Segment based Internetworking to Accommodate Diversity at the Edge

Fahad R. Dogar[*] **Peter Steenkiste**[†]

February 2010
CMU-CS-10-104

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

[*]Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA.
[†]Computer Science Department and Department of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA, USA.

Keywords: network architecture, diversity, heterogeneity, network services, wireless, mobility

Abstract

In this paper, we introduce Tapa, a network architecture that accommodates diversity at the network edge: different access networks, heterogeneous edge devices, and rich applications and network services. The core idea underlying Tapa is to have network *segments* replace IP links as the basis for inter-operability in the Internet. A segment spans a part of the end-to-end path that can be considered homogeneous (e.g., wired Internet or an access network) – network specific functions and optimizations, like congestion control, are implemented *within* a segment, allowing use of customized solutions. On top of segments is the new inter-operability layer, called the *transfer* layer, which manages end-to-end data transfer over multiple segments, supporting better source and path selection mechanisms than IP. Finally, on top of the transfer layer is an end-to-end *transport* layer that mainly deals with implementing a rich set of semantics to accommodate diverse applications and network services. We implemented Tapa and show how it supports diverse applications and optimizations.

1 Introduction

The Internet architecture was designed with fairly homogeneous end-hosts and networks in mind. This is in sharp contrast to today's proliferation of diverse Internet access technologies (e.g., bluetooth, ultra-wide-band) and edge devices (e.g., cell-phone, PDAs, sensors), which has completely transformed the network *edge*. This increased diversity is not limited to the infrastructure, but also extends to applications (e.g., content sharing, gaming, sensing apps), services supported by the network (e.g., caching, mobile users), and the nature of network deployments (e.g., unmanaged residential wireless networks [4]). This diversity well exceeds that originally envisioned in the Internet architecture, creating a significant challenge since properly supporting these technologies often leads to ad-hoc solutions that violate one or more architectural principles of the Internet [7, 11, 20, 14].

Since these technologies are here to stay, it is critical that we develop architectural mechanisms – rather than point solutions – to support them. In this paper, we focus on support for four architectural requirements, that are broadly applicable but essential for accommodating both today's and future diversity at the edge:

- **Network and Device Heterogeneity:** Customized solutions are often needed for heterogeneous networks in the form of new routing, congestion control, or reliability mechanisms [9, 24, 18]. Some end devices may even need to use a completely customized protocol stack [23]. The network architecture should accommodate such solutions for effective integration of these networks and devices with the Internet.

- **Diverse Network Services:** Content and service-centric networking help in a broad range of application scenarios but can be *essential* in a sensor or mobile network setting [16]. The architecture should allow insertion of diverse services (caching, video transcoding, etc) within the network.

- **Relaxed Synchronization between End-points:** Network protocols often assume that both communicating hosts are simultaneously connected to the network. Clearly intermittent connectivity and disconnect operations have been a challenge in mobile computing for a long time, but with the increased interest in energy efficiency it is also becoming relevant for wired hosts [5]. Relaxed synchronization should be supported as a *routine* affair.

- **Topology Control:** The architecture should allow consideration of higher level criteria (e.g., disjoint paths for resilience or availability of services) in path selection as it is useful in general [6]. It is especially beneficial in many access networks, like wireless, which may offer opportunities like multiple interfaces, access points, or nearby sources of data.

This paper introduces Tapa, a network architecture that addresses the above requirements by raising the level of inter-operability of the Internet to a *segment*, a portion of an end-to-end path that is *homogeneous*. Some possible segments include: the "wired Internet", a private network owned by an enterprise, or a wireless mesh network. 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.

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 the network elements, called Transfer Access Points (TAPs), that inter-connect segments. As the number of segments in a path is small, the transfer layer can exert better topological control than IP, enabling multi-path and content-centric optimizations [31]. For example, our prototype implementation of the Tapa transfer service focuses on wireless access networks. We have used it to support several optimizations for wireless and mobile users such as mobility across segments, use of multiple APs/interfaces, and exploiting nearby sources of data.

Finally, the *transport* layer implements specific end-to-end application semantics over the transfer layer, working with data granularity at the level of application data units (ADUs) [13] rather than byte streams or packets. In contrast to, for example, TCP, the Tapa transport layer is relatively light-weight since it only operates over short paths, i.e., 2-3 segments connected by TAPs. The short path, combined with the use of ADUs, means that it is easier to implement 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 the (possibly third party) network service.

In order to address the limitations of the current Internet in accommodating the "transformed edge", this paper makes three fundamental contributions:

- We identify key requirements for the future Internet that have broad applicability but are especially important for diversity in edge networks. (Section 2)
- We introduce Tapa, a network architecture that meets the key requirements for accommodating diversity at the edges. Tapa puts several network specific functions inside a segment which makes the higher layers (inter-networking and transport layers) light-weight and flexible. (Section 3)
- We design, implement, and evaluate a a proof-of-concept prototype of Tapa to show the flexibility of the architecture. This includes: i) a transfer service that is specifically designed for mobile and wireless users, ii) various options for use as segments, and iii) a rich set of transport semantics that can support a variety of applications. (Sections 4 and 5)

2 Network Diversity

We elaborate on four key network requirements that are important for accommodating diversity at the edge.

2.1 Network and Device Heterogeneity

This original ARPANET was designed to inter-connect heterogeneous networks and hosts. However, by today's standards, they were in fact fairly homogeneous. For example, it was assumed that links would provide "reasonable" reliability (¡ 1% loss rate) [12] and that all end-hosts would be

powerful enough to support the full TCP/IP stack. However, 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 [7, 18]. Similarly, devices, such as some sensor and mobile nodes, are highly resource constrained.

