443ms	263ms
SD	380ms	196ms	152ms

Table 5.3: Response Times for 1000 photos. First request takes longer because of cache miss while third request takes less time because photo is cached.

not available at the CDN and there is a cache miss. This is verified by Figure 5.2 which plots the server delay (time between ACK for HTTP request and first packet of HTTP response). The results verify that for the first request, server takes significant time to respond whereas for the other two requests the server delay is much lower.

Table 5.3 shows the average and standard deviation of 1000 photos for all three types of requests. If there is a cache miss (first request) then the average response time is 1sec

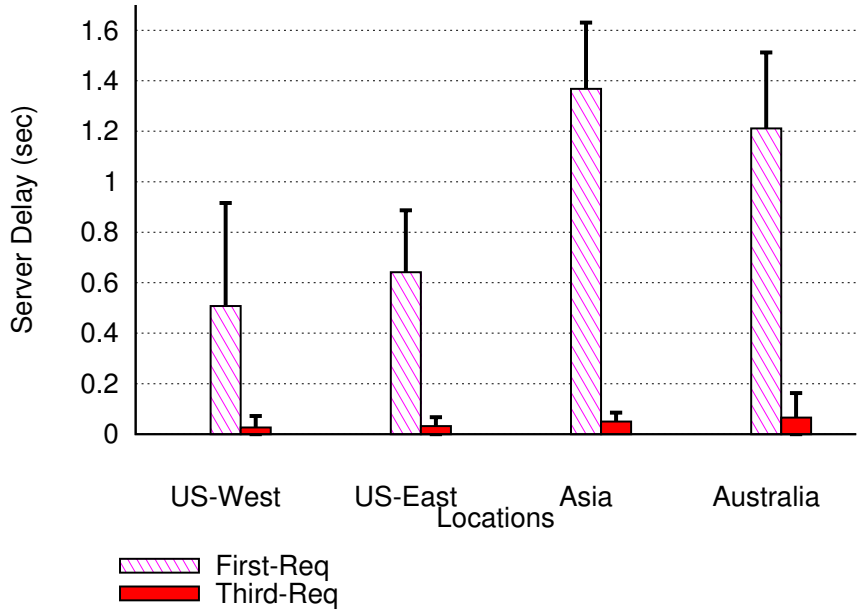


Figure 5.3: Average Server Delay for multiple locations. Error bar shows the standard deviation. First-Request results in cache miss and the impact is more for clients that are far away from US.

whereas if the photo is available at the CDN (third request) then the average response time reduces to 263ms. The difference between the second and third requests suggests that the nearest CDN does not cache an image on the first hit. There appears to be a multi-level cache hierarchy as the response times of the second request is lower than the response times of the first request. Finally, the third request incurs the lowest response time because the image is now cached at the CDN.

Cache Misses - Multiple Sites

We now consider multiple vantage points and evaluate the impact of different locations and RTTs on cache misses.

Experiment Setup: We pick clients at 4 different locations – two in US and one each in Asia and Australia. Each client fetches 100 images that have never been fetched before. Similar to the last experiment we make three back-to-back requests in order to quantify the performance difference when there is a cache miss and when there is a cache hit.

Everyone Observes Cache Misses. Figure 5.3 shows the average server delay for all four client locations. We only show results for the first and third requests i.e., when photo is not available at the nearest CDN and when it is available. The results verify our earlier

observation regarding cache misses – clients in all locations experience cache misses for the first request.

Client location matters when there is a cache miss. The results show a significant difference in the average server delay between different locations. This indicates that if the CDN does not have the photo, it fetches it from the photo root server. This process involves going over a WAN link and therefore the location of the CDN vis-a-vis the root server becomes important. The results hint on the possibility of the root photo server to be in the US. This explains why the average server delay for locations in US are lower compared to server delays experienced by clients in Australia and Asia. Such clients suffer more because their CDN has to possibly go all the way to the US to fetch the image.

Client location does not matter if there is a cache hit. Finally, if there is a cache hit then server delays are roughly the same irrespective of client locations (See results for Third-Req in Figure 5.3). The small difference can be possibly explained by the variable amount of load on different CDNs.

The above result shows that if the photo is available at the nearest CDN then clients in Asia and Australia experience performance which is at par with that of clients in US. This also means that clients in Australia and Asia observe much more variation in performance – a cache miss can increase transfer times by up to 20x compared to if the photo is available at the nearest CDN.

Cache Misses for Real Images

In the previous experiments, we used images that were not accessible to everyone. This helped us in understanding the issue of cache misses. Next, we wanted to observe the impact of cache misses on real images that are uploaded by FB users.

Experiment Setup: We crawled the photo albums posted by multiple friends of a single user. The data-set consisted of 700 images of size 10kB to 100kB. All these images were posted at least one day before we ran the experiments. We adopt the same process of fetching images multiple times but had a smaller gap of 3 seconds between different URL fetch requests.

All first requests are cache misses. We observed that even for real images the first requests were always cache misses. This behavior was observed for all locations. This suggests that there were no recent requests from other users for these images. This led us to investigate the amount of time the images are likely to remain in cache.

Images start getting evicted on the order of hours. We ran a long experiment that spanned one week. We repeatedly fetched the above data-set over this period from

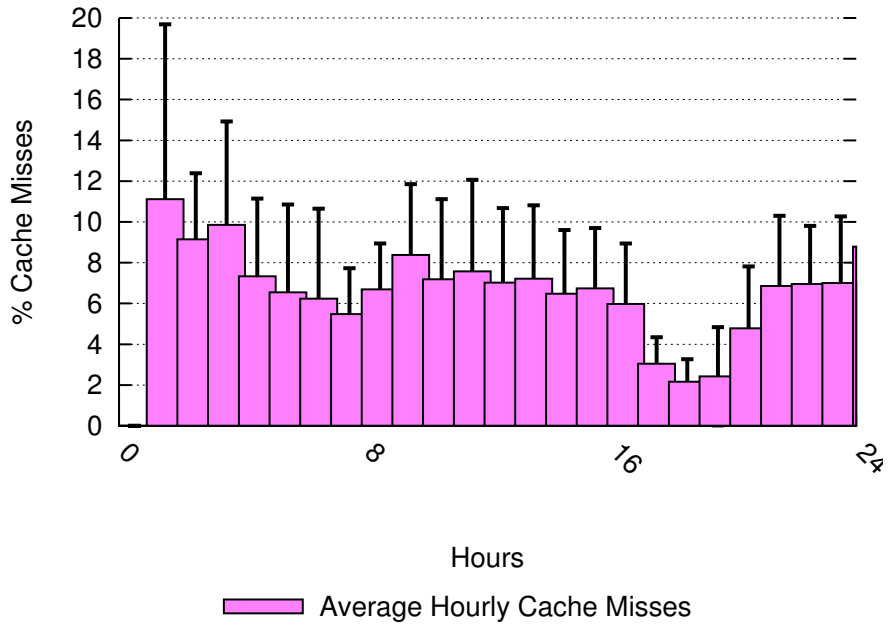


Figure 5.4: Cache Miss Rate at different hours of the day. Average of more than 100K requests over a one week period. Error bar shows the standard deviation. Results indicate that every hour around 5-10% of the photos get evicted from cache.

	US-West	**US-East**	**Europe**	**Asia**	**Aus**
Mean	4.61sec	4.64	5.17	5.17	5.22

Table 5.4: Average Time to upload a photo to FB from different locations. Variance was within 10% of the mean.

a single client, resulting in over 100K image fetches. Each iteration of the data-set took almost one hour, so every hour we make requests for the same images once again.

Figure 5.4 shows the cache miss rate on a per-hour basis. Based on the server delay, we characterize whether a request was a cache miss or a hit. We do not plot the initial part where all the requests result in cache misses. As we can see from the graph, every hour we see around 5-10% of the images being evicted from the cache, thereby causing a cache miss. Recent work by FB researchers [35] also indicates that handling these cache misses is indeed an important problem for them.

Photo Uploads

We also conduct experiments to measure the performance of uploading photos to FB. We write a customized application which uses the graph API provided by FB to upload photos.

This allows us to automate the task of uploading photos, thereby enabling large scale experiments. The API allows a photo to be uploaded to a specific user account – it takes the photo and user account information as parameters and returns a unique id that can be later used to fetch the photo.

Table 5.4 presents the results for uploading photos from five different clients located all around the world. Each client uploads 1000 photos and we record the time it takes to successfully complete the API call – this includes uploading the data as well as getting the id in return. As the results show, the mean time to upload a single photo is quite high – irrespective of the location, it take at least 4 seconds to upload a 100kB photo to FB. We observed that the upload bandwidth is not the limiting factor, rather it is the server overhead of storing the photo and returning its id that becomes the main bottleneck[7].

In summary, the above results show that performance of photo uploads is quite poor despite massive efforts to optimize it on the server end. It shows that a traditional centralized approach of managing uploads does not scale for millions of users, thereby making it hard to provide good performance.

5.3 A Case for Personal CDNs

We first provide a high level motivation behind the use of TAPs to support content distribution in OSNs. This is followed by a discussion on different design options for using TAPs. Finally, we discuss the key advantages and challenges involved with the use of TAPs in OSNs.

5.3.1 Motivating Scenario

Alice is a student living in New York. She uses an OSN to share content with her friends in New York as well as her family back in London. Let us see how a personal CDN amongst TAPs can support her OSN use.

Alice regularly uploads her photo album to the OSN, but she also caches the content on her home TAP. Soon her friends who are online at that time come to know that Alice has uploaded a new photo album. After a recent upload, Julie, her close friend, is the first to make a request to view the new uploaded photos. As Julie's TAP is part of Alice's personal CDN, it is able to retrieve Alice's photos from Alice's TAP rather than going to a far-away server. Performance is good because Alice and Julie live in the same city/region, which is

[7]We identified this by uploading invalid photos that are immediately rejected by the server. Such invalid photos take significantly less time to upload.

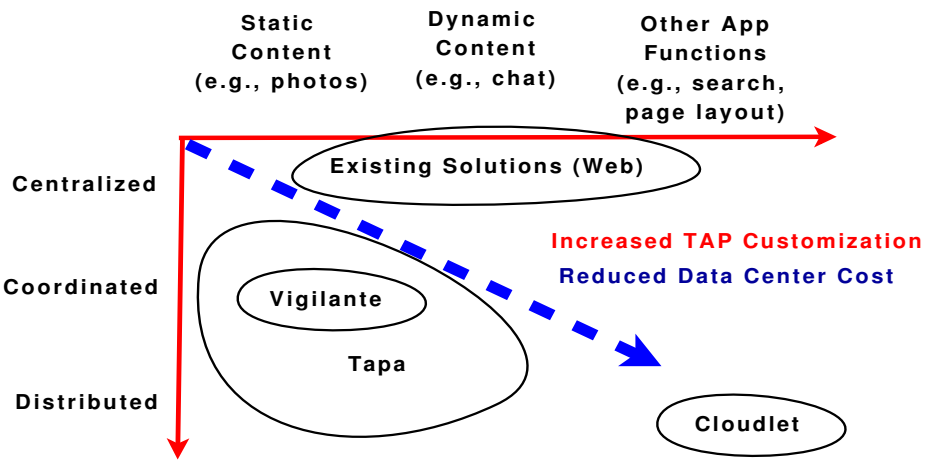


Figure 5.5: Different design options for leveraging TAPs

true for a high percentage of people befriending each other in OSNs [118]. Julie's TAP can also cache Alice's photos, offering an alternate option for future downloads.

When other friends of Alice start making requests to view her photos, they can download the photos from *TAPs of both Alice and Julie* since their TAPs are also part of the same CDN. Friends who live in New York will be able to retrieve the photos with low response times and having multiple copies also improves availability, e.g. in case Alice's TAP is down or unreachable. As more friends cache Alice's photos on their TAP after downloading them, opportunities for fault-tolerance and load balancing for content distribution within the OSN community increase.

Access for Alice's family in London can be supported similarly. The first download may retrieve the photos from an OSN server or from Alice's TAP. Once the content is cached on a London TAP, later accesses by Alice's family can be served locally with low latency. An alternative is that Alice's OSN application *pre-loads* Alice's photos on the TAP of one of her relatives in London. Since the OSN application has access to Alice's social network information, it can determine where pre-loading can be advantageous.

We now elaborate on how a personal CDN can best support the scenarios described in our example.

5.3.2 Design Space

The concept of leveraging TAPs for social networking applications is quite broad and there is room for different design options. Figure 5.5 presents some points in this design space. The y-axis shows different models for managing the personal CDN, while the x-axis shows

what aspects of the OSN applications are supported by the CDN, including distribution of static content (e.g., photos), dynamic content (e.g., chat), and other application functions (e.g., search or displaying ads).

The top of the figure shows the model that is typically used today, **centralized solutions**, in which all functions, including content distribution, are managed in a centralized manner. Centralized solutions have relatively low management overhead since the application logic resides at few sites, but they do not scale well and put a huge burden on the OSN provider for provisioning large data centers and CDNs. Performance can also be poor because central servers are a bottleneck. Traditional web-based CDNs can be used to improve performance, but as we showed in the previous section, the benefits for downloads are uneven, and uploading does not benefit at all.

On the other extreme are **fully distributed solutions** that place TAPs fully in-charge of content distribution (bottom of the figure). Distribution of static and dynamic content is managed cooperatively by the TAPs. This is the model naturally supported by Tapa. The use of TAPs for distributing content reduces the infrastructure requirements for the OSN provider because it no longer needs to support large scale data centers or web CDNs to distribute content. However, providing good performance and availability is a challenge since TAPs are likely to be running in heterogeneous, unmanaged environments (homes). Also, supporting dynamic content often requires more application specific help from the TAP. For example, when a user updates her status, the TAP needs to know who amongst her friends needs to get this update.

If full application functionality is implemented at the TAP (bottom right corner in figure) then we can view the TAPs in the context of cloudlets [103] – the TAP acts as a nearby cloud that can support OSN server and application functions. This has the advantage of reducing the infrastructure that must be deployed and maintained by the OSN provider. However, management becomes more challenging. For example, introducing a new feature would require updating the application plug-ins running on all TAPs.

Coordinated Solutions: We can also imagine a middle ground where the server *aids* the TAPs in performing various functions. This could be in the form of providing information about availability of content at various TAPs or maintaining backups of the data for fault tolerance. Based on this model, we have designed Vigilante, a content distribution service between end-points and TAPs that makes use of some help from the server.

5.3.3 Feasibility

We discuss factors that make personal CDNs an attractive and timely solution for optimizing OSN content distribution.

Unique Application Characteristics. In social networking applications, a user's data access is limited by her social links and interests. This has two implications. First, a user is only interested in a small subset of the overall data stored by the OSN. Second, the content generated by a user is only accessed by a small number of users i.e., belonging to her social community. For example, an average facebook user has 130 friends and creates only 90 pieces of content each month [4]. This small scale makes it relatively easy to apply concepts from content centric networking. For example, content routing/discovery within a social community is much more manageable compared to, say, content routing on a global Internet scale.