Dealing with heterogeneity of devices and networks is difficult as it makes it challenging to assign different functions to different modules. For example, when applying the end-to-end argument [28], we cannot view the communication subsystem as a *single homogeneous module*. As a result, even though customized transport solutions have been developed for specific networks (e.g., mesh networks [24], vehicular networks[18], sensor networks [25]), making them work well on an end-to-end level is complex. The common practice is to use different transport regimes for wired and wireless networks, with a proxy in the middle (e.g., I-TCP [7], PEP [11]). However, in the current architecture these solutions break the end-to-end semantics, introduce problems like fate-sharing, and make hand-offs difficult.

2.2 Diverse network services

The first wave of Internet applications were host-to-host applications that only required "connectivity" as a service. Today, many applications benefit from richer network services, e.g., web caching or video transcoding [16, 27]. Network services are especially relevant to wireless and mobile users since many wireless devices have limited resources, making network-based services essential. Other reasons why network entities may need to be involved in communication include security and management [10].

Unfortunately, inserting services in an end-to-end path is hard in today's Internet for two reasons. First, the data plane of existing protocols (IP and TCP) works with packets and bytes streams, which is too fine a granularity for most in-network services. Second, and more importantly, TCP's end-to-end semantics are very rigid – they do not allow role of an intermediary or accommodate different delivery semantics. As a result, network services are typically deployed in a way which is hidden from the end-points (e.g., transparent web proxies) or as overlays. This is not ideal, since there can be poor interactions with other services, like firewalls, and sharing services between applications, users and providers is hard. Earlier proposals, like DOA [33] and NUTSS [21], deal with network level middleboxes, such as NATs and firewalls, while our focus is "flow middleboxes" [20] that provide higher level services, such as caching.

2.3 Relaxed End-Point Synchronization

Current Internet protocols assume that the communicating endpoints will be simultaneously connected to the network, and key mechanisms for reliability and congestion control implicitly rely on this [12]. However, this is increasingly no longer the case. While DTNs [19] have recently received a lot of attention, intermittent connectivity has been an important research topic in mobile computing for a long time [29]. Relaxing the endpoints synchronization requirements is however not only important for wireless devices as increased interest in energy efficiency means that wired

hosts may also be offline for longer periods of time, making relaxed synchronization an issue for the entire network infrastructure [5].

Today, applications that need to work in the presence of intermittent connectivity either provide this support over UDP, which means that they lose key TCP features such as reliability and congestion control (or need to re-implement them as part of the application), or use short-lived TCP sessions, which is inefficient. Clearly, the architecture must handle these disruptions as a routine matter by providing suitable mechanisms *within* the network, as well as appropriate transport semantics to the applications for dealing with various types of disconnections.

2.4 Topology Control

The network path chosen for data transport is traditionally chosen by the network layer, but researchers have identified many cases where it would be useful to apply higher level criteria, such as identifying disjoint paths (for availability) or considering service availability, during path selection [6]. While BGP does provide policy interfaces, they are limited and are typically not accessible to users.

Topology control is especially important for wireless devices as they often have a choice of access points they can associate with or have multiple interfaces that use different technologies [22, 32]. This leads to a rich set of connectivity options, and the choice will often be based on high level information such as service agreements, service availability, mobility history etc. Moreover, wireless optimizations such as soft hand-off and network coding, increasingly leverage the broadcast nature of wireless, breaking the "wireless link" abstraction, and requiring a more flexible approach to topology control.

2.5 System Concepts

The above requirements are very broad, but we can identify two underlying themes: (1) support for very diverse networks, nodes and services; and (2) richer interactions between end-points and the network both at the infrastructure (topology) and service level. This suggests two systems concepts that may be useful.

The first theme suggests that we should *raise the level at which interoperability is supported in the system*, since this may make it easier to "hide" diversity at the lower layers, e.g., allowing the deployment of custom network protocols. Interoperability affects not only the protocol function, but also addressing, unit of data transfer, etc. Note that raising the level of interoperability is likely to reduce (not increase - theme two) the degree of transparency between the network and end-points.

A second concept, *decoupling network regions with very different properties*, and as a result making them fairly "homogeneous", can help in a number of ways. First, it can simplify the deployment of end-to-end solutions over homogeneous networks. Second, the nodes that implement decoupling can be used to insert additional functions (e.g., network services, storage for caching or relaxed end-point synchronization). Finally, by making the points of decoupling visible to applications, it gives applications some control over topology and to implement specific semantics

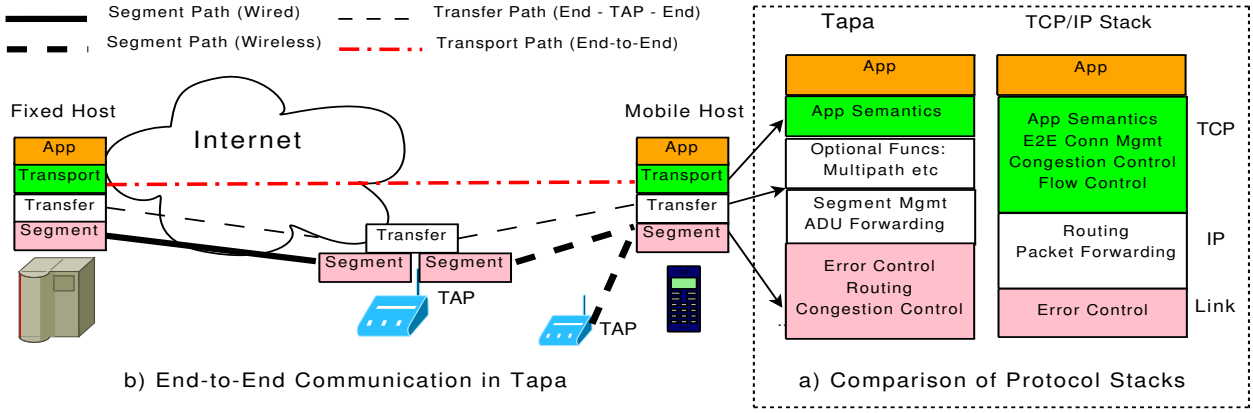


Figure 1: High Level Overview of Tapa

between the end-points and in-network services. Note that we can view the interoperability and decoupling concepts as affecting modularity in the "vertical" (across layers) and "horizontal" (across the network topology) dimensions of the system. In the next section, we build on these two concepts to introduce the Tapa architecture.

3 Tapa: Architecture

We first give a high level overview of Tapa, focusing on how we distribute functions between end-hosts and TAPs. We then elaborate on the key segment, transfer, and transport functions.

3.1 Overview

Figure 1(a) compares the Tapa protocol stack with the current TCP/IP protocol stack, showing how Tapa unbundles the transport layer of the current Internet. The role of the new layers in the Tapa architecture is better understood by considering an end-to-end communication involving a mobile client, one or more TAPs, and a fixed-host (shown in Figure 1(b)). In this case, an end-to-end path consists of two segments, wired and wireless, that are joined by a TAP. Segments can be very diverse, ranging from various wireless access networks (e.g., multi-hop mesh networks, blue-tooth, 802.11 etc), to wired segments (e.g., paths in the Internet or an enterprise network). Each segment is responsible for delivering data from one end of the segment to the other, employing congestion control, routing, and reliability mechanisms that are appropriate for the particular segment[1].

The segments in an end-to-end path are managed by the transfer layer, which spans the client, TAP, and the fixed-host. TAPs provide the glue required to combine multiple segments (e.g., buffer space) and also have sufficient storage that facilitate relaxed synchronization between end-points. TAPs also actively collaborate with the client, sharing information about the services they offer as well as their capabilities. An end-to-end path between a client and server can use multiple segments (paths), either simultaneously or over time, as suggested in Figure 1(b). This adds a topological

[1]Naturally some segments may not need all these mechanisms - e.g., a wireless USB or a point-to-point wired link

element to the transfer layer which provides many advantages to the client, such as support for multi-path and content discovery.

Similar to IP in today's Internet, the segment and transfer layers offer a best effort service to the transfer and transport layer, 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 services is the same as for network layer in today's Internet [28], i.e. full network-level reliability is expensive and not always needed. Therefore, Tapa's transport layer, which is implemented on top of the transfer layer, still provides reliability and other application semantics to the end-points, albeit over a very short path consisting of a small number of segments. For example, a reliable data streaming service would require end-to-end acknowledgements, data ordering, etc. The transport layer can also accommodate the insertion of services on the TAPs, as we elaborate in Section 3.4. Finally, both transfer and transport layers use ADUs [13], rather than byte streams or datagrams, offering a higher level of abstraction while enabling a variety of optimizations (Section 3.3).

3.2 Segment Layer

The segment service is responsible for transferring ADUs across a segment, e.g., client - TAP or TAP - server. Segments can choose the data granularity they use internally (e.g., frames, bytes, etc.) allowing them to optimize communication as appropriate. Segments are fixed and do not move, so they *can* use network-specific locators as addresses to communicate with the other segment point. For example, in the wired Internet IP addresses based on CIDR may provide the necessary scalability, but MAC addresses may be more appropriate in wireless networks.

Best Effort Data Delivery: Tapa's transfer service 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 network coding or opportunistic forwarding and block acknowledgements on a wireless segment [24, 9]. As Tapa targets diverse edge scenarios, the lifetime of a segment may be very short in some instances (e.g., mobility scenarios). Therefore, segments should limit the retransmission of ADUs. For example, the transfer layer should be able to *cancel* the transmission of requested ADUs or the segment retransmission attempts can be based on soft state that should be periodically refreshed by the other end of the segment.

Other functions: Segments should also address *congestion control* and *routing* on a per-segment basis. It is well known that congestion control mechanisms need to be tailored for different types of networks [7, 24]. 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.

Candidates: Segments can be implemented in diverse ways. Figure 2 shows two possible example scenarios. If we have a mesh access network then we can use a segment protocol like HOP [24], which provides its own congestion control and reliability mechanisms. In contrast, we expect more traditional segments (e.g., TCP/IP) on the wired side. Segments can also bypass IP or even traditional link layers, using options as diverse as blue-tooth and wireless USB. This is a direct benefit of raising the level of interoperability – 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

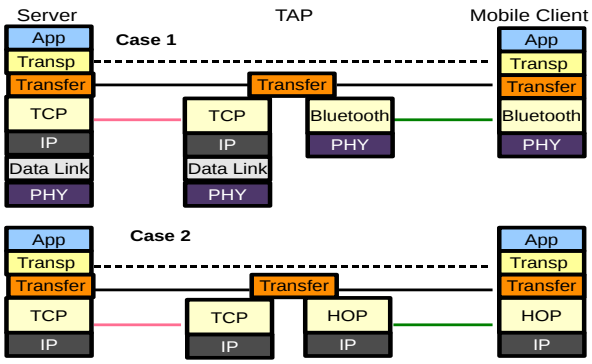


Figure 2: Some sample Tapa configuration cases. Segments can be very diverse and customized for each environment.

stacks for the segment technologies they use. This will lower the barrier for next-generation of embedded devices and sensors to communicate over the Internet.

3.3 Transfer Layer

The transfer layer of Tapa is the inter-networking layer of Tapa, managing end-to-end ADU transfers over multiple segments.

Core Functions: The transfer layer includes a set of *core* functions that are needed for all transfers and must be supported by both end-points and TAP(s) (Figure 1(a)). We broadly divide them into *segment management* (control plane) and *forwarding ADUs* between different segments (data plane). Segment management includes selecting TAPs and establishing segments to ensure an end-to-end path for ADU transfer.