Technology Trends: Several trends in technology make it feasible to have a device in a user's home that can perform the functions of a TAP. First, storage is very cheap and most home devices can easily be supplemented with extra storage in the form of hard disks or flash drives. Second, access link bandwidth is increasing, both in terms of download and upload capacity. Third, devices such as media storage devices and wireless routers are mostly turned on and are sitting idle, so the down-time and churn of TAPs is much lower than for end devices.

5.3.4 Benefits and Challenges

While the benefits depend on the specific design point, the use of personal CDNs can offer advantages in three areas. First, in most cases, clients can obtain data either from their home TAP or a nearby TAP. Data cached on the home TAP can be downloaded using fast LAN technologies (e.g. 802.11n, Ethernet), while transfers from nearby TAPs will benefit from lower round trip times and higher rates. Also, TAP acts as a common cache for all applications or devices in the home environment, which are used to access the OSN. Second, when data is stored at multiple TAPs then this added redundancy can provide improved resilience against network failures. This is due to geographic as well as ISP diversity that is likely to be present within the social community of a user. Finally, the use of TAPs reduces the load on the servers in the OSN data centers. As a result, OSN provider's cost on servers, network bandwidth and indirect expenses (e.g., power, infrastructure, etc) is reduced.

However, obtaining these benefits involves several challenges, especially considering that TAPs are deployed in an un-managed environment. A first challenge is that TAPs have lower processing capabilities and network bandwidth than full scale data centers. As a result,

ensuring consistently good performance requires careful design of the personal CDN. A second challenge is fault tolerance. The likelihood of a single TAP being down or unavailable is possibly higher compared to servers in data centers. As a result, the personal CDN must handle TAP failures effectively. Finally, we would like to avoid or minimize the deployment of application logic on the TAP, since it adds complexity to be deployment and management of OSN applications.

5.4 Vigilante: Leveraging Tapa to Scale OSNs

Tapa can address the challenges identified above such that TAPs can be used effectively to store and distribute OSN content with minimal application help. We first give an overview, Vigilante, a content distribution service in Tapa. This is followed by a description of how Vigilante publishes and retrieves content using key Tapa concepts. Finally, we discuss the application support that is required in Vigilante for achieving the desired performance and fault-tolerance objectives.

5.4.1 Overview

Vigilante has two components. First, a generic content centric network component implements basic CDN functionality. The second component supports appropriate delivery semantics involving the end-points and TAPs, thereby allowing diverse roles for TAP/Server in content distribution within an OSN. Vigilante uses the basic CDN functionality provided by Tapa's transfer layer, but it can also use other content centric networks, either native (e.g. CCN [65]) or overlays [98]. The delivery semantics provided in Vigilante are part of the responsibility of Tapa's session layer and are not provided in the above mentioned content centric architectures.

Tapa's transfer layer implements basic *content centric network* functions between TAPs and end-points of an OSN community. Naturally these functions are performed at the data granularity of application data units (ADU) [45, 54] (e.g., an image). Large objects, such as videos, would be broken up into multiple ADUs. Recall that we use content hashes to identify ADUs [54, 113]; this facilitates application independent caching and content retrieval in Vigilante. So if an application wants to retrieve an ADU, it specifies the self-certifying content identifier corresponding to the ADU and possibly some *hints* on TAPs that can serve this ADU to Tapa. It is then up to the transfer layer to retrieve the ADU based on the application provided hints or its own mechanisms for discovering content availability at other TAPs within the OSN community.

The second key function supported in Vigilante is implementing specific *semantics* for how content should be distributed with the involvement of end-points and TAP(s). This is important in order to support a variety of roles that the TAPs can play in managing OSN content, as described earlier in Figure 5.5 (e.g., as the primary source of data, cache of data kept at the server, etc). These semantics mostly relate to how ADUs are delivered between different entities within the OSN. For example, application may differentiate between when data is received and acknowledged by the TAP compared to when it is acknowledged by the server. To this end, Vigilante makes use of the delegation semantics provided by Tapa's session layer (Chapter 6, Semantics).

We note that Vigilante does not provide any consistency or fault-tolerance and leaves it up to the application to implement its own consistency semantics and fault tolerance on top of the generic content publishing, retrieval, and caching support provided by Vigilante. The issues of consistency arises as Vigilante maintains multiple copies of the data at different TAPs so an update to one copy may render other copies stale. In the current prototype, we only focus on static photos, but applications can easily provide consistency on top of Vigilante. In such cases, Vigilante's use of content hashes to name ADUs ensures that as long as the client has the latest name then it will not get an old copy of the data. Similarly, it is up to the application to ensure that adequate copes of the data are created in advance, depending on the fault tolerance requirements.

1. Applications make a **put** call to Tapa
2. Client **transfers ADU** to the TAP
3. ADU is **cached** by TAP
4. App Notification for new published data is sent
5. ADU is sent from the TAP to the server
6. ADU is pre-loaded to other TAPs (optional)

Figure 5.6: Basic steps in publishing content.

5.4.2 Publishing Content

Figure 5.6 lists the main steps in publishing new content. Application uses the `put` API of Tapa to publish an ADU. The client transfer layer *establishes a segment* with the TAP based on the requirements specified by the application. Broadly, there are two types of segments. "Fast" segments are traditional network connections that use protocols such as TCP to transfer ADUs immediately. "Slow" segments transfer data in the background, in a rate limited fashion; they reduce the load on the network and on the destination device and can be used for ADUs that are not immediately required. These type of segments are

important in a personal CDN where some ADUs may be transferred as part of pre-loading or backup purposes and therefore do not not require immediate delivery.

Once the TAP successfully receives the ADU, it *caches* the ADU and also stores the policies/rules regarding a cache hit as well as eviction policy for the ADU. For example, our prototype supports sending a notification to the server whenever there is a cache hit.

Once data is cached at the TAP, it can potentially be served to consumers, but consumers need to be first notified about the newly published data. This notification can be sent even before the server has a copy of the data, so consumers can retrieve it as soon as possible. Sending the notification is the responsibility of the application and the precise type of notification used for this purpose is highly application dependent. For example, for photo uploads an OSN application notification could consist of a small thumbnail, a description, and information regarding the ADU, such as the ADU_ID and the TAP where it is cached.

Applications can manage these notifications in various ways (e.g., centralized, distributed, etc – see Figure 5.5). The simplest strategy is to manage it through a central server, similar to how it is done today by OSN providers like FB. The server keeps track of the meta-data for the new content, which includes the ADU_ID and caching information that is included in the notification, and notifies the user's friends whenever they check their newsfeeds/OSN account. These applications notifications can also be managed in a distributed manner with the publisher application directly notifying potential consumers. This may require some form of filtering so that only those friends are notified who are likely to be interested in the content. Finally, we can imagine a hybrid setting which allows both distribution modes – some consumers (e.g., close friends or friends who are online at that time) can be notified directly while others can be notified later through the central server.

An orthogonal but related point is whether the central server also maintains a copy of the data – this again depends on the role of the TAP in the personal CDN (see Figure 5.5); If the TAPs function purely as a cache, the ADU also needs to be transferred to the server. In this case the transfer layer establishes a segment between the TAP and server and transfers the ADU. Alternatively, if the personal CDN is the primary source of content then the server may not need the content itself, except maybe for backup purposes. In the latter case we do need to transfer the ADU to the server, but it may be possible to delay it until a time when the server load is low, reducing the peak load the server needs to support.

Also, depending on the role of the TAP/server, the *delivery semantics* may differ. Vigilante supports three options: i) ACK from the server only, ii) ACK from the TAP only, and iii) ACK from both. These semantics ensure correct application behaviour under different content distribution scenarios. For example, applications may make a decision of whether to discard some data based on whether it has been reliably received by the server.

Finally, the application can decide to *pre-load* other TAPs with the newly published ADU. Again, the same infrastructure support and API provided by Tapa is used for this purpose. From the perspective of Tapa this is just another transfer, possibly over a slow segment. The reasons for pre-loading other TAPs could be fault-tolerance or performance benefits. We discuss how OSN applications can make this decision in Section 5.4.5. Note that this application support or the application support required for sending notifications can be implemented either on the client or on the TAP. Implementing it on the TAP has certain benefits as a TAP is always available unlike a client. However, it increases the complexity on the TAP which may not be desirable.

1. Applications make a **get** call
(includes ADU_ID and **hints**)
2. Transfer layer retrieves ADU from the best hint
3. Caches it and serves it to the application
4. Hints are updated

Figure 5.7: Basic steps in retrieving content.

5.4.3 Content Retrieval

Figure 5.7 list the main steps in retrieving a piece of content. The applications use the `get` API to specify the ADU_ID along with passing some *hints* on how to retrieve the ADU. Hints are associated with an ADU and specify nodes/TAPs that are likely sources for retrieving the content. Hints can be managed in a completely application agnostic way, at the level of Tapa, or can involve application help as well.

The transfer layer can manage these hints in an application independent manner by keeping track of cached ADUs and exchanging this information with other peers in the personal CDN. At the start, when an ADU is published, it is cached at the publisher's TAP as well as at other TAPs as part of pre-loading. These TAPs serve as initial hints for the ADU. However, new TAPs can retrieve the ADU and cache it, or previous TAPs can evict the ADU from their caches, so we need a mechanism of *updating the hints*. This can be viewed as the control plane of the CDN which allows the CDN to keep track of content availability at different nodes.

The control plane can be implemented in a distributed manner similar to how routing protocols work in today's Internet. So TAPs can periodically exchange content availability information amongst each other; this information exchange later enables a TAP to retrieve the content from the best possible source as it knows which other TAPs have the ADU in their cache. For example, if all TAPs within a social community exchange ADU availability

information amongst them, which is feasible given the small size of a social community, then whenever a TAP gets an ADU request it can route to the nearest/best TAP. This approach has the advantage that it reduces or eliminates the need to have a central server that is notified of all changes in the cache contents on the TAP.

Alternatively, we can have a centralized solution where every TAP reports its cached ADUs to the server who then updates the hint list based on this information. Naturally, the TAPs also need to report to the server when they evict ADUs from their cache. Our design allows both options but we have only implemented the latter in our current prototype of Vigilante.

OSN Applications can also *help* by providing hints based on social links and preferences of the user. For example, if a user wants to download Alice's image, the application may add Julie's TAP as a hint based on the information that Julie is a good friend of Alice and hence likely to have her image.

Once the transfer layer of TAP has all the hints on likely sources for retrieving the ADU, it has to select the best option from this list. Towards this end, application can specify its preferences by giving a relative ordering of hints, or the the decision could also be based on purely performance metrics (e.g., throughput, latency) as is the case in our prototype implementation.

Finally, once the ADU is retrieved, it is served to the client application. It is also cached at the TAP and the publisher/server is notified so that hint list corresponding to the ADU can be updated.

5.4.4 API Requirements

Vigilante also uses Tapa's put/get API earlier presented in §2.3 but instead of using the default semantics, it needs to use different semantics for both session and transfer layers. It needs to provide applications control over how data should be *delivered* (i.e., fast vs. slow segment)– this changes the default semantics of the transfer layer. Moreover, Vigilante also needs more control over how data is *acknowledged* by the different entities involved in the transfer. This is part of the session layer semantics associated with reliability. In Chapter 6, we discuss how the Tapa API allows applications to easily choose these different semantics.

5.4.5 Application Support

Vigilante also needs minimal application support on top of Tapa to provide greater reliability and performance. This support can be built into the client application or can be ported in the form of an application plug-in which runs on the TAP as a service. The main benefit

of having this support in the form of a service is the "always-up" nature of the TAPs. Application support mainly comes in two forms: i) creating multiple sources of data by pre-loading TAPs with the content and ii) application level hint generation in order to eliminate/reduce the role of the server.

We expect applications to use social network information in order to make these decisions. This can be based on some learning model where a user's access patterns/interests are learnt over time. Alternative, users can themselves provide information about their interests and preferences. For example, OSN applications, like FB, already provide users with options where they can specify their family, close friends, the kind of feeds that interest them, etc. This information can be used to make informed decisions on who is likely to access the data or who is likely to have the data that a user may be looking for. This can help make decisions with regards to pre-loading TAPs as well as in generating suitable hints.

Applications can also make these decisions from a performance perspective based on the type of data and its importance. For example, a wedding photo of a user may be pre-loaded to more TAPs because it is likely to be viewed more compared to an ordinary photo. Also, the choice of TAPs that are pre-loaded can be based on their usefulness in distributing content to other users. For example, a TAP with higher network bandwidth can be a good candidate for pre-loading. Similarly, if a user has a significant number of friends in another location, one or more TAPs in that region may be pre-loaded purely for performance reasons.

5.5 Implementation

We have implemented Vigilante which includes additions to the basic prototype as well as a social networking application that uses Vigilante for distributing photos amongst OSN users. The prototype is implemented in C++ and runs on Linux.

5.5.1 Extending Tapa

Vigilante extends the basic Tapa implementation discussed in Chapter 3 (Swift) in two aspects:

- **TAP Discovery:** Unlike Swift, where TAP discovery was based on lower layer information, in Vigilante, applications guide the TAP discovery process. Specifically, applications provide information about the TAPs of the user's friends to the local

TAP as a list of TAP identifiers. Once the TAPs are discovered, the subsequent steps of establishing segment or conduct probing remain the same.

- **Using Hints:** In previous systems we used a single hint, so there was not an issue of how the transfer layer should select a particular node/TAP from a list of hints. In Vigilante, a list of hints is usually associated with an ADU, so selecting the best hint from this list is an important decision. Vigilante uses latency as the primary metric to pick the best hint from the list. Based on offline RTT measurements, TAPs divide the hints into different clusters and within each cluster a TAP is randomly picked. The system also maintains a history of the performance delivered by the TAPs and uses this information to black-list/favor some TAPs over others. Also, because TAPs could be down or data may not be available at the TAP, the transfer layer moves on to alternate options amongst the hints, if the best hint does not work (signaled by a failure or a timeout).

5.5.2 Photo Sharing Application

The Vigilante implementation also includes a server, client and an application plug-in for a photo sharing social network.

The **server** maintains a list of user's friends along with their TAPs. Once the user connects, the server sends the news feeds. It receives notifications about new uploads from TAPs and adds them as news feeds. The feed consists of a simple message (e.g., "Alice has uploaded a new image") along with the ADU id and the hints information. The server also receives notifications about new TAPs that cache the photo and adds them as hints for subsequent feeds that are sent to clients.

The **client** allows the users to download and upload photos. It allows the user to connect to the server and get the news feeds. Once the feeds are received, the user can decide to download a photo corresponding to a feed. The user can also upload an image.

Like the server, the **application plug-in** also has information about the user's friends and their TAPs. Its basic job is to pre-load other TAPs for performance/fault-tolerance purpose. It pre-loads those TAPs that can aid in distributing content. For example, based on its performance history it selects the best k (a configurable parameter) TAPs to pre-load. Similarly, if the user's friends are geographically dispersed in clusters, it selects a TAP in every cluster to ensure that users in that cluster can get content from a nearby node. This plugin can be used at either the client or the TAP.

5.6 Evaluation

We have conducted an evaluation of Vigilante on the PlanetLab testbed. We quantify the benefits of the key aspects of our design by conducting a variety of experiments. Specifically, we show that on-demand retrieval of photos using a personal CDN provides good performance for most workloads irrespective of client locations. To this end, the response times achieved with Vigilante are lower than the best case performance of FB photo service. We also highlight how pre-loading some TAPs is sufficient to handle flash crowds. Finally, we show that performance degradation is minimal under scenarios where there is a cache miss at the TAP or the TAP is down.