The transfer layer also needs to do congestion control over the multi-segment path to ensure that TAP buffers do not overflow. This task is much simpler compared to congestion control over the entire end-to-end path (including the wired Internet and the access network) as a transfer-level path in Tapa typically only involves a limited number of segments. Note that congestion control in Tapa should be interpreted more broadly than in, e.g. TCP, since relaxed synchronization, segment hand-off, and multi-path operation can create discontinuities in the network path that can interact poorly with congestion control. Several congestion control solutions could be considered, including TCP-style end-to-end based on implicit feedback, various ECN-style solutions based on explicit feedback, and back pressure based on segment-per-segment flow control [24]. Since transfer-level congestion control operates at a higher level in the stack (e.g., on ADUs) and at the edge of the network, we decided to use the last option in the Tapa prototype since it fits well with the segment model and we do not face the extreme scalability requirements of IP in backbone, but other solutions should be evaluated as future research.

Optimizing Performance: Besides these core functions, there are a number of functions that can be viewed as performance optimizations. Examples include *multiplexing multiple segments*, recovery from *TAP failures*, *data recovery* after TAP failures or switching segments. These func-

tions may only be useful in certain scenarios, or their implementation may be scenario specific. We here focus on using Tapa to support wireless and mobile users. In that context, these functions often add a *spatial dimension* to the transfer layer to deal with the environment around the client, providing critical information that can help improve data transfers. For example, Tapa can support out-of-band discovery of nearby TAPs and their capabilities (e.g., services, cached content). Similarly, this spatial aspect can provide information on the failure of a TAP or the presence of an alternate TAP that can help optimize data transfers. Most of these optimization functions only need to be implemented at the network edge (client and TAP connected to the wireless network) while the server only needs to support the core functions.

For most of the above functions, it makes natural sense to have the client be in-charge of making data transfer decisions. This is important, especially in a mobile environment, for two reasons. First, some of the decisions may be driven by application requirements, which only the client can know. Second, the client is usually in the best position to know about the volatile wireless conditions and availability of opportunities for optimizations. Also, if we want to support mobility between different networks (that are part of separate domains), then clearly the client needs to be involved. However, there are situations where the TAP can play a more active role (e.g., prefetching data), so Tapa provides mechanisms that allow a client to delegate functions to a TAP through appropriate transport semantics.

Comparison with IP: The role of the transfer layer is somewhat similar to that of IP in today's Internet: delivering data over multi-hop paths. However, there are several differences in both the control and data plane. Internet routing needs to establish routes in large scale but fairly stable networks; in contrast, the Tapa transfer layer only needs to establish 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 network dynamics). IP packet forwarding is optimized for high throughput despite large forwarding tables; the main challenge in Tapa is dealing with vagaries of the edge as well as the opportunities that it offers. 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 higher level challenges (e.g. selection of segments based on content or service availability, mobility).

Data Units: The transfer layer uses ADUs [13], offering a middle ground between existing applications that generally use containers (e.g., file names) and TCP/IP, which uses datagrams/byte streams.

ADUs are a natural unit for exploiting multiple TAPs or multiple interfaces to download a file, or for inserting services such as caching in the end-to-end path. An important Tapa requirement is that ADUs must be identified using a label (e.g., content based hashing) that is used consistently at the segment, transfer and transport layers. This is important both for data recovery and for certain content-centric optimizations, e.g. DOT [31]. The use of ADUs changes the interface between the transport layer and applications, compared with the socket API. Applications should be able to specify the associated requirements with each ADU such as, importance of delivery, limit of the usefulness of the data, etc.

Identifiers: As the transfer layer offers an end-to-end connectivity service and may need to handle scenarios where segments change at the edge (e.g., mobility), it uses "identifiers" to

uniquely identify the endpoints. As segments may use locators, the transfer service must keep track of the mapping of identifiers to locators. This mapping is automatically maintained as segment use changes during a transfer. For example, when a client moves to a new TAP, it establishes a new segment and the client and server can update the mapping. For cases where some other node needs to start communication with a mobile node, we can also use the notion of a "home TAP", with whom the mobile client always maintains a segment to get incoming bootstrap messages. Once the session starts, the mapping can be updated based on the actual TAP the client wants to use.

3.4 Transport Layer

Tapa's transport functionality is very thin as it operates over a limited number of segments and most of the heavy lifting performed in traditional transport layer functions, is already implemented within a segment layer. As an example, error control is largely handled in the segment layer, and the role of the transport layer is to deal with lost ADUs, e.g. as a result of TAP failures or non-recovered ADUs during hand-off for mobile users. Similarly, the transport layer may need to reorder ADUs that were delivered out of order, e.g. as a result of the use of multiple segments. Important thing to note is that we can easily implement support for different semantics (e.g., partial reliability or out-of-order delivery). Basic functions like reliability, and ordering will typically involve only the end-points, but depending on the semantics it can also involve network services running on one or more TAPs.

A number of other functions, such as caching, data integrity, authentication, and confidentiality are today often implemented in the application. Since they affect the semantics of the connection, it is worth considering making them part of the transport function. A closely related issue is that connections increasingly involve "middleboxes" that affect end-to-end semantics [11, 16]. They are not part of the transport functions today, since protocols such as TCP cannot accommodate intermediaries. In Tapa, TAPs are visible to the end-points, opening the door for richer semantics that explicitly capture the role of services running on the TAP. We first discuss some examples of functions that may need help from network services.

Caching: Web and other types of caching within the network is already widely used and is typically supported at the application level (e.g., HTTP).

Data Integrity: Most applications rely on end-to-end integrity check (e.g., CRC, hashes, etc) for their correct operation, but some network services may change the bit streams in the middle of the end-to-end path (e.g., transcoding).

Authentication: Trust is becoming increasingly more important for users and applications. At times they may have more trust on an end-point (e.g., a bank) and in some cases they may trust an in-network element (e.g., a TAP owned by your enterprise) more than the end-point.

Confidentiality: Data confidentiality is important but is often at odds with the need for in-network elements to see the data contents [10]. The most practical approach is "controlled transparency", allowing explicit involvement of network elements with a flexible degree of control [15].