5.6.1 Experimental Setup and Scenarios

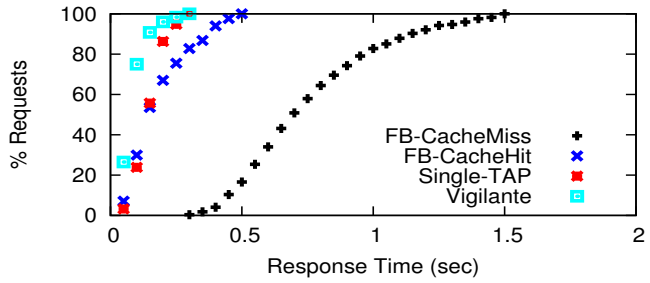
We use around 150 nodes on PlanetLab to form a personal CDN. We run the client application as well as the TAP on the same node. We focus on the performance of photo downloads and consider four schemes/scenarios in this regard: i) Vigilante: This refers to the implementation described in the previous section but without any pre-loading; ii) Single-TAP: Only the publisher's TAP serves the content. We test this scenario by disabling updating of hints; iii) FB-CacheMiss: This refers to downloading the photo from the root photo server of FB. This is the worst case performance with using FB. We follow the same technique we used in Section 5.2 to generate a cache miss; and iv) FB-CacheHit: This refers to the case when the photo is served from the nearest Akamai CDN. This corresponds to the best case for downloading a photo from FB.

In all our experiments, a node on the US East Coast acts as a publisher while all the other PlanetLab nodes are consumers. We divide the consumers into three categories based on their geographic distance from the publisher: i) Same Coast: All nodes with an RTT of less than 40 ms to the publisher are part of this group. ii) Other Coast: All nodes with an RTT of 40 to 100 ms to the publisher are part of this group and iii) Outside US: All nodes with an RTT of more than 100ms are part of this final group.

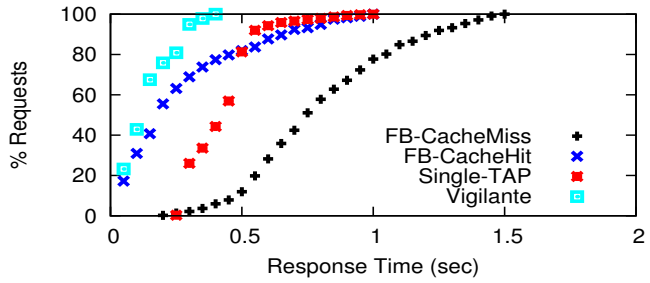
5.6.2 Performance Results

We broadly divide our experimental results into three categories: i) Performance under normal traffic load, ii) Performance under a flash crowd scenario, and iii) performance under scenarios where hint turns out to be *bad* – either because the TAP is down or because content is no longer in TAP's cache.

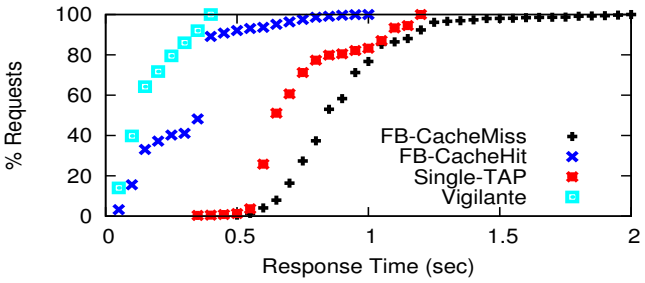
Normal Traffic Load: We first quantify the performance of the four schemes under a



(a) Same Coast



(b) Other Coast



(c) Outside US

Figure 5.8: Performance comparison of various schemes with different geographic location of friends. Vigilante provides low response times irrespective of location.

non-bursty traffic load scenario. We generate a schedule where the consumers make requests to download a 80kB image in a sequential order at roughly 1 photo request per second. This schedule is repeated 10 times and the same process is followed for all four schemes.

Figure 5.6 shows the CDF of the download time of photo requests for all four schemes. We separately plot the results for each geographic category of consumers in order to highlight the key differences that arise due to the location of the consumer.

Figure 5.8(a) shows the results for consumers who are on the US East coast i.e., on the same coast as the publisher TAP. In this case, the performance of FB-CacheMiss is signifi-

Single-TAP	Vigilante	Vigilante-PreLoading
9.39sec	5.6sec	910ms

Table 5.5: Average Time to download a photo in a flash crowd scenario. Pre-loading ensures that there are enough TAPs to handle the load of a flash crowd.

cantly worse compared to the other three schemes. Vigilante provides the fastest download time as nodes are able to locate a nearby copy of the photo. The results show that even the Single-TAP approach provides performance comparable to the best case performance of FB (FB-CacheHit). This shows that if consumers are located close to to the publisher then even the Single-TAP approach is feasible.

As we move to users located on the other coast, we observe that the Single-TAP approach no longer provides the same level of performance, as nodes have to retrieve the content from the publisher, who is located on the other coast. As shown in (Figure 5.8(b)), its mean download time is almost double that of Vigilante and FB-CacheHit. Again, FB-CacheMiss performs significantly worse than the other three schemes while Vigilante results in the fastest download times.

Finally, Figure 5.8(c) shows the results for consumers who are located outside the US. Performance is poor for both Single-TAP and FB-CacheMiss as the consumer has to retrieve the content US, which adds considerable latency. On the other hand, both Vigilante and FB-CacheHit are able to find a nearby cached copy. However, we observed that non-US sites, in general, had a higher latency to the nearest Akamai/Facebook CDN compared to US sites and therefore Vigilante provides greater benefits for such consumers.

Flash Crowd Scenario: We also looked at a scenario where all consumers make download requests within a short span of time, essentially creating a flash crowd scenario. To this end, we created a schedule where all consumers made request to download a 1MB photo within an interval of 30 seconds. This roughly corresponds to 5 photo requests per second.

Table 5.5 shows average download performance achieved by Single-TAP, Vigilante and Vigilante-PreLoading which refers to pre-loading 10% of the TAPs with the photo). We do not consider the Facebook schemes because FB scales down and compresses images, resulting in a size that is much smaller than 1MB. Also, note that because of the large image size, RTT to the publisher does not play a major part. Instead it is the bandwidth which becomes the bottleneck as the load on the TAP increases. As a result, we do not make a distinction amongst nodes based on their geographical location. Also, as PlanetLab nodes have higher upload bandwidth compared to typical DSL/Cable upload bandwidth, the results of this experiment should be interpreted in a relative context i.e., to understand

the additional benefits gained by pre-loading TAPs.

As the results show, the Single-TAP approach does not scale well and download times are close to 10 seconds. Performance with Vigilante is better because of its ability to distribute load. However, it takes time in creating more copies and therefore performance is still poor. Vigilante-PreLoading addresses this concern by creating several copies up-front, allowing multiple TAPs to distribute the photo right from the start. This ensures that all consumers get the photo within 1 second.

Bad Hints: Vigilante keeps an up-to-date record of the cached copies at every TAP, so cache misses are rare. Similarly, TAPs are not expected to be down for long and in case they are down, they are not added to the hint list. So even though such events are rare, we investigated the performance impact of cases where the hint turns out be bad.

We observed that even with a 10-20% uniform probability that a given hint is bad, the impact on performance was negligible. This is because in many cases such bad hints were not the best hint anyway. In cases where they were selected, the added penalty was between *200-500 milliseconds*. In such cases, the consumer downloaded the photo from the next best hint.

5.7 Discussion

We describe how our design supports other use cases that we did not implement or evaluate in our current prototype.

Mobile Access: Many users access OSN applications from their mobile devices. Such users can be broadly put into two categories. The first category includes users who use their mobile devices from their homes and offices or use it while traveling within their home city. These users are still close to their personal CDN so they get benefits of retrieving content from nearby sources. Furthermore, in some scenarios they may also get other benefits in the form of energy savings by offloading processing or download responsibility to the TAP [49, 55].

The second category includes those users who are traveling and are not close to any node of their personal CDN. These users will have to retrieve the content from the nearest available TAP which in many cases will still be better compared to going to a remote central server. Also, if the mobile user gets access to *any* TAP then this new TAP can also join the user's personal CDN. In that case even though it may not have any nearby TAP node to retrieve data, but it can gain benefits by pre-fetching data from far-off TAPs.

Dynamic Data: Although Vigilante focuses on distribution of static data (photos),

Tapa is flexible enough to handle dynamic data as well, such as chat messages or status updates. As this type of data is not suitable for caching or a pull based model, applications will use Tapa's put API to directly push it to the consumers. Application can also specify a group address/identifier as the destination and the TAP can send the content to all consumers who may be part of this group. For example, user can specify an identifier corresponding to friends who should get her status update. She can specify this identifier as the destination address and TAP can multi-cast the message to all users in this group.

Consistency: A related issue is how to provide consistency in case of updates to a copy of the data. Applications can impose different consistency models such as strict consistency or eventual consistency. Given the nature of social networking applications, we expect the latter option to be more popular as it provides better performance. In this case, different replicas could be out-of-sync with each other, and an important consideration is the order in which different replicas are updated. In this context, we can potentially use social networking information to choose the order in which replicas are updated. So close friends and relatives get the update first, followed by other social contacts. Potential consumers can also use this information to retrieve content from a a replica that is more likely to contain an up-to-date copy of the data.

5.8 Related Work

We discuss work that is most relevant to the concepts presented in this chapter.

Online Social Networking: Several measurement studies have looked at traditional CDNs [76, 77, 112] as well as the features of OSNs [36, 41, 104, 115, 117]. Our measurement study is unique because it focuses on *user generated content* being served by traditional CDNs and employs a unique method to conduct controlled experiments. Also, to the best of our knowledge, we are the first to measure the performance of FB's photo service from a client perspective, offering unique insights into the impact of caching on performance.

Several projects have also looked at the problem of *scaling OSNs*. These proposals provide evidence of locality in content access for OSN applications and therefore provide motivation behind the use of personal CDNs. For example, Puyon et al. [95] propose a data partitioning and replication scheme that localizes content belonging to a social community to a single server within a data center or cloud. In our case, such partitioning and replication is implicit as TAPs only cache data belonging to the social community of the user. Another recent proposal by Wittie et al. [118] provides further evidence of locality in OSN content access. Their results are based on actual OSN data and on a simulation based analysis of how proxy based caching can reduce load on the server for dynamic content. We propose a

more general solution based around personal CDNs and provide a design, implementation, and evaluation of our approach.

P2P Systems: Our proposal is an example of a P2P system, but it differs from traditional P2P systems [8, 10] in two ways. First, it is optimized for social networking applications. Specifically, it utilizes knowledge about access patterns to optimize response time for content retrieval. Second, it operates amongst network elements (TAPs) as well as end-points rather than just focusing on end-points. These two types of devices have different properties (e.g. up-time) and they also play a different role in the system (e.g. providing storage with various semantics versus full application execution). These differences result in a unique set of challenges and design decisions. For example, our API and delivery semantics explicitly account for different roles played by the TAP or by end-points.

The Nano Data Center project [7] is a P2P project that shares our vision of leveraging dedicated storage devices at the network edge. However, we use the TAPs to support personal CDNs, while their goal is to provide more traditional general P2P data sharing. Also, we have designed and implemented a complete content distribution system that leverages these personal CDNs.

Publish/Subscribe systems: A key feature of traditional publish/subscribe systems is that they provide data access through content filtering, e.g. based on keywords, attributes, or other application specific criteria [43]. In contrast, we use a content-centric networking approach that uses unique content names – self-certifying identifiers in our case. These two content retrieval models are quite different, resulting in different challenges and APIs. For example, unlike traditional pub/sub systems, Tapa exposes TAPs to the applications through a flexible API and supports different delivery semantics between end-points, TAPs and the server.

5.9 Summary

This chapter makes a case for revisiting content distribution in large scale OSNs. We argue that OSN content distribution fundamentally differs from the traditional web based content distribution, both in terms of how content is generated and accessed. These unique characteristics are best addressed by using a personal CDN of TAPs within a social community. We believe that Tapa can provide the right platform for applications to fully leverage the benefits of a personal CDN. Finally, our prototype implementation and evaluation of Vigilante attest to the potential of personalized CDNs in providing high performance to OSN applications.

Chapter 6

API and Semantics

In this chapter, we present the Tapa API that allows applications to control various semantics associated with end-to-end communication. It supports traditional semantics (e.g., tcp style end-to-end reliability) as well as new semantics that emerge as a result of leveraging intermediate services (e.g., delegation). We show how this API can be used to control the semantics associated with the three services presented in earlier chapters (Swift, Catnap and Vigilante) – both as services running independently as well as when they are used together. We also present the design and implementation of a session layer that implements the various semantics supported by Tapa's API.

The chapter is organized as follows: we provide a brief overview of the semantics supported in today's Internet (§6.1) followed by an overview of the role of different entities towards supporting diverse semantics in Tapa (§6.2). In §6.3 and §6.4 we describe the design of Tapa's API and session layer, respectively. This is followed by a case study (§6.5) that highlights how the API and session layer support the role of intermediate services, including the three services presented in earlier chapters. Finally, we present related work and summarize the key findings of this chapter.

6.1 Background: Communication Semantics in Today's Internet

In this section, we discuss the semantics supported in today's Internet as this helps in later understanding how Tapa offers different and richer semantics.

The semantics offered by today's Internet are determined by the interaction of protocols like IP, TCP, and SSL. IP offers a *best effort* data delivery service i.e., it offers no guarantees with respect to reliability or ordering of data. Furthermore, its service is *transparent*, so it

does not read or modify the data. These weak semantics have two important implications: i) end-points are fully responsible for supporting all the functionality required to implement application semantics (e.g., reliability, ordering, etc), and ii) dealing with failure of various elements (i.e, routers) is simplified — unless there is a total partition of the network, the failure remains transparent to end-points.

TCP provides a reliable, ordered delivery of data between two end-points. Reliability is implemented via acknowledgements and timeouts. A TCP sender holds on to the data until it is reliably received by the other end-point. TCP end-points carry hard state, so a failure of an end-point, even if it is a reboot, disrupts the communication operation and requires application to undertake recovery. Finally, TCP conflates reliability with congestion control, as acknowledgements are used for both error control as well as for signalling that the path is not congested. As a result, it is difficult to implement different semantics, such as partial reliability, because that also impacts the desired congestion control behaviour.

SSL supports many security functions, but those that are of interest to us include confidentiality and data integrity. Confidentiality is provided through encryption while data integrity is ensured through the use of message authentication codes. Standard cryptography techniques are used to implement these functions. SSL works on top of TCP and therefore inherits TCP's reliability and ordering semantics. SSL's failure semantics are also similar to those of TCP.

Both SSL and TCP are used by the applications through the socket interface. To use the socket API, applications must know the address of the other end-point and also choose a specific interface and protocol for the communication operation. So if the other end-point is unavailable the error is exposed to the application. This highlights the host-based nature of the API as well as how it may be too low level for the tasks that we need to do in Tapa (e.g., opportunistic content retrieval from any source, use of services etc). However, it is important to note that these protocols and API were designed with host based communication and a simple network service model in mind. So even though the semantics offered by these protocols have been useful in the past, there is now a need to offer richer semantics given the trend towards data and service oriented networking.

6.2 Semantics in Tapa – Overview

In contrast to today's Internet, Tapa offers very different communication semantics through a new, higher level API. Tapa not only makes it easier to support different end-to-end semantics, but also has an explicit role of in-network services that can perform various actions on the data at the granularity of an ADU. This brings a new dimension to end-to-

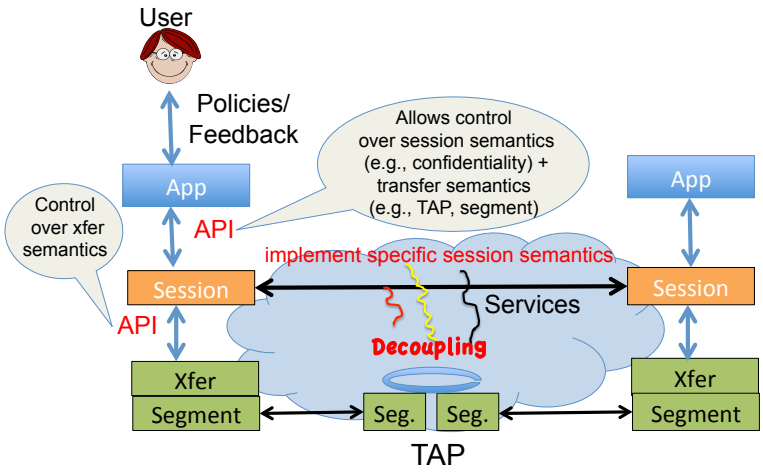


Figure 6.1: Overview of how different entities play their respective roles in supporting diverse semantics in Tapa.

end semantics, which is captured in Figure 6.1 that shows the roles that different entities play in Tapa to support diverse semantics.

Applications specify the semantics via the Tapa API to the session layer. Some of these semantics relate to session layer functions (e.g., reliability, confidentiality) while others may relate to transfer layer functions (e.g,. fast vs. slow segment). Each layer implements the specified semantics that are its responsibility and reports back to its higher layer if this is not possible. The transfer semantics are in turn conveyed to the transfer layer via the session-transfer layer API.

Tapa's session layer currently supports four functions: reliability, confidentiality, data-integrity, and ordering. Semantics associated with these functions apply to both a traditional end-to-end scenario as well as when we have in-network services that impact these functions in some way. Applications specify the desired semantics associated with these functions and the session layer implements them through a suitable protocol that runs between the end-points and optionally in-network services as well. For example, an application can specify its interest in "complete confidentiality" and the session protocol would implement this by doing some form of end-to-end encryption, similar to how SSL works today. In contrast, if the application desires partial confidentiality, implying that an intermediate

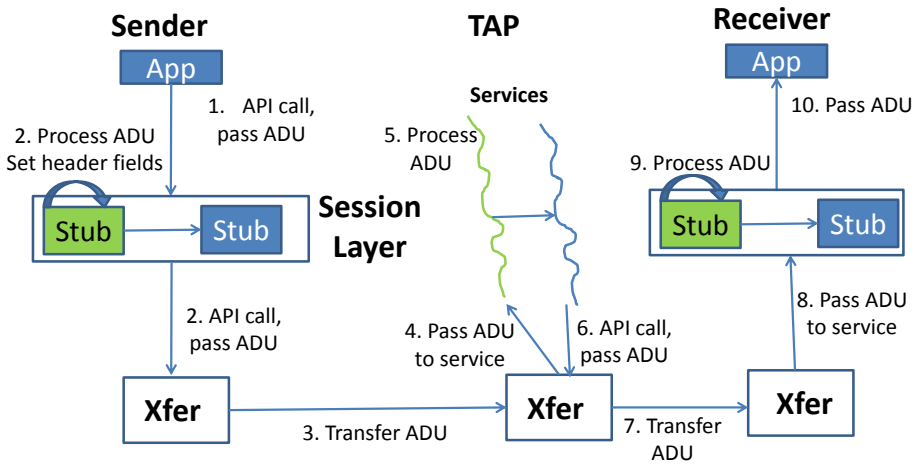


Figure 6.2: Figure shows the steps followed in the data plane for transferring an ADU from a sender to a receiver with two intermediary services. Each service has its stub that runs at end-points.

service may also observe the content, then a per-segment encryption strategy is adapted.

The current design of the session protocol only supports a restrictive set of semantics for the above functions, but future work could expand on this and provide richer semantics using the same API. The current design assumes that services are already composed and are running on the same TAP. Future work could explore how richer semantics could be provided by leveraging prior work on composing services on-the-fly [64, 84].

Figure 6.2 shows the different steps that are followed in a transfer involving a sender, two network services, and a receiver. As shown in the figure, the semantics expressed through the API are converted into header options by the service stubs at end-points. Of course, stubs are only required for those services that also require support at end-points only (e.g., encryption services). Other services (e.g., caching) may not require any stub at end-points and all their code will only run at the TAP.

The service stubs communicate with each other using the same API through which they receive ADU from an application i.e., each service stub is indifferent to whether the ADU it receives is sent by another service stub or the application. The only difference is whether the service header needs to be inserted, which is mostly the case with stubs at end-points, or whether the header is already present and just needs to be processed (based on the header values) by the service stubs. We could also have a single piece of code for service stubs, which understands the semantics of common services, as is the case with our implementation of the session protocol.

A session protocol header is attached to the ADU that contains all the semantics infor-

mation that is required by the in-network services to process the ADU. At the TAP, each service does its own processing based on the header values and then forwards the ADU to the next service, using the same interface that is used by the transfer layer to hand over the ADU. Future work should look at richer interfaces that are more suitable for inter-service communication. In our current design, services are processed in a static order (if they are required) at both the end-points and the TAP, but we can imagine explicitly using service identifiers and representing them as DAGs (similar to XIA [23]), so that the order in which services should process the ADU is explicitly indicated in the header. Note that the order in which services are used may impact the semantics of the transfer – for example, caching and decryption of ADUs are two functions that would result in different semantics depending on their execution order. Once the services process the ADU, they send it to the other end, where a stub code processes them and may also deliver it to the application. If no in-network service is involved then the transfer layer of the TAP directly forwards the ADU to the transfer layer of the other end-point instead of forwarding it to a higher level service.

During the whole communication operation, the session protocol also deals with failures of various sorts, such as a failure/reboot of a TAP that was running a session service. It uses standard failure detection and recovery schemes, and reports the failures to the application, if recovery is not possible. We elaborate on this as part of our discussion on reliaibility/error control in §6.4.1.

As implicitly highlighted in the figure, it is assumed that services know how to communicate with each other. For services running on the same TAP, the same interface that is used to transfer an ADU from the transfer layer upwards is used to transfer ADUs between services. For services running on different TAPs, they must first establish a segment between the TAPs (in the control plane) and only then can they communicate. An example of this would be Swift where a new TAP may need to retrieve ADUs from an old TAP. In such cases, the new TAP would first establish a segment and then the two services can communicate with each other.

Finally, an important point to note is that the session protocol runs over the transfer and segment layers of Tapa; these layers provide the necessary infrastructure support, as well as a separation of concerns, which allows the session layer to focus on end-to-end semantics. For example, establishing segments or TAP/service discovery is supported by these layers, so it is relatively easy to implement the control and data planes of the session protocol. This aspect is further highlighted when we discuss various functions of session protocol in §6.4.

6.3 API

We now describe the API to Tapa's session layer, which captures applications' requirements as well as provides suitable feedback to them to recover from failures. This is a more precise version of the API earlier presented in §2.3.

Our API's design is motivated by two requirements: i) undertaking common tasks (e.g., retrieving some content) should be *simple*, and ii) applications should have *control* over different communication semantics that are possible in Tapa (e.g., use of a specific TAP, service or segment).

We propose a simple API that provides a higher level abstraction compared to the socket API. A higher level API simplifies routine tasks (e.g., retrieving content from *any* source rather than a specific host) and failure recovery, but potentially comes at the cost of losing control over lower level tasks that applications may need to perform in certain cases. To address the latter issue, our basic API also allows extensions that provides applications with full control over the communication semantics.

We first elaborate on the two key requirements that motivate the design of Tapa's API, followed by the API design and its example use-cases.

6.3.1 Requirements

1. Routine tasks should be simple. Our desire for simplicity implies that common tasks that applications need to perform should be straightforward. As most applications today are data oriented in nature, the common tasks that such applications need to undertake are *data retrieval* and *data publishing*. The need to support these two data oriented tasks leads us to three sub requirements that the API should support:

- It should provide *spatial decoupling*: applications should not be required to specify the other end-point. This allows the lower layers to choose *any* host from which they can retrieve data. This is the most important feature as it not only impacts the choice of the other end-point for retrieving data, but also has implication on how communication failures are handled. As the communication is no longer tied to a specific end-point, if the host serving the ADU fails, then the receiver can choose some other host and retrieve the ADU, without letting the application know about the failure.

- It should provide *temporal decoupling*: the publisher of the data need not be present at the time when data is consumed.

Function	Description
`get(ADU, semantics as list of (type, value) pairs`	Call used to pull an ADU
`put(ADU, semantics as list of (type, value) pairs`	Call used to push/publish an ADU

Table 6.1: Tapa interfaces to the Application Layers. Interface between transfer and session uses a similar API but with limited options for semantics.

- It should hide the lower level details associated with the data transfer. For example, applications need not specify the transport/transfer mechanisms that should be used to retrieve some content. Hiding such details behind a suitable abstraction allows applications to benefit from innovations in transfer mechanisms without requiring applications to be re-written.

2. Adequate Control for Applications. The second requirement relates to providing adequate control knobs to applications so that they can make full use of the various Tapa features, including controlling the semantics of various in-network services. For example, the application may want control over the services or hosts that are part of the communication or may want to use a specific transfer protocol or interface. Similarly, they may want different semantics associated with reliability or confidentiality. All these reasons call for an API that provides enough control to applications. However, it is difficult to have an exhaustive list of all knobs, so it may be desirable to have an API that is *extensible* and allows new knobs to be introduced incrementally.

6.3.2 Design

Our API's design meets the requirements identifed in the previous section through a single API that has a *basic* use that works for most routine tasks and *extensions* that can be invoked to provide fine-grained control to the applications. The API allows applications two basic operations on the ADU: i) they can retrieve an ADU using the `get` call and ii) they can publish an ADU using the `put` call. As shown in Table 6.1, the API takes in an ADU, which is the mandatory part and an *optional* list of (key, value) pairs that are used to invoke the extensions to the API.

The simple `put/get` API (without any optional semantics) is inspired by earlier work on data oriented systems (e.g., DOT [113]). The main difference is that Tapa's notion of ADUs is more general than DOT's notion of chunks. We first describe the working of the API in the basic cases, using the default semantics. Later we will discuss how richer semantics could be supported through this API with the use of key/value extensions.

- `get`

  This interface allows applications to retrieve an ADU based on its identifier. In addition, the ADU meta-data also contains a *hint* on how to get this ADU. This hint could correspond to a server that may have the data or a nearby TAP that may have the ADU in its cache.

  There are many aspects that are *implicit* in the get call. First, the application is showing its intent to get an ADU irrespective of which host is used to retrieve it. So lower layers are free to choose any host. Second, it is not specifying any low level details associated with the actual transfer, such as any specific transport protocol or interface; in fact, it is not even specifying that the network must be used, so the ADU could be served from the local storage, if this is possible. Third, the application is choosing *default* semantics associated with functions like reliability and confidentiality. For example, in our implementation, our defaults are fully reliable transfer with no confidentiality. We later discuss how applications can choose different semantics.

  The `get` call returns the actual ADU or a failure notification if the ADU cannot be retrieved. Note that a higher level API means that applications are likely to observe fewer failures because lower layers have more options for failure recovery. For example, even if one host goes down, the ADU could be retrieved from another host. Similarly, even if the device is disconnected from the Internet, it may still be possible to retrieve the ADU through some other mechanism (e.g., using some nearby cache).

- `put`

  This is used to publish an ADU to the default storage, which could be local or some remote service. Applications can use this API to publish their data *once* and a generic storage/transfer service can later take care of serving it to future client requests. This enables temporal decoupling between the publisher and the consumer. Again, there are several aspects that are implicit in this call, such as the use of a default storage service and the requirement to have full reliability. We will later see how this API can be extended to support different semantics.

  The put call can also be used to *push* the data to a specific service that may be running on some remote machine. The latter use is needed to exchange control information regarding ADU ids and to also enable more interactive communication. However, the current design does not support a session oriented, socket-like communication and the application has to specify the other end-point identifier every time it wants to send an ADU. This may be somewhat inconvenient for a session oriented interactive application so future work should explore simple extensions that could allow the use

of a socket between two application services; so we could establish a socket, using calls similar to listen, connect, and accept, and this socket (i.e., its descriptor) could later be passed to the put call for pushing an ADU.

To illustrate the basic use of the API, we now provide a simple example that involves a client getting a file comprising of 2 ADUs from a server.

```
*Server Code*

put(ADU_1); //putting ADU 1 into local storage
put(ADU_2); //putting ADU 2 into local storage
/*server can quit now because the ADUs can now be served
by the local storage/cache service */

*Client Code*
/* Assuming client already has the ADU ids --- ADU_1.id and ADU_2.id */

/* specifying server name as the hint within the ADU*/
ADU_1.hint = "server_name";
ADU_2.hint = "server_name";

/* getting the ADUs*/
get(ADU_1);
get(ADU_2);
```

6.3.3 Extended Use of the API

To address the second requirement of providing more control to the application, the API also supports extensions to the basic call for put/get. This is achieved by specifying an optional `(type, value)` pairs of additional requirements in the put/get call. Table 6.2 lists the various types supported in the current design. The rationale behind some of these options have already been discussed before. For example, the host type could be used to indicate a specific host to which the ADU should be delivered. Rationale behind the other options as well as how they are implemented is discussed in the next section.

Type	Value
host	name - e.g., “skyblue.aura.cs.cmu.edu”
tap	name - e.g., “home-tap”
service	Name – e.g., “vigilante” “catnap-batch”
segment	Slow, fast
reliability	Best effort or full reliability
delegation	Yes/no
ordering	Yes/no
confidentiality	Full, partial, no
integrity	E2e, per-segment
discard_policy	E2e, per-segment
app_notify	Self, intermediary, dest

Table 6.2: Semantics type/value options currently supported in the system.

6.4 Session Semantics - Design and Implementation

The API described in the previous section allows applications to choose their desired semantics and it is the responsibility of the session layer to implement these semantics between end-points and intermediaries. We have designed and implemented a session layer that supports different semantics associated with four key functions: reliability, confidentiality, integrity, and ordering. This includes end-to-end semantics as well as new semantics that emerge if in-network services change the semantics associated with the above functions in some way.

We have implemented the session layer as a user library; it exposes the `put/get` interface to the applications. The library communicates with the transfer layer using a wrapper function that communicates with the transfer layer through a UNIX socket. Our focus is on the data plane, an important component of which is how we specify different semantics as part of the session layer header of an ADU, enabling different parties to take appropriate actions. We make simplifying assumptions regarding the control plane, such as the availability of keys between different parties. We use the `openssl` library for the security related functions [9]. We note that our session layer is just one example of how common application

semantics can be implemented as a light-weight session layer; other designs, which can offer fewer or more semantics, can also be supported in Tapa.