Many application require more than one of the above functions simultaneously, requiring participation of a third-party in some functions while other functions may just involve the end-points. As TCP does not allow such rich semantics, current applications implement this functionality at

the application level, using separate TCP connections with the third-party and between end-points. This adds unnecessary performance and management overhead for the applications. In contrast, Tapa can deal with this as part of transport, *using a single session involving the TAP and the end-points*. We use some example scenarios to illustrate this flexibility:

Application Independent Caching: Imagine a user who is interested in downloading a file or a video and the content is already available in the cache of TAP. In some cases the client may not trust the TAP (e.g., open wifi). So the client transport will request integrity check from the end-point while transport of the TAP can serve the data in an application independent way (similar to DOT [31]. *Transport Semantics:* End-to-end integrity, Caching at TAP.

Video Streaming: Imagine a client streaming a video on her resource constrained mobile phone, requiring a low resolution version of the video. The TAP provides the required video transcoding which means that data integrity checks can only be on a per-segment basis (Server - TAP, and TAP - Client). However, client may still require authorization from the server to get the video from the TAP. *Transport Semantics:* Per-Segment Data Integrity, End-to-End Authentication

Download from un-trusted Server: Imagine an enterprise client downloading a software from an untrusted web server. The client will only run the software if it is verified by the enterprise TAP, which employs latest virus and worm scanning techniques. In this case, there is no direct contact between the client and server transport (they only talk to the TAP). The transport of the TAP will first finish the whole download, perform the checks, and then send the data to the client which will make sure that the download is verified by the TAP (through authentication and integrity between client and TAP). *Transport Semantics:* Per-Segment Authentication and Data Integrity

Secure Chat Session: In this case transport of only the two end-points is involved with no transport layer involvement of the TAP. *Transport Semantics:* End-to-end data integrity, authentication, and confidentiality.

3.5 Tapa and the end-to-end arguments

The end-to-end (e2e) arguments have played a major role in the success of the Internet [28, 12]. Let us briefly comment on two important implications of the e2e arguments and how they apply to Tapa. A first implication is that the network should remain *simple* since a lot of functionality needs to be implemented on end-points anyway. Even though it appears to push more functionality into the network, Tapa is in fact consistent with this argument. For example, by using segments to decouple heterogeneous parts of a network, we can push functions to the segment end-points, if appropriate, while heterogeneity would make it hard to do this for the end-to-end path. As another example, the use of TAPs to support network services on TAPs does not push functionality into the network (i.e., communication subsystem), but rather a way of providing systematic support for an essential network requirement. The second important implication is *transparency* of the network – ”what goes in, comes out”. Tapa embraces the reality that the presence of various players within the network compromise transparency [10]. As advocated by Clark et al. [15] the best way of dealing with this is to provide some form of “controlled transparency“, which Tapa provides (e.g., control over level of confidentiality, etc) as part of the transport semantics.

Function	Description
Application API	
getADU(id(s), requirements, source-hint, source-app, callback)	Call used to pull an ADU
sendADU(id(s), data, requirements, destination, dest-app, callback)	Call used to send an ADU
registerApp(appID, callback)	App registration for receiving pushed ADUs
TAPInfo (id, gps-location, capabilities)	Call used to provide TAP information
Segment Layer API	
register(callback)	Registering to receive incoming Tapa msgs
sendMessage(id/locator, message, reqs, callback)	sending a message across the segment

Table 1: Tapa interfaces to the Application and Segment Layers. Interface between transfer and transport uses a subset of these APIs.

4 System Design

We have designed and implemented a prototype of Tapa with the goal of supporting a comprehensive evaluation of the flexibility of the architecture. As a result, our design focuses on implementing the key features of the architecture and how they interact with each other rather than optimizing individual modules. The major component of our prototype is a transfer service that is specifically designed for mobile and wireless users. It uses various existing options as segment protocols. In addition, we also implement a transport layer that provides support for various application semantics. Finally, a key part of the design is the interface for interacting with application and segment layers (Table 1).

We first describe the control and data planes of the transfer service and then describe the transport protocol.

4.1 Transfer Service - Data Plane

We describe key features of the data plane, focusing on transfers from the server to the client, but 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. 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.

The data plane on the client is responsible for handing the ADUs to the transport 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.

Naming of ADUs allows the transfer service to use ADUs as a unit of transfer. Applications can use any naming convention as long as it adheres to broad rules, e.g. maximum length, etc. For data ADUs for typical applications, our implementation uses content-based names corresponding to the hash of the data contents. This type of naming helps in exploiting content similarity within and across different applications, and can significantly improve caching performance [26, 16]. It also helps in content-centric networking where ADUs can be retrieved from many different sources, which is very useful in a mobile environment.

However, content-based naming may not be suitable for some application scenarios, e.g., dynamic content, control messages, etc. As a result, our implementation uses a random nonce for a request ADU and server reuses the same nonce for the response ADU. Similarly, applications can have custom rules for dynamically generated ADUs. For example, incrementing an ADU "counter" for each successive ADU that is generated and using the counter value to generate the ADU ids. These conventions allow the receiver to *anticipate* the id of the ADU, which helps in retrieving ADUs in scenarios involving mobility.

Transfer Modes: For typical data transfer requests, the client (or a TAP) can specify how the response should be sent by the server. There are two basic options: (i) immediately sending the contents of the ADU. This is useful for short messages or in scenarios where no ADU-based optimization is required (or possible). and (ii) sending the ADU Ids corresponding to the request and then making separate requests for these ADUs (similar to DOT [31]). This is useful for larger transfers or when ADU based optimization is possible. There is also a hybrid mode that combines the above two options.

Headers: The transfer service header consists of `(src-id, dst-id, TAP-id(s), app-id)` while the ADU header consists of: `(id, length, resp-type, TTL)` Most fields are quite standard and self explanatory (*resp-type* correspond to the different transfer modes that we discussed above).

4.2 Transfer Service - Control Plane

As discussed earlier, managing transfers is primarily the responsibility of the mobile client, although some decision making can be delegated to the TAP. At a high level, the control plane is responsible for collecting information about connectivity options in the wireless network(s), managing segments for use by the data plane, and recovering from errors.