We now describe the semantics associated with each of the four functions supported by the session layer.

6.4.1 Reliability

As the transfer and segment layers provide best effort service, the session layer has to provide full reliability if it is required by the application. Applications can choose a fully reliable data delivery (default) or a best effort delivery. We focus on the reliable delivery case as the the session layer plays a more important role in this case. We first describe the scenario where only the end-points are involved in the reliability semantics and then discuss the semantics when an intermediary service is also involved.

For a reliable service, the session layer of the sender holds on to the ADU until it is acknowledged by the other end-point. If required, the session layer has to undertake recovery in the face of different kinds of failures.

Our high level approach towards dealing with failures is simple. The session layer tries to recover from failures on its own (using timeouts and retransmissions) but if that is not possible it exposes the failure to the application, so that applications can undertake recovery as per their requirements. Today's protocols also use a similar strategy but their failure semantics are different because in today's Internet if the other host is unreachable, the failure is exposed to the application. Our API provides a higher level abstraction to the applications, so the semantics of recovery can be different as we explain next.

Many applications may not care about the end-point in some cases, so failures of hosts can be made transparent to applications. For example, in case of a simple `get` by the application, the lower layers may start retrieving the ADU from a certain host, but if that host fails, an alternate host who may have the ADU can be chosen. This process is transparent to the application as they only care about getting data irrespective of which host is actually used to retrieve it. So a failure is reported to the application only when the lower layers are unable to find the ADU from *any* host.

Another key difference in reliability semantics arise due to the presence of intermediate services. It raises two key issues. First, failures of TAPs and higher level services also need to be handled. If the TAP is not running any session or higher level service then the recovery is simpler as it only involves establishing new segments. However, if a session level service was also running on the failed TAP then we need to re-establish a new session involving a new intermediary service and the end-points.

Second, we can tolerate failures caused by temporary disconnections (e.g., due to mobility) by *delegating* data transfer responsibility to an intermediate storage service. So even if the sender disconnects after completing the transfer to the intermediary, the transfer is not affected. Of course, the disconnection may not be forced – the mobile device can disconnect voluntarily for other reasons (e.g., to save power).

The flexibility to delegate is especially useful in scenarios where there is bandwidth discrepancy between links in an end-to-end path (similar to Catnap). In such scenarios, delegation can reduce the vulnerability period during which the disconnection of the sender is critical for the end-to-end transfer. In today's Internet, the vulnerability period is the total time to complete the transfer. However, in Tapa, this vulnerability period can be reduced. For example, in order to upload a large file to a server, a mobile client can burst the data to the TAP using the fast wireless segment, disconnect, and the TAP can take over the responsibility of sending data to the server over the slow wired link. Taking over responsibility means that the TAP initiates any retransmissions if the ADU is not received by the other end-point. Of course, it also means that a TAP failure is fatal in this case as the other end-point cannot get the ADU, similar to how an end-point failure is fatal in today's Internet.

In the above scenario, applications may be interested in differentiating between when the data is received by the TAP compared to when it is received by the end-points. This information can be used by the application as it may discard the photos if they are received by the server but may like to hold on to the photos if they are only received by the intermediary. Recall that Vigilante uses such semantics to support different roles for TAPs in distributing content in OSNs. Our implementation provides the support that allows applications to choose these different options.

Implementation In order to implement the reliability semantics, the session layer maintains the following information for each ADU:

`Reliability_Type:` This refers to whether a best effort service or a full reliability service is required. In case of best effort service, the session layer of the sender sends the ADU and discards it, while if a fully reliable service is desired, it keeps the ADU in the buffer, maintains a timeout value and later initiates retransmissions if required. By default, fully reliable service is assumed but application can specify a best effort service through the API.

`Delegation:` This refers to whether the ADU transfer needs to be delegated to an intermediary. The main implication of this is that the responsibility of retransmissions is taken over by the intermediary instead of the sender. This option is also specified as part

of the session layer header so that the intermediary could take over the responsibility.

`ACK_Req:` This specifies whether an ack is expected from the intermediary service, other end-point or both. This information is also carried in the session layer header that is attached to the ADU.

`Discard_Policy:` This determines the discard policy of the session layer of the sender. For example, whether ADU should be discarded when it is received by the intermediary or when it is received by the end-point (default).

`App_Notify:` This determines the different events that the sender application may be interested in. Options include getting a notification when the ADU is received by the host session layer (default), the intermediary or the other end-point. Applications specify callbacks that are called by the session layer when such events occur. Of course, applications can specify their interest in multiple events as well.

6.4.2 Confidentiality

Tapa's use of ADUs allows applications more flexibility in implementing their confidentiality requirements as they can make fine-grained decisions on whether some data needs to be encrypted or not. Applications can do this using a single session for both encrypted and un-encrypted data rather than having to establish two different connections – one encrypted and the other unencrypted – as is the case in today's Internet.

The main benefit of Tapa, however, comes with the use of in-network services in the confidentiality semantics, allowing them to selectively look at certain data with the explicit permission of the application. This is important for both policy and performance reasons. For example, some intermediaries, like an enterprise or government, may require looking at certain types of data. We allow applications to explicitly state whether certain ADUs can be seen by intermediaries or not providing a form of "controlled transparency" [47].

Similarly, the use of intermediary in the confidentiality semantics can also result in performance benefits. One advantage is that it can reduce the burden of maintaining SSL connections for servers. For example, services like Gmail require their servers to maintain separate SSL sessions for every client and this burden could be a significant bottleneck for some services. In fact, it is common to see popular services, like Facebook, not providing SSL based communication as their default mode because of this extra burden. We can can reduce this burden if the intermediary maintains a single encrypted connection with the server and multiple clients can have their individual encrypted connections with the intermediary. Note that as the intermediary service could be provided by a third-party (e.g., an ISP), the semantics of this communication are very different from those of end-to-

end encryption as the application/user is also trusting a third party and explicitly involving it in the confidentiality semantics.

Another advantage is that it can improve the effectiveness of redundancy elimination techniques, which are ineffective if the traffic is encrypted [24]. Specifically, applications/user can explicitly involve a trusted intermediary (i.e., ISP) in confidentiality such that the intermediary can decrypt the data as it enters its network, use RE techniques inside its network, and then encrypt the data again before sending it to the destination/next ISP.

Implementation Applications can specify three options for confidentiality: i) *full*, which corresponds to end-to-end encryption i.e., no intermediary is allowed to look at the data, ii) *partial*, which allows "trusted" intermediaries to look at the data. We assume that lower layers know which intermediaries can be trusted. This function can be expanded in future implementation such that each application has its own trusted list of intermediaries, and iii) no confidentiality, which is the default. The session layer header has a field `Confidentiality_Type` that carries the chosen option for the ADU.

In order to implement these options, the session layer has to encrypt/decrypt the ADU accordingly. As noted earlier, the session layer already knows the keys corresponding to different entities. For example, the client will have one symmetric key for corresponding with the TAP and another one for communicating directly with the server. So if full confidentiality is desired, it can use the server key while if partial confidentiality is required then it can use the key for the TAP. Similarly, the TAP also maintains a key for both the client and server. Future implementations can use an SSL like protocol to generate these keys.

6.4.3 Data Integrity

In most cases applications care about verifying the integrity of the data. In today's Internet, TCP provides a weak data integrity check in the form of check-sums while SSL provides a stronger support with the use of message authentication codes.

In Tapa, we can provide data integrity through two ways. The first one is specific to the `get` mode where the receiver already has the self certifying identifier corresponding to the ADU (i.e., ADU id), which ensures that it can verify that it received the correct data. Note that we assume that the ADU ids themselves are retrieved in a secure manner, which suggests that we also need a more traditional solution, similar to SSL, which can apply to other types of data exchange as well (e.g., `put` mode).

We use message authentication codes to verify the integrity of data. In addition to

verifying the end-to-end data integrity, we also allow intermediaries to be involved in these semantics as there are scenarios where there could be legitimate reasons why an intermediary may change the bit-stream (e.g., transcoding, virus scanning proxies, etc). By explicitly involving the intermediary, applications can verify that the data was changed *only* by a legitimate party (i.e., the intermediary) and not by someone else.

Data integrity is also closely tied to confidentiality as an application may allow intermediaries to look at the encrypted data but not to modify it (e.g., government or an ISP). With flexible data integrity and confidentiality semantics, applications can choose how an intermediary can ready or modify the data.

Implementation We always provide a MAC of the header as it carries important information about the semantics associated with the ADU transfer. For the data part, the default option is end-to-end data integrity but applications can choose per-segment data integrity too. A third option is no data integrity, in which case we just use the check-sum. Our implementation uses the HMAC function supported in openssl and again we assume that the relevant keys are available.

6.4.4 Ordering

It is well known that the use of ADUs provide greater flexibility to applications who may want to process data in an out-of-order fashion.[1] Tapa's use of ADUs help the session layer in supporting different ordering semantics; these semantics determine whether the session layer presents these ADUs to the application in the same order in which they were generated by the other application end-point, or presented in the order in which they are delivered by the network. Of course, this is based on the assumption that the network does not provide any guarantee with respect to the ordering of ADUs. In fact, the use of Tapa may cause grater reordering compared to today's Internet because of Tapa's use of use of multiple segments for transferring ADUs.

Supporting in-order delivery of ADUs requires that the session layer maintain adequate buffering to store the ADUs that are received out-of-order. As an optimization, the session layer of an end-point can make use of an in-network service's resources for ordering ADUs. This is useful for resource constrained devices who may not have enough buffering to account for reordering of ADUs.

Let us consider an extreme example to illustrate this benefit. Assume that there is a resource constrained sensor node who needs to download ADUs from a server that is

[1] in fact, out-of-order processing was the primary motivation behind the application layer framing proposal [45].

located across a WAN. The sensor node can only process one ADU at a time, turning the communication into a stop-and-wait protocol, as the server can only send another ADU once the previous ADU is consumed. This causes inefficiency as delays over WAN are typically large; this inefficiency can be avoided with the help of a network service that is located close to the client (e.g., at the edge of its access network). The service buffers ADUs, orders them and sends one ADU at a time to the client. This allows the server to continue sending ADUs to the service without worrying about whether or not they are received in order by the sensor node.

Implementation Applications can choose ordered delivery or out-of-order delivery for each ADU. As ADUs are usually suitable for out-of-order processing, our default mode is out-of-order delivery, but applications can use the API to specify a different option.

In order to support ordering requirements, the session layer header associated with an ADU contains two fields: i) `OrderingReq`: this specifies whether the ADU needs to be delivered in-order or not, and ii) `ADU_Seq_No`: this helps the session layer in determining the right order, if in-order delivery is requested.

6.5 Case Studies: Semantics involving Services

With the help of several use-cases, we now describe how the API and session layer work in more complex scenarios that may involve one or more services. The use cases also highlight how the same API is used for all the three systems described in earlier chapters (Swift, Catnap, and Vigilante), both when they run independently as well as when we combine them together. For each use case, we report the API call as well as the important header values that are set in order to support the desired semantics for the call.

We first recap how the API allows use of the services presented in earlier chapters. We then describe how more than one of these services can be combined together. Finally, we discuss how we can support Vigilante++, a system that adds a security service to Vigilante.

6.5.1 Single Service Scenarios

The following API calls show Tapa's API can be used to control the three services presented earlier.

Retrieving a cached ADU from an old TAP in Swift:

```
get (ADU, tap = "old-tap")
session header = default
transfer header = hint -> old-tap
```

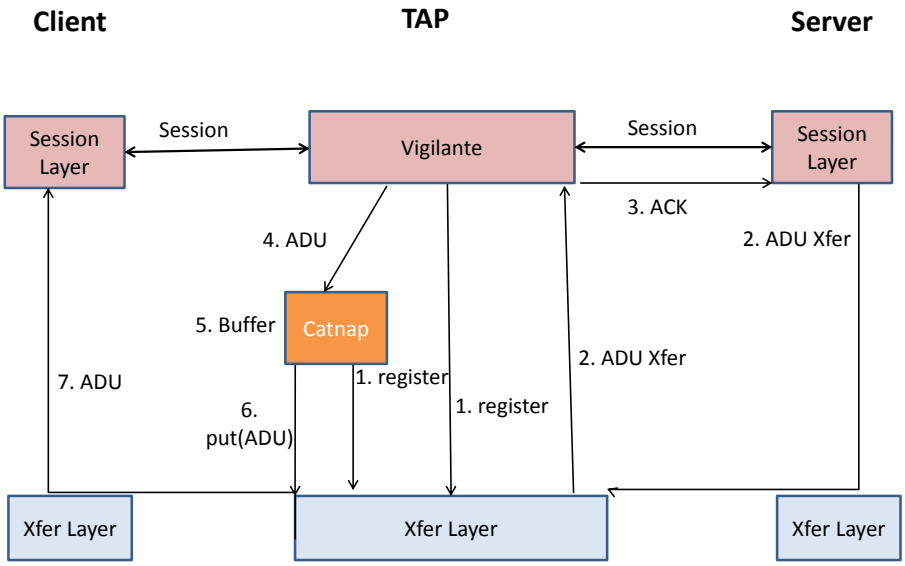


Figure 6.3: Combining the Use of Catnap Batch Mode with Delegation Semantics of Vigilante. Due to Vigilante, session involves the TAP as well.

Retrieving an ADU using Catnap's batch service i.e., showing willingness for added delay.

```
get (ADU, service = "catnap-batch")
session header = service -> 1 (Catnap-Batch)
```

Sending an ADU to the server using the slow segment in Vigilante.

```
put (ADU, service = "vigilante-plugin", host = "server", segment = "slow")
session header = service -> 2 (vigilante)
transfer header = segment -> 1 (Slow Segment b/w TAP and server)
```

6.5.2 Combining Multiple Services

We now discuss how the systems presented in earlier chapters can be used together, describing the issues as well as how the API will be used.

Catnap and Vigilante

Catnap and Vigilante are largely complimentary in nature and can be used together. Vigilante gives benefits by improving the source selection i.e., from where the user downloads

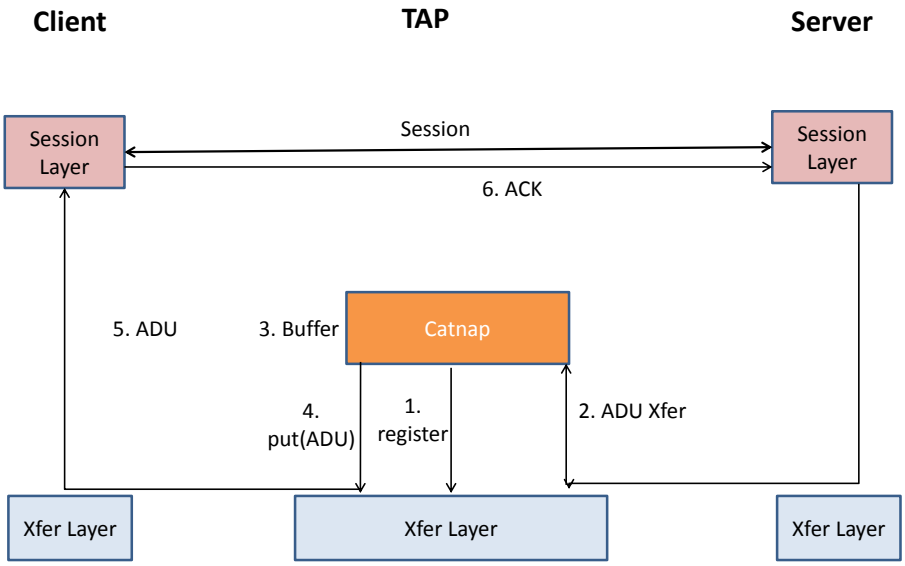


Figure 6.4: Using Catnap's Batch Mode as the only service. Session is only between end-points.

the ADU (local TAP or TAP of a nearby friend). So it can be seen as a control plane enhancement. In contrast, Catnap operates in the data plane and starts operation *once the download starts*. So these two systems are orthogonal to each other and can co-exist without impacting the benefits of each other. In fact, some aspects of Catnap (e.g., bandwidth estimation between TAP and other hosts) can be improved because TAPs can maintain a history of the performance provided by other TAPs in the social network whereas maintaining similar information for any arbitrary server raises scalability concerns. In the data plane, they can also co-exist as Vigilante's delegation semantics can work alongside Catnap. So applications can specify that they want to use Catnap's batch mode as well as the delegation feature of Vigilante.

Using Tapa's API, we can specify that any ADU that is being retrieved from a friend (Vigilante) should be downloaded using Catnap i.e., to gain energy savings at the expense of some added delay. Moreover, the other end-point should delegate the transfer responsibility to the Vigilante service.

```
get (ADU, service = "catnap-batch", service = "vigilante-plugin", delegation
= "yes", host = "friend-TAP")
session header = service -> 5 (catnap-batch + vigilante), delegation = 1
```

```
transfer header = hint -> friend-TAP)
```

The above usage is also illustrated in Figure 6.3. Both services (Vigilante and Catnap) need to register with the transfer layer of the TAP. Because delegation semantics are required, the session involves both the end-points as well as the TAP. The transferred ADUs are first delivered to the Vigilante service which caches it and sends an acknowledgement to the sender who would then discard the ADU because the responsibility has been delegated to the TAP. Vigilante service would then send the ADU to the Catnap service – in our current design, it uses the same interface that the Catnap service uses to communicate with the transfer layer. Future work should explore the question of what kind of interfaces should services expose to other services. Finally, after batching the ADUs the Catnap service will send the ADUs to the receiver.

To fully understand the working of the above scenario that involves use of both Catnap and Vigilante, we contrast it with the case when only Catnap is used. This is shown in Figure 6.4; as we can see, the session only involves the two end-points because the only service that is being used (i.e., Catnap) does not change the session semantics (reliability, confidentiality, etc).

Swift and Catnap

At a high level, it is important to appreciate that Catnap and Swift are targeted towards different kind of scenarios. Catnap works best in typical wireless access scenarios where the user is stationary and is accessing the Internet through some high-speed wireless network. Important mechanisms within Catnap, such as the scheduler, are implemented based on the assumption that the user will remain connected to the Catnap proxy throughout the duration of the ADU transfer. In contrast, Swift focuses on scenarios involving high mobility, such as vehicular Internet access. In such scenarios, a user often doesn't remain connected to the same access point, thereby eliminating the role of a Catnap-like service that can provide energy savings.

Even though the above systems target different scenarios, they can still be used together in Tapa as they both use the same API. Naturally, if we use Catnap in a mobile setting the benefits would change, although the data transfers would still continue to work. For example, if the user was initially static and was leveraging the Catnap service, but later on moves to a different TAP, it could still retrieve the data from the old TAP, using Swift-like functionality. Using Tapa's API, we can specify an "old TAP" that should be used for communication, exploiting the benefits of Swift and also specify preference to use a Catnap batch mode, thereby indicating that some level of delay is acceptable:

```
get (ADU, tap = "old-tap", service = "catnap-batch")
session header = service -> 1 (Catnap-Batch)
transfer header = hint -> old-tap
```

Future work could explore how Catnap's scheduler can account for mobility plans while making the scheduling decision. As a first step, the scheduler needs to know whether the user is likely to move or not. This information can be explicitly communicated to the TAP by the mobile client who is in a better position to anticipate client's mobility. Once the scheduler knows that the client is likely to move, it needs to make an appropriate decision. In the simplest solution, the scheduler can decide not to buffer data for mobile clients so as to ensure that they are not worse-off due to the use of Catnap. We can also imagine the scheduler coordinating with the other TAPs that the client is likely to use in future so as to buffer data at multiple TAPs. So when the user moves to the new TAP, it could still get the ADU in a burst. In this case, we can still provide energy savings, although the benefits would be highly dependent on the mobility pattern as well the wireless conditions.

Swift and Vigilante

Vigilante works best when the user is downloading data from its home TAP, as the home TAP is part of the personal CDN and may have the data in its local storage or even if it needs to retrieve data from a friend, it is likely to be in a good position to decide which friend's TAP should be used. In contrast, Swift is targeted towards a highly mobile setting where the user may need to download data from multiple road-side TAPs as she moves around. Therefore, the target scenarios for Swift and Vigilante are different. So if the user is highly mobile (i.e., Swift) then many of the benefits of Vigilante (e.g., nearby cache) are lost. However, we may still be able to get some benefits by retrieving data from our home TAP even if we are moving around and connected to some road-side TAP. Using Tapa's API, we can specify that the home AP should be used for communication, similar to how Swift provides this support to choose specific TAPs for communication, and also specify preference to use a delay tolerant segment between the TAP and the server.

```
put (ADU, tap = "home-tap", host = "server", segment = "slow")
session header = default
transfer header = tap -> home-tap, segment -> 1 (Slow Segment b/w TAP and server)
```

Catnap, Vigilante, and Swift

All three systems can be used together in the following scenario. The user is mobile, but will reach her home soon. Using Swift like functionality she can request that the ADU be downloaded on her home TAP, and as the ADU is available at her friend's TAP (Vigilante), she can specify the address of her friend's TAP. Finally, for increased energy savings, she may want to use Catnap as well. All this can be specified using the same API:

```
get (ADU, service = "catnap-batch", tap = "home-tap", host = "friend-TAP")
session header = service -> 1 (Catnap-Batch)
transfer header = tap-> home-tap, hint -> friend-tap
```

6.5.3 Vigilante++

For this case study, we build on top of Vigilante. Recall that Vigilante provides storage service at TAPs but no security features for data transfers. We extend the system to support security/privacy features which are quite important in the context of social networking applications. The extended system, Vigilante++, allows applications to support appropriate semantics for their communication corresponding to different levels of trust they may have on the intermediate storage service as well as the server.

We now present some typical uses cases for data transfers in Vigilante++, the semantics associated with these use-cases and how the API allows applications to choose these semantics.

- **Sending authentication info to server:** The application may need to send the user's username/password to the server for authentication. This information should not be read or modified by the intermediate service as well as anyone else. So the semantics associated with this communication would require an end-to-end encryption and data integrity validation. This is specified through the API in the following way:

  ```
  put(ADU, host = "server", "confidentiality = "full")
  session header = confidentiality -> 1
  ```

  Note that end-to-end data integrity is default, so it is not explicitly included. Also, the session layer of the TAP will not be involved in this case.

- **Exchanging Photo Meta-Data:** Recall that Vigilante required clients to send the photo meta-data to the server. The application plugin at the TAP may use the meta-data information to send notifications to certain friends or to do some other application specific task. So it may need to read the meta-data information, but of

course it should not modify it en-route to the server. Moreover, the user may not like to reveal meta-data information to anyone else, so only the intermediate service and the server may look at this information. So the semantics associated with this communication would require per-segment encryption but end-to-end data integrity. This can be specified through the API in the following way:

```
put(ADU, service = "vigilante_plugin", host = "server", confidentiality
= "partial")
session header = confidentiality -> 2 (partial), service -> 2 (vigilante)
```

- **Transferring Photos:** This is the most rich use case as in addition to the security semantics, it also involves the use of the storage service at the TAP and also the use of different types of segments (slow vs. fast). Note that in this case the server will also have to provide the key to other clients who may want to retrieve the photos.

  ```
  put(ADU, service = "vigilante-plugin", host = "server", segment = "slow",
  delegation = "yes", confidentiality = "partial")
  session header = confidentiality -> 2, service -> 2 (vigilante), delegation
  -> 1
  transfer header = segment -> 1
  ```

- **Transferring Photos - Full Privacy:** In this case, the intermediate service is not trusted so photos are stored in an encrypted form and the application chooses end-to-end encryption for transferring the ADU to the server. `put(ADU, host = "server", service = "storage", segment = "slow", delegation = "yes", confidentiality = "full") session header = confidentiality -> 1, service -> 3(storage), delegation -> 1`
  `transfer header = segment -> 1`

6.5.4 Complete Example

We now provide the complete sequence of calls that the server, TAP, and client would make.

Server

```
 //registering service name and the callback to receive incoming ADUs
register("vigilante", msg_callback);

//process incoming messages
```

```
msg_callback(message m)
{
   //app specific code to process messages
}
```

Client

```
//registering service name and the callback to receive incoming ADUs
register("vigilante", msg_callback);

//generating loginADU
loginADU = getLoginInfo();

//sending login ADU to the server
put(loginADU, host = "server", service = "vigilante, "confidentiality = "full");

//sending photo meta data to the server
put(photoMetaDataADU, service = "vigilante_plugin", host = "server",
service = "vigilante", confidentiality = "partial")

//sending photo the default storage service of the server
put(photoADU, tap_service = "storage", host = "server",  segment = "slow",
delegation = "yes", confidentiality = "partial")
```

TAP

```
 //registering service name and the callback to receive incoming ADUs
register("vigilante_plugin", msg_callback);

//process incoming messages
msg_callback(message m)
{
 //app specific code to process messages
}
```

6.6 Related Work

We discuss proposals that are relevant to Tapa's API and session layer services.

6.6.1 API

Tapa's basic `put/get` API is inspired by earlier work on data oriented transfer services (DOT [113]). DOT provides these operations on chunks of data whereas Tapa has a broader notion of ADUs on which these operations are performed. We can also include different information with the ADU, such as their importance, delay requirements, broader form of hints; all this information is not part of the standard DOT API as it focuses on a specific class of applications which has fixed requirements. Finally, the Tapa API provides applications with control over semantics, especially ones associated with in-network services. DOT's API has relatively fixed semantics and does not include any role of intermediaries.

Recent work on netAPI [26] has goals that are very similar to Tapa's API. It also provides a higher level abstraction that provides temporal and spatial decoupling, and does not require applications to specify transfer mechanisms/protocol. Moreover, it also provides control to applications so that if they need to specify the underlying transfer mechanism, they could do so. The main difference between netAPI and Tapa's API is in terms of naming and the role of services in the end-to-end semantics.

6.6.2 Session Services

Services similar to some of Tapa's session services have been considered before. DTNs provide a delegation service based on the concept of custody transfer. So each DTN hop can delegate the responsibility of reliably transferring the data to the next DTN hop. Tapa can support such semantics and offers several other variants of reliability semantics that are possible with the use of an intermediary storage service.

Prior work has also considered various forms of intermediary services in the middle of an SSL session between two end-points [19, 108]. These services are mostly hidden from either one or both the end-points. As a result, their confidentiality and failure semantics are different; the intermediary service usually acts on behalf of one of the end-points and its presence is transparent to the other end-point, similar to how transparency HTTP proxies work. In our design, the intermediary services are explicitly visible to both end-points and therefore can support richer semantics at the granularity of ADUs. For example, the intermediary service can be owned by a third-party and both end-points may have a *variable* degree of trust on the intermediary.

6.7 Summary

In this chapter, we showed how Tapa can support rich communication semantics involving end-points and intermediary services, as part of a session layer. Such rich semantics can be supported in Tapa due to the explicit notion of intermediaries and the necessary infrastructure support provided by the lower layers (transfer and segment layers). We also provided the design of an API that allows applications to choose different semantics based on their requirements. Finally, as a case study, we showed how an OSN application can use these diverse semantics in different scenarios to control how an intermediary service may operate on its data.

Chapter 7

Conclusion

This thesis addresses the problem of how to accommodate the growing diversity of networks and devices, and various types of in-network services in today's Internet. We believe that this diversity is only going to increase in future as mobile and wireless access becomes ubiquitous and the trend towards data and service centric networking continues to grow. Given the importance of this problem, we took a system wide approach, instead of proposing point solutions that have limited benefits. We proposed a new architecture, Tapa, which involves the network in certain functions that were traditionally supported at only the end-points as part of transport and application protocols. We demonstrated the effectiveness of the proposed architecture through a diverse set of case studies, which illustrated the flexibility of the architecture in accommodating diversity at different levels of the system. Our research raises several important questions as well as opens new research avenues that can be explored in future. In the rest of this chapter, we summarize the key contributions of this thesis and discuss some open issues and future research directions created by our work.

7.1 Summary of Key Contributions

- We presented the design of Tapa, an architecture that systematically combines two concepts: i) unbundling today's transport in both the vertical and horizontal dimensions of the system and as a result implementing most of the traditional transport functions on a per-segment basis rather than on an end-to-end basis, and ii) exposing applications' data units and their naming to certain elements within the network. We show how Tapa's modular design, which includes three layers: segment, transfer and session, allows separation of different kinds of concerns and facilitates diversity at

multiple levels. We also presented a new API that enables applications to make full use of Tapa using a simple but powerful interface.

- We presented the design, implementation, and evaluation of Swift, a prototype implementation of Tapa which focuses on the requirements of mobile and wireless users. We showed the benefits of Swift at three levels: i) it can accommodate diverse protocols that are tailored for specific environments (e.g., HOP, Bluetooth) as segment protocols without worrying about their impact on the protocols that are used on the wired side, as well as ensuring that end-to-end semantics are preserved, ii) we can easily support several optimizations, such as multiplexing multiple APs or interfaces, which are difficult to implement in today's Internet, and iii) we can better support mobile data transfers by exploiting caching in the network, resulting in improved performance for mobile clients. These benefits were illustrated on both a real world and an emulator based testbed.

- We presented the design, implementation, and evaluation of Catnap, an in-network traffic shaping service that provides energy savings to mobile clients. The Catnap service builds on top of decoupling of segments, which is a key feature of Tapa, and uses Tapa's notion of ADUs to identify when data is consumed by the application. We showed through a realistic evaluation that Catnap provides up to 2-5x battery life improvement for mobile devices under certain conditions. We also presented case studies that showed how existing protocols, like HTTP and IMAP, can easily use and benefit from Catnap.

- We presented a detailed case study of Facebook, which illustrated the strategies typically adopted by large scale online social networks in distributing their content. Our study also sheds unique insights into the limitations of using existing solutions (i.e., CDNs), which work well for the traditional web, but are not too suitable for distributing content in large scale OSNs.

- We presented the design, implementation, and evaluation of Vigilante, a service that allows network elements (TAPs) to form a personal CDN for storing and distributing OSN content. We presented the evaluation of Vigilante on the PlanetLab testbed and showed that it outperforms the best case performance achieved using Facebook's infrastructure.