4.2.1 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 better 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 in history). 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).

4.2.2 Managing Segments

The transfer control plane oversees all aspects of the end-to-end transfer of ADUs. 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. Moreover, the control plane also selects a suitable response mode for the ADU request which depends on the ADU requirements.

Authentication may be required to establish a segment. Once a TAP is discovered, the spatial 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 Tapa, 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 transfer 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.

It is possible during a transfer that the spatial plane detects another segment that can be used in parallel with the existing segment. Examples of such scenarios include residential wireless sce-

narios where the up-link bandwidth of APs can be aggregated or the simultaneous use of multiple interfaces. If such an opportunity is available, the control plane will establish another segment and will coordinate with the data plane on how to "stripe" the ADUs over the available segments [2]

4.2.3 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, so connectivity disruption is more likely. Second, the state on the TAP is lost. 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.

4.3 Transport

Tapa's end-to-end transport is thin and implements three types of functions: *reliability*, *ordering* and *delegation* oriented semantics. 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 transferred in the same way. In rare cases where the transfer service is unable to recover lost ADUs (e.g., ids of the missing ADUs are not known) then it is left to the end-to-end transport 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 transport may temporarily store the ADUs in the storage if there is significant reordering and it is short of buffers.

The transport also supports functions that can include the TAP in one way or the other, called the delegation oriented semantics (earlier discussed in Section 3.4). The end-point applications can specify caching, authentication, confidentiality, and data integrity requirements in the form of different modes to the transport layer which carries this information as part of the transport header. For example, they specify what type of data-integrity they want (e.g., SHA1 hash, check-sum, etc) and whether it should be provided by the end-point or TAP. Similarly, applications can specify whether an ADU can be cached or whether a cached ADU is acceptable as a response. As part

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

of authentication, the end-points or the TAP can also optionally add a certificate to the transport header to identify itself. These certificate can also allow for delegation e.g., server authorizing the TAP to speak on its behalf. Such certificates are useful for services like caching where the user may not trust the TAP. Our implementation currently does not provide full support for adding and verification of certificates.

4.4 Prototype Implementation

Our prototype is implemented in C++ and operates as a user level server with one binary that supports three different modes: client, TAP, and server. The common functionality shared by all three modes includes most of the data plane functionality. This includes support for caching ADUs, looking them up or transferring them to another node. The server includes functionality in the form of an application library for creation of ADUs (other nodes can also support this functionality), enabling legacy applications to create ADUs. We use `SHA1` of the ADU contents as its id. The unique functionality in the mobile-host includes a live discovery tool that allows us to do fine-grained TAP discovery. The live discovery tool sniffs packet in monitor mode using `libpcap`, and primarily looks for three things: presence of a new TAP, departure of a TAP, and change in signal strength of a node. For the departure of a TAP, we use keep alive messages while exponential weighted average of the signal strength values is used to smooth out fluctuations that may be present in the readings. Finally, we also developed a simple application, Tapa File Transfer Protocol(TFTP), that uses the Tapa prototype for transferring files.

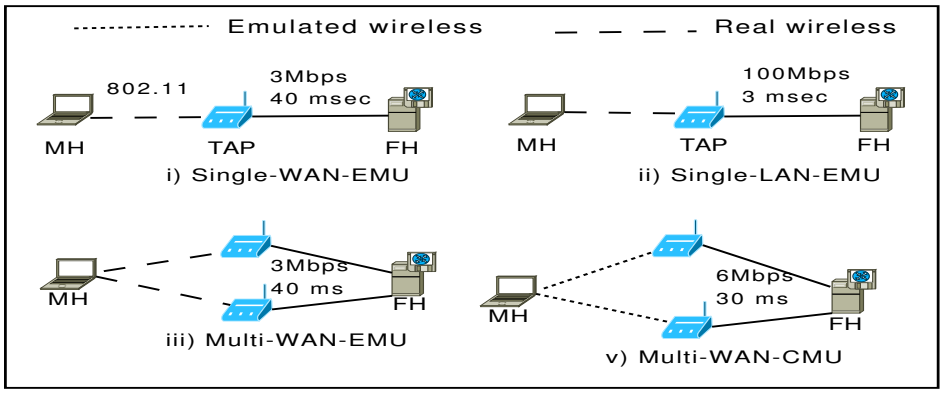


Figure 3: Various Topologies used in the Evaluation. Ones with Emu suffix correspond to Emulab testbed while those with CMU correspond to CMU wireless emulator. Wired links are emulated.

5 Evaluation

The goal of our evaluation is to show the flexibility of Tapa under diverse scenarios. Unfortunately, there is no single agreed upon metric that can quantify flexibility. Therefore, in our evaluation, we have used multiple ways to characterize flexibility of Tapa. Broadly, our evaluation can be divided

Type	Remarks
Segments	HOP, Bluetooth, TCP, Wireless USB(WIP)
Optimizations	Multiplexing Multiple Segments, Caching (On-Path and Off-Path)
Applications	New (TFTP[Sec 4.4]) and Legacy (Firefox)

Table 2: Diverse options supported by Tapa prototype.

into two parts. First, we show how Tapa can support diversity at multiple levels: applications, segment layers, and optimizations. We show flexibility by demonstrating functionality as well as quantifying 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 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 [8]. Unless otherwise noted, we conduct average of five runs (deviation is within 10% of the mean). We varied the ADU size from 64kB to 1MB.

5.1 Accommodating Diverse Options

Table 2 lists the various options supported by our prototype at different levels.