- We presented the design and implementation of a session protocol that provides traditional end-to-end semantics associated with four functions: reliability, ordering,

confidentiality, and data integrity, as well as new semantics that accommodate the role of in-network services. We presented a case study of a social network application to illustrate how real world applications can use these rich semantics and control how in-network services operate on their data.

7.2 Future Work

Like all architectures, we expect Tapa's design to evolve with time as it is put to different kinds of uses. We believe that the core concepts in Tapa are broad and flexible enough to support a wide range of design options that may be required in future. The case studies showed how diverse services can be leveraged in typical wireless access scenarios that involve two segment paths. As we use Tapa in other scenarios (e.g., in the core or in untrusted environments), new issues will arise, creating a plethora of opportunities for future research. We discuss some of the interesting directions that can be pursued in future.

7.2.1 TAPs in the core

In this thesis, we considered scenarios where TAPs were used at the edges and the end-to-end path mostly consisted of two segments — a wired segment and a wireless access segment. For performance and policy reasons, we may also want to use TAPs in the core as well. For example, a customized protocol for high speed optical fiber in the core may provide improved performance or policy reasons may necessitate the introduction of new services in the core network. As an extreme case, we can imagine that every router in today's Internet will be replaced by a TAP in future. Using TAPs in the core will create new challenges and opportunities, such as:

Designing Efficient Segment Protocols: We mostly considered existing protocols or their minor variants as segment protocols (e.g., TCP, HOP, etc). These protocols are designed to work between end-points, where the scalability concerns are different compared to using these protocols at TAPs within in the core of the Internet. This requires that the segment protocol should be scalable enough to handle the number of client requests that a typical core TAP will receive. One possible direction could be to design protocols that are *receiver driven*, instead of being sender driven, because TAPs may have large caches, so they will be acting as sources of data (i.e., senders) in many cases. A receiver driven protocol will put the receiver in-charge of the segment functions, such as maintaining timers for retransmissions, or maintaining segment control information. This will improve

the capacity of TAPs to handle a large number of concurrent clients as the majority of the overhead will be borne by the individual clients rather than the TAPs.

Challenges with Multiple Segment Paths: As we move TAPs to the core, a typical end-to-end path will consist of more than two segments, thereby requiring suitable solutions for TAP discovery, routing, and resource management. For example, in this thesis we considered TAP discovery that was based on either application information (e.g., Vigilante) or lower layer information provided by the 802.11 protocol (e.g., Swift). As we move to multiple segment paths, TAPs will require mechanisms to discover other TAPs in the core network as well as to make appropriate routing decisions. These protocols could leverage the huge amount of work that has been done on control plane protocols for the core Internet as well as on overlay networks [28]. TAP discovery is also related to service discovery as TAPs run a wide variety of services. This makes prior work on service discovery protocols for various types of environments (e.g., wide area, LANs, etc) relevant [12]. We believe that recent work on service oriented network architectures (e.g., XIA [23]) will facilitate these tasks, as service discovery and primitives like anycast will be naturally supported by the network.

Another issue that will become more important is resource management of TAP buffers, as multiple-segment paths will require solutions different than the one we used in the current Tapa prototype. As we considered two segment paths, we used per-segment flow control and back-pressure to ensure that TAP buffers do not overflow. As we discussed earlier, Tapa has several ingredients that can facilitate various other forms of resource management techniques (e.g., congestion avoidance, admission control and policing, etc), which will become more relevant for multiple-segment paths.

7.2.2 Diverse Segment Protocols

Tapa can accommodate a wide range of options as segment protocols. The case studies showed several such examples that were specific to wireless segments. For the wired Internet, we used TCP/IP. Future work should explore diverse options as segment protocols, for emerging edge networks, like data center networks (e.g., DCTCP [21]), as well as for new architectures that are proposed for the core Internet (e.g., XIA [23], NDN [65], MobilityFirst [6], etc). In fact, in many scenarios today, end-to-end paths typically consist of three segments: a wireless access segment, a wired Internet segment, and a data center segment. This represents the typical case of a mobile client accessing some cloud service running in a data center. In such scenarios, if we use Tapa over an architecture like XIA [23]

then we can easily use customized protocols for each segment – for example, a protocol like HOP or Blast for the wireless segment, a TCP-like protocol for the wired Internet, and solutions like DCTCP [21] within the data center network. As these protocols work over XIA, which naturally supports services, several service specific tasks that Tapa need to do are avoided while leveraging additional benefits provided by XIA, like service migration, intrinsic security, late binding, etc.

7.2.3 Non-Data Oriented Applications over Tapa

The focus of this thesis was on data oriented applications as they are the dominant set of applications in use today and are likely to remain popular in the near future too. However, future work should evaluate the benefits of Tapa for non-data oriented applications, e.g., VoIP. Note that even though Tapa is targeted towards data oriented application, it has several features that can benefit *all* applications. For example, decoupling of segments and as a result improving the efficiency of network protocols within a segment will benefit any application. Similarly, many session services, such as controlled transparency, can benefit all applications as the need for confidentiality during communication is important for non data oriented applications as well.

Another related issue is the complexity of making non-data oriented applications use Tapa. This includes writing new applications as well as modifying legacy ones to use Tapa's API. Our experiences with data oriented applications suggest that Tapa's API is quite simple and easy to use but future work should evaluate its efficacy for non-data oriented applications as well.

7.2.4 Policy Implications

This thesis focused on the technical issues involving the use of Tapa, but future work should explore the policy implications of the various important changes that Tapa introduces in the Internet architecture. For example, how does Tapa facilitate competition within the network or impact the debate on network neutrality.

We believe that Tapa has positive impact on most of the key policy issues faced in today's Internet. For example, by making role of the network visible, it facilitates smaller players to introduce different kinds of in-network services without necessarily requiring support from the ISP. This is in contrast to today's Internet, where it is difficult for third-parties to innovate inside the network because of the end-to-end nature of today's Internet. As a result, ISPs have an unfair advantage when it comes to introducing in-network services because it is easier for them to build service on top of their proprietary products. Making

TAPs a part of the architecture standardizes their role and thereby facilitates independent third parties to build diverse network services that can run on TAPs owned by different ISPs. Future work should look at such policy issues in a more systematic manner.

7.2.5 Security Implications

Making TAPs an important part of the end-to-end communication also means that we may end up creating more vulnerabilities in the Internet, especially if TAPs are not properly secured. For example, a compromised TAP can potentially be more harmful compared to a compromised IP router as it offers various important services that are needed for end-to-end communication. This implies that security considerations should be given importance during the design of various protocols within Tapa (e.g., segment protocols, service discovery protocols, etc). Moreover, as we foresee multiple third party services running on the TAP, there should be mechanisms for proper isolation between different services so as to limit the harm caused by a compromised service. We believe that prior work on source authentication [27], accountable protocols [23], and virtualization [34] can be applied to address these concerns, but they will need to be adapted to cater to some of the new aspects of Tapa.

Bibliography

[1] CMU Wireless Emulator. www.cs.cmu.edu/ emulator/.

[2] Emulab Wireless Testbed. www.emulab.net.

[3] Facebook engineering notes. http://www.facebook.com/note.php?note_id=23844338919.

[4] Facebook statistics. http://www.facebook.com/press/info.php?statistics.

[5] Linksys 350n router. http://www.ubergizmo.com/15/archives/2006/10/linksys_wrt350n_gigabit_80211n_router.html.

[6] Mobilityfirst future internet architecture project. http://mobilityfirst.winlab.rutgers.edu/.

[7] Nano data centers. http://www.nanodatacenters.eu/.

[8] Napster. http://www.napster.com/.

[9] OpenSSL: The Open Source Toolkit for SSL/TLS. http://www.openssl.org/.

[10] Peerson. http://www.peerson.net/.

[11] Received signal strength indication. http://en.wikipedia.org/wiki/Received_signal_strength_indication.

[12] Service Location Protocol. RFC 2608.

[13] TCPDUMP/LIBPCAP public library. http://www.tcpdump.org/.

[14] Vanlan. research.microsoft.com/en-us/projects/vanlan/.

[15] Viraltor Proxy Virus Scanner. http://freshmeat.net/projects/viralator/.

[16] Wake-on-lan. en.wikipedia.org/wiki/Wake-on-LAN.

[17] Wake-on-wireless lan. www.intel.com/support/wireless/wlan/sb/CS-029827.htm.

[18] Zigbee alliance. http://www.zigbee.org/.

[19] Ssl splitting: Securely serving data from untrusted caches. *Computer Networks*, 48(5):763 – 779, 2005.

[20] Yuvraj Agarwal, Steve Hodges, Ranveer Chandra, James Scott, Paramvir Bahl, and Rajesh Gupta. Somniloquy: augmenting network interfaces to reduce pc energy usage. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 365–380, Berkeley, CA, USA, 2009. USENIX Association.

[21] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. Data center tcp (dctcp). In *Proceedings of the ACM SIGCOMM 2010 conference on SIGCOMM*, SIGCOMM '10, pages 63–74, New York, NY, USA, 2010. ACM.

[22] M. Allman, K. Christensen, B. Nordman, and V. Paxon. Enabling an Energy-Efficient Future Internet Through Selectively Connected End Systems. In *ACM HotNets*, 2007.

[23] Ashok Anand, Fahad Dogar, Dongsu Han, Boyan Li, Hyeontaek Lim, Michel Machado, Wenfei Wu, Aditya Akella, David Andersen, John Byers, Srinivasan Seshan, and Peter Steenkiste. XIA: An Architecture for an Evolvable and Trustworthy Internet. *CMU-CS-11-100*, 2011.

[24] Ashok Anand, Chitra Muthukrishnan, Aditya Akella, and Ramachandran Ramjee. Redundancy in network traffic: findings and implications. In *Proceedings of the eleventh international joint conference on Measurement and modeling of computer systems*, SIGMETRICS '09, pages 37–48, New York, NY, USA, 2009. ACM.

[25] Manish Anand, Edmund B. Nightingale, and Jason Flinn. Self-tuning wireless network power management. In *MobiCom '03: Proceedings of the 9th annual international conference on Mobile computing and networking*, pages 176–189, New York, NY, USA, 2003. ACM.

[26] Ganesh Ananthanarayanan, Kurtis Heimerl, Matei Zaharia, Michael Demmer, Teemu Koponen, Arsalan Tavakoli, Scott Shenker, and Ion Stoica. Enabling innovation below the communication api. Technical Report UCB/EECS-2009-141, EECS Department, University of California, Berkeley, Oct 2009.

[27] David G. Andersen, Hari Balakrishnan, Nick Feamster, Teemu Koponen, Daekyeong Moon, and Scott Shenker. Accountable Internet Protocol (AIP). In *SIGCOMM*, 2008.

[28] David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Robert Morris. Resilient Overlay Networks. In *ACM SOSP*, pages 131–145, Banff, Canada, October 2001.

[29] Ajay V. Bakre and B.r. Badrinath. Implementation and performance evaluation of indirect tcp. *IEEE Transactions on Computers*, 46(3):260–278, 1997.

[30] Hari Balakrishnan, Hariharan S. Rahul, and Srinivasan Seshan. An integrated congestion management architecture for internet hosts. In *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, SIGCOMM '99, pages 175–187, New York, NY, USA, 1999. ACM.

[31] Hari Balakrishnan, Srinivasan Seshan, Elan Amir, and Randy H. Katz. Improving tci/ip performance over wireless networks. In *MobiCom '95*, pages 2–11, New York, NY, USA, 1995. ACM Press.

[32] Aruna Balasubramanian, Ratul Mahajan, Arun Venkataramani, Brian Neil Levine, and John Zahorjan. Interactive wifi connectivity for moving vehicles. In *SIGCOMM '08*, pages 427–438, New York, NY, USA, 2008. ACM.

[33] Niranjan Balasubramanian, Aruna Balasubramanian, and Arun Venkataramani. Energy Consumption in Mobile Phones: A Measurement Study and Implications for Network Applications. In *Proc. ACM IMC*, November 2009.

[34] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. *SIGOPS Oper. Syst. Rev.*, 37:164–177, October 2003.

[35] Doug. Beaver, Sanjeev Kumar, Hari Li, Jason. Sobel, and Peter Vagjel. Finding a needle in haystack: Facebook's photo storage. OSDI '10.

[36] Fabrício Benevenuto, Tiago Rodrigues, Meeyoung Cha, and Virgílio Almeida. Characterizing user behavior in online social networks. In *IMC '09*, pages 49–62, New York, NY, USA, 2009. ACM.

[37] John Bicket, Daniel Aguayo, Sanjit Biswas, and Robert Morris. Architecture and evaluation of an unplanned 802.11b mesh network. In *MobiCom '05: Proceedings of*

the 11th annual international conference on Mobile computing and networking, pages 31–42, New York, NY, USA, 2005. ACM Press.

[38] Sanjit Biswas and Robert Morris. Exor: opportunistic multi-hop routing for wireless networks. *SIGCOMM CCR*, 35(4):133–144, 2005.

[39] J. Border, M. Kojo, J. Griner, G. Montenegro, and Z. Shelby. Performance enhancing proxies intended to mitigate link-related degradations, 2001.

[40] Lawrence S. Brakmo, Deborah A. Wallach, and Marc A. Viredaz. μsleep: a technique for reducing energy consumption in handheld devices. In *MobiSys '04*, pages 12–22, New York, NY, USA, 2004. ACM.

[41] Meeyoung Cha, Haewoon Kwak, Pablo Rodriguez, Yong-Yeol Ahn, and Sue Moon. I tube, you tube, everybody tubes: analyzing the world's largest user generated content video system. In *IMC '07*, pages 1–14, New York, NY, USA, 2007. ACM.

[42] Surendar Chandra and Amin Vahdat. Application-specific network management for energy-aware streaming of popular multimedia formats. In *USENIX Annual Technical Conference*, Berkeley, CA, USA, 2002. USENIX Association.

[43] Gregory Chockler, Roie Melamed, Yoav Tock, and Roman Vitenberg. Spidercast: a scalable interest-aware overlay for topic-based pub/sub communication. In *Proceedings of the 2007 inaugural international conference on Distributed event-based systems*, DEBS '07, pages 14–25, New York, NY, USA, 2007. ACM.

[44] D. Clark. The design philosophy of the darpa internet protocols. In *SIGCOMM '88*, pages 106–114, New York, NY, USA, 1988. ACM.

[45] D. D. Clark and D. L. Tennenhouse. Architectural considerations for a new generation of protocols. In *SIGCOMM*, New York, NY, USA, 1990. ACM.

[46] David D. Clark, Craig Partridge, Robert T. Braden, Bruce Davie, Sally Floyd, Van Jacobson, Dina Katabi, Greg Minshall, K. K. Ramakrishnan, Timothy Roscoe, Ion Stoica, John Wroclawski, and Lixia Zhang. Making the world (of communications) a different place. *SIGCOMM CCR.*, 35(3):91–96, 2005.

[47] David D. Clark, Karen Sollins, John Wroclawski, and Ted Faber. Addressing reality: an architectural response to real-world demands on the evolving internet. *SIGCOMM Comput. Commun. Rev.*, 33(4):247–257, 2003.

[48] Bram Cohen. Incentives Build Robustness in BitTorrent, 2003.

[49] Eduardo Cuervo, Aruna Balasubramanian, Dae-ki Cho, Alec Wolman, Stefan Saroiu, Ranveer Chandra, and Paramvir Bahl. Maui: making smartphones last longer with code offload. MobiSys '10, pages 49–62, New York, NY, USA, 2010. ACM.

[50] Saumitra M. Das, Himabindu Pucha, and Charlie Y. Hu. Mitigating the gateway bottleneck via transparent cooperative caching in wireless mesh networks. *Ad Hoc Networks (Elsevier) Journal*, Special Issue on Wireless Mesh Networks, 2007.

[51] Marcel Dischinger, Andreas Haeberlen, Krishna P. Gummadi, and Stefan Saroiu. Characterizing Residential Broadband Networks. In *IMC '07*, New York, NY, USA, 2007. ACM Press.

[52] Fahad R. Dogar, Amar Phanishayee, Himabindu Pucha, Olatunji Ruwase, and David G. Andersen. Ditto: a system for opportunistic caching in multi-hop wireless networks. In *ACM MobiCom*, 2008.

[53] Fahad R. Dogar and Peter Steenkiste. M2: Using Visible Middleboxes to Serve Proactive Mobile-Hosts. In *ACM SIGCOMM MobiArch '08*, pages 85–90, New York, NY, USA, 2008. ACM.

[54] Fahad R. Dogar and Peter Steenkiste. Segment based internetworking to accommodate diversity at the edge. *CMU-CS-10-104*, 2010.

[55] Fahad R. Dogar, Peter Steenkiste, and Konstantina Papagiannaki. Catnap: Exploiting high bandwidth wireless interfaces to save energy for mobile devices. In *ACM MobiSys*, pages 107–122, New York, NY, USA, 2010. ACM.

[56] Jakob Eriksson, Hari Balakrishnan, and Samuel Madden. Cabernet: vehicular content delivery using wifi. In *MobiCom '08*, pages 199–210, New York, NY, USA, 2008. ACM.

[57] Carla fabiana Chiasserini and Ramesh R. Rao. Improving battery performance by using traffic shaping techniques. *IEEE Journal on Selected Areas in Communications*, 19:1385–1394, 2001.

[58] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03*, pages 27–34, New York, NY, USA, 2003. ACM.

[59] Jason Flinn and M. Satyanarayanan. Energy-aware adaptation for mobile applications. In *SOSP '99*. ACM, 1999.

[60] Bryan Ford and Janardhan Iyengar. Breaking up the transport logjam. In *ACM Hotnets*, New York, NY, USA, 2008. ACM.

[61] Saikat Guha and Paul Francis. An end-middle-end approach to connection establishment. In *SIGCOMM*, New York, NY, USA, 2007.

[62] Dhruv Gupta, Daniel Wu, Prasant Mohapatra, and Chen-Nee Chuah. Experimental comparison of bandwidth estimation tools for wireless mesh networks. In *IEEE INFOCOM*, 2009.

[63] David Hadaller, Srinivasan Keshav, Tim Brecht, and Shubham Agarwal. Vehicular opportunistic communication under the microscope. In *MobiSys '07: Proceedings of the 5th international conference on Mobile systems, applications and services*, pages 206–219, New York, NY, USA, 2007. ACM.

[64] An.-C. Huang and P. Steenkiste. Building self-configuring services using service-specific knowledge. In *High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on*, pages 45 – 54, june 2004.

[65] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *CoNEXT '09*, pages 1–12, New York, NY, USA, 2009. ACM.

[66] Manish Jain and Constantinos Dovrolis. End-to-end available bandwidth: measurement methodology, dynamics, and relation with tcp throughput. In *SIGCOMM '02*, New York, NY, USA, 2002.

[67] Szymon Jakubczak, David G. Andersen, Michael Kaminsky, Konstantina Papagiannaki, and Srinivasan Seshan. Link-alike: using wireless to share network resources in a neighborhood. *SIGMOBILE Mob. Comput. Commun. Rev.*, 12:1–14, February 2009.

[68] Srikanth Kandula, Kate Ching-Ju Lin, Tural Badirkhanli, and Dina Katabi. FatVAP: Aggregating AP Backhaul Capacity to Maximize Throughput. In *NSDI*, San Francisco, CA, April 2008.

[69] Dina Katabi, Mark Handley, and Charlie Rohrs. Congestion control for high bandwidth-delay product networks. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '02, pages 89–102, New York, NY, USA, 2002. ACM.

[70] Sachin Katti, Hariharan Rahul, Wenjun Hu, Dina Katabi, Muriel Medard, and Jon Crowcroft. Xors in the air: practical wireless network coding. In *SIGCOMM '06*, pages 243–254, New York, NY, USA, 2006. ACM Press.

[71] Eddie Kohler, Mark Handley, and Sally Floyd. Designing dccp: congestion control without reliability. In *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '06, pages 27–38, New York, NY, USA, 2006. ACM.

[72] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *SIGCOMM '07*, pages 181–192, New York, NY, USA, 2007. ACM.

[73] Kopparty, S. Krishnamurthy, S. V. Faloutsos, and M. Tripathi. Split tcp for mobile ad hoc networks. *IEEE Globecom*, pages 138–142, 1998.

[74] Michael Kozuch and M. Satyanarayanan. Internet suspend/resume. In *WMCSA '02*, pages 40–48, Washington, DC, USA, 2002. IEEE Computer Society.

[75] Ronny Krashinsky and Hari Balakrishnan. Minimizing Energy for Wireless Web Access Using Bounded Slowdown. In *ACM MOBICOM 2002*, Atlanta, GA, September 2002.

[76] Balachander Krishnamurthy, Craig Wills, and Yin Zhang. On the use and performance of content distribution networks. In *IMW '01*, pages 169–182, New York, NY, USA, 2001. ACM.

[77] Rupa Krishnan, Harsha V. Madhyastha, Sridhar Srinivasan, Sushant Jain, Arvind Krishnamurthy, Thomas Anderson, and Jie Gao. Moving beyond end-to-end path information to optimize cdn performance. In *IMC '09*, pages 190–201, New York, NY, USA, 2009. ACM.

[78] Rafael Laufer, Theodoros Salonidis, Henrik Lundgren, and Pascal Le Guyadec. Design and implementation of backpressure scheduling in wireless multi-hop networks: from theory to practice. *SIGMOBILE Mob. Comput. Commun. Rev.*, 14:40–42, December 2010.

[79] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, and Stephen Wolff. A brief history of the internet. *SIGCOMM Comput. Commun. Rev.*, 39:22–31, October 2009.

[80] Philip Levis, Sam Madden, Joseph Polastre, Robert Szewczyk, Alec Woo, David Gay, Jason Hill, Matt Welsh, Eric Brewer, and David Culler. Tinyos: An operating system for sensor networks. In *in Ambient Intelligence*. Springer Verlag, 2004.

[81] Ming Li, Devesh Agrawal, Deepak Ganesan, and Arun Venkataramani. Block-switched networks: a new paradigm for wireless transport. In *NSDI'09*, pages 423–436, Berkeley, CA, USA, 2009. USENIX Association.

[82] Jiayang Liu and Lin Zhong. Micro power management of active 802.11 interfaces. In *MobiSys '08: Proceeding of the 6th international conference on Mobile systems, applications, and services*, pages 146–159, New York, NY, USA, 2008. ACM.

[83] Xin Liu, Xiaowei Yang, and Yong Xia. Netfence: preventing internet denial of service from inside out. In *ACM SIGCOMM 2010*.

[84] N. Milanovic and M. Malek. Current solutions for web service composition. *Internet Computing, IEEE*, 8(6):51 – 59, nov.-dec. 2004.

[85] Rohan Murty, Jitendra Padhye, Ranveer Chandra, Alec Wolman, and Brian Zill. Designing high performance enterprise wi-fi networks. In *NSDI'08*, pages 73–88, Berkeley, CA, USA, 2008. USENIX Association.

[86] Vinod Namboodiri and Lixin Gao. Towards energy efficient voip over wireless lans. In *MobiHoc '08: Proceedings of the 9th ACM international symposium on Mobile ad hoc networking and computing*, pages 169–178, New York, NY, USA, 2008. ACM.

[87] Atif Nazir, Saqib Raza, Dhruv Gupta, Chen-Nee Chuah, and Balachander Krishnamurthy. Network level footprints of facebook applications. In *IMC '09*, pages 63–75, New York, NY, USA, 2009. ACM.

[88] Sergiu Nedevschi, Lucian Popa, Gianluca Iannaccone, Sylvia Ratnasamy, and David Wetherall. Reducing network energy consumption via sleeping and rate-adaptation. In *NSDI'08*, pages 323–336, Berkeley, CA, USA, 2008. USENIX Association.

[89] Trevor Pering, Yuvraj Agarwal, Rajesh Gupta, and Roy Want. Coolspots: reducing the power consumption of wireless mobile devices with multiple radio interfaces. In *MobiSys '06: Proceedings of the 4th international conference on Mobile systems, applications and services*, pages 220–232, New York, NY, USA, 2006. ACM.

[90] C. Perkins. Mobile ip. In *International Journal Of Communication Systems*, pages 3–20, 1998.

[91] Christian Poellabauer and Karsten Schwan. Energy-aware traffic shaping for wireless real-time applications. In *RTAS '04: Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium*, page 48, Washington, DC, USA, 2004. IEEE Computer Society.

[92] Lucian Popa, Ali Ghodsi, and Ion Stoica. HTTP as the narrow waist of the future Internet. In *Hotnets 2010*.

[93] J. Postel. NCP/TCP transition plan. In *RFC 801*, Nov 1981.

[94] Himabindu Pucha, David G. Andersen, and Michael Kaminsky. Exploiting similarity for multi-source downloads using file handprints. In *Proc. 4th USENIX NSDI*, Cambridge, MA, April 2007.

[95] Josep M. Pujol, Vijay Erramilli, Georgos Siganos, Xiaoyuan Yang, Nikos Laoutaris, Parminder Chhabra, and Pablo Rodriguez. The little engine(s) that could: scaling online social networks. SIGCOMM '10, pages 375–386, New York, NY, USA, 2010. ACM.

[96] Ihsan Ayyub Qazi, Lachlan L. H. Andrew, and Taieb Znati. Congestion Control using Efficient Explicit Feedback. In *Proc. IEEE INFOCOM*, Rio de Janeiro, Brazil, 20-25 Apr 2009.

[97] Sumit Rangwala, Apoorva Jindal, Ki-Young Jang, Konstantinos Psounis, and Ramesh Govindan. Understanding congestion control in multi-hop wireless mesh networks. In *Proceedings of the 14th ACM international conference on Mobile computing and networking*, MobiCom '08, pages 291–302, New York, NY, USA, 2008. ACM.

[98] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. A scalable content-addressable network. SIGCOMM '01, pages 161–172, New York, NY, USA, 2001. ACM.

[99] Sean Rhea, Brighten Godfrey, Brad Karp, John Kubiatowicz, Sylvia Ratnasamy, Scott Shenker, Ion Stoica, and Harlan Yu. Opendht: a public dht service and its uses. In *Proceedings of the 2005 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM '05, pages 73–84, New York, NY, USA, 2005. ACM.

[100] Marcel C. Rosu, C. Michael Olsen, Chandra Narayanaswami, and Lu Luo. Pawp: A power aware web proxy for wireless lan clients. In *WMCSA '04*, 2004.

[101] Sumit Roy, Bo Shen, Vijay Sundaram, and Raj Kumar. Application level hand-off support for mobile media transcoding sessions. In *NOSSDAV '02*, pages 95–104, New York, NY, USA, 2002. ACM.

[102] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 1984.

[103] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4):14 –23, 2009.

[104] Fabian Schneider, Anja Feldmann, Balachander Krishnamurthy, and Walter Willinger. Understanding online social network usage from a network perspective. In *IMC '09*, pages 35–48, New York, NY, USA, 2009. ACM.

[105] J. Scott, P. Hui, J. Crowcroft, and C. Diot. Haggle: A networking architecture designed around mobile users. In *WONS*, 2006.

[106] Ashish Sharma, Vishnu Navda, Ramachandran Ramjee, Venkata N. Padmanabhan, and Elizabeth M. Belding. Cool-tether: energy efficient on-the-fly wifi hot-spots using mobile phones. In *CoNEXT '09*, New York, NY, USA, 2009. ACM.

[107] Alex C. Snoeren and Hari Balakrishnan. An end-to-end approach to host mobility. In *MobiCom '00*, pages 155–166, New York, NY, USA, 2000. ACM Press.

[108] Yong Song, Victor Leung, and Konstantin Beznosov. Supporting end-to-end security across proxies with multiple-channel ssl. In Yves Deswarte, FrÃĬdÃĬric Cuppens, Sushil Jajodia, and Lingyu Wang, editors, *Security and Protection in Information Processing Systems*, IFIP International Federation for Information Processing, pages 323–337. Springer Boston, 2004.

[109] Jacob Sorber, Nilanjan Banerjee, Mark D. Corner, and Sami Rollins. Turducken: hierarchical power management for mobile devices. In *MobiSys '05*, pages 261–274, New York, NY, USA, 2005. ACM.

[110] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. *SIGCOMM Comput. Commun. Rev.*, 32(4):73–86, 2002.

[111] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for*

computer communications, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. ACM.

[112] Ao-Jan Su, David R. Choffnes, Aleksandar Kuzmanovic, and Fabián E. Bustamante. Drafting behind akamai (travelocity-based detouring). In *SIGCOMM '06*, pages 435–446, New York, NY, USA, 2006. ACM.

[113] Niraj Tolia, Michael Kaminsky, David G. Andersen, and Swapnil Patil. An architecture for internet data transfer. In *NSDI '06*.

[114] Cheng-Lin Tsao and Raghupathy Sivakumar. On effectively exploiting multiple wireless interfaces in mobile hosts. In *CoNEXT '09*, pages 337–348, New York, NY, USA, 2009. ACM.

[115] Masoud Valafar, Reza Rejaie, and Walter Willinger. Beyond friendship graphs: a study of user interactions in flickr. In *WOSN '09*, pages 25–30, New York, NY, USA, 2009. ACM.

[116] Michael Walfish, Jeremy Stribling, Maxwell Krohn, Hari Balakrishnan, Robert Morris, and Scott Shenker. Middleboxes no longer considered harmful. *OSDI*, pages 215–230, 2004.

[117] Walter Willinger, Reza Rejaie, Mojtaba Torkjazi, Masoud Valafar, and Mauro Maggioni. Research on online social networks: time to face the real challenges. *SIGMETRICS Perform. Eval. Rev.*, 37(3):49–54, 2009.

[118] Mike P. Wittie, Veljko Pejovic, Lara Deek, Kevin C. Almeroth, and Ben Y. Zhao. Exploiting locality of interest in online social networks. Co-NEXT '10, pages 25:1–25:12, New York, NY, USA, 2010. ACM.

[119] Ben Y. Zhao, John D. Kubiatowicz, and Anthony D. Joseph. Tapestry: a fault-tolerant wide-area application infrastructure. *SIGCOMM Comput. Commun. Rev.*, 32:81–81, January 2002.