5.1.1 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 experience 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 [24]. 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

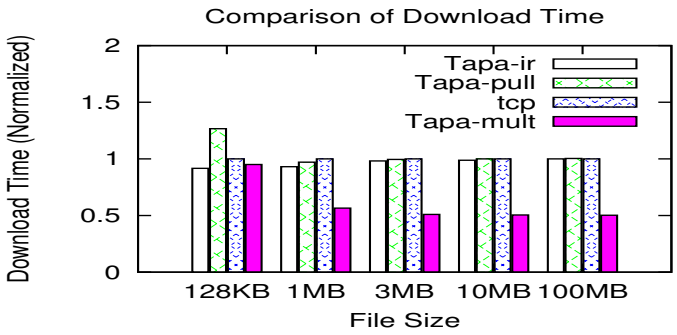


Figure 4: Aggregating uplink bandwidth of multiple TAPs.

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.

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.

Wireless USB: We are currently working on adding support for wireless usb as a Tapa segment layer. This is an extreme option as it is not even considered a networking medium in general. However, it is attractive considering the emerging home wireless scenario where many new devices are now equipped with usb support (e.g., video cameras, TVs etc.). Providing a segment layer would allow such home devices to be part of the Internet. However, compared to other segment layers, wireless usb is a more challenging case due to limited APIs and support documents.

Performance: Table 3 shows the performance of these different segment layers (with tcp on the wired side) in an end-to-end transfer. TCP and HOP used wifi on the emulab testbed while the bluetooth experiment was on the emulator. The results show expected performance under the given conditions.

	TCP	HOP	Bluetooth
Throughput	5.95 Mbps	6.4 Mbps	600 kbps

Table 3: Download of a 10MB file with different segment protocols. TCP is used on the wired segment.

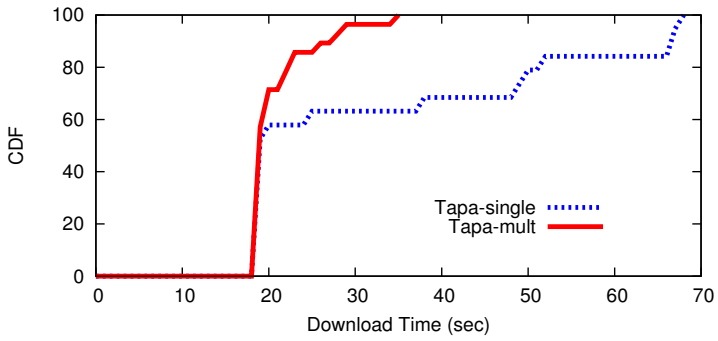


Figure 5: File Download Times in a Vehicular Communication Scenario. Use of multiple segments can mask disruptions that are common in such environments.

5.1.2 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 (listed in Table 2). 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 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. As a proof-of-concept, 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. 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 received; `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 4 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 [22] 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 [3]. We pick the two APs in the MS testbed with the best connectivity with the van and emulate their wireless channels us-

ing the emulator(`Multi-WAN-EMU` topology). The van downloads a 10MB file from a server located over the Internet and having a 60ms RTT with the APs. We compare the performance of Tapa with multiple segments (`Tapa-mult`) with the scenario where only a single segment is used (`Tapa-single`). We make 20 requests for each scenario and start times for the requests are randomly selected.

Figure 5 shows the cdf of the download times achieved in the two scenarios. Around 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.

5.1.3 Supporting Legacy Applications

It is important to consider the effort required in developing applications that can leverage Tapa. We focus on modifying *existing applications* – which were not developed with Tapa in mind – as they represent the hard case. The main challenge is to enable applications to work with ADUs rather than byte streams.

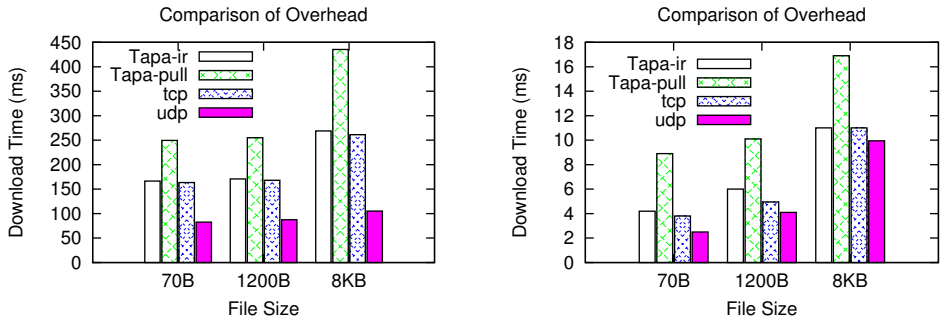
We modified the open source `Mozilla Firefox` browser to make use of Tapa API instead of sockets. Specifically, we created a Tapa stub that provided a socket-like interface to the applications and was used by the browser. This greatly simplified the browser modification process once the code was *separated* from sockets. Although there was a general separation of socket communication code and application logic, some of the browser optimizations violated this separation and made the modification process non-trivial. Unfortunately, this over reliance on sockets by *some applications* is an impediment to new network architectures like Tapa. However, the modification effort required is still manageable. Our experiences, as well as earlier similar efforts [31], suggest that typically it is in the order of *1-2 weeks* for reasonably sized applications. Overall, despite the huge code base of the browser the changes made to the browser code were small (*approximately 200 LOC*). Performance results of web-transfers were similar to the ones presented in Figure 4.

5.2 Overheads: Micro-benchmarks

Tapa offers several optimization 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.

5.2.1 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 segment. 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.



(a) WAN (b) LAN

Figure 6: 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.

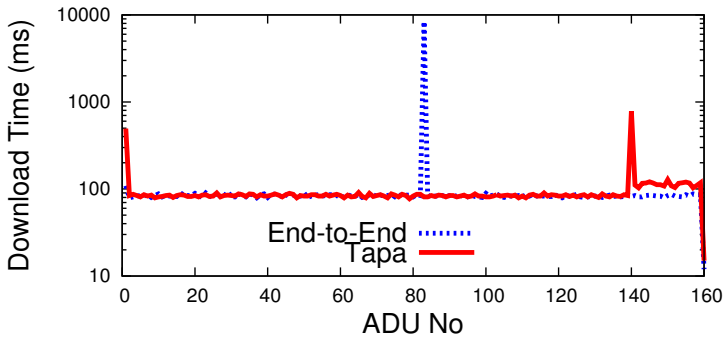


Figure 7: Case of TAP failure. Tapa recovers efficiently from TAP failures. Note log scale.

We compare the time required to complete a short request-response exchange in four different scenarios: `Tapa-ir`, `Tapa-pull`, `TCP` and `udp`. Figure 6(a) shows the results in a WAN setting with `80 ms` RTT. `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 6(b) shows results for a similar experiment on a LAN setting where the RTT is 2ms. We see that even in a LAN setting Tapa does not introduce any noticeable overhead and performs similar to the underlying segment protocols that it uses (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 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.

5.2.2 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 seconds and can start

serving the clients again. This pattern is repeated and clients continue to make requests at random times for a 10MB file. We compare end-to-end tcp 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 7 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 time 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.

6 Related Work

Tapa's design is inspired by a large body of work, including the design of the original Internet, wireless and mobility related proposals, overlay networks, and proposals that deal with visible middleboxes [12, 28, 7, 30, 17]. We give a brief overview of the most relevant proposals, focusing on the key differences with Tapa.

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 [7], PEP [11] 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 [22]) or multiple interfaces [32]. 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. As we show in this paper, the architectural support in Tapa simplifies the deployment of these optimizations, so they can be used by any application, transport or segment protocol.

Mobility and the Internet: Several architectural proposals address specific problems created by mobility [30, 19]. *i3* decouples the act of sending and receiving a *packet*, providing network level support for mobility [30]. 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. DTNs[19] provide a similar level of decoupling as Tapa, but their focus on arbitrary disruptions result in fundamental differences with Tapa: they do not support end-to-end semantics; employ push-based routing which results in poor performance under good connectivity scenarios and also makes it difficult to use the transfer layer optimizations supported by Tapa (multi-path etc) [24].

Overlays: Overlays have been used to support diverse functions and optimizations, such as path redundancy, throughput optimization, multi-path transfers, and content sharing [6]. 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.

Transfer Service: Tapa's transfer service leverages several concepts used in DOT [31], 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.

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 [21, 33]. 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[21] and DOA[33] 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 [20] 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.

7 Conclusions & Future Work

Tapa is an attempt to address the challenges of accommodating the "transformed edge". As we point out at different instances in the paper, there are many open issues that still need to be resolved. We expect that the architecture will evolve as we gain more experience with diverse applications and real users. For example, we hope to design richer interfaces at various levels that could better capture the flexibility offered by the architecture. We believe that Tapa has the right ingredients to facilitate such evolution. As we show in the paper, the separation of segment, transfer, and transport functions offer a lot of flexibility at different levels: use of customized solutions within a segment, various types of multi-path and content-centric optimizations at the transfer layer, and richer application semantics at the transport layer. Tapa is based on the same principles that have played a major role in the success of the Internet (e.g., end-to-end arguments) but also ensures that the realities of today, such as the role of players inside the network, are not ignored. We believe that this flexibility will act as a catalyst for innovation and will allow diverse applications, network services, and devices to be part of the Internet.

owards this end, we identify and provide architectural support for four key mechanisms that are critical for wireless and mobile devices. Our proposed architecture, Tapa, turns hidden wireless middleboxes into TAPs, and moves congestion and error control into a segment layer, where they can be handled in a network specific fashion. As a result, Tapa can support aggressive optimization of wireless resources, flexible hand offs, efficiently recover from disruptions in wireless connectivity, and recover from failed middleboxes. We implemented a prototype of Tapa and ran a wide variety of experiments on two testbeds. Our results show that Tapa can support diverse application requirements, accommodates various customized protocols on the wireless segment, and can effectively combine many optimizations that help mobile and wireless users.

References

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

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

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

[4] Aditya Akella, Glenn Judd, Srinivasan Seshan, and Peter Steenkiste. Self-management in chaotic wireless deployments. In *MobiCom '05*, pages 185–199, New York, NY, USA, 2005. ACM Press.

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

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

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

[8] 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.

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

[10] Marjory S. Blumenthal and David D. Clark. Rethinking the design of the internet: the end-to-end arguments vs. the brave new world. *ACM Trans. Internet Technol.*, 1(1):70–109, 2001.

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

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

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

[14] 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.

[15] 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.

[16] 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.

[17] Fahad R. Dogar and Peter Steenkiste. M2: using visible middleboxes to serve pro-active mobile-hosts. In *ACM SIGCOMM MobiArch '08*, pages 85–90, New York, NY, USA, 2008. ACM.

[18] 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.

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

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

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

[22] 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.

[23] 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.

[24] 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.

[25] Jeongyeup Paek and Ramesh Govindan. Rcrt: rate-controlled reliable transport for wireless sensor networks. In *SenSys '07*, pages 305–319, New York, NY, USA, 2007. ACM.

[26] 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.

[27] 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.

[28] J. H. Saltzer, D. P. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Trans. Comput. Syst.*, 1984.

[29] M. Satyanarayanan. Fundamental challenges in mobile computing. In *PODC '96*, pages 1–7, New York, NY, USA, 1996. ACM.

[30] Ion Stoica, Daniel Adkins, Shelley Zhuang, Scott Shenker, and Sonesh Surana. Internet indirection infrastructure. *SIGCOMM Comput. Commun. Rev.*, 32(4):73–86, 2002.

[31] Niraj Tolia, Michael Kaminsky, David G. Andersen, and Swapnil Patil. An architecture for internet data transfer. In *NSDI '06*.

[32] 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.

[33] Michael Walfish, Jeremy Stribling, Maxwell Krohn, Hari Balakrishnan, Robert Morris, and Scott Shenker. Middleboxes no longer considered harmful. *OSDI*, pages 215–230, 2004.